Szövegbányászat

Szövegbányászat

Tikk Domonkos (szerkesztő)
tikk@tmit.bme.hu

TYPOTEX Kiadó
Budapest, 2007

A könyv az Oktatási és Kulturális Minisztérium támogatásával, a Felsőoktatási Tankönyv- és Szakkönyv-támogatási Pályázat keretében jelent meg.

Farkas Richárd: 4.5–6. szakaszok
Kardkovács Zsolt Tivadar: 9. fejezet
Kovács László: 2.3.2.3 alpont, 8. és 10. fejezet (kivéve 10.1. szakaszt)
Répási Tibor: 8. fejezet
Szarvas György: 4.1–4. szakaszok
Szaszkó Sándor: 10.1. szakasz
Tikk Domonkos: 1., 2. (kivéve 2.3.2.3.), 3.2.2–3., 5–7. fejezetek
Vázsonyi Miklós: 3. fejezet (kivéve 3.2.2–3.)

ISBN 978 963 9664 45 6

Kedves Olvasó!

Önre gondoltunk, amikor a könyv előkészítésén munkálkodtunk. Kapcsolatunkat szorosabbra fűzhetjük, ha belép a Typoklubba, ahonnan értesülhet új kiadványainkról, akcióinkról, programjainkról, és amelyet a www.typotex.hu címen érhet el. Honlapunkon megtalálhatja az egyes könyvekhez tartozó hibajegyzéket is, mert sajnos hibák olykor előfordulnak.

Kiadja a Typotex Elektronikus Kiadó Kft., az 1795-ben alapított
Könyvkiadók és Könyvterjesztők Egyesületének tagja
http://www.typotex.hu
Felelős kiadó: Votisky Zsuzsa
Felelős szerkesző: Gerner József
A borítót tervezte: Tóth Norbert
Terjedelem: 20,56 (B/5) ív
Nyomtatta és kötötte: Séd Nyomda Kft., Szekszárd
Felelős vezető: Katona Szilvia

Tartalomjegyzék

Jelölésjegyzék

Az alábbi táblázat tartalmazza a könyvben használt fontosabb jelöléseket. Amennyiben ettől eltérünk, azt külön jelezzük.

$\mathbb{R}$	a valós számok teste
$\mathbb{N}$	természetes számok halmaza
$\mathbf{A} \in \mathbb{R}^{N \times M}$, a_{ij}	$N \times M$ méretű valós mátrix, ill. i-edik sorának j-edik eleme
$\mathbf{v} = \langle v_1, \ldots, v_n \rangle \in \mathbb{R}^n$	n elemű (valós) vektor
$\mathbf{v} \in \mathbb{R}^{1 \times n}$, $\mathbf{w} \in \mathbb{R}^{n \times 1}$	sorvektor, illetve oszlopvektor (ha hangsúlyozni akarjuk a vektor alakját)
$\langle \mathbf{u}, \mathbf{v} \rangle$	$\mathbf{u}$ és $\mathbf{v}$ vektorok skalárszorzata
$\lvert A \rvert$	A halmaz elemszáma
c; $c_j \in C$	kategória; a kategóriarendszer egy eleme
$\mathbf{c}$	a c kategória kategóriaprofilját megadó vektor
$C = \{c_1, \ldots, c_{\lvert C \rvert}\}$	kategóriák halmaza
cf_k	a t_k szó gyűjteménytámogatottsága
d; $d_i \in D$	dokumentum; a dokumentumgyűjtemény egy eleme
$\mathbf{d}$	a d dokumentum vektorreprezentációja
$D = \{d_1, \ldots, d_N\}$	dokumentumgyűjtemény (korpusz) és elemei
$d(\cdot, \cdot)$	távolságfüggvény
df_k	a t_k szó dokumentumgyakorisága a korpuszban
l_d, l_t	tanító-, ill. tesztdokumentumok átlagos vektormérete (ritka vektorként)
L_d, L_t	tanító-, ill. tesztdokumentumok átlagos hossza (szavak száma)
M, N	egyedi szavak, ill. dokumentumok száma
n_k	a korpusz t_k szót tartalmazó dokumentumainak száma
n_{ki}	a t_k szó előfordulásainak száma d_i dokumentumban
Neg_j	a c_j kategóriába nem tartozó tanítóadatok
Pos_j	a c_j kategóriába tartozó tanítóadatok
$s(\cdot, \cdot)$	hasonlóságfüggvény
$S_k(\mathbf{d}_i)$	$\mathbf{d}_i$-hez legközelebbi k szomszéd halmaza
t, t_k	szó (terminus); a vektortér k-adik dimenziójához

Fontosabb szakkifejezések rövidítésekkel magyarul és angolul

magyar	angol	rövidítés
adaptív szűrés	adaptive filtering	
alulról-felfelé	bottom-up	
alultövezési index	under-stemming index	UI
anaforafeloldás	anaphora resolution	AR
aratórobot	harvester	
átlagos kapcsolódás	group-average link	
balelemző	top-down parser	
csomópont	node	
dokumentumszűrés	text filtering	
dokumentumvezérelt osztályozás	document-pivoted categorization	DPC
döntési fa alapú osztályozó	decision tree classifier	DT-classifier
döntési szabály alapú osztályozó	decision rule classifier	DR-classifier
dzsókerkarakter	wildcard	
egycímkés osztályozás	single-label classification	
egyszerű kapcsolódás	single-link	
eltolás	bias	
erőforrás-leíró keretrendszer	resource description framework	RDF
feldolgozási folyam	stream	
feltételes valószínűségi mező	conditional random fields	CRF
felügyelet nélküli tanulás	unsupervised learning	
felügyelt tanulás	supervised learning	
felülről-lefelé	top-down	
fokozatos tanulás	incremental learning	
fontossági forrás	source of rank	
főkomponens-analízis	principal components analysis	PCA
frázissablon	phrasal template	
gyűjteménytámogatottság	collection frequency	CF
hibavezérelt tanulás	mistake driven learning	
hierarchikus (szöveg)osztályozás	hierarchical text categorization	HTC
információkinyerés	information extraction	IE
információnyereség	information gain	IG
információ-visszakeresés	information retrieval	IR
jellemzőkinyerés	term extraction	
jellemzőkiválasztás	term selection	
jobbelemző	bottom-up parser	
k-átlag	k-means	
kategóriavezérelt osztályozás	category-pivoted categorization	CPC
kereszthivatkozás	co-reference	
kereszthivatkozás-feloldás	co-reference resolution	

magyar	angol	rövidítés
keresztvalidáció, k-szoros	cross-validation, k-fold	
keret	frame	
kéretlen levelek szűrése	spam filtering	
kifejezéssablon	phrasal template	
kiterjesztett vagy bővített átmenet-háló	augmented transition network	ATN
kiválasztási elv	selection policy	
kötegelt tanulás	batch learning	
látens szemantikus indexelés	latent semantic indexing	LSI
legközelebbi szomszéd osztályozó (k-NN osztályozó)	nearest neighbor classifier	k-NN
lineáris legkisebb négyzetek mód-szere	linear least-squares fit	LLSF
lusta tanuló	lazy learner	
maximum entrópia Markov-modell	maximum entropy Markov modell	MEMM
meredekségi faktor	slope factor	
metszés (döntési fáé)	pruning	
minta alapú osztályozó	example-based classifier	
mintaillesztés	pattern matching	
névelem-felismerés	named entity recognition	NER
nyelő	rank sink	
nyelvközi információkinyerés	cross-language information extracti-on	CLIE
osztályozó bizottság	classifier committee, ensemble clas-sifier	
öregedési algoritmus	aging algorithm	
összegzéskészítő eljárás	text summarization method	
párhuzamos feldolgozási elv	parallelization policy	
pillanatkép	snapshot	
radiális bázisfüggvény	radial basis function	RBF
rangsoroló eljárás	ranking algorithm	
rejtett Markov-modell	hidden Markov model	HMM
relevancia-visszacsatolás	relevance feedback	
szekvencia alapú modell	structured prediction	SP
szinguláris értékfelbontás	singular value decomposition	SVD
szó-dokumentum mátrix	term-document matrix	TD matrix
szógyakoriság alapú súlyozás (TF-súlyozás)	term frequency	TF
szótövező	stemmer	
szózsákmodell	bag of words model	

magyar	angol	rövidítés
szövegosztályozás	text categorization/classification	TC
szupportvektor gép	support vektor machine	SVM
támogató osztályozás	categorization assistance	
tanítóhalmaz	training set	
teljes kapcsolódás	complete-link	
természetes nyelvek megértése	natural language understanding	NLU
természetes nyelvű adatbázis-interfész	natural language interfaces to databases	NLIDB
természetes nyelvű mélyhálókereső-interfész	natural language interface to deep web searcher	NLIDW
terminusfrekvencia és inverz dokumentumfrekvencia	term frequency & inverse document frequency	tf-idf
teszthalmaz	test set	
tisztaság	purity	
többcímkés osztályozás	multi-label classification	
többértelmű szavak egyértelműsítése	word sense disambiguation	
többségi döntés	majority voting	
többszintes osztályozás	multi-level classification	
válaszkereső rendszerek	question answering systems	QAS
udvariassági elv	politeness policy	
ugró pointer	skip pointer	
újralátogatási elv	re-visit policy	
újraparametrizálás	re-parametrization	
űrlap/nyomtatvány	form	

Egyéb alkalmazott angol rövidítések, és az esetlegesen kapcsolódó honlapcímek

rövidítés	jelentés	URL
ACE	Automatic Content Extraction	`www.itl.nist.gov/iad/894.01/tests/ace/`
ANSI	American National Standards Institute	`www.ansi.org`
CART	Classification and Regression Trees	`www.salfordsystems.com/cart.php`
CoNLL	Conference on Computational Natural Language Learning	`ifarm.nl/signll/conll/`
ETO	Egyetemes Tizedes Osztályozás	
HITEC	HIerarchical TExt Categorizer	`categorizer.tmit.bme.hu`
ID3	Interactive Dichotomizer 3	
IPC	Internatial Patent Classification (Nemzetközi Szabadalmi Osztályozás)	`www.wipo.int/classifications/ipc/en/`
ISO	International Organization for Standardization	`www.iso.org`
KWIC	Key Word in Context	
MUC	Message Understanding Conferences	`www.itl.nist.gov/iaui/894.02/related_projects/muc/`
OPAC	Open Public Access Catalog	
SMART	Salton's Magical Automatic Retriever of Text	
SQL	Structured Query Language	`www.ncb.ernet.in/education/modules/dbms/sql99index.html`
TREC	Text REtrieval Conference	`trec.nist.gov`
WIPO	World Intellectual Property Organization (Nemzetközi Szellemi Tulajdonok Szervezet)	`www.wipo.int`

Előszó

A szövegbányászat a számítástudomány szöveges elektronikus dokumentumok feldolgozásával és elemzésével foglalkozó szakterülete. Az internet korának egyik jelentős trendje az elektronikus adatok rohamosan növekvő mennyisége, melyek nagy része szöveges. Ez a jelenség a mindennapjainkban is jelentkezik az üzleti- és magánszféra, valamint a tudományos, gazdasági és mérnöki élet számos területén: az írásos kommunikáció, az adminisztráció, a dokumentálás folyamatainak jelentős részében elektronikus szövegeket gyártunk. A nagy mennyiségű szöveges adathalmazok hatékony kezelésében kínál segítséget a szövegbányászat. Módszereivel nemcsak az adatok közti eligazodás és keresés válik lehetővé, hanem támogatást is nyújt a dokumentumokban lévő rejtett összefüggések feltárására és kinyerésére.

Könyvünk az első olyan magyar nyelven megjelenő kötet, amely a szövegbányászat feladataira és módszerekre fókuszál. A szövegbányászat alkalmazásorientált szakterület, ezért fontosnak tartjuk, hogy az eljárások elméleti alapjainak széleskörű és alapos ismertetése mellett gyakorlati feladatok megoldásában is segítséget nyújtsunk az Olvasónak. Ez a törekvésünk megmutatkozik egyrészt abban, hogy az anyag tárgyalása során az algoritmusok gyakorlati megvalósításaival kapcsolatos tényezőknek külön figyelmet szentelünk, másrészt pedig hogy külön fejezetben tárgyaljuk néhány jelentősebb, szövegbányászati módszereket tartalmazó szoftvercsomag vonatkozó részét.

A könyvet egyaránt haszonnal forgathatják tehát a szövegbányászati megoldások bevezetését és alkalmazását tervező szakemberek, döntéshozók, informatikusok, valamint az informatikában jártas, a téma algoritmikus és elméleti alapjai iránt érdeklődő Olvasók is. A kötet tankönyvként és oktatási segédletként is szolgál. Anyaga részben a BME Villamosmérnöki és Informatikai Karán a könyv szerkesztője által tartott azonos című választható tárgy tematikájára és oktatási tapasztalataira, valamint a szerzők szövegbányászattal kapcsolatos kutatási és üzleti munkáira épül.

A kötet tartalma

A bevezető fejezet meghatározza a szövegbányászat feladatát, pozícionálja a szakterületet a kapcsolódó témakörökhöz képest, valamint bemutat néhány tipikus alkalmazási példát.

A 2. fejezet a szövegbányászatban alkalmazott alapvető előfeldolgozási módszereket tárgyalja. Megismertetjük az Olvasót a dokumentumok reprezentálására szolgáló numerikus modellekkel, amelyek közül részletesen foglalkozunk a vektortérmodellel. A dokumentumok vektorreprezentációinak létrehozásánál kitérünk a nyelvspecifikus feldolgozás kérdéseire (pl. szótövezés), külön pontban tárgyalva a magyar vonatkozású eredményeket és eszközöket. Jelentős terjedelemben mutatjuk be a vektortérmodell dimenziójának csökkentésére vonatkozó jellemzőkiválasztó és -kinyerő módszereket.

A 3. fejezetben röviden tárgyaljuk az információ-visszakeresésnek a szövegbányászattal szoros kapcsolatban lévő területeit, különös tekintettel az eredmények relevanciájának, ill. a rendszerek hatékonyságának mérésére. Szintén ez a rész foglalkozik a mintaillesztés alapvető technikáival.

A 4. fejezet elsőként néhány tipikus alkalmazási példán keresztül bemutatja az információkinyerés célját és jelentőségét, valamint összeveti tulajdonságait az információ-visszakeresésével. Ezután röviden elemezzük a legfontosabb részfeladatait: a névelem-felismerést, a kereszthivatkozások, szereplők és köztük lévő kapcsolatok azonosítását, illetve az eseménykeretek illesztését. A továbbiakban a szabály alapú és statisztikai megközelítések tulajdonságait, valamint a nyelvspecifikus problémákat vizsgáljuk. A fejezetet a névelem-felismerés, illetve azon belül a tulajdonnév-felismerés problematikájának tárgyalása zárja.

A tematikus osztályozás a dokumentumok rendszerezésének leggyakrabban alkalmazott módszere. Az 5. fejezet elsőként az osztályozási feladat különböző aleseteit veszi számba, majd néhány jellemző példán keresztül bemutatja az alkalmazási területek sokszínűségét. Ezután a felügyelt tanulási paradigma alapjait tárgyalja a fejezet, amit az osztályozó algoritmusok részletes ismertetése, majd elemzése követ. Külön szakaszban foglalkozunk a hierarchikus osztályozás kérdéseivel.

A dokumentumok tematikus rendszerezésének alternatívája a csoportosítás, ennek módszereit a 6. fejezet veszi górcső alá. A fejezet szerkezete hasonló az előzőhöz. Először a csoportosítási problémák és eljárások fajtáit, valamint az alkalmazási példákat tárgyaljuk, amit a felügyelet nélküli tanulási modell ismertetése követ. A particionáló és hierarchikus csoportosítási eljárásokat külön szakaszok-

ban tárgyaljuk, majd kitérünk a csoportok címkézésének kérdésére. Végül összehasonlító elemzés keretében vizsgáljuk az egyes módszerek hatékonyságát.

A 7. fejezet a dokumentumok tartalmi összegzésével, ezen belül főleg a kivonatolással — azaz a szöveg legrelevánsabb mondatainak meghatározásával — foglalkozik. Először megvizsgáljuk, hogy milyen jellemzők alapján tudjuk meghatározni a mondatnak a dokumentum tartalmára vonatkozó relevanciáját, majd néhány fontosabb módszert ismertetünk. A fejezetet a módszerek összehasonlítása zárja.

A 4–7. fejezetekben olyan módszereket ismertetünk, amelyek a szövegekben lévő nemtriviális vagy rejtett információk kinyerésére nyújtanak megoldásokat; ezeket a feladatokat tekintjük a szövegbányászat legalapvetőbb területeinek. A 8–9. fejezetek a dokumentumkeresés feladatával foglalkoznak, amely témakör szorosan kapcsolódik az információ-visszakeresés területéhez. Ennek ellenére úgy gondoltuk, hogy a szöveges dokumentumok kezelésének teljes körű tárgyalása mindenképpen megkívánja, hogy számottevő terjedelemben tárgyaljuk ezt a témát is.

A 8. fejezet az internetes keresőmotorokkal foglalkozik. A történeti áttekintés után a keresőmotorokkal szemben támasztott követelményeket mutatjuk be. Ezt követi a keresőmotorok felépítésének és a dokumentumok indexelését végző technikáknak az áttekintése. Külön fejezetben tárgyaljuk a piacvezető Google keresési technológiájának alapjait és a PageRank módszert, végül összevetjük a piacon található keresőmotorok hatékonyságát és funkcióit.

A 9. fejezet az információkeresésnek egy magasabb szintű módjával, a válaszkereső rendszerekkel foglalkozik. Előbb a természetes nyelvű adatbázis-interfészek megközelítését ismertetjük, majd pedig az internetes adatbázisok tartalmában, az ún. mélyhálóban való keresés problematikájával foglalkozunk.

A könyv zárófejezete néhány szövegbányászati szoftvercsomagot ismertet. Az első két szakaszban statisztikai és adatbányászati elemzőszoftverek szövegbányászati kiegészítéseit elemezzük: az SPSS Clementine szoftver Text Mining for Clementine modulját és a StatSoft Statistica Text Mining komponensét. A következő szakaszokban az adatbázis-kezelő szoftverek szövegbányász funkcióit tekintjük át. Nagyobb terjedelemben foglalkozunk az Oracle Text komponenssel és a MicroSoft SqlServer szövegkezelő moduljával, majd röviden ismertetjük a mySQL, a DB2 és a Sybase adatbázis-kezelők szöveges dokumentumok kezelésére vonatkozó támogatását. A kötetet gazdag irodalomjegyzék és részletes tárgymutató zárja.

Útmutató a könyv olvasásához

A kötet a szövegbányászat területének elméleti és gyakorlati oldalát egyaránt igyekszik bemutatni. Az elméleti részek tárgyalásánál feltételezzük, hogy az Olvasó legalább alapszintű ismeretekkel rendelkezik a lineáris algebra, a valószínűségszámítás, az adatbázis-kezelés, és a bonyolultság-, valamint az információelmélet területein.

A könyv felépítése lehetővé teszi, hogy bizonyos fejezetek önmagukban is érthetőek legyenek azok számára, akik csak néhány témakör iránt érdeklődnek, vagy már rendelkeznek előismeretekkel. Mindenképpen javasoljuk a 2. fejezet áttanulmányozását, hiszen az ebben tárgyalt részekre a későbbiekben gyakran támaszkodunk.[1] Szintén sokszor használjuk a 3.2.2. pontban tárgyalt mértéket. A többi fejezet egymástól függetlenül is érthető, ezekben hivatkozással jelezzük, ha más fejezetben tárgyalt ismeretekre építünk.

Mint az összes informatikai szakterületnek, a szövegbányászatnak is főleg angol nyelvű a szakirodalma. Könyvünkben ezért a fontosabb fogalmaknál az angol megfelelőt is megadjuk, hogy az Olvasót ezzel is segítsük a téma részletesebb tanulmányozásában. A kiemelt terminológiák magyar és angol megfelelői összegyűjtve is megtalálhatóak a jelölésjegyzékben a 10. oldalon. Bizonyos esetekben nem feltétlenül ragaszkodtunk a terminológia magyarításához, különösen ha a magyar kifejezés használata nem terjedt el, vagy nem egyértelmű.[2]

A különböző jellegű kifejezések kiemelését egymástól eltérő szedéssel jelöljük. *Kurzív* betűtípussal szedjük a fontosabb, tárgymutatóban is szereplő fogalmak előfordulásait, valamint olykor ezt használjuk nyomatékosításra is. ***Dőlt*** betűvel emeljük ki a példák szövegét, illetve a példákban használt szöveges konstansokat. Betűtalp nélküli (sanserif) betűvel szedjük a programkódrészleteket és utasításokat. KISKAPITÁLIS fonttal emeljük ki a kettőnél több karaktert tartalmazó nagybetűs rövidítéseket. Végül az internetes címeket `írógépes` betűtípussal jelöljük, ahol a `http` protokollt alapértelmezésnek tekintettük, és csak az ettől eltérőeket írtuk ki. A szintaktialag helytelen példaszövegeket *-gal jelöljük.

A könyv terjedelmi korlátai miatt számos érdekes és hasznos anyagrész, illetve példa kiszorult a nyomtatott anyagból. Úgy gondoltuk azonban, hogy a téma iránt érdeklődő Olvasók nagy része rendelkezik internet-hozzáféréssel, ezért a könyvhöz készítettünk egy webes mellékletet is, ahol az említett anyagrészeken kívül

[1] Ez alól talán csak a 2.3. kivétel, amelynek anyagára főleg az 5–6. fejezetekben építünk.

[2] Például *karakterfüzér* vagy *-lánc* helyett a *sztring* kifejezést használjuk, a *funkció-/töltelék-/tiltott szó* kifejezések helyett salamoni döntéssel a *stopszó*t alkalmazzuk.

még számos hasznos forrást és linket találhat az érdeklődő. Az alábbiakban ismertetjük a részleteket.

A könyv honlapjáról

A könyv honlapja a

`szovegbanyaszat.typotex.hu`

oldalon található. A honlap az alábbi — a könyvhöz szorosan kapcsolódó — menüpontokat tartalmazza:

- a könyvhöz kapcsolódó példák, anyagrészek és kiegészítések fejezetenként rendezve; a könyv nyomdába adásáig az alábbi anyagok készültek el, illetve vannak előkészületben:

  2. fejezet Mondatokra bontó algoritmus működése (Tikk Domonkos); Porter-, Paice–Husk- és Tordai-féle szótövező részletes leírása példákkal (Tikk Domonkos); MATLAB példa a PCA-algoritmusra (Kovács László)

  4. fejezet Rejtett Markov-modellek és a Viterbi-algoritmus; Maximum entrópia Markov-modell; Feltételes valószínűségi mezők (előkészületben, Farkas Richárd)

  5. fejezet Karakter *n*-gramm alapú nyelvfelismerés (Tikk Domonkos)

  5. fejezet EM-algoritmus részletes leírása (előkészületben, Tikk Domonkos)

  7. fejezet Esettanulmány: böngészés támogatása kivonatolással kézi számítógépeken (Tikk Domonkos)

  10. fejezet Statistica mintapélda dokumentumok osztályozására; Az Oracle Text által nyújtott további keresési lehetőségek és mintapélda; Három példa az SQLSERVER keresési lehetőségeinek illusztrálására (Kovács László)

  Egyéb Tipogenetika; Spektrális szövegbányászat (előkészületben; Vázsonyi Miklós)

  Az elkészült anyagokra a kötet megfelelő pontján utalunk.

- a könyv előszava és tartalomjegyzéke;
- a könyv internetes linkekkel ellátott irodalomjegyzéke, amelynek segítségével a könyvbeli hivatkozások publikusan hozzáférhető része közvetlenül elérhető;
- a könyvben hivatkozott programcsomagok, algoritmusok, dokumentumgyűjtemények, szabványok stb. linkgyűjteménye;
- rövid ismertető a szerzőkről;

- hibajegyzék;
- a könyvről megjelent kritikák, recenziók, visszajelzések.

A honlap céljának tekinti a szövegbányászat népszerűsítését, valamint hogy megjelenési és publikációs fórumot nyisson a szövegbányászat iránt érdeklődőknek, illetve a területen dolgozó hazai szakembereknek, kutatóknak.

A kötet szerzői

A könyv 1–2. (kivéve a 2.3.2.3. alpontot), 5–7. fejezeteit, valamint a 3.2.2–3. pontokat Tikk Domonkos (BME, Távközlési és Médiainformatikai Tanszék; TMIT) írta. A 3. fejezet fennmaradó része Vázsonyi Miklós (BME, Kognitív Tudományi Tanszék) munkája. A 4.1–4. szakaszokat Szarvas György (Szegedi Tudományegyetem, Informatikai Tancsékcsoport; SZTE IT), a 4.5–6. szakaszokat Farkas Richárd (SZTE IT) jegyzi. A 8. és a 10. fejezet (kivéve 10.1. szakaszt), valamint a 2.3.2.3. alpont szerzője Kovács László (Miskolci Egyetem, Általános Informatikai Tanszék; ME ÁIT), a 8. fejezet társszerzője Répási Tibor (ME ÁIT). A 9. fejezet Kardkovács Zsolt Tivadar (BME TMIT) munkája, a 10.1. szakaszt pedig Szaszkó Sándor (BME TMIT) írta.

Köszönetnyilvánítás

A szerzők szeretnének köszönetet mondani mindazoknak, akik segítették a könyv létrejöttét. Külön köszönet jár azoknak, akik részt vettek a könyv kéziratának javításában, és értékes megjegyzéseikkel segítették munkánkat: Bodon Ferenc, Gál Viktor, Halácsy Péter, Körmendy György, Lopata Antal, Pilászy István, Szidarovszky Ferenc P., Takács Gábor. Szintén köszönjük Kiss Ferenc, Pléh Csaba és Infopark Alapítvány szakmai támogatását.

A Clementine és a Text Mining for Clementine adat- és szövegbányászati programcsomagokat az SPSS Hungary bocsátotta rendelkezésünkre, a Statistica szoftvert és Text Mining kiegészítését a StatSoft Hungary Kft-től kaptuk.

Köszönettel tartozunk az *Oktatási és Kulturális Minisztériumnak* a *Felsőoktatási Tankönyv- és Szakkönyvtámogatási Pályázat* keretében nyújtott segítségéért, valamint a TypoTEX Kiadó minden érintett munkatársának a könyv megjelenésében való segítségéért.

Minden igyekezetünk ellenére maradhattak hibák a könyvben. Kérjük, hogy amennyiben hibára bukkan, tájékoztasson bennünket a

`szovegbanyaszat@typotex.hu`

e-mail címen.

1. fejezet

Bevezetés

1.1. A szövegbányászat feladata

Az írástudó emberi civilizációk kialakulása óta a tudást szöveges dokumentumok formájában tárolják. Az ősi egyiptomiak is szöveges dokumentumokat hagytak az utókorra, azonban hieroglifikus írásuk megfejtése korántsem bizonyult könnyű feladatnak. A szöveg megértését végül az segítette elő, hogy a feliratok több nyelven szerepeltek ugyanazon a kövön, amelyek közül az egyik a görög volt, a másik kettő pedig egyiptomi. A görög nyelv kulcsként szolgált a hieroglifák megfejtéséhez. Az ősi egyiptomi hieroglifák megfejtéséből két tanulságot vonhatunk le: (1) a szöveges dokumentumok az emberiség egyik ősi emlékezeti mechanizmusa, ezért fontos biztonságosan és ugyanakkor visszanyerhető módon tárolni az adatokat; (2) a dokumentumokhoz való hozzáférés a tudás feltárásához nem elegendő, ez speciális gyakorlatot és erőforrást igényel.

Napjainkban, amikor a dokumentálási és adminisztrációs folyamatok túlnyomó része elektronikusan valósul meg — és ezáltal rendkívül nagy mennyiségű elektronikus dokumentum keletkezik — megfigyelhető az a trend, hogy az adminisztratív munkát végzők munkaidejük egyre növekvő hányadát fordítják (elektronikus) dokumentumok kezelésére. Míg ez 1997-ben csupán 20%-ot tett ki, addigra 2003-ra már a 30–40%-ot is elérte a Gartner Group becslése szerint. A Meryll Lynch elemzése szerint az üzleti információk 85%-a *strukturálatlan*, illetve *gyengén strukturált adat*, pl. e-mailek, emlékeztetők, üzleti és kutatási beszámolók, prezentációk, hírek, reklámanyagok, weboldalak, ügyfélszolgálati tevékenység jegyzetei stb. formájában áll rendelkezésre [26].

Adatbányászati módszerekkel *strukturált*, gyakran adatbázisokban tárolt adatokból nyerhetők ki összefüggések. Ezek a módszerek többnyire kihasználják, hogy a tárolásra szolgáló adatstruktúra információt ad az adat szemantikájára vonatkozóan is. Például egy személyek adatait tároló adattáblában a *születési év* mezőben szereplő évszámból sokkal könnyebben kinyerhető az életkor, mint amikor ugyanez az évszám a megfelelő kontextusban egy önéletrajzban, szabad szöveg-

ben fordul elő.[1] Ez utóbbi adattípust *strukturálatlan*nak nevezzük. Ezen azt értjük, hogy az adat szemantikájára nem utal a tároló adatstruktúra. *Gyengén strukturált adat*nak tekinthető pl. az XML, ahol bizonyos szemantikus vagy szerkezeti információk rendelkezésre állhatnak. Az adatbányászati módszerek közvetlenül nem alkalmazhatóak a jellemzően strukturálatlan, általános típusú, szöveges adatokra, amelyek tehát más megoldásokat tesznek szükségessé. Az ezzel foglalkozó szakterületet szövegbányászatnak nevezzük. Az 1.1. táblázatban összehasonlítjuk a szöveg- és adatbányászat alapvető ismérveit.

1.1. táblázat. Az adat- és szövegbányászat összehasonlítása ([100] alapján)

	Adatbányászat	Szövegbányászat
az elemzés tárgya	numerikus és kategorikus	szabad formátumú szöveges dokumentum
az adatok jellege	strukturált	strukturálatlan, gyengén strukturált
az adatok tárolási helye	(relációs) adatbázis	tetszőleges dokumentumgyűjtemény
feladat	összefüggések feltárása, jövőbeni szituációk előrejelzése	szövegelemzés, információkinyerés, osztályozás, csoportosítás, összegzéskészítés, vizualizálás, kereséstámogatás stb.
módszerek	neurális hálózatok, döntési fák, statisztikai modellek, klaszteranalízis, idősorok elemzése stb.	dokumentumindexelés, felügyelt és felügyelet nélküli gépi tanulók, számítógépes nyelvészeti eszközök, ontológiák
a világpiac jelenlegi mérete	100 000 elemző közepes és nagyvállalatoknál	100 000 000 vállalati munkatárs és egyéni felhasználó
széleskörű piaci megjelenés	1994-től	2000-től

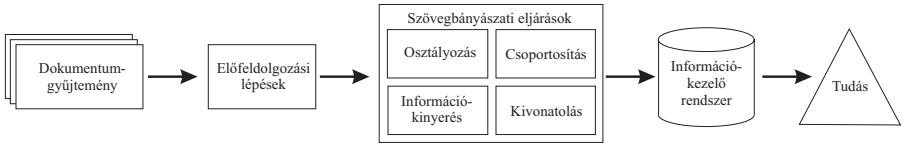
A *szövegbányászat*ot szöveges adatokon végzett feldolgozási és elemzési tevékenységként definiáljuk, melynek célja a dokumentumokban rejtetten meglévő

[1] Ez persze nem meglepő, hiszen az adat tárolási formája a megcélzott felhasználástól függ. A szöveges önéletrajz olvasásra, az adatbázis gépi feldolgozásra készül. A két tárolási mód közötti átjárás jelentős transzformációs költséggel jár.

új információk feltárása, azonosítása és elemzése. Ez a meghatározás analóg az adatbányászat definíciójával.

A szövegbányászat alapvető problémája nyilvánvaló: a természetes nyelvek emberek közötti — elsősorban szóbeli, majd később írásbeli — kommunikáció céljára alakultak ki és fejlődtek, nem a számítógépes feldolgozás szempontjai szerint. Az emberek könnyedén felismerik és alkalmazzák a nyelvi mintákat, és általában nem okoznak gondot nekik olyan, a számítógépek számára nehezen megoldható feladatok, mint pl. a különböző helyesírási variációk kezelése, a kontextus felismerése vagy a stilisztikai jelleg azonosítása. Nyelvi tudásunk lehetővé teszi a strukturálatlan szövegek megértését, ugyanakkor nincs meg bennünk a számítógépeknek az a képessége, hogy a szöveget nagy mennyiségben, vagy nagy sebességgel dolgozzuk fel. A szövegbányászat általános célja tehát az emberi nyelvi tudás ötvözése a számítógép nagy feldolgozási kapacitásával [65].

A szövegbányászat interdiszciplináris alkalmazásorientált szakterület. A szövegbányászati feladatok megoldása során egyaránt szükség van a matematikai, az informatikai, azon belül főként a gépi tanulással kapcsolatos eszköztárak alkalmazására, valamint emellett a természetes nyelvek feldolgozásával foglalkozó területek, a *számítógépes nyelvészet*, a *nyelvtechnológia* eredményeire. Fontos látni, hogy a szövegbányászat és az utóbbi szakterületek céljai különbözőek: míg a nyelvtechnológia alapvetően a nyelvészeti feladatok — pl. morfológiai, szintaktikai, ill. szemantikai elemzés — automatizálását tekinti feladatának, addig a szövegbányászat szövegekkel kapcsolatos informatikai problémák algoritmikus megoldásait keresi, amihez gyakran felhasznál nyelvtechnológiai eszközöket is.



1.1. ábra. A szövegbányászat általános modellje

A szövegbányászat általános modellje az 1.1. ábrán látható. A dokumentumokon először előfeldolgozási lépéseket hajtunk végre, ennek eredményeként előáll a dokumentumhalmaz adott feladatnak megfelelő reprezentációja. A reprezentáció legtöbbször numerikus, esetleg strukturált (pl. XML) szöveges formátumú. Az előfeldolgozás során gyakran alkalmazunk nyelvtechnológiai eszközöket is. Ezután hajtjuk végre a szövegbányászati eljárásokat. Az eredményeket célszerű a hatékony hozzáférés érdekében információkezelő rendszerben tárolni.

1.2. A szövegbányászat alkalmazási területei

Az üzleti élet szereplői és az átlagos felhasználók egyaránt gyakran találkoznak olyan problémákkal, amelyekre a szövegbányászat nyújthat megoldást. Az alábbiakban ízelítőt nyújtunk azon területekből, amelyek az utóbbi években a legjellemzőbb alkalmazói voltak a szövegbányászati megoldásoknak, illetve ahol a közeljövőben várható a szövegbányászat eszköztárának elterjedése. Ezen kívül a könyv több fejezetében ismertetünk az adott problémakörhöz kötődő alkalmazási példákat és lehetőségeket, pl. az információkinyerésnél (82. oldal), a szövegosztályozásnál (107. oldal), a szövegek csoportosításánál (147. oldal), a kivonatolásnál (166. oldal), ill. a szövegbányászati szoftverek ismertetésénél (250. oldal).

Ügyfélszolgálati tevékenység A nagy forgalmat lebonyolító ügyfélszolgálatoknál hatalmas mennyiségű ügyféllel történő beszélgetés zajlik naponta. Ezek jellemző tartalma, fontosabb témái, az ügyfélkör igényeinek változása a szolgáltatónak fontos információt jelent, amellyel hatékonyan reagálhat a piac változásának kihívására.

Biztonság, bűnüldözés A terrorveszély elhárítása érdekében szövegbányászati módszereket is bevetnek a biztonsági szervek. A lefoglalt ill. megszerzett szöveges digitális adatok nagy mennyisége miatt információkinyerő és szövegelemző eljárásokat alkalmaznak az adatok átvizsgálásánál, amelynek segítségével hatékonyan tudnak neveket, helyszíneket, kapcsolatokat, egyéb összefüggéseket azonosítani. A technológiát a bűnüldözés egyéb területein — pl. gazdasági csalások — is hatékonyan alkalmazzák.

Üzleti intelligencia és információszerzés Az üzleti életben rendkívül fontos az információhoz jutás sebessége: a releváns adatoknak minél gyorsabban kell a megfelelő formában a döntéshozók, ill. az elemzők rendelkezésére állnia. Jelentős piaci előnyre tehet szert az, aki idejében tud reagálni egy gazdasági információra vagy kiszivárogtatott hírre — ugyanez fennáll a tőzsdei ügyleteknél is. Mivel az információtermelés sebessége gazdasági és üzleti dokumentumok esetén is folyamatosan növekszik, ezért ezt emberi kapacitással nem, vagy csak igen költségigényesen lehet követni. Helyette automatikus szövegfeldolgozást is tartalmazó hírfigyelő és -elemző rendszereket alkalmaznak. A technológia ugyancsak felhasználható az olyan üzleti hírekről való automatikus értesítésre, amelyek konkurens cégekről, ill. termékekről tartalmaznak információt.

Gyógyszerkutatás A farmakológia és a kapcsolódó orvosi tudományok az egyik tipikus alkalmazási területe a szövegbányászatnak, mivel itt rendkívül nagy

mennyiségű szöveges dokumentum keletkezik a különböző publikációkban, beszámolókban, feljegyzésekben, jelentésekben stb. A szakterület dokumentumainak egy része nagy méretű adatbázisokban található (pl. Medline). Az ezekben való hatékony keresésre szövegbányászati eszközöket alkalmaznak, amelyek képesek pl. bizonyos betegségfajták, tünetek, gyógymódok stb. együttes vagy kombinált előfordulását is kinyerni az adatokból.

Államigazgatás, e-kormányzat Ezen a területen szintén nagy mennyiségű dokumentumot kell hatékonyan kezelni. A szövegbányászati technológiát alkalmazni lehet pl. az írásbeli beadványok megfelelő ügyintézőhöz való irányítására, az egyszerűbb kérdések automatikus megválaszolására. Ennek segítségével a fenntartási költség és az ügyfelek várakozási ideje egyaránt csökkenthető.

Internetes keresés A kulcsszó alapú keresés korlátaival a legtöbb felhasználó szembesült már. Ha többértelmű keresőkifejezést használunk — a tipikus példák: *jaguár* (állat, autómárka), *saturn* (bolygó, elektronikai cég, autótípus), *tus* (zuhany, írószer, vívás, zene)[2] —, akkor a kívánt információ megszerzéséhez a keresés finomítására van szükség. Ennek kiküszöbölésére egyes keresőszolgáltatások lehetővé teszik a keresés kontextusának megadását, amelyet szövegbányászati eljárásokkal valósítanak meg. Alternatív megoldást jelent a keresés megkönnyítésére a tartalmak tematizált tárolása, itt a keresők szövegosztályozási módszereket használnak a téma szerinti besorolásra.

A kereséstámogatás másik tipikus példáját az motiválja, hogy a találatok gyakran nagyméretű, akár több száz oldalas dokumentumok, amelyek több témát is tárgyalnak, és nem feltétlenül relevánsak a kereső számára. Ahhoz, hogy a felhasználó megtalálja a neki fontos információt, el kell mélyednie a szövegben, ami rendkívül időigényes. Erre a problémára a szövegbányászat az összegzéskészítő módszereket kínálja megoldásként, amelyek automatikusan összefoglalják a dokumentum tartalmát, így segítve a gyorsabb keresést, böngészést.

[2] Érdekes, hogy a nemzetközi keresők erre a keresőszóra a nyomtatóval kapcsolatos cikkeket is találnak a *tűs* szó ékezetnélküli reprezentációja miatt. Ez a példa is jól mutatja, hogy a hatékony szövegbányászati alkalmazások — bizonyos fokig — nyelvfüggők.

2. fejezet

Előfeldolgozás, modellalkotás, reprezentáció

A szövegbányászat számos feladata visszavezethető már ismert és megoldott — statisztikai, gépi tanulási, ill. adatbányászati — problémára. Ahhoz azonban, hogy ezeket a megközelítéseket alkalmazni tudjuk, a strukturálatlan szöveges dokumentumokat olyan, jellemzően numerikus objektumokká kell átalakítanunk, amelyekre egyrészt már könnyen adaptálhatóak az ismert algoritmusok, másrészt pedig a kiinduló szöveges dokumentumot az adott problémának megfelelően reprezentálják. Ezért a szövegbányászati feladatmegoldás első és egyben az egyik legfontosabb lépése az *előfeldolgozás*, melynek során a dokumentumokat az adott feladatnak megfelelő, és a szövegek sajátosságainak tárolására alkalmas *modell* szerinti alakra hozzuk, azaz elkészítjük a dokumentumok *reprezentáció*ját a modellben. A reprezentáció segítségével a szöveges dokumentumokat matematikai eszközökkel hatékonyan kezelhetjük a modellben. A dokumentumkezelés konkrét lehetőségei a modelltől, és a megvalósítására szolgáló adatábrázolási technikáktól függenek.

Gyakorlatilag minden szövegbányászati elemzést előfeldolgozás előz meg. Az előfeldolgozás legfőbb feladata a további elemzés *hatékonyság*ának megteremtése. Ennek érdekében *egységesítési*, *formalizálási*, *normalizációs* lépéseket tartalmazhat. Mivel a szövegbányászati feladatokat gyakran rendkívül nagy méretű adathalmazon kell végrehajtani — nem ritka a több GB, sőt TB-os adatmennyiség sem (ld. a 10.3.1. pontot) — ezért a szövegek hatékony reprezentációjának megoldása elsődleges szempont. Adattárolási szempontból is célszerű a dokumentumok karakteres alakja helyett numerikus reprezentációt alkalmazni. Ez könnyen belátható, ha azt tekintjük, hogy egy szó szöveges tárolásához előfordulásonként és karakterenként legalább 1 bájtra[1] van szükség, míg numerikus tárolásnál elegendő előfordulásonként egy legfeljebb 4 bájtos szám, valamint egy különálló indexállomány, amelyben az adott szót és a számot egymáshoz rendeljük. Az adatábrázolás

[1] Egy karakter tárolásának tárigénye a karakterkódolástól függ. Például az UTF-8 kódolás változó számú (maximum 6) bájtot használ.

sajátossága miatt a karakterláncokra vonatkozó műveletek (keresés, összehasonlítás stb.) is sokkal időigényesebbek, mint a számokra vonatkozóak. Következésképpen a numerikus reprezentációval gyakran nagyságrendekkel csökkenthető az adatok tárigénye és az algoritmusok futási ideje is. Az előfeldolgozást a gyakorlatban a rendelkezésre álló informatikai eszközök (tárolási kapacitás, memória, műveleti sebesség) is befolyásolják; ezekkel kalkulálni kell a rendszer tervezése során.

2.1. Az előfeldolgozásnál vizsgált dokumentumjellemzők

A szövegbányászati rendszerek forrásadatai a szöveges dokumentumok, amelyek a tartalmazott szövegen kívül számos más jellemzővel is rendelkeznek. Ahhoz, hogy a dokumentumok egységes kezelhetőségét biztosítsuk, fel kell térképezni a *dokumentumgyűjtemény*nek, másnéven a *korpusz*nak,[2] és elemeinek jellemzőit, és fel kell készülnünk arra, hogy szükség esetén átalakításokat végezzünk el rajtuk. Az alábbiakban sorba vesszük a dokumentumok fontosabb jellemzőit, majd kiemelten foglalkozunk azokkal, amelyek leginkább befolyásolhatják egy szövegbányászati rendszer hatékony működését, tehát kezelésére mindenképpen figyelmet kell fordítani.

2.1.1. Alapvető jellemzők

- *A dokumentumot hordozó médium*
  Az írásosság kialakulásának kezdetétől az ember számos fizikai médiumot használt szövegek tárolására. Az ókorban agyag- és kőtábla, bőrtekercs, pergamen, ill. papírusz volt a jellemző adathordozó, majd a kései középkortól kezdve elterjedt a papír. Mára az információ digitalizált, elektronikus formában való tárolása hódított teret. Az adattárolás fejlődése során fontos törekvés volt, hogy fizikailag minél kisebb helyen lehessen minél nagyobb mennyiségű adatot eltárolni, s a tár könnyen hordozható, sokszorosítható legyen. Ennek következményeként a ma jellemző fizikai adathordozón az adat már ember számára közvetlenül nem olvasható formában van jelen, csak számítógép segítségével válik értelmezhetővé.[3] Könyvünkben csak digitális adat-

[2] A terminológiát a szövegbányászat a számítógépes nyelvészetből vette át, ahol általában szakértők által grammatikai jellemzőkkel annotált, jellemzően nagy méretű szöveggyűjteményt értenek alatta. Könyvünkben a dokumentumgyűjtemény szinonimájaként használjuk.

[3] Éppen ezért nem várható az, hogy a szövegek papíron való tárolása hosszú távon teljesen háttérbe szoruljon. Ráadásul a legidőtállóbb tárolási forma még mindig a papír, gondoljunk csak az

hordozón tárolt szöveges dokumentumokkal foglalkozunk. Szövegbányászati feldolgozáshoz az egyéb formában lévő adatokat digitalizálni kell, illetve a digitális képi formátumú adatokat szöveggé kell konvertálni. Ezt manuálisan (begépelés), vagy jobbára automatizált módon (ha szükséges szkenneléssel, majd optikai karakterfelismeréssel — ld. webes melléklet) lehet végrehajtani.

- *A dokumentum fizikai elérési helye*
  Minden dokumentum rendelkezik fizikai elérési hellyel, ami nem digitális dokumentumok esetén lehet pl. egy könyvtár valamelyik polca, elektronikus dokumentum esetén pedig az adathordozónak egy szegmense. A szövegbányászati feldolgozás végrehajtásához a rendszernek tudnia kell, hogy az adott elektronikus dokumentum hol érhető el (URL, fájlnév), és legalább olvasási jogosultsággal kell rendelkeznie.

- *A dokumentum mérete*
  Egy szöveges dokumentum méretét többféleképpen definiálhatjuk, például a dokumentumban szereplő karakterek, szavak vagy oldalak számával, illetve a dokumentum által az adathordozón elfoglalt hely bájtban kifejezett mennyiségével. Nyers, azaz még feldolgozatlan dokumentumok esetén többnyire a fájlméretet használjuk, pl. a szöveges korpuszokat a dokumentumaik összesített méretével szokták jellemezni. Az előfeldolgozási fázis után a dokumentumok méretét karaktereik vagy szavaik számával, egy modell szerinti reprezentációjuk méretét pedig a felírásukra használt egységek számával (pl. tokenek, ill. egyedi szavak száma, indextömb mérete) lehet megadni.

- *A dokumentum statisztikai jellemzői*
  A méreten kívül egyéb statisztikai mutatók is érdekesek lehetnek a dokumentumok feldolgozása során. A legfontosabb jellemző a szavak eloszlása — ezzel részletesen foglalkozunk jelen fejezetben. Ezen kívül vizsgálható még például a karakterek eloszlása, a szóhosszak eloszlása és átlaga, magánhangzók és mássalhangzók száma — ezekből többek közt a dokumentum nyelvére vonatkozóan lehet következtetéseket levonni. Szükség esetén különböző szempontok szerint részletesen megvizsgálhatjuk a szöveg betűit (magas és mély magánhangzók eloszlása, zöngés és zöngétlen mássalhangzók eloszlása), de ezeknek a vizsgálatoknak többsége már részletes nyelvi elemzést is kíván.

elektronikus adathordozók (CD, DVD, floppy, DAT szalag, merevlemez) várható élettartamára és sérülékenységére. 10–20 évnél régebbi adathordozó esetén már az olvasóberendezés biztosítása és működtetése is komoly gondot jelenthet.

- *A dokumentum metaadatai*
  A dokumentumhoz eredendően tartoznak bizonyos fizikai metaadatok, amelyeket az adathordozón való tároláskor kap. Ilyen például a dokumentum keletkezési, ill. utolsó módosítási ideje, az azonosítására szolgáló fájlnév, a hozzáférési attribútumok stb. Fontos jellemzője a dokumentumnak, hogy milyen eszközzel lett létrehozva, ez határozza meg a formátumát. Ennek ismerete nélkül a dokumentum helyes feldolgozásának kimenetele igen kétséges — a formátummal kiemelten foglalkozunk a 2.1.2. pontban. Szövegbányászati szempontból ennél fontosabbak a dokumentum tartalmára és keletkezési körülményeire vonatkozó metaadatok, mint pl. a nyelv, a szerző vagy szerkesztő neve, a közzétevő neve, a tényleges keletkezési idő és hely, a téma vagy a tetszőleges besorolási rend szerinti jelzet. Az itt felsorolt adatok természetesen nem minden dokumentum esetén állnak rendelkezésre. Meghatározásuk, ha lehetséges egyáltalán, gyakran igen költséges. Néhány jellemző esetén — pl. nyelv, téma — ezt a feladatot szövegbányászati algoritmusok segítségével automatikusan is elvégezhetjük, de az ilyen algoritmusok hatékonysága igen eltérő lehet. A tematikus besorolást az 5. fejezet, a nyelvmeghatározást pedig ugyanazon fejezet online melléklete tárgyalja részletesen.
  A felsorolt jellemzők mellett a szövegnek vannak egyéb jellemzői is — pl. stílus, zsáner, nyelvezet stb. Ezek meghatározása nem tartozik a tipikus szövegbányászati alkalmazások közé, de ennek inkább csak célszerűségi okai vannak. Informatikai szempontból ilyen jellemzők alapján is elvégezhetjük a dokumentumok osztályozását vagy csoportosítását, ha a feladat végrehajtásához szükséges tanítókörnyezet, illetve elemző háttér rendelkezésünkre áll.

A dokumentumjellemzők tárgyalásának végére hagytuk azt a kettőt, amely az előfeldolgozás szempontjából a legfontosabb: a formátumot és a karakterkódolást.

2.1.2. A dokumentum formátuma és karakterkódolása

Elektronikus szöveges dokumentumot nagyon sokféle alkalmazással lehet létrehozni, és a tárolási formátum alapvetően a szoftver lehetőségeitől és beállításaitól függ. A leggyakoribb szöveges formátumok: txt, doc, pdf, html, xml, rtf, ps, tex, qxd, pub stb. A formátumok a szövegen kívül számos egyéb információt is tartalmazhatnak. Ezek lehetnek formázási vagy a szerkesztésre vonatkozó (meta)adatok, a dokumentum korábbi verziói stb. A szövegbányászati feldolgozás esetén az adott célfeladattól, illetve a tervezett megvalósítástól függ, hogy ezek közül az információk közül melyek hasznosak. Ezek hatékony kezeléséről a szö-

vegbányászati rendszer tervezőjének kell gondoskodnia. Például keresőrendszer készítésénél, ahol a dokumentumokban szereplő kiemelő formázási utasításokkal (szedés, kapitalizáció, eltérő fontméret vagy -típus, -szín) megjelölt szövegek relevánsabbak lehetnek az adott dokumentum jellemzésére, ezért érdemes a feldolgozásnál ezt az információt a reprezentációba átvinni, és a keresési algoritmusban is figyelembe venni (ld. a 204. oldalt).

A szövegbányászati rendszereknél általános esetben azonban a legfontosabb feladat a különböző formátumú dokumentumok egységes formára hozása, amelynek során a formázási, strukturális és szerkesztési stb. adatokat általában elhagyjuk, és kizárólag a szöveget, valamint a feladat szempontjából fontos metaadatot őrzünk meg. Egységesítéskor a legfontosabb a különböző konverziós modulok elkészítése. A konverzió célja tehát egy egységes, és a továbbiakban egyszerűen kezelhető szöveges formátum elérése, amelynek kimenete leginkább ASCII szöveg vagy XML, azaz a konvertálás megelőzi a modell szerinti reprezentációs alakra hozás műveletét.

A konvertálásnál megkülönböztetett figyelmet kell fordítani a szövegek karakterkódolásának helyes kezelésére. A karakterkódolásra vonatkozó információt esetenként tartalmazza a dokumentum maga, ismerete nélkül a feldolgozás helyessége nem garantálható. A dokumentum leírására használt kódkészlet a dokumentum nyelvével vagy nyelveivel függ össze. A hangjelölő írásmódot használó nyelvek kódolására — melyek ábécéje csak néhány tucat karaktert tartalmaz (az összes európai nyelv ilyen, pl. az angol, a magyar, a görög, a orosz, de számos ázsiai nyelv is ebbe a csoportba tartozik pl., az indonéz, a thai, a mongol [44]) — elegendő 1 bájt, azaz $2^8 = 256$ különböző karakterkód. A szótag-, vagy szójelölő írásmódot alkalmazó nyelvek (kínai, japán, koreai) teljes kódkészletét csak 2 bájton lehet tárolni. Erre fejlesztették ki az unicode kódtáblát, amely a nagy ábécéjű nyelvek összes karakterét képes kódolni. Az unicode-ra épül az UTF-8 kódolás, mely a karakterek unicode szabványon alapuló kódolásakor változó hosszúságú bináris egységeket használ. Ez azt jelenti, hogy csak szükség esetén használ két bájtot egy karakter kódolására, például a latin ábécé karaktereit egy bájton tárolja.

Még egynyelvű dokumentumgyűjtemény esetén is gyakran előfordul, hogy az egyes dokumentumok karakterkódolása eltérő. Magyar nyelvű szövegek leginkább közép-európai (ISO-8859-2-es) vagy UTF-8-as kódolást használnak, de a gyakorlatban nem ritkán akadnak olyan dokumentumok is, amelyek nyugat-európai kódolásúak, és ezért az *ő* és *ű* betűk helytelenül jelennek meg. Az előfeldolgozás feladatai közé tartozik a karakterkódolások egységesítése, illetve amennyiben lehetséges, a kódolási hibák javítása. Könnyen látható, hogy az egész előfeldol-

gozás hibás eredményt adhat, ha téves karakterkódolással olvasunk be egy dokumentumot,[4] különösen ha az eltérő bájtszámokra nem vagyunk tekintettel.

Vannak bizonyos formátumok, ilyen pl. a pdf, amelyekből komoly kihívás jelent a megfelelő kódolású teljes szöveg kinyerése. Mivel számos alkalmazás képes pdf formátumú dokumentum generálására, ezek sokfélesége már eleve bonyolítja a feldolgozást, sőt a pdf lehetőséget kínál a szövegnek képként való tárolására is. Ha nagyon heterogén a dokumentumok származása, akkor többnyire manuális ellenőrzési lépés beiktatására is szükség van az előfeldolgozás során [183]. További buktatókat rejt az, hogy a karakterkódolások egységesítését milyen eszközzel végezzük. Ügyelni kell arra, hogy ellenőrizzük a kiválasztott programnyelv alapértelmezett és opcionális kódkészletét, hiszen előfordul, hogy bizonyos kódolásokat egyáltalán nem támogat a szoftver.[5] Nem feledve, hogy a karakterkódolások egységesítése korántsem triviális feladat, a továbbiakban úgy tekintjük, hogy a dokumentumok már egységes kódolással állnak rendelkezésre.

2.2. Dokumentum reprezentálása vektortérmodellben

2.2.1. Dokumentumreprezentációs modellek

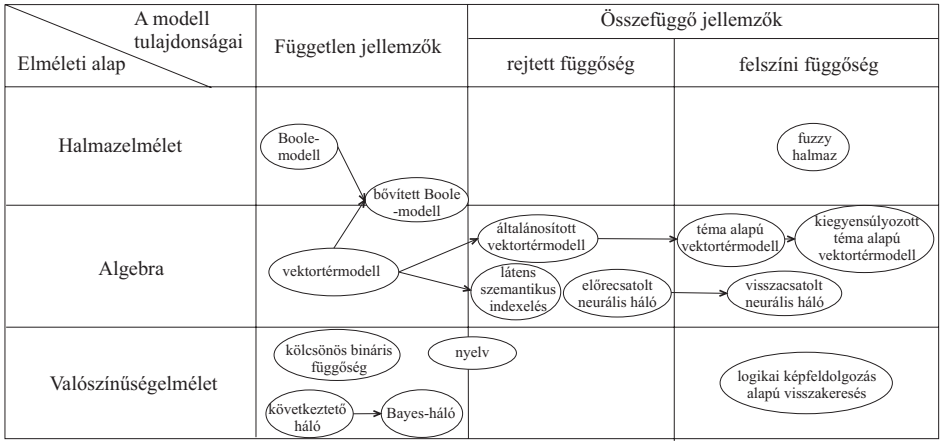
Egy probléma megoldásánál a modell kiválasztását a feladat jellege határozza meg. A szövegbányászati feladatok két alaptípusa a keresés és a rendszerezés. Az első esetben olyan dokumentumokat keresünk, amelyekben egy adott keresőkifejezés előfordul, utóbbi esetben pedig dokumentumokat hasonlítunk össze, és ennek alapján soroljuk őket egy adott kategóriarendszer elemeihez, vagy előre nem rögzített csoportokba. Rögtön látszik, hogy a két feladat eltérő adatábrázolást tesz szükségessé. Ugyanis az egyszerű keresést egy megfelelően előállított szóelőfordulási táblázat alapján elvégezhetjük, és minden dokumentumra fennáll, hogy vagy része az eredményhalmaznak, vagy nem. Ugyanakkor a rendszerezés esetén dokumentumokat kell összehasonlítani egymással, ami más megközelítést igényel, és az összehasonlítás eredménye is más skálán mozog, diszkrét logikai értékek helyett folytonos értékeket vehet fel.[6]

[4] Érdemes kísérletezni, hogy például MS-WORD-ben hogyan változik az egyszerű szövegként való beolvasásnál a megjelenített szöveg, ha helytelen kódolást választunk, pl. magyar szöveget unicode-ként értelmezünk.

[5] Egyes Java fejlesztőkörnyezetek XML-elemzői az ISO-8859-2-es kódolást nem kezelik.

[6] Amint azt látni fogjuk, a keresésnél is szükség van összehasonlításra — a válaszok rangsorolásánál.

A dokumentumok reprezentálására három megközelítés terjedt el: a halmazelmélet alapú, az algebrai és a valószínűségi modell. Ezek rendszerezését ismerteti a 2.1. ábra [107].



2.1. ábra. A dokumentumreprezentációs modellek rendszerezése a matematikai megközelítés és a reprezentáció elemeinek kapcsolata szempontjából [107]

A halmazelméleti modellek a dokumentumok hasonlóságát halmazműveletek segítségével határozzák meg. Ezt a megközelítés elsősorban a keresőrendszerek alkalmazzák. Ezzel a témával könyvünk a 8. fejezete foglalkozik, amelyben a keresőkben alkalmazott adatszerkezeteket, valamint a technológiai hátteret is ismertetjük.

Az algebrai modellek a dokumentumokat algebrai objektumokként, vektor vagy mátrix alakban ábrázolják, és algebrai műveletekkel hasonlítják őket össze. A fejezet további részében a leggyakrabban használt *vektortérmodell*t és kiterjesztett változatait tárgyaljuk részletesen. Az algebrai alapú modellek már alkalmasak dokumentumok rendszerezésére is.

A valószínűségi modellek a dokumentumokat valószínűségi eseményként kezelik, a hasonlóságot pedig feltételes valószínűségi becslésként határozzák meg. A modellek részletes ismertetése túlmutat könyvünk keretein, az érdeklődő Olvasónak a [71, 178, 198] munkákat ajánljuk.

A fenti felosztáson kívüli modellek is léteznek, pl. komplex függvénytani eszközöket alkalmaz a spektrális szövegbányászati modell.

2.2.2. A vektortérmodell

Legyen adva egy dokumentumgyűjtemény, amelynek elemein valamilyen rendszerezési műveletet kívánunk végrehajtani. Ehhez olyan modellt kell felépítenünk, amiben a dokumentumok távolságát vagy hasonlóságát egyszerűen meg tudjuk határozni. Intuitív módon nyilván azok a dokumentumok hasonlítanak egymásra, amelyeknek a szókészlete átfedi egymást, és a hasonlóság mértéke az átfedéssel arányos. Ezt a megfigyelést használja fel az információ-visszakeresésben (information retrieval, IR) széles körben használt *vektortérmodell* [162].

A vektortérmodell egy sokdimenziós vektortérben ábrázolja a dokumentumokat. A dokumentumokat vektorokkal reprezentáljuk, a vektortér dimenziói pedig a dokumentumgyűjteményben előforduló egyedi szavak.[7] Másképp megfogalmazva: a dokumentumgyűjtemény egyedi szavai által kifeszített vektortérben, ahol minden tengely egy szót reprezentál, a dokumentumokat a szavaikból álló *vektor*ként ábrázoljuk.

A vektortérmodellben a $D = \{d_1, \ldots, d_N\}$ dokumentumgyűjteményt a *szó-dokumentum mátrix*szal (term-document matrix) reprezentáljuk ($\mathbf{D} \in \mathbb{R}^{M \times N}$), ahol a mátrix d_{ki} eleme a k-adik szó (t_k) relevanciáját reprezentálja az i-edik dokumentumban, d_i-ben. A d_i dokumentumot reprezentáló dokumentumvektort $\mathbf{d}_i = \langle d_{1i}, \ldots, d_{Mi} \rangle$-vel jelöljük. A $\mathbf{D}$ mátrixban a sorok száma, M megegyezik az egyedi szavak számával, másképpen a vektortér *dimenziójá*val, N pedig a dokumentumok száma:

$$\mathbf{D} = \begin{matrix} \mathbf{d}_1 \\ \downarrow \end{matrix} \begin{matrix} \mathbf{d}_2 \\ \downarrow \end{matrix} \quad \begin{matrix} \\ \ldots \end{matrix} \quad \begin{matrix} \mathbf{d}_N \\ \downarrow \end{matrix} \\ \begin{pmatrix} d_{11} & d_{12} & \ldots & d_{1N} \\ a_{21} & d_{22} & \ldots & d_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ d_{M1} & d_{M2} & \ldots & d_{MN} \end{pmatrix} \tag{2.1}$$

A dokumentumvektorok tehát a mátrix oszlopai lesznek. A mátrix egy sora azon pozíciókban tartalmaz nullától különböző értéket, amelyekhez tartozó dokumentumokban a szó nem nulla relevanciájú.[8] Az egyedi szavak összességét *szótár*nak vagy *lexikon*nak nevezzük, jelölése T. A szótár mérete tehát $|T| = M$. Mivel általában egy dokumentumban a szótár szavainak csak egy kis töredéke fordul elő, ezért a $\mathbf{D}$ mátrix ritka. Ugyanakkor az egyedi szavak száma rendkívül nagy lehet,

[7] Ezt majd később pontosítani fogjuk.

[8] Ez általában ekvivalens azzal, hogy a szó nem szerepel a dokumentumban, de ettől eltérő esetekben is lehet a relevancia nulla.

akár a milliós nagyságrendet is elérheti. Ezért a szó-dokumentum mátrix hatékony tárolása és kezelése fontos feladat. A konkrét szövegbányászati feladat jellegétől függően a mátrix mérete nyelvtechnológiai és matematikai eljárások segítségével jelentősen csökkenthető. Ezeket a nyelvtechnológiai megoldásokat a 2.2.4.3. alpontban, míg a matematikai technikákat a 2.3. szakaszban ismertetjük.

Vegyük észre, hogy a vektortérmodell szerinti reprezentációban a szavak szövegen belüli pozíciójára és sorrendjére vonatkozó információ elvész. Ezt a megközelítést *szózsákmodell*nek (bag of words) is nevezi a szakirodalom. Zsáknak hívjuk azt a halmazfogalom általánosításaként kapott matematikai objektumot, amelyben az elemek többszörösen is szerepelhetnek.[9] Ebben a megközelítésben a *Nietzsche mondta: Isten halott* és az *Isten mondta: Nietzsche halott* dokumentumok reprezentációi megegyeznek.

A vektortérmodellt leggyakrabban a szavak sorrendjének figyelmen kívül hagyása miatt éri kritika. Ez a jellemző tulajdonság a modell szemléletéből — a hasonló szavakat tartalmazó dokumentumok hasonló tartalmúak — intuitíve következik. A modell által alkalmazott egyszerűsítés többnyire nem okoz problémát, hiszen a gyakorlati feladatok jó részében a szavak sorrendje nem számít, és a fenti példában mutatott egyezés elhanyagolható jelentőségű. A vektortérmodell alkalmazása melletti legfőbb érv az, hogy a dokumentumok kezelésének hatékony megvalósítását támogatja. Mindemellett persze lehetőség van a vektortérmodell olyan módosítására, illetve olyan más modellekkel való együttes alkalmazására, amelyekkel ez a probléma — a hatékonyság rovására — részben vagy teljesen kiküszöbölhető (pl. sztring- és n-gramm-kernelek).

2.2.3. Súlyozási sémák

A d_{ki} érték megválasztására több lehetőség van. A legegyszerűbb a *bináris reprezentáció*:

$$d_{ki}^{(1)} = \begin{cases} 1, & \text{ha } n_{ki} > 0 \\ 0, & \text{ha } n_{ki} = 0 \end{cases}, \quad \textit{bináris súlyozás} \tag{2.2}$$

ahol n_{ki} a t_k szó *előfordulás*ainak száma (másképpen *támogatottság*a) a d_i dokumentumban.[10] A (2.2) súlyozás figyelmen kívül hagyja az előfordulások számát,

[9] A szózsák elnevezés arra a képzeletbeli műveletre utal, amikor a dokumentum szavait beszórjuk egy zsákba, ahol tetszőlegesen keveredhetnek, és ezek után az eredeti sorrend már nem rekonstruálható.

[10] Mind a magyar, mind az angol szakirodalomban gyakran keveredik az *előfordulás* és a *gyakoriság* fogalma — gyakran a gyakoriságot (frequency) használják mindkét mennyiség megnevezésére.

ugyanakkora súlya van a szónak ha csak egyszer, vagy ha 10-szer szerepel. Természetesen az előfordulások számát felhasználhatjuk az adott szó fontosságának reprezentálására:

$$d_{ki}^{(2)} = n_{ki} \quad \textit{előfordulás alapú súlyozás}. \tag{2.3}$$

A (2.3) súlyozással kapcsolatban azonban jogos kérdésként merül fel, hogy ténylegesen lineáris függvénye-e a relevancia a szó támogatottságának. Ha egy szó 10-szer fordul elő egy dokumentumban, az vajon ténylegesen 10-szer fontosabb is? Érezhetően sokkal nagyobb különbség az, hogy egy dokumentum egy szót egyáltalán nem vagy csak egyszer, mint az, hogy egyszer vagy kétszer, illetve 9-szer vagy 10-szer tartalmaz. Számos olyan alternatív súlyfüggvény van, amely ezt a szempontot is figyelembe veszi, ilyen pl. a gyakran alkalmazott *logaritmikus súlyfüggvény*:

$$w_{ki} := d_{ki}^{(3)} = \begin{cases} 1+\log n_{ki} & \text{ha } n_{ki} > 0 \\ 0 & \text{egyébként} \end{cases} \quad \textit{logaritmikus súlyozás}. \tag{2.4}$$

Az eddig ismertetett súlyozási sémák figyelmen kívül hagyták a dokumentumok hosszát, noha egy 100 szavas dokumentumban egy szó tízszeri előfordulása nyilván sokkal jelentősebb, mint egy 10 000 szavasban. Ezt a szempontot a dokumentumok hossz szerinti normálásával vehetjük számításba. Normálásra a

$$\|\mathbf{d}_i\|_1 = |\mathbf{d}_i| = \sum_{k=1}^{M} n_{ki}, \quad \|\mathbf{d}_i\|_2 = \sqrt{\sum_{k=1}^{M} n_{ki}^2}, \quad \|\mathbf{d}_i\|_\infty = \max_{k=1}^{M} n_{ki} \tag{2.5}$$

függvényeket szokták használni. Ha ezek közül $\|\cdot\|_1$-t választjuk — amely az i-edik dokumentum szavainak számát adja meg —, akkor az előfordulás alapú súlyozásból az alábbi kifejezést kapjuk:

$$d_{ki}^{(4)} = n_{ki}/|\mathbf{d}_i| =: f_{ki} \quad \textit{gyakoriság alapú súlyozás}. \tag{2.6}$$

Az így definiált f_{ki} a t_k szó d_i dokumentumbeli *gyakoriság*a vagy *frekvenciá*ja, amit az angol elnevezés rövidítése alapján *TF-súlyozás*nak (term frequency) is

Könyvünkben előfordulás alatt mindig egész számot értünk, míg gyakoriság alatt az előfordulások — pl. dokumentumhossz szerint — $[0,1]$ intervallumba normált valós értékét. Ez utóbbit szokás *relatív gyakoriság*nak is nevezni.

hívnak. Hasonlóképpen elvégezhetjük a normálást a logaritmikus súlyfüggvényből (2.4) kiindulva, ekkor a

$$d_{ki}^{(5)} = w_{ki} / \sum_{k=1}^{M} w_{ki} \quad \textit{normalizált logaritmikus súlyozás} \tag{2.7}$$

sémát kapjuk.

Mindeddig a dokumentumokban (szótárban) előforduló összes szót egyenrangúnak tekintettük, holott a dokumentumok tartalmi jellemzését illetően a szavak jelentősége eltérő. A témaspecifikus szavak, mint pl. *szövegbányászat* sokkal jobban jellemeznek egy dokumentumot az általánosabb kontextusban használt szavaknál, mint pl. *biztosítás*, *elmélet*, és vannak olyan szavak is, amelyek a témától függetlenül bármely dokumentumban előfordulhatnak (névelők, határozószók, névutók stb. — pl. *az*, *hogy*, *alatt*, *is*). Az ilyen szavakat az angol terminológiát átvéve *stopszavak*nak nevezzük (stop words)[11], kezelésükkel kapcsolatban lásd még a 2.2.4.4. alpontot.

A t_k szó fontosságának megállapítására elsőre kézenfekvő mérőszámnak tűnik a szó előfordulásainak száma a teljes gyűjteményben, a *gyűjteménytámogatottság* (collection frequency, cf): $cf_k = \sum_{i=1}^{N} n_{ki}$. Nem mindegy azonban, hogy milyen az előfordulások eloszlása a korpuszon, azaz a t_k szó hány dokumentumban szerepel, ugyanis két azonos *cf* értékű szó közül nyilván az a fontosabb, amelyik koncentráltan, kevés dokumentumban, de azokon belül nagy gyakorisággal fordul elő, semmint az, amelyik sok dokumentumban alacsony gyakorisággal. Jelöljük, tehát n_k-val azon dokumentumok számát, amelyben a t_k szó előfordul. Ekkor az $n_k/N =: df$ hányados, amit *dokumentumgyakoriság*nak (document frequency, df) neveznek, jól jellemzi a szó ritkaságát a korpuszban. Ez az érték megadja, hogy mekkora megkülönböztető ereje van, avagy mennyire tekinthető indikátornak a szó jelenléte (és előfordulásainak száma) a dokumentum tartalmára vonatkozóan. A súlyozási sémákban inkább a dokumentumgyakoriság *inverz*ével számolnak (inverse document frequency, idf), amit legtöbbször az

$$idf(t_k) = \log(N/n_k) \quad \textit{dokumentumgyakoriság inverze} \tag{2.8}$$

[11] Több sikertelen kísérlet volt a *stopszó* kifejezés magyarítására: *funkciószó*, *töltelékszó*, *tiltott szó*, amelyek a különböző felhasználási aspektusokat igyekeztek megragadni, de véleményünk szerint az angol kifejezés átfordítása a legkevésbé félreérthető.

képlettel adnak meg.[12] Az idf tényező használatával a vektortérmodell tengelyeit a reprezentált szavak relevanciájával arányos mértékben nyújthatjuk meg. Így kapjuk a leggyakrabban használt *tf-idf súlyozás*t (az angol elnevezés rövidítéséből: term frequency & inverse document frequency)[13]

$$d_{ki}^{(6)} = f_{ki} \cdot idf(t_k). \tag{2.9}$$

A tf-idf súlyozás értéke tehát

1. magas lesz azon szavak esetében, amelyek az adott d_i dokumentumban gyakran fordulnak elő, míg a teljes korpuszban ritkán (nagy a megkülönböztető képességük);
2. alacsonyabb lesz azon szavak esetén, amelyek a d_i dokumentumban ritkábban, vagy a korpuszban gyakrabban fordulnak elő;
3. és kicsiny (akár zérus) lesz azon szavakra, amelyek (szinte) a korpusz összes dokumentumában előfordulnak.

A (2.9) képletbe a szógyakoriság helyett alternatív kifejezést, pl. n_{ki}-t, vagy (2.7)-t téve, további súlyozási sémákat kaphatunk [1, 54, 161]. Gyakran használják például az ún. *normalizált tf-idf súlyozás*t, ahol n_{ki} használata mellett $\|\cdot\|_2$-es norma szerint normalizálják a teljes súlyértéket:

$$d_{ki}^{(7)} = \frac{n_{ki} \cdot idf(t_k)}{\sqrt{\sum_{i=1}^{N} (n_{ki} \cdot idf(t_k))^2}}. \tag{2.10}$$

Az *entrópiasúlyozás* a szavak információmennyiségét is figyelembe veszi a szó-dokumentum mátrix értékeinek kiszámolásánál:

$$d_{ki}^{(8)} = w_{ki} \left(1 + \frac{1}{\log N} \sum_{j=1}^{N} \left[\frac{n_{kj}}{n_k} \log \left(\frac{n_{kj}}{n_k} \right) \right] \right), \tag{2.11}$$

ahol $\frac{1}{\log N} \sum_{j=1}^{N} \left[\frac{n_{kj}}{n_k} \log \left(\frac{n_{kj}}{n_k} \right) \right]$ a t_k szó átlagos entrópiája a korpuszban. A szó-dokumentum mátrix értékeinek (2.11) kifejezéssel való meghatározása nagyobb számítási igénnyel bír az eddig ismertetetteknél, mivel a számolásnál szükség van

[12] Az *idf*-nek több alternatív definíciója létezik, amelyek a normálás, a logaritmus vagy más tényezőben térnek el a (2.8) kifejezésben megadottól, pl.: $idf(t_k) = 1 + \log(n_k/N)$.

[13] Magyar megfelelője a *terminusfrekvencia és inverz dokumentumfrekvencia* elnevezés lehet.

az előfordulások számára, n_k-ra. Ügyes implementációval azonban a teljes dokumentumgyűjteménynek csupán egyszeri beolvasásával is megoldható a számítás. Az erre fordított energia kifizetődhet, ugyanis bizonyos szövegbányászati feladatok, pl. osztályozás esetén az entrópiasúlyozás alkalmazása jelentősen növelheti a hatékonyságot [54, 186].

2.2.4. A szöveg felbontása és a szótár felépítése

A vektortérmodell szerinti reprezentáció felírásához a súlyozási séma kiválasztásán kívül több előfeldolgozási lépést kell végrehajtani. A szöveges dokumentumokat először is fel kell bontani reprezentációs egységekre. Ez többnyire kimerül a dokumentum szavakra történő bontásában, de egyes feladatoknál — lásd pl. kivonatolás, 7. fejezet — bekezdés, illetve mondatszintű egységek kezelését is elő kell készíteni. A vektortérmodell alkalmazásához meg kell határozni, hogy melyek a szótár elemei. Ez a legegyszerűbb esetben a dokumentumgyűjteményben található egyedi szavak kinyerését jelenti, de gyakran már ennél a lépésnél célszerű csökkenteni a vektortér méretét, azaz az indexelt szavak számát. A következőkben áttekintjük a szövegek különböző reprezentációs egységekre való felbontását, kiemelten tárgyalva azt a kérdést, hogy mikor mit érdemes indexelni, és ez milyen következményekkel jár a szótár felépítésénél.

2.2.4.1. Strukturális szegmentálás

A szöveges dokumentumok felépítése hierarchikus. A dokumentum összetettségétől, hosszától stb. függően egy szövegben számos hierarchiaszint előfordulhat, pl. strukturális egységek (kötet, rész, fejezet, szakasz, pont stb.), bekezdések, mondatok, szavak, illetve szintaktikailag releváns egységek. A strukturális egységek feltérképezésének lehetőségei a szöveg formátumától függenek. Ennek a könyvnek a kézirata LaTeX-ben készült, ezért a strukturális egységek meghatározása és a strukturált *dokumentumtérkép* automatikus elkészítése egyszerűen megvalósítható feladat, de egy txt, nem szabványos címkezeléssel készült MS-WORD-, vagy PDF-formátumú dokumentum esetén ez már általában sokkal nehezebben megy. Szintén a formátumtól, valamint a dokumentumot generáló szerkesztőalkalmazástól függ a szöveg bekezdések szerinti felbontásának megoldása, bár ez általában egyszerűen megoldható. A szöveg strukturális egységekre, ill. bekezdésekre bontásával kapcsolatos problémákat egy szövegbányászati rendszerben mindig a rendelkezésre álló dokumentumok és a keletkezésükre vonatkozó egyéb adatok alapján egyedileg kell megvalósítani.

2.2.4.2. Mondatokra bontás

Egy dokumentum mondatokra bontása hatékonyan automatizálható, de korántsem triviális feladat. Már az is eleve kérdéseket vet fel, hogy mi tekintendő pontosan egy mondatnak [79]. Az első kézenfekvő ötlet a szövegben található mondatzáró írásjeleknél[14] való szegmentáció, ami azonban számos ponton finomításra szorul, hiszen rövidítésekben, dátumokban, sorszámokban, IP- és e-mail címekben, URL-ekben stb. egyaránt előfordulhat a pont (.) karakter.

Leggyakrabban szabály alapú megoldást alkalmaznak, amelyben a potenciális mondathatár környezetének tulajdonságai alapján hoz döntést az algoritmus. A szabályok a különböző típusú „álmondathatár"-okra utaló vagy azt cáfoló tulajdonságokat írnak le, amelyeket az algoritmus a mondathatárjelölt kiértékelésénél illesztéssel ellenőriz. A tulajdonságok egy része felszíni jellemzőkön alapul (nagy kezdőbetű, csupa nagybetű, szám és betű vegyesen van a szóban, szóközi pont vagy más írásjel stb.), másik listabeli előfordulásokat ellenőriz (rövidítéslista) — ezek a tulajdonságok tehát nyelvfüggőek. A szabályok tartalmazzák, hogy a mondathatárjelölt pozíciójához képest milyen távolságban lévő szót (pontosabban tokent, ld. 2.2.4.3. alpont) kell vizsgálni, valamint a szabályt jellemző súlyértéket is megadhatnak. Több illeszkedő szabály esetén a szabályok súlyértékei alapján (pl. additív vagy multiplikatív módon) számolható a végső érték. Mindenképpen akadnak azonban olyan esetek, amikor a szabály alapú módszer nem képes jól dönteni, mert csak a szöveg értelméből derül ki, hogy az adott helyen mondathatár van-e, vagy sem:

A következő megálló a Margit krt. Moszkva tér felőli végén lesz.
A következő megálló a Margit krt. Moszkva térnél majd át kell szállnia.

Egy egyszerű reguláris kifejezéseken alapuló szabályrendszer a könyv honlapján megtalálható. Itt részletes lista foglalja össze a leggyakoribb álmondathatárokat, amelynek alapján a kiinduló szabályrendszer bővíthető. Magyar nyelvű szövegekre a HunToken alkalmazás a legismertebb, ennek a dokumentációja azonban csak a közelmúltban, 2007 tavaszán lett elérhető [136].[15]

[14] Ez a feladat nyelvfüggő, pl. spanyolban a kérdő, illetve felkiáltó mondatok rendre a mondatzáró írásjel fordítottjával kezdődnek: ¿, ¡.

[15] `mokk.bme.hu/resources/huntoken/huntoken.pdf`

2.2.4.3. Tokenizálás

*Tokenizálás*nak hívjuk azt a szövegfeldolgozási lépést, amely során a dokumentumot *token*ek sorozatára bontjuk. Tokennek nevezzük egy karaktersorozat konkrét dokumentumbeli előfordulását, míg *típus*nak hívjuk az azonos karaktersorozatot tartalmazó tokenek osztályát. A típusok összessége alapján állítjuk elő — esetleg további feldolgozási lépések alkalmazásával — a *szótár*at, azaz a típusok szótárelemjelölteknek tekinthetők. A tokenizálás a legtöbb szövegbányászati feladatnak részét képezi, a kereséstől az osztályozáson át a kérdésmegválaszolásig. Nézzünk rá egy példát:

Itt egy frappáns példamondat.

Itt	egy	frappáns	példamondat

A tokenizálás első ránézésre triviális feladatnak tűnik: elhagyjuk az írásjeleket, és a szöveget a szóközök mentén szavakra bontjuk. Az adott feladattól függően aztán számos kérdés vetődik fel: hogyan kezeljük a kötőjelet (*Voltaire-rel*, *Guy-Lussac-kal*, *Boyle–Mariotte-törvény*, *adatbázis-kezelő*), az aposztrófot (*I'm*, *l'Équipe*, *nemt'om*), vagy egyéb írásjelet tartalmazó szavakat (*ML@UToronto*, *C++*, *W.C.*). Szintén fontos kérdés, hogy a szintaktikailag egy egységnek tekinthető szószekvenciákat (*Bartók Béla*, *Kispál és a Borz*, *1848. március 15.*) egy tokenként kezeljük-e.[16] Természetesen a megoldandó feladatok köre nyelvenként változik — angol nyelv esetén máshogy kell megoldani a kötőjelek helyes kezelését, mint magyar szöveg tokenizálásánál. Hasonlóan, azoknál a nyelveknél, ahol a szóösszetételeket egybeírják (német, magyar, finn), kitüntetett figyelmet kell szentelni a jelenség kezelésére [82].

A tokenizálás a felsorolt szempontok összességének figyelembevételével már meglehetősen komplikált feladat, aminél a hibamentes működés nagy adathalmazon gyakorlatilag elérhetetlen.

Egy alternatív lehetőség bizonyos feladatok esetén a dokumentumok *karakter n-gramm*okra való bontása. Ekkor a dokumentumot nem önmagában értelmes egységekre bontjuk, hanem n hosszúságú sztringeket állítunk elő belőle az alábbi módon:

karakter 4-gramm
kara, arak, rakt, akte, kter, ter␣, er␣4, r␣4-, ␣4-g, 4-gr, -gra, gram, ramm

[16] Az ilyen többszavas kifejezéseknek csak az utolsó szavát toldalékoljuk, ezért szintaktikailag egybe tartoznak.

A megoldás nagy előnye, hogy nyelvfüggetlen. Gyakran ezt a módszert használják szótárépítéshez az olyan távol-keleti nyelvek esetén (kínai, japán, koreai), ahol nincsenek szóközök a szavak elválasztására. Európai nyelveknél azonban közvetlenül nem vethető be olyan alkalmazásokban (pl. keresés), ahol a szótár elemeinek értelmes nyelvi egységnek kell lenniük. Nagyon jó eredményeket ad viszont ez a megközelítés egyszerű osztályozási problémák esetén, pl. nyelvmeghatározásnál — ld. a 109. oldalt.

A tokenizálás eredményét felhasználva előállítható a nyers szótár, azaz a típusok listája, ebből alakítjuk ki — általában különböző szűrőműveletek elvégzésével — a végső szótárat. A továbbiak során, ha ettől eltérően nem jelezzük, a szótár elemeit *szavak*nak tekintjük, tehát a vektortérmodellben a szótár elemei lesznek a szöveget reprezentáló egységek.

2.2.4.4. Stopszószűrés

A szövegbányászati feladatok többségénél a gyakran előforduló, tartalmi információt egyáltalán nem tartalmazó, megkülönböztető képesség nélküli ún. *stopszavak*at már a tokenizálás fázisa után eldobják. Mivel stopszavak szinte minden dokumentumban előfordulnak, ezért a szótárból való elhagyásukkal lényegesen csökkenthető a szó-dokumentum mátrix nemzérus bejegyzéseinek száma.

A potenciális stopszavakat a dokumentumgyűjtemény szógyakorisági adatai alapján állítják elő, amit azután manuálisan ellenőriznek. A stopszólista mérete szintén függ az alkalmazás jellegétől. Keresés esetén többnyire igen kevés szót hagynak el, ha egyáltalán alkalmazzák ezt a megoldást. Bár a keresés minőségét nem befolyásolja jelentősen, ha a stopszavak nincsenek a szótárban, azaz *csak* ezekre keresve nem kapunk eredményt, de ennek ellenére akadnak olyan kifejezések — pl. közismert dal- vagy verscímek: *Lenni vagy nem lenni...*, *Let it be*, *The only way is up* —, amiket ezzel kizárunk a keresésből. Szintén gondot jelent a kifejezéskeresésnél, ha a stopszavak nincsenek indexelve, pl. *The lord of the rings*, *A hölgy vagy a tigris* — ezt részletesebben a 8. fejezetben tárgyaljuk. Ezért a nagy keresőszolgáltatóknál megfigyelhető az a trend, hogy egyre kisebb stopszólistákat alkalmaznak. Tíz éve még 200–300-as lista volt a jellemző, manapság legfeljebb 7–12 szó [125]. Ezt egyrészt az indexállományok tömörítési módszereinek fejlődése, másrészt a tár- és memóriakapacitás bővülése, illetve drasztikus költségcsökkenése magyarázza.

Sokkal kevésbé érzékenyek a stopszavakra a pontos egyezést nem igénylő szövegbányászati feladatok (pl. osztályozás, csoportosítás). Ekkor akár 500–1000 szavas stopszólistákat is alkalmaznak nyelvtől és az alkalmazott nyelvtechnológi-

ától függően, sőt a vektortérmodell dimenzióját további, csak az adott korpuszban stopszónak tekinthető, kicsiny megkülönböztető képességű szavak elhagyásával is gyakran csökkentik (ld. még a 2.3. szakaszt).

2.2.5. Lemmatizálás és szótövezés

A legtöbb nyelvben a szavaknak vannak toldalékolt vagy módosított alakjaik is. Különösen jellemző ez az olyan gazdag inflexiós és derivációs morfológiájú nyelvekre, mint a magyar, de az angol nyelv morfológiája is tartalmaz ragokat, ill. képzőket. A vektortérmodell kialakításánál kézenfekvő ötlet, hogy az azonos szavaknak különböző szóalakú előfordulásait közös *kanonikus alak*ban vonjuk össze, és összesítve reprezentáljuk. Ezzel ugyan a szóalakra vonatkozó információt elveszítjük,[17] de a vektortérmodell méretét jelentősen csökkenthetjük, mivel a szóalakonkénti dimenzió helyett a kanonikus alakhoz csak a vektortér egy dimenziója tartozik. A csökkentés mértéke a nyelv morfológiájának gazdagságától, a korpusz méretétől és a szöveg típusától is függ. Angol nyelvre kb. 40–70%-os csökkentést jelent ez a módszer, de magyar nyelvre akár a 90%-ot is meghaladhatja a csökkenés mértéke, azaz az eredeti modellméret akár a tizedére is csökkenhet. A kanonikus alakú reprezentációt — bár a modell méretét jelentősen befolyásolhatja — itt, és nem a dimenzióredukciós módszereknél tárgyaljuk, mivel a szótár felépítésére is jelentős hatással van.

A kanonikus alak meghatározásához szócsonkolást, illetve -átalakítást végző nyelvtechnológiai eszközöket alkalmazunk. A szavak csonkolására két megközelítést használ a szakirodalom. A nyelvészetben alkalmazott terminológia *lemmatizálás*nak nevezi a szó *lemmá*jának (normalizált, vagy szótári alakjának) előállítását, illetve meghatározását. Az alkalmazásorientált számítógépes nyelvészeti szakirodalom *(szó)tövezés*nek[18] hívja azt az eljárást, amikor az adott szó *szótövé*nek meghatározása a cél.

A két eljárás között az a különbség, hogy míg a nyelvészeti motivációjú lemmatizálás mindig *értelmes szóalakot* állít elő — hiszen a lemma definíció szerint értelmes szótári szó —, addig a szótövezés során jellemzően a szó csonkolása történik, amely gyakran nem értelmes szótári alakot ad eredményül. Természetesen az is gyakran előfordul, hogy a lemmatizálásnak és a szótövezésnek azonos ered-

[17] Ez a legtöbb alkalmazásban azonban nem jelent hátrányt, sőt a kanonikus alakban történő tárolás gyakran előnyös is.

[18] Az angol terminológia *stemming*, illetve *stemmer*, amelyet gyakran a magyar szakzsargon is átvesz.

ménye van. Fontos különbség még a két módszer között, hogy míg a nyelvészetileg motivált lemmatizálásnál az eredmény — amely több elemű is lehet, hiszen a lemmatizálás nem egyértelmű, ld. például *termet* → *terem, termet*; *falunk* → *falu, fal* — jól definiált, addig a szótövezés eredménye függ az adott módszertől, aminek köszönhetően egy szótövező eljárás rendszerint csak önmagával konzisztens.

Mindkét megközelítés esetén el kell dönteni, hogy csak az inflexiós toldalékokat (ragokat és jeleket) vagy képzőket is levágjuk. Az előbbi esetben az *igazságoságainkért* kanonikus alakja az *igazságoság*, utóbbi esetben az *igaz*.

Vizsgáljuk meg a lemmatizálás és a szótövezés különbségeit az eredmények tükrében is. Angolban például a *works*, *worked*, *working* szavaknak egyaránt *work* a lemmája és a szótöve, ugyanakkor a *loves*, *loving*, *loved* szavak lemmája *love*, míg szótöve *lov* — pl. a klasszikusnak nevezhető Porter-szótövező [149] alapján (ld. a 45. oldalon). Ugyanez a kettősség a magyar esetén is megfigyelhető. Míg a *munkát*, *munkám* stb. szavak lemmája és szótöve egyaránt *munka*, a *lovak*, *ló*, *lovát* stb. szavak lemmája *ló*, míg szótöve bizonyos szótövezők használata esetén *lo*, más esetben pedig *ló*. Általában a szövegbányászati alkalmazások igényeinek tekintetében kellően hatékony és pontos megoldást adnak a szótövező eljárások is, ezért a továbbiakban elsősorban ezekre koncentrálunk.

A szótövezők típusai és tipikus hibái. A szótövezést több, lényegileg különböző megközelítéssel lehet megvalósítani.

- Algoritmikusan, nyelvspecifikus átírószabályok implementálásával [150]. E módszerek gyorsak, egy jól megírt algoritmus egymillió szót néhány másodperc alatt feldolgoz egy mai átlagos személyi számítógépen. Hibaarányuk természetesen nyelv- és szövegfüggő, de általában néhány (1–5) százalékra tehető, azaz az esetek nagy többségében jól használható eredményt adnak. Hatékonyságuk kivételszótár alkalmazásával nagymértékben javítható. Megvalósításuk nehezebb, mint a következő pontban ismertetett szótár alapú módszereké, nyelvészeti ismereteket igényel. Angol nyelvre a legelterjedtebb Porter algoritmusa [149], melynek működési elvét több nyelvre adaptálták (ld. a Snowball szótövező nyelvet, ill. programcsomagot [150]). További fontos módszereket alkotott Lovins, Paice és Husk, illetve Krovetz [105, 123, 140]; ezeket bővebben ismertetjük a továbbiakban ill. a könyv honlapján.
- A szavakat és szótövüket, szótöveiket tartalmazó szótár alkalmazásával. Ezek jobban kezelik a rendhagyó eseteket, mint a szabály alapú módszerek, vi-

szont csak azokra a szavakra működnek, amelyek szerepelnek a szótárban. A módszer keretében könnyen megoldható, hogy egy szóhoz több szótövet is hozzárendeljünk [191] (például *palánk* → *pala, palánk*). Hátránya viszont, hogy a szótár rendszeres karbantartást igényel, különben a nyelv folyamatos változása miatt könnyen elavulttá válhat.

- Egyéb módon, legtöbbször statisztikai módszereket felhasználva. A [12]-ben említett módszer a szavak lehetséges felbontásainak gyakoriságát ellenőrzi egy szavakat és toldalékokat is tartalmazó szótárban. Egy másik megoldás a szótár klaszterezése alapján állapítja meg, hogy mely szavakhoz tartozik azonos tő. Ez a módszer nyelvfüggetlen, bármiféle nyelvészeti ismeret nélkül alkalmazható tetszőleges nyelvre.

A megközelítések közötti határok nem élesek: a szabály alapú szótövezőket gyakran egészítik ki kivételszótárral, valamint a szótár alapú szótövezők is használhatnak egyszerűbb nyelvi szabályokat. A természetes nyelvek bonyolultsága miatt (vö. ezek a 0-ás Chomsky-nyelvosztályba tartozó nyelvek) nem várható el, hogy egy szótövező algoritmus hibátlanul működjön. Alapvetően háromféle tévesztést különböztetünk meg [150]:

- *Alultövezés*ről beszélünk, ha két szóhoz, amelyek jelentése ekvivalens a feldolgozás szempontjából, az algoritmus különböző tövet rendel.
- *Túltövezés* akkor fordul elő, ha túl sokat vágunk le egy szóból, és emiatt két különböző jelentésű szóhoz ugyanazt a szótövet rendeljük. Például *házas* → *ház*, *házak* → *ház*
- *Félreelemzés* pedig az az eset, ha olyan végződést vágunk le, ami valójában a tő része. Például *német* → *nem*, *nemzet* → *nemz* [főnév].

Az utóbbi két hibatípus gyakorisága kivételszótárral csökkenthető.

Szótövezők kiértékelése. Szótövezők kiértékelésére leggyakrabban a Paice által bevezetett hibaszámláláson alapuló alul- és túltövezési indexeket használják [140]. Az indexeket szóhalmazokra értelmezik. Az *alultövezési index*et (under-stemming index, UI) azon szópárokra határozzák meg, amelyeket közös tőre kell hoznia az algoritmusnak, és a sikeresen közös tőre redukált szópárok arányából számolják a következőképpen:

$$\mathrm{UI} = 1 - \frac{\#\text{sikeresen közös tőre redukált szópárok}}{\#\text{közös tővel rendelkező szópárok}}. \tag{2.12}$$

Hasonlóan, a *túltövezési index*et azon szavakra határozzák meg, amelyeknek páronként különböző szótöve van. Ennek értékét a tövezés után is külön szótővel

rendelkező párok arányából számolják:

$$\mathrm{OI} = 1 - \frac{\#\text{sikeresen eltérő tőre redukált szópárok}}{\#\text{eltérő tővel rendelkező szópárok}}. \tag{2.13}$$

A két mérték dichotóm kapcsolatban van egymással. Ha az alultövezési indexet csökkentjük, akkor általában nő a túltövezési index és fordítva. A *szótövező súlyát*, azaz azt a tulajdonságát, hogy mennyire hajlamos a szavak redukciójára, az $\mathrm{SW} = \mathrm{OI}/\mathrm{UI}$ (stemming weight) hányadossal mérhetjük. Egy szótövező *gyenge*, ha kevés alakot von össze közös szótőre, és módon *erős* vagy *agresszív*, ha jellemzően sok alakot redukál közös tőre. A szótövezőnek ezt a tulajdonságát az SW értéken kívül az alábbi mennyiségekkel jellemezhetjük [68]:

$$\text{átlagos tövezési index} = \frac{\text{szótár mérete tövezés előtt}}{\text{szótár mérete tövezés után}} =: \frac{|T|}{|T|_{\mathrm{red}}}, \tag{2.14}$$

$$\text{tömörítési arány} = \frac{|T| - |T|_{\mathrm{red}}}{|T|}. \tag{2.15}$$

Minél nagyobb az átlagos tövezési érték, illetve minél kisebb a tömörítési arány, annál erősebb a tövező. A szótövezők erősségét a törölt karakterek számával is lehet jellemezni. Ezt a módosított Hamming-távolság alapján (ld. (3.7) képlet) határozzák meg.

2.1. PÉLDA. Tekintsük az alábbi szavakat: *eszik, eszünk, evett, eszes, étel.* Szótövezőként használjuk a truncate(n)-et, amely n karaktert hagy meg a szó prefixéből. Ekkor a truncate(3) tövező rendre az alábbi alakokat állítja elő: *esz, esz, eve, esz, éte.* Az alultövezési index ekkor

$$\mathrm{UI} = 1 - \frac{1}{3} = \frac{2}{3},$$

hiszen az első három szóból alkotott három párnak azonos a töve, de ebből csak egyet hozott azonos alakra. A túltövezési index

$$\mathrm{OI} = 1 - \frac{5}{7} = \frac{2}{7},$$

mivel az utolsó két szó egymástól és az első háromtól is különböző tővel rendelkezik, de ebből 2 párat azonos alakhoz rendelt a tövező. Továbbá $|T| = 5$, $|T|_{\mathrm{red}} = 3$, így a (2.14) és a (2.15) képletek értéke rendre $\frac{5}{3}$ és $\frac{2}{5}$.

2.2.5.1. Szabály alapú szótövezők angol nyelvre

Angol nyelvre számos szótövezőt alkottak a 60-as évek vége óta. Ezek közül egyszerűsége és hatékonysága miatt a *Porter-féle algoritmus* a legelterjedtebb [149]. Népszerűségét annak is köszönheti, hogy honlapján[19] számos programozási nyelvre létezik publikus, letölthető implementációja.

A szótövezést átírószabályok alapján 5 fő lépésben hajtja végre az algoritmus. A lépéseket rögzített sorrendben alkalmazza, és minden lépés alternatív szabályokat tartalmaz. A szabályok szóvégződésekre vonatkozó karaktercseréket írnak elő, amelyekben a szóban szereplő magánhangzó-, illetve mássalhangzó-szekvenciákra vonatkozó feltételek is szerepelhetnek. Formálisan tehát a szabályok

$$(\textit{feltétel})\ S_1 \rightarrow S_2 \tag{2.16}$$

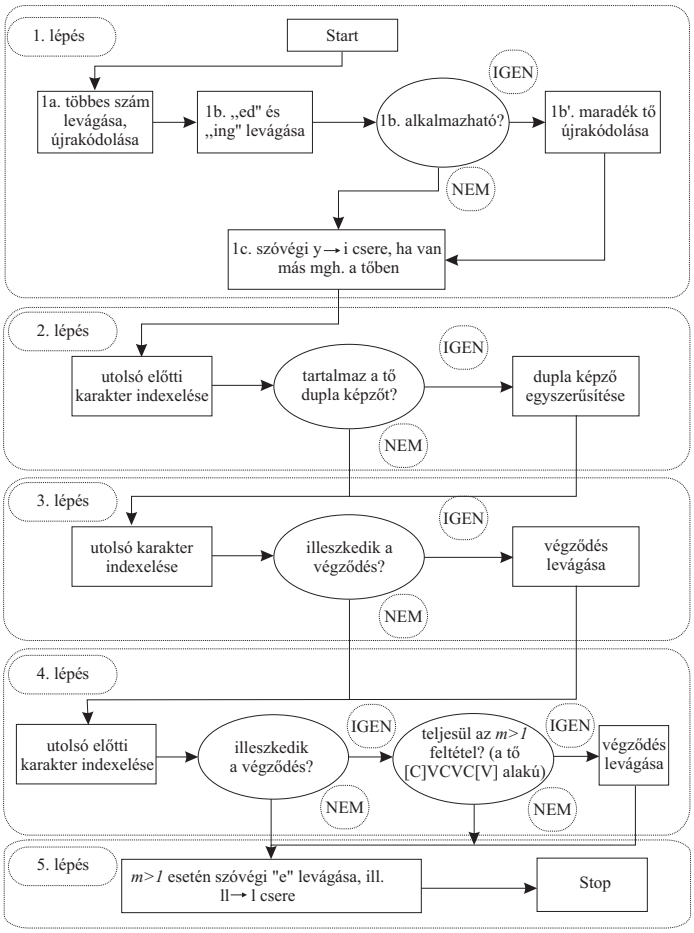
alakúak. (2.16) tehát akkor alkalmazható, ha a szóalak végződése illeszkedik S_1-re, és az S_1 levágása után megmaradt tőre *feltétel* teljesül. Ha egy adott lépés esetén több szabályra is illeszkedik a vizsgált szó, akkor a leghosszabban illeszkedő szabály kerül alkalmazásra. Értelemszerűen, ha egy lépés valamely szabályát sikeresen alkalmazta az algoritmus, akkor az eljárás a következő szabálycsoportra lép. Ha az adott lépésben nincs alkalmazható szabály, akkor a következő lépésre ugrik a szótövező. Az algoritmus az ötödik szabálycsoport után terminál. A végeredményként kapott szóalak többnyire nem azonos a szó lemmájával. Az algoritmus működését a 2.2. ábra szemlélteti, a könyv honlapja pedig példákon keresztül részletesen ismerteti.

Porter kifejlesztett egy általános, szótövezőket leíró nyelvet is, a *Snowball*t [150], amely már nem csak szóvégződéseket, ún. szuffixeket képes levágni, hanem prefixeket is (pl. *legjobb* → *jó*, vagy *gefragt* → *fragen*). Ennek segítségével 14 európai nyelvhez gyártottak szótövezőt, köztük a magyarhoz is [192] (ld. még a 2.2.5.2. alpontot).

Porter tövezője mellett több alternatív angol szótövezőt is érdemes megemlíteni. A legkorábban kifejlesztett angol szótövező Lovins nevéhez kötődik [123], amelyet eredetileg egyaránt szántak információ-visszakeresési és számítógépes nyelvészeti alkalmazásra. A *Lovins-tövező* a maga korában úttörőnek számított, és jelentős motiváló hatása volt a területen folyó későbbi munkákra.

A Lovins-szótövező egy nagyobb korpusz átvizsgálása alapján készült szabály alapú, egymenetes, kontextusérzékeny módszer, amely a Porter-algoritmushoz hasonlóan a leghosszabb egyezés alapján választ illeszkedő szabályt. Az algoritmus

[19] `www.tartarus.org/~martin/PorterStemmer/`



2.2. ábra. A Porter-szótövező működési vázlata

kétlépéses: az első lépésben a leghosszabb illeszkedő, és a feltételnek eleget tevő végződés alapján redukálja a szóalakot, a második lépésben pedig átírószabályokat alkalmazva a végződések átkódolását végzi. A módszer egyik alkalmazási területen sem váltotta be a hozzá fűzött reményeket, ugyanis nyelvészeti alkalmazáshoz a szabályrendszere nem elegendően széleskörű, viszont a korabeli IR-rendszerekben való alkalmazáshoz ez a szabályrendszer — amely 294 végződést, 29 feltételt, és 35 átírószabályt tartalmazott — túl bonyolult, és ezért lassú volt:[20]

[20] Részletes leírás: `snowball.tartarus.org/algorithms/lovins/stemmer.html`

A Lovins-algoritmus nyelvészeti alkalmazásokat megcélzó továbbfejlesztett változata a *Dawson-tövező* [50], amely már mintegy 1200 végződést kezel. Az eredeti módszer megbízhatatlanul működő második átkódoló lépése helyett egy részleges illeszkedésen alapuló eljárást adott.

A *Paice–Husk-szótövező* egy egyszerű szabály alapú iteratív módszer, amelyet 1990-ben [140] publikáltak. A tövező a szavak végéről a végződéseket változó számú lépésben vágja le. Az algoritmus igen agresszív tövező, azaz inkább a túltövezésre hajlamos, mintsem alultövezésre, ezért leginkább indexállományok, szótárak tömörítésére alkalmas.

A szabályok a szóvégződés szerint csoportokba vannak rendezve, ami elősegíti az illeszkedő szabályok hatékony keresését. Minden szabály egységes alakú, és a szóvégződés redukálását vagy cseréjét írja elő. A végződéscsere technikájával az algoritmus elkerüli a szóvégek újrakódolását, s ezzel csökkenti a tövezéshez szükséges lépések számát. Bizonyos szabályok csak az eredeti, még tövezetlen szóra alkalmazhatóak. Az algoritmus négy fő lépésből áll (ld. még a 2.3. ábrát):

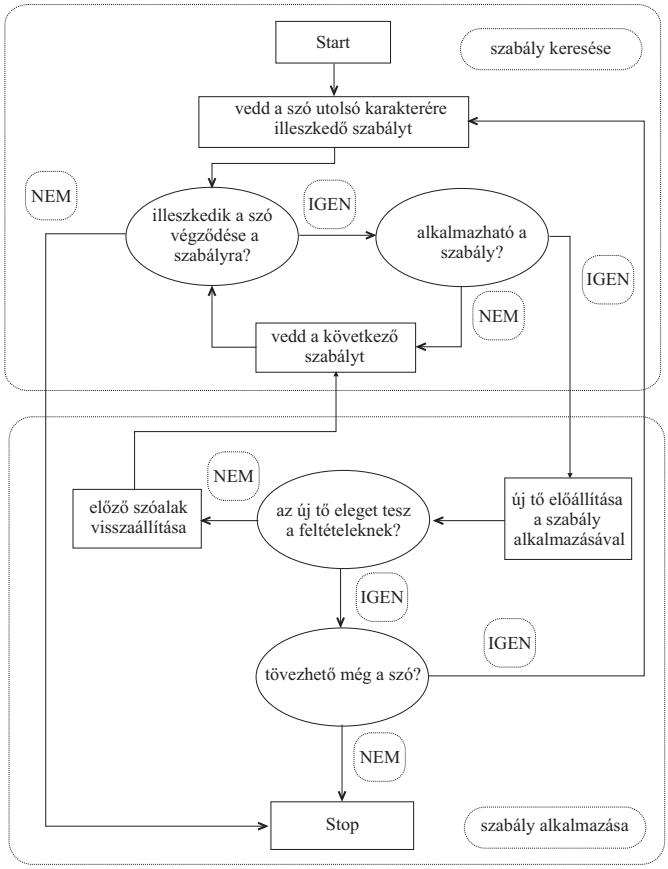
1. A szabálycsoport meghatározása, és azon belül az első szabály kiválasztása (nem minden végződésre van szabály).
2. A szabály alkalmazhatóságának vizsgálata: ha nem illeszkedik a végződésre a szó, ill. az eredetiségi vagy az elfogadhatósági kritérium sérül, akkor a 4. lépés következik.
3. A szabály alkalmazása: ekkor hajtja végre az algoritmus a szóvégződés törlését vagy cseréjét, majd a szabályban kódolt információ alapján terminál vagy az 1. lépésre ugrik.
4. A következő szabály keresése: az adott szabálycsoport következő szabályára lép, ha ilyen nincs, akkor terminál, ha van, akkor a 2. lépéssel folytatódik az algoritmus.

Az elfogadhatósági kritériumok a maradék szóalakra vonatkozó feltételekkel korlátozzák valamelyest a módszer túltövezésre való hajlamát. A módszer részletes működését a könyv honlapján található kiegészítés példákkal ismerteti.

Az ismertetett módszereken kívül még számos szótövező eljárást dolgoztak ki angol nyelvre — pl. Krovetz-tövező [105] —, amelyek ismertetése túlmutat könyvünk keretein.

2.2.5.2. Szótövezők magyar nyelvre

A magyar nyelv morfológiája rendkívül gazdag. Egyaránt jellemző rá a gazdag toldalékolás, az összetett szavak magas aránya és a derivációs (képzett) alakok



2.3. ábra. Paice–Husk-szótövező folyamatábrája

nagy száma. A különböző nyelvi osztályozási rendszerek szerint egy főnévnek 16–24 különböző esete lehet, amit ha a birtokos, személyes ragokkal kombinálunk, akár 1400 különböző szóalakot is kaphatunk. Melléknevek esetén ez a szám a fokozás miatt akár 2700 is lehet, míg igéknél lényegesen csak kevesebb, 59 alak lehetséges. Ezek a számok jól illusztrálják a magyar nyelv morfológiai gazdagságát, és rámutatnak a szótövező eljárások jelentőségére szövegbányászati, illetve információ-visszakereső alkalmazásokban.

Magyar nyelvre elsőként a Morphologic Kft. készített szótövező eljárást. A HelyesLem nevű alkalmazás fejlesztése 1992-ben kezdődött. Mivel a fejlesztő cég csak térítéses formában adja közre a terméket, ezért működési elvéről nem

sok tudható; csak valószínűsíthető, hogy szabály alapú eljárást alkalmaz. A hazai nyelvtechnológiai piacon a Szószablya projekt keretében a BME MOKK által kifejlesztett HunMorph programcsomaghoz [195] kapcsolódó HunStem szabály alapú szótövező terjedt el, amely bárki számára forráskóddal együtt ingyenesen hozzáférhető. A HunStem szótövezőt a morphdb.hu alapú magyar nyelvi erőforrások bemutatásánál ismertetjük (ld. 2.2.6. szakasz). A fentieken kívül még két figyelemreméltó kezdeményezést kell említeni: a Snowball-nyelvre adaptált Tordai-féle szótövezőket [192], illetve az óvatos szótövezőt [190].

Snowball alapú magyar tövező. A *Tordai-féle szótövező* az inflexiós toldalékok levágását végzi el, azaz csak ragokat és jeleket vág le, képzőket nem [192]. A kutatók négy különböző erősségű tövezőt készítettek, amelyeket az annotált Szeged Korpuszon [47] tanítottak be és teszteltek:

1. Light1 – csak néhány gyakori főnévi esetet, többes számot és birtokost kezel;
2. Light2 – az összes főnévi esetet, többes számot és birtokost kezeli;
3. Medium – a gyakori főnévi eseteket, többes számot és birtokosokat, valamint a gyakori igealakokat kezeli;
4. Heavy – a legtöbb inflexiós toldalékot kezeli.

Főnevek esetén az algoritmus 9 lépésben határozza meg a szótövet. Az egyes verziók abban térnek el egymástól, hogy mely lépéseket alkalmazzák, és azon belül milyen toldalékcsoportokat kezelnek. Az algoritmust részletesebben bemutatjuk a könyv honlapján.

A fejlesztők felhívják a figyelmet arra, hogy a Heavy verzió már kimondottan hajlamos a túltövezésre. Homonimák esetén — pl. *nevet*, amely egyaránt tövezhető a *név* és *nevet* szavakra — ez a változat a rövidebbik alakot választja. A két lehetséges alak közül csak további információk, pl. kontextus alapján lehet egyértelműen valamelyik mellett dönteni, ellenkező esetben mindkét alternatívát kezelni kell. Mivel az ismertetett rendszert az egyértelműen annotált Szeged Korpuszon tanították be és tesztelték, ahol az alanyeset után a második leggyakoribb esetrag az akkuzatívusz volt, ezért a fenti példánál a *-t* toldalék levágása mellett döntöttek, ami a szintén igen gyakori *-t* végű igék túltövezéséhez vezetett.

A tövezők összehasonlítását több kiértékelő függvény segítségével végezték el (ld. a 2.1. táblázatot).

Az alultövezési értékek megfelelnek az előzetesen elvártaknak: a legtöbb szót a Light1 szótövező hagyja meg eredeti alakjában vagy tövezi alul, míg ez az érték a Heavy módszer esetén a legkisebb. Ugyanakkor első látásra meglepő eredmény,

2.1. táblázat. A Tordai-féle Snowball alapú szótövezők összehasonlítása [192]

	UI	OI	SW
Light1	0,75	$2,8 \cdot 10^{-6}$	$3,7 \cdot 10^{-6}$
Light2	0,59	$5,3 \cdot 10^{-6}$	$8,9 \cdot 10^{-6}$
Medium	0,64	$8,1 \cdot 10^{-6}$	$1,3 \cdot 10^{-5}$
Heavy	0,53	$1,3 \cdot 10^{-5}$	$2,5 \cdot 10^{-5}$

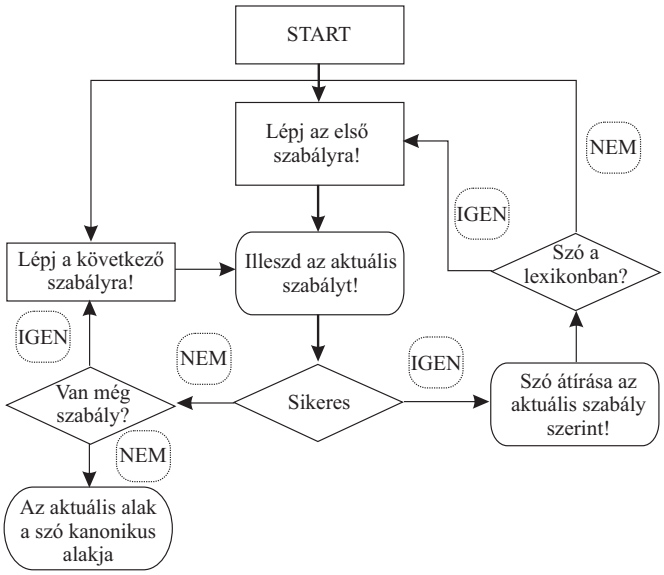
hogy a Medium magasabb alultövezési indexszel rendelkezik, mint a Light2 verzió. Ennek a magyarázata abban rejlik, hogy a tesztszavak között a főnevek szerepeltek legnagyobb arányban (54%), és ezekre a Light2 jobban működik, mint a gyakori főnévi és igei esetekre koncentráló Medium. Összességében tehát a négy verzióból a Light2 és a Heavy bizonyult a leghatékonyabbnak.

A szerzők a 2004-es CLEF (Cross Language Evaluation Forum) angol nyelvű, valamint a 2005-ös CLEF magyar nyelvű korpusza alapján végzett kísérleteikben azt is megvizsgálták, milyen jelentősége van a szótövezésnek a tipikus információ-visszakeresési feladatok esetén. A stopszavak eltávolítása után az angol korpusz 65%-a főnév, 12%-a melléknév és 10%-a ige volt, ugyanezek a számok 60%, 23%, 17% a magyarra. A Light2 és Heavy szótövezők magasabb felidézést biztosítanak, mint egy szótövező nélküli, illetve a másik két verzió által tövezett változatok. Ugyanakkor a Heavy szótövezőnek negatív hatása van a pontosságra, mivel hajlamos a túltövezésre, ezért az információ-visszakereső alkalmazásokra a Light2 verziót javasolták, és ezt is alkalmazták a 2005-ös CLEF magyar anyagainak feldolgozásánál.

Az óvatos szótövező. Az *óvatos szótövező* (timid stemmer) szótár és szabály alapú eljárás [190]. Az elnevezés arra utal, hogy a szóalakok redukálását csak akkor végzi el, ha az így kapott alak szerepel a szótárban. Legyen adott egy kiindulási lexikon[21] és a toldalékok levágását előíró szabályrendszer. Az eljárás sorrendben illeszteni próbálja az átírószabályokat a vizsgált szóra, és amennyiben sikerül, megvizsgálja, hogy az így kapott szóalak szerepel-e a lexikonban. Amennyiben igen, alkalmazza az átírószabályt, és az így kapott szóra rekurzívan próbálja a lehető legrövidebb kanonikus alakot megtalálni. Amennyiben nem, további il-

[21] Ez nem feltétlenül azonos a vektortér egyes dimenzióit reprezentáló szótárral, ezért használjuk ezt a terminológiát.

lesztésekkel kísérletezik. Az átírószabályok sorrendje egyben prioritást is jelent. Ha az illesztés egyik lehetséges átírószabály esetén sem sikeres, akkor a szó az eredeti alakjában bekerül a lexikonba (ld. 2.4. ábra).



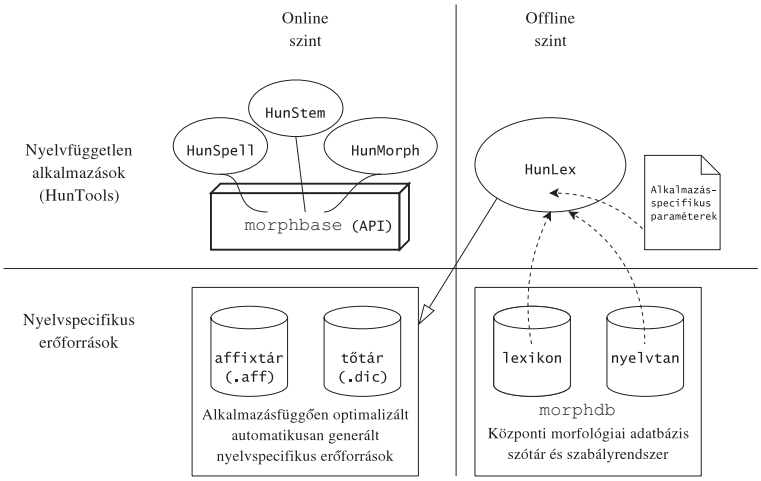
2.4. ábra. Az óvatos szótövező működési elve

Ilyen módon a szöveg maga gazdagítja a lexikont, lehetőséget nyújtva speciális szókészletű szövegek pontosabb szótövezésére. Az átírószabály vonatkozhat szuffixek és prefixek levágására is. A lexikonban lehetőség van dzsókerkarakterek (helyettesítő jelek; wildcard) használatára, valamint kivételek és szinonimák megadására is. Ez utóbbi egyben egy alternatív lehetőséget is kínál a nyelvi alapú dimenzióredukcióra.

Az óvatos szótövezőnek előnye, hogy nincs hozzá feltétlenül szükség kiinduló lexikonra, hiszen azt a korpusz alapján is létre tudja hozni. Természetesen egy átfogó kiinduló lexikon (illetve egyéb nyelvspecifikus adatok, pl. stopszavak listájának megadása) növeli a szótövezés hatékonyságát. Az eljárás könnyen adaptálható bármely nyelvre. Az óvatos szótövezőt elsősorban jó hibatűrésű szövegbányászati eljárásokhoz javasolt alkalmazni, ahol a szótövezés pontossága mellett az eljárás végrehajtásának sebessége az elsődleges, hiszen lényegesen gyorsabb végrehajtást tesz lehetővé, mint a komolyabb segédeszközökkel dolgozó HunStem szótövező.

2.2.6. Morphdb.hu alapú magyar nyelvi erőforrások

A magyar nyelv jelenleg publikusan hozzáférhető legteljesebb, elméleti alapokon álló morfológiai leírása a morphdb.hu adatbázis és a hozzákapcsolódó eszközök.[22] A HunLex keretrendszerben leírt adatbázis több alkalmazásnak — helyesírás-ellenőrző, szótövező, morfológiai elemző stb. — elsődleges nyelvi erőforrása. A rendszer architektúráját a 2.5. ábra ismerteti.



2.5. ábra. Az elemzési technológia komponenseinek architektúrája

HunLex. A rendszer megalkotásakor a legfontosabb szempont a kényelmes bővíthetőség és fenntarthatóság volt [193]. Ennek érdekében a nyelvi (morfológiai és lexikai) információkat külön komponensben lehet megadni, és rugalmasan módosítani, és egy ettől elkülönített rétegben lehet ezek alapján az elemzőalkalmazások erőforrásait előállítani. A kétféle szintű erőforrás között a HunLex konfigurálható előfeldolgozó rendszer közvetít.

A számítógépes feldolgozás során további fontos szempont a futásidejű elemzés hatékonysága, ami viszont az adott feldolgozási feladat jellegétől függ. Például egy helyesírás-ellenőrző alkalmazásnál nem lényeges, hogy mi a vizsgált szó töve, és milyen toldalékai vannak, ezek pontos meghatározása csak hatékonysági kérdésként merül fel. Ugyanakkor egy szótövezőnél ezek már releváns kérdések, de még mindig nincsen szükség a teljes morfológiai elemzői eszköztárra, hiszen

[22] mokk.bme.hu/resources/hunmorph/index_html

nem kell például kezelni a toldalékok többértelműségét.[23] A kiinduló morfológiai adatbázis formátumát azonban lehetetlen az egyes elemzési technológiák igényeihez optimalizálni. Különböző elemzőalkalmazásokhoz tehát más és más erőforrás az optimális, előállításukat azonban érdemes egy központi adatbázisból automatikusan végezni. Végül, de nem utolsósorban fontos szempont a rugalmas alkalmazásfüggő erőforrás-generálás. Egy morfológiai elemzőtől nagyobb rugalmasságot várunk el az akadémiai helyesírási szabályzat követésében, mint egy helyesírás-ellenőrzőtől. Keresési alkalmazásnál, indexelésre használt szótövezőnél többnyire a gyenge szótövező előnyösebb, míg osztályozásnál vagy csoportosításnál kedvezőbb lehet egy agresszív, a képzők levágását is végrehajtó erős szótövező. Ez úgy érhető el, ha a megfelelő erőforrás előállításakor szigorú és engedékeny elemzők egyaránt előállíthatóak a központi adatbázisból.

A HunLex bemenete egy központi nyelvi adatbázis, kimenete pedig a MySpell/ISpell helyesírás-ellenőrző által is használt formátum szerinti [101, 193], tőtár (dic), illetve toldaléktár (affixumtár – aff) (ld. még a 2.5. ábrát). Az erőforrás-generálás számos paraméter mentén konfigurálható, pl.

- beállítható, hogy helyesírás-ellenőrzés, tövezés, illetve morfológiai elemzés számára optimalizált kimenet készüljön;
- széles skálán szabályozható az előállított elemző erőssége;
- kiválasztható, hogy milyen mélységű elemzőt készítsen a rendszer;
- megválasztható az erőforrások kimeneti formalizmusa.

Morphdb.hu. A morphdb.hu adatbázis a nyelv szókészletét és morfológiájának leírását tartalmazza, amelynek alapján a HunLex képes előállítani a megfelelő tőtár- és toldaléktár-állományokat. A nyelvtan morfológiai operációkat formalizáló szabályrendszerként van definiálva. Az egyes szabályok egy toldalék tulajdonságait írják le, ahol a különböző toldalékalternatívák (pl. *-n, -on, -en* vagy *-ön*) érvényességét a szóalakra vonatkozó jellemzők együttese határozza meg. A jellemzőket a szókészletet leíró lexikon tartalmazza. Az adatbázis részletes leírása [194]-ben megtalálható. Szótári anyagának elkészítéséhez három önmagában is nagy lefedettségű elektronikus szótárat — a magyar Ispell szótárat, az Elekfi-féle Magyar Ragozási Szótár digitalizált változatát, és az ún. FKP-szótár anyagát — használták fel. A teljes anyag így több mint 120 000 lemmát tartalmaz.

[23] Nem fontos, hogy a *mentek* alak a *megy* ige múlt idejű többes szám harmadik személyű (3pl) alakja, vagy jelen idejű többes szám első személyű (1pl) alakja).

Nyelvtechnológiai eszközök: HunSpell, HunStem, HunMorph.[24] A morphdb.hu szókincset és nyelvtant leíró adatbázisából a HunLex keretrendszerrel generált, alkalmazásfüggően optimalizált nyelvi erőforrásokat (tő- és toldaléktárak) összefoglaló néven HunTools-nak nevezzük. A HunTools elemei nyelvfüggetlen alkalmazások, amelyeket nyelvtechnológiát igénylő, különböző nehézségű problémák megoldására használhatunk. A kimenetként generált címkék feldolgozásának bonyolultsága szerint ezek az alábbiak:

- A legegyszerűbb a HunSpell helyesírás-ellenőrző, amely nem végez kimeneticímke-feldolgozást, csak visszaadja, hogy a vizsgált szó eleme-e a nyelvnek.
- A következő a HunStem szótövező alkalmazás, amely a kimeneti címkéknek csak a szótövét keresi meg a szótárban, és ezeket adja vissza.
- Végül a kimenet teljes feldolgozása a HunMorph morfológiai elemzőben történik meg, amelynél minden toldalékolási szabályra vonatkozó információ megjelenítésre kerül.

Az adatbázisban való keresés szempontjából a három alkalmazás eltérő módon viselkedik. Míg a HunSpell esetén az első találat után a keresés leállítható, és a HunStem esetén is csak korlátozott (eltérő szótövenkénti multiplicitással rendelkező) keresést kell végrehajtani, addig a HunMorph morfológiai elemzőnél a teljes állományon keresni kell, és az összes morfológiai elemzési alternatívát meg kell adni.

A HunStem standard kimenete egy adott szóra a szó legvalószínűbb szótöve, de megfelelő paraméterezéssel beállítható, hogy ne csak egy szótövet adjon vissza, hanem a homonimák egyértelműsítése érdekében — amennyiben erre szükség van — a szó szófaját is megadja.

Az elemzők kimenete az ún. *KR-kódolást*[25] használja. Ennek az adatstruktúrája egy speciális fagráf, amely mind inflexiós, mind derivációs információ megragadására alkalmas. A formalizmus alkalmazására nézzük az alábbi példákat:

```
kutya
  <+NOUN<-PLUR><-POSS><-ANP><-CAS>>
  <NOUN>
```

[24] Az elnevezések a verziószámtól függően ettől eltérhetnek. Az aktuális OCAML-ben írt változatnál a Hun- helyett Oca- előtag szerepel, pl. OcaStem.

[25] Az alkotók, Kornai és Rebrus, neve után.

kutyáink
<+NOUN<+PLUR<-FAM>><+POSS<+1><-2><+PLUR>><-ANP><-CAS>>
<NOUN<PLUR><POSS<1><PLUR>>>

kutyáéi
<+NOUN<-PLUR><-POSS><+ANP<+PLUR>><-CAS>>
<NOUN<ANP<PLUR>>>

kutyáikéit
<+NOUN<+PLUR<-FAM>><+POSS<-1><-2><+PLUR>><+ANP<+PLUR>>
<+CAS<+ACC>>><NOUN<PLUR><POSS<PLUR>>
<ANP<PLUR>><CAS<ACC>>>

2.3. A vektortérmodell dimenziójának csökkentése

A vektortérmodellt alkalmazó, gépi tanuláson alapuló szövegbányászati feladatok egyik fő problémája a vektortér magas dimenziószáma.[26] Vannak olyan bonyolult, nem skálázható algoritmusok, amelyek magas dimenziószám esetén nem alkalmazhatóak, de általánosan is igaz, hogy alacsonyabb dimenziónál az eljárások gyorsabban és hatékonyabban működnek. Ezért az előfeldolgozás során rendszerint egy dimenzióredukciós lépést is elvégeznek, aminek célja tehát az eredeti szótárméret, $|T|$, redukciója $|T_{\text{red}}| \ll |T|$-re.

A dimenzió csökkentésének nyelvtechnológiai megközelítésen alapuló módszereit — stopszószűrés, szótövezés — már vizsgáltuk, és itt feltételezzük, hogy ezeket a szűrőeljárásokat már végrehajtottuk a modellméret csökkentésének érdekében. Ebben a szakaszban a probléma matematikai eljárásokkal való megoldási lehetőségeit tekintjük át. Ezeket a módszereket elsősorban osztályozási és csoportosítási feladatoknál[27] használják a problématér méretének csökkentésére, ekkor ugyanis a redukcióból eredő esetleges információvesztés nem jelent problémát. Sőt, a dimenzióredukciós eljárások alkalmazása zajszűrésnek is felfogható. Általában csökkenti a felügyelt gépi tanulóalgoritmusoknál a túltanulás lehetőségét, ezért bár mérsékelt módon ($< 5\%$), de gyakran növeli a módszerek hatékonyságát [219].

A dimenzióredukciós technikáknak két típusát különböztetjük meg. *Jellemzőkiválasztás* (term selection) esetén a problématér azon jellemzőihez (szavaihoz) tartozó dimenziókat hagyjuk el, amelyek az adott feladat megoldásához legfel-

[26] A keresési feladatokban ez korántsem jelent ekkora problémát.

[27] Ebből adódóan a módszerek ismertetésénél esetenként hivatkozunk az 5. és a 6. fejezet alapvető fogalmaira.

jebb jelentéktelen módon járulnak hozzá. A stopszószűrés (ld. 2.2.4.4. alpont) például a jellemzőkiválasztásnak egy nyelvi-statisztikai alapokon nyugvó módszere. *Jellemzőkinyerő* (term extraction) módszereknek nevezzük az olyan eljárásokat, amelyek az eredeti jellemzők kombinációjaként új, kevesebb számú jellemzőt állítanak elő. Ezt az eljáráscsaládot akkor lehet alkalmazni, ha a feladat szempontjából a vektortér egyes dimenzióihoz tartozó reprezentánsok nem játszanak közvetlen szerepet, hiszen ezek már nem egyes szavakhoz, hanem azok kombinációihoz tartoznak.

Osztályozási feladatoknál való alkalmazás esetén a módszerek megkülönböztethetők még attól függően is, hogy *lokális*an, azaz kategóriánként alkalmazzuk őket — ekkor minden kategóriához külön szótár tartozik —, vagy *globális*an, amikor is az összes kategóriára vonatkozóan hajtjuk végre őket.

2.3.1. Jellemzőkiválasztó módszerek

Dokumentumgyakoriság szerinti szűrés. Egy egyszerű, de mégis hatékony redukciós módszer a dokumentumgyakoriságon alapuló szűrés, amely csak a legnagyobb *df* értékkel rendelkező szavakat tartja meg. Ekkor feltesszük, hogy az így eltávolított szavak információtartalma elenyésző, és a rendszer hatékonyságát nem befolyásolja. A vizsgálatok azt mutatták [219], hogy osztályozási feladatoknál e módszerrel akár a tizedére is csökkenthető a modellméret anélkül, hogy a hatékonyság rovására menne, sőt még a szótár 100-adára való csökkentése is csak csekély mértékben rontja a hatékonyságot. Ez az eredmény csak látszólag mond ellent annak a klasszikus „törvénynek”, miszerint az alacsony-közepes dokumentumgyakoriságú szavaknak a legnagyobb az információtartalma. Ez a szűrés ugyanis csak rendkívül nagy számú, nagyon alacsony *df* értékű szót távolít el a modellből. Ekkor tehát azokat a szavakat tartjuk meg, amelyre $df(t_k) > \theta$, ahol a θ alkalmas küszöbérték.

A módszernek egy alternatív változata, amikor a gyűjteménytámogatottság, *cf* alapján végzik a ritka szavak kiszűrését.

A következő három technikát osztályozási feladatoknál alkalmazzák, ahol a dokumentumok $c_1, \ldots, c_{|C|}$ kategóriákba tartoznak (ld. még a 104. oldalt). Ezek a függvények azt az intuitív megfontolást próbálják különböző módon interpretálni, hogy a legjobb jellemzők azok lesznek, amelyek legegyenetlenebbül oszlanak meg a kategóriák pozitív és negatív tanítóadatai között.

Információnyereség. Az első módszer a kategóriabecslésnél kapott *információnyereség*et (information gain) határozza meg bitben, a vizsgált t_k szó dokumen-

tumbeli előfordulásának, illetve hiányának függvényében:

$$\sum_{c\in c_j,\overline{c}_j}\sum_{t\in t_k,\overline{t}_k} P(t,c)\log\frac{P(t,c)}{P(t)\cdot P(c)} \tag{2.17}$$

Itt a $P(c_j)$ valószínűségeket a c_j-beli dokumentumok teljes gyűjteményben való előfordulásának arányával lehet becsülni, $P(t_k)$-t pedig a dokumentumgyakorisággal (df). Hasonlóan becslés adható a komplementer eseményekre, és a $P(t,c)$ együttes valószínűségek 4 esetére.

Dimenziócsökkentésnél a (2.17)-ban valamely küszöbérték alatti IG értékkel rendelkező szavakat töröljük a szótárból.

χ-négyzet statisztika. A *χ-négyzet* statisztika a szavak és kategóriák függetlenségének hiányát méri, más szóval azt, hogy mennyire jelentős az összefüggés a t_k szó és a c_j kategória közt:

$$\chi^2(t_k,c_j)=\frac{N-(n(t_k,c_j)n(\overline{t_k},\overline{c_j})-n(t_k,\overline{c_j})n(\overline{t_k},c_j))^2}{n(c_j)\cdot n(\overline{c_j})\cdot n(t_k)\cdot n(\overline{t_k})}. \tag{2.18}$$

Itt könyvünk szokásos jelölésmódjától eltérően a jobb olvashatóság érdekében nem alsó indexben, hanem az argumentumban jelöltük, hogy az egyes mennyiségek mire vonatkoznak. Eszerint $n(c_j)$ a c_j osztályba tartozó dokumentumok száma, $n(t_k)=n_i$ a t_k szót tartalmazó dokumentumok száma; $n(t_k,c_j)$ pedig azon c_j-beli dokumentumok száma, amelyek tartalmazzák a t_k szót. A felülvonással jelzett mennyiségek a c_j osztályba nem tartozó, illetve a t_k szót nem tartalmazó dokumentumok számára vonatkoznak.

Kölcsönös információ. A χ-négyzet statisztikához hasonlóan a kölcsönös információ is a kategória és a szó kölcsönös összefüggését határozza meg bitben mérve:

$$MI(t_i,c_j)=\log\frac{P(t_i,c_j)}{P(t_i)P(c_j)}. \tag{2.19}$$

Szűrésnél csak a magas χ^2-értékű szavakat tartjuk meg a modellben.

A fentieken kívül egyéb módszereket is tartalmaz a [169] 16. oldalán található összefoglaló táblázat.

Összehasonlítás. A (2.17)–(2.19) függvények lokálisan, egy adott kategóriára vonatkozóan határozzák meg egy szó fontosságát vagy függetlenségét. Egy szóra

vonatkozó globális értéket az egyes kategóriákra kapott értékek normál vagy súlyozott összegeként, illetve maximumaként szokták meghatározni:

$$f_{\text{összeg}}(t_k) = \sum_{j=1}^{|C|} f(t_k, c_j);\ f_{\text{s. összeg}}(t_k) = \sum_{j=1}^{|C|} P(c_j) f(t_k, c_j);\ f_{\max}(t_k) = \max_{j=1}^{|C|} f(t_k, c_j),$$

ahol $f(\cdot,\cdot)$ a (2.17)–(2.19) lokális függvények valamelyike.

A jellemzőkiválasztó függvények eltérő hatékonyságúak. Osztályozási feladatoknál a bemutatott dimenziócsökkentő eljárások közül az első három módszer bizonyult hatékonynak [219]. Megállapították, hogy az $IG_{\text{összeg}}$ és $\chi^2_{\max}$ használatával akár 100-adára is csökkenthető a vektortér dimenziója a hatékonyság jelentősebb visszaesése nélkül. Az irodalomban publikált tapasztalatok alapján összességében a $\{\chi^2_{\max}, IG_{\text{összeg}}\} > \chi^2_{\text{s. összeg}} \gg \{MI_{\text{s. összeg}}, MI_{\max}\}$ sorrend állítható fel a jellemzőkiválasztó eljárások között, ahol a $>$ jel jelentése: „hatékonyabb”.

2.3.2. Jellemzőkinyerő módszerek

A jellemzőkinyerő módszerek az eredeti T halmazból úgy kísérlik meg a leghatékonyabb megoldást biztosító redukált T_{red} előállítását, hogy *szintetikus* jellemzőket használnak, azaz olyanokat, amelyek az eredeti dokumentumokban nem fordulnak elő. Az olyan nyelvi jelenségek, mint a többértelműség, ill. szinonima miatt az eredeti szavak ugyanis nem feltétlenül ideálisak a dokumentumok tartalmi dimenzióinak megragadására. A jellemzőkinyerő módszerek két lépésben működnek: először meghatározzák a dokumentumokban szereplő szavak alapján az új jellemzőket, majd a dokumentumokat az új reprezentációnak megfelelő alakra hozzák.

2.3.2.1. Jellemzők csoportosítása

A *jellemzők csoportosításán* alapuló módszer a szavakat a köztük lévő szemantikus kapcsolat alapján csoportokba rendezi, és az egyes csoportok reprezentatív elemét választja ki az új dimenzió jellemzésére. A módszer hatékonysága erősen függ az alkalmazott csoportosító módszertől (ld. 6. fejezet). Könnyen előfordulhat, hogy az eredményül kapott szóklaszterekben a szavak közti kapcsolat esetleges, és így az eljárás inkább ront a hatékonyságon [115]. Későbbi vizsgálatok viszont azt mutatták [14, 118], hogy függetlenül az alkalmazott csoportosító eljárástól, a dimenziócsökkentésnek ez a módja jelentéktelennek mondható, legfeljebb 2% javulást hoz a hatékonyságot illetően.

2.3.2.2. Látens szemantikus indexelés

A *látens szemantikus indexelés* (latent semantic indexing, LSI) [23, 51] szinguláris értékfelbontáson (singular value decomposition, SVD) alapuló vektortranszformáció segítségével az eredeti dokumentumvektorokat kisebb dimenziójú vektorokká alakítja át. Az új dimenziók a dokumentumokban felfedezhető együttes szóelőfordulási mintázatok alapján az eredetiek kombinációjaként kaphatók meg. Az így kapott dimenziók közvetlenül nem interpretálhatók, de meglepően eredményesen jellemzik a dokumentumgyűjtemény szókészletének „látens" szemantikai szerkezetét. Ennek következtében olyan dokumentumok is a megfelelő kategóriába kerülhetnek, amelyek nem tartalmaznak egyet sem valamely kategóriára jellemző eredeti szavak közül, de olyanokat igen, amelyek a kategória szavaival gyakran fordulnak elő közösen [168].

Nézzük tehát részletesen az LSI működését! Tekintsük a dokumentumgyűjteményt leíró $\mathbf{D} \in \mathbb{R}^{M \times N}$ szó-dokumentum mátrixot (ld. (2.1)), ahol M a szavak, N pedig a dokumentumok száma. Legyen

$$\mathbf{D} = \mathbf{U}\Sigma\mathbf{V}^T$$

a $\mathbf{D}$ mátrix szinguláris értékfelbontása, ahol az $\mathbf{U} \in \mathbb{R}^{M \times R}$ és $\mathbf{V} \in \mathbb{R}^{N \times R}$ mátrixok oszlopai ortonormáltak, azaz egymásra ortogonálisok, és a $\Sigma \in \mathbb{R}^{R \times R}$ diagonális mátrix tartalmazza a sajátértékeket. Itt $R \leq \min(M,N)$ a $\mathbf{D}$ mátrix rangja. Rendezzük át a Σ és ezzel párhuzamosan az $\mathbf{U}$ és $\mathbf{V}$ mátrixokat, úgy hogy a sajátértékeket csökkenő sorrendben tartalmazza. Ekkor a Σ zérus vagy közel zérus sajátértékei a mátrix jobb alsó részmátrixába kerülnek, mely értékekhez tartozó sorok és oszlopok elhagyásával csökkenthető a $\mathbf{D}$ mátrix dimenziója. Ha nem csak nulla sajátértékeket hagyunk el, akkor az új $\mathbf{D}_{\text{red}}$ mátrix az eredeti mátrixnak R_{red} rangú közelítése lesz, ahol R_{red} a meghagyott sajátértékek száma:

$$\mathbf{D} \approx \mathbf{D}_{\text{red}}, \qquad \mathbf{D}_{\text{red}} = \mathbf{U}_{\text{red}}\Sigma_{\text{red}}\mathbf{V}_{\text{red}}^T. \tag{2.20}$$

A (2.20) képletben $\Sigma_{\text{red}} \in \mathbb{R}^{R_{\text{red}} \times R_{\text{red}}}$ a Σ-ból a kihagyott sajátértékekhez tartozó sorok és oszlopok törlésevel kapott mátrix, az $\mathbf{U}_{\text{red}} \in \mathbb{R}^{M \times R_{\text{red}}}$ és $\mathbf{V}_{\text{red}} \in \mathbb{R}^{N \times R_{\text{red}}}$ mátrixokat pedig szintén úgy kapjuk, hogy az eredetiekből töröljük a megfelelő oszlopokat, illetve sorokat.

A $\mathbf{D}_{\text{red}}$ mátrix tehát megtartja az eredeti mátrix legfontosabb szerkezeti elemeit, ugyanakkor a zajt, illetve a szóhasználat eltéréséből eredő különbségeket eliminálja. A szótár mérete ekkor $R_{\text{red}} \ll M$ lesz. Az új dimenziók az $\mathbf{U}_{\text{red}}\Sigma_{\text{red}}$ sorai lesznek, a dokumentumokat pedig a $\mathbf{V}_{\text{red}}\Sigma_{\text{red}}$ sorai reprezentálják a transzformált

térben. Következésképp két szó hasonlóságát, vagyis azt, hogy előfordulási mintázatuk mennyire azonos, az $\mathbf{U}_{\text{red}}\Sigma_{\text{red}}$ megfelelő sorainak, két dokumentum hasonlóságát pedig a $\mathbf{V}_{\text{red}}\Sigma_{\text{red}}$ sorainak hasonlósága alapján határozhatjuk meg. Egy új $\mathbf{d}$ dokumentum $\mathbf{d}'$ reprezentációját a transzformált vektortérben a

$$\mathbf{d}' = \mathbf{d}^T \mathbf{U}_{\text{red}} \Sigma_{\text{red}}^{-1}$$

képlet alapján határozzuk meg.

Ha a dokumentumgyűjtemény bővül, akkor a transzformált vektortérmodellt újra kell számolni. Ez igen időigényes lehet, és gyakori alkalmazása nagy gyűjtemények esetén megvalósíthatatlan. Ezért különböző megközelítéseket alkalmaznak az új dokumentumok és szókészletük modellbe történő integrálására (ld. pl. [23]).

Az LSI-t alkalmazó munkák mindegyike, pl. [118, 168, 169, 209], a módszernek a jellemzőkiválasztó technikákat meghaladó hatékonyságáról számol be.

2.3.2.3. A főkomponens-analízis módszere

Az LSI-hez hasonló a *főkomponens-analízis* (principal components analysis; PCA) módszere, amely báziscserét alkalmazva minimális információveszteség mellett csökkenti egy vektorhalmaz dimenziószámát. A módszert Pearson dolgozta ki az 1900-as évek elején. A dimenziószám csökkentésének haszna, hogy a kapott vektorhalmaz kisebb költséggel és gyorsabban dolgozható fel. Szövegbányászatban a vektortérmodell dimenziójának csökkentésére alkalmazzuk elsősorban osztályozási és csoportosítási feladatoknál, ekkor ugyanis a módszerből eredő esetleges információvesztés nem jelent problémát. Csoportosításnál (ld. 6. fejezet) például a dokumentumvektorok távolsága határozza meg a kialakult csoportokat. Emiatt az eredeti és a transzformált térben elvégzett csoportképzés akkor lesz összhangban egymással, ha a távolsági viszonyok nem változnak számottevően a bázistranszformáció során.

A PCA-módszer a dimenziószám csökkentését kisebb elemszámú bázisra való áttéréssel éri el. A báziscserét jelentő koordináta-transzformációnál megváltozik a pontok egymáshoz viszonyított helyzete, és ez információveszteséget okoz. Az átalakítás során ezért törekedni kell arra, hogy a pontok közötti távolságviszonyok a redukált térben a lehető legjobban közelítsék az eredeti térben meglévő viszonyokat. Intuitíve érezhető, hogy a távolsági viszonyok megőrzése és a csoportképzés szempontjából az a dimenzió a legfontosabb, ahol a pontok a legjobban szeparálódnak. A PCA-eljárásban ezért úgy választjuk ki az új bázisvektort, hogy a mintahalmaz minden irányban a lehető legjobban terüljön szét. Ekkor a

mintapontoknak az új **b** bázisvektorra vett koordinátaértékei is maximális szórást mutatnak.

Tekintsük ezúttal a szó-dokumentum mátrix $\mathbf{D}^T \in \mathbb{R}^{N\times M}$ transzponáltját, amelynek most a sorai tartalmazzák a $\mathbf{d}_i = \langle d_{i1}, d_{i2}, \ldots, d_{iM} \rangle$, $i \in [1,N]$ dokumentumvektorokat:[28]

$$\mathbf{D} = \begin{pmatrix} \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_N \end{pmatrix}.$$

Jelöljük $\overline{x}$-szel az **x** vektor értékeinek átlagát. Ha feltesszük, hogy a mintapontok kezdeti koordinátaértékeinek átlaga mindegyik k dimenzió mentén zérus, azaz

$$E\left(\overline{\mathbf{d}}\right) = 0, \quad \overline{\mathbf{d}} = \left\langle \overline{d}_1, \ldots, \overline{d}_M \right\rangle, \quad \text{ahol} \quad \overline{d}_k = \sum_{i=1}^{N} d_{ik}/N,$$

akkor a keresett új bázisvektorra teljesül, hogy az adatmátrixszal való szorzatának normája maximális:

$$\sum_{i=1}^{N} E\|\mathbf{b}^T \cdot \mathbf{d}_i\|_2 \rightarrow \max. \tag{2.21}$$

Az így meghatározott **b** bázisvektort használva a mintapontok csoportosítása sokkal hatékonyabban megoldható, mint egy olyan bázisvektor mentén, ahol a koordinátaértékek a (2.21) kifejezés értéke kisebb. Ez utóbbi esetben ugyanis a pontok koordinátaértékei közelebb esnek egymáshoz, így nehezebbőket külön csoportokba szeparálni.

A keresett bázisvektor a *kovarianciamátrix* elemzése alapján határozható meg [70]. A módszerrel előállítható a mintavektortérnek az az altere, amelyben a mintapontok elhelyezkedése a legjobban reprezentálja az alaptérben való eloszlást a négyzetes hibaértékre vonatkozólag. A módszer az alábbi lépésekből épül fel:

- A mintaadathalmaz elemeiből előállítjuk a $\mathbf{D} \in \mathbb{R}^{N\times M}$ mátrixot, ahol N a dokumentumok, M a jellemzők száma.
- Minden dimenzióra kiszámoljuk a koordinátaértékek átlagértékét, $\overline{d}_k$-t. A mintaadatmátrix minden eleméből kivonva a $\overline{\mathbf{d}}$ átlagvektort a

$$\overline{\mathbf{D}} = \mathbf{D} - \mathbf{1} \cdot \overline{\mathbf{d}}^T$$

[28] Az egyszerűség kedvéért a PCA ismertetésénél a dokumentum-szó mátrixot jelöljük a továbbiakban **D**-vel, azaz $\mathbf{D} \in \mathbb{R}^{N\times M}$.

kifejezést kapjuk, ahol az $\mathbf{1} \in \mathbb{R}^N$ vektor minden értéke 1. Itt $\overline{\mathbf{D}} \in \mathbb{R}^{N \times M}$.

- A $\overline{\mathbf{D}}$ mátrixnak elkészítjük a kovarianciamátrixát. Az $\mathbf{x}, \mathbf{y} \in \mathbb{R}^M$ vektorok kovarianciaértéke az alábbi módon számítható ki:

$$\operatorname{cov}(\mathbf{x}, \mathbf{y}) = \frac{\sum_{k=1}^{M} (x_k - \overline{x}) \cdot (y_k - \overline{y})}{M - 1}.$$

  A $\mathbf{C} \in \mathbb{R}^{M \times M}$ kovarianciamátrix elemeinek az értéke:

$$C_{ij} = \operatorname{cov}(\mathbf{d}_i, \mathbf{d}_j).$$

- Kiszámítjuk a $\mathbf{C}$ mátrix sajátvektorait és sajátértékeit. Egy négyzetes $\mathbf{A} \in \mathbb{R}^{M \times M}$ mátrixnak λ a sajátértéke és $\mathbf{v}$ a sajátvektora, ha

$$\mathbf{A} \cdot \mathbf{v} = \lambda \cdot \mathbf{v} \quad (\mathbf{v} \neq 0)$$

  teljesül. Ha egy $M \times M$-es mátrixnak van sajátvektora, akkor
  - M darab sajátvektora van, és
  - a különböző sajátvektorok ortogonálisak.

  A sajátértékek vektorát jelölje $\mathbf{s}$.
- Rendezzük az $\mathbf{s}$ elemeit csökkenő sorrendbe! Belátható, hogy a nagyobb sajátértékhez tartozó sajátvektor menténok nagyobb az elemek elhelyezkedési szórása [176]. Következésképpen az elemek elrendezése során a kisebb sajátértékű sajátvektorok elhagyásával veszíthetjük el a legkevesebb információt. Ha $L < M$ dimenzióra akarunk áttérni, akkor $\mathbf{s}$-ből kiválasztjuk az első L darab legnagyobb sajátértéket. A kiválasztott sajátértékekhez tartozó sajátvektorokból alkotott mátrixot jelölje $\mathbf{W} \in \mathbb{R}^{L \times M}$. Ez a mátrix tárolja az új, redukált tér bázisvektorait.
- A mintavektorokat transzformáljuk az új bázisra:

$$\mathbf{D}' = \overline{\mathbf{D}} \cdot \mathbf{W}^T.$$

A kapott $\mathbf{D}' \in \mathbb{R}^{N \times L}$ adatmátrix tartalmazza a vektorok új koordinátaértékeit.

A dokumentumhalmazt feldolgozó szövegbányászati algoritmusok bonyolultsága rendszerint a dimenziószám $O(M)$-es függvénye (ld. 3.2.3. pont). Mivel egy dokumentumhalmaz több tízezer vagy százezer kifejezést is tartalmazhat, igen fontos kérdés a dimenziószám csökkentése, ami ráadásul jelentősen nem rontja, sőt olykor javítja is az osztályozó és csoportosító algoritmusok hatékonyságát (zajszűrés). A [22] könyvben felsorolt példák is mind azt mutatják, hogy az előfeldolgozási lépés költsége később, a számítási költségnél megtérül.

A főkomponens-analízis alkalmazását bemutató példa megtalálható a könyv online mellékletében.

3. fejezet

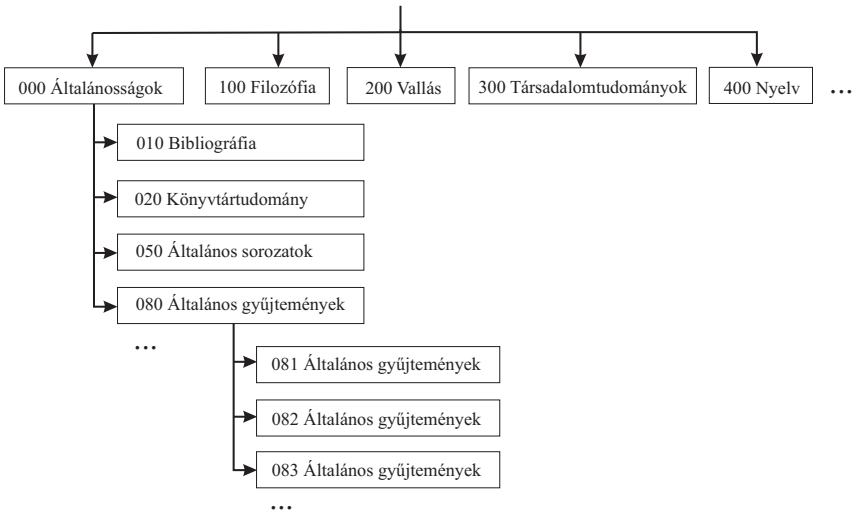
Az információ-visszakeresés alapjai

3.1. Az információ-visszakeresés modellje

Az *információ-visszakeresés*en olyan strukturálatlan, többnyire szöveges anyagok (dokumentumok) keresését jelentő tevékenységet értünk, amelynek célja a kereső információigényének kielégítése dokumentumok sokasága alapján. A *dokumentumgyűjtemény* egy számítógép háttértárolóján, lokális szervereken, vagy az interneten található általában. Míg néhány évtizede információ-visszakeresést csak specialisták végeztek — könyvtárosok, ügyvédek, tudományos kutatók stb. —, addigra manapság az internet széleskörű elterjedésével napi rendszerességgel tömegek foglalkoznak különböző formáival, pl. internetes keresőmotorokat használnak, e-maileket keresnek. Mára, megelőzve a hagyományos adatbázis alapú keresést, ez vált az információ-hozzáférés domináns formájává.

Az információ-visszakeresés története egészen az ókorig nyúlik vissza. Asszurbanipál asszír király i.e. 650 körül felismerte, hogy az információtárolás mellett az információhoz való hatékony hozzáféréshez valamilyen keresést támogató segédlet is szükséges. Ezért a mintegy 30 000 kőtáblát és 1200 szöveget tartalmazó könyvtárában bevezette az első könyvtári visszakereső rendszert. Később a görög világban is megjelent a szöveges információkeresés támogatása. A görög agyagtábla-könyvtárakban alapvető témák alapján sorolták az egyes köteteket kategóriákba a keresés megkönnyítése céljából. Alexandria híres könyvtárában Callimachus az i.e. 3. században vezetett be kategorizálás alapú besorolási rendet, az ún. kronológiai tárgykatalógust (Callimachus Pinakes). A könyvtári anyag katalogizálásának eredményeként 120 kötetben foglalták össze a teljes állományt. A következő nagy lépés a könyvtári keresőrendszerek fejlődésében csak jóval később, a XIX. század vége felé történt. Dewey 1876-ban vezette be a róla elnevezett tizedes osztályozási rendszert (Dewey Decimal Classification), amely 10-es tematikus felosztási rendszert alkalmazott. Ez a könyvtárakban ma is elterjedt ETO tárgyszórendszer előzményének tekinthető. Minden kategóriának 10 alkategóriája volt, a kategóriarendszer mélysége pedig a tématerület részletességétől függött

(ld. 3.1. ábra) [131]. A rendszer nagy előnye az egyszerű bővíthetőségében rejlett. Minden kötethez 3×5 hüvelykes katalóguscédulát készítettek, amelyen az alapvető információk voltak (cím, szerző, kiadó, téma), ezek alapján keresték vissza a könyveket.



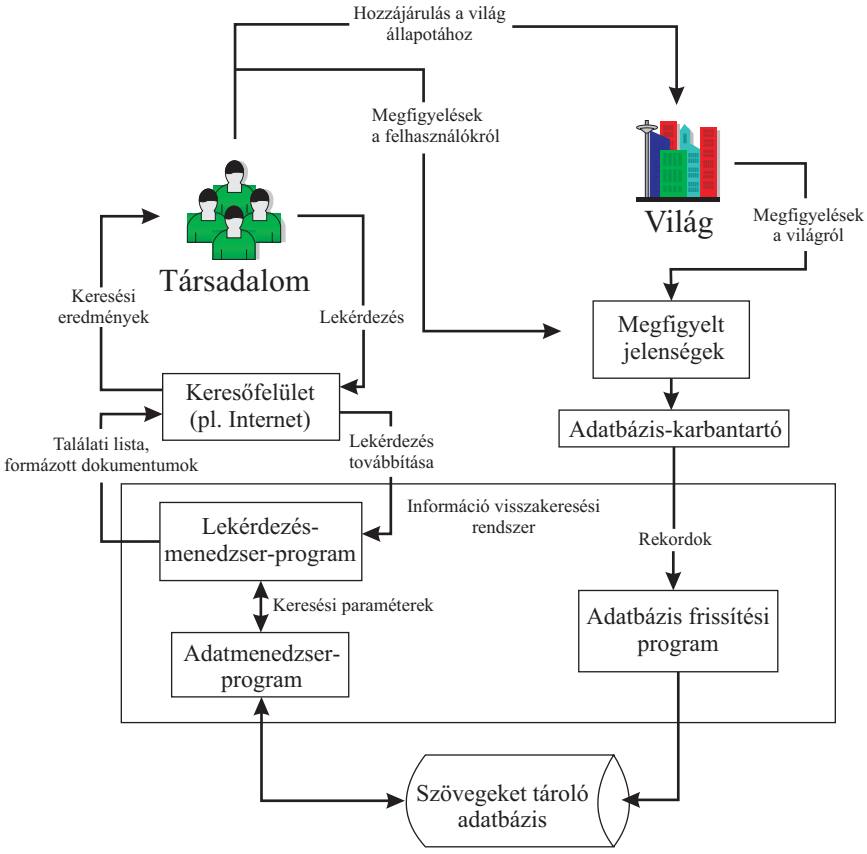
3.1. ábra. Dewey tizedes osztályozási rendszere

Az első teljesen automatizált szöveges visszakereső rendszer a KWIC (Key Word in Context) volt, amelyet az információ-visszakeresés atyjának is nevezett Hans Peter Luhn készített. Az információkat a számítógép lyukkártyákon kapta, és már stopszószűrést is alkalmazott. Az 1960-as években Gerard Salton fejlesztette ki a SMART [160] (Salton's Magical Automatic Retriever of Text) rendszert, amely sokáig a vonatkozó kutatások alapjaként szolgált, és számos későbbi rendszer használta fel az elsőként benne megvalósított módszereket.

Az információ-visszakeresés az internet rohamos elterjedésével vált kiemelten fontossá az 1990-es évek elejétől kezdődően. A felhasználó a keresést a keresőfelületen megadott keresőkifejezéssel[1] indíthatja el, eredményként egy találati listában megkapja a relevánsnak ítélt dokumentumokat többnyire a relevancia szerinti sorrendben megjelenítve. A korai keresőmotorok csak a keresőkifejezés egyezését vették figyelembe a relevancia kiszámításánál. A fejlettebb keresőmotorok már sokkal kifinomultabban működnek, képesek tanulni, és akár a felhasználó

[1] Állhat egy-két szóból, de akár egy teljes dokumentum is lehet.

érdeklődési körétől, valamint kereséseinek előzményétől függően testre szabott eredménylistát előállítani [207]. A szövegbányászati információ-visszakeresés modellje a 3.2. ábrán látható.



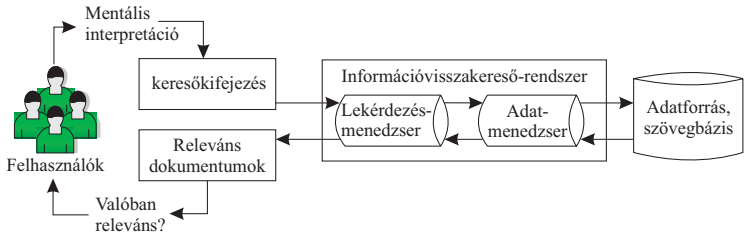
3.2. ábra. Az információ-visszakeresés modellje

Ahogy arra a fejezet elején megadott definícióban is rámutattunk, a keresés célja a felhasználó információigényének kielégítése. Az igény felismerése még nem jelenti azt, hogy az igényt a felhasználó pontosan, keresőkifejezés formájában meg is tudja fogalmazni — gyakran fordul elő az, hogy csak homályos elképzelése van a keresés elvárt eredményéről. Ha valaki például a *piacon elérhető legfejlettebb spektrális szövegbányászati rendszerről* szeretne információkat gyűjteni, akkor legfeljebb a dőlten szedett kifejezést tudja megadni a keresőprog-

ramnak, a szoftver nevét magát nem. A felhasználó számára a keresés folyamán körvonalazódik egyre jobban, hogy mit is keres valójában, azaz az információs-visszakeresés folyamata többnyire iteratív. A felhasználó a keresési eredmények alapján képes pontosabb keresőkifejezéseket megadni. A keresőkifejezések tartalmazhatnak a keresőszavakon kívül vezérlőoperátorokat is. A legtöbb keresőmotor elfogad logikai vezérlőoperátorokat (and, or stb.), sőt újabban már lehetősége van a hozzáértő felhasználónak a keresőmotor speciális opcióinak vezérlőoperátorként való használatával a leghatékonyabban kihasználni a keresőmotor képességeit (ld. 8.7. szakaszt).

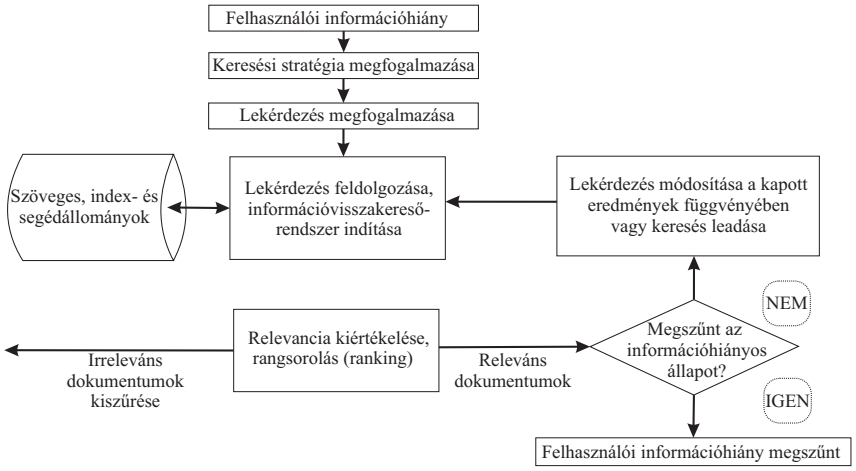
A következő lépésben az IR-rendszer feldolgozza a keresőkifejezést. Ez a legegyszerűbb esetben kizárólag a keresőszavak, illetve logikai operátorok értelmezését jelenti. Fejlettebb IR-rendszereknél ez kiegészülhet egyéb eszköztárakkal vagy tényezőkkel, például tezaurusz szótárakkal, ismert felhasználói preferenciák felhasználásával. A feldolgozás két lépésben történik: elsőként a lekérdezésmenedzser-modul létrehozza a belső reprezentációs alakot, majd ennek alapján az adatmenedzser-modul visszakeresi az adatbázisban a releváns adatokat, és elvégzi a találatok rangsorolását. Ez azt jelenti, hogy a keresőkifejezés helyes értelmezése csakis a lekérdezésmenedzser-modul működésén múlik, az adatmenedzser-modulnak csak végrehajtói szerepe van. Mivel az adatbázis általában rendkívül nagy méretű, ezért az IR-rendszer, és azon belül az adatmenedzser-modul tervezésénél elsődleges szempont a hatékony tárhelykihasználás és a gyors futásidő. A keresőrendszerek adatábrázolási módjának részleteire a 8. fejezetben térünk vissza.

A korai információvisszakereső-rendszerek (Dialog, MEDLINE) esetén a keresőkifejezést a felhasználónak pontos logikai formulával kellett megadni, mivel a lekérdezésmenedzser-modul erre vonatkozó kiértékelést nem végzett. Ez mostanra már megváltozott, a lekérdezésmenedzsertől egyre magasabb szintű értelme-



3.3. ábra. A lekérdezés iteratív folyamata

zést várunk el, amivel a hibázás lehetősége is nő. Minél nagyobb szabadsága van a keresőnek a keresőkifejezés megfogalmazásában, annál nagyobb a félreértelmezés esélye is. Fokozott jelentősége van a helyes szemantikai interpretációnak természetes nyelvi keresőkifejezések esetén — a megoldási lehetőségeket a 9. fejezet tárgyalja. Ezért gyakran csak keresések sorozatán keresztül jut csak el a felhasználó addig, hogy az információigényére megfelelő választ kapjon — azaz a keresési tevékenység a keresőkifejezés megadása, a találatok kiértékelése, a keresőkifejezés pontosítása lépések ismétlődéséből áll (ld. 3.3. ábra). A keresési tevékenység hatékonyságán sokat javít, ha a felhasználónak van némi elképzelése a keresőrendszer működéséről. Ekkor sokkal célratörőbben képes a találatok ismeretében a keresését finomítani, pontosítani. A keresési folyamat összefoglalása a 3.4. ábrán látható [141].



3.4. ábra. Dokumentum-visszakeresési folyamat

Az IR-rendszerek, azon belül is az internetes keresőmotorok működését és adatábrázolási technikáit a 8. fejezet tárgyalja.

3.2. Az információvisszakereső-rendszerek értékelési módszerei

3.2.1. Az egyes komponensek szerepe

Az IR-rendszerek értékelésének három alapvető szempontja az alábbi:

- Teljesítmény: a rendszer algoritmikus hatékonyságának vizsgálata a tárhelyigény és a futásidő szempontjából.

- Relevancia: a találatok relevanciája a keresőkifejezéshez.
- Ergonómia: a felhasználói felület és a keresőszolgáltatás felhasználóbarátságának vizsgálata.

Az információ-visszakeresési tevékenységnek három fő komponense van, amelyeket a fenti értékelési szempontok vizsgálata során figyelembe kell vennünk:

- a felhasználó,
- az adatforrás (adatbázis),
- és maga az információvisszakereső-rendszer.

Ha az IR-rendszer értékelését kiemelten, az emberi és adatforrás-tényezőktől függetlenül végezzük, torz eredményeket kaphatunk. Ezeket a faktorokat együttesen kell értékelnünk annak érdekében, hogy átfogó képet kapjunk a rendszer hatékonyságáról.

A keresés eredményességét nyilván jelentősen befolyásolja, hogy megfelelő bemenetet kapott-e a rendszer. A felhasználó gyakorlottságától erősen függ, hogy mennyire pontosan tudja megfogalmazni információigényét, azaz milyen hatékonyan képes azt az IR-rendszer logikáját is figyelembe véve nyelvileg interpretálni. Ugyancsak a kereső tapasztaltságától függ, hogy az elvárásainak nem eleget tevő találati lista alapján hogyan tudja módosítani, finomítani, adaptálni keresését. Ha ugyanis pontatlanul, nem elég lényegretörően adja meg a keresőkifejezést, akkor könnyen lehet, hogy bár az adatbázis tartalmaz releváns dokumentumokat, a találatok között ezek mégsem szerepelnek. Tegyük fel például, hogy a felhasználó Kolumbusz három hajójának a nevére kíváncsi, amelyekkel először az Újvilágba hajózott 1492-ben. Ha az *újvilági vitorlások* keresőkifejezést alkalmazza, akkor valószínűleg nem fogja megtalálni a találati listában a kívánt dokumentumokat. Evvel szemben az *1492 Kolumbusz* keresőkifejezés használata esetén már az első találatok között megtalálja a három hajó nevét is.

Az emberi tényezőhöz hasonlóan az adatforrás minősége is jelentős hatással van az IR-rendszer hatékonyságára. Nyilván, ha az adatbázis nem tartalmaz a keresőkifejezéshez releváns dokumentumokat, akkor bármilyen jól működjön is maga a visszakereső módszer, nem kapunk a keresésre releváns találatot. A felhasználó szemszögéből azonban úgy is tűnhet, hogy ez a keresőrendszer hibája. Internetes keresőrendszerek esetén tehát a szolgáltatás minősége kiemelten függ attól is, hogy mennyire hatékonyan gyűjti be a rendszer a keresést kiszolgáló adatbázisba a dokumentumokat. Az adatgyűjtés technikáit részletesen ismertetjük a 8.3.1. pontban.

3.2.2. A relevancia mérése

3.2.2.1. Rangsorfüggetlen mértékek

Az előbbiekben említett szempontok közvetett hatással vannak az IR-rendszer hatékonyságára. A rendszer legfőbb jellemzőit azonban az határozza meg, hogy milyen találatokat ad a keresőkifejezésre, és azokat hogyan rangsorolja.

A keresés hatékonyságánál két alapvető szempontot mérlegeljük. Az egyik az, hogy az összes releváns dokumentum közül mennyi szerepel a találatok között. Ezt a mértéket *felidézés*nek vagy *fedés*nek esetleg *teljesség*nek (recall) hívjuk. Nem biztos persze, hogy a találati lista minden eleme releváns. Minél kevesebb az irreleváns találat, azaz minél kevésbé zajos a keresés eredménye, annál jobban működik a rendszer. Ezt a mértéket *pontosság*nak vagy *megbízhatóság*nak (precision) nevezzük. E két mérték meghatározásához először vizsgáljuk meg, milyen lehetséges kimenetei lehetnek a keresésnek a találati lista és a relevancia szempontjából. Az eseményrendszert a 3.1. táblázat tartalmazza, ahol

- TP – a találati listában szereplő releváns találatok száma;[2]
- FP – a találati listában szereplő nem releváns találatok száma;
- FN – a találati listában nem szereplő releváns találatok száma;
- TN – a találati listában nem szereplő nem releváns találatok száma.

3.1. táblázat. Dokumentumok felosztása a találati lista és a relevancia szempontjából

	releváns	nem releváns
találati listában	TP	FP
nem találat	FN	TN

Ennek alapján a felidézést (R) és a pontosságot (P) rendre az

$$R = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} \quad \text{és} \quad P = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} \tag{3.1}$$

képletekkel definiálhatjuk. Hatékony rendszerek esetén mindkét érték magas. A két mennyiség egymásnak duálisa abban az értelemben, hogy adott algoritmus

[2] A mennyiségeket az angol rövidítések kezdőbetűivel jelöljük: true/false positive/negative.

mellett csak egymás rovására javíthatók. A felidézés értékét például maximalizálhatjuk úgy, ha az összes dokumentumot visszaadjuk, ekkor FN $= 0$, de a pontosság értéke nagyon alacsony lesz, hiszen a hamis találatok, FP száma rendkívül magas lesz. Ha az összes releváns közül csak nagyon kevés (de legalább 1) találatok adunk vissza, akkor — feltéve, hogy a találatok mind relevánsak — a pontosság lesz maximális, mivel FP $= 0$, és a felidézés lesz nagyon alacsony. A két mérték (parametrikus) harmonikus közepével az IR-rendszerek hatékonysága egyetlen kombinált mérőszámmal jellemezhető [198], amit F-mértéknek nevezünk:

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R}, \tag{3.2}$$

ahol az α, illetve a β paraméter értéke a két mérték súlyát határozza meg. Gyakorlatban általában a kiegyensúlyozott F-mértéket használják, amikor mindkét tényező egyenlő súllyal szerepel, ekkor $\beta = 1$, illetve $\alpha = 1/2$. Ezt rendszerint F_1-gyel jelölik.

3.1. PÉLDA. Tegyük fel, hogy adott egy tesztkorpusz, ahol néhány keresőkifejezéshez rendelkezésre állnak a releváns dokumentumok. Legyen a q keresés esetén a releváns dokumentumok halmaza

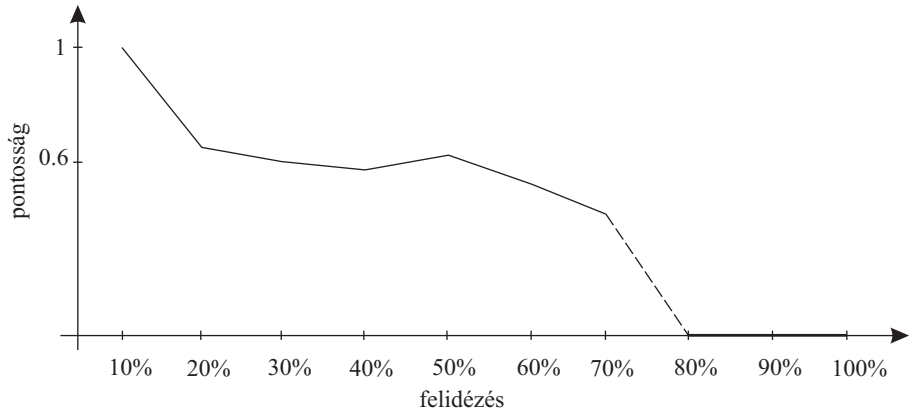
$$R_q = \{d_1, d_2, d_3, d_5, d_7, d_{11}, d_{13}, d_{17}, d_{19}, d_{23}\},$$

a keresőrendszer által megadott találati lista pedig

$$R_s = \{\mathbf{d_5},\ d_4,\ \mathbf{d_1},\ d_{38},\ \mathbf{d_{11}},\ d_{12},\ \mathbf{d_{23}},\ \mathbf{d_{19}},\ d_{18},\ d_{71},\ \mathbf{d_{17}},\ d_{32},\ d_{16},\ d_{48},\ d_{67},\ \mathbf{d_3}\}.$$
$$1.\quad 2.\quad 3.\quad 4.\quad 5.\quad 6.\quad 7.\quad 8.\quad 9.\quad 10.\quad 11.\quad 12.\quad 13.\quad 14.\quad 15.\quad 16.$$

(A releváns találatokat félkövér szedéssel jelöltük.) Ha a találati lista legelső elemét tartalmazó szeletet nézzük, erre $P = 1$ és $R = 1/10$, mivel a tíz releváns dokumentumból ebben csak egy szerepel. Mivel a következő releváns dokumentum a harmadik, ezért 20%-os felidézés mellett $P = 2/3$. Minden releváns találat esetén meghatározható a hozzá tartozó pontosság, ami egy pontosság-felidézés görbével jellemzi a keresőrendszer hatékonyságát a vizsgált q keresésre (ld. 3.5. ábra). 70% feletti felidézés esetén a pontosság 0 lesz, mivel nem kapjuk vissza az összes releváns dokumentumot. A teljes listát figyelembe véve tehát a $P = 7/16 = 0{,}4375$, $R = 0{,}7$ értékeket kapjuk.

A felidézés-pontosság görbét általában 11 (és nem 10) standard felidézési szintre határozzuk meg: 0%, 10%, 20%, ..., 100%. Az egyes felidézési szintek közötti, illetve a 0%-hoz tartozó pontosság értékét interpolációval kapjuk. A szinten-



3.5. ábra. Felidézés-pontosság görbe a 3.1. példához

kénti értékek átlagaként definiáljuk a *11 pontos átlagos pontosság*ot:

$$\sum_{i=0}^{10} P_i/11, \tag{3.3}$$

ahol P_i a megfelelő felidézési arányra vonatkozó pontosság.

A pontosságon és felidézésen kívül a 3.1. táblázatban szereplő mennyiségekkel még az alábbi mértékeket definiálhatjuk:

$$\begin{aligned} \textit{selejt}\ (\text{fallout}) &= \frac{\text{FN}}{\text{FN}+\text{TN}}, \\ \textit{szabatosság}\ (\text{accuracy}) &= \frac{\text{TP}+\text{TN}}{\text{TP}+\text{FP}+\text{TN}+\text{FN}}, \\ \textit{hiba}\ (\text{error}) &= \frac{\text{FP}+\text{FN}}{\text{TP}+\text{FP}+\text{TN}+\text{FN}}. \end{aligned} \tag{3.4}$$

Ezeket a mértékeket azonban ritkábban alkalmazzuk IR-rendszerek értékelésénél.

Nem minden esetben lehet teljes bizonyossággal eldönteni, hogy egy adott dokumentum releváns-e a keresésre. Ezt alátámasztja az is, hogy a tesztkorpuszok készítésére felkért szakértők sem értenek gyakran egyet a dokumentumok relevanciájának megítélésében. A relevancia bizonytalansága ekkor úgy modellezhető, hogy bináris helyett valós $[0,1]$-beli relevanciaértéket adunk meg. Ez esetben a pontosság és felidézés meghatározása az alábbi képletek szerint történhet:

$$P_f = \sum_{i=1}^{\mathrm{TP}+\mathrm{FP}} \frac{r_i}{(\mathrm{TP}+\mathrm{FP})\max_i r_i},$$
$$R_f = \sum_{i=1}^{\mathrm{TP}+\mathrm{FN}} \frac{r_i}{(\mathrm{TP}+\mathrm{FN})\max_i r_i},$$

ahol $r_i \in [0,1]$ a találati lista i-edik dokumentumának relevanciaértéke a q keresésre vonatkozóan. A két mennyiséget szokták *fuzzy felidézésnek* és *pontosságnak* is nevezni [131].

3.2.2.2. Rangsorfüggő mértékek

A felidézés, a pontosság, valamint az F_1-mértékek nem veszik figyelembe, hogy az adott dokumentum a találati listában hol szerepel, noha a gyakorlatban nagyon fontos, hogy a találati lista elején minél relevánsabb dokumentumok legyenek, mert a felhasználók szinte sosem nézik végig a teljes listát. Ennek mérésére leggyakrabban az *átlagos pontosság* mértéket használják a kísérletekben. Ezt az értéket minden releváns találat megfigyelése esetén mért egyedi pontossági értékek átlagaként számolják. A találati listában nem szereplő releváns dokumentumokra a pontosság értéke 0. Megegyezés szerint az értéket a találati lista első 1000 elemére határozzák meg. A mérték értékkészlete $[0,1]$. Minimumát akkor veszi fel, ha egy releváns találat sincs, maximumát akkor, ha a lista elején szerepel az összes releváns dokumentum. Egy keresőrendszer hatékonyságát a teszt-keresőkifejezésekre mért *átlagos pontossági értékek átlaga*ként definiálják (mean average precision, MAP).

3.2. PÉLDA. A 3.1. példa esetében az alábbi átlagos pontossági értéket kapjuk:

$$AP = \frac{1+2/3+3/5+4/7+5/8+6/11+7/16+0+0+0}{10} = 0{,}4446.$$

Míg az átlagos pontosság mértéke a teljes listát — illetve annak jelentős részét — figyelembe veszi, az alábbiakban ismertetett mértékek kizárólag a lista legelejére koncentrálnak. A *reciprok rang* (reciprocal rank, RR) értéke egy adott keresés esetén $1/r$, ahol r az első releváns dokumentum pozíciója a találati listában, azaz ha például rögtön az első találat releváns (ld. 3.1. példa), akkor értéke 1, ha csak a negyedik, akkor 0,25. A teljes rendszer hatékonyságát itt is több keresőkifejezésre meghatározott *reciprok rangok átlaga*ként kaphatjuk meg.

Közismert, hogy a legtöbb internetes keresés esetén csak az első találati oldalt nézik meg a felhasználók [174]. Ezt a tendenciát veszi figyelembe a *10-pontosság*

(precision at 10, P10), amely a találati lista első 10 eleme alapján számolja a pontosságot. A 3.1. példa esetén $P_{10} = 0{,}5$.

3.2.3. Egyéb hatékonysági mértékek

Egy IR-rendszer értékelésekor a 3.1. táblázatban szereplő mennyiségeken alapuló, a találatok relevanciáját vizsgáló mértékeken kívül a visszakeresési folyamat egyéb jellemzőire is kíváncsiak lehetünk. Számítástudományi módszerekkel a teljesítményre vonatkozó tényezőket tudjuk vizsgálni. A két legfontosabb mutató a *futásidő* és a *tárhelyigény*, amelyeket a bemenet függvényében vizsgálhatunk a számítási bonyolultságelmélet eszközeivel. Napjainkban a két faktor közül a futásidő a jelentősebb, mivel a visszakeresési idő sokkal inkább tekinthető szűk keresztmetszetnek. A háttértárolók kapacitásának növekedése és piaci áruk csökkenése a tárhelykapacitást másodlagos szemponttá tette a rendszerek tervezésekor.

A futásidő azt fejezi ki, hogy az algoritmus elvégzéséhez szükséges lépésszám milyen mértékben függ a bemenettől. IR-rendszerek esetén a bemeneten elsősorban az adatbázis méretét értjük, azaz annak meghatározása a feladat, hogy egy keresőkifejezéshez tartozó találatokat az adatbázis méretének függvényében hány lépésben határozza meg, és ebből következően mennyi idő alatt ad eredményt a rendszer. Ez elsősorban az adatmenedzser-modul megvalósításán múlik (ld. a 3.2. ábrát; bővebben a 8. fejezetben foglalkozunk a témával).

A futásidőt nagyságrendileg felülről becsüljük a bemenet méretének (n) függvényében. Az A algoritmus időbeli bonyolultságának nagyságrendjét $O(f(n))$-nek (ejtsd: nagy ordó $f(n)$) mondjuk, ha létezik olyan M konstans, hogy elég nagy n-re

$$|A| \leq M|f(n)|,$$

azaz a lépésszámra aszimptotikus felső becslést adunk konstansszorzótól eltekintve. Például

- $O(1)$, azaz konstans nagyságrendű az értékadás vagy valamely érték kiolvasása a memóriából;
- $O(\log_2(n))$, azaz logaritmikus nagyságrendű a bináris keresés tömbben;
- $O(n)$, azaz lineáris nagyságrendű egy lista végigolvasása, szélsőértékének, átlagának középső értékének (mediánjának) meghatározása;
- $O(n\log_2(n))$ a hatékony listarendezések nagyságrendje (pl. kupacrendezés).

Hasonló módon nagyságrendileg becsülhető egy algoritmus tárhelyigénye a bemenet méretének függvényében. Az IR-rendszerek tervezésénél elsődleges szempont, hogy minél gyorsabban eredményt adjunk, s mindezt a lehető legkevesebb

tárhely felhasználásával tegyük. Mivel a futásidő és a tárhelyigény optimalizálása általában dichotóm feladat, ezért jellemzően a futásidő csökkentése mellett döntenek. Az internetes keresők tárgyalásánál (ld. 8. fejezet) még visszatérünk erre a témára.

3.3. Mintaillesztés

Az információ-visszakeresés egyik alapművelete a *mintaillesztés* (pattern matching), amely adott minták szövegbeli előfordulásait határozza meg. A szövegbányászatban *mintá*n előre adott karaktersorozatot (sztringet) értünk. Mintaillesztésnél a sztringek (szószerkezetek, kifejezések, szavak, esetleg ennél kisebb egységek) mintához való hasonlóságát vizsgáljuk. A mintaillesztés előtt a szövegeket tokenizálni kell, azaz elemi egységekre kell bontani (ld. a 2.2.4.3. alpontot).

3.3.1. Hibatűrő mintaillesztés sztringmetrikákkal

Gyakran előfordul, hogy a feldolgozandó szöveg minősége nem tökéletes, azaz helyesírási hibákat, karaktertévesztéseket, elírásokat tartalmaz. Ennek egyaránt lehet oka a dokumentum alkotójának figyelmetlensége vagy ismerethiánya (idegen nyelven különösen), a szöveg automatikus optikai karakterfelismeréssel történt digitalizálásakor vétett hibák, illetve ezek kombinációja. Ilyen esetben a pontos mintaillesztésen alapuló módszerek nem találják meg a keresett információt. Ugyanezt tapasztaljuk, ha a felhasználó által adott információ, pl. a keresőkifejezés elírást tartalmaz. A felhasználónak ugyanakkor fontos lehet, hogy a keresett kifejezés hibás helyesírású előfordulásait is megtalálja, ezért a mintaillesztést végző alkalmazásoktól — szövegbányászati rendszerektől, keresőktől és különösen az információ-visszakereső eljárásoktól — elvárható, hogy valamilyen *hibatűrő* algoritmust használjanak a mintaillesztésnél. A hibatűrő mintaillesztést *fuzzy illeszkedés*nek (fuzzy matching) is nevezik.

Hibatűrő mintaillesztés esetén nemcsak a vizsgált mintakifejezéssel pontosan megegyező szövegrészletek jelentenek találatot, hanem valamilyen távolsági metrika szerint adott $\theta \geq 0$ küszöbértéknél nem nagyobb távolságra lévő sztringek is. Formálisan, legyen s_0 a minta. Ekkor minden olyan s találat lesz, ahol $d(s_0, s) \leq \theta$. Itt $d(\cdot,\cdot)$ két sztring közötti távolságot (illetve hasonlóságot) meghatározó *sztring hasonlósági metrika*, vagy röviden *sztringmetrika* (string linkage methods/metrics). $\theta = 0$ értékre kapjuk meg a pontos illeszkedéseket. A sztringmetrikák természetesen teljesítik a metrikus tereken definiált metrikák axiómáit, azaz

1. $d(s,t) \geq 0$, és az egyenlőség a.cs.a. áll fenn, ha $s = t$ (reflexív tulajdonság)
2. $d(s,t) = d(t,s)$ (szimmetrikus),
3. $d(s,t) + d(t,u) \geq d(s,u)$, azaz teljesíti a háromszög-egyenlőtlenséget.

Bár a minta akár egy teljes dokumentum is lehet, a sztringmetrikák alkalmazása dokumentumok összehasonlítására a gyakorlatban nem célszerű megoldás. A szrtingmetrikákat inkább kisebb szöveges egységek, szavak, kifejezések összehasonlítására használjuk. A sztringmetrikákat elsősorban relációs adatbázisok *adattisztítás*ára, illetve *adatbázisok migráció*jára dolgozták ki, vagyis az eltérő írásmóddal betáplált hasonló rekordok feltérképezésére. Mára azonban a szövegbányászat elemi eszköztárának része lett és széleskörben alkalmazzák őket.[3] A következőkben a legfontosabb sztringmetrikákat ismertetjük.[4] Az érdeklődő Olvasóknak ajánljuk még a távolságmetrikákat áttekintő [40] tanulmányt.

3.3.1.1. Hamming-távolság

A *Hamming-távolság* a legegyszerűbb sztringmetrika. Bináris változata megadja, hogy két bináris sztring hány bitben különbözik egymástól, azaz hány bitet kell kicserélni az egyik sztringben, hogy a másikat kapjuk. Például az $s = 10011010$ és a $t = 10001101$ bináris sztringek közötti Hamming-távolság $d_{\mathrm{H}}(s,t) = 4$. A Hamming-távolság könnyen értelmezhető azonos hosszúságú karakteres sztringekre is. Ekkor a távolságot a pozíciónként eltérő karakterek összegeként definiáljuk:

$$d_{\mathrm{H}}(s,t) = \sum_{i=1}^{|s|} \delta(s_i,t_i), \qquad \text{feltéve, hogy } |s| = |t|. \tag{3.5}$$

Itt s_i, ill. t_i rendre az s, ill. t sztring i-edik karaktere, $|s|$ az s sztring hossza (karaktereinek száma), és $\delta(i,j)$ a *Kronecker-szimbólum*:

$$\delta(i,j) = \begin{cases} 1, & \text{ha } i = j, \\ 0, & \text{egyébként.} \end{cases} \tag{3.6}$$

Különböző hosszúságú sztringeket a *módosított Hamming-távolság*gal hasonlíthatunk össze. Legyen $|t| > |s|$, ekkor

$$d_{\mathrm{mH}}(s,t) = d_{\mathrm{H}}(s,t_{|s|}) + (|t| - |s|), \tag{3.7}$$

[3] Lásd pl. a Google „Did you mean” szolgáltatását.

[4] Lásd még a `www.dcs.shef.ac.uk/~sam/stringmetrics.html` honlapot.

itt $t_{|s|}$ a t sztring első $|s|$ karakterét jelöli, az összeg második tagja pedig a két sztring hosszának különbsége.

3.3. PÉLDA. A *gyors*, ill. *gyros* szavak Hamming-távolsága $d_{\mathrm{H}}(gyors, gyros) = 2$. Az $s =$ *teremtés* és $t =$ *természet* szavak módosított Hamming-távolsága: $d_{\mathrm{mH}}(s,t) = 5$.

3.3.1.2. Levenshtein-távolság és általánosításai

Az egyik legkorábbi és legismertebb sztringmetrika a szerkesztési távolság alapú *Levenshtein-távolság*. A metrika megadja azon szerkesztési műveleteknek a minimális számát, amelyeket egyik sztring másikba alakításához végre kell hajtani. A szerkesztési lépések számát egy költségfüggvény határozza meg, ahol a négy lehetséges szerkesztési művelet az alábbi:

- karakter másolása egyik sztringből a másikba (0 költségű);
- karakter törlése (1 költségű);
- karakter beszúrása (1 költségű);
- karakter cseréje (1 költségű).

A szerkesztési műveletek segítségével a Levenshtein-távolság az alábbi módon definiálható:

$$d_{\mathrm{L}}(i,j) = \min \begin{cases} d_{\mathrm{L}}(i-1,j-1) + d(s_i,t_j) & \text{karaktermásolás vagy -csere,} \\ d_{\mathrm{L}}(i-1,j) + 1 & \text{karakter törlése,} \\ d_{\mathrm{L}}(i,j-1) + 1 & \text{karakter beszúrása,} \end{cases} \tag{3.8}$$

ahol $d(i,j) = (1 - \delta(i,j))$. Definíció szerint $d_{\mathrm{L}}(i,0) = i$ és $d_{\mathrm{L}}(0,j) = j$ minden $i \in [0,|s|]$ és $j \in [0,|t|]$ értékre. A Levenshtein-távolságnak több változata ismert, amelyek elsősorban a $d(i,j)$ függvényben térnek el egymástól.

3.4. PÉLDA. A 3.6. ábra a *gyors* és *gyros* szavak távolságának meghatározását mutatja. Az ábra segítségével a két szó összes részsztringjének távolsága is megadható. Az azonos hosszúságú részsztringekhez tartozó részmátrixok szimmetrikusak. A két teljes sztring Levenshtein-távolsága 2 (két csere), ezt az értéket a jobb alsó cellából olvashatjuk ki. A legkevesebb szerkesztési lépéssel járó, a két sztringet egymásba átvivő utat nyilak ábrázolják. Más alternatív szerkesztési utak is lehetnek ugyanekkora költséggel.

	G	Y	R	O	S
G	0	1	2	3	4
Y	1	0	1	2	3
O	2	1	1	1	2
R	3	2	1	2	3
S	4	3	2	3	2

3.6. ábra. Levenshtein-távolság levezetése egy konkrét példára

A *Needleman–Wunch-távolság*[5] a Levenshtein-távolság általánosítása olyan értelemben, hogy a törlés és a beszúrás műveletek költségei tetszőlegesen megadhatók:

$$d_{\mathrm{NW}}(i,j) = \min \begin{cases} d_{\mathrm{NW}}(i-1,j-1) + d(s_i,t_j) & \text{karaktermásolás vagy csere,} \\ d_{\mathrm{NW}}(i-1,j-k) + G(k) & \text{karakter törlése,} \\ d_{\mathrm{NW}}(i-k,j-1) + G(k) & \text{karakter beszúrása,} \end{cases} \tag{3.9}$$

ahol $G(k)$ az úgynevezett ugrás (gap) költségfüggvény, amely karaktertörlés, illetve -beszúrás esetében lép fel, és k az ugrás hossza. A Levenshtein-képletet $G(k) = k$ esetén kapjuk vissza.

3.3.1.3. Smith–Waterman-távolság

A *Smith–Waterman-távolságfüggvény*t eredetileg DNS és proteinláncok leghosszabb egyező molekuláris szubszekvenciáinak meghatározására fejleszették ki. A távolságfüggvényt a következő képlettel definiáljuk:

$$d_{\mathrm{SW}}(i,j) = \max \begin{cases} 0 & \text{újrakezdés,} \\ d_{\mathrm{SW}}(i-1,j-1) + d_{\mathrm{SW}}(s_i,t_j) & \text{karaktermásolás vagy -csere,} \\ \max_k(d_{\mathrm{SW}}(i-k,j) - G_k) & \text{karakter beszúrása,} \\ \max_\ell(d_{\mathrm{SW}}(i,j-\ell) - G_\ell) & \text{karakter törlése.} \end{cases} \tag{3.10}$$

A végső távolság a $d_{\mathrm{SW}}(i,j)$ táblázat maximális cellaértéke. A d_{SW} függvény két sztringnek a hasonlóságát méri, azaz értéke annál nagyobb, minél hasonlóbbak az

[5] Ez a távolságfüggvény még az alábbi neveken is ismert: Needleman–Wunch–Sellers és (javított) Sellers-algoritmus.

összehasonlított karakterláncok, pontosabban minél nagyobb a leghosszabb egyező részsztringjük.

A (3.10) formulában a d költségfüggvény, amely az alkalmazástól függően változhat. A paraméterek lehetséges értékei:

- $G = 1$: az ugrásköltség (törléskor és beszúráskor);
- $d(c,c) = -2$: alkalmazásfüggő helyettesítési költség;
- $d(c,d) = +1$: alkalmazásfüggő helyettesítési költség.

3.5. PÉLDA. A fenti paraméterértékek mellett a *GGGG PQRS AAAA* és a *TTTT PQRS CCCC* karaktersorozatok távolságmérését a 3.7. ábrán látható mátrix szemlélteti. A két sztring közötti távolság 6, hiszen ez a $D(i,j)$ mátrix maximális eleme. Ez megegyezik a leghosszabb közös részsztring, *_PQRS_* hosszával [40].

	G	G	G	G		P	Q	R	S		A	A	A	A
T	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	1	0.5	0	0	0	1	0.5	0	0	0
P	0	0	0	0	0.5	2	1.5	1	0.5	0.5	0	0	0	0
Q	0	0	0	0	0	1.5	3	2.5	2	1.5	1	0.5	0	0
R	0	0	0	0	0	1	2.5	4	3.5	3	2.5	2	1.5	1
S	0	0	0	0	0	0.5	2	3.5	5	4.5	4	3.5	3	2.5
	0	0	0	0	1	0.5	1.5	3	4.5	**6**	5.5	5	4.5	4
C	0	0	0	0	0.5	0	1	2.5	4	5.5	5	4.5	4	3.5
C	0	0	0	0	0	0	0.5	2	3.5	5	4.5	4	3.5	3
C	0	0	0	0	0	0	0	1.5	3	4.5	4	3.5	3	2.5
C	0	0	0	0	0	0	0	1	2.5	4	3.5	3	2.5	2

3.7. ábra. Smith–Waterman-távolság kiszámolása egy konkrét példára

3.3.1.4. Manhattan-távolság

A *Manhattan-távolság*[6] egy vektortérbeli távolságfüggvény, amely két sztring távolságát a vektortérbeli reprezentációjuk közötti rácshálós élek számának össze-

[6] további elnevezései: blokk-távolság, L_1-távolság

geként határozza meg. A távolságfüggvény képlete:

$$L_1(\mathbf{s},\mathbf{t}) = \sum_{i=1}^{N} |s_i - t_i| = \|\mathbf{s}-\mathbf{t}\|_1, \text{ahol } \mathbf{s},\mathbf{t} \in \mathbb{R}^N.$$

Ez megegyezik a sztringek különbségének 1-es normájával (ld. (2.5) képlet). A távolságfüggvény Manhattan utcáiról kapta nevét, ahol két dimenzióban két utcasarok közötti távolság megegyezik a köztük lévő utcaszakaszok hosszának összegével.

3.3.2. Mintaillesztés reguláris kifejezésekkel

A *reguláris kifejezés*ek[7] segítségével szabály alapú mintaillesztést végezhetünk. A reguláris kifejezés szintaktikai szabályok szerint felépülő, speciális értékkészlettel rendelkező sztringhalmazt reprezentáló sztring,[8] amely kézenfekvő és kompakt lehetőséget nyújt a szabályokkal leírható, nyílt — azaz nem felsorolható — tokenosztályok felismerésére. Reguláris kifejezések használata igen elterjedt, közvetlenül alkalmazza pl. a UNIX operációs rendszer grep parancsa, és számos programozási nyelv (pl. Perl) támogatja.

A reguláris kifejezés egy tömör formában megadott minta, amely akár végtelen számú sztringet is képes kódolni. Ez annak köszönhető, hogy a reguláris szintaxis olyan elemeket is tartalmaz, amelyek ciklikus mintaillesztést is lehetővé tesznek. Vegyük sorra, milyen szintaktikai elemekből épülnek fel a reguláris kifejezések!

1. Alternatíva:
   - |: A reguláris kifejezésekben az alternatívák elválasztására szolgál. Például a de|du egy szövegben egyaránt felismeri a *de* és a *du* rövidítéseket.
2. Számosság: a mintában lévő karakterek számosságának megadására az alábbi lehetőségek vannak. Az operátorok mindig az őket megelőző karakterre vonatkoznak.
   - ?: Azok a sztringek illeszkednek rá, ahol a karakter 0 vagy 1 alkalommal fordul elő. Például a gyerm?ek-re egyaránt illeszkedik a *gyermek* és a *gyerek* szó.

[7] Az angol terminológia a regular expression kifejezésből adódóan a *regex* és *regexp* rövidítéseket is használja.

[8] A reguláris jelző arra utal, hogy a reguláris kifejezések ekvivalensek Chomsky-nyelvosztályok közül a reguláris (3-as típusú) nyelvekkel [13].

- *: Azok a sztringek illeszkednek rá, ahol a karakter tetszőleges alkalommal (0-t beleértve) fordul elő. Például a gó*l-ra illeszkednek a *gl*, *gól*, *góól*, *góóól* stb. szavak.
- +: Azok a sztringek illeszkednek rá, ahol a karakter legalább egyszer előfordul. Például a gó+l-ra illeszkednek a *gól*, *góól*, *góóól* stb. szavak, de a *gl* sztring nem.
- .: A megadott pozíción tetszőleges karakter szerepelhet pontosan egyszer. Például a l.p reguláris kifejezés felismeri a *lép*, *láp*, *lop*, *lep*, *ltp* sztringeket.

3. Csoportosítás:
   - ( ): Az operátorok tartományát gömbölyű zárójelek segítségével lehet módosítani. Például a re(ce)+ reguláris kifejezés felismeri a *rece*, *rececе*, *rececece* stb. szavakat is.
   - [ ]: A szögletes zárójellel az alternatíva operátort válthatjuk ki: a köztük megadott karakterekből pontosan egynek kell szerepelnie az illeszkedő sztringekben. Például a r[éőú]t-ra a *rét*, *rőt*, *rút* sztringek illeszkednek.
   - -: A kötőjel operátorral intervallumokat adhatunk meg, így nem szükséges a két szélsőérték közti karaktereket felsorolni. Például a kisbetűket a [a-z] reguláris kifejezéssel adhatjuk meg.
   - { }: A kapcsos zárójelek sztring hosszának megadásra szolgálnak. Egyetlen értéket vagy intervallumot is megadhatunk. Például a [a-z]{2-5}-re az összes legalább kettő és maximum 5 karakter hosszúságú, kisbetűből álló sztring illeszkedik.
   - \: A backslash karakter az eddig ismertetett vezérlőkarakterek karakterként való használatát teszi lehetővé. Például a \\begin\{[a-z]+\} kifejezés *begin{x}*-et definiálja, ahol *x* tetszőleges kisbetűkből álló nem üres sztring.

3.6. PÉLDA. Alkalmazzunk reguláris kifejezéseket e-mail címek kinyerésére. A nagybetűs e-mail címeket az alábbi reguláris kifejezéssel adhatjuk meg:

[A-Z0-9._-]+@[A-Z0-9.-]+\.[A-Z]{2,4}.

Itt az első szögletes zárójelben a @ előtti részt adjuk meg. Utána előírjuk, hogy kötelezően szerepelnie kell a @ jelnek. Ezt ismét a megadott karakterek követik, végül a sztringet egy pontot követő legalább 2, de legfeljebb 4 hosszúsága csak alfabetikus karakterekből álló országdomén zárja.[9]

[9] Ez a példa csak a reguláris kifejezések szemléltetésére szolgál, hiszen olyan sztringeket is felismer, amelyek valójában nem érvényes e-mail címek: –@ASDF.JPG.

4. fejezet

Információkinyerés

4.1. Bevezető

Információkinyerés alatt (information extraction, IE) a szövegbányászati feladatok egy speciális esetét értjük, ahol a cél nem elsősorban a felhasználó számára releváns információ lokalizálása (megfelelő dokumentum kiválasztása egy nagyobb kollekcióból), hanem az adott feladat szempontjából fontos szövegrészek (információk, tények) kigyűjtése a dokumentumokból, azaz strukturálatlan szövegekből *strukturált információ* előállítása.

A brit CGNU 102 millió euróért vásárolja meg az ABN AMRO magyar életbiztosítási leányvállalatát, a MÉBIT-et.
A Transyl-West Vagyonkezelő Kft. két magánszemélynek értékesítette a Biodiszkont Rt.-ben lévő aranyrészvényeit.

Vevő	Áru	Érték	Eladó
a brit CGNU	MÉBIT	102 millió euró	
két magánszemély	Biodiszkont Rt.-ben lévő aranyrészvények		a Transyl-West Vagyonkezelő Kft.

Ilyen munkafolyamatok az élet sok területén előfordulnak, általában statisztikák, összesítések, elemzések készítéséhez szoktak a rendelkezésre álló dokumentumokból adatokat gyűjteni. Tipikus, az IE-technikák által támogatható munkafolyamatok például:

- orvosi vizsgálati dokumentumokból összesítések készítése (pl. gyógyszerkutatás támogatására);
- sajtófigyelő szolgáltatások esetén a sajtóban megjelenő hírek gyűjtése (pl. egy adott vállalat ügyleteivel kapcsolatban, vagy egy kijelölt pozíciót tekintve a személycserék összegyűjtése újsághírekből);
- egy információs rendszer naplófájljaiból leállások, hibás működések, illetve azok körülményeinek (paraméterek) kigyűjtése.

Az információkinyerés a szövegbányászat egyik legintenzívebben kutatott területe, mivel egyrészt nagy élőmunkaigényű feladatok (részlegesen vagy teljesen) automatizált megoldásával foglalkozik általában, másrészt a létező megoldások legtöbbször probléma- és környezetspecifikusak, azaz nehezen alkalmazhatók új problémákra.

A terület intenzívebb kutatása a 90-es évektől kezdődött, eleinte angol nyelvű tartalmak feldolgozásával. Később egyértelművé vált, hogy a vizsgált IE-problémák különösen fontosak más, akár ritkán beszélt nyelveken is, hiszen az információkinyerés egyik kézenfekvő alkalmazása a nyelvi korlátok áthidalása — a felhasználó könnyedén értelmezni tud idegen (általa nem beszélt) nyelven íródott dokumentumhalmazból kinyert információt is, hiszen az IE-rendszerek kimenetei fix formátumúak és egyértelműek (így fordításuk is viszonylag egyszerűen megoldható). Ennek megfelelően az utóbbi években a kutatás fókusza újabb nyelvek, illetve a *nyelvközi* (cross-language) *kinyerés* irányába tolódott el.

A terület legismertebb szakmai fórumát a 90-es években a Dokumentum Megértési Konferenciák (MUC), majd napjainkra az Automatikus Tartalomkinyerés Program (ACE), és a Szövegkinyerési Konferenciák (TREC) képezik. E fórumok honlapján (ld. a 13. oldalt) számos konkrét rendszert bemutató tudományos anyag, illetve sokféle, részben az információkinyerés területéhez köthető feladathoz tartozó referencia-adatbázis elérhető.

4.1.1. Példák alkalmazott IE-re

McCallum 2005-ös cikke [128] számos valós életből vett példát említ, ahol egy-egy információkinyerő alkalmazás piaci sikert hozott az azt bevezető vállalatnak. Az alábbiakban bemutatjuk ezeket az eseteket, hiszen egy probléma fontosságát a gyakorlati alkalmazás igénye mutatja a legjobban:

Továbbtanulási lehetőségeket ismertető weboldal. 2001-ben az Egyesült Államok Foglalkoztatásügyi Minisztériuma megbízást kapott egy olyan honlap létrehozására, amely segíti az embereket a különböző magán- és állami iskolák (főiskolák, egyetemek), illetve egyéb szervezetek által kínált továbbtanulási lehetőségekről való informálódásban. Az újonnan létrehozandó szolgáltatással szemben természetes követelményként fogalmazódott meg, hogy támogasson különböző keresési lehetőségeket, például helyek, időpontok, témakörök, előfeltételek, a kurzusok előadói stb. szerint. A legnagyobb probléma az oldal tartalmának feltöltésekor az volt, hogy noha gyakorlatilag minden releváns információ rendelkezésre állt (például az oktatási tevékenységet végző szervezetek honlapjain), de ezek töredéke volt csak hozzáférhető

többé-kevésbé strukturált formában. Bonyolította a helyzetet, hogy a részben szabályos formalizmust követő adatok is rendre eltérően formázva jelentek meg a különböző weboldalakon, hiszen azok tartalmát emberi feldolgozásra tervezték, és szinte semmi sem volt elérhető közvetlenül az adatbázisba feltölthető formában.

Természetesen lehetőség lett volna a tudásbázis emberi erővel történő feltöltésére, azonban ez a megoldás a hosszú távon nem tartható velejárók miatt (folyamatos frissítés szükségessége a régi forrásokból, illetve új szervezetek megjelenése stb.) túlzottan költséges lett volna. A weboldal létrehozására és menedzselésére sikerrel állítottak üzembe egy információkinyerő rendszert, mely az alábbi feladatok (automatizált) megoldásával előállította a naprakész, kereshető webes felületet:

- webrobot, mely a releváns információt (kurzusok adatai) tartalmazó dokumentumokat kereste, illetve szűrte
- a releváns lapokról az egy kurzushoz tartozó információk összegyűjtése (ezek jöhettek pl. táblázatokból, szöveges leírás esetén egy, de akár több szövegrészből is)
- az adatbázis mezőinek keresése és bejelölése a dokumentumban (mindezt nagy pontossággal, hogy a kurzus címe, előadó neve stb. teljes egészében kerüljön az adatbázisba, de ne tartalmazzon felesleges szövegelemet)
- redundancia szűrése (ugyanannak a kurzusnak az adatai néha több helyről, vagy akár ugyanarról a helyről is több példányban előállhattak, a duplikátumokat azonosítani és szűrni kellett)
- az adatbázis fölé — amely az előző lépések segítségével automatikusan került feltöltésre — a felhasználók számára jól kezelhető webes felület generálása.

A feladatot olyan pontossággal sikerült megoldani, hogy az adatbázis manuális ellenőrzésére sem volt szükség (az adatok közvetlenül bekerülhettek a rendszerbe).

FlipDog online álláskereső A 2000-ben induló `www.flipdog.com` álláskereső portál nagy sikert ért el, az oldalán keresztül kereshető nagy mennyiségű munkalehetőségnek köszönhetően (kb. kétszer annyi állást hirdettek mint riválisaik). A szolgáltató ötlete azon alapult, hogy — ahelyett, hogy a munkáltatóktól várta volna el az álláslehetőségek feltöltését a rendszerbe — automatikusan állította elő az álláshirdetéseket, közvetlenül feldolgozva több mint 60 000 cég weboldalát.

Zoominfo A `www.zoominfo.com` oldal emberekről gyűjt adatokat (cím, elérhetőség, végzettség, korábbi iskolák, munkahelyek stb.) weben elérhető tartalmak elemzésével. Adatbázisukban tízmilliós nagyságrendben találhatók adatok.

CiteSeer A `www.citeseer.org` automatikusan nyer ki hivatkozási és egyéb adatokat (szerzők, publikáció éve, helye stb.) tudományos cikkekből. Ezeket a kinyert adatok mentén kereshető formában teszi elérhetővé, ennek segítségével gyorsan megkapható egy adott cikkre hivatkozó összes cikk listája, illetve a hasonló témán dolgozó kutatók listája stb.

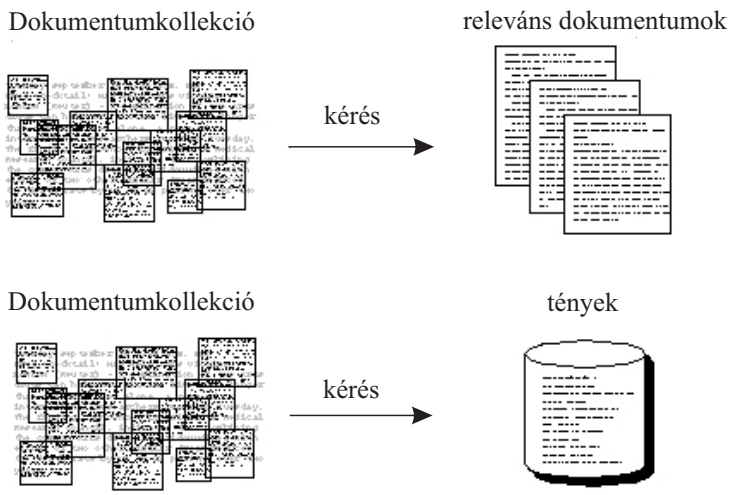
Elektronikus levelezések feldolgozása Az USA-beli Carnegie Mellon Egyetemen olyan rendszert fejlesztettek ki, amely elektronikus levelezésekből automatikusan címlistákat állít elő (levelezési címmel, foglalkozással stb.). Létezik olyan alkalmazás is, amely elektronikus naptárat tölt fel határidőkkel, időpontokkal beérkező e-mailek alapján.

4.1.2. Az információkinyerés és -visszakeresés összehasonlítása

Információkinyerés alatt mindig elektronikusan feldolgozható, strukturálatlan szöveges dokumentumokból a feldolgozás szempontjainak megfelelő, rendszerezett tudás előállítását értjük. A kinyert információ (tipikusan névelemek, relációk, események) eltárolható valamilyen relációs adatbázisban (adattáblákban), ezáltal hozzáférhetővé, gyorsan feldolgozhatóvá válik (nem csak az ember, de a gép számára is), pl. lekérdezések által. Az IE-rendszer működése az adatbázis aktualizálását, feltöltését jelenti. Az információkinyerés és -visszakeresés közti fő különbséget a 4.1. ábra szemlélteti.

IE:

- A felhasználó a (releváns dokumentumokból) kivont tényeket elemzi/használja.
- Nehezebb feladat, nagyobb szakértői tudást igényel.
- Az IE-rendszerek általában doménfüggők.
- Legtöbbször csak rögzített típusú elemeket képesek kigyűjteni (a kérések jellege előre megadott).
- Lassabb, sokszor pontatlanabb a végeredmény.
- Jóval hatékonyabb, hiszen a feldolgozásban egy lépéssel túlmutat az IR-en: a felhasználónak kevesebb ideig tart feldolgoznia a kimenetet.



4.1. ábra. Az IE- és IR-rendszerek összehasonlítása. *Forrás: GATE honlapja*, [45]

IR:

- A felhasználó a releváns dokumentumokat kapja vissza, a feldolgozásukat maga végzi el.
- Könnyebb feladat, kevesebb háttérismeret kell az elvégzéséhez.
- Általános eljárások léteznek a megoldására.
- Tetszőleges lekérdezést tud kezelni.
- Gyors, kevésbé pontatlan (hiszen a felhasználó elemzi a visszaadott dokumentumokat).
- Kevésbé hatékony: a felhasználónak több időbe telik a kimenetet feldolgozni.

4.2. Az információkinyerés tipikus részfeladatai

A fentiekben bemutattuk az IE-feladatokat általánosságban. Ebben a szakaszban röviden, példákon keresztül ismertetjük az információkinyerés legfontosabb részproblémáit. Minden feladatra igyekszünk több eltérő alkalmazás esetén is példát adni, ezzel is hangsúlyozva a feladatok probléma- és korpuszspecifikusságát. Az IE-n belül tehát leggyakrabban az alábbi (egymásra épülő) feladatokat különböztetik meg:

Névelemek felismerése Névelem alatt az adott szövegben előforduló tulajdonneveket, illetve entitások, mértékegységek stb. neveit értjük. Névelemek felismerése alatt

- *újsághírek* esetén általában személyek, szervezetek, termékek, helyek neveit *(Milyen cégek szerepelnek a korpuszban?)*;
- *orvosi dokumentumok* esetén személyek, betegségek, tünetek, gyógyszerek stb. neveit;
- *biológiai szövegek* esetén pl. gének, proteinek, fajok, sejtek neveit értjük.

Kereszthivatkozások azonosítása A kereszthivatkozások azonosítása tipikusan közbenső feldolgozási lépés egy IE-rendszerben, célja az adott dokumentumban ugyanazon egyedre utaló névelemek egymáshoz rendelése. Idetartozik a névmásoknak és utalószavaknak a megfelelő névelemhez történő hozzárendelése, az *anaforafeloldás* is.

Szereplők azonosítása Szereplők azonosítása alatt bizonyos előre definiált eseménysablonok előírásainak megfelelő névelemek megkeresését értjük. Tipikus szereplők lehetnek

- *újsághírek* esetén például vásárlók, áruk, eseményhelyszínek *(Kik vásároltak?)*;
- *orvosi szövegek* esetén (adott betegség) tünetei, gyógyszerei stb.;
- *biológiai szövegek* esetén pl. (valamilyen folyamatra nézve) gátló/serkentő hatású gének, körülmények, anyagok.

Szereplők közti relációk azonosítása Ez a feladat az előre adott eseménysablonok bizonyos szereplői között értelmezhető kapcsolatok felismerését foglalja magába. Ilyen relációk lehetnek

- *újsághírek* esetén például vásárlók–vásárolt áru, áru–ár stb. kapcsolatok *(Kik vásároltak fel olajipari vállalatot?)*;
- *orvosi szövegek* esetén (adott betegséghez) veszélyes mellékhatással járó gyógyszerek, kockázati tényezőt jelentő szokások (dohányzás, ...);
- *biológiai szövegek* esetén pl. egy adott gén expresszióját gátló/serkentő fehérjék, körülmények, anyagok.

Eseménykeretek illesztése Egy elméleti IE-rendszer kimenete az előre adott típusú események előfordulásai a dokumentumokban, az azokban részt vevő szereplőkkel, és a köztük lévő összes (igazolható) relációs viszonnyal. Ez az összes eddig felsorolt részfeladat teljes körű megoldását igényli, és olyan lekérdezéseket tesz lehetővé, mint pl: *Kiktől vásárolt 300 tonnánál nagyobb mennyiségű nyersanyagot a MOL 2006-ban?*.

4.3. Szabály alapú és statisztikai megközelítések az IE-ben

A legegyszerűbb megközelítés előre definiált típusú entitások, relációk, események szövegből való kinyerésére, hogy nyelvi mintákat definiálunk, amelyek lefedik a feltételeknek megfelelő szövegelemek egy részét. Egy lehetséges minta:

[Nagybetűs szó]* Kft. → "<szervezet> névelem"

Az ilyen szabály alapú modelleknek nagy előnye, hogy a megtalált szövegelemek szinte biztosan a keresett típusú névelem, reláció vagy esemény előfordulásai lesznek (a pontosság igen magas). Legtöbb esetben azonban, főleg a névelem-felismerésnél összetettebb problémákra a gyakorlatban nem lehetséges jól működő szabályrendszert építeni. Ennek oka elsősorban a nyelv változatossága: ugyanaz az információt rendkívül sokféleképpen fejezhető ki. Egy hatékony IE-rendszernek szüksége van a dokumentumok szövegeinek bizonyos fokú nyelvi elemzésére, hogy az információ megjelenésének sokféleségét kezelni tudjuk. Általában

- a szabályrendszerünk felidézése kritikusan alacsony lesz, azaz a feldolgozott dokumentumokban található információnak csak töredékét tudjuk kinyerni,
- a konstruált szabályrendszer mérete irreálisan nagy lesz, ha sok nyelvi mintázatot akarunk azonosítani — és így a szabályrendszer megalkotása és karbantartása túlzottan költséges lesz.

Az egyes egyedek közti relációkat, eseményeket leíró szerkezetekre vonatkozó szabályrendszerek általánosításának tekinthetők az ún. *keretek*. A keretek az adott összefüggés minden szereplőjére definiálnak feltételeket (a keretek illesztésénél nem kötelező minden szerepnek szerepelnie a mondatban). Ezek a feltételek vonatkozhatnak a vizsgált szövegrészlet

- nyelvtani tulajdonságaira, amilyen például a szófaj, az eset, az egyes/többes szám,
- jelentésére, például szervezet vagy cselekvőképes,
- másik szereptől való függőségére.

A következő példa egy vásárlással kapcsolatos eseménykeretet ír le. Megjegyezzük, hogy természetesen egy esemény azonosításához általában több keret szükséges. A példa a magyar nyelvű információkinyerő rendszerből, a NewsProból [152] származik:

```
<event schema="owner_changed.buy_sell.1">
  <rv role="buyer">msd=N case=nom</rv>
  <rv role="T1">msd=V lemma=vásárol | lemma=vesz | lemma=megvesz |
  lemma=felvásárol | lemma=megszerez | lemma=visszavásárol |
  lemma=átvesz</rv>
  <rv role="T2" modified_by_adj="new_share">msd=N direct_object
  lemma=részesedés | lemma=tulajdon | lemma=tulajdonrész | lemma=rész</rv>
  <rv role="new_share">msd=A lemma=kisebbségi | lemma=többségi</rv>
  <rv role="goods">msd=N case=ine company</rv>
</event>
```

A példakeret központi szereplői a *T1* és *T2* szerepek (célszavak). A *goods* (áru) szerepnél megfigyelhetjük a megkövetelt morfoszintaktikai (főnév -ban/-ben raggal) és szemantikai (*company* = vállalat) *jegy*eket.[1] *T2*-nél láthatunk egy példát a szerepek közti függőség megkövetelésére, a *new_share* szerep a *T2* jelzője kell legyen.

A legismertebb manuálisan kialakított referencia-kerethalmaz a *FrameNet*, ahol körülbelül 300 metakeretet definiáltak a fejlesztők [159]. A metakeretek annyira általános strukturával rendelkeznek, hogy egy eseményt egyetlen keret képes azonosítani (az *értékesítés* kerethez például 18 különböző szerepet definiáltak). A keretek középpontjában célszavak (szemantikailag hasonló kifejezések) halmaza áll. Az egyes szerepeknél a kérdéses frázis típusára (például főnév) és a célszótól való grammatikai függőségére (pl. alanya-e az igének) fogalmaznak meg feltételeket.

A FrameNetben a keretek közt is határoznak meg kapcsolatokat. Egyrészt a keretek több szempont szerint is kategorizálva vannak, másrészt a fejlesztők definiáltak hat különböző típusú relációt, amelyek maguk a keretek közt állnak fenn. Ilyen relációtípus például az általánosítás (a *csere* esemény a *vásárlás* esemény általánosítása).

Mint az összes szövegbányászati problémát, információkinyeréssel kapcsolatos feladatokat is megoldhatunk statisztikai (pl. felügyelt gépi tanulási) eszközökkel. Ehhez mindig szükség van — a feldolgozandó dokumentumhalmazon túl — tanítóadatokra, vagyis IE esetén olyan szöveges adatbázisra, amelyben a kinyerni kívánt információ meg van jelölve. Ha ilyen rendelkezésre áll, a megfelelő előfeldolgozás és modellalkotási lépések után a feladat a címkézett adatokhoz hasonló szövegelemek keresése, amelyek várhatóan szintén a kinyerni kívánt információt

[1] A nyelvtechnológiaban *jellemző* helyett a *jegy* kifejezést használják.

tartalmazó szövegelemek lesznek. Napjainkban a statisztikai IE-modellek egyre népszerűbbek, hiszen számos kedvező tulajdonsággal rendelkeznek a szakértői szabályokon alapuló rendszerekhez képest:

1. Maga a módszer lehet problémafüggetlen (más IE-feladatra is alkalmazható, másik címkézett adatbázist használva).
2. Pontossága gyakran, felidézése majdnem mindig nagyobb, mint a hasonló célt szolgáló, szakértői szabályokat használó rendszeré.

4.4. IE során felmerülő nyelvészeti problémák

Egy információkinyerő alkalmazásnak a következő nyelvi jelenségeket (vagy legalább egy részüket) kell tudnia kezelni, hogy a kívánt entitásokat, relációkat tetszőleges szövegben azonosítani tudja:

Morfológiai, szófaji azonosítás A keresett eseményeket legtöbbször igék azonosítják. Amennyiben a kérdéses szóalaknak lehetséges főnévi előfordulása is (lásd *vár*), azt nem kell figyelembe venni az elemzéskor, így szükség lehet szófaji információra. Általában véve is, bizonyos szereplők (relációk) csak jól meghatározott mondatrészekben fordulhatnak elő, amelyek azonosítása így szükséges lehet (például tárgyrag: *vásárol valamit*).

Főnévi csoportok azonosítása Az egyes névelemeket, amelyek a felismerni kívánt relációk, események résztvevői lehetnek, gyakran különböző leíró főnévi frázisok követik, amelyek hatására a névelem távol kerülhet az eseményt azonosító igétől (pl. *A MOL felvásárolta ...* vagy *A MOL, a legnagyobb közép-európai olajipari vállalat felvásárolta ...*).

Időbeliség, módok kezelése A tények, információk kivonatolása szempontjából lényeges, hogy a cselekményt azonosító ige jelen, jövő, vagy múlt időben, illetve kijelentő vagy feltételes módban fordul elő (pl. *A MOL felvásárolta ...* vagy *A MOL felvásárolná ...*).

Tagadás Szintén fontos, hogy az adott információ állító vagy tagadó értelemben szerepel a szövegben.

Cselekményt ige helyett névszói frázissal írjuk le Például *A MOL felvásárolta ...* vagy *A MOL bejelentette a ... felvásárlását.*

Mondatszerkezet meghatározása A *Ki vásárolt?*, *Mit vásárolt?* és hasonló tények kereséséhez szükség lehet az ige és bővítményei (alany, tárgy, határozók) közötti viszony bizonyos szintű feltárására.

Névmások és utalószók értelmezése Például *A MOL tovább terjeszkedik a balkáni piacon, a vállalat újabb 30 benzinkutat nyit meg Romániában* mondatban a *vállalat* utalószó.

Szövegértelmezés Néha az adott tény megállapításához szükség van a teljes szöveg megértésére. Az ilyen bonyolult leírások automatikus feldolgozása napjainkban még nemigen lehetséges (pl. *A MOL régóta tervezte a ... felvásárlását. Most, 1 hónappal a tranzakció után ...*).

Általánosságban elmondható, hogy a legtöbb konkrét alkalmazáshoz nem szükséges a szövegek túlzottan részletes elemzése (például a teljes mondatszerkezet feltárása helyett általában elegendő az egybefüggő főnévi csoportok azonosítása, a tagadás kezelése stb.) [80]. A korábban említett — jól megoldott — kurzusinformációkat bányászó alkalmazáshoz is elegendő volt a legegyszerűbb nyelvi jelenségek hatékony kezelése.

4.5. Tulajdonnév-felismerés

A legtöbb információkinyerési feladat egyik lépése a lényeges szereplők azonosítása a szövegben. A lényeges (valós információigényhez kapcsolódó) szereplők a dokumentumokban gyakran tulajdonnévvel definiáltak.

A tulajdonnév-felismerés amellett, hogy fontos előfeldolgozási lépés, akár önálló alkalmazás is lehet. Például egy alkalmazás gyűjthet személynévlistát (mondjuk egy honlap bejárása közben) vagy akár vizsgálhatunk egyszerű együttes előfordulási gyakoriságokat, pl.: Milyen személynevek fordulnak gyakran elő egy mondatban az al-Kaidával?

Egy másik érdekes alkalmazás az orvosi kutatásokat segítheti: ha az orvosi dokumentumokat (például zárójelentéseket) a kórházak egymás közt megoszthatnák, akkor létrejöhetne egy olyan szöveges adatbázis, amiből hasznos statisztikákat nyerhetnek ki a kutatók. Ennek legnagyobb gátja a személyiségi jogok védelme. A feladat tehát a dokumentumokból eltávolítani minden olyan elemet, ami személyiségi jogokat sérthet (*anonimizáció*) [173]. Az ilyen kifejezések éppen a névelemek: a páciens és az orvos neve, a kórház, a helyszín, az azonosítók, a dátumok, a telefonszámok stb.

Érdemes végül megemlítenünk, hogy a tulajdonnevek azonosítása nem csak a szövegbányászatban, de más természetes nyelvi problémák megoldásában is kulcsszerepet játszhat. Például a gépi fordításban más szabályokat kell alkalmaznunk a tulajdonnevekre, mint egyéb frázisokra:

George Bush $\neq$ György Bokor

4.5.1. Névelem

Névelem (named entity) minden olyan tokensorozat, amely a világ valamely entitására egyedi módon (unikálisan) referál. A névelemek halmazába tartoznak a tulajdonnevek, az azonosítók, a telefonszámok, az e-mail címek stb. Mivel a tulajdonnéven kívüli névelem-kategóriák viszonylag egyszerűen felismerhetőek reguláris kifejezések (ld. a 3.3.2. pontot) segítségével, ebben a részben csak a tulajdonnév-felismerés komplikáltabb problémájával foglalkozunk.

Gyakorlati alkalmazásokban nem ragaszkodnak a tulajdonnév *A magyar helyesírás szabályai* szerinti definíciójához, helyette az adott kinyerési feladatra szabott meghatározást követik. Például az *Árkád bevásárlóközpont* kifejezést érdemes egy egységként kezelni, bár a helyesírási szabályok szerint a *bevásárlóközpont* nem része a tulajdonnévnek.

A ma hatékonyan működő rendszerek a felismerendő névelemosztályokat példák segítségével írják le. Ebben a klasszikus felügyelt tanulási (részletesen ld. az 5. fejezetet) megközelítésben az adott problémához rendelkezésre áll egy tanító-adathalmaz, ahol a megfelelő tulajdonneveket manuálisan bejelölték. A cél ennek felhasználásával olyan modell megtanulása, amely ismeretlen szövegen is hatékonyan felismeri az adott kategóriákat. Fontos megjegyeznünk, hogy a tanult modell csak a tanítóhalmazzal megegyező tulajdonságú szövegeken működik pontosan.

A tulajdonnév-felismerés problémájával a 90-es évek közepétől foglalkoznak. A MUC [39] sorozat angol nyelvű újsághírek automatikus feldolgozását tűzte ki célul. A MUC-7 során a névelemek azonosítása és a személynév, földrajzi név, szervezet, egyéb tulajdonnév, időpontot jelölő kifejezés, valutanem kategóriákba sorolása volt a feladat. Erre a klasszikus tulajdonnév-felismerési feladatra példa az alábbi mondat címkézése:

> *[Sólyom László]*$_{\text{PER}}$ *[Magyarország]*$_{\text{LOC}}$ *köztársasági elnöke az [MTV-nek]*$_{\text{ORG}}$ *elmondta . . .*
> PER = személynév (person)
> LOC = helyre utaló kifejezés (location)
> ORG = szervezet (organisation)

Az utóbbi években az angol nyelv mellett előtérbe kerültek más nyelvek is, például a spanyol, a német, a kínai[2] stb. A CoNNL által meghirdetett versenysorozaton 2003-ban a MUC tulajdonnévosztályainak azonosítása volt a feladat egyazon modellel angol és német nyelvű szövegben [163]. Ez az adatbázis a

[2] A feladat kínai nyelven különösen bonyolult, hiszen a szóhatárokat nem jelölik írásban.

tulajdonnév-felismerés talán legfontosabb angol nyelvű referenciakorpusza. A *CoNLL-adatbázis* az RCV1-korpusznak (ld. a 143. oldalt) egy szegmense, amelyben a négy fentebb említett tulajdonnév-kategóriát kézzel bejelölték. Az adatbázis újságcikkekből áll (mérete 330 ezer token), a hírek témájukat tekintve igen változatosak, felölelik a gazdaság, a sport, a politika témakörét.

Az utóbbi néhány évben megfigyelhető a tulajdonnév-felismeréssel foglalkozó közösség egy másik irányba történő elmozdulása. Ez a különböző tématerületek (domain) tulajdonneveinek felismerése. A legkutatottabb területek a biológiai szövegek (például fehérjenevek felismerése) [98] és az orvosi szakszövegek — érdekesség például, hogy itt meg kell különböztetni a személyneveken belül az orvosok és pácienseik nevét [173]. Amellett, hogy az egyes szakterületek szövegei eltérő jellegzetességekkel bírnak, a felismerendő tulajdonnévkategória-rendszer is jelentősen eltérhet, például:

Analysis of [myeloid-associated genes]$_{\text{DNA}}$ *in [human hemetopoietic cells]*$_{\text{TYPE}}$
DNA = DNS
TYPE = sejttípus

Több próbálkozás is született a tulajdonnevek általános és teljes kategorizálására. A legismertebb Sekine hierarchikus rendszere [171], amely 140 tulajdonnév-kategóriát különböztet meg.

A magyar nyelvre jelenleg egyetlen tulajdonnévkorpusz létezik, ami különböző statisztikai módszerek tanítási adatbázisául szolgálhat. A SzegedNE korpusz[3] [181] annotációs sémáját tekintve megegyezik a CoNLL-korpusszal, mérete 200 ezer token. Az annotáció folyamán a Szeged TreeBank gazdasági rövidhíreket tartalmazó alkorpuszának tulajdonnevei lettek kézzel bejelölve majd egyértelműsítve.

4.5.2. A tulajdonnév-felismerés problémaköre

A megoldandó feladat kétszintű: egyrészt fel kell ismerni a szöveg(ek)ben az előre definiált kategóriákba tartozó tokensorozatokat, másrészt be kell sorolni azokat a megfelelő osztályokba.

Az osztályozás során meg kell különböztetni a tulajdonnevek kezdő tokenjeit és a tulajdonnév részét képező belső elemeket. Ennek akkor van jelentősége, amikor a szövegben egymást követően több, azonos kategóriába tartozó tulajdonnév található, mert ilyenkor ezek segítségével állapítható meg, hogy hol ér véget az egyik és kezdődik a másik. Az annotáció minden egyes tokenhez hozzárendel egy

[3] Kutatási célokra ingyenesen felhasználható. Letölthető: `inf.u-szeged.hu/hlt`

tulajdonnévosztály-címkét. A címke prefixe alapesetben 'I-'. Ha a szóban forgó tokennel egy tulajdonnév kezdődik, aminek típusa megegyezik az előző tokenével, akkor 'B'-vel jelöljük a címkét. Megjegyezzük azonban, hogy az ilyen esetek nagyon ritkák, a CoNLL-adatbázisban kevesebb mint húsz ilyen fordul elő.

Az $OTP_{\text{I-ORG}}$ $Bank_{\text{I-ORG}}$ $Postabankra_{\text{B-ORG}}$ *tett ajánlatának részletei titkosak.*

A tulajdonnév-felismerést a következő két ok teszi nehéz problémává:

- Az egyes osztályok nyíltak, azaz nem lehet elemeiket felsorolni. Minden nap új cégek alakulnak, újabb márkák kerülnek a piacra. Még olyan listát sem tudunk megadni, amiben például minden Magyarországon előforduló keresztnév szerepel, gondoljunk csak a bevándorlókra.
- Az azonosított tokensorozatok osztályokba sorolása függ a szövegkörnyezettől, ugyanis ugyanaz a kifejezés több tulajdonnévosztályba is kerülhet. Ennek eldöntése néha még az ember számára sem egyszerű feladat.

  [H. Ford]$_{\text{PER}}$ *a [Ford]*$_{\text{ORG}}$ *cég alapítója.*
  A [Ford repülőtérre]$_{\text{LOC}}$ *[Ford]*$_{\text{MISC}}$ *gépjárművel érkező . . .*

  A tulajdonnév-felismerés tulajdonképpen már félúton helyezkedik el a szintaktikai és a szemantikai analízis között: szintaktikai jegyek felhasználásával szemantikai döntést kell meghoznunk.

Automatikus tulajdonnév-felismerő rendszerek kiértékelésekor először kiszámolják az egyes tulajdonnévosztályokra vonatkozó pontosságértékeket, majd ezeket valamilyen módon átlagolják (általában mintaszámmal arányos súlyozással), hogy az egész rendszer jóságára vonatkozó mértéket kapjanak. Az egyes feladatokon használt kiértékelők azonban eltérhetnek abban, hogy mit tekintenek a pontosság és a felidézés alapegységének. A MUC-feladat kiírásában tokenenként számolták ezeket az értékeket, míg a CoNLL-versenyen frázisszinten. Ezért nem szabad a MUC 95%-os eredményeit a CoNNL-verseny legjobb rendszerének 89%-os eredményével közvetlenül összehasonlítani.

etalon: $Nemzeti_{\text{ORG}}$ $Kutatási_{\text{ORG}}$ $és_{\text{ORG}}$ $Technológiai_{\text{ORG}}$ $Hivatal_{\text{ORG}}$
predikált: $Nemzeti_{\text{ORG}}$ $Kutatási_{\text{ORG}}$ *és* $Technológiai_{\text{LOC}}$ $Hivatal_{\text{ORG}}$

A fenti példára a tokenszintű kiértékelési metrika a

$$\text{Pontosság}_{\text{ORG}} = 3/3 \quad \text{Felidézés}_{\text{ORG}} = 3/5 \quad F_{1,\text{ORG}} = 75\%$$

értékeket adja, hiszen három tokenre helyes predikciót adott a rendszerünk. A frázisszintű azonban a

$$\text{Pontosság}_{\text{ORG}} = 0/2 \quad \text{Felidézés}_{\text{ORG}} = 0/1 \quad F_{1,\text{ORG}} = 0\%$$

értékeket adja, hiszen a rendszer által jelölt két frázisból (*Nemzeti Kutatási*, *Hivatal*) egy sem helyes, valamint az etalonban található egy darab frázis (*Nemzeti Kutatási és Technológiai Hivatal*) nincs bejelölve. Ráadásul, ha kiszámoljuk az $F_{1,\mathrm{LOC}}$ értéket is, láthatjuk, hogy ennek az egy tokennek a félrejelölésével három hibapontot gyűjtöttünk be. Ennek a mértéknek a nyilvánvaló célja az, hogy a frázisok határainak pontos felismerésére kényszerítse a rendszereket.

A kiértékelő mértékek az aggregált (egész rendszerre vonatkozó) mérték számításakor általában csak a tulajdonnévosztályokat veszik figyelembe, tehát az átlagba nem számít bele — a mennyisége miatt általában $\geq 99\%$-os — nem-tulajdonnévi osztály. Ez alól a szabály alól is vannak azonban kivételek. Például az orvosi anonimizálási feladatnál fontos, hogy a nem-tulajdonnévi tokeneken milyen az anonimizálás pontossága (lényeges, hogy a rendszer ne távolítson el feleslegesen szövegelemeket).

4.5.3. A tulajdonnév-felismerésben hasznosítható jellemzők

Az alábbi egy-egy tokenre vonatkozó jellemzőkre alapozva építhetőek gépi tanulási modellek, de ezeknek a jegyeknek a felhasználása szabály alapú szakértői rendszerek kialakításához is szükséges.

Felszíni jellemzők A szóalakra, mint betűsorozatra vonatkozó információk:

- A token kis vagy nagy kezdőbetűvel kezdődik.
- A token arab szám, ill. lehet-e római szám.
- Márkanevek, cégnevek, fehérjék nevei gyakran tartalmaznak számot, nagybetűt a szó belsejében (például *PostaBank*, *Mig-29*).
- Egyszerű statisztikai módszerekkel kigyűjthetünk olyan betűpárokat, -hármasokat, amelyek valamelyik tulajdonnévosztályra jellemzőek (például a *vic* végződés a szláv személynevekben).

Környezeti jellemzők A token környezetére vonatkozó hasznos információk:

- Mondaton belüli pozíció, különös tekintettel arra, hogy a token az első szó-e a mondatban.
- Az adott token zárójelek vagy idézőjelek közt szerepel.
- Az adott tokent megelőző/követő szavak vagy kifejezések gyakran indikálják az egyes tulajdonnévosztályokat, például eléggé valószínű, hogy a *dr.* token után egy orvosnak a neve következik. Ilyen releváns tokeneket kereshetünk a tanító-adatbázisban, például a szóalakok egyes osztályok közti entrópiája alapján:

$$H(k) = \sum_{c \in C} \frac{cf_k}{n_{k,c}} \log \frac{cf_k}{n_{k,c}},$$

ahol C az osztálycímkék halmaza, cf_k a t_k előfordulásainak száma az adatbázisban (gyűjteménytámogatottság), és $n_{k,c}$ t_k azon előfordulásainak száma, ahol közvetlenül c osztálybeli elem előtt áll.

Frekvenciainformációk Ha rendelkezésre áll olyan releváns méretű szöveghalmaz, amelyből az adott nyelv legtöbb szóalakjának gyakoriságára gyűjthető adat, az alábbi információk igen hasznosak lehetnek a tulajdonnév-felismerés számára. Magyar nyelvre a SzóSzablya [83] projekt keretében a magyar web 1,5 milliárd szövegszava alapján készítettek ilyen gyakorisági listát.

- A nagyon ritkán előforduló (vagy korábban soha nem látott) tokenek gyakran tulajdonnevek részei, így maga a token előfordulási gyakorisága is egy igen hasznos jellemző.
- A szóalak-gyakorisági adatokból egyszerűen választ kaphatunk arra a kérdésre, hogy az adott token milyen arányban szerepelt kis, illetve nagy kezdőbetűvel a webkorpuszban.
- A gyakorisági szótárakban általában jelölik, hogy az adott szóalak a mondat elején szerepelt-e, így azt is tudhatjuk, hogy a nagy kezdőbetűs előfordulások hány százaléka mondat eleji.

Tulajdonnévszótárak Külső forrásból összegyűjtött listák, például keresztnevek, vállalattípusok (mint pl. *kft.*, *rt.*), nagy városok és országok stb. Az internetről néhány óra alatt összegyűjthetőek a feladat szempontjából legfontosabb listák. A listák egyrészt soha nem lehetnek teljesek (nyílt osztályok), és az sem biztos, hogy ha valami szerepel egy listában, akkor az tényleg az adott osztályba tartozik (a szövegkörnyezet befolyásolhatja azt — lásd fentebb a *Ford* példáját), mégis egy statisztikai modell számára hasznos bemeneti információt szolgáltathatnak.

Egyértelmű tulajdonnevek listája Tulajdonnévlistákat előállíthatunk a tanítóhalmaz alapján is, az ott egyértelműen osztályozhatónak tartott névelemekből. A túltanulás elkerülése érdekében az osztályokba tartozás gyakorisága mellett itt érdemes a globális gyakoriságot is figyelembe venni (a ritka kifejezéseket figyelmen kívül kell hagyni).

Ezeknek az egyszerű jegyeknek a felhasználásával már 90% körüli pontosságú statisztikai tulajdonnév-felismerő rendszerek építhetőek. A mélyebb szintaktikai

információk (POS-kódok vagy chunk-kódok), szakterületfüggő nyelvi erőforrások használata általában nem javít szignifikánsan a rendszereken, sőt mivel azok automatikus jelölése igen zajos lehet, néhol még össze is zavarhatják a gépi tanulókat.

Másrészről éppen ezek az egyszerű jegyek teszik lehetővé, hogy a felismerő rendszerek nagyon könnyen adaptálhatóak egyik nyelvről a másikra, illetve egyik tulajdonnév-felismerési feladatról a másikra. Megfelelő méretű adatbázis megléte esetén ugyanis a statisztikai modellek a felszíni jegyekből is képesek megtanulni szabályszerűségeket az adott nyelvre és feladatra vonatkozóan [66].

4.5.4. Szekvencia és token alapú modellek

A tulajdonnév-felismerés problémájára épített statisztikai rendszerek két különböző alapgondolat mögé sorakoztathatók fel. Az egyik megközelítés a tokenek szekvenciáit (általában a mondatokat) tekinti alapegységnek. Célja olyan modell építése, ami egy egész mondat osztálycímke-sorozatát képes predikálni (kiválasztja a legvalószínűbb sorozatot). Ezt a szekvencia alapú megközelítést a szakirodalom *strukturált előrejelzés*nek (structured prediction) is hívja, mivel a cél a címkék egy — változó hosszúságú — struktúrájának az előrejelzése.

A másik irányzat az egyes tokeneket különálló egyedként kezeli, és olyan modell építése a célja, amely új, korábban még nem látott tokenek címkéjét automatikusan tudja előrejelezni. Természetesen ebben a modellben sem vesznek el a token környezetére vonatkozó információk, a környező szavak legfontosabb jellemzőit felvehetjük a tokent leíró jellemzővektorba.

Ennek a megközelítésnek a legfőbb előnye, hogy a problémát visszavezeti a klasszikus felügyelt tanulási feladatra, amihez kapcsolódóan számos hatékony és gyors algoritmus létezik (a legfontosabbak bemutatása megtalálható az 5. fejezetben). Emellett a token alapú modellmegközelítés — mivel az egymást követő tokeneket is függetlennek kezeli — lehetővé teszi, hogy a mondatból kiemeljünk tokeneket, így akár tetszőleges eloszlású mintákat generáljunk. Például egy — osztálycímkék tekintetében — kiegyensúlyozott minta gyakran kedvez a tanuló-algoritmusoknak.

Ebben a részben a három legismertebb szekvencia alapú tanulómodellel foglalkozunk részletesebben. Ennek két oka van. Egyrészt ezek a modellek általában sikeresebbek a tulajdonnév-felismerési feladatokban, másrészt a token alapú megközelítéshez kapcsolodó felügyelt tanulási módszereket az 5. fejezet részletesen tárgyalja.

A legkorábbi szekvencia alapú tanuló a *rejtett Markov-modell* (hidden Markov model, HMM) [126]. Egy elsőrendű Markov-modell két típusú valószínűséget használ:

- Az átmeneti valószínűségek hivatottak jelezni — a tulajdonnév-felismerés feladatánál maradva — az egymást követő címkék közti átmenetek valószínűségeit, tehát például a $P(\text{ORG}|\text{LOC})$ feltételes valószínűség jelöli, hogy milyen eséllyel követ egy földrajzi kifejezést egy szervezetre vonatkozó kifejezés.
- Az emissziós eloszlások azt írják le, hogy az egyes jellemzőértékek jelenléte esetén mekkora a valószínűsége az egyes tulajdonnévosztályoknak (például $P(\text{ORG}|\text{nagykezdőbetűs-e a token})$).

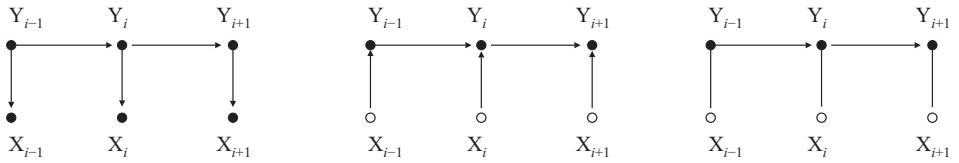
Amennyiben minden átmenet és emissziós eloszlás ismert a szekvenciára (mondat), a feladat a legvalószínűbb út (címkeszekvencia) megtalálása. A legismertebb ilyen módszer *Viterbi algoritmusa*. Magukat az eloszlásokat egy felcímkézett adatbázis felhasználásával becsülhetjük például a *Baum–Welsh-eljárás* segítségével.

A rejtett Markov-modellekkel kapcsolatban két problémát említenek a szakirodalomban:

- A modell feltételezi az egyes tokeneket leíró jellemzők egymástól való függetlenségét. Ez a tulajdonnév-felismerésben — ahol nagy számú egymással összefüggő jellemző használatára van lehetőség – különösen megbízhatatlanná teszi ezeket a modelleket.
- A tanulóalgoritmus tulajdonképpen a tokensorozat valószínűségében maximalizál, holott a cél a legvalószínűbb címkesorozat megtalálása.

A maximum entrópia Markov-modell (maximum entropy Markov model, MEMM) [129] nem használ külön átmeneti és emissziós valószínűségeket, hanem egyetlen feltételes valószínűségbe vonja ezeket össze: mekkora az egyes címkék valószínűsége a megelőző címke és a megfigyelt jellemzővektor függvényében. Ezt az eloszlást egy exponenciális modell segítségével becsli meg, ahol az optimalizációs feladat tulajdonképpen az egyes jellemzők súlyának a megtanulása. Így a MEMM tanítása csak némileg komplexebb, mint a HMM-é, viszont empirikusan bizonyított, hogy a legtöbb probléma esetén sokkal hatékonyabb.

A MEMM problemája az, hogy a tanítás folyamán megelőző címkeként a zsinórmértéknek tekintett (gold standard) címkéket használja, a predikciós fázisban azonban az előző címke nem biztos, hogy helyes, így ez a „jellemző” igen zajossá válhat (a *label bias probléma*).



4.2. ábra. A HMM-, a MEMM- és a láncstruktúrájú CRF-modellek grafikus reprezentációja (balról jobbra). Az üres pont azt jelöli, hogy a változót nem a modell generálja. *Forrás:* [108]

A HMM és a MEMM is lokális eloszlásbecsléseket hajt végre, majd a Viterbi-algoritmus segítségével kiválasztják a legvalószínűbb utat. Ezzel szemben a label bias problémára megoldást kínáló *feltételes valószínűségi mező* (conditional random fields, CRF) [108] valóban az egész struktúra előrejelzését végzi el. A tanítás effektíven csak a legegyszerűbb, láncszerkezetű véletlen valószínűségi mezőn végezhető el a *forward-backward algoritmus* több iteráción keresztül történő futtatásával. Napjainkban a legsikeresebb a tulajdonnév-felismerő rendszerek CRF-et használnak. A 4.2. ábrán a három röviden bemutatott modell grafikus ábrázolása látható.

4.5.5. Ingyenes tulajdonnév-felismerő rendszerek

A világban számos ingyenesen hozzáférhető tulajdonnév-felismerő rendszert implementáltak. A két legszélesebb körben használt és megbízható rendszer:

MALLET-csomag (egyéb statisztikai nyelvfeldolgozási modulok mellett) tartalmaz egy könnyen kezelhető CRF-implementációt. A Java API segítségével bármilyen adatbázisra taníthatunk CRF-et [127].[4]

LingPipe szintén egy tanítható keretrendszer. Egy HMM alapú tulajdonnév-felismerő rendszert (Java) tartalmaz, valamint számos modell is letölthető, amelyeket a fejlesztők a legismertebb referencia-adatbázisokon optimalizáltak [36].[5]

4.6. Kereszthivatkozások feloldása

Kereszthivatkozások feloldása alatt annak eldöntését értjük, hogy két természetes nyelvi kifejezés ugyanazt az entitást jelöli-e vagy sem. Ez a feldolgozási lépés

[4] Letölthető: `mallet.cs.umass.edu/index.php/Obtaining_MALLET`

[5] Letölthető: `www.alias-i.com/lingpipe/web/download.html`

kritikus fontosságú az információkinyerő rendszerek szempontjából, hiszen a kinyert tények legtöbbször valós személyekre, cégekre, tárgyakra stb. vonatkoznak. Vegyük a következő példát:

> *„Clinton a demokraták elnökjelöltje", „Hillary Rodham Clinton"*
> Felvegyük-e a *„Hillary Rodham Clinton"* nevet az Egyesült Államok demokrata elnökjelöltjeit tartalmazó listába vagy sem?

Jól mutatja ez a példa az IE-feladatok korpuszfüggőségét — és így időfüggőségét. Egy korábbi évekből származó szöveg esetén a válasz egyértelműen nem, de 2008-ban talán Hillary Clinton is elnökjelölt lesz.

A kereszthivatkozások feloldása két fő lépésből áll. Az egyik az ugyanarra az entitásra hivatkozó névelemek egymáshoz rendelése (kereszthivatkozási lánc előállítása), a másik feladat pedig a névmások, visszautaló névszói csoportok hozzárendelése a hivatkozott névelemhez, és ezáltal bekapcsolásuk a kereszthivatkozási láncba. Az utóbbi feladatot, ahol a visszautaló névszói csoportokat a hivatkozott (szövegben korábban megjelenő) névelemhez rendeljük, *anaforafeloldás*nak nevezzük. A koreferenciafeloldás és azon belül az anaforafeloldás is intenzíven kutatott probléma nemcsak szöveges dokumentumokon [59]. Megoldásához a szöveg szintaktikai elemzését [28], ontológiákat, enciklopédikus tudást [147] is felhasználva gyakran alkalmaznak gépi tanulási technikákat [177]. A magyar nyelvű koreferencia- és anaforafeloldásra is születtek már eredmények, ld. [114, 146, 200].

Egy példa kereszthivatkozási láncra:

> *„George Bush elnök kedvenc háziállata a Spotty nevű spániel. George megetette a kutyát. Ezután az elégedetten leheveredett."*

A példában található kereszthivatkozási láncok:

George Bush ⟵ *George; Spotty* ⟵ *a kutyát* ⟵ *az*

Egy egyszerű, szabály alapú kereszthivatkozás-feloldó

1. fázis A névelemek összerendelése:

1. Teljes egyezés: két egyforma névelem ugyanarra hivatkozik.
2. Ekvivalens névelemek (szinonimák): a gyakori alternatív elnevezéseket, pl. *Szeged – „a napfény városa", Széchenyi István – „a legnagyobb magyar", IBM – „Big Blue"* egy szinonimalista alapján rendeljük össze.
3. Toldalékolt alakok: ugyanannak a névelemnek a toldalékolt alakjait egymáshoz rendeljük, mert legtöbbször ugyanazt az entitást hivatkozzák, pl. *Szeged, Szegeden, Szegedről.*

4. Kivétellista: az olyan gyakori névelemeket, amelyekről tudjuk, hogy nem rendelhetők össze, szintén összegyűjthetjük egy listában, hogy a későbbi lépésekben ne kerüljenek összevonásra, pl. *Siemens – Siemens Transzformátor Kft.*
5. Tokenszintű egyezés: a különböző írott változatokat rendeli össze, pl. *Kovács István – István, Kovács.*
6. Kifejezés eleji egyezés: ha egy névelem egy másiknak a rövidebb formája, akkor valószínűleg összetartoznak, így összerendelhetők. Kivétel, ha több lehetséges jelölt van, azaz a rövidebb alak több hosszabb dokumentumbeli névelemre is illeszkedik, pl. *Kovács István – Kovács* (ha nincs másik *Kovács* a dokumentumban).
7. Akronim illesztés: a betűszavak összerendelhetők a hosszú alakkal, pl. *International Business Machines – IBM.*
8. Többszavas illesztés: egy rövidebb névelem minden szava megtalálható pontosan egy hosszabb alakban, pl. *Első Beton Sportegyesület – Első Beton* (de itt sem illesztünk, ha szerepel a szövegben pl. *Első Beton Kft.* is).
9. Heurisztikus összerendelés: kereshetünk egyéb olyan, egyszerű heurisztikákat, ami a feldolgozandó dokumentumhalmazban helyesen rendel össze névelemeket, pl. a tokenszintű és toldalékolt alakok egyezését egyben vizsgálva összerendelhetők a *Budapesti Műszaki Egyetem – Műszaki Egyetem, Budapest* alakok (amennyiben a korpuszban jól működik egy efféle illesztő).

2. fázis A névmások és hivatkozó értelmű, tartalmas főnévi csoportok feloldása:

1. A névmás (pl. *ő*) / névszói csoport (pl. *a cég*) a helyettesítő szövegkörnyezetének (érvényességi körének) meghatározása, pl. ilyen lehet az adott bekezdés.
2. A lehetséges helyettesített elemek (ezek sokszor névelemek) keresése a szövegkörnyezetben.
3. A jelöltek szűrése, pl. típus (*a cég* csak szervezetre referálhat), nem (*ő* általában személyre, *az* általában termékre vagy cégre vonatkozik, *úr / hölgy / Mr. / Mrs.* csak megfelelő nemű egyedre hivatkozhat) stb. alapján.
4. A jelöltek rangsorolása, illetve a rangsor alapján a legvalószínűbb jelölt és a helyettesítő összerendelése. A rangsor alapja lehet (összerendelési lehetőségekről, a nyelv visszautalási sajátosságairól bővebb információ található a [146, 200] tanulmányokban):

- A helyettesítőtől vett szótávolság, pl. sokszor a legutóbbi elemet helyettesítjük névmással.
- Mondattani információkat is felhasználó heurisztika, pl. a mondatfában a legközelebbi jelölt választása, a mondatok közti illesztésnél, pl. alany–alany összerendelése.
- Tartalmas névszói csoportok névelemhez rendelése esetén taxonómia is felhasználható az összerendelés támogatására, pl. *Jancsi megetette a kutyát. A puli ezután elégedetten leheveredett.* – itt ha egy taxonómia segítségével tudjuk, hogy a puli a kutya egy fajtája, akkor az *A puli* névszói csoport feloldása triviális.
- Az adott nyelv sajátosságait kihasználó heurisztika. A magyarban, ha hiányzik az alany egy mondatból, akkor az alany megegyezik az előző mondatéval, ha változik az alany, kötelező kitenni a névmást, pl. *János megcsókolta Marit. Leült és rágyújtott.* – itt *János* a második mondat alanya, de ha a második mondat *Az leült és rágyújtott.*, akkor az „*az*” *Marira* vonatkozik.

5. fejezet

Osztályozás

Amint azt a bevezetőben bemutattuk, a szöveges adatok egyik legfőbb jellemzője, hogy az információt strukturálatlan vagy gyengén strukturált formában tartalmazzák. A dokumentumokhoz ritkán tartoznak leíró metaadatok — holott erre számos elterjedt formátum (PDF, MS-WORD stb.) lehetőséget nyújt —, legfeljebb csak néhány automatikusan kitölthető mező, pl. cím, szerző van megadva. Amíg csak néhány tucat, vagy esetleg pár száz dokumentumról van szó, addig ez a hiányosság nem jelent feltétlenül számottevő problémát, bár a kezelhetőséget nem könnyíti meg. A dokumentumok visszakeresését nagymértékben segíti, ha rendszerezve tároljuk őket. Éppen ezért sok felhasználó a saját számítógépén rendezve tárolja szöveges adatait, pl. dokumentumait strukturált könyvtárrendszerben, e-mailjeit tematikus mappákban tárolja. Általánosan is megoldást jelent a strukturálatlan adatok kezelésére a strukturált tárolási mód, vagyis amikor a dokumentumokat hierarchikus rendszerbe, ún. *taxonómiá*ba rendszerezzük. A taxonómia úgy működik, mint egy számítógépes könyvtárstruktúra, ami kézenfekvő és intuitív eszközt ad a navigálásra és az információk elérésére, keresésére [26]. Bár a besorolást több független szempont szerint is végrehajthatjuk, ezek közül szinte mindig a *tartalom* szerinti besorolás a legfontosabb. Emellett párhuzamosan alkalmazható pl. keletkezési idő és hely szerinti besorolás is.

A dokumentumok tartalom szerinti rendszerezésének automatizálása az egyik legalapvetőbb szövegbányászati feladat, amelyet szövegosztályozásnak vagy egyszerűen osztályozásnak neveznek. A *szövegosztályozás*[1] célja szöveges dokumentumok előre definiált halmazból vett tematikus kategóriacímkékkel való ellátása. Az osztályozás a 90-es évek elejétől kezdve került a kutatások homlokterébe, ekkor a számos alkalmazási területen megjelenő érdeklődést a nagykapacitású számítógépes háttér elérhetővé válása is elősegítette. Az alkalmazások spektruma igen széles, pl. a rögzített tárgyszóhalmaz alapú dokumentumannotálás, a doku-

[1] Ezt a tevékenységet más néven *kategorizálás*nak, *klasszifikáció*nak, ill. *besorolás*nak is nevezik (text categorization/classification, TC).

mentumszűrés (pl. kéretlen leveleké), az automatikus metaadat-generálás, a dokumentumok nyelvének, ill. szerzőjének meghatározása, többértelmű szavak egyértelműsítése, internetes keresések támogatása a keresőkifejezések tematikus besorolásával, webes dokumentumgyűjtemények feltöltése, vélemények értékelése, illetve általánosan minden olyan feladat, amelynél szükség van dokumentumok rendszerezésére vagy adaptív válogatására (részletesebben ld. az 5.2. szakaszt).

A szövegosztályozás területén egészen a 80-as évek végéig az a tudásmérnöki megközelítés volt az uralkodó, amely szabályokba kódolva adta meg a dokumentumok osztályozásáról rendelkezésre álló szakértői tudást. Ez aztán átadta helyét a *gépi tanulás*i (machine learning) paradigmának, ahol tanítóadatok alapján egy automatikus következtető folyamat eredményeképpen készül el a kategóriák jellegzetességeit felismerni képes osztályozó. Ez a megközelítés gyakorta az emberekéhez hasonló, sőt időnként azt felülmúló pontosságú módszert hoz létre. Az emberi erőforrásra csak a *kategóriákból* és a *hozzájuk rendelt dokumentumokból* álló *tanítókörnyezet* előállításánál van szükség, ettől eltekintve az osztályozó létrehozása teljesen automatizálható.

A szövegosztályozás bármely területen való sikeres alkalmazásának előfeltétele a megfelelő minőségű *tanítókörnyezet* megteremtése. Például üzleti célú felhasználásnál ez azt jelenti, hogy létre kell hozni a szervezet profilját leíró vállalati taxonómiát, és ezt fel kell tölteni elegendő mennyiségű (kategóriánként legalább 5–10) jó minőségű tanítóadattal. A taxonómia megalkotása és fenntartása — bár a piacon vannak ezt támogató szoftverek: Verify, Stratify, Inxight, Autonomy — korántsem triviális feladat: olyan szakértőt kíván, aki átlátja az egész cég üzleti szervezetét, és rendszerező képességgel bír. A kategóriák száma gyakran elérheti a tízezres nagyságrendet is.[2] Egy cég profiljának, termékeinek változása a taxonómia változtatásának szükségességét is magával vonja, ami szintén időigényes és költséges feladat. Hasonlóan nagy volumenű munka a taxonómia feltöltése tanítóadatokkal, amelyre többnyire nem szánnak elegendő időt az üzleti felhasználók. A nagy szoftvercégek piacon lévő szoftverei tudományos szempontból már gyakran elavultnak tekinthető osztályozó módszerek implementációit tartalmazzák, ezért egy nagy dokumentumkezelő keretrendszer bevezetése helyett alternatívaként megfontolandó az egyedi fejlesztésű alkalmazások létrehozása is.

[2] Erre jó példa a nemzetközi szabadalmi hivatal által kifejlesztett IPC taxonómia: `www.wipo.org/classifications/fulltext/new_ipc/index.htm`

5.1. Az osztályozás definíciója és alesetei

Az osztályozás feladata dokumentumok kategóriacímkékkel való ellátása. Formálisan egy $\Phi : \mathcal{D} \rightarrow 2^C$ osztályozófüggvény megalkotása a cél, amely a $\mathcal{D}$ dokumentumtér elemeihez a $C = \{c_1, \ldots, c_{|C|}\}$ *kategóriarendszer*ből vett *kategóriá*k (másnéven *osztály*ok) halmazát rendeli. A Φ függvényt röviden *osztályozó*nak nevezzük. Az osztályozó megalkotásakor a cél az ismeretlen $\tilde{\Phi} : \mathcal{D} \rightarrow 2^C$ célfüggvény minél pontosabb közelítése, azaz Φ és $\tilde{\Phi}$ eltérésének minimalizálása. Az eltérés, azaz a közelítés hatékonyságának mérését az osztályozó módszerek összehasonlításánál tárgyaljuk (ld. az 5.5.2. szakaszt).

Az osztályozót ismert kategóriájú dokumentumokból kinyert információk alapján építjük fel. Ezeket *tanítódokumentum*oknak nevezik, amelyeket — csakúgy mint a kategóriarendszert — többnyire humán szakértők állítják elő. A tanítódokumentumok a hozzájuk rendelt osztályokat jól jellemző mintaadatok. Az osztályozó tanulóalgoritmusa a tanítódokumentumok alapján megtanulja az egyes osztályok jellegzetességeit, ebből modellt készít, amellyel becslés adható ismeretlen kategóriájú dokumentumok címkéjére. A tanulásnak ezt a fajtáját *felügyelt tanulás*nak (supervised learning) nevezik, ahol a szakértő az általa megadott tanítóadatokon keresztül „felügyeli” a tanulás folyamatát.

Az általános megközelítés érdekében az ismertetésre kerülő eljárásokn csak a tanítódokumentumokból kinyert információkat (ún. belső forrásból származó adatokat) használnak fel. Úgy tekintjük, hogy sem a kategóriákhoz, sem a dokumentumokhoz nem állnak rendelkezésre további külső adatok — pl. leíró értelmezés, ill. a publikáció dátuma, forrása, formátuma stb. Gyakorlati feladatok megoldásánál azonban a hatékonyság maximalizálása érdekében figyelembe lehet venni tetszőleges, külső forrásból származó adatot is.

5.1.1. Az osztályozás fajtái kategóriák száma szerint

A gyakorlati alkalmazástól függően különböző kényszerfeltételeket írhatunk elő az osztályozófüggvényre vonatkozóan. Az egyik kézenfekvő lehetőség a dokumentumhoz rendelt kategóriák számának korlátozása. Ettől függően az osztályozási feladatoknak két alaptípusát különböztetjük meg:

- *egycímkés* (single-label) esetnek nevezzük azt, amikor minden dokumentumot pontosan egy kategóriához kell hozzárendelni;

- *többcímkés* (multi-label) esetről beszélünk, amikor a dokumentumok 0-tól $|C|$-ig tetszőleges számú kategóriába besorolhatók.

Az egycímkés osztályozásnak speciális esete a *bináris osztályozás*, amikor mindössze két osztályunk van, c és komplementere, $\overline{c}$. Ha a kategóriák függetlenek egymástól, akkor a bináris osztályozókból egyszerűen készíthető többcímkés osztályozó. Ehhez elég a $c_1, \ldots, c_{|C|}$ kategóriákra vonatkozó feladatot dekomponálni $|C|$ számú független $c_j, \overline{c}_j$ bináris osztályozási feladatra ($c_j \in C$). Fordítva ugyanez nem igaz: egy többcímkés osztályozó nem alakítható át automatikusan binárisra vagy egycímkésre. Ugyanis ha az előbbi egynél több kategóriát rendel egy dokumentumhoz, akkor nem feltétlenül egyértelmű, hogy ezek közül hogyan kell a legmegfelelőbbet kiválasztani, és ha egyet sem rendel, akkor különösen nehéz a legkevésbé rossz kategória meghatározása. Tehát a bináris osztályozó — független kategóriák esetén — általánosabb eszköz, mint a többcímkés [169], ezért, valamint a könnyebb érthetőség miatt, az osztályozási módszereket bináris, ill. egycímkés feladaton mutatjuk be (ld. az 5.4. szakaszt). Az eddig ismertetett feladattípusokat együttesen *egyszerű osztályozási feladat*nak nevezzük.

Az osztályozási feladat lényegesen eltér az egyszerű osztályozásnál látott problématípusoktól, ha a kategóriák nem függetlenek egymástól. Ilyenkor általában hierarchikus taxonómiába kell besorolni a dokumentumokat. A taxonómia rendszerint irányított fagráf, illetve ritkábban körmentes irányított gráf struktúrájú. Ekkor egy gyerekkategóriába tartozó dokumentum a szülőkategóriába is beletartozik. A taxonómia jellege miatt ezt a feladattípust *hierarchikus (szöveg)osztályozás*nak nevezik (hierarchical text categorization). Ezzel részletesen az 5.6. szakaszban foglalkozunk.

További speciális eset a *többszintű* (multi-level) *osztályozás*, amely esetben a dokumentumnak lehetnek elsődleges, másodlagos stb. kategóriái. A gyakorlatban ekkor általában többcímkés, taxonómiába[3] történő hierarchikus osztályozási feladatról van szó, aminek eredményét le kell bontani az egyes szintekre — ezért ezt az alesetet is a hierarchikus osztályozásnál tárgyaljuk röviden.

5.1.2. Dokumentum- és kategóriavezérelt osztályozás

A szövegosztályozókat kétféleképpen használhatjuk. Eddig a *dokumentumvezérelt osztályozás*t (document-pivoted categorization, DPC) tárgyaltuk, amikor adott $d \in \mathcal{D}$ esetén keressük az összes d-hez tartozó kategóriát. Ezzel szemben *kategóriavezérelt osztályozás*nak (category-pivoted categorization, CPC) nevezzük azt a megközelítést, amikor adott $c \in C$ kategória esetén keressük a c-be tartozó összes dokumentumot. Ha a Φ osztályozófüggvényt úgy módosítjuk, hogy

[3] bár strukturálatlan kategóriarendszer esetén is értelmezhető a probléma

dokumentum-kategória párokhoz rendeljen értelemszerűen 0 vagy 1 értéket,

$$\Phi : \mathcal{D} \times C \rightarrow \{0,1\}, \tag{5.1}$$

akkor mindkét megközelítés egyaránt leírható vele.

A kétfajta megközelítés megkülönböztetése különösen akkor indokolt, ha az osztályozás kezdetekor nem áll rendelkezésre az összes dokumentum vagy kategória.[4] A dokumentumvezérelt megközelítést érdemes alkalmazni, ha a dokumentumok nem egyszerre válnak elérhetővé, pl. szűrési feladatoknál. A kategóriavezérelt megközelítés akkor előnyös, ha a kezdeti kategóriarendszer bővülhet, és a dokumentumokat az új kategóriákba is be kell sorolni. A gyakorlatban inkább a dokumentumvezérelt mód használata a jellemző, de kevés kivételtől eltekintve az összes osztályozó algoritmus támogatja mindkét típusú felhasználást.

5.1.3. Az eredmény típusa: kiválasztó és rangsoroló osztályozás

A megbízhatóság érdeke bizonyos szövegosztályozási alkalmazásoknál megkövetelheti, hogy a dokumentumok besorolásánál a végső döntést humán szakértők végezzék. Ilyen esetben is rendkívül megkönnyíti a szakértők munkáját és felgyorsítja a folyamatot, ha nem a teljes kategóriahalmazból kell a megfelelő osztálycímkéket kiválasztani, hanem az osztályozó rangsorolja a címkéket, és csak a rangsor elejéről kell a legadekvátabb címkét kiválasztani. Ezt *félautomatikus vagy támogató osztályozás*nak hívják (categorization assistance), jellemző alkalmazási területe például a szabadalomosztályozás [184] (ld. még az 5.6.4. pontot). Ekkor az (5.1) függvény értékkészlete nem bináris, hanem tetszőleges $[0,1]$-beli valós szám lesz:

$$\Phi : \mathcal{D} \times C \rightarrow [0,1]. \tag{5.2}$$

Így dokumentumvezérelt esetben egy adott $d \in \mathcal{D}$ dokumentumhoz rendelt kategóriák relevanciáját a C elemeihez rendelt érték alapján lehet rangsorba állítani. Az alternatív kategóriavezérelt esetben pedig analóg módon egy adott $c \in C$-hez tartozó dokumentumokat lehet ez alapján rangsorolni.

A félautomatikus osztályozás akkor is szerepet kaphat, ha a tanítóadatok minősége vagy mennyisége nem megfelelő, ezért jó eséllyel előfordulhat, hogy az ismeretlen dokumentumot félreosztályozza az adott módszer.

[4] Emellett egyes módszerek jobban támogatják a két megközelítés egyikét, ld. pl. a k-NN-t a 124. oldalon.

5.2. Az osztályozás alkalmazásai

A fejezet bevezetőjében már röviden tárgyaltuk az osztályozás lehetséges alkalmazási területeit. Tekintsünk most át ezek közül néhány tipikus feladatot (ld. még [169]).

Az osztályozás első alkalmazásainak tipikus példája az *automatikus annotálás* vagy *indexelés*. A korai IR-rendszerekben humán szakértők végezték az annotálás költségigényes feladatát, azaz a dokumentumok tárgyszavakkal való ellátását rögzített szószedet alapján. Mivel egy dokumentumhoz több tárgyszót is lehet rendelni, ezért ez egy többcímkés osztályozási feladat, ahol a kategóriák a szószedet elemei. A feladat nyilván dokumentumvezérelt megközelítést kíván, az osztályozó a még nem annotált dokumentumokhoz rendel tárgyszavakat. Mivel az esetleges tévesztés nem jár kritikus következményekkel, ezért ha az osztályozó hatékonysága a tesztelés során elegendően jónak bizonyul, akkor a folyamat teljesen automatizálható.

Az előzőhöz nagyon hasonló az *automatikus metaadat-generálás* feladata. A (digitális) könyvtárakban a dokumentumokat különböző szempontok szerinti (téma, keletkezés ideje, dokumentum típusa, elérhetősége stb.) metaadatokkal látják el. A tematikus metaadatokat egy rögzített bibliográfiai kódkészletből választják ki. Az ilyen típusú metaadatok hozzárendelését az előző bekezdésben ismertetett módon lehet osztályozással automatizálni.

A fenti két alkalmazási példa a *dokumentumrendszerezés* tágabb feladattípusának kiemelt esetei, de ugyanide tartoznak a bevezetőben említett általános rendszerezési feladatok is. A dokumentum alapú besorolásnak ez a fajtája talán az automatikus osztályozók legeklatánsabb alkalmazási területe. Az egyik tipikus alkalmazói csoport az újságok szerkesztősége, ahol akár a hirdetések, akár a cikkek kategorizálása is automatizálható. Az osztályozás ekkor egycímkés vagy hierarchikus attól függően, hogy a kategóriarendszer elemei függetlenek, vagy hierarchikus struktúrába rendezettek. Ez utóbbinak kiemelt esete a szabadalmi osztályozás.

A szabadalmi hivatalokban több feladatnál is nagy segítséget jelenthet a hierarchikus osztályozók alkalmazása [112, 184]. A szabadalmak feldolgozása során a beadványokat emberi munkával elemzik és továbbítják a megfelelő szakcsoporthoz, akik a szabadalom szakmai elbírálását és besorolását elvégzik. A szakcsoportok meghatározása akár automatikus osztályozással is kivitelezhető — ekkor az osztályozó eljárások pontatlanságából eredő hibák tolerálhatók és korrigálhatók, vagy felügyelt félautomatikus módon is elvégezhető. Az osztályozó javaslattétel formájában a szakértőknek is segítséget adhat a beadványok kategóriájának

meghatározásához. A szabadalmi hivatalok esetében ráadásul rendelkezésre állnak az osztályozók betanításához szükséges előfeltételek: a jól definiált, rögzített taxonómia és a nagy számú, jó minőségű tanítóadat.

A hierarchikus osztályozás másik alkalmazása az internetes katalógusoldalak dokumentumokkal való feltöltése. Az internetes keresés egyik alternatívája a hierarchikus taxonómiába rendezett katalógusoldalakon történő navigáció. A katalógus tartalommal való feltöltését, azaz a weboldalak besorolását a taxonómiába csak automatizáltan lehet idő- és költséghatékonyan megvalósítani. Egyes alkalmazások a dokumentumok hiperlinkjeit is felhasználják.

Az osztályozók alkalmazásának alternatív lehetősége a *dokumentumszűrés* (text filtering), amikor az osztályozó az információ-előállító és -fogyasztó közötti információs csatornán érkező dokumentumokat szűri meg a dokumentum jellege és a fogyasztó előre megadott érdeklődési profilja alapján. A dokumentumszűrésnek egyik legegyszerűbb példája a *kéretlen levelek szűrése* (spam filtering). Ekkor a csatorna a levelezőszerver és a levelezőkliens közötti kapcsolat, és alapesetben csak bináris osztályozást igényel a feladat: a kéretlen és a fontos levelek megkülönböztetését. Mivel a kéretlen leveleket gyártók újabb és újabb trükköket alkalmaznak arra, hogy a leveleik átjussanak a szűrőn, ezért a szűrőnek adaptívnak kell lennie, azaz alkalmazkodnia kell a módosult helyzethez. Ezt a feladattípust általánosan *adaptív szűrés*nek (adaptive filtering) is nevezik. A bináris osztályozási feladat rendre független vagy hierarchikus egycímkés osztályozásra általánosítható, ha a fontos leveleket az osztályozó a felhasználó által megadott egyszerű vagy hierarchikus levelezési mappastruktúrába sorolja be.

A dokumentumszűrés másik gyakori példája a hírügynökségek és szerkesztőségek közti dokumentumok válogatása, ugyanis a tematikus újságok számára (pl. sportnapilap) csak meghatározott témájú hírek érdekesek. Szintén hasonló feladat az előre megadott felhasználói érdeklődési körök (profilok) alapján történő dokumentumszűrés, amely a profil és a dokumentumtípusok időbeni változása miatt az adaptív szűrés egyik esete.

Az osztályozás egyik számítógépes nyelvészeti alkalmazása a *szavak egyértelműsítése* (word sense disambiguation), amikor többértelmű vagy azonos alakú szavak előfordulásainál[5] meg kell határozni az adott szó értelmét, azaz szemantikáját. Szövegosztályozási szempontból ez egy dokumentumvezérelt egycímkés osztályozási feladat, ahol a lehetséges szemantikákat tekintjük a kategóriáknak, és a szó szövegkörnyezetét a dokumentumoknak. Számítógépes nyelvészetben

[5] angol példa: *bank* – bank, ill. part; magyar példa *nyúl*, *termet*, *átejt* (átvitt vagy konkrét értelemben)

az osztályozásnak további, főleg egyértelműsítési célú alkalmazásai is vannak, pl. automatikus kontextusfüggő helyesírás-javítás, szófaj-meghatározás, automatikus fordítás esetén a szóválasztás támogatása stb. — bővebben ld. pl. [158].

Az osztályozás speciális esete a *dokumentum nyelvének meghatározása* többnyelvű korpuszok esetén. Ha a korpuszban nincsenek vegyes nyelvű dokumentumok, akkor a feladat az egycímkés osztályozás alá tartozik. A feladat megvalósítására egy egyszerű, ám mégis nagyon hatékony megoldás terjedt el, amely a szokásos szóalapú indexelés helyett karakter n-gramm alapú indexelést használ (ld. a 39. oldalt) [34]. Terjedelmi okok miatt e fontos módszer részletes ismertetése könyvünk internetes mellékletében található.

5.3. A tanítókörnyezet és dokumentummodell

5.3.1. A dokumentumgyűjtemény particionálása

A felügyelt tanulási paradigma esetén az osztályozó felépítéséhez rendelkezésre áll a tanítókörnyezet, azaz a $C = \{c_1, \ldots, c_{|C|}\}$ kategóriahalmaz és az ebbe besorolt $D = \{d_1, \ldots, d_N\}$ kiinduló dokumentumkorpusz. Ezen dokumentumok esetén tehát ismert a $\tilde{\Phi}$ célfüggvény értéke minden C-beli kategóriára. Egy d dokumentum a c kategóriára vonatkozóan *pozitív*, illetve *negatív tanítóadat*, ha $\tilde{\Phi}(d,c) = 1$, illetve 0.

Ahhoz, hogy a felépített osztályozó hatékonyságát meghatározhassuk, szükség van tesztadatokra. Ennek érdekében a kiinduló D dokumentumgyűjteményt két diszjunkt halmazra bontjuk, *tanító-* és *teszthalmaz*ra (training/test set): $D_{\text{Train}} \cap D_{\text{Test}} = \emptyset$, és $D_{\text{Train}} \cup D_{\text{Test}} = D$. Az osztályozót D_{Train} dokumentumain megfigyelt jellemzők alapján induktív következtetéssel hozzuk létre. A D_{Test} teszthalmaz elemeit az osztályozó hatékonyságának megállapítására használjuk: az osztályozót futtatjuk a tesztdokumentumokra, és a kapott $\Phi(d_{\text{test}}, c)$ értékeket összehasonlítjuk a célfüggvények ismert $\tilde{\Phi}(d_{\text{test}}, c)$ értékeivel. A tesztadatok semmilyen módon nem használhatóak fel az osztályozó felépítésénél, különben irreális és tudománytalan eredményeket kapunk a hatékonyságra. Ezt szem előtt tartva ugyanakkor, gyakorlati esetekben a hatékonyság maximalizálása érdekében célszerű a teljes tanítókörnyezeten újratanítani az osztályozót.

A fent ismertetett módszertan az osztályozó kiértékelésének legegyszerűbb változata. Ennek továbbfejlesztett verziója a *k-szoros keresztvalidáció* [133] (k-fold cross-validation), amikor a D kiinduló dokumentumgyűjteményt k diszjunkt részre osztjuk $D_{\text{Test}}^{(1)}, \ldots, D_{\text{Test}}^{(k)}$. Ezután a $D_{\text{Train}}^{(i)} = D \setminus D_{\text{Test}}^{(i)}$ tanítóhalmazokon tanítva felépítünk k darab Φ_i osztályozót ($i \in [1,k]$), és ezeket rendre a $D_{\text{Test}}^{(i)}$ halmazo-

kon teszteljük. A modell hatékonyságát ezután a k darab osztályozón mért hatékonyságértékek átlagaként határozzuk meg. A gyakorlatban legtöbbször 10-szeres keresztvalidációt alkalmazunk, a kiinduló dokumentumgyűjteményt lehetőleg 10 azonos méretű részre osztva.

A tanulóalgoritmusok paramétereinek finomítására gyakran további tesztek elvégzésére van szükség. Erre a célra a tanítóhalmaz egy részét szokták elkülöníteni, amelyet *validációs halmaz*nak neveznek: $D_{\text{Valid}} = D_{\text{Train}} \setminus D_{\text{NewTrain}}$. Ezt a módszertant közvetlenül a tanító–teszt kiértékelési megközelítésnél lehet használni, de könnyen adaptálható a keresztvalidációs változatra is.

*Túltanulás*nak nevezzük azt a jelenséget, amikor az osztályozó a tanítóadatokon lényegesen nagyobb hatékonysággal működik, mint a tesztadatokon. A túltanulás összefügg a modell általánosító képességével, amiről részletesen értekezünk az 5.5.1. pontban.

5.3.2. Dokumentummodell

A szövegosztályozó algoritmusok a dokumentumoknak szinte kivétel nélkül a vektortérmodell szerinti reprezentációját használják, ahol a vektortér tengelyei szavakat reprezentálnak. Ezen belül a dokumentumok indexelésére általában a normalizált tf-idf súlyozást (2.9) szokták alkalmazni, bár több munka is arról számolt be [54, 186], hogy az entrópiasúlyozás használata növeli a hatékonyságot.

Ha nem áll rendelkezésre a teljes dokumentumgyűjtemény a modell megalkotásánál, mint például felhasználók által megadott profilok alapján történő *adaptív dokumentumszűrés* esetén, akkor a hiányzó értékeket becsléssel állapítják meg [48].

A vektortér dimenziójának csökkentésére több módszert is alkalmaznak. A nyelvi alapú dimenzióredukciós eljárások közül az indexelés előtt szinte minden esetben alkalmaznak *stopszószűrés*t (ld. a 40. oldalon), ugyanakkor a *szótövezés* hasznosságának megítélése változó. Angol nyelvű korpuszoknál egyes források a hatékonyság romlásáról számolnak be [14], mások ugyanakkor néhány %-os növekedést tapasztaltak szótövezés használatakor. Általánosságban elmondható, hogy a szótövezés nincs számottevő negatív hatással a hatékonyságra, viszont jelentős mértékben csökkenti a problémateret, ezért a tendencia alkalmazásának általános elfogadása felé mutat. Gazdagabb morfológiával rendelkező nyelvek esetén mindenképpen javasolt az alkalmazása, hiszen ekkor akár a tizedére is csökkenthető a szótár mérete.

A nyelvi dimenziócsökkentés mellett gyakran alkalmazzák a 2.3. szakaszban ismertetett jellemzőkiválasztó és -kinyerő módszerek valamelyikét is. Egyes szer-

zők igen kis jellemzőszám mellett is jó eredményeket kaptak például hierarchikus osztályozás esetén [99, 208]. Általában a konkrét feladat jellemzői — pl. korpuszméret, osztályozás típusa — és az alkalmazott osztályozó algoritmus határozza meg, hogy melyik redukciós technikát érdemes alkalmazni. Kis tanítókorpusz esetén nem érdemes agresszív dimenziócsökkentést alkalmazni, mivel a szótár mérete eleve nem nagy, ezért az algoritmusok futtatásánál ez nem jelent korlátot, ugyanakkor fennáll annak a veszélye, hogy a fürdővízzel a gyereket is kiöntjük, azaz információtartalommal bíró értékes jellemzőket is eliminálunk.

A fejezet további részében az alábbi jelöléseket fogjuk használni. A c_j kategóriába tartozó tanítódokumentumokat Pos_j jelöli: $\mathrm{Pos}_j = \{d_i \in D_{\mathrm{Train}} | \tilde{\Phi}(d_i, c_j) = 1\}$. A Pos_j halmaz komplementerét, azaz a c_j osztályba nem tartozó dokumentumokat Neg_j-vel jelöljük: $\mathrm{Neg}_j = \{d_i \in D_{\mathrm{Train}} | \tilde{\Phi}(d_i, c_j) = 0\}$. Ha minden tanítódokumentum pontosan egy kategóriába tartozik, akkor $\sum_{j=1}^{|C|} |\mathrm{Pos}_j| = |D_{\mathrm{Train}}|$.

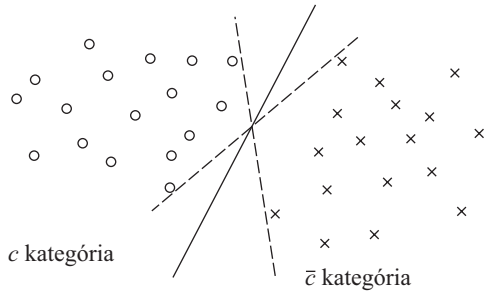
5.4. Osztályozó algoritmusok

Azokat a bináris osztályozókat, amelyek az osztályozást az M dimenziós vektortér $M-1$ dimenziós *szeparáló* vagy *döntési hipersík*kal való kettéosztásával végzik el *lineáris osztályozó*nak nevezzük.[6] Egy d dokumentum c kategóriához való tartozását aszerint döntjük el, hogy a vektora a döntési hipersík melyik oldalára esik (ld. 5.1. ábra). A legegyszerűbb kétdimenziós esetben a hipersík egy $w_1x_1 + w_2x_2 = b$ egyenes, ahol w_1, w_2 és b rendre a c kategóriát jellemző súly- és küszöbparaméterek. Ha a két jellemzővel reprezentált $\mathbf{d} = \langle d_1, d_2 \rangle$ vektorra $w_1d_1 + w_2d_2 > b$, akkor $\mathbf{d}$ a c kategóriához tartozik, ha $w_1d_1 + w_2d_2 \leq b$ akkor nem. Magasabb dimenziószám esetén a hipersíkot a $\mathbf{w}^T\mathbf{d} = b$ egyenlet definiálja.

A lineáris osztályozók feladata tehát a döntési hipersík meghatározása, amit általában végtelen sok lehetőség közül kell kiválasztani. Az optimális szeparáló hipersík meghatározása korántsem egyszerű feladat. A lineáris osztályozók széles skálán mozognak atekintetben, hogy mennyire képesek jó általánosító képességű hipersíkot meghatározni (ld. még az 5.5.1. pontot). Az osztályozócsalád tagjai közül a lineáris SVM, a regularizált lineáris regresszió, illetve a HITEC osztályozók a leghatékonyabb algoritmusok közé tartoznak.

Elsőként azokat a lineáris osztályozókat tárgyaljuk, amelyek a döntési hipersíkot közvetlenül a c kategóriát jellemző súlyvektorként állítják elő.

[6] Általánosan a lineáris osztályozók nem csak bináris feladatra alkamazhatók, hanem a probléma dekomponálásával tetszőleges többcímkés esetre (ld. még az 5.1.1. pontot).



5.1. ábra. Lineáris osztályozó két dimenzióban végtelen sok szeparáló hipersíkkal

5.4.1. Rocchio-osztályozó

A lineáris osztályozók építésének egyik módja a c_j kategóriát reprezentáló $\mathbf{c}_j = \langle w_{j1}, \ldots, w_{jM} \rangle$ *kategóriavektor*nak a dokumentumtér elemeként való meghatározása. Ez utóbbi azt jelenti, hogy a $\mathbf{c}_j$ vektor értékei a megfelelő jellemzőkhöz (szavakhoz) tartozó súlyokat reprezentálják, ezért a $\mathbf{c}_j$ vektor felfogható úgy is, mint a c_j kategóriába tartozó dokumentumok *prototípus*a, amit *kategóriaprofil*nak is neveznek. Ennek a megközelítésnek nagy előnye az, hogy a kategóriának emberek számára is könnyen értelmezhető leírását adja.

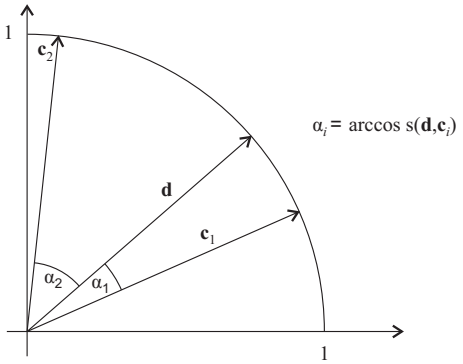
A dokumentum és a kategória hasonlósága ekkor egyszerűen meghatározható a két vektor *koszinusztávolság*aként[7] (5.2. ábra):

$$s(\mathbf{d}_i, \mathbf{c}_j) = \frac{\sum_{k=1}^{M} w_{ik} w_{jk}}{\sqrt{\sum_{k=1}^{M} w_{ik}^2} \sqrt{\sum_{k=1}^{M} w_{jk}^2}} \tag{5.3}$$

Ha a dokumentum- és kategóriavektorok súlyai pl. (2.10) szerint normalizáltak, azaz a vektorértékek a dokumentum hosszával, $\sqrt{\sum_{k=1}^{M} w_{ik}^2}$-tel le vannak osztva (ld. még (2.5) képleteket), akkor a vektorok hossza 1 lesz, és a nevező az (5.3) kifejezésből elhagyható. A $s(\mathbf{d}_i, \mathbf{c}_j)$ érték a dokumentum- és a kategóriavektorok által bezárt szög koszinuszát adja meg. Ez az IR-rendszerekben alkalmazott egyik standard hasonlósági mérték.

Az osztályozás folyamata ezután a következőképpen történik. Meghatározzuk a c_j és a komplementer $\overline{c}_j$ kategóriák profilját, és minden d_i dokumentumot ahhoz a kategóriához rendelünk, amelyikre az (5.3) hasonlósági mérték nagyobb

[7] Ezt a mennyiséget a két vektor *skaláris* vagy *belső szorzat*ának is nevezik, és $\langle \mathbf{d}_i, \mathbf{c}_j \rangle$-vel jelölik.



5.2. ábra. A dokumentum- és kategóriavektorok távolságának számítása koszinusztávolság alapján

lesz. Ekkor a két kategóriaprofiltól egyenlő távolságban lévő felezősík képezi a szeparáló hipersíkot.

A legegyszerűbb módszer a $\mathbf{c}_j$ kategóriavektor meghatározására a centroidok használata. A c_j osztály centroidját az osztályba tartozó dokumentumvektorok átlagaként definiáljuk:

$$\mathbf{c}_j := \mathbf{v}(c_j) = \frac{1}{|\mathrm{Pos}_j|} \sum_{\mathbf{d}_i \in \mathrm{Pos}_j} \mathbf{d}_i. \tag{5.4}$$

Az osztályozás lépései tehát az alábbiak lesznek (egycímkés eset):

- *Tanulás*: a tanítóadatok átlagaként határozzuk meg minden c_j kategória centroidját.
- *Tesztelés*: a d_i dokumentum osztályozásakor számítsuk ki az $s(\mathbf{d}_i, \mathbf{c}_j)$ hasonlósági mértéket az összes c_j kategóriára, és válasszuk ki a leghasonlóbb kategóriát:

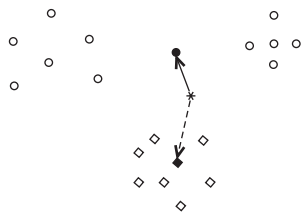
$$c = \arg\max_{c_j} s(\mathbf{d}_i, \mathbf{c}_j). \tag{5.5}$$

  A hasonlóság mérésére az (5.3) formulát vagy más távolságmértéket használhatunk.

A centroid alapú lineáris osztályozót Rocchio-osztályozónak nevezik, mivel működési elve megegyezik az IR-rendszerekben relevancia-visszacsatolásra (relevance feedback) alkalmazott Rocchio-algoritmuséval [156]. Az algoritmust szövegosztályozásra elsőként a [89] munkában adaptálták, és azóta is számos tanulmány foglalkozott vele (ld. [169]).

A módszer előnye a kicsiny számításigény, ezért a tanulás nagyon gyors. A sűrű kategóriavektorok (ld. a 151. oldalt) és a ritka dokumentumvektorok hasonlósági mértékének számítása tanítódokumentumok esetén $O(|D_{\text{Train}}|L_d + |C|M)$ nagyságrendű. Itt az első tag a dokumentumhalmaz előfeldolgozását és a kategóriaprofilok előállítását jelenti — ahol L_d a tanítódokumentumok átlagos hossza —, a második tag pedig a centroidok hossznormalizálása. Azaz a tanulás számításigénye lineáris a tanítókorpusz méretében. A tesztelés $O(L_t + |C|l_t)$ nagyságrendű, ahol L_t a tesztdokumentumok átlagos hossza, l_t pedig a méretük ritka vektorként, azaz a szótárban szereplő nemnulla jellemzőinek/szavainak száma.

A módszer hátránya, hogy heterogén tartalmú kategóriák esetén — pl. ha a *Sport* kategóriában teniszről és vitorlázásról szóló tanítódokumentumok is vannak — előfordulhat, hogy az osztályozó a dokumentumok legtöbbjét helytelenül sorolja be, mert a dokumentumok centroidja kívül esik az osztályon (5.3. ábra). Ez a jelenség abból adódik, hogy a dokumentumteret egyetlen hipersíkkal felosztó lineáris osztályozók érzékenyek a többértelmű, a reprezentációban több maggal rendelkező osztályokra, az ilyen osztályok szeparálására nem képesek.



5.3. ábra. A körök a *Sport*, a rombuszok a *Szabadidő* osztály tanítódokumentumait reprezentálják; a csillaggal jelölt dokumentumot a rombusz osztály helyett a kör osztályba sorolja az osztályozó, mivel ennek centroidjához van közelebb

A Rocchio-algoritmus alapmódszerének több módosított változata van. A negatív tanítóadatokat is figyelembe véve a kategóriaprofil súlyértékeit a

$$w_{jk} = \alpha \cdot \sum_{d_i \in \text{Pos}_j} \frac{w_{ik}}{|\,\text{Pos}_j\,|} - \beta \cdot \sum_{d_i \in \text{Neg}_j} \frac{w_{ik}}{|\,\text{Neg}_j\,|} \tag{5.6}$$

képlettel adhatjuk meg, ahol α és β a pozitív, ill. negatív tanítóadatok relatív fontosságának beállítására szolgáló paraméterek. A centroid alapú Rocchio-osztályozót $\alpha = 1$ és $\beta = 0$ esetén kapjuk vissza. A módszer azokat a dokumentumokat jutalmazza, amelyek közel vannak a centroidhoz, és távol esnek a negatív taní-

tóadatoktól. Általában a negatív tanítóadatokat kisebb súllyal veszik figyelembe, tipikus értékek ekkor az $\alpha = 16$ és $\beta = 4$ [41, 91, 169].

Lényeges hatékonysági javulás érhető el akkor [165, 209], ha a negatív példák közül csak azokat vesszük figyelembe, melyek közel vannak a pozitív tanítópéldákhoz, ezek ugyanis azok, amelyeket a legnehezebb megkülönböztetni a pozitív tanítóadatoktól, tehát a döntési hipersík pontos meghatározásában legjelentősebb a szerepük. Ekkor Npos_j-vel jelölve a c_j-hez közeli negatív tanítódokumentumokat az (5.6) képlet így módosul:

$$w_{jk} = \alpha \cdot \sum_{d_i \in \text{Pos}_j} \frac{w_{ik}}{|\,\text{Pos}_j\,|} - \beta \cdot \sum_{d_i \in \text{Npos}_j} \frac{w_{ik}}{|\,\text{Npos}_j\,|}. \tag{5.7}$$

Egyszerű osztályozás esetén Npos_j elemeit a negatív tanítóadatok $\mathbf{v}(c_j)$ centroidtól való távolsága alapján határozhatjuk meg. Hierarchikus osztályozás esetén pedig a testvér-kategóriák pozitív tanítóadatait használhatjuk erre a célra.

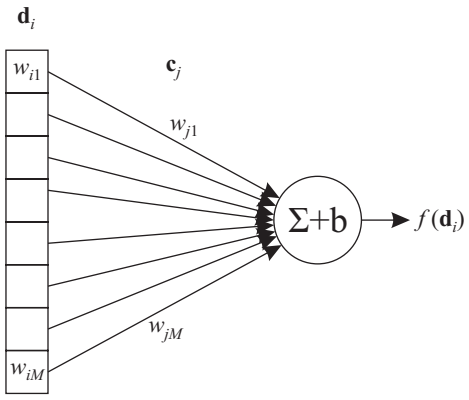
5.4.2. Neurális hálózat alapú módszerek

A Rocchio-osztályozó *kötegelt tanulás*t (batch learning) végez, azaz az összes tanítóadatot egyszerre használja fel. Ezzel szemben a következőkben ismertetésre kerülő lineáris osztályozók *fokozatos* (másnéven *inkrementális* vagy *online*; incremental learning) *tanulás* végrehajtására is képesek, amikor is a tanítóadatokat nem egyszerre dolgozza fel az osztályozó. Ekkor az első néhány tanítóadat alapján felépített kezdeti osztályozó az újabb tanítódokumentumok vizsgálata során változhat. Ez az adaptivitás előnyös lehet, ha a tanulás kezdetén nem áll rendelkezésre az összes tanítóadat, és így a kategóriák szemantikája módosulhat.

Lineáris osztályozás esetén a neurális hálózat csak két rétegből épül fel. A bemeneti réteg neuronjai a jellemzőknek (szavaknak) felelnek meg, a kimeneti réteg neuronjai pedig a kategóriákat reprezentálják — bináris esetben csak c_j-t és komplementerét. A rétegek közötti súlyok vektora adja ki a c_j kategóriaprofilját, amely a bemenet és a kimenet közötti függőségi relációt jellemzi. Osztályozáskor a bemeneten betápláljuk dokumentumvektort, és a hálózat kimenete határozza meg az osztályozási döntést. A hálózat tanítása *hibavezérelt* visszacsatolásos módszerrel történik: ha egy szöveget rosszul kategorizál a hálózat, akkor a hibát visszacsatolva módosítjuk a súlyok értékét a hiba csökkentése érdekében.

A legegyszerűbb bináris osztályozást végző neurális hálózat a *perceptron* [157]. Szövegosztályozó perceptronoknál a dokumentumok bináris súlyozását használják (ld. (2.2)) [48, 168, 209]. A perceptron (5.4. ábra) aktivációs függvénye az

$$f(\mathbf{d}_i) = \langle \mathbf{d}_i, \mathbf{c}_j \rangle + b. \tag{5.8}$$



5.4. ábra. Rejtett réteg nélküli perceptron architektúrája

Az osztályozás kimenete az f előjelétől függ: ha pozitív, akkor c_j-be sorolja a d_i dokumentumot, ellenkező esetben pedig $\overline{c}_j$-be. A b küszöbparaméter (másképpen *eltolás* (bias)) a szeparáló hipersík pozícióját határozza meg, de az irányára nincs hatása, és értéke nem függ a bemenettől. b értéke pozitív, hiszen azt határozza meg, hogy a pozitív vektorok skalárszorzatának milyen minimális értékénél rendeljük a d_i dokumentumot c_j-be. Az egyszerűbb tárgyalás érdekében most a negatív tanítóadatokra legyen $\tilde{\Phi}(d_i, c_j) = -1$. Tehát a cél egy olyan szeparáló hipersík megtalálása, amire: $f(\mathbf{d}_i) > 0$, ha $d_i \in \mathrm{Pos}_j$, azaz $\tilde{\Phi}(d_i, c_j) = 1$, és $f(\mathbf{d}_i) \leq 0$, ha $d_i \in \mathrm{Neg}_j$, azaz $\tilde{\Phi}(d_i, c_j) = -1$, másképpen $f(\mathbf{d}_i) \cdot \tilde{\Phi}(d_i, c_j) > 0$ minden d_i-re. Ha létezik ilyen hipersík, akkor a tanítóhalmaz c_j szerint *lineárisan szeparálható*, és a $\mathbf{c}_j$ súlyvektor lesz a hipersík normálvektora.

Jelöljük A-val a sikeresen, B-vel a sikertelenül osztályozott d_i-k halmazát:

$$A = \{d_i | f(\mathbf{d}_i) \cdot \tilde{\Phi}(d_i, c_j) > 0\}; \quad B = \{d_i | f(\mathbf{d}_i) \cdot \tilde{\Phi}(d_i, c_j) \leq 0\}.$$

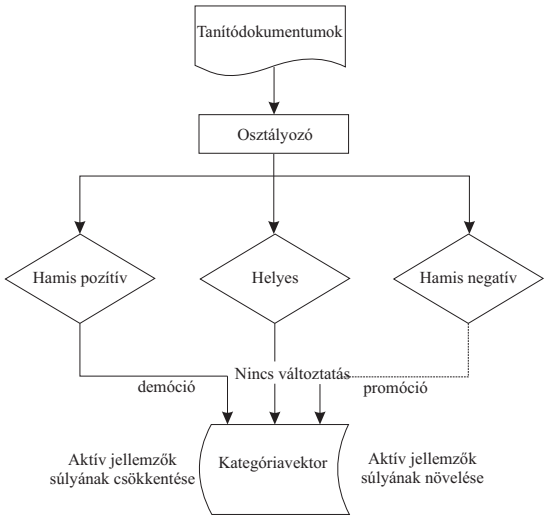
A perceptron készítésekor a súlyok kezdeti értékét azonosra állítjuk. Az osztályozó *hibavezérelt tanulás*t (mistake driven learning) végez (5.5. ábra). A tanítódokumentumot először osztályozzuk az (5.8) függvény kiszámításával. Ha ez sikeres, akkor semmit nem módosítunk rajta, ellenkező esetben az alábbi, *additív súlybeállítási stratégiát* követjük a súlyok módosításánál [25]. Ha $d_i \in \mathrm{Pos}_j$, akkor a kategóriaprofilnak a d_i „aktív” — a dokumentum előforduló: $w_{ik} = 1$ — szavaihoz tartozó súlyát a $\gamma > 0$ *tanulási rátá*val növeljük; ellenkező esetben pedig

γ-val csökkentjük:

$$w_{jk} \leftarrow \begin{cases} w_{jk} + \tilde{\Phi}(d_i, c_j)\gamma, & \text{ha } d_i \in B \text{ és } w_{ik} = 1; \\ w_{jk}, & \text{különben.} \end{cases} \tag{5.9}$$

Az iterációt a terminálási feltétel teljesüléséig végezzük, amely az iterációk számára vagy a tanító-, ill. validáló korpuszon mért együttes hiba változására szab korlátokat. A hibát a rosszul osztályozott elemek szeparálási hipersíktól mért távolságával határozzuk meg:

$$e = \sum_{d_i \in B} f(\mathbf{d}_i).$$



5.5. ábra. Hibavezérelt tanulási séma

A perceptronra vonatkozó konvergenciatétel szerint, ha a tanítóhalmaz lineárisan szeparálható, akkor a perceptron véges számú lépésben talál szeparáló hipersíkot. A módszer értelemszerűen kiterjeszthető az általános egycímkés osztályozási feladatra.

A tanulás végén a kicsiny súlyú szavak szótárból való elhagyásával lokális dimenzióredukciót lehet végrehajtani, mivel ezek negatív példákat jelentenek a kategóriára vonatkozóan [48].

Azonos előrecsatolt hibajavító tanulási séma mellett *multiplikatív súlybeállítás*t alkalmaznak a különböző verziójú WINNOW-algoritmusok [48, 120]. A *pozitív* WINNOW félreosztályozás esetén az aktív szavak súlyát a kategóriaprofilban

rendre a $\gamma_1 > 1$ *promóciós*, ill. $0 < \gamma_2 < 1$ *demóciós konstans*okkal való szorzással növeli, ill. csökkenti:

$$w_{jk} \leftarrow \begin{cases} w_{jk} \cdot \gamma_1, & \text{ha } d_i \in B \cap \text{Pos}_j \text{ és } w_{ik} = 1; \\ w_{jk} \cdot \gamma_2, & \text{ha } d_i \in B \cap \text{Neg}_j \text{ és } w_{ik} = 1; \\ w_{jk}, & \text{különben.} \end{cases} \tag{5.10}$$

Ha a $\mathbf{c}_j$ vektor súlyainak kezdeti értékét azonosan b/l_d-re állítják, ahol l_d a tanítódokumentumokban lévő aktív szavak átlagos számát jelöli, akkor a skalárszorzat kezdeti értékei b körül lesznek.

A *kiegyensúlyozott* WINNOW minden jellemzőhöz két súlyt rendel, w_{jk}^+-t és w_{jk}^--t, amiket rendre a pozitív, ill. negatív példák szabályoznak. Az (5.8) aktiváló függvényben a $\mathbf{c}_j$ súlyait a két súlyvektor különbségeként kapjuk meg, ezért a súly értéke negatív is lehet:

$$w_{jk} = w_{jk}^+ - w_{jk}^-.$$

A kezdeti súlyok beállítására w_{jk}^+ esetén a $2b/l_d$, w_{jk}^- esetén a b/l_d pozitív konstansokat használják [48].

A tanulási eljárásnál itt is csak az aktív jellemzőkhöz tartozó súlyokat módosítjuk. Ha az eljárás pozitív tanítóadaton hibázik, akkor az aktív jellemzőkhöz tartozó w_{jk}^+ súlyokat a promóciós, a w_{jk}^- súlyokat a demóciós faktorral szorozzuk. Negatív tanítóadaton való tévesztés esetén éppen fordítva: a w_{jk}^+ súlyokat a demóciós, a w_{jk}^- súlyokat a promóciós faktorral szorozzuk. Az első esetben az egyesített w_{jk} súlyérték nő, míg a másodikban csökken. Az alábbi képletben az aktív súlyok módosításai szerepelnek, mindig feltesszük, hogy $w_{ik} = 1$.

$$\begin{aligned} w_{jk}^+ &\leftarrow \begin{cases} w_{jk}^+ \cdot \gamma_1, & \text{ha } d_i \in B \cap \text{Pos}_j, \\ w_{jk}^+ \cdot \gamma_2, & \text{ha } d_i \in B \cap \text{Neg}_j; \end{cases} \\ w_{jk}^- &\leftarrow \begin{cases} w_{jk}^- \cdot \gamma_2, & \text{ha } d_i \in B \cap \text{Pos}_j, \\ w_{jk}^- \cdot \gamma_1, & \text{ha } d_i \in B \cap \text{Neg}_j. \end{cases} \end{aligned} \tag{5.11}$$

A neurális hálózat alapú módszerek tanulásának időigénye erősen függ az elvégzendő iterációk számától: $O(|D_{\text{Train}}|L_d + IWl_d)$, ahol I az iterációk, W pedig az átlagos súlymódosítások szám iterációnként. Mivel csak az aktív jellemzőket vesszük figyelembe a súlyállításnál, ezért a szorzótényező l_d és nem L_d. A tesztelési idő ebben az esetben is $O(L_t + |C|l_t)$ nagyságrendű.

A WINNOW-eljárásoknak léteznek olyan verziói is, amelyek

- nem csak az aktív súlyokat módosítják;
- nem csak hibázás esetén módosítják a súlyokat.

Megjegyezzük, hogy a hierarchikus osztályozóknál tárgyalt HITEC-algoritmus is az itt ismertetett fokozatos tanulást végző osztályozók családjába tartozik.

A WINNOW-algoritmusok, csakúgy mint a perceptron, véges számú lépésben megtalálnak egy szeparáló hipersíkot, ha ilyen létezik. Sőt akkor is elég hatékony megoldást adnak, ha lineárisan nem szeparálható a feladat, ami a módszer két tulajdonságával indokolható. Egyrészt a hibaérték csak az aktív jellemzők számának lineáris függvénye, az összes jellemzők számát tekintve logaritmikus összefüggés van. Másrészt a hibavezérelt tanulási séma sajátos tulajdonsága, hogy érzékenyebb a jellemzők közti látens összefüggésekre, mint a kizárólag a jellemzők előfordulásán alapuló módszerek [121]. Ezért a hibavezérelt fokozatos tanulást végző osztályozók általában igen hatékonyak, kevéssé érzékenyek a jelentéktelen jellemzőkre és a zajra, és az időben változó célfüggvényhez is jól adaptálódnak. Ez utóbbi tulajdonságuk miatt kimondottan alkalmasak az effajta igényt támasztó feladatok megoldására.

Az eddig ismertetett neurális hálózat alapú módszerek lineáris osztályozók, mivel a hálózat kimenete lineárisan függ a bemenettől. Egyszerűségük ellenére a leghatékonyabb eljárások közé tartoznak. Több munka is megvizsgálta a nemlineáris neurális hálózatok alkalmazását egy vagy több rejtett réteget illesztve a hálózatba. Ez a módosítás azonban az osztályozó hatékonyságára vonatkozóan semmilyen [168], vagy csak igen csekély [209] javulást eredményezett.

5.4.3. Valószínűség alapú osztályozás: a naiv Bayes-módszer

Valószínűségelméleti megközelítésben a Φ osztályozó megkonstruálásának feladatát a $P(c_j|\mathbf{d}_i)$ feltételes valószínűségi értékekre vonatkozó becslésként fogalmazhatjuk meg. Ez az érték megadja, hogy milyen valószínűséggel tartozik a d_i dokumentum a c_j kategóriába. A becslést a Bayes-tétel alapján végezzük, amely az alábbi összefüggést mondja ki feltételes valószínűségekre:

$$P(c_j|\mathbf{d}_i) = \frac{P(c_j)P(\mathbf{d}_i|c_j)}{P(\mathbf{d}_i)}. \tag{5.12}$$

A d_i dokumentumot — akár bináris, akár egycímkés osztályozás esetén — ahhoz a c kategóriához rendeljük, amelyikre a $P(c|\mathbf{d}_i)$ értéke maximális:

$$c_{\text{MAP}} = \arg\max_{j=1,\ldots,|C|} P(c_j|\mathbf{d}_i) = \arg\max_j \frac{P(c_j)P(\mathbf{d}_i|c_j)}{P(\mathbf{d}_i)} = \arg\max_j P(c_j)P(\mathbf{d}_i|c_j). \tag{5.13}$$

Itt az alsó index a becslés maximum aposzteriori voltára utal, vagyis arra, hogy a valószínűséget a rendelkezésre álló megfigyelések figyelembe vételével határoztuk meg. A $P(\mathbf{d}_i)$ érték a számlálóból elhagyható, mert ez minden osztály esetén ugyanaz lesz. Nézzük tehát, milyen becslések adhatók az (5.13) egyenlet jobboldalán álló valószínűségekre!

$P(c_i)$ értéke — ami annak a valószínűsége, hogy egy véletlenszerűen kiválasztott dokumentum a c_j osztályba esik —, a c_j osztály tanítókorpuszban megfigyelt gyakoriságával becsülhető, vagyis a

$$\hat{P}(c_j) = |\,\mathrm{Pos}_j\,|/|D_{\mathrm{Train}}| = |\,\mathrm{Pos}_j\,|/N$$

aránnyal. A $P(\mathbf{d}_i|c_j)$ meghatározásánál a lehetséges $\mathbf{d}_i \in \mathbb{R}^M$ vektorok nagy száma gondot jelent, hiszen összesen $2^M|C|$ paramétert kéne megbízhatóan megbecsülni, ami rendkívül nagyméretű tanítóhalmazt igényelne. Ahhoz, hogy a paraméterek száma kezelhető legyen feltesszük, hogy ha az osztály adott, a jellemzők — azaz a szavak előfordulása a dokumentumokban — függetlenek egymástól. Ezt a feltételes függetlenségi feltevést *naiv Bayes-feltételezés*nek nevezzük, ahol a jelző arra utal, hogy a gyakorlatban a feltevés sérül. Ekkor

$$P(\mathbf{d}_i|c_j) = \prod_{k=1}^{M} P(w_{ik}|c_j), \tag{5.14}$$

azaz $M|C|$ számú paramétert kell csak megbecsülni. Szövegosztályozás esetén a jellemzők általában szavakat reprezentálnak, ezért a $P(w_{ik}|c_j)$ értékét a t_k szó c_j kategóriában való előfordulási valószínűségeként adhatjuk meg:

$$\hat{P}(w_{ik}|c_j) = \hat{P}(w_k|c_j) = \frac{N_{kj}}{\sum_{\ell=1}^{M} N_{\ell j}}, \tag{5.15}$$

ahol N_{kj} a t_k szó előfordulásainak száma a Pos_j dokumentumokban. A tanítóadatok ritkasága miatt azonban (5.15)-ben N_{kj} értéke gyakran 0, így az (5.14) szorzat is zérus lesz. Ezt elkerülendő *Laplace-simítás*t alkalmazunk:[8]

$$\hat{P}(w_k|c_j) = \frac{1 + N_{kj}}{M + \sum_{\ell=1}^{M} N_{\ell j}}. \tag{5.16}$$

Foglaljuk össze a naiv Bayes-osztályozó működését!

[8] Ez interpretálható úgy, mint egy egyenletes prior valószínűségi eloszlás — minden szó egyszer szerepel minden osztályban — kombinálása a tanítóadatokon megfigyelt előfordulási értékekkel.

- *Tanulás*: becslést adunk a $P(c_j)$ és $P(w_k|c_j)$ valószínűségekre, ehhez minden c_j-re meghatározzuk
  - $|\mathrm{Pos}_j|$-t, a c_j osztályba tartozó tanítódokumentumok számát,
  - $\hat{P}(c_j) = |\mathrm{Pos}_j|/N$-t, a c_j osztályba eső dokumentumok arányát,
  - N_{kj}-t, a t_k szó előfordulását a Pos_j dokumentumhalmazban (ez pl. a Pos_j-beli dokumentumokból alkotott szuperdokumentum elemzésével hajtható egyszerűen végre),
  - a $\hat{P}(w_k|c_j) = \frac{N_{kj}}{\sum_{\ell=1}^{M} N_{\ell j}}$ valószínűséget.
- *Tesztelés*: $\mathbf{d}_i$ tesztdokumentumra meghatározzuk a legvalószínűbb osztályt azon szóelőfordulásai alapján, amelyek a tanítókorpuszban is szerepeltek:

$$c_{\mathrm{NB}} = \arg\max_{c_j \in C} P(c_j) \prod_{k:w_{ik}>0} P(w_k|c_j). \tag{5.17}$$

A tanulóalgoritmus bonyolultsága $O(|D_{\mathrm{Train}}|L_d + |C|M)$, ahol az első tag az átlagosan L_d hosszú dokumentumok előfeldolgozásából adódik, a második tag pedig a $|C| + |C|M$ számú valószínűségi becslés meghatározásából. Egy dokumentum tesztelésének időigénye $O(|C|L_t)$ nagyságrendű. Mivel a tanulásnál az első tag dominál, $|C|M < |D_{\mathrm{Train}}|L_d$, ezért a feldolgozás lineáris a dokumentumok méretében. E kedvező tulajdonsága miatt a naiv Bayes-algoritmus az egyik leggyakrabban alkalmazott osztályozó.

Mivel az (5.14) és (5.17) kifejezésekben sok nagyon kicsiny valószínűségi értéket szorzunk össze, ez alulcsordulást eredményezhet. Ezért a gyakorlatban érdemes a logaritmikus térben számolni, és a valószínűségek összeszorzása helyett azok logaritmusának összeadását végrehajtani:

$$c_{\mathrm{NB}} = \arg\max_{c_j \in C} \log P(c_j) + \sum_{k:w_{ik}>0} \log P(w_k|c_j), \tag{5.18}$$

ami a logaritmusfüggvény monotitása miatt nem változtat a legnagyobb érték kiválasztásának eredményén.

Az (5.18) kifejezésből könnyen megmutatható, hogy a naiv Bayes lineáris osztályozó a logaritmikus problématérben. Bináris feladat esetén ugyanis a

$$\log \frac{P(c|\mathbf{d})}{P(\overline{c}|\mathbf{d})} = \log \frac{P(c)}{P(\overline{c})} + \sum_{k:w_{ik}>0} \log \frac{P(w_k|c)}{P(w_k|\overline{c})}, \tag{5.19}$$

ahol akkor döntünk c mellett, ha (5.19) nagyobb zérusnál. Ez pedig egy $M-1$ dimenziós hipersíkot definiál, ahol a súlyok értéke $\log\frac{P(w_k|c)}{P(w_k|\overline{c})}$, az eltolás pedig $-\log\frac{P(c)}{P(\overline{c})}$ lesz.

A dokumentumok reprezentálására általában bináris súlyozást alkalmaznak. Ha a dokumentumok nagyon hosszúak, akkor a kategóriához való tartozástól függetlenül sok w_{ik} érték lesz pozitív, ezért szükség lehet hossznormalizálásra, pl. tf-idf súlyozással [14, 38, 130], ami viszont a valószínűségi modell olyan átalakítását vonja maga után, amely a feltételes függetlenségi feltevés alkalmazásának igazolhatóságát még inkább megkérdőjelezi [116, 169].

A fentiekben ismertetett algoritmus az ún. *multinomiális* naiv Bayes-módszer, ahol a $P(w_k|c_j)$ valószínűségek becslése a szózsákmodellen alapult, azaz implicite alkalmazta a *pozíciókra vonatkozó feltételes függetlenség*et. A $P(w_k|c_j)$ értékek becslésénél a c_j kategóriára lokális *cf* értékeket számoljuk ki. Egy másik lehetőség ennek becslésére a lokális *df* értékek használata, vagyis az alapján számolni $P(w_k|c_j)$-t, hogy hány c_j-beli dokumentumban fordult elő a t_k szó. Ez a *binomiális* naiv Bayes-modell, amely néhány kedvezőtlen tulajdonsága miatt — főleg kisméretű dokumentumokra és kicsiny szótárméret mellett alkalmazható — nem terjedt el annyira [116, 125].

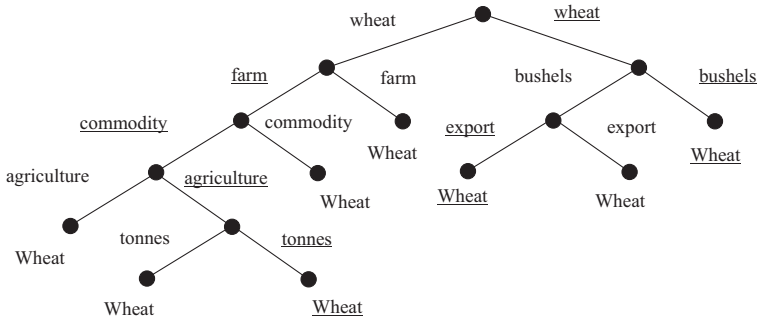
Érdekes módon annak ellenére, hogy a szavak független előfordulására vonatkozó feltételezés általában nem igaz, a módszer igen jó eredményt ad, amit elméleti eredmények is alátámasztanak [53]. Sőt ha bonyolultabb, s ezáltal nagyobb számításigényű, függőségi viszonyokat legalább szópárokra figyelembe vevő valószínűségi modellt használunk [99], akkor sem javul jelentékenyen a hatékonyság. A naiv Bayes-módszer hatékonysága elsősorban azon múlik, hogy bár a valószínűségi becslések gyakran pontatlanok, azok nagyságrendje és egymáshoz való viszonya — melyik osztály valószínűbb a másiknál — már helyes. A „győztes” osztály valószínűsége általában sokkal nagyobb a többinél, normalizálás után rendszerint 1-hez közeli értéket ad. A kedvező bonyolultsági értékek mellett egyszerűsége és robosztussága miatt alkalmazzák előszeretettel, főleg ha nagyméretű tanítókorpusz áll rendelkezésre [125].

5.4.4. Döntési fa alapú szövegosztályozók

A numerikus és valószínűségi modellt alkalmazó osztályozótípusokat, pl. a naiv Bayes-, ill. neurális hálózat alapú osztályozókat, gyakran éri az a kritika, hogy emberek számára nehezen interpretálható döntési szabályok alapján működnek. Ezt a problémát küszöböli ki a *szimbolikus tanuló*k családja, ahova az itt részletesebben ismertetésre kerülő *döntési fa alapú osztályozó*kon (decision tree classifier) kívül pl. a *döntési szabály alapú osztályozó*k (decision rule classifier) is tartoznak.

Döntési fán alapuló szövegosztályozó egy olyan fa, amelyben a közbenső csomópontok szavakat reprezentálnak, a csomópontokból kiinduló ágak ellenőrzési

feltételeket írnak elő az adott szóra vonatkozóan a tesztdokumentumra, végül pedig a levelek kategóriákkal vannak címkézve (ld. 5.6. ábra). A **d** tesztdokumentum osztályozása a döntési fa csomópontjaihoz tartozó szavak **d**-beli súlyának vizsgálata alapján rekurzív módon történik, a dokumentumhoz végül a levél kategóriacímkéjét rendeljük hozzá. A döntési fa alapú szövegosztályozók általában bináris reprezentációt használnak, így a döntési fa is bináris.



5.6. ábra. Döntési fa részlete a Reuters-21578 (137. oldal) adatbázis *Wheat* kategóriája szerinti osztályozáshoz. Az aláhúzott szavak, illetve kategórianevek a szó hiányát, illetve a komplementer kategóriát jelölik

A legtöbb szövegosztályozó standard döntési fa tanulóalgoritmust használ, mint az ID3 és továbbfejlesztései: a C4.5 és a C5, ill. a CART vagy a CHAID. Néhány példa a konkrét alkalmazásukra megtalálható a [41, 118] munkákban.

Általánosságban a c_j kategóriához tartozó döntési fa megtanulása az *„oszd meg és uralkodj" stratégia* két lépéséből áll: (1) annak ellenőrzése, hogy a csomóponthoz tartozó minden tanítódokumentumnak ugyanaz-e a címkéje (c_j vagy $\overline{c}_j$); (2) ha nem, akkor azon t_k szó kiválasztása, amely alapján elvégezzük a tanítódokumentumok particionálását úgy, hogy az egyes partíciókban a t_k-hoz tartozó d_{ik} értékekek megegyezőek legyenek. A t_k kiválasztásánál törekedni kell arra, hogy a kapott partíciók az adott osztályra vonatkozóan minél homogénebbek legyenek. Az optimális t_k kiválasztását az *információnyereség* (ld. az 57. oldalon), vagy a *Gini-index* alapján határozzák meg mohó kereséssel, azaz előretekintés nélkül, csak az adott csomópont szerinti legjobb particionálást veszik figyelembe. A módszer rekurzíve addig folytatódik, amíg az egyes levelekben csak azonos kategóriájú tanítóadatok lesznek.[9] Ezzel a kategóriával címkézzük a leveleket.

[9] Alternatív terminálási feltételek: már az összes szót felhasználtuk particionálásra (szövegosztályozásnál nem szokott előfordulni), ill. elértük a döntési fa megengedett maximális mélységét.

Ha a döntési fa nagyon sok csomópontot tartalmaz — szövegosztályozásnál nem ritka, hogy több százezer jellemző van —, akkor megnő a túltanulás veszélye, mivel a döntési fa egyes részei túlságosan illeszkednek a tanítóhalmazra. A túltanulást a döntési fa *metszés*ével (pruning) lehet megakadályozni. Két fajta metszést szoktak alkalmazni: elő- és utómetszést. Az előbbi esetén a döntési fa konstruálása során metszési feltételeket állítunk fel, amelyek teljesülése esetén terminálják az építést. Az utóbbi esetben a teljes fát megalkotjuk, majd annak részfáit a várható hibaarány függvényében levágjuk [2]. Előmetszés esetén előnyös, hogy nem kell a teljes fát felépíteni, ugyanakkor nehéz meghatározni a faépítésre vonatkozó leállási feltételt, és így könnyen pontatlan lehet az osztályozó. Ezért, bár az utómetszés időigényét tekintve kevésbé hatékony, egyes döntési fa programcsomagok (C4.5, CART) nagyobb pontossága miatt mégis ezt alkalmazzák, míg a CHAID előmetszést használ.

A döntési fa szintén lineáris osztályozó, egy csomópont egy adott dimenzió szerinti szeparátort határoz meg. Így összességében a bináris kategóriavektorban csak a döntési fa csomópontjaihoz rendelt jellemzőkre vonatkozó súlyértékek szerepelnek.

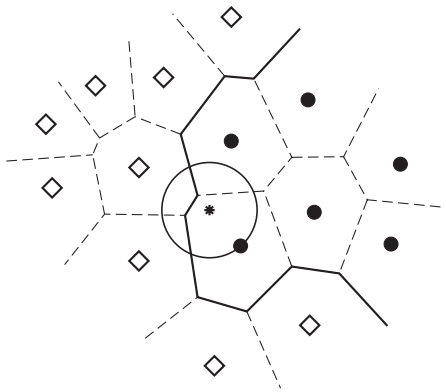
A döntési fák nem érzékenyek a tanítókorpusz méretére, mindamellett akkor működnek igazán hatékonyan, ha van a feladatnak néhány nagy diszkriminációs képességgel rendelkező jellemzője, aminek alapján a döntés meghozható. Ezt a tulajdonságát tekintve ellentéte a naiv Bayes-osztályozónak, amelyik sok, hasonló fontosságú jellemző esetén működik hatékonyan. A döntési fákat részletesen tárgyalja pl. a [133, 3. fejezet] és a [2, 5. fejezet].

5.4.5. Legközelebbi szomszédokon alapuló osztályozó (*k*-NN)

A lineáris osztályozókkal ellentétben a *legközelebbi szomszédokon alapuló osztályozó* (nearest neighbor, NN) *lokálisan*, az adott tesztdokumentumhoz hasonló tanítódokumentumok címkéje alapján osztályoz. E tulajdonsága miatt a *minta alapú osztályozó*k (example-based classifier) közé sorolják. Ezt az osztályozócsaládot *lusta tanuló*knak (lazy learner) is nevezik, mivel az osztályozási modellt lokálisan és csak akkor építik fel, amikor a tesztdokumentumot osztályozni kell. A módszer legfontosabb paramétere, hogy a döntés meghozatalánál hány hasonló tanítódokumentumot vesz figyelembe. Ezt a paramétert k-val jelöljük, az osztályozót általánosan pedig k-NN osztályozónak nevezzük.

A legegyszerűbb változat az 1-NN osztályozó, amely a tesztdokumentum legközelebbi szomszédjának címkéje alapján osztályoz. Ekkor a döntési felületet a tanítódokumentumok által a dokumentumtérben generált Voronoi-tesszelláció

adja meg (5.7. ábra), ahol a felosztás celláira igaz az, hogy belső pontjaik a reprezentáns ponthoz vannak a legközelebb (ld. [125]). Általában azonban az 1-NN osztályozás igen pontatlan, ezért nagyobb k értéket alkalmaznak.



5.7. ábra. 1-NN osztályozó működése; a vastagabb elválasztó vonal a döntési felület

Az adott dokumentum címkéjét k legközelebbi szomszéd esetén többségi döntés alapján határozzák meg. Bináris feladatnál ez azt jelenti, hogy megvizsgáljuk d_i k szomszédjából hány tartozik a c_j kategóriába, és ha ezek aránya elég nagy (pl. több mint a szomszédok fele), akkor c_j-hez rendeljük, egyébként a komplementeréhez. Egycímkés esetben azt a kategóriát választjuk, amelyiknek a legtöbb példánya szerepel a k szomszéd között. Ha több ilyen kategória is van, akkor véletlenszerűen döntünk az egyik mellett. Annak érdekében, hogy ez az eset ritkán forduljon elő, k értékét páratlannak szokták választani. Valószínűségi modellt használva a $P(c_j|\mathbf{d}_i)$ értékét a c_j tanítóadatai arányaként határozhatjuk meg.

Legyen tehát a $\mathbf{d}_i$-hez legközelebbi k szomszédok halmaza $S_k(\mathbf{d}_i)$. Ekkor a döntést a

$$c_{\text{kNN}} = \arg\max_{c_j \in C} \sum_{d_\ell \in S_k(\mathbf{d}_i)} \tilde{\Phi}(d_\ell, c_j) \tag{5.20}$$

kifejezés maximalizálásával végezzük. Valószínűségi megközelítést használva a

$$\hat{P}(c_j|\mathbf{d}_i) = \frac{|\text{Pos}_j \cap S_k(\mathbf{d}_i)|}{|S_k(\mathbf{d}_i)|}$$

becslést végezzük el. A hasonlóság mérésére az (5.3) koszinusztávolságot szokták alkalmazni. A távolságot súlyozó tényezőként is figyelembe lehet venni a döntés

meghozatalánál [215], ekkor az (5.20) képlet így módosul:

$$c_{\text{kNN}} = \arg\max_{c_i \in C} \sum_{d_\ell \in S_k(\mathbf{d}_i)} s(\mathbf{d}_\ell, \mathbf{d}_i)\tilde{\Phi}(d_\ell, c_j). \tag{5.21}$$

A k paraméter beállítását tapasztalati úton végzik a *validációs adatok*on. Egyes vizsgálatok azt mutatták ki [216], hogy $30 \leq k \leq 45$ értékek adják a legjobb eredményt, más munkák [125] lényegesen kisebb, $k = 3$, ill. 5 értéket találták optimálisnak.

Foglaljuk össze a k-NN osztályozó működését!

- *Tanulás*: Dokumentumok előfeldolgozása és k értékének meghatározása.
- *Tesztelés*: $S_k(\mathbf{d}_i)$ halmaz létrehozása és a döntés meghozatala, ehhez szükséges
  - $\mathbf{d}_i$ és az összes tanítóadat hasonlóságának kiszámítása,
  - $S_k(\mathbf{d}_i)$ meghatározása,
  - minden c_j osztályra $|\text{Pos}_j \cap S_k(\mathbf{d}_i)|$ alapján $P(c_j|\mathbf{d}_i)$ értékének meghatározása.

Az algoritmus bonyolultági értékei jelentősen eltérnek az eddig látottaktól. Az előfeldolgozás $O(|D|L_d)$, azaz itt is arányos a tanítókorpusz méretével, azonban a tesztelés — bár független a kategóriák számától — ismét arányos a teljes tanítókorpusz méretével: $O(L_t + |D|l_t) = O(|D|l_t)$. Ez azt jelenti, hogy gyakorlatilag a teljes korpuszt fel kell dolgozni a döntés végrehajtásához.

Felmerül a kérdés, hogy nem lehet-e valahogy csökkenteni a futásidőben szükséges feldolgozási lépések számát. Két javítási lehetőséget említünk itt meg. Felismerve azt, hogy a k-NN osztályozó azonos működési elvet követ, mint a keresőmotorok, azaz a legrelevánsabb k dokumentumot keresik egy adott dokumentumhoz,[10] [215]-ben az invertált index felhasználását javasolják (ld. a 8.4. szakaszt). Ezzel a keresés azon tanítódokumentumokra szűkíthető, amelyeknek van közös szavuk a tesztdokumentummal. Csakhogy a hosszabb dokumentumokat tartalmazó osztályozási feladatoknál ez szinte az összes dokumentumra igaz, hacsak valamilyen agresszív dimenzióredukciós előfeldolgozást nem alkalmazunk. Ha azonban a dokumentumok rövidek, és csak kevés jellemzőt hagyunk meg, akkor akár 100-adára is csökkenhet a tesztelési idő. Egy másik alternatíva (ld. pl. [109]), hogy az előfeldolgozás során, amelynek időigénye nem kritikus, elvégezzük a tanítódokumentumok *csoportosítás*át (ld. a 6. fejezetet), majd meghatározzuk a

[10] Ténylegesen csak a keresődokumentum hosszában van különbség a két probléma között.

csoportprofilokat pl. a Rocchio-algoritmussal. Ezután a tesztelésnél a csoportprofilokhoz hasonlítjuk először a tesztdokumentumot, majd a leghasonlóbb néhány (2–5) csoport dokumentumai alapján hozunk döntést.

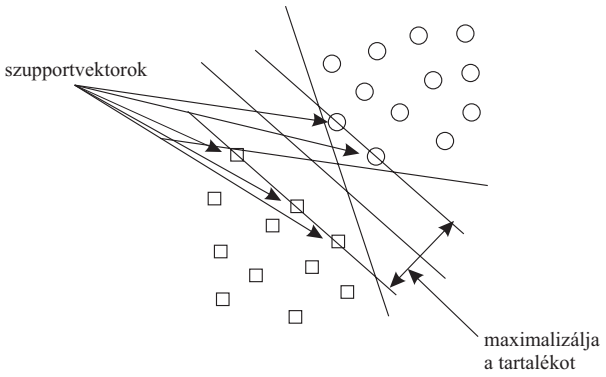
A *k*-NN osztályozó *nemlineáris*, ezért a Rocchio-eljárásnál ismertetett problémák nem jelentkeznek, a vektortér tetszőleges formát öltő osztályához képes helyesen besorolni dokumentumokat. Ennek is köszönhető az, hogy már az alapmódszer is igen hatékony. A legfőbb hátránya a futási időben jelentkező magas számítási igény, hiszen akármilyen kifinomult előfeldolgozást alkalmazunk, lényegesen lassabb lesz, mint a lineáris osztályozók esetén az osztályonkénti egy skaláris szorzás végrehajtása.

Megjegyezzük még, hogy a *k*-NN osztályozó és általában a lusta tanulók csak dokumentumvezérelt osztályozásra használhatók, mivel a kategóriavezérelt működéshez tárolni kéne az összes tesztadatra vonatkozó részeredményt, ami nyilvánvalóan alkalmatlanná teszi az ilyen felhasználásra.

5.4.6. Szupportvektor-gépek (SVM)

A számos más alkalmazási területen is jó eredményeket adó szupportvektor-gépekkel (support vector machine, SVM) történő osztályozás szöveges dokumentumok esetén is egyike a leghatékonyabb módszereknek [92]. Az SVM alapverziója lineáris osztályozók családjába tartozik, és bináris osztályozási problémák megoldására alkalmas. A többi lineáris osztályozóhoz képest az a fő ismérve, hogy nemcsak egyszerűen egy olyan hipersíkot keres, amely elválasztja a pozitív és negatív tanítómintákat, hanem ezek közül a legjobbat, vagyis intuitíve azt, amelyik a két osztály mintái között épp „középen” fekszik (5.8. ábra). Másképp fogalmazva olyan döntési hipersíkot határoz meg, amely maximalizálja a *tartalék*ot, azaz a hipersík és a hozzá legközelebbi pozitív és negatív tanítóadatok közti eltérést. Ezeket a tanítóadatokat *szupportvektor*oknak nevezzük. A hipersík meghatározásában a tanítóadatok közül csak a szupportvektorok játszanak szerepet.

Nézzük meg, miért előnyös, ha a fenti módszerrel adjuk meg a szeparáló hipersíkot! Egyrészt azért, mert a döntési felülethez közeli pontok osztálybasorolása a legbizonytalanabb. Minél kevesebb pont esik erre a területre, annál kevesebb bizonytalan döntést hoz az osztályozó. Másrészt a maximális tartalék által meghatározott szélességű szeparáló sáv elhelyezésére sokkal kevesebb lehetőség van, mint egy tetszőleges szeparáló hipersík esetén. Ez azt jelenti, hogy kevésbé függ a konkrét adatoktól, ezért nagyobb általánosító képességgel bír az osztályozási modell.



5.8. ábra. Az SVM a tartalékot maximalizáló hipersíkot választja

Az alábbiakban először a lineárisan szeparálható esetet vizsgáljuk, majd megmutatjuk, hogy milyen alternatív megoldások vannak, ha a szeparabilitási feltétel sérül.

5.4.6.1. A szeparábilis eset

A SVM formális definiálásához először idézzük fel a perceptronnál (ld. a 115. oldalon, (5.8) képlet) megadott döntési hipersík egyenletét. Az SVM esetén — ahogy a perceptronnál is — a negatív tanítóadatokra $\tilde{\Phi}(\mathbf{d}_i, c_j) = -1$ értéket alkalmazzuk, és a $\mathbf{d}_i$ címkéje röviden legyen $y_i \in \{+1, -1\}$. Továbbra is alkalmazkodva az irodalomban szokásos jelöléshez, a c kategóriát határoló felület normálvektorát $\mathbf{w}$-vel, a $\mathbf{d}_i$ dokumentumvektort pedig $\mathbf{x}_i$-vel jelöljük. Ekkor tehát a $\text{sign}(\mathbf{w}^T\mathbf{x}_i + b)$ értéke alapján döntünk. A hipersíkhoz legközelebb eső elem távolsága alapján a hipersík paraméterei skálázhatók úgy, hogy fennálljon

$$\text{minden } \mathbf{x}_i\text{-re az } \; y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \tag{5.22}$$

egyenlőtlenség, ahol az egyenlőség a szupportvektorokra teljesül. Ekkor egy tetszőleges $\mathbf{x}$ pont távolsága a hipersíktól

$$d(\mathbf{x}, \mathbf{w}, b) = \frac{|\mathbf{w}^T\mathbf{d}_i + b|}{\|\mathbf{w}\|}. \tag{5.23}$$

Az optimális hipersík tehát az, amelyik maximalizálja az (5.23) távolságot a szupportvektorokra. Ekkor a tartalék a képlet által megadott távolság kétszerese lesz: $2/\|\mathbf{w}\|$. Ez akkor lesz maximális, ha $\|\mathbf{w}\|$ minimális.

Az optimális hipersíknak tehát az alábbi feltételeket kell együttesen kielégítenie: (1) $\|\mathbf{w}\| = \mathbf{w}^T\mathbf{w}$ minimális; (2) (5.22) fennáll. Ez egy standard lineáris kényszerfeltételekkel bíró *kvadratikus optimalizálás*i feladat, amely egy jól ismert, és számos megoldással rendelkező matematikai probléma. Itt csak a megoldás vázlatát ismertetjük, valamint kitérünk néhány fontos részletre.

A szélsőérték-keresési problémát az alábbi módon definiáljuk:

$$J(\mathbf{w}, b, \alpha) = \frac{1}{2}\|\mathbf{w}\| - \sum_{i=1}^{N} \alpha_i \left(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 \right), \tag{5.24}$$

ahol $\alpha_i \geq 0$-k a *Lagrange-multiplikátor*ok. Az (5.24) kifejezést $\mathbf{w}$ és b szerint minimalizálni, illetve α szerint maximalizálni kell. A $\mathbf{w}$ szerinti minimalizálásból $\mathbf{w} = \sum_{i=1}^{N} \alpha_i \mathbf{x}_i y_i$, a b szerintiből a $\sum_{i=1}^{N} \alpha_i y_i = 0$ egyenleteket kapjuk. Ezeket behelyettesítve kapjuk a duális problémát:

$$\max_{\alpha} \left(\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j \right) \tag{5.25}$$

az $\alpha_i \geq 0$ minden i-re, és $\sum_{i=1}^{N} \alpha_i y_i = 0$ kényszerfeltételekkel. A megoldásban a legtöbb α_i értéke 0 lesz. A pozitív α-khoz tartoznak a szupportvektorok. Az (5.25) megoldása után a hipersík egyenletét meghatározó paramétereket a $\mathbf{w}=\sum_{i=1}^{N} \alpha_i \mathbf{x}_i y_i$ és $b = y_k - \mathbf{w}^T\mathbf{x}_k$ egyenletekkel kapjuk, az utóbbinál $\mathbf{x}_k$ a szupportvektorokat jelöli, ahol $\alpha_k > 0$. Ezután az SVM osztályozófüggvénye a

$$\Phi(\mathbf{x}_i, c) = \text{sign} \left(\sum_{j=1}^{N} \alpha_j \mathbf{x}_j^T \mathbf{x}_i y_j + b \right) \tag{5.26}$$

alakban írható fel.

Vegyük észre, hogy a dokumentumok adatait csak belső szorzatban használjuk fel mind a duális probléma felírásában, mind az osztályozófüggvény megadásánál.

5.4.6.2. A nemszeparábilis eset

Nagy adathalmazok esetén a szeparabilitási feltétel gyakran nem teljesül, sőt lineárisan szeparálható halmazok esetén is lehetnek olyan kilógó, izolált vagy zajos pontok, amelyeket célszerűbb figyelmen kívül hagyni az osztályozó megkonstruálásánál. Ebben az esetben az előbbiekben formalizált feladat a ξ_i *gyengítő változó*k bevezetésével kis módosítással érvényben marad. Az (5.22) kifejezés ekkor

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \text{minden } \mathbf{x}_i\text{-re}, \tag{5.27}$$

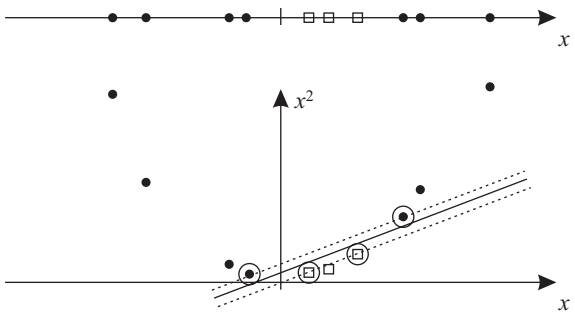
egyenlőtlenségekre változik. Ha minden i-re $0 < \xi_i \leq 1$ teljesül, akkor csak a biztonsági sávba eső tanítóadatok vannak, ha létezik $\xi_i > 1$, akkor a vonatkozó tanítóadat a hipersík hibás oldalára esik. Ekkor a szélsőérték-probléma a $\|\mathbf{w}\| = \mathbf{w}^T\mathbf{w} + C\sum_i \xi_i$ minimálizálását, és az (5.27) maximalizálását jelenti. A feladat megoldásánál a tartalék szélessége és a hibásan osztályozott, illetve a biztonsági sávba benyúló pontok száma közötti optimalizálást hajtunk végre. A C a *regularizációs faktor*, amellyel az adatok fontossága szabályozható. Ha kicsi, akkor kevés büntetést fizetünk a túllógó pontokért, ha nagy, akkor sokat.

A duális feladat megegyezik az (5.25)-ben megadottal, csak az α_i-re vonatkozó kényszerfeltételek változnak: $C \geq \alpha_i \geq 0$ minden i-re, és $\sum_{i=1}^{N} \alpha_i y_i = 0$. A hipersík paraméterei közül $\mathbf{w}$ változatlan marad, csak b változik $b = y_k(1-\xi_k) - \mathbf{w}^T\mathbf{x}_k$-ra, ahol $k = \arg\max_k \alpha_k$.

5.4.6.3. A nemlineáris eset

Előfordulhat olyan eset is, amikor a probléma nemlinearitása olyan nagyságrendű, hogy az előző alpontban tárgyalt módosítással sem lesz hatékony az osztályozó. Ilyen helyzetre mutat példát az 5.9. ábra. A probléma egy lehetséges megoldása az, ha az adatokat nagyobb dimenziójú térbe transzformáljuk, ahol az adathalmaz már szeparálható. Az erre alkalmas matematikai objektumokat *kernel*nek vagy *magfüggvény*nek nevezik. A magfüggvények segítségével a lineárisan nem szeparálható feladatok szeparálhatóvá tehetők az adatoknak jobban reprezentáló problématérbe való transzformálásával.

Ahogy már rámutattunk, a dokumentumok adatai csak belső szorzatban szerepelnek az (5.26) képletben. Jelöljük ezt $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T\mathbf{x}_j$-vel. Ekkor, ha a bemenő



5.9. ábra. Példa lineárisan nem szeparálható adathalmazra, amely a transzformált térben már szeparálhatóvá válik

adatokra alkalmazzuk a $\phi(\mathbf{x})$ koordináta-transzformációt, akkor a belső szorzat $\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$-re módosul. Ekkor kedvező esetben a ϕ koordináta-transzformációt nem kell elvégezni, csak a transzformáció által meghatározott belső szorzatot kell kiszámolni az eredeti adatpontok alapján. Ezt az összefüggést határozza meg a K magfüggvény, ami tehát a transzformált térben érvényes belső szorzat függvénye. Ekkor elegendő K-t alkalmazni az osztályozófüggvénnyel való számoláskor:

$$\Phi(\mathbf{x}_i, c) = \operatorname{sign}\left(\sum_{j=1}^{N} \alpha_j K(\mathbf{x}_j^T, \mathbf{x}_i) y_i + b\right). \tag{5.28}$$

Azt, hogy egy függvény magfüggvény-e, a Mercer-tétel segítségével lehet eldönteni. A gyakorlatban az alábbi magfüggvényeket szokták alkalmazni:

- polinomiális: $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j + 1)^d$, $d \in \mathbb{N}$.
- radiális bázisfüggvény (RBF): $K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x}_i - \mathbf{x}_j\|\right)$.
- kétrétegű perceptron: $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh\left(\beta_0 \mathbf{x}_i^T \mathbf{x}_j + \beta_1\right)$.

Ez utóbbi a Mercer-tételt csak bizonyos β_0 és β_1 értékek esetén teljesíti.

Bár az SVM-osztályozó formalizmusa lényegesen eltér az eddigiektől, a gyakorlatban ez is egy hipersíkot határoz meg az eredeti vagy — magfüggvények használata esetén — a transzformált problématérben. A kernel segítségével gyakorlatilag új jellemzőket vezetünk be. Polinomiális kernel esetén például a jellemzők konjunkcióját (szavak együttes előfordulását) is alkalmazhatjuk a polinom nagyságrendjéig. Az RBF segítségével hipergömb alakú jellemzőket is megadhatunk, és ilyenek kombinációival igen bonyolult döntési felületeket alkothatunk.

5.4.6.4. SVM alkalmazása szövegosztályozásra

Az SVM-et szövegosztályozásra elsőként Joachims [92] alkalmazta. A nemszeparábilis esetre vonatkozó módosítást [218] használták először. A módszer az egyik leghatékonyabb osztályozónak bizonyult szövegekre (ld. az 5.5.3. szakaszt). Előnyei közé tartozik, hogy az SVM használata mellett általában nincs szükség a problématér redukciójára, mivel egyrészt a módszer elég robosztus a túltanulásra, másrészt jól skálázható, azaz nem érzékeny a magas dimenziószámra. Az eljárás további kedvező tulajdonsága, hogy nincs szükség validálásra, mert a felhasznált matematikai háttér segítségével a módszer elméleti, és nem tapasztalati úton eleve az optimális döntési hipersíkot határozza meg.

Az SVM hatékonyságának kulcsa, hogy jó általánosító képességgel bír, azaz ismeretlen adatokra is igen pontos becslést képes adni. A tanítóadatok tulajdonsá-

gai és a módszer tanítóadatokon való viselkedése alapján elméleti felső hibakorlát adható az új adatokra vonatkozó hibára.

Az SVM hatékonysága és kedvező tulajdonságai számos kutatót ösztönzött a módszer vizsgálatára, továbbfejlesztésére. Megmutatták például, hogy a módszer nagyon nagy adathalmazok esetén alkalmazható úgy, hogy a végső SVM-et a tanítóadatok kisebb részhalmazaira megalkotott SVM-ek kombinációiként állítják elő. Az [55] munka egy, a tanítókorpusz méretében lineáris tanulóalgoritmust ismertet, amely tehát azonos nagyságrendűvé teszi az SVM-módszer tanulási sebességét a Rocchio-eljárásával.

Az SVM-módszer elterjedését és térhódítását nagyban elősegítette a Joachims által közreadott ingyenes SVM-osztályozó programcsomag az SVMLIGHT.[11] Az érdeklődő Olvasónak ajánljuk még az alábbi SVM-mel foglalkozó munkákat: [32, 166, 199].

5.4.7. Regressziós modellek

Már az SVM első publikálása előtt megjelentek az első munkák, amelyek igen hasonló matematikai módszereket vetettek be a szövegosztályozási feladat megoldására. Ezek közül kiemelkednek a különböző regressziós módszereken alapuló megoldások, amelyek az osztályozás problémáját a valós értékű $\tilde{\Phi}$ függvény approximációjaként fogják fel, ahol a Φ approximáló függvény a lehető legjobban illeszkedik a tanítóadatokra.

Az egyik első ilyen eljárás a *lineáris legkisebb négyzetek módszere* (linear least-squares fit, LLSF) [217], amely a bemeneti M-hosszú $\mathbf{d}_i$ dokumentumvektorok és a kimeneti $|C|$-elemű $\mathbf{y}_i$ címkevektorok[12] közötti függvénykapcsolat közelítő meghatározását valósítja meg. Az osztályozás ekkor a tesztdokumentumokra történő címkevektor generálását jelenti, amelynek elemei — ellentétben a tanítódokumentumokkal — nem bináris, hanem valós számok lesznek, azaz a módszer közvetlenül alkalmas rangsoroló osztályozási feladatra.

A közelítést egy olyan $\tilde{\mathbf{A}} \in \mathbb{R}^{|C| \times M}$ mátrix meghatározásával végezzük, amely minimalizálja a legkisebb négyzetek szerinti hibát, azaz amelyre:

$$\tilde{\mathbf{A}} = \arg\min_{\mathbf{A}} \|\mathbf{AD} - \mathbf{Y}\|_F, \quad \text{ahol} \quad \|\mathbf{B}\|_F = \sqrt{\sum_{j=1}^{|C|} \sum_{k=1}^{M} b_{ij}^2} \tag{5.29}$$

[11] `svmlight.joachims.org`

[12] értéke azon a helyen 1, amelyre $\tilde{\Phi}(\mathbf{d}_i, c_j) = 1$, amúgy 0

a $|C| \times M$-es mátrix *Frobenius-normá*ja, $\mathbf{D} \in \mathbb{R}^{M \times N}$ a tanítódokumentumok mátrixa, $\mathbf{Y} \in \mathbb{R}^{|C| \times N}$ pedig a címkevektorokból képzett mátrix. Az (5.29) kifejezést mininalizáló $\tilde{\mathbf{A}}$ mátrixot például szinguláris értékfelbontással lehet meghatározni (ld. még a 2.3.2.2. alpontot).

A módszer az egyik leghatékonyabb szövegosztályozó, de az SVD-eljárás időigénye miatt lényegesen lassabb, mint az egyszerűbb lineáris osztályozók, pl. a naiv Bayes vagy a Rocchio-módszer.

A regressziós modellek megbízhatósága jelentősen javítható, ha regularizációs tagot illesztünk a minimalizálandó kifejezésbe (5.29). Ez a technika nagyon hasonló ahhoz, ahogy az SVM kezeli a nemszeparábilis esetet. A [221] tanulmány részletes elemzést ad a különböző regularizált regressziós modellekről és megmutatja, hogy hatékonyságuk az SVM-mel gyakorlatilag azonos.

5.4.8. Osztályozók kombinációja

Az osztályozók kombinálása azon az ötleten alapszik, hogy ha több, esetleg különböző megközelítést alkalmazó osztályozót építünk ugyanarra a feladatra, akkor a módszerek együttes eredményei alapján jobb döntés hozható, mint csupán egy osztályozó predikciójára hagyatkozva. A kiválasztott osztályozók együttesét *(osztályozó) bizottságnak* (classifier committee, ensemble classifier), elemeit *tagok*nak nevezzük. Az osztályozó bizottság döntési mechanizmusát *szavazásos osztályozás*nak hívjuk.

A bizottság tagjainak kiválasztásánál általában úgy járunk el, hogy a tagok lehetőleg minél függetlenebbek legyenek, azaz különböző elven működő osztályozó algoritmust alkalmazzanak [196], ugyanakkor az előfeldolgozási fázisuk (indexelés, szótárépítés) azonos legyen. Az eredmények kombinációjára számos eljárás létezik [118], amelyek eltérő mértékben és módon veszik figyelembe a tagok hatékonyságát.

A legegyszerűbb verzió a bináris osztályozási feladatra illeszkedő *többségi döntés*i stratégia (majority voting), amikor azt az osztályt választjuk, amelyikre legalább $\frac{R+1}{2}$ tag szavaz, ahol a tagok száma, R, páratlan [119]:

$$c_{\mathrm{MV}} = \arg\max_{j} \sum_{r=1}^{R} \Phi_k(\mathbf{d}, c_j). \tag{5.30}$$

Általános esetben jobban alkalmazható a súlyozott döntés, ahol az (5.30) képletben az egyes osztályozókhoz hatékonyságuk alapján súlyokat rendelünk. A súlyok értékeit a validációs halmazon szokták beállítani. További kombinációs módszereket tartalmaz [118].

Az eredeti tanítóhalmazból kialakított ideiglenes tanítóhalmazok megvalósításától függően is több verziója létezik a szavazásos osztályozásnak.

Az ún. *bagging* módszer [29] esetén az eredeti N elemű tanítóhalmazból *ismétléses módon* véletlenszerűen kiválasztunk N elemet. Az így kapott tanítóhalmaz az eredetiből bizonyos elemeket többször, másokat egyszer sem tartalmaz. Az eredetiből kivett elemek gyakoriságát diszkrét Poisson-eloszlással modellezzük. Ezt R-szer elvégezve ugyanennyi különböző dokumentumgyűjteményhez jutunk, amire a bizottsági tagokat lefuttatva R eredményt kapunk. A vizsgált $\mathbf{d}$ dokumentumot (5.30) szerint ahhoz a kategóriához rendeljük hozzá, amelyikre a legtöbb tag „szavaz”.

Az *AdaBoost* eljárás verziói [164, 165] ugyanazt az osztályozót alkalmazzák egymás után különböző tanítóhalmazzal. A tanítóadatok súlyát adaptív módon attól függően változtatják, hogy milyen eredményt adott az előző osztályozásoknál. Egy dokumentum súlyát növelik, ha az osztályozás sikertelen volt, és csökkentik, ha sikeres. A végső osztályozó az R-edik osztályozó eredményeként áll elő.

A bizottságokat osztályozók albizottságaiként összeállítva [170], illetve a bizottságok döntési fákkal való kombinációját [206] alkalmazva tovább lehet javítani a *boosting* típusú módszerek hatékonyságán.

5.5. Osztályozók elemzése

5.5.1. Elfogultság és variancia közötti kompromisszum

Az ismertetett modellek legnagyobb része a lineáris osztályozók nagy családjába tartozik, amelyek egy hipersík segítségével osztják két részre a problémateret. A lineáris osztályozókon belül két nagy csoportot különböztethetünk meg:

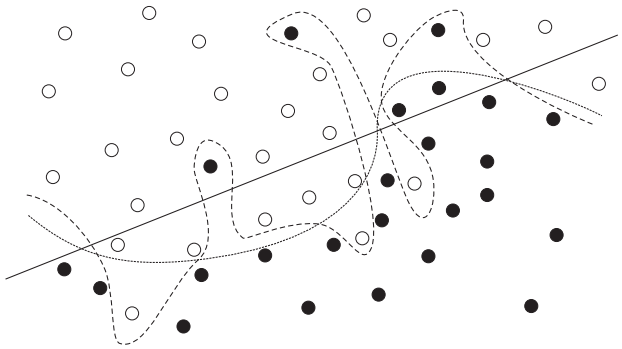
- A dokumentumok globális eloszlásán alapuló osztályozók csoportja, amely a tanítókorpusz tulajdonságait átlagolva kezeli, ezért az abban esetlegesen meglévő zajra, illetve egyedi eltérésekre kevéssé érzékeny. Ezért ezek az osztályozók robosztusak, pl. naiv Bayes-, Rocchio-módszer, ill. lineáris regresszió.
- A döntési hipersíkoz közel eső dokumentumoknak nagyobb súlyt adó osztályozók csoportja, amely megfelelő minőségű és mennyiségű tanítóadat esetén pontosabbak az előzőeknél. Ilyen például a lineáris magfüggvényű SVM, és a logikai regressziós eljárás.

Vizsgáljuk meg, mi az oka annak, hogy a lineáris osztályozók a leghatékonyabb módszerek közé tartoznak, és miért nem jobbak náluk szignifikánsan a bonyolultabb döntési felületet megtanulni képes nemlineáris osztályozók, mint pl. a

k-NN-algoritmus! A válasz az osztályozási modellek leíróereje, elfogultsága és varianciája közötti összefüggésben, valamint a szövegosztályozási problémák jellegében rejlik. Itt variancián az osztályozó megépítésére vonatkozó lehetőségek változatosságát értjük egy véletlenszerűen generált tanítókorpusz függvényében.

A lineáris osztályozók leíróerejét a modell erősen korlátozza. Az ilyen modell *elfogultsága* (bias) magas, viszont *varianciája* (variance) kicsi, hiszen csak egyfajta — lineáris — döntési felület képes meghatározni. A nemlineáris osztályozók esetén éppen fordított a helyzet: ezek elfogultsága alacsony és varianciájuk magas, nincsenek „elkötelezve" egy adott döntési felülettípus mellett, ezért megfelelő mennyiségű tanítóadat esetén tetszőleges döntési felület megalkotható a segítségükkel. Lineáris osztályozók esetén a variancia kicsiny, mivel a döntési felület — a tanítókorpusztól függetlenül — mindig egy hipersík lesz. A *k*-NN osztályozó esetén ellenben a döntési felület nagymértékben függ a tanítókorpusz jellemzőitől, ami érzékennyé teszi a modellt a zajos adatokra, különösen ha a tanítókorpusz kicsiny.

A nagy variancia tehát nem feltétlenül előnyös, hiszen a modell érzékennyé válik a tanítókorpusz hibáira, és mivel a megalkotott döntési felület nagymértékben függ a tanítóadatoktól, ezért a nemlineáris osztályozók hajlamosak a túltanulásra. A magas elfogultság szintén hátrányos lehet, ha az adott problémánál nagyon sérül a lineáris szeparabilitási feltételezés. A megoldást a két jellemző közötti kompromisszum jelenti (ld. az 5.10. ábrát). Az ábrán mind a lineáris, mind pedig a kacskaringós szaggatott vonallal megadott, a tanítóadatokhoz tökéletesen illeszkedő megoldás rossz osztályozási tulajdonságot fog eredményezni. A kettő között húzódó, a tanítóadatok tényleges eloszlását figyelembe vevő pontozott vonallal jelölt döntési felületnek sokkal jobb *általánosító képessége* van.



5.10. ábra. Elfogultság és variancia közötti kompromisszum

Szöveges dokumentumok osztályozása esetén joggal feltételeznénk, hogy a sokdimenziós problématérben az optimális döntési felületet nem lehet egyszerű hipersíkkal leírni. Meglepő, de ez a feltevés téves! A dimenziók száma gyakran meghaladja a dokumentumok számát, így mindig lehet találni egy olyant, amely szerint az adott dokumentum a megfelelő térrészbe sorolható. Sőt, a tanítókorpusz lineáris szeparálhatósága a gyakorlatban csekély jelentőségű. Sokkal fontosabb, hogy a modell által alkotott döntési felület jó általánosító képességű legyen, és a teszthalmazra is jól működjön, semmint hogy minden tanítóadatot jól osztályozzon. Ezért a lineáris osztályozók nagy dimenziójú terekben hatékony eszköznek bizonyulnak. A nemlineáris osztályozók egyes esetekben ugyan jobb eredményt adhatnak, de ez korántsem tekinthető általános érvényű tendenciának [125].

5.5.2. Hatékonyságmérés

Az osztályozási módszerek hatékonyságának mérésére az IR-rendszerekben alkalmazott relevanciamértékeket adaptáljuk (ld. a 3.2.2. pontot). Bináris osztályozási feladat esetén közvetlenül alkalmazhatjuk a pontosság (P) és a felidézés (R) korábbi definícióit ((3.1) képletek), ekkor ugyanis a kategóriába tartozás és a keresőkifejezéshez való relevancia egymásnak megfeleltethető tulajdonságok. Korai munkákban előfordult a *szabatosság* (3.4) alkalmazása is, mint a felidézés és pontosság egyfajta kombinációja, de mivel itt a nevező akár nagyságrendekkel is nagyobb lehet a számlálónál, ezért a (3.4) kevésbé érzékeny az utóbbi megváltozására, emiatt használata nem vált be [169, 216]. A P és R együttes maximalizálásának további lehetősége az *egyensúlyi pont* meghatározása, amire $P \approx R$ teljesül. Az egyensúlyi pontot az adott módszer paramétereinek beállításával kaphatjuk meg. Itt problémát jelenthet az, hogy egyes módszereknél esetleg nincs ilyen paraméterbeállítás, illetve, hogy a két érték azonossága nem feltétlenül kívánatos cél [115]. Az IR-rendszerekhez hasonlóan osztályozási feladatoknál is alkalmazhatjuk a két alapmértékből származtatott F_β-mértéket is (ld. (3.2)-t). Ez lett az utóbbi időben az osztályozókra a legáltalánosabban használt jósági kritérium.

Az eddig tárgyalt mértékek a bináris feladatra, azaz egy kategóriára vonatkoztak. Könnyen lehet azonban átlagolással adaptálni őket egy- és többcímkés osztályozáshoz is (többszintű osztályozás kiértékelését az 5.6. pont megfelelő részében tárgyaljuk). Az átlagolást kétféleképpen lehet elvégezni: *mikro-átlagolt* mértékek esetén dokumentumonként számoljuk ki az adott mértéket, majd ezeket átlagoljuk, *makro-átlagolt* esetben a kategóriákra számolt mértékek átlagaként kapjuk az eredményt.

$$R^{\mu} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} = \frac{\sum_{j=1}^{|C|}\mathrm{TP}_j}{\sum_{j=1}^{|C|}\mathrm{TP}_j+\mathrm{FN}_j}, \quad P^{\mu} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} = \frac{\sum_{j=1}^{|C|}\mathrm{TP}_j}{\sum_{j=1}^{|C|}\mathrm{TP}_j+\mathrm{FP}_j}; \quad (5.31)$$

$$R^{M} = \frac{\sum_{j=1}^{|C|}R_j}{|C|}, \quad P^{M} = \frac{\sum_{j=1}^{|C|}P_j}{|C|}. \quad (5.32)$$

Itt a μ felső index a mikro-, M a makro-átlagolást jelöli, R_j, ill. P_j a c_j kategóriára vonatkozó felidézés, ill. pontosság. A mikro-átlagolás tehát a dokumentumokhoz, míg a makro-átlagolás a kategóriákhoz rendel azonos súlyt.

Rangsoroló osztályozók hatékonyságát (ld. a 106. oldalon) a *11 pontos átlagos pontosság*gal mérhetjük, ld. (3.3). Ekkor egy tesztdokumentumra a felidézést

$$R = \frac{\text{A listában szereplő helyes kategóriák száma}}{\text{Az összes helyes kategóriák száma}} \quad (5.33)$$

képlettel számoljuk, majd a hányados 11 rögzített értékét meghatározzuk. Ez megadja, hogy a 11 pontossági érték számolásánál a listának hány elemét kell figyelembe venni. A teljes D_{Test} halmazra vonatkozó globális hatékonyság értéket a dokumentumonként kiszámolt értékek átlagaként kapjuk.

5.5.3. Osztályozók összehasonlítása

A módszerek összehasonlítását standard dokumentumgyűjtemények segítségével végezhetjük el. A módszerek korrekt összehasonlításához teljesülnie kell, hogy

1. egyazon gyűjteményen, ugyanazokkal a dokumentumokkal és kategóriákkal teszteljünk;
2. ugyanazt a tanító/teszt felosztást használjuk;
3. ugyanazt a hatékonysági mértéket alkalmazzuk rögzített paraméterbeállítással.

Egy- és többcímkés szövegosztályozási feladatok vizsgálatára legtöbbet a Reuters-21578 korpuszt,[13] illetve ennek különböző verzióit alkalmazták összevetési alapnak,[14] bár a fenti irányelveket nem mindig tartották szem előtt. Ennek köszönhetően a teljes kép kialakításához öt, többé-kevésbé standard beállításra

[13] `www.daviddlewis.com/resources/testcollections/reuters21578/`

[14] Gyakran használták tesztelésre az OHSUMED [87] és a 20-Newsgroups gyűjteményeket is [111].

kapott eredménysort szükséges összehasonlítani [169]. A gyűjtemény SGML-formátumú híranyagokat tartalmaz, amelyek 135 gazdasági jellegű kategóriába vannak besorolva. A gyűjteményen többféle tanító/teszt felosztással végeztek kísérleteket, ezek közül a legelterjedtebb az Apté által javasolt [9] (9603 tanító-, 3299 tesztdokumentum), illetve ennek bizonyos módosításai, egyszerűsítései. A korpusz dokumentumai több kategóriához is hozzá lehetnek rendelve, de előfordulnak kategória nélküli dokumentumok is. A dokumentumonkénti átlagos kategóriaszám csupán 1,24. A tanítódokumentumok eloszlása is egyenetlen, a legnagyobb elemszámú kategóriának 2709 tanítódokumentuma van, de a kategóriák feléhez kevesebb mint 10 dokumentum tartozik.

A Reuters-21578-on tesztelt osztályozók legátfogóbb összehasonlítása Sebastiani munkájában található [169]. Ennek alapján a következő megállapításokat tehetjük:

- A legjobb hatékonyságú osztályozók a boosting technikát alkalmazó bizottságok, az SVM, valamint a k-NN módszert alkalmazó algoritmusok.
- A neurális hálózat alapú módszerek szintén jó teljesítményt nyújtanak, bár az előző csoportba sorolt eljárásoknál valamivel rosszabb eredményt adnak. Speciális átmeneti függvény használatával azonban ez a módszer is az előző csoporthoz hasonló, vagy akár jobb eredményekre is képes [186].
- A harmadik csoportba a Rocchio-eljárás és a naiv Bayes-alapú módszerek tartoznak, ezeknek a leggyengébb az osztályozóképességük. Itt fontos megemlíteni, hogy az előbbi a majdnem pozitív tanítóadatok alkalmazásával (ld. az (5.7) képletet) lényegesen javítható. Ide sorolhatók még a döntési fa alapú módszerek is, amelyek alapesetben szintén a kevésbé hatékony eljárások közé tartoznak, de módosításokkal ezek is lényegesen javíthatók [55].

Fontosnak tartjuk azonban hangsúlyozni, hogy a fenti összehasonlítás mindössze egy dokumentumkorpusz alapján készült, ezért a kapott eredmények csak irányadónak tekinthetők. Minden osztályozási feladat esetén mindig az adott problémának legmegfelelőbb eljárást kell alkalmazni. Az osztályozási módszer, illetve az azt megelőző előfeldolgozási lépések kiválasztása függ

- a rendelkezésre álló tanítókorpusz méretétől és minőségétől,
- a dokumentumok nyelvétől és nyelvezetétől,
- a korpusz által generált szótár méretétől,
- a feladat jellegétől, azaz, hogy mi a célja az osztályozásnak, mennyire kritikus a hatékonyság,

- az osztályozó felépítésére rendelkezésre álló időtől stb.

A 2007-ben orvosi szövegek osztályozására kiírt nemzetközi verseny[15] győztese például a fenti összehasonlításban hátrasorolt C4.5 döntési fa osztályozót alkalmazott. Más szöveg- (és adat-)bányászati versenyeken az osztályozóbizottságokra, illetve neurális hálózatokra épülő megoldások szerepeltek sikeresen [94, 172, 189].

5.6. Hierarchikus osztályozás

5.6.1. A taxonómia felhasználása

Független kategóriákból álló kategóriarendszer esetén a dokumentumok számának növekedése, és a lefedett témakörök sokfélesége áttekinthetetlen kategóriarendszert eredményezhet. Ezt a problémát a kategóriák hierarchizálásával, azaz *taxonómiá*ba rendezésével könnyen át lehet hidalni. A kategóriák közti összefüggés figyelembevételével a taxonómián működő osztályozók hatékonysága növelhető, ami az algoritmusok kisebb-nagyobb módosítását teszi szükségessé.

A feladat megoldására kézenfekvő ötlet a probléma szintenkénti dekomponálása: a gyökérkategóriából kiindulva kiválasztjuk az első szinten a legjobb kategóriát, majd ezt az algoritmust folytatjuk a kiválasztott kategóriából kiindulva, amíg a módszer levélkategóriához érve nem terminál. A kiválasztásnál általában a *mohó algoritmus*t vagy annak valamilyen gyengített változatát használhatjuk. A hierarchikus osztályozást az algoritmusok kevés kivételtől eltekintve egyszerű osztályozási feladatok sorozataként oldják meg.

Hierarchikus osztályozásra a naiv Bayes-módszert alkalmazza a [130] munka. A kevés tanítóadattal rendelkező levélkategóriák paramétereit az ún. *shrinkage* (apadás) statisztikai simító eljárás segítségével határozza meg, a szülő kategóriák megfelelő adatait felhasználva. A módszer segítségével a mohó algoritmus egyik jellegzetes hibája — ti., hogy a taxonómia felső szintjén elkövetett osztályozási hibát már nem lehet korrigálni — nagy részben kiküszöbölhető.

5.6.2. HITEC osztályozó

A neurális hálózatok architektúrájának és a taxonómiáknak strukturális analógiája egyszerű lehetőséget nyújt az előbbiek adaptációjára hierarchikus osztályozás esetén. A HITEC[16] neurális hálózat alapú, hibavezérelt (ld. 5.5. ábra), akár kötegelt,

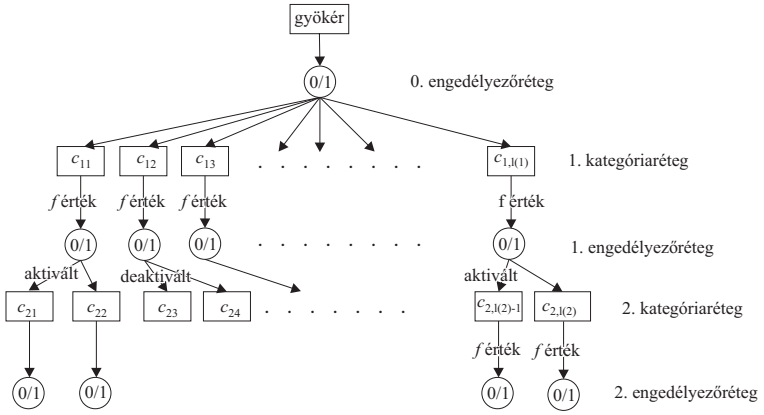
[15] computationalmedicine.org/challenge/

[16] categorizer.tmit.bme.hu

akár fokozatos tanulási feladatra alkalmazható szövegosztályozó [184, 185]. A HITEC kisebb módosításokkal alkalmazza a fent vázolt problémadekompozíciós sémát. A neurális hálózat bemenete a dokumentumvektor, kimenetét a levélkategóriák képezik, a két szint között pedig a taxonómia architektúrájának megfelelő rejtett rétegek találhatók.[17] A rejtett rétegeknek két típusa van: kategória- és engedélyező réteg (ld. az 5.11. ábrát). A kategóriarétegek gyakorlatilag az egyszerű osztályozás kimeneti rétegének felelnek meg, az engedélyező réteg pedig eldönti, hogy a következő szinten az adott kategória gyerekkategóriáit aktivizáljuk-e, vagyis részt vesznek-e a kiértékelésben. A kiértékelő függvény általános alakja:

$$f(\mathbf{d}_i) = (1+\eta)g\left(\sum_{k=1}^{M} w_{ik}w_{jk} + b\right), \tag{5.34}$$

ahol η és b a vetítést és eltolást meghatározó küszöbértékek, g pedig a skalárszorzaton alkalmazott simítófüggvény, többnyire a $\tanh(\cdot)$ függvény. Vegyük észre, hogy ekkor a g magfüggvény lesz, amelyet a dokumentum- és kategóriavektorok belső szorzatára alkalmazunk. Az (5.34) függvény mind kiválasztó, mind rangsoroló osztályozás esetén alkalmazható.



5.11. ábra. A HITEC neurális hálózatának felépítése [184]

A HITEC a mohó algoritmus gyengített változatának alkalmazásával bővíti az egy szinten kiválasztott kategóriák körét, amivel minimalizálja a mohó jellegű

[17] Megjegyezzük, hogy a taxonómiának nem kell kiegyensúlyozottnak lennie, a különböző mélységű taxonómiarészek egységes kezelése pl. pszeudo-kategóriák beiktatásával megoldható.

következtetés fent jelzett hibáját. Egy kategóriát akkor választ ki, ha

$$f(\mathbf{d}_i) \geq \max(\vartheta, f_\ell(\mathbf{d}_i) \cdot \rho) \tag{5.35}$$

teljesül rá, ahol $\vartheta \in [0,1]$ a minimális tüzelési érték, f_ℓ az f maximuma a vizsgált ℓ-edik szinten, $\rho \in (0,1]$ pedig a legmagasabb értéktől való eltérést adja meg. Ezzel a szelekciós technikával több *kategóriaösvény*t lehet párhuzamosan versenyeztetni, a paraméterek segítségével a keresési tér szélessége szabályozható. ϑ és ρ alacsony értékei esetén a keresési tér széles lesz, magas értékek esetén pedig keskeny. Az előbbit kisméretű taxonómiákra és tanítókorpuszokra érdemes alkalmazni. Az 5.6.4. pontban ismertetett több ezer kategóriás taxonómiát és nagy tanítókorpuszt tartalmazó példa esetén a $\vartheta = 0{,}2$, $\rho = 0{,}8$ értékek bizonyultak optimálisnak. A levélszinten kiválasztó osztályozás esetén a legnagyobb f értékű kategóriát, rangsoroló osztályozásnál pedig közvetlenül az f értékeket használhatjuk fel.

5.6.3. Hatékonyságmérés

Taxonómiába való osztályozáskor a tanítódokumentumok osztálybasorolásának jellegétől függően több lehetőség van a hatékonyság mérésére. Ha a dokumentumokat úgy tekintjük, hogy csak levélkategóriákba vannak besorolva, akkor az egyszerű osztályozásnál ismertetett mértékeket lehet alkalmazni a levélkategóriák összességére (ld. az 5.5.2. pontot). Ez azonban félrevezető eredményt is adhat, hiszen általában „kevésbé rossz" az az osztályozási következtetés, amely egy levélkategória helyett annak testvérét találja meg (tehát a szüleik közösek), mint az, amelyik a kategóriarendszer teljesen más ágához rendeli a dokumentumot. Ha egy dokumentumot nemcsak a levélkategóriához tartozónak tekintünk, hanem a gyökértől az adott levélig található kategóriaösvény összes eleméhez hozzárendeljük, akkor pontosabb képet kaphatunk az osztályozás értékelésekor, feltéve, ha a teljes — tehát nem csak levélszintű kategóriákra — taxonómiára számoljuk a pontosság, felidézés, F-mérték értékeit. Különösen indokolt ez akkor, ha vannak olyan dokumentumok, amelyek a taxonómia közbenső csomópontjaihoz vannak rendelve.

A hierarchikus osztályozás általános dekompozíciós megközelítése következtében a taxonómiában a levelek felé haladva az osztályozási hibák kumulálódnak. Ez a tendencia jól megfigyelhető, ha a szokásos mértékeket (pontosság, felidézés, F-mérték) *szintenként* számítjuk ki.

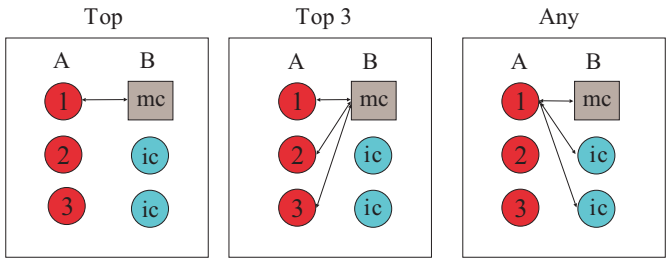
Hierarchikus osztályozásnál gyakran találkozunk a többszintű osztályozás problémájával, amikor tehát egy dokumentumnak vannak elsőrendű, másodrendű stb.

kategóriái. Itt a kétszintű osztályozás esetével foglalkozunk.[18] Erre az esetre a szakirodalom az egyszerű osztályozástól eltérő mérőszámokat javasol a hatékonyság mérésére, amelyeket célzottan az 5.6.4. pontban ismertetésre kerülő szabadalmi dokumentumgyűjteményhez alakítottak ki [104] (ld. az 5.12. ábrát).

Top: Az osztályozó által elsőnek rangsorolt kategóriát hasonlítja a dokumentum elsődleges kategóriájához.

Top 3: Az osztályozó által első háromként rangsorolt kategóriát hasonlítja a dokumentum elsődleges kategóriájához. Ha a három közül valamelyik talál, akkor az osztályozás sikeresnek számít.

Any: Az osztályozó által elsőnek rangsorolt kategóriát hasonlítja össze a dokumentumhoz tartozó összes (elsődleges, másodlagos) kategóriával. Ha valamelyikkel megegyezik, akkor az osztályozás sikeresnek számít.



5.12. ábra. Magyarázat a többszintű osztályozásnál alkalmazott mértékekhez (mc – főkategória; ic – egyéb kategória; A – az osztályozó predikciója; B – tényleges érték) [104]

A felsorolt mértékek mindegyike bináris. A mértéket átlagolhatjuk taxonómiaszintenként vagy az egész tesztkorpuszra vonatkozóan. A Top 3 és Any mértékek jól illeszkednek a félautomatikus vagy támogató osztályozás forgatókönyvéhez, ekkor ugyanis a humán szakértőnek jelentős segítséget nyújt már az is, ha a rangsor elején megtalálja a megfelelő kategóriát.

5.6.4. Hierarchikus osztályozók összehasonlítása

Mivel a hierarchikus osztályozással csak a 90-es évek végétől kezdtek el foglalkozni, ezért sokáig nem volt olyan dokumentumgyűjtemény, amelyen a különbö-

[18] Természetesen a taxonómia szintjeinek számára (mélységére) nem teszünk megkötést.

ző módszereket össze lehetett volna hasonlítani. A kutatók ezért különböző korpuszokon tesztelték algoritmusaikat, pl. a Reuters-gyűjtemény kategóriáit[19] rendezték különböző taxonómiákba [38, 49, 99, 208], és ezen végezték méréseiket. Ezek az eredmények azonban csak erős fenntartással hasonlíthatók össze, hiszen az 5.5.3. pontban ismertetett irányelvek nem teljesültek, sőt még a kategóriák halmaza is többnyire eltért.

A szabadalmi hivatalokban végzett munka során több különböző típusú, nagy munkaerő- és költségigényű hierarchikus osztályozási feladatot kell megoldaniuk (ld. az 5.2. szakaszt). Ennek az érdekeltségnek is köszönhető, hogy az első, kimondottan hierarchikus osztályozók tesztelésére alkalmasa dokumentumgyűjteményt a WIPO tette közzé 2002 végén [64], amely angol nyelvű szabadalmi szövegeket tartalmazott. Nem sokkal később egy német nyelvű gyűjteményt is publikáltak [63]. Az angol gyűjtemény mintegy 75000 XML-formátumú dokumentumból áll, amely összesen 3 GB-ot tesz ki, a német pedig 110 ezer XML-dokumentumot tartalmaz. A gyűjtemények fel vannak osztva tanító- és tesztadatokra. A dokumentumok az IPC kategóriarendszerének[20] felső négy szintjébe (osztály, szekció, alszekció, főcsoport) vannak besorolva, amely kb. 5000 kategóriát jelent összesen. Minden dokumentumnak pontosan egy elsőrendű (fő)kategóriája és tetszőleges számú, átlagosan 4–5 másodrendű kategóriája van.

Ezen a gyűjteményen végzett átfogó összehasonlítást az 5.12. ábrán látható mértékekkel egy nemzetközi kutatócsoport [62]. Munkájukban a naiv Bayes-eljárás, a legközelebbi szomszédok módszer, az SVM, és a WINNOW egy-egy hierarchikus osztályozásra specializált verzióját hasonlították össze. A módszerek hatékonyságát szekció és alszekció szintjén vizsgálták, az eredményeket az 5.1. táblázat tartalmazza. Ugyanezen a gyűjteményen a neurális hálózat alapú HITEC-et is tesztelték, és lényegesen jobb eredményeket kaptak: a taxonómiában egy szinttel lejjebb volt képes a HITEC a többi módszer által egy szinttel feljebb elért eredményre [184, 186]. Ennek alapján megállapítható, hogy a taxonómia architektúráját kihasználó, hierarchikus osztályozásra kifejlesztett neurális hálózaton működő algoritmus kedvezőbb eredményeket szolgáltat.

Másik nagyméretű dokumentumgyűjtemény a Reuters Corpus Volume 1 (RCV1),[21] amely mintegy 800 000 híranyagot tartalmaz, és az XML-dokumentumok három különböző taxonómiába vannak besorolva (téma, ipari kód, és területi

[19] gyakran csak egy kisebb részhalmazt

[20] `www.wipo.org/classifications/fulltext/new_ipc/index.htm`

[21] `about.reuters.com/researchandstandards/corpus/`

5.1. táblázat. A WIPO-alpha angol nyelvű szabadalmi dokumentumgyűjteményen elért eredmények összehasonlítása

Mérték	Módszer	IPC szint		
		szekció	alszekció	főcsoport
Top	HITEC	66.41	54.63	38.38
	NB	55.00	33.00	–
	SVM	55.00	41.00	–
	k-NN	51.00	39.00	–
	WINNOW	51.00	36.00	–
Top 3	HITEC	89.41	79.48	59.64
	NB	79.00	53.00	–
	SVM	73.00	59.00	–
	k-NN	77.00	62.00	–
	WINNOW	74.00	58.00	–
Any	HITEC	76.46	66.36	50.90
	NB	63.00	41.00	–
	SVM	62.00	48.00	–
	k-NN	58.00	46.00	–
	WINNOW	58.00	44.00	–

kód szerint). A kategóriák száma azonban itt sokkal kisebb, mint a szabadalmi korpuszok esetében, a taxonómiák mindössze 103 téma, 364 ipari és 366 területi kódot tartalmaznak. A dokumentumgyűjtemény teljes feldolgozását Lewis és munkatársai végezték el [117]. A korpusz egyre népszerűbbé válik a kutatók körében is, bár eddig a teljes körű vizsgálat eredményeiről csak elvétve publikáltak.

6. fejezet

Csoportosítás

A dokumentumok rendszerezésének másik lehetséges módja az osztályozás mellett a *csoportosítás*, vagy *klaszterezés*. A cél a dokumentumokból olyan elkülönült csoportokat alkotni, hogy az egy csoportba kerülők minél hasonlóbbak, az eltérő csoportokban lévők pedig minél különbözőbbek legyenek. A csoportba szervezés tehát a dokumentumok hasonlóságán alapul.

Az osztályozás és a csoportosítás között az a leglényegesebb eltérés, hogy az utóbbi esetén nem áll rendelkezésre tematikus kategóriarendszer, amelybe a dokumentumokat be lehetne sorolni. Az osztályozástól eltérően tehát a dokumentumok csoportosításánál

- nincsenek ismert címkéjű tanítódokumentumok, továbbá a csoportok általában a feladat elvégzése után sem jellemezhetők automatikusan címkékkel;
- a csoportok száma nem rögzített.

A csoportok kialakításához tehát semmiféle előre definiált struktúra nem áll rendelkezésre, nincs olyan minta, mint az osztályozás esetén a tanítókörnyezet, amiből tanulva a modellt fel lehet építeni. E tulajdonsága miatt a csoportosítás a *felügyelet nélküli gépi tanuló módszer*ek (unsupervised learning) családjába tartozik. A csoportokat — a tanítókörnyezetben manifesztálódó szakértői segédlet nélkül — a dokumentumok eloszlása és jellemzői alapján alakíthatjuk ki.

Hasonlóan az osztályozó módszerekhez, a szövegek csoportosítását is általános, tetszőleges objektumok csoportosítására alkalmas módszerek adaptálásával végezzük. A dokumentumok jellegéből adódóan a következő problémák megoldására kell felkészülni a vektortérmodell használatakor [18]:

- A dokumentumot leíró jellemzők nagy száma miatt a problématér dimenziója általában 10 000-es nagyságrendű. Ugyanakkor a dokumentumvektorok nagyon ritkák. Az eljárásoknak ezt a kettősséget hatékonyan kell kezelniük.
- A dokumentumgyűjtemények nagy mérete miatt a módszereknek hatékonyan kell működniük, és skálázhatónak kell lenniük.

- Lehetőség szerint, az eredmény közvetlen felhasználhatósága érdekében a csoportokat megfelelően jellemző címkével kell ellátni.

Dokumentumok csoportosítása esetén a csoportokat rendszerint *tematikus* hasonlóság alapján alakítjuk ki. Az eljárások alapja a *klaszterhipotézis*, amely szerint a szóhasználatukban hasonló dokumentumok tartalma hasonló. Jelen fejezetben ismertetett módszerek és alkalmazások esetén ezért a klaszterhipotézis teljesülését implicite mindig feltesszük.

A csoportosító algoritmusok kutatásához az információvisszakereső-rendszerek hatékonyságának növelése adta a kezdeti motivációt. Az utóbbi években az internetes keresők támogatása lett a legjellemzőbb alkalmazási terület: klaszterező eljárást alkalmaznak dokumentumgyűjtemények böngészésének támogatására, illetve internetes keresések eredményeinek csoportokba szervezésére. Szintén gyakori alkalmazás az internetes dokumentumok vizsgálata alapján történő automatikus taxonómiagenerálás,[1] továbbá a már meglévő taxonómia-csomópontok dokumentumainak további csoportosítása, amelynek eredményét a taxonómia finomítására lehet felhasználni.

6.1. A csoportosító módszerek típusai

A csoportosítás feladata tehát a dokumentumok csoportokba rendezése valamilyen tulajdonságuk — téma, nyelv, stílus stb. — mentén. A csoportosításnak két alapvetően különböző megközelítése van. Az egyszerűbb *particionáló* módszerek a dokumentumoknak egy lehetséges felosztását adják meg, ahol a csoportok függetlenek egymástól. Ennek a megoldásnak előnye a kedvező számításigény, hátránya a rugalmatlanság, ami abból is következik, hogy a csoportok számát előre rögzíteni kell. Más megközelítést alkalmaznak a *hierarchikus* csoportosító módszerek, amelyek a lehetséges csoportosítások egymásba ágyazott sorozatát adják eredményül. Itt a sorozat elemei eltérő számú csoportot tartalmaznak. A módszercsalád előnye, hogy nagyobb választási lehetőséget ad a megoldásra, hátránya a nagyobb számításigény: míg ugyanis a particionáló módszereknél a futásidő a dokumentumgyűjtemény méretének lineáris, addig a hierarchikus eljárások esetén annak négyzetes függvénye.

A csoportképzés eredménye a csoportbatartozás szempontjából kétféle lehet. A *szigorú* módszerek esetén minden dokumentum pontosan egy csoportba tartozik, míg a *lágy* csoportosító eljárások esetén egy dokumentum több csoportba is tartozhat, a csoportbatartozás mértékét pedig valószínűségi értékekkel jellemzik.

[1] a Yahoo! vagy a Dmoz tematikus rendszerezéséhez hasonló könyvtárstruktúra felépítése

6.2. A csoportosítás alkalmazásai

A legáltalánosabb értelemben vett csoportosítási feladatokkal a hétköznapi életben is számtalan alkalommal találkozhatunk vásárlás, kommunikáció, kapcsolatépítés során — ezekről rövid áttekintés található [2] 4. fejezetének bevezetőjében. Ha a csoportosítandó objektumok dokumentumok, akkor az információ-visszakereséssel kapcsolatos alkalmazások a jellemzőek.

Az internetes keresők többnyire listában adják vissza a találatokat. Mivel a találatok száma rendkívül nagy lehet, a teljes lista áttekintése gyakran lehetetlen. Ilyen esetben a találatok csoportosítása hatékonyan segítheti a felhasználót az eredmények áttekintésében [220]. Ezzel egyrészt könnyebben azonosítható a találati listának a felhasználó számára releváns részhalmaza, azaz az a csoport, amelybe nagyobb eséllyel tartoznak releváns találatok, másrészt ennek a csoportnak az átfutása is sokkal kevesebb időt vesz igénybe. A keresési találatok csoportosítása különösen akkor előnyös, ha a keresőszó többértelmű, vagy több nyelven is értelmes szó. Ilyen megoldást kínál a Vivísimo keresőszolgáltatás ClusterEngine motorja (ld. a 198. oldalt).

Szintén a keresésekkel kapcsolatos a következő alkalmazás. A vektortér alapú IR-rendszereknél futási időben kell a keresőkifejezést vagy -dokumentumot a teljes korpusszal összehasonlítani. Ez nagyméretű korpusz esetén rendkívül időigényes lehet. A keresés ilyenkor hatékonyan gyorsítható az alábbi, a lusta tanulóknál már említett megoldással (ld. a 126. oldalt). Az előfeldolgozási fázisban a korpusz dokumentumait csoportosítjuk, és a keresőkifejezést a csoportok reprezentáns eleméhez hasonlítjuk. Ezután két lehetőség van: (1) az összes olyan csoportot felvesszük a találatok közé, amelynek a reprezentánsa hasonló; (2) kimerítő keresést csak a kiválasztott csoportokon belül végzünk, immár sokkal kevesebb dokumentumon. Az első megoldás akkor célszerű, ha a keresőkifejezés csak néhány szót tartalmaz. Ez azt is biztosítja, hogy olyan találatokat is visszaadjon a kereső, amelyek szemantikailag közeliek a kereséshez,[2] de a keresőkifejezés elemeit nem tartalmazzák — így ezzel a módosítással nő a releváns találatok száma, azaz a felidézés. Az utóbbi módon akkor indokolt eljárni, ha teljes dokumentummal indítjuk a keresést, amellyel sok reprezentánsnak van metszete. Ekkor elég csak a leginkább hasonló néhány reprezentáns klaszterére szűkíteni a keresést.

A keresések találati halmazában történő navigálást támogatja, ha az eredmények böngészésénél rendelkezésre áll egy hierarchikus témastruktúra — taxonómia. A taxonómia manuális létrehozása meglehetősen idő- és költségigényes fel-

[2] Ez akkor teljesül, ha a klaszterhipotézis igaz.

adat. Ezt — legalábbis részben — ki lehet váltani a dokumentumok hierarchikus csoportosításával [46]. Ekkor különösen fontos, hogy a csoportokhoz megfelelő címkét rendeljünk, hiszen ez vezérli a felhasználót a navigálásban (ld. a 6.6. szakaszt). Ezzel a módszerrel a taxonómia bővítése, ill. finomítása is megoldható. Ilyenkor inkább particionáló osztályozást kell végrehajtani a bővítendő csomópont dokumentumain, és célszerű az elvárt csoportszám értékét is megadni.

6.3. Reprezentáció

A dokumentumok reprezentálására a csoportosításnál is a vektortérmodellt használjuk. A legtöbb klaszterező eljárásnál a dokumentumvektorok megadásakor valamelyik normalizált súlyozási sémát követjük (ld. a 2.2.3. pontot, ill. a (2.7) és (2.10) képleteket). Kivételt jelent e tekintetben a valószínűségi modellt használó EM-algoritmus, ahol bináris súlyozást használunk.

A klaszterhipotézisen alapuló csoportosító módszerek a dokumentumok tartalmi hasonlóságát a bennük szereplő szavak együttes előfordulásai alapján határozzák meg. A vektortérmodellben ez a feladat a dokumentumvektorok hasonlóságának meghatározását jelenti. Hasonlósági mértékként a normákon alapuló távolságfüggvények közül az euklideszi- avagy koszinusztávolságot használjuk leggyakrabban (ld. az (5.3) képletet a 112. oldalon).

A csoportosítás esetén általában célszerű szótövezést alkalmazni. Tartalmi csoportosítás esetén a stopszavak elhagyására nem érzékenyek a módszerek. Ha azonban a dokumentumokat nyelv szerint csoportosítjuk, akkor a stopszavak a dokumentum nyelvének fontos indikátorai, ezért nem érdemes kiszűrni őket.

6.4. Particionáló módszerek

Particionáló csoportosítás esetén formálisan egy $\Psi : D \to \{1, \ldots, K\}$ csoportosítófüggvény meghatározása a cél, ahol $D = \{d_1, \ldots, d_N\}$ a dokumentumhalmaz, és K a csoportok előre rögzített száma. A dokumentumok hasonlóságát egy előre definiált *hasonlósági mérték*kel mérjük. A Ψ csoportosítófüggvényt úgy kell megalkotni, hogy optimalizálja a megadott hasonlósági mértéktől függő kiértékelő függvényt. Mivel a legtöbb hasonlósági mértékhez megadható egy távolságmérték, ezért a dokumentumok összehasonlítása során e két fogalom felcserélhető.

A sarkalatos kérdés K értékének megválasztása. Egyes alkalmazások eleve korlátokat szabnak a csoportok számára az eredmények áttekinthetőségére vonatkozó követelménnyel. Például, ha internetes keresések eredményét csoportosítjuk, a célszerű csoportszám 10–20, több csoportot ugyanis nehéz fogalmilag megkülön-

böztetni egymástól. Bizonyos particionáló csoportosítók esetén a K-t heurisztikus módszerrel határozhatjuk meg.

A kiértékelő függvény optimalizálása keresési feladatnak is tekinthető. Mivel azonban N dokumentumot K csoportba $K^N/K!$ módon lehet felosztani, ezért az összes lehetséges csoportosítást kiértékelő primitív módszer nem megvalósítható ($N \gg K$). Ezért a particionáló algoritmusok működési elve az alábbi: egy kezdeti particionálásból kiindulva fokozatosan kívánja elérni az optimális megoldást az alábbi lépések végrehajtásával:

1. Adjuk meg az N dokumentum kezdeti particionálását.
2. A kiértékelő függvénnyel vizsgáljuk meg a csoportosítás jóságát, és lehetőség szerint a dokumentumok áthelyezésével javítsuk a jósági mutató értékét.
3. A 2. lépést addig ismételjük, amíg a terminálási feltétel nem teljesül.

Az algoritmus eredményessége erősen függ a megfelelő kezdeti particionálás megválasztásától. Ha a keresést kedvezőtlen particionálással kezdjük, akkor lehet, hogy nem találjuk meg a globálisan optimális megoldást. Ezért a particionáló klaszterezőknél a kezdeti felosztás jó megválasztása igen fontos kérdés.

6.4.1. A k-átlag módszer

A *k-átlag* (k-means) a legfontosabb particionáló klaszterező. A módszer azon a feltevésen alapszik, hogy egy csoport középponti eleme jól reprezentálja a csoport dokumentumait. A módszer a dokumentumok csoportközépponttól mért távolságnégyzetének átlagát alkalmazza kiértékelő függvényként, ahol a csoport középpontját a csoportba tartozó dokumentumok átlagaként vagy centroidjaként definiálja (vö. (5.4)-gyel):

$$\mathbf{v}(c_j) = \frac{1}{c_j} \sum_{\mathbf{d} \in c_j} \mathbf{d}. \tag{6.1}$$

Vegyük észre, hogy a Rocchio-osztályozónál nagyon hasonló módon számoltuk a centroidot, csak ott az osztálycímkék rendelkezésre álltak, itt viszont nem. Ekkor a kiértékelő függvényt a dokumentum- és centroidvektorok távolságnégyzetének átlagaként, az

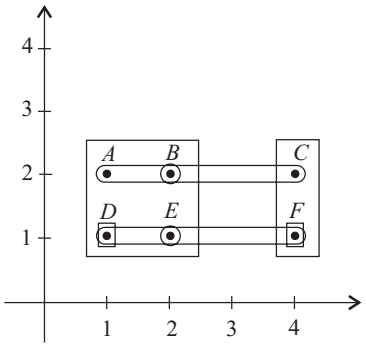
$$\mathrm{RSS} = \sum_{j=1}^{K} \mathrm{RSS}_j, \quad \text{ahol } \mathrm{RSS}_j = \sum_{\mathbf{d} \in c_j} |\mathbf{d} - \mathbf{v}(c_j)|^2 = \sum_{k=1}^{M} (d_{ik} - v_i(c_j))^2 \tag{6.2}$$

kifejezéssel definiáljuk. Az RSS függvény használata a csoportok gömbszerű alakját feltételezi.

A *k*-átlag módszer az alábbi lépésekből áll:

1. Inicializálás: válasszuk ki véletlenszerűen a csoportosítás *mag*jait, a kezdeti particionálást meghatározó K dokumentumot. Ezek lesznek a kezdeti centroidok.
2. Hozzárendelés: rendeljünk hozzá minden $\mathbf{d}_i \in D$ dokumentumot a legközelebbi centroidhoz, azaz amire $d(\mathbf{d}_i, \mathbf{v}(c_j))$ minimális.
3. Újraszámolás: Számítsuk ki a kialakult csoportok centroidját (6.1) szerint.
4. Továbblépés vagy terminálás: A 2. és 3. lépést addig ismételjük, amíg a terminálási feltétel nem teljesül.

Az algoritmus terminál, ha a csoportok vagy a centroidok nem változnak, illetve elérünk egy rögzített iterációszámot. Belátható, hogy az RSS értéke az algoritmus működése során monoton csökken, ezért a *k*-átlag konvergál, és az első két leállási feltétel alkalmazása esetén is terminál. Az RSS nem nő a hozzárendelési lépés során, mert a legközelebbi centroidhoz rendelünk minden dokumentumot, és az újraszámolási lépésnél sem nőhet az értéke, mivel az új centroid minimalizálja RSS_j értékét (ld. még [125]-öt). A terminálás biztosítása érdekében konzisztens módon kell a dokumentumtól azonos távolságra lévő centroidok közül az egyiket kiválasztani, ellenkező esetben az algoritmus végtelen ciklusba kerülhet azonos RSS értékű megoldások között lépegetve. Erre jó megoldás, ha pl. a dokumentumot mindig a legkisebb indexű csoporthoz rendeljük. Bár a *k*-átlag módszer konvergál, a globális optimum elérése nem biztosított. A 6.1. ábra példáján rosszul megválasztott magok esetén az algoritmus nem találja meg a legjobb csoportosítást.



6.1. ábra. A B és E magok választása esetén az $[ABC]$, $[DEF]$, a D és F magok esetén pedig az $[ABDE]$, $[DF]$ csoportosításhoz konvergál a *k*-átlag. Ezek közül az utóbbi megoldás az optimális [125]

A magok kiválasztásánál fontos például elkerülni, hogy izolált pontok legyenek közöttük, ezek ui. egyelemű csoportokat eredményezhetnek. Erre az egyik módszer az, hogy több kiinduló maghalmazt választunk, és a legjobb RSS értéket nyújtó csoportosítást vesszük eredményül. A magok kiválasztását hierarchikus csoportosítással is segíthetjük: néhány K dokumentumra alkalmazva ezt a módszert, jó magokat kaphatunk. Magok gyanánt nem feltétlenül kell dokumentumokat választani.

Nézzük meg a k-átlag módszer bonyolultságát. A hozzárendelési lépésben KN távolságot $O(KNM)$ komplexitással számoljuk, az újraszámolási lépésben $O(NM)$ költségű az N darab M-dimenziós vektor összeadása. Mindez az I iterációra $O(IKNM)$ nagyságrendű, azaz a k-átlag a csoportok és a dokumentumok számában, valamint a dimenzióméretben is lineáris. A gyakorlati tapasztalatok azt mutatják, hogy az iterációk száma általában mérsékelt, és a módszer gyorsan konvergál.

A K, N és M tényezők közül általában az utóbbi nagyságrendje a domináns. Ritka vektorok különbsége esetén csak a jellemzők egy töredékére kell a műveletet végrehajtani. A centroidok azonban *sűrűek*, mivel a centroidvektorban a csoportban előforduló minden szónak nem nulla az értéke, ezért alapesetben ténylegesen M-mel lesz arányos a távolság számítása. Ezen azonban egyszerű heurisztikával könnyen segíthetünk. Ha pl. csak a legfontosabb 1000 jellemzőt tartjuk meg a centroidvektorban, akkor a csoportosítás minősége alig változik, míg a távolságszámítás felgyorsul.

Egy másik módszer ugyanennek a problémának a kezelésére az, hogy a csoportok reprezentáns elemeként nem a centroidot, hanem a *medoid*ot, a centroidhoz legközelebb eső dokumentumot használjuk. Ekkor két ritka vektor távolságát kell számolni. Ezt a csoportosító módszert *k-medoid*nak nevezik.

Ha nincsenek alkalmazásfüggő korlátok vagy többletinformáción alapuló becslés K célszerű értékére, akkor heurisztikus módon kell meghatározni. Az RSS K szerinti minimalizálása nem célravezető, mert ennek minimuma $K = N$, azaz ha minden dokumentum külön csoportot képez. Egy egyszerű heurisztika pl. különböző K-kra RSS csökkenésének mértékét vizsgálni, és ahol leginkább esik az érték, azt a K-t választani. Kifinomultabb alternatíva egy olyan kombinált kiértékelő függvény használata, amely K magas értékét bünteti. Erre többféle információelméleti feltételt szoktak alkalmazni, ld. [125].

6.4.2. További particionáló módszerek

6.4.2.1. Kettészelő k-átlag módszer

A *k*-átlag módszer egy továbbfejlesztett változata a *kettészelő k-átlag* (bisecting *k*-means) eljárás [179]. Az algoritmus a teljes dokumentumhalmazból indul ki, és a következő lépésekből áll:

1. Inicializálás: Válasszunk ki egy felosztandó klasztert.
2. Kettészelés: Osszuk ezt fel két részre a 2-átlag eljárással.
3. Végezzük el a 2. lépést I-szer, különböző magokból kiindulva, és válasszuk ki azt a vágást, amelyikre RSS maximális.
4. Ismételjük meg a fenti 3 lépést, ameddig a szükséges csoportszámot nem érjük el.

Az első lépésben felosztandó klaszterként pl. a legnagyobb méretű vagy a legnagyobb RSS_j értékű csoportot választhatjuk.

A kettészelő *k*-átlag módszer alkalmas mind particionáló, mind hierarchikus csoportok létrehozására. Mint minden hierarchikus klaszterezőnél, itt is lehetőség van a csoportosítás oly módon való finomítására, hogy az eredményül kapott klaszterekből kiindulva a *k*-átlag eljárást lefuttatjuk. Az eljárás időigénye — finomítással is — lineáris a dokumentumok számának függvényében.

6.4.2.2. Az EM-algoritmus

A reprezentáns elemtől mért távolságok minimalizálásától lényegesen eltérő megközelítést használ a valószínűségi modellre építő *EM-algoritmus* (Expectation Maximization). Ez a módszer a csoportosítást mint egy Θ parametrikus modellt tekinti, és azokat a paramétereket keresi, amelyek a legnagyobb valószínűséggel generálják a D dokumentumkorpuszt.

A Bayes-elméletre épülő EM-algoritmus lágy csoportokat képez, azaz egy dokumentum – különböző mértékben — több csoportba is tartozhat, ahol a mértéket a $P(c_j|\mathbf{d})$ becslés adja meg. Így pl. a 2012-es foci EB-döntést kommentáló dokumentum egyszerre tartozhat a *Politika* és a *Sport* csoportokba is. A módszer abból a szempontból is rugalmasabb a *k*-átlag és a hierarchikus klaszterező eljárásoknál, hogy lehetőséget ad szakterület-specifikus tudás beépítésére a modellbe. Ahogy említettük, a *k*-átlag módszer gömb alakú klasztereket feltételez, ezzel szemben az EM-algoritmus a dokumentumok eloszlásáról rendelkezésre álló adatokat (pl. binomiális vagy normális) a modellalkotásnál figyelembe tudja venni.

Az EM-algoritmus a naiv Bayes-módszerhez (ld. az 5.4.3. pontot) hasonlóan feltételezi a dokumentumokban előforduló szavak feltételes függetlenségét, hogy a modell paramétereinek száma kezelhető nagyságrendű legyen. A módszer vázlata hasonló a *k*-átlagéhoz, az E- és az M-lépések iteratív váltogatását végzi, ahol az E-lépés megfelelője a hozzárendelés, az M-lépésé pedig az újraszámolás (ld. a 150. oldalt). Terjedelmi okok miatt a módszer részletes ismertetését a könyv internetes melléklete tartalmazza.

A gyakorlati tapasztalatok azt mutatják, hogy az EM-algoritmus különösen érzékeny a magok megválasztására, rossz magok esetén lokális minimumban ragad. Ezért a *k*-átlagnál már bevezetett módon több magkészlettel indítják el a módszert, és ezek közül választják ki a legjobb csoportosítást produkálót. A módszer legnagyobb előnye, hogy lehetőséget nyújt a modellalkotásnál a korpusszal kapcsolatos feltételezések beépítésére, amelyek az eredmények tükrében igazolódhatnak vagy módosításra szorulhatnak.

6.5. Hierarchikus csoportosítók

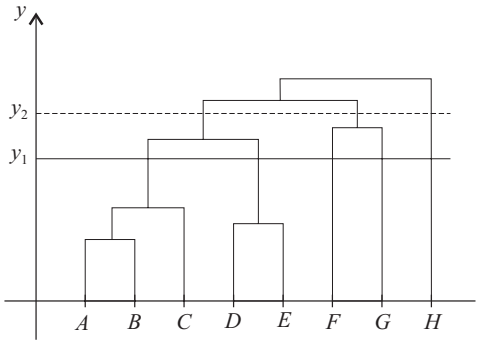
6.5.1. Egyesítő és felosztó módszerek, illusztráció

A hierarchikus eljárásoknak két altípusa létezik, amelyek abban különböznek, hogy a csoportok egymásba ágyazását eltérő sorrendbe végzik. Az *egyesítő*, avagy *agglomeratív hierarchikus csoportosító módszer*ek kiinduláskor a dokumentumokat különálló csoportokként kezelik, és lépésenként a két leghasonlóbb csoportot egyesítik, amíg az összes dokumentum egy nagy csoportot nem alkot. A *felosztó*, avagy *divizív hierarchikus csoportosító módszer*ek által alkalmazott sorrend fordított.[3] A teljes dokumentumgyűjteményt tartalmazó csoportból indulnak ki, és minden lépésben két részre osztanak egy csoportot egészen addig, amíg minden dokumentum külön csoportba nem tartozik. Az egyesítő megközelítés *alulról-felfelé* (bottom-up), a felosztó pedig *felülről-lefelé* (top-down) építi fel a csoportok egymásba ágyazott hierarchiáját.

A hierarchikus csoportosító módszerek működését *dendogramm* segítségével illusztrálják (ld. a 6.2. ábrát). Két csoport egyesítését (felosztását) vízszintes vonal jelöli, amelyből lefele kiinduló két függőleges vonal az egyesített csoportokat (két részre osztott csoportot) köti össze. Az *y*-tengely vonatkozó értéke az egyesített (felosztott) csoportok *együttes hasonlóság*át vagy a lépések számát jelöli. Az előbbi esetben súlyozott, az utóbbiban súlyozatlan dendogrammról beszélünk.

[3] A két módszertípusra a fejezet további részében *egyesítő* és *felosztó* jelzővel hivatkozunk, a hierarchikus szót, ha ez nem zavaró, az egyszerűség kedvéért elhagyjuk.

Az y-tengelyen az algoritmus működésének megfelelő irányba mozogva rekonstruálhatók a csoportosítás előzményei. Mivel egyesítő módszerek esetén a leghasonlóbb csoportokat vonjuk össze, felosztó esetben pedig a leginhomogénebb csoportot osztjuk ketté, a két megközelítés egymásnak nem inverze. Ezért az általuk generált dendogrammok nem lesznek azonosak, hiszen semmi sem garantálja, hogy az egyesített két csoport a következő lépésben a leginhomogénebb csoport legyen. Hierarchikus csoportosításból mindig könnyen kaphatunk particionáló csoportosítást, ha valamely y érték mentén elvágjuk a dendogrammot. Hasonlósági skálával ellátott dendogramm esetén célszerű úgy megválasztani a vágást, hogy lehetőleg olyan szinteket vágjon el egymástól, amelyek között nagy ugrás van.



6.2. ábra. Hierarchikus csoportosítás illusztrálása dendogrammal; az y_1 vágás az $[ABC]$ $[DE]$ $[F]$ $[G]$ $[H]$, az y_2 vágás az $[ABCDE]$ $[FG]$ $[H]$ csoportosítást generálja

Egyesítő csoportosítás esetén egy adott lépésben kevesebb lehetőség közül kell kiválasztani a legjobbat, mint a felosztó csoportosításnál. Míg ugyanis az előbbi esetben k számú csoportnál $\binom{k}{2}$ lehetőség van a csoportok összevonására, addig felosztó esetben $O(k \cdot 2^{|\overline{c}|})$, ahol $|\overline{c}|$ az átlagos csoportméret. Kiinduláskor a kifejezés pontos értéke $\frac{2^N-1}{2}$ (összes lehetséges részhalmaz és komplementere). Mivel az egyesítő módszerek — nem utolsó sorban az egyszerűbb egyesítő lépésnek köszönhetően — a gyakorlatban elterjedtebbek, ezért itt ezekre fókuszálunk.

6.5.2. Egyesítő módszerek

Az egyesítő módszerek a csoportok összevonására a mohó algoritmust követik: mindig a két leghasonlóbb csoportot vonják össze. Ennek az a — dendogrammon is látható — következménye, hogy az együttes hasonlóságok, $s_1 \geq s_2 \geq \ldots \geq s_{N-1}$, monoton csökkennek. Ha a monotonitási feltétel ugyanis nem áll fent, akkor az

egyesítő módszer legalább egy *inverzió*t tartalmaz, azaz létezik olyan i, amire $s_i < s_{i+1}$ — ez bizonyos hasonlósági függvények használata esetén előfordul.

Az egyesítő módszerek az alábbi lépéseket hajtják végre:

1. Inicializálás: hasonlósági mátrix kiszámolása az N kiinduló dokumentumra. Legyen $c_i = \{d_i\}$, $i \in [1,N]$. Az aktív és összevont csoportok nyilvántartására szolgáló tömb inicializálása.
2. Egyesítés: a leghasonlóbb két aktív csoport kiválasztása,
$$(i,j) = \underset{i \neq j,\ \text{aktívak}}{\arg\max}\ s(c_i,c_j). \tag{6.3}$$
3. Újraszámolás: hasonlósági mátrixban a c_i-hez tartozó oszlopok és sorok újraszámolása és c_j deaktiválása.
4. A 2. és 3. lépést $N-1$-szer kell végrehajtani, amíg az összes dokumentumot egy csoportba vonjuk össze.

Ha két csoportpár azonos hasonlósági értékkel bír, akkor véletlenszerűen választjuk ki közülük az összevonásra kerülőket.

6.5.2.1. Hasonlósági mértékek

Attól függően, hogy a (6.3) kifejezésben a hasonlóságot hogyan számítjuk ki, az alábbi négy egyesítő hierarchikus módszert különböztetjük meg: *egyszerű kapcsolódás* (single-link), *teljes kapcsolódás* (complete-link), *átlagos kapcsolódás* (group-average), és *centroid kapcsolodás*.[4]

Az *egyszerű kapcsolódás* módszere azokat a csoportokat vonja össze, amelyeknek a legközelebbi pontjai közötti távolság a legkisebb:

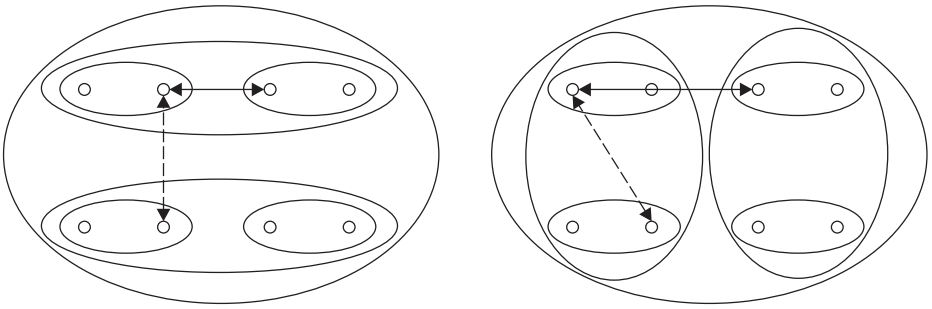
$$s_{\mathrm{SL}}(c_i,c_j) = \max_{d_k \in c_i, d_\ell \in c_j} (s(d_k,d_\ell)) = \min_{d_k \in c_i, d_\ell \in c_j} \langle \mathbf{d}_k, \mathbf{d}_\ell \rangle. \tag{6.4}$$

A *teljes kapcsolódás* esetén az új csoport azon két klaszter összevonásával keletkezik, ahol a legtávolabbi pontok közötti távolság a legkisebb:

$$s_{\mathrm{CL}}(c_i,c_j) = \min_{d_k \in c_i, d_\ell \in c_j} (s(d_k,d_\ell)) = \max_{d_k \in c_i, d_\ell \in c_j} \langle \mathbf{d}_k, \mathbf{d}_\ell \rangle. \tag{6.5}$$

Míg az első esetben az összevonási kritérium lokális, azaz a csoportoknak csak az egymáshoz legközelebb eső részétől függ, a teljes hasonlóság mértéke globális, a csoportok összes eleme beleszámít az összevonási döntésbe. A két mérték használatát illusztrálja a 6.3. ábra.

[4] ennek analóg változata a *medoid kapcsolódás*



6.3. ábra. A bal, ill. jobb oldali ábra az egyszerű és a teljes kapcsolódás mértéket alkalmazó egyesítő csoportosítás működését mutatja [125]

Az egyszerű és a teljes kapcsolódás módszere csak egy-egy dokumentum hasonlósága, ill. távolsága alapján dönt. Az *átlagos kapcsolódás* a két csoport összes dokumentumának átlagos távolsága alapján számítja ki a csoportok hasonlóságát, amelyben a csoporton belüli dokumentumok távolságát is figyelembe veszi — azaz az összevont csoport belső hasonlóságát számolja ki:

$$s_{\mathrm{GA}}(c_i,c_j) = \frac{1}{(|c_i|+|c_j|)(|c_i|+|c_j|-1)} \sum_{d_k,d_\ell \in c_i \cup c_j, c_k \neq c_\ell} \langle \mathbf{d}_k, \mathbf{d}_\ell \rangle$$
$$= \frac{1}{(|c_i|+|c_j|)(|c_i|+|c_j|-1)} \left(\left(\sum_{d_k \in c_i \cup c_j} \mathbf{d}_k \right)^2 - (|c_i|+|c_j|) \right). \tag{6.6}$$

Hasonló elvet követ a *centroid kapcsolódás*, amely azt a két csoportot vonja össze, amelyek centroidja a legközelebbi. A centroidok hasonlóságát a

$$s_{\mathrm{C}}(c_i,c_j) = \frac{1}{|c_i|\,|c_j|} \sum_{d_k \in c_i, d_\ell \in c_j} \langle \mathbf{d}_k, \mathbf{d}_\ell \rangle = \frac{1}{|c_i|\,|c_j|} \left(\sum_{d_k \in c_i} \mathbf{d}_k \right) \cdot \left(\sum_{d_\ell \in c_j} \mathbf{d}_\ell \right) \tag{6.7}$$

függvény adja meg. A medoid kapcsolódás analóg módon a medoidokra számolja ki ugyanezt. A (6.6) és (6.7) képletek gyorsan számolhatók, mivel az egyes vektorok hasonlóságának átlaga megegyezik a centroidok hasonlóságával. A két mérték között az egyik leglényegesebb különbség, hogy az átlagos kapcsolódás nem veszi figyelembe az „önhasonlóságokat”. E két mértéket csak a dokumentumok vektortérmodell szerinti reprezentációja esetén lehet alkalmazni, míg az egyszerű és a teljes kapcsolódásnál általánosabb reprezentáció is lehetséges.

A 155. oldalon ismertetett naiv algoritmusvázlat költségigénye $O(N^3)$, mivel az $N-1$ lépés mindegyikében a teljes $N \times N$-es hasonlósági mátrixon kimerítő keresést hajtunk végre. Ennél lényegesen kisebb költséggel is megvalósítható az egyesítő algoritmus.

Egyszerű kapcsolódás esetén vezessünk be egy tömböt, amely minden csoportra tartalmazza, hogy melyik a *legközelebbi szomszéd*ja. Ezt a tömböt az eljárás elején $O(N^2)$ költséggel inicializálhatjuk, és minden összevonás után $O(N)$-nel frissíthetjük. Ezután elég ebben a tömbben keresni a legkisebb elemet, amely szintén $O(N)$ költségű. Tehát összességében $O(N^2+N-1(N+N)) = O(N^2)$, azaz egy teljes nagyságrenddel csökkenthető ekkor a költségigény.

A többi módszernek nagyobb a költségigénye a következőek miatt. Míg a legközelebbi szomszéd tömb esetén egy klaszterre a frissítés egy maximumművelettel elvégezhető[5] — az összevont csoportok közül melyik volt közelebb az aktuális csoporthoz —, addig a csoportképzés a többi módszernél az összevont és a többi csoportok közötti értékek újraszámolását igényli. Ekkor az alábbi módszer a leghatékonyabb. Vezessünk be egy rendezett prioritási sorokat tartalmazó tömböt, ahol az egyes sorok az adott csoportra tartalmazzák a többi csoport indexét, hasonlóság szerint csökkenő sorrendben. A rendezett sorok tömbjének hasonlósági mátrix alapján történő inicializálása $O(N^2 \log N)$ költségű. Az optimális csoportpárt minden lépésben a prioritási sorok első maximális elemének meghatározásával kapjuk meg — ez $O(N)$ lépés. Ezután az összes prioritási sorban aktualizálni kell az összevont csoportokhoz tartozó értéket. Az egyik (reprezentánsnak választott) csoport esetén újra kell számolni a hasonlóságot, és a megfelelő helyre be kell illeszteni a prioritási sorba, a másik csoporthoz tartozó értéket pedig törölni kell. Ez az N prioritási sorra összesen $O(N \log N)$ műveletet jelent egy összevonásnál, összesen tehát $O(N^2 \log N)$ lesz a költség.

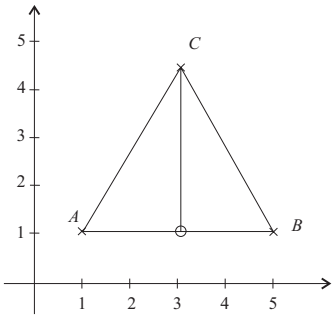
6.5.2.2. Hasonlósági mértékek elemzése

Az egyszerű kapcsolódás mértéke jól alkalmazható, ha a dokumentumok tömör, egymástól jól elkülönülő, tetszőleges alakú csoportokat képeznek. Ez a módszer nem érzékeny az izolált pontokra, képes ezeket is jól felismerni, viszont hajlamos hosszan elnyúló csoportokat alkotni (ld. a 6.3. ábra bal oldalát), amit *láncolás*i effektusnak nevezünk. A teljes kapcsolódás élesen elhatárolt, ellipszoid alakú, illetve tömör klaszterek felismerésére képes. Az izolált pontokra különösen érzé-

[5] Ez annak a következménye, hogy a legközelebbi csoport tulajdonság *perzisztens*, azaz ha c_i, c_j az optimális pár, és c_i-t c_k-val vonjuk mégis össze, akkor $c_i \cup c_k$-hoz is c_j lesz a legközelebbi csoport.

keny: ezek csoportba vonásával hajlamos óriási átmérőjű csoportokat képezni, és teljesen hibás eredményt adni végső csoportosításként.

A centroid kapcsolódás által használt hasonlósági mérték nem köthető közvetlenül az összevont csoport együttes hasonlósági mértékéhez. A másik három módszernél direkt kapcsolat áll fenn az együttes hasonlóság és az összevonásnál használt kritérium közt. Ezeknél ugyanis az új csoport felbontható az eredeti csoportokra annak alapján, hogy (1) bármely pontra a legközelebbi szomszédjától való távolság maximális (egyszerű), (2) bármely két pont csoporton belüli távolsága maximális (teljes), (3) a csoporton belül az átlagos távolság minimális (átlagos). Ezzel szemben a centroid kapcsolódással összevont csoportoknál, ha különböző módon vágjuk kétfelé a csoportot, akkor mindig más lesz a centroidok távolsága. Ezért a centroid kapcsolódással generált csoportosítás nem reprodukálható. További hátránya, hogy felléphet az *inverzió* jelensége (ld. 6.4. ábra), azaz nem garantálható egy adott hasonlósági küszöbnél vett csoportosítás optimalitása. Ugyanez jellemzi a medoid kapcsolódást is.



6.4. ábra. Az A,B,C pontok koordinátái rendre $(1+\varepsilon,1)$, $(5,1)$, és $(3,1+2\sqrt{3})$, azaz a három pont majdnem egyenlő oldalú háromszöget alkot, de A és B közelebb van egymáshoz. Az első csoportképzésnél az összevont csoportok távolsága ≈ 4. Második esetben $[AB]$ és $[C]$ távolsága csupán $\approx 2\sqrt{3} \approx 3{,}46 < 4$, azaz a hasonlóság nő [125]

Bonyolultág szempontjából az egyszerű kapcsolódás a legkedvezőbb tulajdonságú hasonlósági mérték. Az átlagos és centroid kapcsolódás esetén további nehézséget jelent, hogy nem érdemes előre kiszámolni az összes lehetséges csoporthasonlóságot, mivel azok nem egy-egy dokumentum távolságán alapulnak, mint az egyszerű és teljes kapcsolódás esetén, hanem az összes csoportbeli dokumentumot figyelembe veszik.

6.1. táblázat. Egyesítő hierarchikus klaszterezők összehasonlítása ([125] alapján)

Módszer	hasonlósági mérték	költség	megjegyzés
egyszerű	legközelebbi két pont maximuma	$O(N^2)$	láncolási effektus
teljes	legtávolabbi két pont minimuma	$O(N^2 \log N)$	érzékeny az izolált pontokra
centroid	centroidok hasonlósága	$O(N^2 \log N)$	nem rekonstruálható, inverzió
átlagos	átlagos hasonlóság	$O(N^2 \log N)$	

A mértékek felsorolt szempontok szerinti összehasonlítását a 6.1. táblázat tartalmazza. Ennek alapján az általánosan leginkább használható módszer az átlagos kapcsolódás. Gyakorlati esetekben az $O(\log N)$ faktorral nagyobb időigény nem szokott lényeges sebességi különbséget okozni, ugyanakkor ez a módszer nem rendelkezik jelentős hátránnyal.

6.6. Csoportok címkézése

A csoportosítás legtöbb gyakorlati alkalmazásánál — különösen fontos ez az elemző és a felhasználói felülettel kapcsolatos feladatok esetén — az eredményt emberek dolgozzák fel. Ilyenkor a csoportokat fel kell *címkézni*, hogy a felhasználók tudják, milyen dokumentumokat tartalmazhatnak az egyes csoportok. Ez a feladat hasonló a kivonatoláshoz (ld. a 7. fejezetet), de míg ott egyes dokumentumok tartalmi összegzését kell létrehozni, itt teljes dokumentumok csoportjára kell ezt a feladatot elvégezni.

Az egyik legegyszerűbb heurisztika ilyenkor a csoportot reprezentáló dokumentum(ok) — pl. a medoid, vagy a centroidhoz legközelebbi néhány dokumentum — teljes vagy rövidített címét választani a csoport címkéjének. Mivel a szerzők többnyire a cikk tartalma alapján adnak címet a dokumentumoknak,[6] ezért a címek jó leírásnak bizonyulnak.

A csoportban magas előfordulási értékkel rendelkező szavak és kifejezések — különösen a főnevek, ill. főnévi csoportok — halmaza hasonlóan alkalmas a csoport tartalmának jellemzésére. A stopszavakat ilyenkor természetesen szűrni kell.

[6] A szépirodalomban erre persze számos ellenpélda akad, pl. *Ahogy tetszik*, *Se dobok, se trombiták*.

Webdokumentumok esetén a dokumentumra mutató linkek szövegei, a horgonyok szintén felhasználhatók a csoportra jellemző szavak kigyűjtésénél.

Az eddig felsorolt heurisztikák csak a csoporton belüli dokumentumokat veszik figyelembe. Így előfordulhat, hogy olyan szavakat választanak egy csoport jellemzésére, amelyek több csoportra is jellemzőek, sőt, esetleg korpuszspecifikus stopszavak, azaz szinte az összes dokumentumban előfordulnak (pl. számítógépes szövegekben a *computer* szó).

Az olyan címkéző módszereket, amelyek egy csoport címkézésénél a többi csoport jellemző szavait is figyelembe veszik *differenciális csoportcímkézés*nek nevezzük. Erre a célra a 2.3.1. pontban ismertetett jellemzőkiválasztó módszerek — pl. az információnyereség, a χ^2-statisztika, ill. a kölcsönös információ — egyaránt alkalmasak, mivel ezek olyan szavakat adnak meg, amelyek *megkülönböztetik* az egyes csoportokat.

6.2. táblázat. Címkézési módszerek összehasonlítása ([125] alapján)

Dokumentumok száma	Címkézési módszer előfordulás alapú	kölcsönös információ	cím
622	000 million newsroom wednesday company tuesday year percent thursday state	plant oil production barrels crude bpd mexico dolly capacity petroleum	MEXICO: Hurricane Dolly heads for Mexico coast
1017	told thursday people year wednesday tuesday government president friday week	police killed military security peace told troops forces rebels people	RUSSIA: Russia's Lebed meets rebel chief in Chechnya
1259	000 prices 20 10 traders market september 50 00 price	delivery traders futures tonne tonnes desk wheat prices 000 00	USA: Export Business – Grain/oilseeds complex

A 6.2. táblázat az RCV1-korpusz (ld. a 143. oldalon) egy kis részhalmazán végzett k-átlag csoportosítás ($K = 10$) automatikus címkézését mutatja 3 csoport esetén [125]. A χ^2-statisztika és a kölcsönös információ módszer gyakorlatilag azonos eredményt adott, ezért csak az utóbbit tüntetjük fel. A táblázatban jól lát-

szik, hogy az előfordulás alapú megközelítés hátránya az, hogy számos általános jelentésű, megkülönböztető jelleggel nem bíró szót ad meg (pl. *tuesday, year, week*). A medoid dokumentumok címkéi sem mindig jól adják vissza a csoportban lévő dokumentumok közös szemantikai tartalmát; jó példa erre az olajjal kapcsolatos dokumentumoknál a hurrikánról szóló cím. Ezek előnye viszont, hogy könnyebben olvashatóak és értelmezhetőek, mint a szólista jellegű eredmények.

Hierarchikus csoportok esetén különösen nehéz a csoportokat jól jellemző, és őket egymástól megkülönböztető címkék kiválasztása. Ekkor ugyanis egy csoportot címkéjével nem csak az azonos szinten lévő (testvér)csoportoktól, hanem a hierarchiában felette, ill. alatta lévő szülő- és gyermekcsoportjaitól is meg kell különböztetni. Ekkor tehát a teljes csoporthierarchiát figyelembe kell venni a címkék meghatározásánál.

6.7. A csoportosító módszerek elemzése

6.7.1. A hatékonyság mérése

A csoportosítás minőségét két típusú mértékkel lehet vizsgálni. Az első típusba az ún. *belső mérték*ek tartoznak, amelyek nem használnak fel külső tudást a csoportosítás jóságának meghatározásához. A második típusba a *külső mérték*ek tartoznak, amelyeket akkor lehet alkalmazni, ha rendelkezésre állnak a dokumentumok osztálycímkéi; ekkor ezeket hasonlítjuk össze a klaszterező által meghatározott csoportokkal.

A belső mértékek a csoportosításnál alkalmazott kiértékelő- és hasonlósági függvények, amelyek az alkalmazott távolságfüggvény szerint optimalizálják a *csoporton belüli közelség*et és a *csoportok közti távolság*ot. Ez azonban nem garantálja, hogy a gyakorlati alkalmazás szempontjából is jó lesz a csoportosítás eredménye — elképzelhető például, hogy a klaszterhipotézis nem teljesül, vagy K értékét határozzuk meg rosszul. Éppen ezért célszerű külső mértékeket is bevonni a kiértékelésbe.

Az egyik kézenfekvő lehetőség a csoportosító algoritmusok tesztelésére, ha olyan dokumentumgyűjteményt használunk, amely rendelkezik osztálycímkékkel.[7] Ekkor a klaszterezés hatékonyságát annak alapján adjuk meg, hogy a doku-

[7] Ideális esetben olyan gyűjteményt kell használni, amelyet több szakértő címkézett fel, és csak az egységesen címkézett dokumentumok szerepelnek a korpuszban. Ezzel csökkenthető a szubjektivitásból eredő hiba.

mentumok osztálycímke szerinti particionálását a csoportosításnak milyen mértékben sikerül reprodukálnia.[8]

Az egyik legegyszerűbb mérőszám erre a *tisztaság* (purity) mérték, amelyet akkor lehet alkalmazni, ha a csoportok száma adott. Legyen az osztálycímkék által meghatározott partíció a $c_1, \ldots, c_{|C|}$, a csoportosítás eredménye pedig a $\tilde{c}_1, \ldots, \tilde{c}_K$ dokumentumhalmazok. Ekkor egy csoportra, illetve a teljes csoportosításra a

$$\rho(\tilde{c}_k) = \frac{1}{|\tilde{c}_k|} \max_{j \in [1,|C|]} |c_j \cap \tilde{c}_k|, \quad \rho = \sum_{k=1}^{K} \frac{1}{|\tilde{c}_k|} \rho(\tilde{c}_k) \tag{6.8}$$

kifejezésekkel számolhatjuk a tisztaság mértékét. Magas $\rho(\tilde{c}_k)$ érték azt mutatja, hogy a csoport valamelyik c_j csoportnak „tiszta" részhalmaza. Ezért a ρ értéke a maximális 1 lesz akkor is, ha minden dokumentum külön csoportba kerül, így a tisztaság az optimális K meghatározására nem alkalmazható.

Egy másik lehetőség a csoportok átlagos entrópiájának mérése. Egy csoportra az entrópiát a

$$H(\tilde{c}_k) = -\frac{1}{\log |C|} \sum_{j=1}^{|C|} \frac{|c_j \cap \tilde{c}_k|}{|\tilde{c}_k|} \log \left(\frac{|c_j \cap \tilde{c}_k|}{|\tilde{c}_k|} \right) \tag{6.9}$$

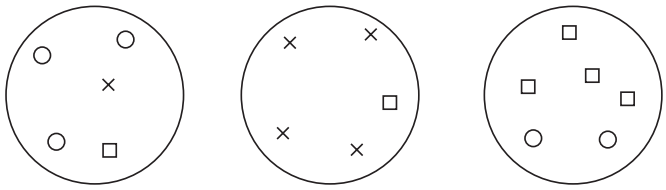
kifejezés adja, ahol $\frac{|c_j \cap \tilde{c}_k|}{|\tilde{c}_k|}$ a $P(c_j \cap \tilde{c}_k)$ valószínűség becslése. A (6.9) értékek $k = 1, \ldots, K$ csoportokra számolt, csoportmérettel súlyozott átlagaként kapjuk az átlagos entrópiát. A csoportosítás annál jobb, minél kisebb az entrópiája. Ha egy csoportra az entrópia értéke 0, akkor azt jelenti, hogy a csoport teljes egészében egy osztály része, ha pedig 1, akkor az osztálycímkék egyenletes keverékét tartalmazza. Az elkülönülő csoportok büntetésére ez a módszer tehát ugyancsak nem alkalmazható.

Ha a csoportosítás feladatát a dokumentumok páronkénti osztályba tartozása alapján értékeljük, akkor a hamis pozitív (FP) döntés két különböző osztályba tartozó dokumentum azonos csoportba rendelése, a hamis negatív (FN) pedig két azonos osztályba tartozó dokumentum különböző csoportba rendelését jelenti. Az eddigi mértékek csak a hamis pozitív eseteket büntették.

Mindkét fajú hibát figyelembe veszi a szokásos IR-mértékek közül a *szabatosság* (3.4) és az *F-mérték* (3.2). Az előbbit csoportosítás esetén *Rand-indexnek* is

[8] Ez a vizsgálati módszer figyelmen kívül hagyja azt, hogy az osztályozásnak és a csoportosításnak eltérő céljai lehetnek. Míg az osztályozás taxonómiája egy permanens és általánosan használható navigációs eszközt kíván nyújtani, addig a csoportosításnál sokkal inkább figyelembe kell venni a korpusz specialitásait.

nevezik. A gyakorlatban e két mérték használata terjedt el, mivel ezek a hatékonyság meghatározásánál a csoportok számát is figyelembe veszik.



6.5. ábra. Csoportosítás hatékonyságának vizsgálata. A keresztek, karikák és négyzetek az eredeti osztálycímkéket jelzik

6.1. PÉLDA. A hatékonysági mértékek használatát a 6.5. ábra példáján illusztráljuk. Ekkor a tisztaság a három csoportra: $\rho(\tilde{c}_1) = \frac{3}{5}$, $\rho(\tilde{c}_2) = \frac{4}{5}$, $\rho(\tilde{c}_3) = \frac{4}{6}$. Az átlagos tisztaság tehát $\rho = \frac{5}{16} \cdot \frac{3}{5} + \frac{5}{16} \cdot \frac{4}{5} + \frac{6}{16} \cdot \frac{4}{6} = \frac{33}{48} = 0{,}6875$.

A három csoportra az entrópiaértékek (6.9) alapján az alábbiak: $\rho(\tilde{c}_1) \approx 0{,}285$, $\rho(\tilde{c}_2) \approx 0{,}189$, $\rho(\tilde{c}_3) \approx 0{,}275$. Az egyesített entrópia: $0{,}251$.

A TP, FP, TN és FN értékeket az alábbiak szerint számolhatjuk. A három csoportban az összes párok száma: $\mathrm{TP}+\mathrm{FP} = 2 \cdot \binom{5}{2} + \binom{6}{2} = 35$. A helyes párok száma $\mathrm{TP} = \binom{3}{2} + 2 \cdot \binom{4}{2} + \binom{2}{2} = 16$, ebből $\mathrm{FP} = 19$. A hamis negatívok száma rendre a kereszt, kör és négyzettel jelölt osztályokra: $\mathrm{FN} = 4+6+8 = 18$. A hamis negatív párok száma: 66. Ennek alapján a Rand-index, $\mathrm{RI} = \frac{16+66}{16+66+18+19} \approx 0{,}689$. A pontosság és a felidézés értéke rendre $P = \frac{16}{35} \approx 0{,}457$, $P = \frac{16}{34} \approx 0{,}471$, ezért $F_1 \approx 0{,}464$.

További entrópia alapú hatékonysági mérték a *perplexitás*, $PPL = 2^H$, amelyet főleg beszédfelismerésnél alkalmaznak [102]. Csoporthierarchiák összehasonlítására is alkalmas mértékeket a [43] munka definiál.

6.7.2. Dokumentumgyűjtemények

A 6.3. táblázatban a csoportosító algoritmusok elemzésére használt dokumentumgyűjtemények adatait foglaltuk össze [179]. A RE0 és RE1 korpusz a már ismertetett Reuters-adatok (ld. 137. oldal) részhalmazaként állt elő. A TR31 és TR45 a TREC-gyűjteményben találhatóak, a kategóriák címkéi pedig az ugyanott

megadott fontossági értékek alapján adhatók meg.[9] Szintén TREC gyűjtemény az FBIS, illetve az LA1 és LA2, amelyek rendre a Foreign Broadcast Information Service és a Los Angeles Times kollekciók adatait tartalmazzák. Ez utóbbi esetben az osztálycímkéket a cikkek rovatai alapján határozták meg. Végül a WAP gyűjtemény a WEBACE-projekt [84] keretében a Yahoo! taxonómiából összegyűjtött és felcímkézett dokumentumokat tartalmaz.

Szintén több kutató használta a CLASSIC3 tesztkorpuszt, amely 1400 repülésügyi rendszereket (CRANFIELD) tárgyaló, 1033 orvosi témájú (MEDLINE), és 1460 információ-visszakereséssel foglalkozó (CISI) dokumentumot tartalmaz.[10]

6.3. táblázat. Klaszterező eljárások elemzésére használt dokumentumgyűjtemények adatai (a jelölések feloldását ld. a szövegben) [179]

Név	Forrás	Dokumentumok száma	Kategóriák száma	Min osztályméret	Max osztályméret	Átlagos osztályméret	Szótár mérete
RE0	R-21578	1504	13	11	608	115,7	11465
RE1	R-21578	1657	25	10	371	66,3	3758
WAP	WebAce	1560	20	5	341	78,0	8460
TR31	TREC	927	7	2	352	132,4	10128
TR45	TREC	690	10	14	160	69,0	8261
FBIS	TREC	2463	17	38	506	144,9	2000
LA1	TREC	3204	6	273	943	534,0	31472
LA2	TREC	3075	6	248	905	512,5	31472

6.7.3. Csoportosító algoritmusok összehasonlítása

A [179] tanulmányban a k-átlagot, a kettészelő k-átlagot, és három egyesítő hierarchikus csoportosítót hasonlítottak össze a 6.3. táblázat nyolc korpuszán. Az egyesítő módszerek közül az átlagos és a centroid kapcsolódást, valamint a centroid módszernek a csoportméret szerinti normalizálás nélküli verzióját vizsgálták![11] A klaszterezőket F-mérték és entrópia szerint hasonlították össze.

[9] részleteket ld. [179]; forrás `trec.nist.gov/data/qrels_eng/index.html`

[10] `ftp.cs.cornell.edu/pub/smart`

[11] Azaz (6.7) kifejezésből a $\frac{1}{|c_i||c_j|}$ tényezőt elhagyták.

A három egyesítő módszer közül a centroid kapcsolódás adja a legjobb eredményt az F-mérték szerint az összes vizsgált gyűjtemény esetén, bár a másik két módszer sem ad lényegesen rosszabb értékeket. Entrópiamérték szerint a centroid és az átlagos kapcsolódás közel azonos eredményeket ad, míg a normalizálatlan centroid a másik kettőnél lényegesen rosszabb. Érdekesség, hogy az egyesítő algoritmus első néhány iterációjában még hasonló eredményeket ad mindhárom módszer, de később az utóbbi kezd több hibát véteni [179].

A *k*-átlag alapú particionáló klaszterezőket az említett eredmények miatt csak a centroid módszerrel vetették össze, és az alábbiakra jutottak:

- A kettészelő *k*-átlag módszer mind a *k*-átlag, mind a centroid kapcsolódás módszernél mindkét mérték szerint némileg jobb a vizsgált 8 korpusz legtöbbjén.
- A *k*-átlag módszer lényegesen jobb eredményeket ad, mint a centroid módszer.
- Noha a két *k*-átlag alapú eljárás eredménye több futás átlagaként állt elő, ezeknek a többszörös futási ideje sem éri el az egyesítő hierarchikus klaszterező futási idejét, mivel egy futáson a különbség mintegy 80–100-szoros.

Az egyesítő hierarchikus algoritmusok gyengébb teljesítményére a magyarázat a dokumentumok jellegzetességében rejlik. Gyakran előfordul ugyanis, hogy egy dokumentum legközelebbi szomszédja másik kategóriába tartozik. Az ilyen legközelebbi szomszédok aránya a vizsgált korpuszok esetében 5–30% volt! Erre a tulajdonságra a *k*-átlag alapú módszerek sokkal kevésbé érzékenyek, mint az egyesítő algoritmusok.

7. fejezet

Kivonatolás

Internetes keresés esetén szinte mindenki találkozott már azzal a problémával, hogy a találatok egy — gyakran jelentős — része nem releváns, illetve nem felel meg a felhasználó információigényének. A relevancia kérdésének eldöntéséhez azonban a találat címe, és a keresőkifejezésre illeszkedő néhány szavas részlete (snippet) gyakran nem szolgáltat elegendő információt. Ehhez olykor a teljes dokumentumot le kell tölteni és át kell futni, azaz időigényes munkát jelent. Sokban segítené és gyorsítaná a döntést, ha a találati dokumentumokról rendelkezésre állna egy rövid tartalmi összegzés, illetve kivonat.[1] A kereső- és tartalomszolgáltatók részéről ugyanakkor nem várható el, hogy nagy költséggel szakértők seregét finanszírozza kivonatkészítésre. Sokkal egyszerűbb és költséghatékonyabb a feladatot automatikus szövegbányászati módszerekkel megvalósítani.

Az automatikus tartalmi összefoglalásokat gyártó módszereket *összegzéskészítő eljárás*oknak (text summarization method) nevezzük. Ezek a fenti példán kívül számos más feladatra is használhatók, pl. összehasonlító táblázatok készítésére, többnyelvű információkinyerés támogatására, biográfiai profilok készítésére, kisképernyős (kézi számítógép, mobiltelefon) tartalomszolgáltatás támogatására,[2] dokumentumok tartalmának automatikus feldolgozásával strukturált adatbázisépítésére stb.

7.1. Az összegzéskészítő eljárások típusai

Az *összegzéskészítő eljárás*okat a szakirodalom az összegzés előállításának módja szerint két alapvetően különböző csoportba sorolja. *Kivonatolás*nak (extraction)

[1] A tudományos publikációknál már jó ideje általános követelmény, hogy a cikk elején rövid kivonat foglalja össze annak tartalmát, újdonságait. A kutatók ennek alapján el tudják dönteni, hogy érdekes-e számukra a tanulmány. Térítéses publikációs adatbázisoknál ezért a kivonat mindig ingyenesen hozzáférhető, és csak a teljes cikkért kell fizetni, ld. pl. `portal.acm.org`, `ieeexplore.ieee.org`.

[2] Ezzel kapcsolatos esettanulmány könyvünk honlapján is szerepel.

hívjuk az olyan eljárásokat, amelynek eredménye kizárólag az eredeti szövegből vett részeket tartalmaz. Az *összefoglalás-generáló* (abstraction) módszerek által előállított szöveg viszont olyan szintetizált szövegrészeket is tartalmaz, ami nem része a feldolgozott dokumentumnak.

Az emberi gondolkodás és információfeldolgozás modellezése — így az összegzéskészítése is — bonyolult feladat. Az összefoglalás függ a készítő személyétől, szaktudásától, különbözhet méretben, nyelvezetben, stílusban és részletezettségben. Az összegzéskészítés folyamatának matematikai vagy logikai formulákkal való leírása rendkívül komplex feladat [90]. Az utóbbi években a nyelvtechnológiai eszközök fejlődése azonban lehetőséget adott olyan rendszerek megalkotására, amelyek képesek szövegek szemantikai feldolgozására is. Ilyen eszközök segítségével, a szövegben található frázisok és lexikai láncok meghatározásával, majd azok összefűzésével, lehetőség van *összegfoglalás*ok automatikus *generálásá*ra is. Ennél lényegesen egyszerűbb a kivonatoló eljárások működése, amelyek a kivonatot az eredeti szöveget leginkább jellemző szövegegységek (mondatok, bekezdések stb.) kiválasztásával állítják elő. Vizsgáljuk meg az egyes megközelítések hátrányos tulajdonságait.

A kivonatoló eljárások esetén:

- A kiválasztott mondatok az átlagosnál hosszabbak (ld. 7.2. pont), és egyes részeik gyakran nem tartalmaznak lényegi információkat, így indokolatlanul szerepelnek a kivonatban.
- A legfontosabb információegységek általában elszórtan vannak jelen az egész dokumentumban, és ezt a kivonatoló módszerek nem képesek feldolgozni.
- A szövegben szereplő ellentmondó információkat a kivonat nem képes feldolgozni.

Az összefoglaló eljárások esetén:

- A felhasználók jobban kedvelik a kivonatolással készült összegzést, mint a generált összefoglalókat [60]. Ennek oka, hogy a kivonat a szerző eredeti kifejezéseit, szóhasználatát tartalmazza, valamint olykor lehetőséget nyújt a sorok közötti információk olvasására is.
- Jelenleg a mondatszintézis szakterülete még, angol nyelvre is, gyerekcipőben jár, ezért az automatikusan generált szövegekben gyakoriak még a mondaton belüli ellentmondások is, így az egész szöveg könnyen összefüggéstelenné válik. Kivonat esetén inkoherencia csak a mondatok határainál fordul elő.

A felhasználási cél alapján az összegzéskészítő eljárások az alábbi szempontok szerint rendszerezhetők [124]:

- Részletezettség: *indikatív* vagy *informatív*. Az indikatív összegzés azt tartalmazza, hogy a szövegnek mi a témája, míg az informatív összegzés ugyanannak egy speciális részletét tárgyalja.
- Tartalom: *általános* vagy *kérdésvezérelt*. Az összegzés lehet egy dokumentum tartalmának általános leírása, vagy kiemelheti a tartalomnak a felhasználó által megadott kérdéssel kapcsolatos részét.
- Megközelítés: *téma-*, ill. *típusspecifikus* vagy *független*. A tapasztalatok azt mutatják, hogy különböző típusú (pl. rövidhír, tudományos publikáció) dokumentumokban a lényegi információ más helyen található.

Mivel a legtöbb működő alkalmazás a kivonatolás módszerét alkalmazza, ezért a továbbiakban erre fókuszálunk.

7.2. A kivonatolásnál használt jellemzők

Az összegzéskészítő algoritmusok feladata némiképpen eltér az előző fejezetekben elemzett információkinyerő és dokumentumgyűjtemények rendszerezésével kapcsolatos szövegbányászati problémáktól,[3] ezért az itt alkalmazott módszerek gyakran heurisztikus megoldásokra épülnek. A dokumentumok feldolgozásánál a standard vektortér-, illetve valószínűségi token alapú megközelítés mellett és helyett más jellemzőket is használunk a dokumentumok és alegységeik leírására.

A kivonatoló technikák egyik jellemző működési sémája mondatkiválasztáson alapul, amelynek előfeltétele, hogy a dokumentumot mondatokra szegmentáljuk (ld. a 2.2.4.2. pontot). Ekkor az egyes mondatokhoz valamilyen heurisztikus módon meghatározott értéket rendelünk hozzá, és a legmagasabb pontszámot elérő mondatokat tesszük bele a kivonatba. A mondatokhoz rendelt értéket az alábbi tényezők növelik:

Kulcsszó-előfordulás: Azok a mondatok, amelyekben a szöveg leggyakoribb szavai szerepelnek, általában jól reprezentálják a dokumentumot.

Címbeli kulcsszó: A címben szereplő szavak általában utalnak a dokumentum tartalmára is, ezért az olyan szövegközi mondatok, amelyekben címszavak szerepelnek, az átlagosnál jobban jellemeznek egy dokumentumot.

Előfordulási hely heurisztika: Újsághírek esetén többnyire az első mondat, technikai-tudományos szövegeknél az összefoglalás utolsó mondatai, illetve a konklúzió tartalma jól jellemzi az adott dokumentumot.

[3] Kivételt csak a klaszterezők eredményeinek felcímkézése jelent.

Utaló frázisok: Az olyan kulcsszavakat tartalmazó mondatok, mint pl. *ez a cikk*, *a tanulmány*, *jelen munkánkban* az átlagosnál több információt hordoznak a szöveg egészéről.

Nagybetűs szavak: Rövidítéseket vagy tulajdonneveket tartalmazó mondatok általában többletinformációt hordoznak.

A mondatokhoz rendelt értéket az alábbi tényezők csökkentik:

Rövid mondatok kiszűrése: A kivonatban jellemzően nincsenek rövid, néhány szavas mondatok.

Névmások: Személyes, vonatkozó, birtokos stb. névmásokat tartalmazó mondatok csak akkor kerülnek be a kivonatba, ha meghatározható, hogy mire vonatkoznak (ld. anaforafeloldás, 4.6. szakasz). Ekkor a kivonatba kerülő mondatban a névmást a vonatkozó kifejezéssel helyettesítjük.

Informális és pontatlan szavak: A gyakori és sok jelentéssel bíró vagy pontatlan szavak negatív tényezők a mondatkiválasztásnál.

Idézésre utaló szavak: Angol nyelvű híreknél az idézésre utaló szavak szintén negatív faktorok: *adding*, *said*, *according* stb.

Redundanciacsökkentés: Ezt a pontszámot olyan eljárásokban alkalmazzák, ahol egyenként határozzák meg a kivonatba kerülő mondatokat. Az értéket minden új mondat kiválasztásánál újraszámolják, megelőzendő azt, hogy a kiválasztott mondat valamelyik, a kivonatba már korábban bekerült mondathoz hasonlítson, pl. úgy, hogy arányosan csökkentik a még nem beválasztott mondatok pontszámát aszerint, hogy mennyire hasonlítanak az aktuális kivonathoz [77, 153].

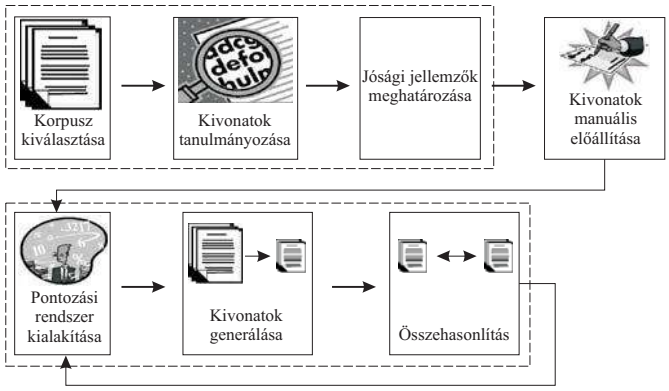
Az alkalmazott jellemzők jellege szerint megkülönböztethetünk nyelvi, statisztikai, információelméleti és kombinált módszereket.

7.3. Kivonatoló módszerek

7.3.1. A klasszikus módszer

Bár az automatikus összegzéskészítő eljárások csak az internet és a keresőmotorok széleskörű elterjedésével kerültek a kutatások homlokterébe, az első eredmények e témában már a 60-as évek elején megszülettek [57]. Edmundson korai munkájában összefoglalta az akkor ismeretes eljárásokat, és lerakta a kivonatoló technikáknak mind a mai napig érvényben lévő alapjait [211]. Módszere az alábbi lépésekből áll (7.1. ábra):

1. Emberek által készített kivonatok tanulmányozásának segítségével határozzuk meg az automatizált kivonatolás eredményétől elvárt jellemzőket.
2. Készítsünk ennek megfelelő kivonatokat emberi munkával.
3. Tervezzünk olyan matematikai és logikai formulákat a mondatok pontozására és rangsorolására, hogy a kívánt (manuálisan gyártott) eredményt kapjuk.
4. A pontozási-kiválasztási rendszer finomítása mellett addig ismételjük a módszert, amíg a manuálisan és automatikusan generált kivonatok nem lesznek azonosak.



7.1. ábra. Kivonatoló algoritmusok működési vázlata

A stopszavak kiszűrése és a szótövesítés (ld. a 40. oldaltól) elvégzése után az Edmundson-féle eljárás az alábbi jellemzőket veszi figyelembe a pontozási-kiválasztási rendszer paraméterezése során:

- utaló szavak és frázisok;
- gyakori és egyben informatív szavak (kulcsszavak);
- címbeli kulcsszavak;
- előfordulási hely heurisztika.

Ezek után minden s_i mondatra meghatározzuk az alábbi $S(s_i)$ kifejezés értékét:

$$S(s_i) = w_1 \cdot R_i + w_2 \cdot K_i + w_3 \cdot T_i + w_4 \cdot L_i, \tag{7.1}$$

ahol R_i, K_i, T_i, és L_i rendre a mondatban szereplő utaló frázisok, kulcsszavak, címszavak száma, illetve az előfordulási hely heurisztika által meghatározott érték. A w_i $(i = 1,\ldots,4)$ együtthatók az egyes tényezőkhöz rendelt súlyfaktor.

Az $S(s_i)$ értékek alapján vagy a k legmagasabb értékkel rendelkező, vagy egy meghatározott küszöbértéknél nagyobb pontszámmal rendelkező mondatokból alkotjuk a kivonatot.

A módszer időtállóságát mutatja, hogy még manapság is vannak ilyen alapon működő kivonatolók [211]. Egyetlen komoly hiányossága, hogy nem veszi figyelembe a kiválasztott mondatok hasonlóságát, és nem alkalmaz redundanciacsökkentő módszereket.

7.3.2. A tf-idf alapú módszer

A tf-idf módszer az információ-visszakeresési paradigma alkalmazása kivonatolási célra [135]. A módszer elsősorban kérdésvezérelt kivonat előállítására alkalmas, de megfelelő módosítással általános kivonat létrehozására is használható.

Készítsük el a dokumentum mondatainak vektortérmodell szerinti reprezentációját a tf-idf súlyozással (ld. a (2.9) képletet).[4] A mondatvektorokat hasonlítsuk össze a keresőkifejezésből generált vektorral — erre például a koszinusztávolságot alkalmazhatjuk (ld. az (5.3) képletet és az 5.2. ábrát) —, és válasszuk ki a leginkább hasonló mondatokat. Általános kivonat létrehozásához a dokumentum leggyakoribb (informatív) kulcsszavaiból képezzük a keresővektort. Mivel elvileg ezek reprezentálják leginkább a dokumentum tartalmát, a dokumentumból a fentiek szerint kiválasztott hasonló mondatok a szöveg általános összegzésének tekinthető.

Ennek az eljárásnak több gyenge pontja is van. Egyrészt a felhasználói szokások alapján megadott kérdés is legtöbbször általános kivonatot eredményezhet, hiszen ha tág értelmű szavakkal keresünk egy szövegben mint pl. *information retrieval*, akkor nem jutunk specifikus információhoz. Másrészt, mivel a módszer csak olyan mondatokat választ ki, amelyekben a keresőszavak szerepelnek, biztosan kimarad néhány nagyon fontos és informatív mondat a kivonatból, ami pl. már az adott dokumentum témáját részletesebben ismerteti. Ugyancsak emiatt a kivonat rendkívül redundáns lesz.

7.3.3. Csoportosítás alapú módszerek

A jól megírt dokumentumok általában azt a *szerkesztési elv*et követik, hogy egy bővebb szakterülethez tartozó témákat tárgyalnak egymás után. Ennek alapján (ténylegesen vagy implicite) szakaszokra bonthatók szét. A dokumentum tematikus szerkezetét a kivonatnak is tükröznie kell, hiszen szerepelnie kell benne a

[4] itt a reprezentáció egységei tehát a mondatok lesznek és nem a dokumentumok.

szövegben tárgyalt fontosabb témáknak. Ezt a szerkesztési elvet egyes kivonatolók csoportosító eljárások alkalmazásával valósítják meg. Megjegyezzük, hogy e módszerrel nem csak dokumentumok, hanem egész dokumentumgyűjtemények kivonatolása is elvégezhető.

7.3.3.1. Az MMR-módszer

Az MMR-módszer (maximum marginal relevance) [35, 76, 77] alapötlete, hogy egyszerre maximalizálja a keresőkifejezéshez való hasonlóságot és a már kiválasztott mondatoktól való eltérést. Az első tényező a relevanciát növeli, míg a második a redundanciát csökkenti a kivonatban. A módszer mind statisztikai, mind nyelvi jellemzőket felhasznál a mondatok kiválasztása során. Ezért egyaránt figyelembe tudja venni a kulcs- és címszavak előfordulását, az időrendi sorrendet, kérdéshez/szakterülethez való hasonlóságot (tehát általános és kérdésvezérelt kivonatot is képes előállítani), a névmások előfordulásának büntetését. A mondatokat az alábbi formula szerint pontozza:

$$\mathrm{MMR}(p_{ij}) = \lambda s_1(p_{ij}, q, c_{ij}) - (1-\lambda) \max_{s \in S} s_2(p_{ij}, s_{k\ell}, C, S), \qquad (7.2)$$

és mindig a maximális pontszámot elérő p_{ij}-t választja ki. Itt p_{ij} a d_i dokumentum j-edik szövegegysége (frázisa, mondata vagy bekezdése; közösen ezeket *passzus*nak nevezzük), q a keresőkifejezés, c_{ij} a C klaszterezés azon csoportjai, amelyek a p_{ij} passzust tartalmazzák és S a már kiválasztott passzusok halmaza. A λ paraméter a két optimalitási kritérium (relevancia és újdonság) fontosságát szabályozza. Ha $\lambda = 1$, akkor csak a relevanciát veszi figyelembe, ha $\lambda = 0$, akkor az eddigiektől legeltérőbb mondatot választja ki. A két hasonlósági mérték, s_1 és s_2 meghatározásánál számos tényezőt figyelembe lehet venni, amelyeknek a (7.1) kifejezéshez hasonlóan súlyozott lineáris kombinációját vesszük. Ez a módszer lehetővé teszi azt is, hogy tetszőleges méretű kivonatot generáljunk, hiszen a kivonat bővítése bármikor befejezhető.

Az MMR-eljárás az alábbi centroid kapcsolódás jellegű egyesítő hierarchikus eljárással végzi el a dokumentum (vagy dokumentumgyűjtemény) passzusainak csoportosítását. Az eljárás fő paramétere a kiinduláskor meghatározott hasonlósági küszöbérték, θ. Az egyesítő lépés során (ld. a 155. oldal, 2. lépését) az s_C hasonlóság szerint az összes olyan csoportot összevonja, amelyre $s_C(c_i, c_j) \geq \theta$ teljesül. Egy passzus így több csoportba is bekerülhet. Az összevonás után kiszámítjuk a csoportok új centroidját és addig ismételjük az egyesítő műveletet, amíg elég közeli csoportok találhatók. Az eredményül kapott csoportosítás elemeit a

medoidok reprezentálják. Ezek általában a csoport leghosszabbak mondatai lesznek.

A módszer a hasonlósági mértékek egyes komponenseinek számításánál felhasználja a dokumentumok csoportosítását [77]. Az s_1 hasonlóságnál többek közt figyelembe veszi, hogy a passzus hány csoportban szerepel, és mekkora a csoportok elemszáma. Ha ugyanis ezek az értékek nagyok, akkor a passzus nagy szeletét lefedi a dokumentumnak, ill. a korpusznak. A s_2 hasonlóságnál azt bünteti, ha az adott csoportból már más dokumentum bekerült a kivonatba.

A θ paraméter értéke jelentősen befolyásolja a kialakult csoportokat, ezért ez az eljárás nem túl robosztus. Ez az érzékenység javítható, ha nem a távolsági mérték alapján képezzük a csoportokat, hanem a leggyakoribb kulcsszavak mondatokban való előfordulása alapján [73].

7.3.3.2. A MEAD-rendszer

A dokumentumgyűjtemények kivonatolására alkalmas MEAD-rendszert [153] a Michigani Egyetemen fejlesztették ki. A dokumentumokat a vektortérmodellben tf-idf súlyozás segítségével írjuk le. A MEAD bemenete a dokumentumok csoportosításának eredménye. Ekkor minden csoportot egy külön témának lehet tekinteni, ahol a csoportok tartalmát a centroid legnagyobb tf-idf értékű szavai jellemzik.

A hírarchívumok kivonatolása esetén a mondatkiválasztás három tényezőt vesz figyelembe. Elsőként a csoport centroidjától való távolságot (C_i), a mondatnak a dokumentumon belüli pozícióját[5] (L_i) és a dokumentum első mondatával való hasonlóságot (T_i). Ezen mennyiségek lineáris kombinációjaként áll elő egy mondat pontszáma, a korábbiakhoz hasonló módon (ld. pl. a (7.1) képletet). A különbség az, hogy a MEAD a mondat pontszámát redundanciacsökkentés érdekében újraszámolja az új mondatok bevétele után.

A módszer hátránya, hogy a tf-idf súlyozási sémát alkalmazza, amely kivonatolási technikák esetén nem a leghatékonyabb. Másik gyenge pontja, hogy elsősorban hírarchívumok feldolgozására alkalmas, hiszen a három tényezőből kettő is (T_i és L_i) erősen a dokumentumok elején lévő mondatokat favorizálja.

7.3.4. Gráfelméleti megközelítések

Ahogy az előzőekben láttuk, több kivonatoló módszer esetén is az első lépés a mondatok vagy dokumentumok csoportosítása. A mondatok gráfelméleti repre-

[5] Minél közelebb van egy mondat a dokumentum elejéhez, annál nagyobb ez az érték.

zentációja ugyancsak alkalmas eszköz témák meghatározására [167]. A szokásos előfeldolgozási lépések után a mondatokat egy irányítatlan gráf csomópontjaival reprezentáljuk, ahol a csomópontok közötti éleket a mondatokban előforduló közös szavak számával súlyozzuk. Az élek súlyára vonatkozóan minimális küszöbértéket is meghatározhatunk.

Ennek a reprezentációnak két eredménye van: a gráfpartíciók vagy -klikkek az egy témához tartozó mondatokat azonosítják, és így egy csoportbarendezést generálnak. A klikkek erősségét, tehát az egy csoportba tartozó mondatok kohézióját a küszöbérték növelésével emelhetjük, és ezzel egyúttal a témák számát is szabályozhatjuk. Ez a reprezentáció egyaránt lehetőséget nyújt általános és kérdésvezérelt kivonatok létrehozására. Az előbbi esetben minden gráfklikkből egy-egy mondatot választva lefedjük az egész dokumentum(gyűjtemény) tématerületét, míg az utóbbi esetben elegendő a kérdéssel egy klikkben lévő mondatok közül kiválasztani néhányat.

A másik fontos eredmény, hogy a nagyszámú éllel rendelkező csomópontok a dokumentum(gyűjtemény) fontos mondatait is meghatározzák, amelyeknek ezáltal nagyobb esélye van a kivonatba kerülésre. A grafikus megközelítés könnyen hasznosítható dokumentumon belüli és dokumentumok közötti összefüggések vizuális megjelenítésére is.

7.3.5. Az LSI használata a kivonatolásban

A szinguláris értékfelbontáson alapuló látens szemantikus indexelés módszerét (ld. a 2.3.2.2. pontot) kivonatolásnál a mondatok közötti szemantikus összefüggések felfedésére is alkalmazhatjuk [212]. Mivel az LSI képes többdimenziós adatok ortogonális dimenzióinak megtalálására, ezért a szó-dokumentum mátrixokra alkalmazva olyan mondatokat is azonos szinguláris dimenzióhoz rendel, amelyek nem tartalmaznak közös szavakat, de ugyanakkor témájuk hasonló. Az LSI nagy előnye, hogy a szemantikus összefüggéseket automatikusan az emberi agy által reprezentált módon képes megragadni [110].

Az eljárás jól használható témákra jellemző szavak, illetve mondatok meghatározására egyaránt. Mivel az SVD fontossági rangsorba állítja a szinguláris értékek alapján a kölcsönösen ortogonális szinguláris irányokat a mondatvektorok terében, ezért ha ezekből a dimenziókból választjuk ki a reprezentatív mondatokat, akkor egyrészt biztosítva lesz a dokumentum teljes tematikájának lefedettsége, másrészt az ortogonalitás garantálja a redundanciamentességet [78]. Egyetlen megszorítás, hogy csak eredendően tematikus egységekbe rendezett szövegekre

alkalmazható hatékonyan, ám a legtöbb dokumentum ilyen szerkesztési elvet követ.

7.4. A kivonatolás hatékonyságának mérése

Egy összegzés megítélése személyenként változó lehet, függetlenül attól, hogy gép vagy ember által készített anyagról van szó. A [77] tanulmányban ismertetett kutatás során felkértek néhány embert dokumentumok legfontosabb mondatainak kiválasztására, valamint a gép által készített kivonatok értékelésére. A résztvevők által kiválasztott mondatok között nagyon kicsiny volt az egyezés, és hasonlóan eltérően ítélték meg az automatikusan generált kivonatok minőségét is. Ez a példa jól illusztrálja, hogy nem egyszerű feladat a kivonatok minőségi követelményeinek meghatározása. Mindazonáltal, az alábbi intuitív szempontok iránymutatónak tekinthetőek egy kivonat hasznosságának és teljességének megítélésében [124, 76]:

- Meg tudja-e válaszolni a felhasználó mindazokat a kérdéseket a kivonat elolvasása után, amelyekre az egész szöveg elolvasása esetén képes lenne?
- Mi a tömörítési aránya a kivonatnak az eredeti szöveghez képest?
- Van-e a kivonatolt szövegben ismétlődés, redundancia?

A kivonatok egyéb jellemzőit, pl. kohézió, olvashatóság, sokkal nehezebb mérni.

A kivonatolás minőségére vonatkozóan is megkülönböztetnek belső és külső mértékeket [76] aszerint, hogy csak a kivonat tulajdonságait veszi-e figyelembe az adott mérték, vagy a kivonat minőségét valamely más cél elvégzésében nyújtott támogatás hatékonyságának tükrében vizsgálja. A felsorolt mértékek közül a második az előbbi, míg az első az utóbbi kategóriába tartozik. Kizárólag a tömörítési arány nem megfelelő jellemzője a kivonat minőségének, hiszen pl. a redundanciát vagy az információ hasznosságát nem veszi figyelembe.

A kivonatoló technikák kiértékelésére külső mértékként az Edmundson által javasolt mértéket [56] használják a leggyakrabban. A módszer zsinórmértékként szakértők által elkészített kivonatokat használ. Először meghatározzák a gép és a szakértők által készített kivonatokban az egyező mondatok számát. Ezután a szokásos IR-mértékekkel — pontosság, felidézés — jellemzik a kivonatolás minőségét. A módszer hátránya, hogy költséges a referenciaadatok előállítása. Ez ugyanakkor növeli a megbízhatóságot: a szakértői kivonatokkal szemben sikeresen tesztelt eljárások általában hatékonynak bizonyulnak.

8. fejezet

Tartalomkeresés webdokumentumokban

8.1. Történeti áttekintés

8.1.1. Hipertext-dokumentumok kialakulása

A weben tárolt dokumentumok egyik sajátos és fontos tulajdonsága, hogy tartalmazhatnak más dokumentumokra hivatkozó kapcsolati elemeket. Mivel a webes dokumentumok esetében ez a kapcsolat rögtön aktivizálható, ezt a mechanizmust teljes körű *hiperlink*nek nevezik. A hiperlinkeket tartalmazó dokumentumokat *hipertext*nek is szokás nevezni. A hipertext ötlete 1945 körül jelent meg. Az első jelentős megvalósulása a MEMEX-projekthez [33] köthető, amely egy naplóra hasonlító rendszer kialakítását tűzte ki célul. A fejlesztés fő iránya a gyors és hatékony információ-visszakeresés biztosítása volt. A MEMEX-terv ugyan nem valósult meg teljes egészében, de részsikereivel nyitánya lett az informatika egy új területének.

A hipertextrendszerek kezdeti formális megközelítései közül a legismertebbek közé tartozik a strukturális orientáltságú Hipertext Absztrakt Gép- (Hypertext Abstract Machine; HAM-) modell [15] és az 1990-ben publikált Dexter Hypertext-modell [81]. E modellek szerkezeti szemléletmódból kiindulva adják meg a dokumentumok leírását. Más jellegű megközelítést adnak a formális logikán vagy a Petri-hálók elméletén alapuló munkák [180].

8.1.1.1. Szerkezeti szemléletű modellek

A HAM-modell egy hipertext-referenciamodell, amely az alábbi rétegekből áll:

- felhasználói felület;
- alkalmazási réteg;
- hipertext-tárolási réteg;
- operációsrendszer-réteg.

A modell a teljes dokumentumkezelési környezetet magába foglalja, azon belül pedig elkülöníti a tárolás és a feldolgozás moduljait. A HAM koncepciójában a moduláris felépítés előnyei nyilvánulnak meg.

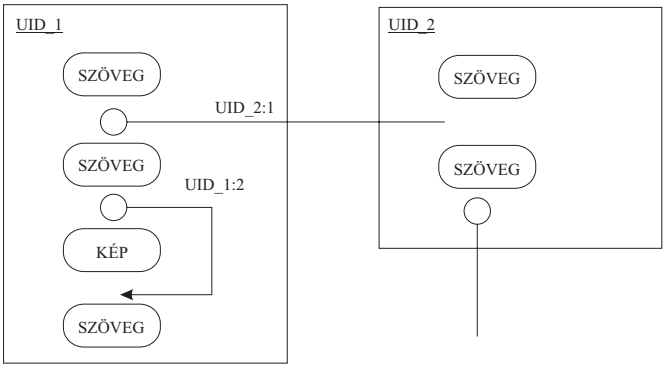
A *Dexter*-referenciamodell a hipertext-dokumentumokhoz három különböző megközelítési szintet rendel:

- megjelenítési szint;
- tárolási réteg;
- komponensréteg.

A Dexter értelmezése szerint a hipertext-dokumentum egymással hivatkozási kapcsolatban álló komponensek rendszere. A modell legalsó szintje ezen komponensek belső strukturális szintjét jelöli. A középső szint a komponensek külsőleg is látható felületét és a kapcsolatok leírási módját határozza meg. A legfelső réteg a felhasználói, kezelőfelületi réteg, amely a dokumentumon végezhető operátorok szintjét jelöli. A komponensek vonatkozásában három altípust különböztethetünk meg:

- atomi komponens (atom);
- összetett komponens (composite);
- kapcsolati komponens (link).

A Dexter-modellt igen elterjedten használják a hipertextrendszerek induló alapmodelljeként.



8.1. ábra. Dexter-féle hipertextmodell

8.1.1.2. A Petri-hálók elméletén alapuló modell

A Dexter-féle megközelítés jól leírja a dokumentumok statikus részét, de nem tér ki a dokumentumok dinamikus oldalára. A dinamikát is magába foglaló modellek közül az egyik legismertebb a *Trellis*-modell [180]. A modell nemcsak a hipertextstruktúrát kívánja vizsgálni, hanem tágabb kontextusban a felhasználó és a hipertextrendszer interakcióját is [15]. A Trellis-modell is három rétegre bontja a hipertextrendszert:

- absztrakt réteg;
- tartalomréteg;
- megjelenítési réteg.

A Trellis-modell a Petri-hálók elméletén alapszik [144], amely lehetővé teszi a struktúra és a dinamika együttes kezelését. A Petri-hálók szerkezete az alábbi elemekből épül fel. A háló egy

$$< S, T, F > \tag{8.1}$$

hármassal adható meg, ahol

- S: a helyek halmaza (dokumentumok és kezelőablakok);
- T: az állapotátmeneti csomópontok halmaza (GUI navigációs elemek);
- F: a helyeket és a csomópontokat összekötő élek, kapcsolatok halmaza, tehát

$$F \subset ((S \times T) \cup (T \times S))$$

teljesül.

A hipertext alapú keresés hatékonyságának tárgyszerű értékelésére Bruza 1990-ben kidolgozott egy szempontrendszert [31]. A keresés mérőszámai:

- *pontosság* (precision): a kiválasztott releváns dokumentumok számának aránya a kiválasztott dokumentumok számához viszonyítva;
- *felidézés* (recall): a kiválasztott dokumentumok számának és a releváns dokumentumok számának aránya;
- *tartalomegyezés* (exhausitivy): a kiválasztott objektumok és a keresőkifejezés tartalmának egyezése;
- *kifejezőerő* (power): a leíró kulcsszó szelektivitásának és hosszának az aránya;

- *eliminálhatóság* (eliminalibity): az irrelevancia minél hamarabbi felfedésének képessége a keresési feltételek között;
- *érthetőség* (clarity): a kulcsszavak közérthetősége;
- *megjósolhatóság* (predictability): a keresési eredmények előrejelezhetősége.

A webet uraló HTML/XML-formátumok is szorosan kötődnek a hipertextdokumentum-rendszerekhez, hiszen a HTML-szabvány is egy elosztott hipertextrendszer leíró nyelveként lett kidolgozva 1989-ben [21]. A nyelvet az SGML [75] mintájára alkották meg. A kialakított XML-struktúrát [52] hierarchiának tekinthetjük, és a dokumentum egy irányított, körmentes gráfnak foghatjuk fel. Ekkor a H dokumentum szerkezete:

$$H = n^*, \quad \text{ahol} \quad n = n^*|w,$$

itt n egy csoportot jelöl, w egy szó, és a * az iteráció (Kleene-csillag, a konkatenálásra vonatkozó lezárás) operátora. Adott továbbá egy típusfüggvény is:

$$\tau : N \to T, \quad \text{ahol} \quad N = \{n\}, \qquad T : \text{típusok halmaza.}$$

Az így definiált szerkezetre az alábbi megkötéseket tesszük:

- Egy és csak egy gyökérelem van $(H = n)$;
- $T = \{\text{text, attribute, empty, compound, instruction}\}$.

Az elkészült HTML-rendszer lehetőséget ad a kapcsolatok megvalósítására is. Az első implementációkban csak elemi, egyirányú kapcsolatok voltak jelen. A hivatkozásokat a szöveges és képi elemekhez lehetett hozzákötni:

.

A későbbi változatokba kiegészítő navigációs elemek is bekerültek, pl.

- visszalépés;
- előrelépés;
- ugrás az induló lapra.

A rendszer elvégzi a végrehajtott navigációs lépések naplózását is. A beépített lehetőségek közé tartozik az is, hogy a dokumentum egyes részeit azonosítónévvel láthatjuk el. Ezáltal a hivatkozásokban a dokumentum mellett a dokumentumrészlet azonosítója is szerepelhet. A dokumentum betöltésekor a hivatkozó rész kiemelten jelenik meg. Az eredeti tervezetben olyan kapcsolódási elemek is voltak, amelyek végül nem váltak általánossá, mint pl. a számított, származtatott kapcsolati elem és a tipizált kapcsolódás.

Összességében megállapíthatjuk, hogy jelenleg a webes tárolási lehetőségeket legjobban kihasználó dokumentumtípus a hipertext. A hipertext-dokumentumok szerves része a kapcsolati elem, amelynek révén egy dokumentumháló alakulhat ki. A hivatkozások a dokumentumok tetszőleges részeire mutathatnak.

8.1.2. A keresőmotorok kialakulása

Az internet létrejöttével a számítógépek közvetlenül kapcsolódhatnak egymáshoz. Ezzel lehetőség nyílt arra, hogy az egyik számítógépen lévő adathalmazt egyszerű módon átvigyük egy másik számítógépre. Mivel a hasznos adatokat, programokat sokan szívesen látják a saját gépükön, igen hamar megnőtt az igény az áttöltések iránt. A technológiai háttér létrejöttével hamarosan megjelent az információkeresés igénye is: honnan tudjuk letölteni a szükséges programot, milyen programok állnak egyáltalán rendelkezésre?

A felhasználó szemszögéből csak akkor hasznos a világháló, ha el tudja érni a keresett információt. Az internet fejlődésével nagyon rövid időn belül felmerült a tartalomkeresés iránti igény. Az első tartalomkereső rendszerek megelőzték a hipertext és a világháló korát. Ezek egyszerű fájlkeresők voltak, amelyek az interneten működő ftp-szervereken lévő tartalmak keresését támogatták.

Az internetes keresési igények kielégítésére alkalmas programok az 1990-es évek elején jelentek meg. Első neves képviselői az Archie és Veronica [88]. Az *Archie* az ftp-szervereken felkínált állományokról készített nyilvántartást egy automatikus indexelési mód segítségével: egy segédprogram lekérte a szerverekről a rendelkezésre álló állományok listáját, majd ezt a listát beépítette a saját nyilvántartásába. A felhasználók keresőkifejezésként egy fájlnévmaszkot adhattak meg, és válaszként a maszkra illeszkedő fájlok elérési adatait kapták meg.

Az Archie úttörő szerepet játszott az interneten történő információkeresés világának létrehozásában. Ez a világ igen gyorsan fejlődött és fejlődik ma is, hiszen egyre újabb felhasználói igények jelennek meg. Az Archie-rendszerben például nem lehetett téma szerint keresni, a rendszer ugyanis a fájlnév, és nem az állomány tartalma alapján indexelt. A felhasználók viszont legtöbbször csak a témakört tudják megadni, amiről bővebb információt szeretnének kapni, de fogalmuk sincs, hogy az igényelt információk milyen elnevezésű állományokban fordulhatnak elő. A tartalom alapú indexelés és keresés azonban technikailag már sokkal összetettebb feladat, mint a fájlnév alapú keresés.

Az első tartalom alapú keresőrendszerek egyike az *Aliweb* [197] volt. Ebben a rendszerben a felhasználók maguk készítik el a dokumentumaik leírását, ami alapján az Aliweb az indexelést végzi. Ezt a megoldást az támasztotta alá, hogy

így pontos, a lényeget kiemelő leírás készült, továbbá nem kellett adatszűrőket kidolgozni a különböző adatformátumú tartalmak felfedéséhez, egységesítéséhez. Sajnos az emberi munkán alapuló precíz lényegkiemelés igen költség- és időigényes folyamat. Mivel az összefoglaló leírás elkészítése nem egyszerű és nem is gyors, és az indexelt dokumentumok köre viszonylag szűk maradt, ezért a rendszer nem volt versenyképes. Jelenleg is manuálisan felépített témahierarchiát alkalmaz a *Tradewave Galaxy* rendszere. Ebben a dokumentumok indexelését is humán szakértők végzik, ezáltal valóban nagyon jó találati pontosságot kapunk, de az indexelt dokumentumok halmaza igen kicsiny.

A keresőrendszerek fejlesztése tehát az automatikus indexelés felé fordult. A megvalósítás egyik előfeltétele, hogy a tárolt dokumentumok tartalma viszonylag egyszerűen elérhető, olvasható legyen. A HTML alapú dokumentumformátum igen alkalmas erre a célra, hiszen könnyen feldolgozható, normál szöveges ASCII-kódolást használ. Az automatikus indexeléshez hamarosan kifejlesztették a keresőrobotokat, amelyek automatikusan felkeresik az interneten elérhető szervereket, letöltik az ott elérhető dokumentumokat, s azokat feldolgozva kiemelik a hasznos információkat. A kiemelt adatokat a hatékony elérés biztosítása végett indexelik. A kinyert információt sokszor arra is felhasználják, hogy meghatározzák a további keresési lépéseket. Az első indexelő robotok egyike a *Wandex*, amelynek gyenge pontja a lassúsága volt, mivel egy-egy weboldalt naponta többször is felkeresett és indexelt.

A webrobotok áttekintő listája elérhető a Web Robots Database[1] honlapon. A több száz robotprogram zömében indexel, de emellett a statisztikák készítését, és a karbantartást is végezhetnek. A programok többnyire UNIX alatt futnak, de akadnak platformfüggetlen megoldások is.

A Search Engine Watch 2006 júliusi felmérése szerint a keresőmotorok piacának öt jelentős, legalább 5%-os részesedésű képviselője van: a Google, a Yahoo, az MSN, az AOL és az Ask. A felmérés csak az angol nyelvű kereséseket vette alapul. A vizsgált keresőrendszerek együttesen napi átlagban 213 millió, havi átlagban 6,4 milliárd keresést végeznek, a becslések szerint több mint 30–40 milliárd szöveges dokumentumon. Ezek a számok jól érzékeltetik, hogy milyen nagy számítási és tárolási kapacitás szükséges a korszerű keresőrendszerek működtetéséhez.

[1] `www.robotstxt.prg`

8.2. Követelmények a keresőmotorokkal szemben

A felmérések szerint a webes keresőrendszerek üzemeltetői több mint 8 milliárd amerikai dollár bevétellel zárták a 2005-ös évet. A keresőrendszerek fejlődését tehát nagy mértékben az üzleti érdekek befolyásolják. Emiatt a követelmények között első helyen szerepel a felhasználói elégedettség. Ennek érdekében először is meg kell érteni a felhasználó szándékát, amit a felhasználó által adott keresési kulcsszavak többértelműsége, valamint a kontextus hiánya nehezít. Ezt követően releváns választ kell generálni, amelyet a keresőrendszer által tárolt dokumentumok mennyisége és a tartalom tárolási formátuma miatt szintén nehéz feladat. Végül a keresés eredményét úgy kell megjeleníteni, hogy az a felhasználó számára hasznos információkat nyújtson, itt kap szerepet a rangsorolás és a megjelenítés módja.

A keresőrendszerek minőségének vizsgálatánál beszélhetünk a keresőrendszer (1) széleskörűségi, (2) naprakészségi, (3) rangsorolási és (4) megjelenítési követelményeiről.

A *széleskörűség* megköveteli, hogy a keresőrendszerben az összes fontos oldal elérhető legyen a felhasználó számára. A világhálót — weblapok igen nagy száma és változási dinamikája miatt — tekinthetjük a weblapok végtelen halmazának. A cél az, hogy minél teljesebb indexet készítsünk erről a sokaságról, miközben a keresőrendszer rendelkezésére álló erőforrások (tárolókapacitás és hálózati sávszélesség) korlátozottak. A követelmény teljesítéséhez szembe kell nézni a kiválasztási problémával, amely szerint két aspektusban kell döntéseket hoznia a rendszernek: mely lapokat látogassa meg az adatgyűjtő webrobot, és mely oldalakat indexelje az indexelő. A keresőrendszerben az adatgyűjtési elv írja le azt a szabályrendszert, amelynek alapján az adatgyűjtő által meglátogatandó lapok kiválasztásra kerülnek. Gyakran egy indulókészletet definiálnak, amelyből kiindulva keresi fel a webrobot az oldalakat. A meglátogatott lapokból kigyűjti a hiperhivatkozásokat, majd meghatározza a további oldalak felkeresésének sorrendjét. A rendezés alapja az adott lap és a hivatkozott lapok látogatottsági adatai. A legtöbb rendszer a sorrendképzés minőségi hangolására is lehetőséget ad.

A *naprakészség* követelménye azt jelenti, hogy a keresőrendszer indexe mindig aktuális legyen, azaz hűen tükrözze a web állapotát. Teljes naprakészséget elérni a web mai dinamizmusa mellett gyakorlatilag lehetetlen. Az index folyamatos frissítésére van tehát szükség, amit úgy lehet megoldani, hogy a webrobot időről-időre visszatér a már indexelt oldalakra, és ismételten letölti azokat. Ily módon lehetőség nyílik a már indexelt oldalak változásainak nyomkövetésére. Mivel az oldalak *újralátogatása* (revisit) is a webrobot feladata, így egyensúlyt kell terem-

teni az új lapok felkeresése és a már indexelt lapok újralátogatása között. Heurisztikus irányelv kialakítására ad lehetőséget az a megfigyelés, hogy a weblapok egy csoportja szinte állandóan változik, míg más oldalak statikusnak tekinthetők.

A *rangsorolás*i követelmény lényege az, hogy egy adott lekérdezésnél a legnagyobb relevanciával bíró találatok kerüljenek előre a válaszlistában. Ehhez minden találatnak meg kell határozni a relevanciaértékét. Itt két forrás áll rendelkezésre: a szerkesztőség, vagyis a keresőrendszer üzemeltetőjének humán erőforrása, és a felhasználók viselkedésének naplózása. A rangsorolási problémán az egyes rangsorolási jellemzők valószínűségi alapú meghatározását értjük. A rangsorolási jellemzőket két csoportba oszthatjuk: a lekérdezéstől függő és az attól független jellemzőkre. A lekérdezéstől függő rangsorolási jellemzőt az határozza meg, hogy a lekérdezés mennyire illeszkedik a találat tartalmára vagy metaadataira. A nagyobb illeszkedés nagyobb rangot jelent, és ezáltal a lap a találati listán előrébb kerül. A lekérdezéstől független rangsorolási jellemzők az egyes oldalak minőségi jellemzőiből és a spamszűrő által adott értékből tevődnek össze. Fontos látni, hogy a rangsorolási jellemzők az első esetben a lekérdezéssel egyidejűleg kerülnek kiértékelésre, míg a második esetben előre meghatározhatóak.

A lekérdezéseknél nagy hangsúllyal bír a kéréskiszolgáló modul válaszideje. A lekérdezés végrehajtása több elemi tevékenységből áll össze, s emellett a begyűjtött információk aktualizálása is folyamatos igényként jelentkezik. A megfelelően kis válaszidő eléréséhez a keresőrendszerek a lekérdezéseket elosztott rendszeren dolgozzák fel, és az indexadatbázist egy nagy teljesítményű adatbázisklaszterre telepítik.

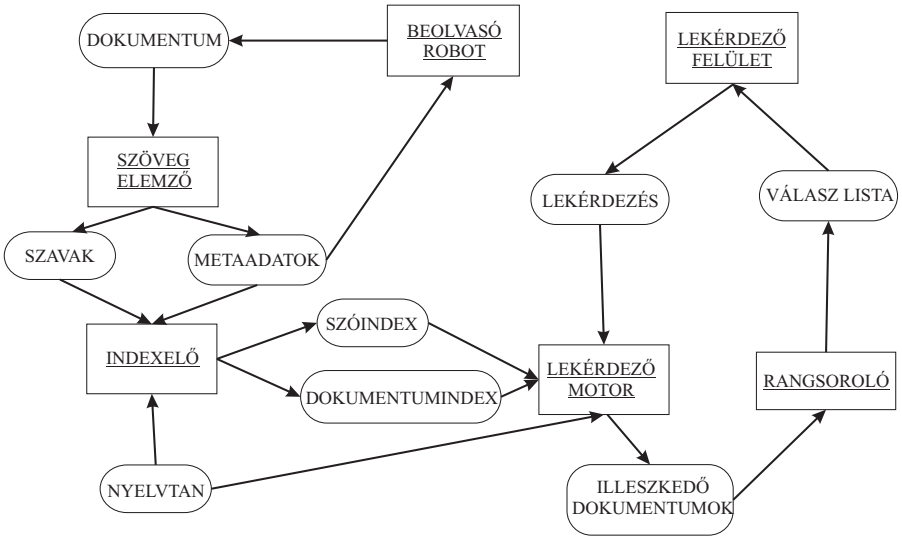
Az utolsó követelmény a keresés eredményének *megjelenítés*ére vonatkozik. Az eredményeknek jól olvasható formában kell megjelenniük, a találatokat tartalmazó oldalon szerepelnie kell a találat címének, kivonatának, valamint a találatot közvetlenül elérhetővé tevő hiperhivatkozásnak is. A kifinomultabb elvárások közé tartozhat a nyelvi helyesség, azaz az esetlegesen rosszul írt vagy elgépelt keresési kulcsszavak automatikus javítása — akár a találati oldalon is.

8.3. A keresőmotorok struktúrája

A mai keresőmotorok általános struktúrájában az alábbi feladatokat kell alapvetően megoldani (ld. 8.2. ábra):

- feldolgozandó dokumentumok kijelölése;
- dokumentumok letöltése;
- dokumentumok szóhalmazának előállítása;

- dokumentumok metaadatainak előállítása;
- dokumentumok indexelése;
- dokumentumok hatékony tárolása;
- keresési adatszótárak nyilvántartása;
- keresőmotorok megvalósítása;
- keresési metaadatok feldolgozása.



8.2. ábra. Keresőmotorok általános sémája

Logikailag a keresőrendszer két komponenscsoportra osztható. Az egyik azokat a modulokat tartalmazza, amelyek a web felderítéséért, indexeléséért felelősek, a másik csoportbeliek a felhasználói lekérdezések kiszolgálását biztosítják.

A betöltő robot feladata a web böngészése, és az elért dokumentumokról URL-jük szerint *pillanatkép* (snapshot) készítése, tárolása. Ezt a modult szokták *webrobotnak*, *aratórobotnak* (harvester) vagy *crawlernek* nevezni. Egy keresőrendszer általában több webrobotot is működtet párhuzamosan, amelyek osztott adatbázisba mentik a pillanatképeket. A betakarított dokumentumokat további két modul dolgozza fel: a dokumentum hiperhivatkozásait a *webtérképkezelő* modul, szöveges tartalmát pedig az *indexelő* modul. A webtérképmodul feladata az, hogy a web dokumentumairól és a köztük lévő hiperhivatkozásokról térképet készítsen.

A térkép rendszerint egy irányított háló, amelyben a csomópontok a dokumentumokat, az élek pedig a hiperhivatkozásokat reprezentálják. A világháló térképét a *metaadatok adatbázisa* tárolja. Az indexelő modul feladata a web indexállományának előállítása, amelyet a begyűjtött dokumentumok szöveges tartalmának szövegbányászati elemzésével és a web kapcsolatrendszerének feltárásával épít fel. Az indexelő modul kimenete a webindexet tároló adatbázis lesz, amely a felhasználói lekérdezések kiszolgálásának alapja.

A Google keresési algoritmusának egyszerűsített vázlata [30] alapján a webes dokumentumok feldolgozása a következő lépésekből áll:

- a dokumentum feldarabolása szavakra, operátorokra (tokenizálás);
- szavak átalakítása belső szóazonosítókra;
- a dokumentumok hozzárendelése belső dokumentumazonosítókhoz;
- a szóhoz tartozó előfordulásokat nyilvántartó invertált index elkészítése;
- a szóhoz tartozó metaadatok kigyűjtése;
- kapcsolati indexek létrehozása.

A keresés bemenete a keresőkifejezés. Az illeszkedő dokumentumok meghatározása a következő általános séma szerint megy végbe:

- keresőkifejezés elemzése, felbontása szavakra;
- szavak konvertálása a megfelelő nyelvtani alakra;
- illeszkedő dokumentumok meghatározása az invertált lista alapján;
- az illeszkedő dokumentumok rangsorba állítása;
- találati lista limitálása;
- limitált találati lista visszaküldése.

A Google az indexelt dokumentumokat tömörített formában a saját szerverein tárolja. Tömörítésre a Zlib könyvtárat használják, amely megfelelő egyensúlyt biztosít a tömörítési arány és a feldolgozási sebesség között.

8.3.1. Webrobot – webes begyűjtő

Keresőrendszerekben a webrobotnak hármas szerepe van: (1) a világháló kapcsolatrendszerének felderítése, (2) dokumentumok letöltése indexelés céljából, és (3) a már indexelt dokumentumok ismételt letöltése az index frissítése, aktualitásának megőrzése céljából. A világháló két olyan sajátossággal is rendelkezik, ami ezt a feladatot megnehezíti. Egyrészt a világháló mérete, dokumentumainak igen

nagy száma, másrészt pedig a dinamizmusa, vagyis tartalmának gyors változása. Nap mint nap új dokumentumok jelennek meg, meglévő dokumentumok tartalma változik és dokumentumok szűnnek meg — mindez a dokumentumok számának állandóan növekvő trendje mellett. A dokumentumok rendkívül nagy száma miatt a robot nem képes az összes dokumentum letöltésére. A változás gyorsasága miatt pedig a robot által letöltött dokumentum gyorsan elavulhat. További megkötést jelent a robotok számára, hogy a rendelkezésre álló hálózati erőforrások korlátozottak. Figyelembe véve, hogy a felhasználható sávszélesség korlátosságát is, a web dokumentumainak begyűjtését szabályozó algoritmus hatékonysága alapvető követelmény [58]. Nézzük meg a webrobotok összetett viselkedését befolyásoló tényezőket, amelyek az alábbi elvek mögé sorakoztathatók fel:

- a *kiválasztási elv* (selection policy) szabályozza a letöltendő dokumentumok kiválasztását;
- az *újralátogatási elv* (re-visit policy) szabályozza, hogy melyik dokumentumokat kell újra letölteni az index frissítése céljából (tartalmi nyomkövetés);
- az *udvariassági elv* (politeness policy) szabályozza, hogy mekkora többletterhelést okozhat egy robot az egyes webszervereken;
- a *párhuzamos feldolgozási elv* (parallelization policy) összehangolja a párhuzamosan dolgozó robotok munkáját, hogy azok osztott rendszerként működjenek.

8.3.1.1. Kiválasztási elv

Nagy mérete miatt lehetetlen a világháló teljes tartalmának letöltése. A kiválasztási elv feladata a világháló lényeges tartalmi elemeinek *kiválasztása*. Ez a dokumentum fontossága alapján valósul meg, ami függ a dokumentum tartalmának minőségétől, látogatottságától, URL-jétől és a rámutató linkektől. Ha csupán egy vagy néhány domén URL-jeit indexelik, vertikális keresőrendszerekről beszélünk. A kiválasztási elv alapját néhány elterjedt bejáró algoritmus adja, amelyek korlátozhatják vagy bővíthetik a robot által meglátogatandó linkek listáját.

A letöltött dokumentumok linkjeit a robot megvizsgálja, hogy megakadályozza a nem szöveges tartalmú dokumentumok letöltését. Ennek érdekében http head lekérdezést használ, amely visszaadja az URL-en található dokumentum tartalmának MIME típusát. A headkérések számának csökkentése érdekében a robot megszűri az URL tartalmát és a szokásosan HTML-tartalmú kiterjesztéseket keresi (pl. html, htm, asp, aspx, php, és az URL-t záró per jel). Sok robot kerüli a kérdőjel tartalmú URL-ek letöltését, mivel azok többnyire dinamikusan előállított

dokumentumokat adnak vissza, amelyek — akár rosszindulatúan is — befolyásolhatják a robot működését. Ebben a tekintetben a Google robotja kivétel, mivel éppen ezt használja ki a web rejtett tartalmának felderítésére.

A webrobotok speciális típusa az *aratórobot*, amely a letöltendő URL-ből megkísérel több kapcsolódó URL-t előállítani úgy, hogy az URL-útvonalat inkrementálisan darabolja.[2] Az összes letöltött dokumentumból ismételten kinyeri a linkek URL-jét és azok mentén folytatja az oldalak bejárását. Ezzel a módszerrel a robot képes a hierarchia bármely eleméből kiindulva a teljes hierarchiát letölteni.

Webdokumentumok fontosságát mérhetjük egymáshoz való hasonlóságuk alapján is, azt feltételezve, hogy az összetartozó információkat hasonló kinézetű oldalakra tördelik. Az ilyen, adott kinézetű dokumentumokat kereső robotot, *fókuszált robot*nak nevezik. A kiválasztás nehézsége abban rejlik, hogy a hasonlóság két dokumentum között nehezen állapítható meg a dokumentum letöltése előtt. Egy lehetséges előrejelzést adhat például a link szövege a dokumentumra hivatkozó oldalon.

A *mélyháló* (rejtett vagy láthatatlan háló, angolul deep, invisible vagy hidden web) a világháló azon része, amely közvetlenül nem látható a robotok számára. Olyan dokumentumokra gondoljunk, amelyek csak valamilyen űrlap kitöltésével, adatbázisból való lekérdezés útján érhetők el, esetleg a világháló kapcsolatrendszerében izoláltan vannak jelen. A mélyhálót csak speciálisan erre kifejlesztett algoritmust alkalmazó robotok képesek elérni és felderíteni, ld. pl. a Google Sitemap Protocolt. A témával részletesen foglalkozunk a 9.2. szakaszban.

8.3.1.2. Újralátogatási elv

A világhálónak még a részleges bejárásához is hetek, hónapok kellenek. Ennyi idő alatt azonban a web dokumentumainak jelentős része megváltozik. A változások, melyek új oldalak létrehozását, oldalak tartalmának megváltoztatását vagy oldalak törlését jelentik, csak az oldalak rendszeres *újralátogatásával* kerülhetnek be a keresőrendszer indexébe. A robot tehát két okból látogathat meg egy oldalt: első letöltés vagy újralátogatás céljából. Teljesítményhangolás szempontjából fontos az *újralátogatási arány* értéke, amely megadja, hogy a keresőrendszer robotja milyen arányban végezzen újralátogatást az összes letöltésre vonatkoztatva. Magas

[2] Például a `www.uni-miskolc.hu/uni/dept/faculties/indexe.shtml` URL alapján a robot letölti a `www.uni-miskolc.hu/uni/dept/faculties/`, `www.uni-miskolc.hu/uni/dept/`, `www.uni-miskolc.hu/uni/` és `www.uni-miskolc.hu/` oldalakat is arra számítva, hogy ott összetartozó, és általánosabb információkat talál.

újralátogatási arány esetén az index növekedésének sebessége csökken, ugyanakkor a megszűnő oldalak újralátogatásával azok bejegyzései törlődnek az indexből. Értelmezzük még az újralátogatási terhelés fogalmát is, amely az újralátogatásra váró oldalak számával áll arányban.

Az újralátogatási elv alapját az *öregedési algoritmus* (aging) képezi, amely az oldalakhoz és a belőlük képzett indexértékekhez rendel életkort. Ha az oldal aktuális tartalma nem felel meg a korábban létrehozott indexbejegyzésnek, akkor érvénytelenítjük a bejegyzést. Általánosan két minőségi követelményt fogalmazhatunk meg a keresőrendszer indexével kapcsolatban. Egyrészt felső korlátot adhatunk az index bejegyzésének korára, másrészt az index érvénytelen bejegyzéseinek arányára. Mivel a keresőrendszer számára az indexbejegyzés érvénytelenné válása, azaz az oldal megváltozásának időpontja ismeretlen, nincs lehetőség a minőségi követelmények pontos teljesítésére. Az újralátogatási elv meghatározása valószínűségi modell alapján történik, amelynek két alapváltozata létezik.

Uniform újralátogatási elv esetén az oldalak újralátogatásának valószínűsége az indexbejegyzés korával arányos. Ez az elv minden oldalra egyformán érvényes, vagyis minél régebben lett letöltve az adott oldal, annál nagyobb valószínűséggel látogatja meg újra a kereső. Mivel minden indexbejegyzés öregszik, könnyű belátni, hogy a bejegyzések számának növekedésével arányosan a robot újralátogatási terhelése is nő. Emiatt az új oldalak felderítési követelménye összeütközésbe kerül az oldalak újralátogatásának igényével. A minőségi kívánalmak teljesítése érdekében tehát növelni kell a robot letöltési teljesítményét. Másrészt látható az is, hogy a létrehozott index a világháló méretével együtt növekedni fog. A robot letöltési teljesítményének az index és a világháló együttes növekedésével kellene lépést tartania.

Az *arányos* újralátogatási elv alkalmazásával minden dokumentumhoz *újralátogatási gyakoriság*ot rendelünk, melynek értékét számos szempont figyelembe vételével állítja be a keresőrendszer. A fontosabb hangolási szempontok:

- az oldal tartalmának dinamizmusa, vagyis ha egy oldal az újralátogatások során nem változik, akkor csökken az újralátogatás gyakorisága;
- az oldal rangja, mely heurisztikusan növeli az újralátogatás gyakoriságát, így a magas rangú oldalak indexének frissítése magasabb prioritást kap;
- az oldal relevanciája, vagyis az a tulajdonsága, hogy a találatok közül milyen gyakran választják a felhasználók az adott oldalt;
- az oldal tartalmának jellege, amely egy heurisztikus módszeren keresztül befolyásolja az oldal újralátogatási gyakoriságát.

8.3.1.3. Udvariassági elv

A robotok sokkal gyorsabban képesek bejárni a webet, mint az emberek, akik egy letöltött dokumentum olvasásával gyakran hosszasan elidőznek. A robotok igen nagy intenzitással küldhetik kéréseiket a webszerver felé, ami akár le is bénítha-tja azt. Ennek elkerülése érdekében az *udvariassági elv* korlátozza a robot egy webszerverre jutó lekérdezésének sűrűségét. A terhelés azonban nemcsak a webszervereken jelentkezik, hanem a hálózati átjárókon is, emiatt a robotnak ügyelnie kell a hálózati útvonalak terhelésére is. Az udvariassági elvvel kapcsolatosan az alábbi két fontos etikai szabályt fogalmazták meg:

- A robot egy adott webszerveren ne okozzon jelentősen nagyobb terhelést, mint egy humán felhasználó.
- Bármely weblap gazdájának legyen lehetősége a weboldalára látogató robot működését szabályozni, és a robot ezt a szabályozást vegye figyelembe. Egyezményesen a weboldal gyökerében elhelyezett robots.txt állományban adható meg ilyen szabályozás, és az etikus robot legelőször ezt az állományt tölti le a weboldalról. A HTML-dokumentumok fejlécében is lehetőség van a dokumentumra vonatkozó szabályok megadására.

8.3.1.4. Párhuzamos feldolgozási elv

A robotok hatékonyságának növelése érdekében úgynevezett párhuzamos robotokat alkalmaznak. Ezek a robotok több példányból állnak, melyek több szálon, esetleg több különálló gépen futnak. A cél a minél nagyobb letöltési teljesítmény elérése, ugyanakkor a párhuzamosítás többletköltségeinek minimalizálása, és egyazon oldal többszörös letöltésének megakadályozásához szükség van a *párhuzamos feldolgozási elv*re. A párhuzamosan futó robotok alkalmazásának előnye, hogy így a hálózati kapcsolat terhelése is megoszlik, mivel ugyanannak a keresőrendszernek a robotjai a világháló több különböző pontjáról dolgozhatnak.

A többszörös letöltéseket a *hozzárendelési elv* segítségével akadályozzák meg, amely szabályozza, hogy melyik URL-t melyik robot dolgozza fel. Általánosságban beszélhetünk dinamikus és statikus hozzárendelési politikáról. Dinamikus hozzárendelés esetén a robotokat egy központi szerver látja el URL-listákkal, gondoskodva róla, hogy azok ne tartalmazzanak ismétlődéseket. Ez lehetőséget ad a robotok számának dinamikus változtatására és terhelésük dinamikus megosztására. A dinamikus hozzárendelés hátránya, hogy a központi URL-kiosztó szerver szűk keresztmetszetet jelent, mivel annak túlterhelése vagy kiesése kihat az összes robot működésére, és drasztikusan rontja a letöltési teljesítményt.

Statikus hozzárendelés esetén a robotpéldányok rögzített szabályok alapján választanak céldokumentumot egy osztott URL-listából. A szabályt általában egy hash-függvény írja le, mely az URL alapján kiválaszt egy robotpéldányt. A hash-függvénnyel szemben az az elvárás, hogy a rendelkezésre álló robotpéldányokhoz egyenletesen rendeljen URL-eket. A statikus hozzárendelés azonban nem biztosítja a robotok terhelésének egyenletes eloszlását, mivel az egyes URL-ek mögötti dokumentumok letöltési költsége jelentősen eltérhet egymástól. Ugyanakkor a módszer előnye, hogy a hash-függvény megvalósítható minden robotpéldányban, így decentralizáltan, a központi erőforrásoktól függetlenül működhetnek. Nehézséget jelenthet a robotpéldányok számának megváltozása, mert ilyenkor a hash-függvénynek is változnia kell, és a változást az összes robotpéldánynak egyszerre kell átvenni.

Statikus hozzárendelés egy lehetséges hash-függvénye az alábbi:

$$H_n(U) = \left(\sum_i u_i \right) \mod n,$$

ahol $U = (u_1, u_2, \ldots)$ a sztringként értelmezett URL, melynek karaktereire u_i-vel hivatkozunk. A H_n hash-függvény az n robotpéldány valamelyikéhez rendeli U-t (a robotokat 0-tól $n-1$-ig számozzuk).

8.4. A dokumentumok indexelése

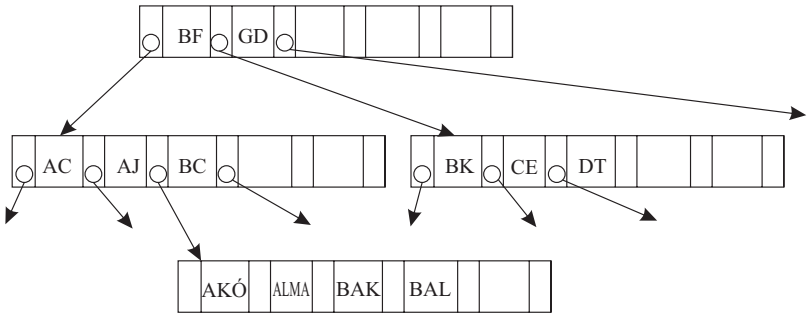
Az automatikus dokumentumindexelési módszereknél több sarkalatos kérdés merül fel az indexelés technikájára vonatkozólag: (1) milyen adatelemeket indexeljünk; (2) milyen struktúrában tároljuk az adatokat; (3) milyen jellegű keresést kell kiszolgálni az index alapján, és (4) mikor indexeljünk.

8.4.1. Adatstruktúrák

A dokumentumok ábrázolásának egyik klasszikus formája a *vektortérmodell* (ld. a 2.2.2. szakaszt) [22]. A tapasztalat szerint [30] azonban ez a módszer nem ad megfelelő eredményt a webes dokumentumkeresési feladatoknál. Ugyanis a klasszikus vektortávolság alkalmazásakor a keresőkifejezéshez méretben is hasonlító, rövid dokumentumok kerülnek be elsődlegesen a találati listába, illetve a nagyobb távolságú dokumentumok esetében nagy a téma szerinti szórás a találatok között. Ennek az a fő oka, hogy a modellben minden szó, a megközelítés nem tud különbséget tenni a témakör szerint fontos és lényegtelen szavak között. A webes keresés tehát egy sajátos távolságértelmezést igényel, melyben lehetőség

van a témakör alapján vett távolság alkalmazására, ezáltal biztosítva azt, hogy a keresett témához kapcsolódó dokumentumokat kapjuk vissza. Emellett fontos a dokumentumok közötti hiperlinkrendszer bevonása is, hiszen a kapcsolatok hatékonyan felhasználhatók a dokumentumok lényegkiemelésére és a találatok rangsorolására is.

Az adatbázisok klasszikus indexelési technikája a *B-fa* struktúrán alapul [103]. A B-fa egy kiegyensúlyozott indexfa, melyben egy csomópont rendszerint egy lemezblokk méretű. A blokkban a kulcsértékek és a megfelelő részfákra mutató pointerek helyezkednek el (ld. a 8.3. ábrát). A pointerek és indexek kapcsolata megfelel az általános keresőfa-struktúrának. A fa sajátossága, hogy alulról felfelé nő, azaz egy új elem beszúrásakor előbb leviszi azt a megfelelő levélbe, majd ha az megtelt, a középső elemet kiemelve és a szülőbe áthelyezve egy testvérlevelet hoz létre. Egy kulcsérték átlagos keresési költsége $O(\log_M(N))$, ahol M a fa fokszáma és N az indexelt elemek száma.[3]



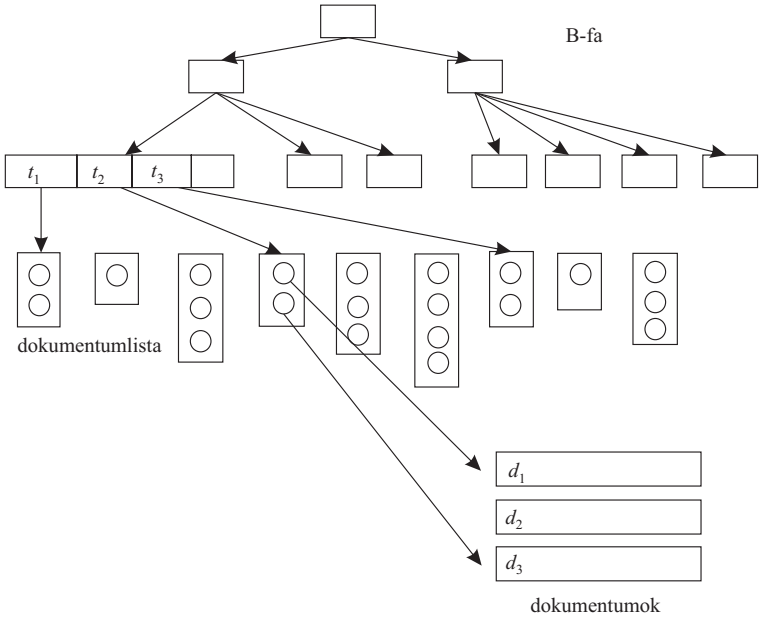
8.3. ábra. B-fa struktúra

Az indexelés egy másik fontos változata az *indexszekvenciális szervezés* (ISAM) [103]. Az ISAM-struktúrában a rekordok egy megadott kulcs alapján sorbarendezetten helyezkednek el. A keresés gyorsítása érdekében az alapadatokhoz egy indexet is létrehoznak. Ez az index általában a tárolt kulcsoknak csak egy részét tartalmazza. Így egy indexbejegyzés a folytonos tárterület egy tartományát jelöli ki. Minden tartományhoz hozzárendelhető egy egyértelmű kulcsértéktartomány. A keresés során a keresett elemhez közeli indexelt elemek pozícióját határozzuk meg elsőként, majd a keresés az így kijelölt blokkokban folytatódik.

A klasszikus B-fa indexelésnél egy kulcsértékhez egy rekord előfordulási helyet kijelölő pointer társul. Ez a struktúra jól illeszkedik a hagyományos adatbá-

[3] Ebben a szakaszban N tehát nem a dokumentumok számát jelöli.

zisok világára, hiszen ott az egyedintegritás elve miatt a kulcsok egyediek, csak egyszer fordulnak elő az adatbázistáblában. A dokumentum kezelésénél a legkézenfekvőbb módszer az, amikor a dokumentumokat a bennük előforduló szavak alapján indexeljük. Az indexelés célja, hogy egy kulcsszóhoz hatékonyan megtaláljuk az azt tartalmazó dokumentumokat. Mivel itt egy szóhoz, azaz a kulcshoz több előfordulás is társul, ezért egy olyan indexstruktúra épül fel, melyben minden szóhoz az azt tartalmazó dokumentumok azonosítóinak láncolata kapcsolódik. Ezt a szerkezetet *invertált indexstruktúrának* [103] nevezik. A dokumentumok gyors megtalálásához a tartalmazott szavakat egy keresőfába rendezzük, majd a szóhoz a szokásos pozíciópointer helyett egy listát rendelünk. A lista egy bejegyzése alapesetben egy dokumentumot jelöl. Az invertált indexstruktúra szerkezetét a 8.4. ábra szemlélteti. Az ábrában a w egy kulcsszóra utal, míg a d-lista a szimbólum előfordulási pozícióját tároló listát jelöli.

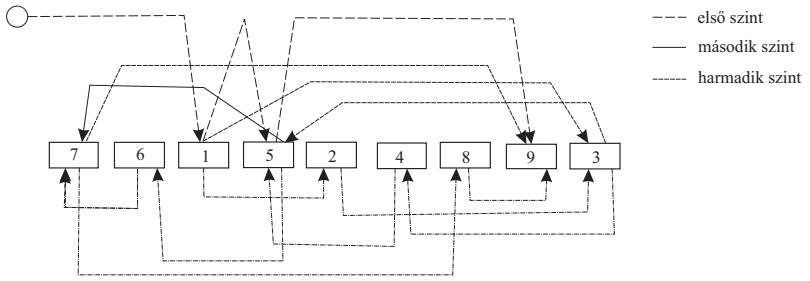


8.4. ábra. Invertált indexstruktúra

Az invertált index esetében a legnagyobb feladat az előfordulási, úgynevezett pozíciólista karbantartása. Mivel a lista tetszőleges hosszúságú lehet a különböző kulcsértékeknél, ezért ennek kezelésére külön módszerek jöttek létre. A legegyszerűbbnek tűnő megvalósítás, hogy az egyes listákat közvetlenül egymás után

tároljuk, ám ez a lista módosításakor igen nagy költséget jelentene, hiszen ekkor egy lista bővítése a mögötte álló listák áthelyezését igényelné. Ez a változás az adatmozgatás mellett még azzal a plusz feladattal is járna, hogy aktualizálni kellene az érintett listák kulcsbejegyzéseinél a pointerek értékét. Mindez a lista hosszával lineárisan arányos költséget jelent, ami nagy méretű problémák esetében nem elfogadható.

A probléma egy szokásos megoldása, hogy az egyszer letett bejegyzéseket nem mozgatjuk át, ehelyett egy pointerláncon keresztül kapcsoljuk össze az egymást követő elemeket. Egy pointer alapesetben két szomszédos elemet köt össze. Ekkor az új elemet a listák végére tesszük, majd megkeressük a logikai helyét és aktualizáljuk a pointerláncot. Ez a megvalósítás jól használható a lista szekvenciális bejárására, viszont nem alkalmas az elemek gyorsabb, például bináris keresésen alapuló megkeresésére. Ezen feladat megvalósítása érdekében úgy módosítják a pointerláncot, hogy az alap pointerlánc mellett magasabb szintű pointerláncokat is létrehoznak a listához. Egy magasabb szintű, ún. *ugró pointer* (skip pointer) már nem két szomszédos elemet köt össze, hanem két egymástól távolabb elhelyezkedő elemet, mintegy ugrást téve a listában. Az így kialakult listát *skiplistának* is nevezik (ld a 8.5. ábrát). A bináris keresés támogatása esetén a skiplista legfelső szintje a két szélső és a középső elemet köti össze. Egy szinttel lejjebb az aktuális intervallumok felezőpontjai is belekerülnek a listába. Ía skiplisták elemszáma lefelé duplázódik. A teljes skiplista létrehozásához $O(L)$ nagyságrendű pointerre van szükség, ahol L a listabejegyzések száma. Fontos, hogy ekkor a skiplista minden szintje egy adott szempont szerint rendezett listát kezel. Közvetlen elemkeresésnél ekkor a legfelső szintből kiindulva, majd mindig az alatta lévő szintre átlépve lehet egy elemre leszűkíteni a vizsgált intervallum hosszát. Egy elem beszúrásának és keresésének költsége $O(\log L)$ nagyságrendű.



8.5. ábra. A skiplista felépítése

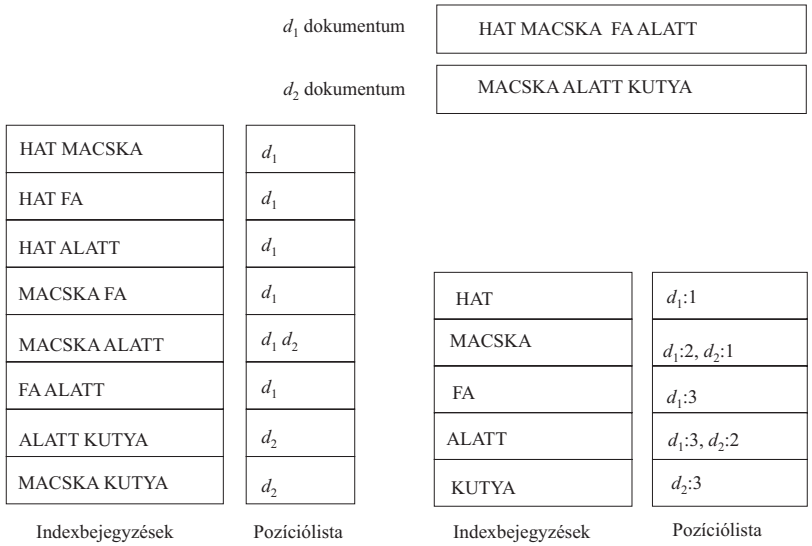
Abban az esetben, ha a keresőkifejezés nem csak kulcsszavakat, hanem kifejezéseket[4] is tartalmaz, akkor figyelembe kell venni a szavak dokumentumon belüli pozícióit is. Ennek egyik megoldása az ún. *biword index* alkalmazása. Ez az index nem szavakat, hanem szópárokat tartalmaz. A szópár első tagja előbb helyezkedik el a dokumentumon belül, mint a második tagja. A biword index egyik alapparamétere, hogy milyen távoli szavakhoz készüljön szópárbejegyzés. Ha minden pár feljegyzésre kerül, akkor $O(K^2)$ nagyságrendű páros jön létre egy dokumentumhoz, ahol K a dokumentum szavainak számát jelöli. Kereséskor a keresőkifejezéshez is elkészítik a kapcsolódó szópárokat. Például a *Péter almát eszik* keresőkifejezés esetében az indexben keresendő elemek a (*Péter*, *almát*), (*Péter*, *eszik*) és az (*almát*, *eszik*) szópárok lesznek. A biword index alapján gyorsan meghatározható, hogy egy kifejezés szerepel-e a dokumentumban, a tárolás azonban igen nagy tárigényű (ld. a 8.6. ábrát).

Egy másik megközelítés a probléma megoldására, amikor a szavak előfordulásai mellett a dokumentumon belüli pozíciókat is tároljuk. A *pozícióindex* esetében egy bejegyzés így egy szóelőfordulás helyét adja meg. A pozícióindex hossza az invertált indexhez képest lényegesen nagyobb, hiszen egy dokumentumhivatkozás helyett általában több előfordulás-hivatkozás is szerepel benne. Cserébe viszont a szó minden előfordulása elérhető, és egyszerű számítással az is meghatározható, hogy a vizsgált kifejezés szavai a megfelelő sorrendben fordulnak-e elő a dokumentumban.

A *kifejezés alapú indexelésnél* az egyik fontos szempont az, hogy mit tekintsünk pontosan kifejezésnek. Két fontos paraméter van: az egyik a szavak közötti távolság, a másik a kifejezés teljes hossza. Az első szempontnál az egyik lehetőség az, hogy csak a pontosan egymás után következő szavak alkothatnak kifejezést, a másik pedig az, hogy a szavak között *ugrások* (gap) is megengedettek. Az ugrások bevonása jelentősen megnöveli a végrehajtási költséget, hiszen ez jóval több kifejezést eredményez. A biword indexelés mellett léteznek olyan rendszerek is, melyben szóhármasokat, szó n-eseket indexelnek.

Egyedi indexelési technikára van szükség azon esetekben is, amikor a keresőkifejezésekben nem a teljes szót adjuk meg, hanem annak csak egy részét. Például az összes olyan dokumentumra keresünk, amelyben szerepel az *autó* sztring valamely szó részeként. Ennek szó eleji előfordulásait találjuk például az *autókereskedő*, *autótolvaj* vagy az *autóverseny* szavakban, ekkor a keresőkifejezés rendszerint *autó** alakban adott. Általánosabb keresésnél nem feltétlenül kötjük

[4] szóközt is tartalmazó szószekvenciát; a gyakorlatban többnyire idézőjelek között kell megadni



8.6. ábra. Biword index (balra) és pozícióindex (jobbra) összehasonlítása

meg, hogy az adott sztring a szó elején szerepeljen, ekkor találat lesz a *mentőautó* és a *sínautókról* szó is.

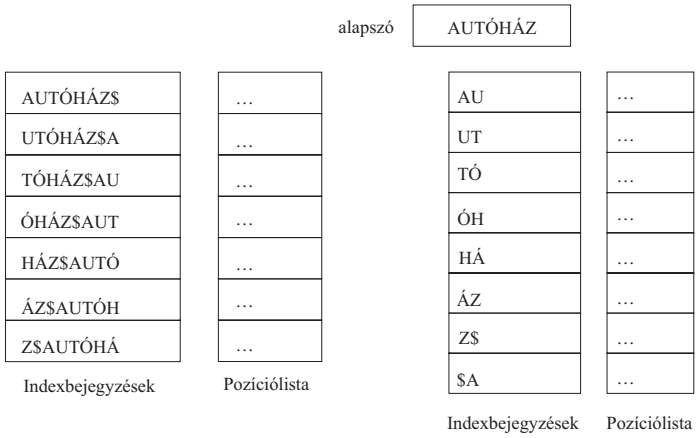
Az ilyen jellegű lekérdezések hatékony kezelésére szolgál a *permuterm index* (ld. a 8.7. ábrát). Az elnevezés arra utal, hogy az indexstruktúrába nemcsak az alapszót, hanem annak rotációit is belevesszük. A szóvég jelzésére egy külön szimbólumot, a $ jelet szokták alkalmazni.

8.1. PÉLDA. Az *autóház* szó esetében az alábbi rotációkat tároljuk el:

autóház$, utóház$a, tóház$au, óház$aut, ház$autó, áz$autóh, z$autóhá

Ekkor a **ház* keresőkifejezéssel olyan szavakat keresünk, melyben a *ház* sztring a szó végén van. A szokásos B-fa alapú index használata mellett erre a kifejezésre a rotációk közül a *ház$autó* szó illeszkedik, melyből rögtön megkapjuk az *autóház* alapszót is. A permuterm index hátránya, hogy az indexbejegyzések számát az átlagos szóhossz faktorral megtöbbszörözi.

A részszóra való keresést támogatja az *n-gramm index* is. Ennél az indexelési formánál egy szóhoz annak összes n hosszúságú részsztringjét tároljuk és indexeljük. Például bigramokat használva az *almafa* szó esetén a következő bejegyzések kerülnek be az indexbe: *al, lm, ma, af, fa.* Az n-gramm alapú keresésnél a keresésőkifejezést felbontjuk n-grammokra, és ezeket keressük meg az indexállomány-



8.7. ábra. Permuterm (balra) és bigram (jobbra) indexek összehasonlítása

ban. A kapott találati listák feldolgozásával, a listák metszeteként meghatározható a tényleges találati lista. A módszer hátránya, hogy itt is nagy indexállományra és igen tekintélyes utómunkákra van szükség a végleges találati lista előállítására. Az indexbejegyzések száma itt is az átlagos szóhosszal többszöröződik.

Nagyobb méretű pozíciólisták esetén már magának a listának a kezelése is hatékony algoritmust igényel. Az egyik hatékonyságnövelő mechanizmus az adattárolás megosztása. Ekkor a logikai indexstruktúra több csomópontra szétosztva valósul meg, és így az indexkezelő feladatok párhuzamosíthatók. A megosztás irányításához szükség van egy vezérlő csomópontra is. Ennek egyik feladata az indexelési részfeladatok kiosztása a feldolgozó csomópontoknak. Mivel egy feldolgozó csomópont működése bármikor leállhat, a feladatok kiosztása dinamikusan, az aktuális rendelkezésre állási és terhelési adatok figyelembe vételével történik. A párhuzamos hozzáférések okozta zárolások minimalizálására a listaállományt is partíciókra bontják. Ha a feldolgozók nem közös, hanem lokális adatbázissal dolgoznak, akkor a lokális listák elkészítése után szükség van a listák egyesítésére is. Rendezett listák esetén az egyik legismertebb módszer a *block merge*, melyben két rendezett listát (blokkot) fűz össze egy új listába (blokkba). A módszer alapelve, hogy egy-egy pointert állítunk a két lista elejére. Ha az elemeket növekvő sorrendbe kívánjuk rendezni, akkor a két kijelölt elemet összehasonlítva a kisebb értékűt tesszük előbb át az összefűzött listába, és ennek a pointerét eggyel előre léptetjük. Ezt addig folytatjuk míg a listák végére nem érünk. Az összefűzés költsége $O(L)$ nagyságrendű, itt L szintén a listabejegyzések számát jelöli.

A dokumentumkereső-rendszerekben a dokumentumokhoz nemcsak a bennük lévő szavak listáját tárolják, hanem egyéb metaadatokat is. Ilyen leíró adat lehet például a dokumentum keletkezési dátuma, a létrehozó vagy szerkesztő neve. A keresőkifejezés ezen adatokra is vonatkozhat, emiatt az indexelésnél ezen attribútumokra is gondolni kell. Az ilyen attribútumok indexelésére a B-fa index mellett a bitmap index használatos. A *bitmap index* esetén egy bináris mátrixot tárolunk, melynek egy sorai a kulcsértékeket, oszlopai pedig az objektumoka (itt dokumentumok) reprezentálják (ld. a 8.8. ábrát). A mátrix egy cellájának értéke azt mutatja, hogy az adott objektumnál a kulcsérték teljesül-e vagy sem. Mint látható, ez a módszer akkor előnyös, ha a kulcsértékek száma viszonylag kicsi. Ilyen lehet például a dokumentum formátuma. Előnyös a mátrixtárolás tömörsége, valamint az, hogy lehetővé teszi a logikai operátorokat is tartalmazó kifejezések gyorsabb kiértékelését. A normál bitmap index mellett használatos még az intervallum bitmap és a kompozit bitmap indextípus is [125].

dokumentumok

dokumentum-formátum	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}
DOC	0	0	0	1	0	0	1	0	0	0	0
TXT	1	0	0	0	0	1	0	0	1	0	0
PDF	0	1	0	0	0	0	0	1	0	1	0
XML	0	0	0	0	1	0	0	0	0	0	0
DBF	0	0	0	0	0	0	0	0	0	0	0
XLS	0	0	0	0	0	0	0	0	0	0	1

8.8. ábra. Bitmap index

8.4.2. Az indexelés gyakorlati kérdései

Mivel a dokumentumokban sok különböző szó szerepel, az indexállomány mérete tekintélyes nagyságúra nőhet. Egyes felmérések szerint a mai angol nyelvben használt szavak száma 700 000–1 400 000 között mozog. Egy ember szókincse viszont ennél jóval kisebb: egy átlagosan művelt ember 20 000 szót használ aktívan, és a számítások szerint egy hét alatt csak körülbelül 2000 szóból építkezünk. A vizsgálatok szerint összességében a magyar nyelvben is hasonló nagyságrendű, mintegy egymillió lehet a szavak száma. Az angol nyelvtől eltérően itt kevesebb

a lemma (egyedi szótári bejegyzés), és jóval több a toldalékolt, illetve összetett szó. Mivel egy dokumentumban akár 20 000–30 000 szó is előfordulhat, a dokumentumok teljes szóállományának indexelése csak jelentős költséggel oldható meg.

E költségproblémák miatt az első szóindexelő rendszerek nem a teljes dokumentumot, hanem annak csak egy részét vonták be az indexelésbe: általában csak pár száz szót dolgoztak fel. A szavak kiválasztása pozíciójuk alapján történt, többnyire a dokumentumok elejét indexelték. A *Webcrawler*-rendszer [145] egyik újítása az volt, hogy a teljes dokumentumot indexelte, amihez természetesen nagyobb erőforrásigény is társult.

A szöveges dokumentumok indexelése mellett már más, pl. a képformátumok nyilvántartását is megvalósítja az 1998-ban indult, de ma már világelső keresőrendszer, a Google. A nemszöveges dokumentumok esetében is a szöveges kulcsértékeket használnak, amelyeket a rendszer a képek előfordulási környezetéből állít elő. A Google a feltárt dokumentumok nagy részét lokálisan is megőrzi, és így a dokumentum abban az esetben is elérhető, ha az eredeti tárolóhely szervere éppen nem üzemel, vagy az eredeti dokumentum már nem is létezik a keresés időpontjában.

A keresések találati listája rendszerint egy igen hosszú dokumentumlista. A találatoknak olykor jelentős része nem releváns a vizsgált problémakörben, de ezt a felhasználó sokszor csak a dokumentum tartalma alapján tudja eldönteni. A felhasználó részéről természetes igényként merül fel, hogy nagyobb segítséget kapjon a keresőrendszertől a válaszhalmaz áttekintésére. Erre a problémára kínál megoldást a *Vivísimo* keresőszolgáltatás[5] ClusterEngine nevű [201] *csoportosító motor*ja. A szolgáltatás a találati lista dokumentumait tartalmuk alapján csoportokba rendezi. A csoportba rendezést klaszterezési algoritmussal oldja meg (ld. 6. fejezet). A keresés eredményeként dokumentumcsoportokat kapunk, amelyek a csoport legjellemzőbb kulcsszavaival, ill. témaköreival vannak felcímkézve.

A keresőrendszerek napjainkban egyre több funkcióval vannak ellátva. A funkcióbővítés miatt rendszerint a kezelőfelület is összetettebbé válik, amelyet már egyre nehezebben látnak át a felhasználók. Felhasználóbarát megoldást jelentene, ha egyáltalán nem kellene újat tanulni a sajátos kezelőfelület megértéséhez, és természetes nyelvet használhatnánk a parancsok kiadására. Számos kutatás indult az elképzelés nyomán a *természetes nyelvű kezelőfelület* kialakítására (natural language interface, NLI). Az NLI első eredményei az Altavistához köthetők. A

[5] www.vivisimo.com

rendszer lehetőséget kínált arra, hogy a felhasználó angol nyelvű mondatokban tegye fel a kérdéseit, például:

What is the distance between London and Paris?

A keresőmotor megpróbálta a keresőmondat alapján azokat a dokumentumokat kiválogatni, amelyekben a mondat jelentéséhez, lényegéhez kötődő szavak előfordulnak.

A technológia egy másik neves képviselője az 1999-ben megjelent *AskJeeves*[6] keresőrendszer, amely a választ is természetes mondat formájában adja meg. Így a lekérdezés eredménye nem egy dokumentumlista, hanem egy, a lényeget összefoglaló mondat. A fenti kérdésre az AskJeeves az alábbi választ ajda:

The distance between London, England and Paris, France is 341 km

A rendszer igen hatékonyan működik, de széleskörű elterjedését gátolja, hogy a rendszer saját adatbázisából veszi a válasz előállításához szükséges információkat, s nem az elérhető weblapokat dolgozza fel. Ez azért probléma, mert ezt az adatbázist manuálisan, szakemberek alkalmazásával töltik fel. Ez viszont azt jelenti, hogy viszonylag szűk az feldolgozott témák köre. Ám a feldolgozott tématerületek egyre bővülnek, és ma már pontos választ kaphatunk egy olyan kérdésre is, mint

When was the steam engine invented?

de a mélyebb elemzést igénylő kérdéseknél, mint például

Who was the winner of Marathon running in Atlanta?

egyelőre még nem számíthatunk helyes válaszra.

8.4.3. Alkalmazott indexelési technikák

Ahogy azt láthattuk, a keresőmotorok indexelési feladatainak megoldásához több különböző módszerre is szükség lehet. Az alábbiakban összefoglaljuk a legfontosabb indexelési technikákat.

8.4.3.1. Szó és kifejezés alapú indexelés

Ilyenkor a jelentést és a nyelvi elemeket figyelmen kívül hagyják, pusztán az egymástól izolált szavak halmazában kereshetünk. A főbb keresési lehetőségek:

- pozíció alapján történő szűrés;
- teljes szóhalmaz;

[6] `www.ask.com`

- relevancia alapú szűrés.

8.4.3.2. Tartalom és hivatkozás alapú indexelés

A szó alapú keresés egyik hátránya, hogy a tartalmazott szavak és a témakört kifejező szavak rendszerint nem esnek egybe. Mivel a felhasználó nem tudhatja előre, hogy milyen szavak fordulnak elő a találatban, a kulcsszó alapú keresés igen pontatlan felidézési arányt tud csak biztosítani. Ezen segít a témaszavas keresés, ahol a felhasználónak elegendő a témakört bejelölni, nem keresőkifejezéssel kell meghatároznia keresési igényét.

Ennek előfeltétele az, hogy a rendszer társítani tudja az egyes kulcsszavakat a témakörökhöz. A témaszótár kialakítása viszont igen nehéz és költséges feladat, amit rendszerint manuális módon hajtanak végre. A probléma egy igen ötletes és hatékony megoldása a dokumentumra való hivatkozási szövegek, ún. *horgonyok* (anchor text) felhasználása. Ennél a módszernél a hivatkozó szöveghez a hivatkozott dokumentumot indexelik, mégpedig nagy súlyértékkel. A módszer azon a meggondoláson alapszik, hogy a hivatkozó szövegeket emberek készítik, s abban röviden összefoglalják a hivatkozott dokumentum lényegét. Így a hivatkozó szöveg egy tömör leírásnak is tekinthető, amely alkalmas a téma szerinti keresés megvalósítására is. A módszer egyik első alkalmazója a *Word Wide Web Worm* keresőrendszer volt, de a *Google* keresője ezt alkalmazza.

8.4.3.3. Szöveg és metaadat alapú indexelés

A normál indexelési mechanizmusban a dokumentumot szavak halmazaként tekintjük, ezt *szózsákmodellnek* nevezzük (ld. a 33. oldalt). Ekkor a dokumentum szavainak csak a gyakoriságát rögzítjük, a pozícióját nem, majd ez alapján elvégezzük az indexelést. Ezzel a módszerrel elveszítjük a dokumentumban a szavak eredeti környezetére és pozíciójára vonatkozó információkat. Ezek teljes vagy részleges megőrzésére számos módosítást alkalmaznak, bővítve az indexben tárolt információkat. A szó előfordulási környezetét leíró metaadatokból kiemelhetők az alábbi, gyakrabban használt elemek:

- *Közelítő pozíció megőrzése*: a dokumentumot belső struktúrával látják el, az egyes részek kiemelt szerepet töltenek be a dokumentum témakörének meghatározásában. Így például a dokumentumok eleje hasznosabb a témakör meghatározásánál, mint a záró vagy a középső rész (ld. még a 7. fejezetet).
- *Dokumentumszekció-pozíció megőrzése*: egyes dokumentumtípusok — mint például a HTML-dokumentumok — jól definiált és könnyen feltárható belső

struktúrával rendelkeznek. A belső strukturális elemek között vannak olyanok, amelyek a dokumentum lényegi, összefoglaló adatait, míg más részek a kifejtő leírást tartalmazzák. Így például a HTML-dokumentumokban a címsor (title) vagy leírás (meta) részek fontosabbak a többi szövegrésznél.

- *Formátuminformációk*: A szöveges dokumentumokban a szerzők vizuális eszközöket is alkalmaznak a lényegkiemeléshez, pl. félkövér szedés, a dőlt betű, nagyobb betűméret alkalmazása. A hatékony keresőrendszerek alkalmasak e formai elemek feltárására is, így nagyobb súlyt tudnak rendelni a kiemelt szövegrészekhez. Így dolgozik többek között a Google keresőrendszer is, amely mind a szövegrész betűtípusát, mind a dokumentumban való előfordulás helyét nyilvántartja a belső indexében.

8.4.3.4. Tartalom és struktúra alapú indexek

A hipertext-dokumentumokat kezelő keresőmotorokban a dokumentumok szövegtartalma mellett igen fontos információ a dokumentumok közötti kapcsolatok kezelése is. A kapcsolati információk gyors elérésére egy külön indexet hoznak létre, melyet kapcsolatleíró vagy *linkindexnek* neveznek. A kapcsolati információ felhasználható többek között az egy adott lapra mutató dokumentumok keresésénél.

8.4.3.5. Indexelt dokumentumok köre

Mivel a felhasználók a szöveges dokumentumok mellett egyre növekvő mértékben képi és hangállományokban tárolják az információkat, szükség van ezeknek az új médiumoknak az indexelésére is. Egyes célrendszerek, mint például az *Oracle Media Server*, a tartalom közvetlen kinyerésére is kísérletet tesznek, az általános webes keresőrendszerek azonban csak szöveges metaadatokat használnak. Az indexelt dokumentumtípusok köre:

- szöveg,
- kép,
- hang (MP3).

Itt ismét utalunk arra, hogy a *mélyhálón* [20, 188] található tartalmak a hagyományos keresőmotorokkal nem indexelhetőek. A mélyhálón található dokumentumok lehetnek dinamikusan előálló, védett, kódolt (nem szabványos tárolási formátum) vagy esetleg új, izolált, még nem indexelt dokumentumok.

A mélyhálón lévő dokumentumok száma a felmérések szerint nagyságrendekkel nagyobb a látható, ún. *felszíni hálón* elérhető dokumentumokénál. A mélyháló jelentőségét mutatják az alábbi, 2001-es statisztikai adatok is [20]:

- a mélyháló tartalma néhány százszor nagyobb, mint a felszíni hálóé;
- a mélyháló kb. 500 milliárd dokumentumot foglalhat magába;
- a mélyháló dokumentumai konkrétabb, mélyebb és pontosabb információtartalommal bírnak;
- a mélyháló dokumentumait összességében másfélszer több alkalommal keresték fel, mint a felszíni háló dokumentumait.

A láthatatlan web láthatóvá tétele az ott tárolt információk fontossága ellenére sem megoldott feladat. A védelem és a kódolás miatt rejtett dokumentumok szándékosan rejtettek, ezért nem lehet őket elérhetővé tenni. A technológia fejlődésével viszont a dinamikus dokumentumok bevonása és az új dokumentumok gyorsabb észlelése megvalósítható feladatnak tűnik — ezt bővebben a 9.2. szakaszban tárgyalja könyvünk.

8.5. A Google áttekintése

Napjaink legelterjedtebb keresőrendszere a Google, amelynek adatbázisa becslések szerint jelenleg mintegy 20–25 milliárd indexelt dokumentumot tartalmaz. E hatalmas adattömeg hatékony kezelése igen bonyolult feladat, melyre a Google szakemberei számos egyedi megoldást dolgoztak ki.

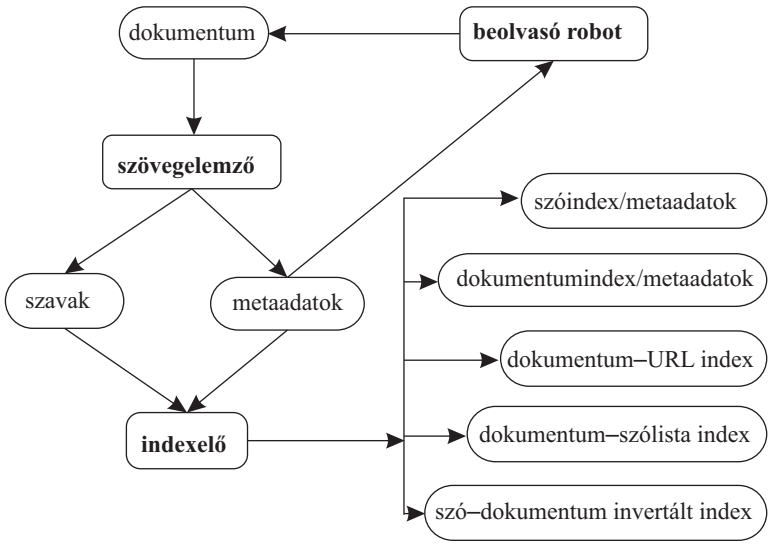
A Google komplex hardver-szoftver rendszerét Googleplex-nek [10] hívják. A Googleplex egyik sajátossága, hogy drága céleszközök helyett olcsó, hagyományos hardverre, PC-kre támaszkodik. A rendszer több tízezer (talán már több százezer) olcsó gépet tartalmaz. Felépítése hierarchikus, minden szinten hasonló struktúrával. Az átlagos PC-k használata olcsó hardvert és a gyártóktól való függetlenséget biztosít, viszont a gyenge pontjuk a megbízhatóság. A hardverhibák okozta kiesések áthidalására a rendszer egy többszörösen redundáns megoldást alkalmaz. Minden állományt 3–6 példányban tárolnak különböző helyeken. Ezáltal az egyik előfordulás kiesése gyorsan pótolható. A felügyelő szoftver automatikusan kezeli a hibát, és átirányítja az elérési útvonalakat. A felügyeletre a Linuxnak egy speciális, testre szabott változatát dolgozták ki. A hardverhibák egyszerű javítása érdekében a rendszer a számítógépeket speciális kialakítású, moduláris felépítésű rekeszekben tárolja fizikailag, így a meghibásodott egységek igen egyszerűen kicserélhetők. A belépő új komponenst a vezérlő program automatikusan felveszi a klaszterbe, és bevonja az adatkezelés feladatába.

A Google sikerének kulcsa a testreszabott hardverstruktúra mellett a speciális kereső és rangsoroló algoritmus. A kidolgozott dokumentumértékelő mechanizmus lehetővé teszi a lapok felhasználói szempontokhoz közel álló rangsorolását, elfogadható számítási költség mellett.

8.5.1. A Google indexelési mechanizmusa

A *Google* a dokumentumok indexelésére [30] több, logikailag különböző funkciót betöltő indexállományt hoz létre (ld. 8.9. ábra):

- dokumentumok, és azok metaadatainak indexelése;
- URL és dokumentum összerendelési index;
- szavak és metaadataik indexelése;
- szóelőfordulások indexelése.



8.9. ábra. A Google indexelési mechanizmusa

A dokumentumindex az ISAM-technológián alapszik. Ez minden dokumentumhoz hozzárendeli az elérési adatokat, a dokumentum hosszát, a dátumadatokat és az egyéb kiegészítő információkat. A második, URL-index is hasonló szerkezetű, ezzel lehet az URL alapján gyorsan kikeresni a belső DocID- (dokumentum-azonosító) értéket. A szóindex is normál indexszerkezetet használ. Segítségével

gyorsan elérhető egy kulcsszó azonosítója (WordID) és a hozzá kapcsolódó metaadatok. A hatékony keresés alapja a szóelőfordulás-index. A Google ehhez két indexet is implementál. Az egyik a dokumentum-szó kapcsolatot tároló *forward index*, a másik a szó-dokumentum kapcsolatot tároló *invertált index*. A *szóelőfordulások* (hits) tárolásánál kétféle típust különböztetnek meg:

- normál (plain hit);
- kiemelt (fancy hit).

A kiemelt előfordulás esetén a szó a dokumentum fontosabb szekcióiban, például a címben vagy a hivatkozó szövegben szerepel. A forward index esetén egy DocID-hez a WordID-k listája kapcsolódik, mely a kód mellett a kapcsolódó szóelőfordulás-leírásokat is tartalmazza. Az invertált index esetében a WordID-hez a következő információk társulnak:

- a tartalmazó dokumentumok száma;
- a tartalmazó dokumentumok listája;
- az előfordulási pozíciók listája.

A keresőrendszer előbb a forward indexet létre hozza, majd aktualizálja az invertált indexet.

8.5.2. PageRank-módszer

A keresés eredménye a találati lista. A felhasználók többsége csak az első találati oldalt nézi át, ezért fontos, hogy ott a legrelevánsabb dokumentumok szerepeljenek. A követelmény teljesítésére egy rangsoroló algoritmust kell alkalmazni. A legfontosabb rangsoroló algoritmusok:

- kulcsszavak gyakorisága;
- hivatkozások (hiperlink) fontossága (PageRank);
- felkeresési gyakoriság (hits).

A Google által kifejlesztett *PageRank* célja, hogy egy objektív mérőszámmal fejezze ki az emberek szubjektív véleményét egy dokumentum hasznosságáról. Kiindulási pontja az akadémiai körökben ismert hivatkozásindex, amely szerint egy tudományos cikk annál fontosabb, minél többen hivatkoznak rá. A PageRank továbbfejleszti a módszert azáltal, hogy nem egyforma súllyal veszi figyelembe a dokumentumra mutató hivatkozásokat, és normalizálja a hivatkozásindexet a hivatkozások száma alapján.

Kiinduláskor a j-edik dokumentum PageRank-értékét, p_j-t az alábbi képlet adja meg:

$$p_j = \alpha \left(\frac{p_1}{c_1} + \cdots + \frac{p_N}{c_N} \right), \tag{8.2}$$

ahol az összegzés az összes dokumentumra vonatkozik. A c_i az i-edik dokumentumból kiinduló hivatkozások számát jelöli. A képlet azt mutatja, hogy egy lap fontossága a rá hivatkozó lapok fontosságától függ. A saját fontosságértékét minden lap szétosztja a hivatkozott lapok között. Az α érték egy alkalmas normalizációs faktort jelöl.

A fenti egyszerűsített PageRank-értéket megadó kifejezés átalakítható a következő mátrixformára:

$$\mathbf{p} = \alpha \cdot \mathbf{C} \cdot \mathbf{p}, \tag{8.3}$$

ahol $\mathbf{p}$ a p_i értékek vektora. A (8.3) képletben a $\mathbf{C}$ szimbólum a c_i értékekből képzett mátrix, melyre teljesül, hogy

$$C_{i,j} = \begin{cases} \frac{1}{c_i}, & \text{ha van hivatkozás az } i\text{-edik és } j\text{-edik dokumentum között,} \\ 0, & \text{különben.} \end{cases}$$

A fenti egyenletből látható, hogy a $\mathbf{p}$ fontossági értékek a $\mathbf{C}$ mátrix sajátvektorai lesznek, és az α a mátrix sajátértékét jelöli. A felhasznált formula azonban hibás értéket ad bizonyos esetekre, többek között *nyelő*k (rank sink) esetére. Ez akkor fordul elő, ha egy dokumentumhalmaznak csak bemenő kapcsolatai vannak, de kifelé nem hivatkozik. Ezen anomáliák elkerülésére *fontossági forrás*okat (source of rank) vezettek be. Minden dokumentumhoz tartozik egy ilyen forrásérték, melyek együttesét az $\mathbf{e} = \langle e_1, \ldots, e_N \rangle$ vektorral jelöljük. Ekkor a PageRank értékét megadó kifejezés a következő formában írható fel:

$$p_j = \alpha \cdot e_j + \alpha \cdot \sum_{i=1}^{N} \frac{p_i}{c_i}, \tag{8.4}$$

ahol az α ismét a megfelelően választott faktort jelöli.

A fenti képlet jól szemlélteti a felhasználóknak a web böngészése során tanúsított viselkedését is, az ún. *véletlen szörföző modell*t. Eszerint a felhasználó egy véletlenül kiválasztott lapból kiindulva járja be a világhálót. Egy adott lapnál követheti a lapon lévő hiperlinkek valamelyikét, vagy egy teljesen új oldalt választ véletlenszerűen. Egy oldal p fontossági értéke megfelel az oldal felkeresési valószínűségének. A (8.4) képletben szereplő e_i értékek annak a valószínűségét

mutatják, hogy a felhasználó egy véletlen új lap kiválasztásával éppen az i-edik oldalra kerül.

Mivel a (8.4) képlet átírható az alábbi mátrixalakra:

$$\mathbf{p} = \alpha \cdot (\mathbf{C} \cdot \mathbf{p} + \mathbf{e}), \tag{8.5}$$

a $\mathbf{p}$ vektorra vonatkozó összefüggést tovább alakíthatjuk az alábbi formára:

$$\mathbf{p} = \alpha \cdot (\mathbf{C} + \mathbf{e} \cdot \mathbf{1}) \cdot \mathbf{p}, \tag{8.6}$$

ahol $\mathbf{1} \in \mathbb{R}^{1 \times N}$ csak egyesekből álló sorvektor. Az átalakításnál felhasználtuk, hogy a $\mathbf{p}$ vektor elemei valószínűségi eloszlást alkotnak, tehát összegük 1. Tekintsük ugyanis a $\mathbf{D} = \mathbf{e} \cdot \mathbf{1}$ mátrixot, amelyre $D_{i,j} = e_i$ teljesül. Ekkor az $\mathbf{e} \cdot \mathbf{1} \cdot \mathbf{p} = \mathbf{D} \cdot \mathbf{p}$ mátrix i-edik elemére fennáll, hogy

$$(\mathbf{D} \cdot \mathbf{p})_i = \sum D_{i,j} \cdot p_j = \sum e_i \cdot p_j = e_i \cdot \sum p_j = e_i,$$

azaz $\mathbf{e} \cdot \mathbf{1} \cdot \mathbf{p} = \mathbf{e}$.

A mátrixalak alapján látható, hogy a $\mathbf{p}$ PageRank-vektor a

$$(\mathbf{C} + \mathbf{e} \cdot \mathbf{1})$$

mátrix sajátvektora lesz (ld. a 60. oldalt). Tehát a lapok fontosságértéke visszavezethető a kapcsolati mátrix sajátérték-sajátvektor problémájára. Az $\mathbf{e}$ elemeire a legelterjedtebb módszer egy egyenletes elosztást határoz meg a következő értékkel:

$$e_i = \frac{0{,}15}{n}.$$

Az eredmény lehetőséget ad a $\mathbf{p}$ vektor pontos meghatározására, hiszen a sajátérték és a sajátvektor kiszámolására már számos módszer ismert. Ezek a módszerek azonban polinomiális költségűek, ami a valós kapcsolati mátrixok esetén túl nagy időköltséget jelent, ezért a keresőrendszerekben a fenti sajátvektor-probléma pontos megoldása nem kivitelezhető.

Gyakorlati szempontból viszont elegendő a feladat közelítő megoldásának meghatározása is. Ehhez a $\mathbf{p}$ vektort egy iterációs algoritmussal határozzák meg a

$$\mathbf{p}_{i+1} = \alpha \cdot (\mathbf{C} + \mathbf{e} \times \mathbf{1}) \cdot \mathbf{p}_i$$

formula segítségével. Az iteráció leállási feltétele:

$$\|\mathbf{p}_{i+1} - \mathbf{p}_i\| < \varepsilon.$$

Az iterációs megoldásnál a költségek bizonytalanok. Ha ugyanis a sorozat divergens, vagy túl sok lépés kell a megfelelő eredmény eléréséhez, akkor nem kifizetődő ezt a módszert választani. Szerencsére a tapasztalat azt mutatja, hogy igen nagy kapcsolati mátrix esetén is meglepően gyors a konvergencia. A Google által nyilvánosságra hozott adatok szerint 322 millió kapcsolatot tartalmazó kapcsolati mátrix esetén is elegendő volt 52 iterációs lépés a leállási feltétel eléréséhez. Egy másik vizsgálat szerint egy 26 millió dokumentumot tartalmazó dokumentumgyűjteményre egy közepes teljesítményű munkaállomáson a számítás csupán néhány órát vesz igénybe [30]. Az elméleti vizsgálatok szerint ez a jó tulajdonság a valós kapcsolatok jellegéből fakad: a lapok hivatkozásai elég heterogének és a lapok kis töredéke rendelkezik csak jelentős hivatkozási értékkel [139].

8.6. A keresési technikák áttekintése

A felhasználók felé a keresőrendszerek a keresési felületükön keresztül jelennek meg. A keresőmodul szolgáltatásai alapvetően befolyásolják a rendszer hasznosságát. A szokásos keresési felületeknél a felhasználó megadhat egy vagy több kulcsszót, és a rendszer válaszként visszaadja a kulcsszót tartalmazó dokumentumok halmazát. A háttérben megvalósított indexelési technikától függően az alap keresési módszer további szolgáltatásokkal bővíthető. A mai rendszerek az alábbi technikákat alkalmazzák:

- *Taxonómia alapú keresés*:
  A rendszer a nyilvántartott dokumentumokat kategóriákhoz rendeli. A kategorizálást rendszerint manuálisan végzik. Szakértők hozzák létre a taxonómiát (a kategóriák hierarchikus rendszerét), és rendelik a kategóriákat a dokumentumokhoz. A keresés ilyenkor a főkategóriák szintjén kezdődik, majd a taxonómiában lefelé navigálva egyre szűkítheti a keresésbe bevont dokumentumok körét.

- *Szó alapú keresés*:
  A kulcsszavak szintjén történő keresés. A felhasználó egy vagy több kulcsszót ad meg, és a rendszer megkeresi azon dokumentumokat, amelyekben ezek a szavak pontosan előfordulnak.

- *Kifejezés alapú keresés*:
  A felhasználó szavakból felépített kifejezéseket is megadhat, és az illeszkedő dokumentumnak a megadott sorrendben kell tartalmaznia a kifejezés szavait. Az ilyen keresések megvalósításához a szavak előfordulási pozícióit is tárolnia kell a rendszernek.

- *Összetett feltétel alkalmazása*:
  Több elemi feltételből logikai operátorokkal (and, or, ...) összeállított keresési feltétel alkalmazása. A szó alapú keresés továbbfejlesztésének tekinthető az összetett keresési feltételek megvalósítása. Ezekben a rendszerekben a felhasználó szavanként pontosíthatja a keresés paramétereit. Egy szó ugyanis lehet *kötelező*, *opcionális* vagy *tiltott*.
- *Finomítható keresés*:
  Az egyszerű kulcsszavas keresésnél az illeszkedő dokumentumok száma nehezen becsülhető. Sok esetben a találatok száma olyan hatalmas, hogy további szűkítésre van igény. Ekkor a találati halmazon végzett kereséssel szűkíthető az eredményhalmaz.
- *Kiterjesztett keresés*:
  Már a klasszikus szövegkeresési módszereknél is bevált hatékonyságnövelő lépés volt a keresőkifejezés automatikus kibővítése. A módszer első fázisában a megadott keresési mintára illeszkedő dokumentumokból viszonylag kis számú elem kerül kiválasztásra. Ezt követően a kiválasztott dokumentumokból egy nagyobb számú releváns szót emel ki az algoritmus. E szavak relevanciaértékeit a Rocchio-módszerrel [91] számítják ki. Az így kibővített keresőkifejezéssel végzett keresés eredményét tartalmazza a felhasználó számára megjelenített találati lista.
- *Statisztika alapú kiterjesztett lekérdezés*:
  Ezt a megoldást olyan rendszerekre javasolták [19], amelyeknél rendelkezésre áll a lekérdezések és az elfogadott válaszok mintahalmaza. A mintahalmaz alapján a rendszer korreláció jellegű kapcsolatot állít fel a keresőkifejezés szavai és a válaszdokumentum szavai között. Ezáltal a rendszer megtanulja tipikus kérdés-válasz kapcsolatot. A lekérdezés második, kibővített fázisában nem az elsőként kapott releváns dokumentumokat veszi alapul, hanem a letárolt mintahalmazt. A szópárok kapcsolatának mérése a *kölcsönös információtartalmon* alapul (ld. 57. oldal).
- *Témaorientált keresés*:
  Ekkor a keresés nem kulcsszó alapján történik, hanem azt vizsgálja a keresőrendszer, hogy a dokumentum tartalmaz-e a kulcsszóhoz mint témakörhöz kapcsolódó szavakat. Hasonló a taxonómia alapú kereséshez, de nem a navigáción alapul.
- *Klaszter alapú keresés*:
  E módszer jellegzetessége, hogy a találatokat csoportokba rendezve adja meg.

Az eredményként kapott csoportokat szükség esetén további csoportokra lehet bontani.

- *Szótő alapú keresés*:
  A speciális, megadott nyelv nyelvtanával kibővített keresőrendszerekben lehetőség van a szótő alapú keresésre is. Ebben az esetben a rendszer a keresőkifejezést és a dokumentumok szavait is szótövekre konvertálja, és a keresést a szótövek halmazán hajtja végre.
- *Szekció szerinti szűkítés*:
  A keresési feltétel megadásakor nemcsak a keresett szót adhatjuk meg, hanem azt is, hogy a dokumentum mely részében (szekciójában) kell a keresett szónak előfordulnia.
- *Metaadatok szerinti szűkítés*:
  Egy dokumentum szavaihoz a keletkezési és megjelenítési körülményeket leíró kiegészítő információk társulhatnak. Az ilyen metaadatokra a keresőkifejezésekben is hivatkozhatunk, így például a szűrési feltételben szerepelhet a dokumentum utolsó módosításának dátuma, a dokumentum nyelve, formátuma és doménje.
- *Természetes nyelvi keresőkifejezések*:
  A keresésőkifejezést nem kulcsszavakkal, hanem természetes nyelven megfogalmazott mondatot adjuk meg. A rendszer a beépített nyelvi elemzőn keresztül átkonvertálja a kérdést a kulcsszavak, témakörök szintjére. Egyes rendszerekben a válasz meghatározása is természetes nyelven történik. A legfontosabb természetes nyelvi lekérdező felületetet megvalósító rendszerek az AskJeeves, Northern Light és az Altavista (ld. még a 9.1. szakaszt).
- *Beszéd alapú parancsbevitel*:
  A kutatások egyik iránya a beszéd alapú parancsbevitel megvalósítása. Ennek során a megoldandó probléma a hangalakok és szavak összerendelése. A [69] munkában bemutatott módszer egy működő webes keresőmotor lekérdezési naplója alapján tanítja be a hangalakok és szavak kapcsolatát. A feldolgozás főbb lépései:
  - a keresőkifejezések kigyűjtése napló alapján;
  - a nem angol keresőkifejezések kérések elvetése;
  - a nyelvtan ellenőrzése és korrekció végrehajtása;
  - a gyakori szavak és szókapcsolatok meghatározása;
  - a szóalakok fonetikai alakra történő konvertálása;
  - a hangalakszótár felépítése;

  - a lekérdezési hangmintára közelítő keresés végrehajtása;
  - a jelöltek kiértékelése.

- *Szemantikus háló alapú dokumentumkiértékelés*:
  A kutatási célok között szerepel egy olyan intelligens keresőrendszer kialakítása, amely az emberi gondolkodás képességeit szimulálva képes a feltett kérdésből az érdeklődési témakör felismerésére, a témakörhöz tartozó dokumentumok kiválasztására, valamint a dokumentumok alapján a kérdésre tartalmilag megfelelő válasz megadására. A rendszer egyik előfutára az IBM által indított *WebFountain*-projekt [138]. Ennek keretében nemcsak a dokumentumok tartalmi megértését célozzák meg, hanem intelligens párbeszéd kialakítására, szabályfeltárási funkciók megvalósítására is törekednek. Az IBM egyik stratégiai terve, hogy létrehozzon egy óriási kapacitású szerverrendszert, amely az összes kapcsolódó vállalat információfeldolgozási igényét ki tudja elégíteni. A rendszer adatbányászati módszereket alkalmazva fogja felderíteni a dokumentumok tartalmából kinyerhető szabályszerűségeket.

8.7. A piaci keresőrendszerek működésének áttekintése

A weben elérhető nyilvános keresőrendszerek egyszerű elemzésére egy tesztkérdést tettünk fel a vizsgált rendszereknek. Az összehasonlítás során elsősorban a szolgáltatás által nyújtott keresési funkciók körét és a kapott válaszok minőségét vizsgáltuk. Azt tapasztaltuk, hogy a különböző keresőrendszerek igen jelentős eltérést mutattak a kapott válaszok terén. Mintakérdésként a

'University of Miskolc' Databases

keresőkifejezést használtuk fel, melyben tehát egy kifejezés és egy kulcsszó szerepelt. Meglepő módon, még a könyv írásakor, 2007-ben is több olyan keresőrendszer működött, amelyekben a kifejezés alapú keresés lehetősége nem áll rendelkezésre. Továbbá olyan keresőmotorokat is találtunk, melyek nem adtak találatok a fenti kérdésre. A legnagyobb releváns válaszhalmaz ezzel szemben mintegy 700 dokumentumból állt. A tesztek 2006-ban és 2007-ben is elvégeztük. A két időpont között 10 hónap volt. Az eredmények összevetéséből az derül ki, hogy a találatok száma mintegy 40–50 %-kal nőtt. A funkciók tekintetében néhány apróbb változás tapasztalható, legjellemzőbb változás, hogy kibővültek a témaspecifikus lekérdezési felületek. A teszt eredményeit a keresőmotorok jellege alapján vett csoportosításban mutatjuk be.

8.7.1. Taxonómia alapú keresők

A keresőrendszer egy taxonómiát biztosít, melyben fentről lefelé haladva szűkíthető a témakör. A taxonómia alapú keresés mellett a kulcsszavas keresés is támogatott. E rendszerek csak limitált dokumentumkört dolgoznak fel.

- *Best of the web* (`www.botw.org`)
  A taxonómia legfelső szintje 15 témakört tartalmaz. A találatok keresése során a 'Science' > 'Educational Resources' útvonalon haladtunk, de nem lehetett magyarországi intézményekre továbblépni. A 'Regional' > 'Europe' > 'Hungary' > 'Education' útvonal üres listához vezet. Mivel a rendszer nem keres kifejezésekre, csak kulcsszavas keresést lehet használni. Ebben az esetben mintegy 12000 (2007), illetve 8000 (2006) nem releváns találatot kaptunk. A keresőrendszer ASP technológiára épül. A felhasználók saját lapjaikat 200$-ért vetethetik fel a katalógusba.

- *Skaffe* (`www.skaffe.com`)
  A taxonómia legfelső szintje 22 témakört tartalmaz. A keresésnél az 'Education and Research' > 'Colleges and Universities' útvonalon haladtunk, de nem lehet magyarországi intézményeket elérni. A 'Regional' > 'Europe' > 'Hungary' > 'Education' útvonal olyan listához vezet, melyben három magyar intézmény szerepel csak (hallatlan.hu, sulinet.hu, angol.hu). A kereső csak a kulcsszavas keresést támogatja, kifejezésre nem lehet keresni. A rendszer 460 (2007), illetve 825 (2006) nem releváns találatot adott. A fejlesztés PHP alapokon nyugszik. A felhasználók ingyen kérhetik lapjaik felvételét a katalógusba.

- WOW (`www.wowdirectory.com`)
  A taxonómia legfelső szintje 28 témakört tartalmaz. A keresésnél az 'Education' > 'Local and regional web-site directory' > 'Hungary' útvonalon haladtunk, de itt csak egy üres 'Budapest' témakör található. Mivel a rendszer nem keres kifejezésekre, csak a kulcsszavas keresés használható, amelynek eredménye 95 (2007), illetve 272 (2006) nem releváns találat. Ezt a rendszert is PHP-ban fejlesztették ki.

8.7.2. Általános keresők

- *Google* (`www.google.com`)
  A szolgáltatás alapértelmezésben a szöveges dokumentumokban, valamint képek metaadataiban való keresést támogatja. A kifejezéskeresés mellett szá-

mos további szolgáltatás és speciális szűrési funkció közül lehet választani. A főlapon elérhető funkciók:

- logikai operátorok (and, or, not);[7]
- kifejezések keresése;
- egy laphoz hasonló lapok keresése (similar page funkció);
- egy lapra mutató lapok keresése (link: <URL>);
- könyvek keresése (books <search text>);
- definíciók keresése (define: <search text>);
- USA-beli repülőtéri információk lekérdezése (kód airport);
- filmekkel kapcsolatos információk keresése (movie: <search text>);
- USA-beli városok térképei (irányítószám város utca);
- USA-beli telefonkönyv-keresés
- aktuális tőzsdei adatok (tőzsdenév);
- időjárási információk keresése (weather <search text>);
- automatikus fordítás más nyelvre (A támogatott nyelvek köre az angol, francia, német, olasz, spanyol, portugál, kínai, japán, koreai nyelvekre terjed ki. Elérése a translate this page funkcióval történik. Ez a funkció csak akkor aktív, ha a felhasználó nyelve megegyezik valamelyik támogatott nyelvvel);
- a dokumentumhoz rendelt országdomén (kulcsszó site:domén).

A speciális keresések (advanced search) lapon elérhető funkciók:

- dokumentum nyelve;
- fájlformátum;
- dátum szerinti szűrés;
- domén alapú szűrés;
- dokumentumon belüli tartomány szerinti szűrés;
- hasonló lapok keresése;
- hivatkozó lapok keresése.

A keresés eredménye: 728 (2007), illetve 541 (2006) találat, melyek mindegyike relevánsnak tekinthető. A kulcsszavas keresés 24 000 találatot adott. A Google további speciális lehetőségei közé tartozik a levelezői csoportok kezelése és a témakör, taxonómia alapú keresést biztosító webcímtár funkció. Ebben a kategória listában a 'magyar' > 'tudomány' > 'oktatás' útvonalon elérhető a Miskolci Egyetem honlapja.

[7] Az összes keresőmotor ezt a három operátort támogatja, ezért a továbbiakban nem írjuk ki őket.

- *Altavista* (`www.altavista.com`)
  A kereső alapértelmezésben három dokumentumtípust támogat, a szöveges és képi formátum (metaadatok) mellett hangállományok közt is lehet keresni. A keresésnél itt az alábbi speciális szűrési lehetőségek (advanced search) közül választhatunk:
  - dokumentum országdoménje,
  - fájlformátum,
  - dátum szerinti szűrés,
  - kifejezések keresése.

  A keresés eredménye 316 (2007), illetve 297 (2006) találat, alapvetően releváns adatokkal.
- MSN (`search.msn.com`)
  A kereső alapértelmezésben három dokumentumtípust támogat, a szöveges és képi formátum (metaadatok) mellett hangállományok között is lehet keresni. A keresésnél az alábbi opcionális szűrési lehetőségek (speciális feliratú funkció) közül választhatunk:
  - összetett kifejezések keresése,
  - dokumentum országdoménje,
  - egy lapra mutató lapok keresése,
  - nyelv szerinti szűrés.

  A keresőrendszer egyedi szolgáltatásai közé tartozik az RSS-lapokra történő szűrés[8] és a testre szabás (egyedi lap struktúra) funkciója. A keresés eredménye 298 (2007), illetve 270 (2006) találat, alapvetően releváns adatokkal.
- *Yahoo!* (`www.yahoo.com`)
  A keresésnél az alábbi speciális szűrési (advanced search) lehetőségek adottak:
  - összetett kifejezések keresése,
  - dátum szerinti szűrés,
  - dokumentum országdoménje,
  - területdomén használata,
  - fájltípus szerinti szűrés,

[8] Az RSS a webes együttműködést szolgáló üzenetcsere formátumát leíró szabvány. Az RSS-üzenetben a hírszolgáltatók megadhatják a hír rövid leírását és a tényleges forrásdokumentum elérését. A felhasználó ugyancsak RSS-formátumban előírhatja, hogy milyen jellegű tájékoztatókat kíván fogadni a hírszerverekről.

  - nyelv szerinti szűrés,
  - korhatáros lapok szűrése.

  Az egyedi elemek közé tartozik a Creative Common keresés, amely a szabadon felhasználható dokumentumo keresését támogatja. Emellett lehet témakör-specifikus kereséseket is indítani. Ide tartozik többek között az álláskeresés, kapcsolatkeresés, hírek vagy térképek keresése. E modulok a Search Service funkción keresztül érhetők el. A keresés eredménye 315 (2007), illetve 302 (2006) találat, alapvetően releváns adatokkal.

- *Ask* (`www.ask.com`)
  A keresésnél az alábbi speciális szűrési lehetőségek (advanced search) állnak rendelkezésre:
  - összetett kifejezések keresése,
  - dátum szerinti szűrés,
  - dokumentum országdoménje,
  - területdomén meghatározása,
  - nyelv szerinti szűrés,
  - dokumentumon belüli tartomány szerinti szűrés.

  Az Ask keresőmotor egyedi vonása az NLI elemeket is tartalmazó lekérdező felület. A keresés eredménye 255 (2007), illetve 150 (2006) találat, releváns adatokkal.
- *Gigablast* (`www.gigablast.com`)
  A keresésnél az alábbi speciális szűrési lehetőségeket használhatjuk: összetett kifejezések keresése, ill. egy megadott lapra mutató lapok keresése. A keresés eredménye 103 (2007), illetve 7 (2006) találat, releváns adatokkal.
- *Hotbot* (`www.hotbot.com`) és *Lycos* (`www.lycos.com`)
  Csak nyelv szerinti válogatásra van lehetőség. A keresés eredménye 1039 találat, nagyon kevés releváns dokumentummal.

8.7.3. Metakeresők

- *Mamma* (`www.mamma.com`)
  Az alábbi keresőmotorok indexállományait használhatjuk: *Ask*, *Gigablast*, *WiseNut*, *EntireWeb*. A keresési lehetőségek kimerülnek a kulcsszó, ill. kifejezés szerinti keresésben, valamint a forráskeresők kijelölésében. A keresés találati listája 30 (2007), illetve 10 (2006) dokumentumot tartalmaz.
- *Metacrawler* (`www.metacrawler.com`)
  A keresési lehetőségek között szerepel a kifejezés, a dátum, az országdomén

szerinti szűrés és a forráskeresők kijelölhetősége. A tesztünk eredménye: 67 (2007), illetve 50 (2006) dokumentum volt.

8.7.4. Mélyhálókeresők

Completeplanet (`www.completeplanet.com`)
A szolgáltatás eddig több mint 70 000 dinamikus adatbázist indexelt. A rendelkezésre álló speciális szűrési feltételek: összetett kifejezés, dátumra és a kulcsszó dokumentumon belüli elhelyezkedésére vonatkozó szűrések. A tesztünkre a találati lista üres, azaz 0 találat.

8.7.5. Keresőmotorok funkcióinak összefoglalása

A 8.1. táblázat az egyes keresőmotorok funkció szerinti összesítő összehasonlítását tartalmazza.

Keresőmotor	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
Google	I	I	I	I	I	I	I	N	I	I	N
Yahoo	I	I	I	I	I	I	I	I	N	I	N
Altavista	I	I	I	I	I	I	I	I	N	N	N
MSN	I	I	I	I	I	I	I	N	N	I	N
Ask	I	I	I	N	I	N	I	I	N	N	I
Gigablast	I	I	N	I	N	N	N	N	N	N	N
Hotbot	I	N	N	N	I	N	N	N	N	N	N
Metacrawler	I	N	I	N	N	N	N	I	N	N	N
WOW	N	N	N	N	N	N	N	N	N	N	N

8.1. táblázat. Keresőmotorok funkció szerinti összehasonlítása

A táblázatban az oszlopok az alábbi funkciókat jelölik:

- F1: kifejezés szerinti szűrés;
- F2: elemi logikai operátorok használata (and, or, not);
- F3: domén szerinti szűrés;
- F4: lapra mutató lapok keresése;
- F5: nyelv szerinti szűrés;
- F6: fájlformátum szerinti szűrés;
- F7: dokumentumon belüli pozíció szerinti szűrés;
- F8: dátum szerinti szűrés;
- F9: megadott laphoz hasonló lapok keresése;

- F10: speciális tartalom keresése;
- F11: NLI-felület.

Fontosnak tartjuk végül megjegyezni, hogy a keresőmotorok szolgáltatásai állandó fejlődésben vannak. A fenti funkcióelemzések 2006 közepén és 2007 elején készültek, pár év vagy akár pár hónap múlva valószínűleg nagyobb teret kap a tartalom, a nyelvtan alapú lekérdezés is.

9. fejezet

Válaszkereső rendszerek

Az *információ-visszakeresés*ben jellemzően olyan dokumentumokat, szövegeket keresnek vissza, amelyek a leginkább illeszkednek a bemenetként kapott dokumentumhoz.[1] A válaszkereső rendszerek (question answering systems, QAS) tekinthetőek olyan IR-rendszernek, ahol a bemenet szigorúan természetes nyelvű kérdésekből áll.[2] Persze, a pontosan feltett kérdésre általában rövid, zárt alakban megadott választ várunk, nem elégséges egy dokumentumhalmazt kijelölni, amely potenciálisan a választ is tartalmazhatja.

Éppen ezért az IR-megoldások — bár általában alkalmazhatók a válaszkeresésre is — erre önmagukban csak igen rossz hatásfokkal képesek. Ennek oka elsősorban az, hogy az igazán sikeres IR-algoritmusok a dokumentumok közötti sajátosságoknak javarészt csak a statisztikai hasonlóságait veszik alapul, míg a kérdésben hordozott konkrét tartalommal ténylegesen nem foglalkoznak. Például vizsgáljuk meg mit tenne egy átlagos IR-algoritmus *Az Egyesült Államok melyik elnöke látogatott Oroszországba?* kérdéssel! Mindenekelőtt tokenizálná, így megtalálná és szótövezésnek vetné alá feltehetően az *Egyesült Államok*, *elnöke*, *látogatott*, *Oroszországba*, illetve eltávolítaná az *Az* és a *melyik* szavakat. Ezután a kapott négy kifejezéshez illeszkedő dokumentumokat keresne. Csakhogy azok a dokumentumok, amelyekre e négy szótő illeszkedik, igen sokrétű tartalmat hordozhatnak. Ennek belátására elegendő arra gondolni, hogy a következő jelentősen eltérő értelmű kérdések mindegyikére ugyanezt az eredményt kapnánk az előfeldolgozás során — tehát az IR-algoritmusnak az ezekre adott válaszokat is szükségszerűen meg kell jelenítenie, és rangsorolnia kell a találatai között.

- *Oroszország melyik elnöke látogatott az Egyesült Államokba?*

[1] A korábbiakkal összhangban dokumentumnak tekintjük a szavak egy nem üres, véges halmazát.

[2] A *válaszkereső rendszerek*kel foglalkozó szakirodalom bemenetként megengedi a számítógép számára kiadott természetes nyelvű utasítást is (pl. *Mondd meg, hány oldal a Szövegbányászat c. könyv!*, *Mutasd a Szövegbányászat c. könyv szerzőinek egyéb műveit!*). Ebben, és csak is ebben az értelemben a hasonló utasításokat is kérdéseknek tekintjük.

- *Melyik elnök látogatott az Egyesült Államokba és Oroszországba?*
- *Mikor látogatott Oroszország elnöke az Egyesült Államokba?*
- *Az Egyesült Államok hány elnöke látogatott Oroszországba?*
- *Oroszország melyik elnöke látogatta meg az Egyesült Államok elnökét?*
- *Hányan látogatták Oroszország és az Egyesült Államok elnökeit?*

A felhasználó a megjelenített dokumentumok mennyiségét, illetve az első néhány találat felesleges tanulmányozását látva hamar lemond az IR természetes nyelvű kérdésekre való alkalmazásáról. Nem véletlen, hogy a pontos tartalmak megtalálásához a népszerű internetes keresőmotorok használatát is tanulni, gyakorolni érdemes (ld. a 8.6. szakaszt), és hogy a legtöbb felhasználó legfeljebb 3–4 szavas keresőkifejezést ad meg, szinte sosem írva le teljes mondatokat. Látni kell azonban azt is, hogy emellett az internetes keresőmotorok nyelvtől függetlenül lényegében azonos hatásfokkal működnek, tehát a nyelvfüggetlenség, az egyszerűség és a széleskörű használhatóság oltárán szükségszerűen feláldozzák a mondat „megértésének” képességét. Bizonyos alkalmazások esetében azonban ennél többre van szükség. Ha például azt szeretnénk elérni, hogy az informatikai rendszerek használata minél egyszerűbb és elfogadottabb legyen, ha fontos, hogy a tartalommal teljesen összhangban levő néhány dokumentumot találjuk csak meg, illetve ne kelljen bizonyos szakértői rendszerekben mélyre ható informatikai képzettséggel rendelkezni, akkor már érdemes kinyerni a kérdésekből a mélyebb tartalmakat is.

Hogy milyen járulékos feladatokat kell megoldani a mondatfeldolgozás során, az elsősorban alkalmazásfüggő kérdés — ezeket fogjuk áttekinteni a következőkben. Annyi bizonyosan elmondható, hogy valamiféle mondattani vagy szintaktikai elemzésre mindenképpen szükség van. Az elemzési feladat részletes ismertetése jelentősen meghaladná e könyv terjedelmét és el is térne a tárgyától, ezért a szintaktikai elemzésre a továbbiakban fekete dobozként, készen kapott alkalmazásként tekintünk.

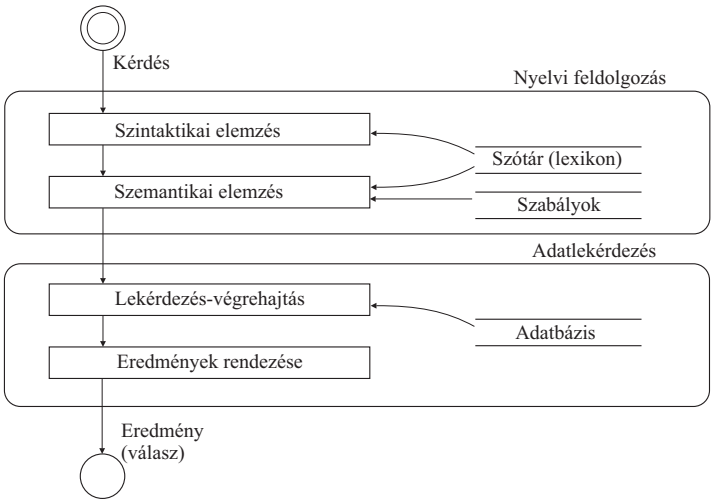
9.1. Természetes nyelvű adatbázis-interfészek

A *természetes nyelvű adatbázis-interfész*eknek (natural language interfaces to databases, NLIDB) azokat a szoftvereket nevezzük, amelyeknél egy adott természetes nyelven megfogalmazott, nyelvtanilag helyes kérdés (bemenet) megadásával adatbázisok rekordjainak halmazát (kimenet) kérdezzük le vagy dolgozzuk fel. Azaz az NLIDB a kérdésfeltevéstől az adatbázis-lekérdezésig vezető folyamatért felelős.

Az NLIDB egy speciális válaszkereső rendszer, amelyek alapvető célja, hogy természetes nyelvű, vagy ahhoz közeli dialógusban egyértelmű válaszokat adjanak a nem feltétlenül helyesen feltett felhasználói kérdésekre is, függetlenül attól, hogy a válaszadást ontológia, logika, adatbázis vagy egy neurális háló támogatja.

Hasonlóan elmondható, hogy a *természetes nyelvek megértése* (natural language understanding, *Nlu*) terén elért eredmények is széleskörűen használhatóak az NLIDB világában. Azonban egyfelől a kérdés „megértése" nem feltétlenül szükséges a megfelelő eredmény eléréséhez, hiszen az NLIDB tulajdonképpen egy komplex nyelvet annak egy jóval egyszerűbb, formális *részhalmazára* — a *lekérdezőnyelv*re — képezi le. Másfelől az is igaz, hogy olykor önmagában még a „megértés" sem elegendő a probléma megoldásához, hiszen egy jól feltett kérdés lekérdezéssé alakítása még az ember számára is kihívásokat jelenthet éppen amiatt, hogy a szűk nyelvi korlátok és a limitált kifejezőerő megfelelő kombinálásával kell a kívánt hatást elérni.[3]

Az NLIDB-k általános modelljét négy jellemző szakaszra lehet bontatni (lásd a 9.1. ábrát).



9.1. ábra. Az NLIDB általános, blokkvázlatos modellje

[3] Gondoljunk csak arra, hogy hogyan programoznánk le SQL nyelven egy olyan viszonylag egyszerű kérdést, mint *Ki volt Magyarország hetedik miniszterelnöke?*! A helyes és adaptív lekérdezés megtalálásához hosszú időre van szüksége még a jól felkészült, informatikus végzettségű szakembernek is.

1. szakasz: szintaktikai elemzés. A szintaktikai elemzés célja a nyelvi szerkezetek átalakítása számítógéppel könnyen feldolgozható struktúrává. Ahhoz, hogy a számítógép által jól értelmezhető, algoritmusok segítségével könnyen feldolgozható strukturált kifejezéseket kapjunk természetes nyelvű mondatainkból, elsősorban alkalmazott nyelvészeti és számítástudományi (formális nyelvi) ismeretekkel kell rendelkeznünk. A problémakör tehát jellemzően a tág értelemben vett alkalmazott nyelvészettel foglalkozó tudományágak kutatási területe. A továbbiakban erről csak érintőlegesen, a mélyebb összefüggések megértése végett lesz szó.

Itt jegyezzük meg, hogy bár sokan vélik úgy, hogy a szintaktikai elemzés határozza meg leginkább az NLIDB-k jóságát, illetve használhatóságát — és ez kétségkívül elhagyhatatlan és kiemelten fontos része a teljes folyamatnak —, mégis pl. az egyik legsikeresebb angol nyelvű természetes nyelvű kereső, az Ask[4] motorjánál a nyelvi-szintaktikai kérdések a találati pontosságot, meglepő módon, mindössze 5%-ban befolyásolják [72].

2. szakasz: szemantikai elemzés. A szemantikai elemzés során a számítógép által értelmezhető struktúrában szereplő kódolt utasítások algoritmizált leképezését végezzük el egy korlátozott kifejező erejű formális nyelvre, jellemzően egy univerzális adatbázis-lekérdezőnyelvre.

A számítógép számára sem a szavak egymásutánisága, sem a nyelvtani szerkezetek nem hordoznak semmilyen tartalomra utaló adatot, hiszen a nyelvtani szerkezet többnyire nem a mondanivalóval áll összefüggésben, a szavak, kifejezések pedig túl „zajosak” ahhoz, hogy adatokat, utasításokat ismerjünk fel, azonosítsunk belőlük. Ráadásul a számítógépek világról alkotott képe meglehetősen korlátozott, éppen ezért a helyes működéshez a számítógép számára explicit módon, adott matematikai struktúrában kell leírnunk a valós világot. Ez a világ azonban korántsem olyan széles, mint amilyennel a mesterséges intelligencia tudományterületéhez sorolt NLU kapcsán szokás foglalkozni, hiszen most kizárólag adatbázisok lekérdezését szeretnénk elvégezni. Márpedig minden adatbázis létrehozásakor már eleve megalkotunk egy egyszerűsített világmodellt, és ezt képezzük le egy megfelelő adatmodellre; ez minden adatbázis logikai szerkezetének alapja.

Következésképp az NLIDB-k esetében a szemantikai kapcsolatok leírásánál három kérdést kell alaposan megvizsgálni:

- Hogyan reprezentálják a világot az adatbázisok általánosított logikai szerkezetében?

[4] www.ask.com

- Hogyan feleltetjük meg a nyelvtani szerkezetekben kódolt utasításokat adatbázis-műveletek sorozatának?

- Hogyan képezzük le a bemenetként kapott struktúra egyes elemeit az adatbázis elemeire?

Röviden összefoglalva: a szintaktikai elemzést követően a feladat az, hogy a kapott struktúrát illesszük az NLIDB hátteréül szolgáló adatbázisok — röviden támogató adatbázisok — univerzális világmodelljére.

3. szakasz: az adatbázisok illesztése. A szemantikai elemzés során célszerűbb egyetlen univerzális adatbázis logikai modelljével foglalkozni, mint a támogató adatbázisok — az esetek többségében lényegesen eltérő — logikai szerkezetének összességével. Következésképp a szemantikai elemzés során előálló univerzális lekérdezést a támogató adatbázisok mindegyikére le kell tudnunk fordítani. Az NLIDB esetében az igazi kihívásokat az automatizmusok minél szélesebb körű alkalmazása jelenti, azaz azt kell elérni, hogy a támogató adatbázisok oldaláról minél kevesebb emberi erőforrást igényeljen a leképezési algoritmus megtalálása és végrehajtása.

4. szakasz: az eredmények rendezése. Az eredményeket a felhasználó szempontjai szerint érdemes rendezni, csoportosítani. A rendezési szempontok általában nem hasonlíthatók össze a keresési motoroknál megszokott rangsoroló eljárásokkal (ranking algorithm, ld. a 8.5.2. szakaszt), hiszen minden adatbázisbeli találat pontosnak számít. Éppen emiatt a válaszok csoportosítását a kiszolgáló adatbázisok (forrás) szerint, illetve a bemenetként kapott mondat alternatív értelmezései szerint érdemes elvégezni.

9.1.1. Egy rövid történeti áttekintés

A sikeres NLIDB-k megvalósítására négy különböző stratégiát alkalmaztak eddig: a szintaktikai elemzésre, a szemantikai elemzésre, a sablon alapú mintaillesztésre, valamint a köztes lépcsős leképezésre épülő eljárásokat. Az egyes megoldások témafüggetlenségi és kifejezőerőbeli sajátosságait a következőkben mutatjuk be. Részletesebb, a nyelvészeti kérdéseket is kimerítően tárgyaló összehasonlítás található a [8, 42, 134, 143, 151] irodalmakban.

Szintaktikus elemzőre épülő NLIDB. Az első NLIDB-k az első adatmodellekkel és adatbázisokkal párhuzamosan már az 1960-as években megjelentek. Az első sikeres, független adatbázisra épülő megvalósítás, a LUNAR [214] volt (1968), amely a Holdról származó, az Apollo-11 legénysége által gyűjtött kőzetek katalogizá-

lásában és kémiai elemzésében támogatta felhasználóit. Tipikus kérdés, amelyet már a LUNAR is meg tudott válaszolni:

1. *What samples contain magnesium?*
2. *Give me the modal analysis of magnesium in those samples.*
3. *What is the specific activity of A126 in soil?*[5]

A LUNAR működési elvének kialakításakor abból indultak ki, hogy a lényegesen különböző információk megszerzésére irányuló természetes nyelvű kérdések eltérő szintaktikai struktúrával rendelkeznek – azaz elegendő a szintaktikai elemzési fára építve egy olyan programkódot előállítani, amely a megfelelő átírási szabályok segítségével a kívánt hatást eléri.

Maga az átírás folyamata egy ún. ATN- (Augmented Transition Network, kiterjesztett vagy bővített átmenetháló) nyelvtan, és egy, az azt feldolgozó balelemző (top-down)[6] alkalmazásával valósult meg (lásd erről bővebben [4, 151, 213]).

A fókuszált felhasználási területnek köszönhetően a LUNAR, többszöri javítgatás után, a felhasználói kérdések 90%-át volt képes helyesen feldolgozni. Ehhez azonban számos olyan heurisztikával látták el, ami a későbbiekben a hordozhatóságnak — azaz újabb tématerületre való adaptálásának — egyik gátjává vált [8, 214]. A hordozhatóságnak azonban volt egy másik, sokkal nehezebben leküzdhető akadálya is, nevezetesen a LUNAR az adatbázis-lekérdezés létrehozásakor kifejezetten a szintaktikára koncentrált, és egyáltalán nem használta ki az adatbázisban tárolt világmodellben rejlő lehetőségeket. Ennek megfelelően nem is tudott adaptív módon illeszkedni egy újabb, jelentősen eltérő adatbázis-struktúrához.

A szemantikus elemzés módszere. Az 1970-es években megjelenő új generáció az ún. *szemantikai elemzés*t próbálta előtérbe helyezni. A szemantikus elemzés újítása az volt, hogy a mondatfeldolgozás során bizonyos szemantikai információkat is figyelembe vett, azaz egyrészt eliminálta a szemantikailag helytelen szintaktikai elemzési fák jelentős részét, másrészt közvetlenül adatbázisra illeszthető kódot állított elő. Szemantikus elemzést használtak a REL [122], PLANES [202], LADDER [85], EUFID [182] és az LDC-1 [16] rendszerekben is [8, 143]. A szemantikus elemzés módszerét annak talán legismertebb képviselőjén, a LADDER példáján keresztül mutatjuk be.

[5] A három példamondat fordítása: Milyen minták tartalmaznak magnéziumot? Mutasd azokban a mintákban a magnézium jellemzőit! Milyen sajátos viselkedése van az A126-nak talajban?

[6] a terminológiát Bach Iván Formális nyelvek [13] c. könyvéből kölcsönözve

A LADDER a LUNAR-hoz hasonlóan erősen témafókuszált volt, elsősorban az Egyesült Államok haditengerészeti döntéshozói számára tervezték egyszerű információkeresési feladatok (pl. *Hol tartózkodik a JFK?*, *Milyen osztályú anyahajók vannak Fort Laudersdale-ben?*) teljesítésére. A szemantikus elemzéshez egy jobbelemző (bottom-up parser) előszűrés után egy balelemzőt (top-down parser) (vö. Coke–Younger–Kasami-szűrő [13]) használtak[7] [142], ami a LADDER-hoz képest jelentősen javított a feldolgozási sebességen.

A LADDER működését illusztrálandó legyen a bemenet elemzésére használt Γ-val jelölt nyelvtanunk az alábbi:

S	→	Minta_kérdés | Hajó_kérdés
Minta_kérdés	→	Minta Kibocsátás | Minta Tartalmazás
Minta	→	*'melyik kőzet'* | *'melyik minta'*
Kibocsátás	→	*'bocsát ki'* Sugárzás
Sugárzás	→	*'sugárzást'* | *'fényt'*
Tartalmazás	→	*'tartalmaz'* Anyag
Anyag	→	*'magnéziumot'* | *'kalciumot'*
Hajó_kérdés	→	Hajót Indulás | Hajó Érkezés
Hajót	→	*'melyik űrhajót'* | *'melyik rakétát'*
Hajó	→	*'melyik űrhajó'* | *'melyik rakéta'*
Indulás	→	*'lőtték fel'* Dátum | *'indították'* Dátum
Érkezés	→	*'tért vissza'* Dátum | *'ért földet'* Dátum

Mivel a formális nyelvtani szabályok konstrukciójának köszönhetően csak bizonyos szemantikafüggő elágazások mentén tudunk haladni, így olyan mondatokat, mint a *Melyik kőzet tartalmaz fényt?* azonnal elvethetjük — szemben a LUNAR megoldásával. Ráadásul a csomópont bejárása során fokozatosan azonosíthatjuk a kérdés kontextusát. Lényegében a fa bejárása során már előbb is tudjuk állítani a megfelelő lekérdezést.

Felmerülhet persze a kérdés, miért volt fontos, hogy az értelmetlen, helytelen mondatok elemzését gyorsan vessék el? A tapasztalatok [86, 137] azt mutatták, hogy a fel nem dolgozott mondatok mintegy 80%-ában a felhasználók nyelvhelyességi hiányosságai okozzák a problémát. Márpedig a korabeli számítógépek sebességi korlátainak köszönhetően a felhasználók aránylag hosszú időt várhattak egy olyan válaszra, amelyből kiderül, hogy már a kérdés maga is rosszul lett feltéve, ez pedig hosszabb távon a szoftver használhatatlanságához vezetett volna.

Míg a LADDER az elemzés sebessége és pontossága terén aránylag jó eredményt ért el, addig a hordozhatóság tekintetében súlyos hiányosságai voltak. A

[7] A szélességi bejáráson alapuló balelemzés lényegesen rosszabb eredmény adott [85, 175].

szemantikailag helyes kombinációk kimerítő leírásával a megoldás tökéletesen alkalmatlan új tématerületekre való alkalmazásra, hiszen minden egyes tématerületre egy teljesen új szabályrendszert kell az alapoktól kezdve megalkotni. Ráadásul kifejezetten nyelvfüggő, hiszen például a magyarban — az angollal ellentétben — a toldalékolás, illetve a kötelező vonzatok miatt az elágazások száma exponenciálisan megnőhet (lásd a fenti példában a Hajó és Hajót eseteket), ami a hatékony felhasználás rovására mehet.

A hordozhatóság fontosabbá válásával a szemantikai elemzésnek e módszere fokozatosan eltűnt.

A sablon alapú mintaillesztő eljárás. A módszer egyszerűsítésével azonban, kisebb kompromisszumok árán, csökkenteni lehet a témafókuszáltságból eredő problémákat. Nevezetesen, ha nem formális nyelvtannal, hanem mondat- vagy *kifejezéssablon*okkal (frázissablon, phrasal template) való mintaillesztéssel próbálkozunk, akkor a szemantikus elemzés legfontosabb képességeit megőrizzük, miközben a szabályrendszerünk hordozhatóvá válik. Például a fenti Γ-nyelvtant sablonok segítségével az alábbi módon írhatjuk fel:

Melyik <alany> <ige> <dátum>?
Melyik <tárgy> <ige> <dátum>?
Melyik <alany> <ige>?

A sablonokra illeszkedő kérdőmondatokat aztán egységes formában lehet lefordítani adatbázis-lekérdezésekké. Természetesen, a szabályok általánosításával a helytelen mondatokat már nem feltétlenül vetjük el, így a *Melyik **űrhajó** lőtték fel 1968-ban?** kérdésre valamilyen — nem feltétlenül helyes — választ fogunk kapni.

A figyelmes Olvasó észreveheti, hogy a sablon alapú eljárás nagyon hasonlatos az *n*-grammok módszerének (ld. 39. oldal) szó alapú verziójához. Ekkor a szöveg összes lehetséges *n* számú egymást követő szavát vesszük figyelembe az elemzés során, és abból vonunk le következtetéseket. A sablon alapú módszer az *n*-grammok egy speciális esete. Itt egyrészt a kifejezések hossza meghatározza *n* értékét, másrészt a nyelvtani szabályoknak megfelelően a felesleges *n*-grammokat figyelmen kívül hagyjuk.

Az eljárást a START [95] QAS-rendszerben[8] tökéletesítették 1988–1993 között, majd a szemantikus web és a mélyháló mint probléma megjelenésével kiterjesztették szövegadatbázisokra is [96, 97]. A START, illetve a hasonló elven működő

[8] `start.csail.mit.edu/`

Ask alapjául szolgáló InPlainWords motor esetében a sablon az <alany>, <tulajdonság>, <érték>[9] trigrammból áll [72, 96].

A sablon alapú mintaillesztést követő eljárással 75%-os találati pontosságot értek el egy családjogi tanácsadó honlap keresőjén végzett tesztelés során [72]. A helytelen válaszok 50%-a a helytelen elemzéssel, 20%-a az adott szó vagy kifejezés értelmezhetőségének hiányával volt magyarázható, míg a sikertelen kísérletek 30%-ára azért nem adott választ, mert nem is volt róla információ a honlapon.

A módszer alkalmazása azonban meglehetősen költséges: a családjogi tanácsadó esetében négy iterációs fejlesztés során mintegy 2000 új sablont és hozzá kapcsolódó átírási szabályt kellett feltölteni a rendszerbe [72]. Ráadásul az eljárás nem, vagy csak részben képes feloldani számos jellegzetes nyelvtani szerkezetet, mint amilyen például a gyakran használatos birtokos szerkezet vagy a felsőfokú jelzővel ellátott főnévi frázis.

Köztes lépcsős megvalósítás. A szintaktikai elemzés témafókuszáltságból eredő túloptimalizálásának feloldására, a hordozhatóság növelése érdekében egy köztes reprezentációs nyelvet érdemes beiktatni. A köztes nyelv szinte minden esetben egy formális logikai nyelv, a JANUS [11, 204], a PHILQA1 [151], az IRUS [17, 205] és a GPSG [74] esetében a Montague-nyelvtan, a CLE [5, 6], a CHAT-80[10] [203] és a Masque/SQL [7] esetében az elsőrendű logika nyelve.

A köztes lépcső beiktatásával — a hordozhatóságon túl — a többnyelvűség megteremtésére is lehetőség kínálkozik, hiszen erre az elvre építve akár kétnyelvű NLIDB-t is létre lehet hozni (lásd pl. [93, 155]).

A köztes lépcsős eljárás viselkedését nézzük meg a *Masque/*SQL [7, 8] példáján keresztül! A Masque a szintaktikai elemzés során előálló elemzési fát először a szintaktikai konstrukciónak megfelelően, félig vagy teljesen kitöltött logikai kifejezéssé alakítja. Például a *Mely kőzetek tartalmaznak magnéziumot?* kérdésben szereplő atomi (kőzet, tartalmaz, magnézium) kifejezéseket rendre az alábbi logikai kifejezésekké alakítja egyszerű szótári leképezés segítségével: $\lambda x.\mathsf{is\_rock}(x)$, $\lambda x.\lambda y.\mathsf{contains}(x,y)$, 'magnézium'. A szintaktikai elemzési fa bejárásával, univer-

[9] Nem összekeverendő a szemantikus web RDF (Resource Description Framework) esetében használatos alany, állítmány, tárgy hármassal. Az RDF feladata az univerzális adatbázis-leírás, így minden esetben két fogalmat kapcsol össze egy reláció mentén. A START azonban az adatbázis-lekérdezést, illetve a válaszkeresést tekinti fő feladatának, így a kérdés megválaszolása szempontjából magát a kérdést tipizálja az alany, tulajdonság, érték hármassal. A különbségre jó példa lehet, hogy míg az RDF állítások gyakran szimmetrikusan is értelmezhetőek, a START trigrammjaira ez nem igaz.

[10] `web.sfc.keio.ac.jp/~t03712sn/chat80/chat80.cgi`

zális egyesítési szabályok alapján pedig az alábbi Prolog-programot építi belőle, amit közvetlenül logikai adatbázisra is lehet alkalmazni, de egyszerű eljárással akár SQL nyelvre is fordíthatjuk.

```
answer( X ) :-
  is_rock( X ),
  contains( X, 'magnézium' ).
```

A módszer igazi ereje abban rejlik, hogy csak azokat a kifejezéseket és szerkezeteket veszi figyelembe, amely a szoftver hátteréül szolgáló adatbázisban közvetlenül elérhetők predikátumok formájában. Hordozható, hiszen a lexikon tartalmát kell csak leképezni az adatbázis megfelelő elemeire — ezt pedig mindenképpen el kell végezni bármilyen NLIDB-megvalósításról is legyen szó.

Jól látszik azonban az is, hogy a köztes lépcsős megoldás jósága nagyban függ az adatbázis szerkezetétől. Vegyük például azt az életszerű helyzetet, hogy pilótákat, repülőgépeket, célállomásokat és indulási időpontokat tárolunk az adatbázisunkban, azaz hogy melyik repülőgép, kivel, hova és mikor repül. Joggal várnánk el ettől a rendszertől, hogy a *Melyik pilóta repül Berlinbe?*, *Mikor repül a Boeing-747-es?*, *Mikor repül a 2008. december 24-i járat?*, *Hova repül a Boeing-747-es gép 2008. december 24-én?*, *Mikor repül Gipsz Jakab?* kérdések mindegyikére választ adjon — feltéve persze, hogy a rendszer számára rendelkezésre állnak a szükséges adatok. A *repül* ige megfeleltetése (interpretálása) azonban itt komoly nehézségekbe ütközik, hiszen az vagy egy négy argumentumból álló predikátumra (repül(mi, kivel, hová, mikor)), vagy négy darab, eltérő argumentumú predikátumra képzendő le (azaz repül(ki, hova), repül(mi, mikor), repül(mi, mikor, hova), repül(ki, mikor)). Előbbi esetben az univerzális egyesítés okoz majd nehézséget, hiszen nem tudhatjuk előre, hogy melyik argumentum helyére írandó az adott kifejezés — mivel csak a behelyettesítendő elem típusából következne ez az információ, ha az egyértelmű —, míg az utóbbi esetben mind a lehetséges kiértékelendő lekérdezések (kombinációk) száma, mind a kiértékelés válaszideje is jelentősen megnőhet.

Az újabb fejlesztésű, valószínűségi környezetfüggetlen nyelv alapuló *Precise* [148] az előbbi megoldást választja méghozzá úgy, hogy minden névelemhez számon tartja, hogy az mely attribútum értékeként fordulhat elő az adatbázisban. A Precise csak addig működik helyesen, amíg a mondaton belül szereplő névelemek csak különböző típusba sorolhatók: így helyesen működik a *Milyen munkákat kínál a HP Unix rendszerek programozására?* kérdésre (nincs átfedés), de helytelenül a *Hol született Margaret Thatcher édesanyja?* (van átfedés: Margaret Thatcher édesanya is lehet egyben).

Birtokos típusok	Példakifejezések	
	magyar	angol
származás-, forrásleírás	Moszkva küldötte	men of Rome
anyagleírás	–	ring of gold
rész-egész viszony	a tanszék vezetője	head of department
mennyiségi leírás	húsnak kilója	pound of beef
(állandósult) kapcsolat	Péter felesége	Pam's address
birtoklás	Sára sapkája	John's coat
alanyiság	Verdi operája	dramas of Shakespeare
tárgyiasság	Budapest látképe	portrait of Elisabeth II
cél- és szándékleírás	dolgozók iskolája	school of girls
láncolás (halmozás)	Ábel apjának barátja	name of Tom's wife

9.1. táblázat. Birtokos szerkezetek típusai

A rendszer azt is feltételezi, hogy minden nyelvi szerkezet általánosan megfeleltethető logikai állításoknak, függetlenül a szerkezetben szereplő konkrét elemektől. A természetes nyelvtani struktúrák azonban, mint pl. a birtokos szerkezet, nagyon változatos szemantikai kapcsolatokat képesek kifejezni (lásd a 9.1. táblázatot). Éppen ezért legfeljebb a kifejezésben szereplő elemek tipizálásával, az elemek egymáshoz való viszonyának elemzésével állapítható meg helyesen, hogy milyen megfelelő logikai formulára lehet, illetve kell lefordítani a kifejezést. A köztes lépcsős megoldások azonban erre nem kínálnak érdemi lehetőséget.

Nem angol nyelvű NLIDB*-k.* Az 1980-as éveket követően az angol nyelvű NLIDB területén végzett kutatások háttérbe szorultak. Ez részben amiatt következett be, mert a kereskedelmi forgalomban nem sikerült az NLIDB-vel versenyelőnyre szert tenni a közismert SQL nyelven lekérdezhető adatbázisokkal szemben, sőt, számos (jogos) negatív kritika is érte az NLIDB-k általános használhatóságát [137, 175].

Ráadásul 1986-ban az SQL nyelvet az ANSI, és tőle függetlenül az ISO szabványügyi testület is a relációs adatbázisok standard lekérdezőnyelveként fogadta el.[11] Ezzel párhuzamosan pedig az angolszász nyelvterületeken egyre több és képzettebb szakember használta már a számítógépet, így az NLIDB-k kutatása döntően a kevesebb számítástechnikailag képzett munkaerővel bíró országokra tevődött

[11] A világ egészére kiterjedő közös (ANSI/ISO/IEC) szabvány viszont csak 1989-ben született meg.

át. A teljesség igénye nélkül az alábbi nyelveken valósítottak meg NLIDB-t az elmúlt húsz évben:

- Német nyelvre: IDA [210] (szemantikus elemzés), PLIDIS [24, 151] (szintaktikai elemzés), NAUDA [106] (sablon),
- Spanyol nyelvre: Spanish NLQ [154] (köztes lépcsős megvalósítás, klm), Sylvia-NLQ,[12]
- Svéd nyelvre: Phoenix [37] (klm),
- Portugál nyelvre: Edite [155] (klm), LIL/SQL [67] (klm),
- Kínai nyelvre: NChiql [132] (klm),
- Koreai nyelvre: KID [113] (szintaktikus elemző), SiteQ [93] (klm),
- Lengyel nyelvre: Dialog [27] (szintaktikus elemző visszacsatolt ATN-nel),
- Magyar nyelvre: THALES [61], CONSTRUCTOR [3] (szintaktikus elemző attribútumnyelvtannal)

A súlypont eltolódása azonban — ahogyan a fenti felsorolás is jelzi — nem jelentett lényeges szemléletváltást. Az újabb megvalósítások főleg az angol nyelvre már bevált, bizonyított módszereket próbálták egy új nyelvterületen — azaz főleg a bemenet megváltoztatásával — újrahasznosítani. Az NLIDB-k kutatása csak a mélyháló (lásd 9.2. szakasz) problémájának [20] 2001-es megjelenésével került újra reflektorfénybe — immár egészen más céllal és megközelítésben.

9.2. Keresés a mélyhálóban

Az interneten *szabadon* hozzáférhető (web)oldalakat jellemzően két nagy csoportba szokás sorolni attól függően, hogy az általánosan elérhető, böngészhető tartalmak közvetlenül elérhetőek, vagy kérésre — programok, folyamatok — állítják őket elő futási időben. Az előbbieket *statikus*, az utóbbiakat *dinamikus weboldal*aknak nevezzük.

A megjelenítő eszközök, böngészők a statikus oldalakat egyetlen dokumentumként olvassák végig, és tartalmi elemeiket a dokumentumon belüli vezérlési információk segítségével jelenítik meg a képernyőn. Ez azt jelenti, hogy a statikus oldalak alapvetően képtelenek állapotok tárolására, azaz bármilyen felhasználói művelet csak külön kérésre, a szükséges legrövidebb időtartamra épül fel. Továbbá a kiszolgáló voltaképpen nem emlékszik vissza a felhasználóra, vagyis a tizedik látogatása a kiszolgáló számára pont ugyanolyan, mint az első volt.

[12] `www.lllf.uam.es/proyectos/sylvia.html`

A statikus oldalak elsődleges előnye az egyszerűség, valamint az, hogy egyszeri információközlés esetén feldolgozásuk jó hatásfokú.

A dinamikus oldalakat ezzel ellentétben emlékezettel, sőt interaktív elemekkel lehet ellátni. Az internet történetében akkor kezdtek ezek a technológiák megjelenni, amikor felmerült az igény gyorsan változó tartalmak közlésére. Ha a tartalom (hírek, játékok, vagy személyre szabható oldalak esetében) gyorsan változik a kiszolgáló szemszögéből nézve(!), akkor az erőforrásokat arra kell koncentrálni, hogy az oldal kinézete változatlan maradhasson, azaz a szerkesztést végző felhasználó kizárólag a legfontosabb, nélkülözhetetlen feladatokat végezze csak el. Úgyis lehet fogalmazni, hogy az olvasói felhasználóbarátság mellett megjelent a szerkesztői felhasználóbarátság igénye is.

A két típus között lényeges különbség alapvetően a változás sebességében van. Természetesen itt nem arról van szó, hogy pl. egy cikk tartalma folyamatos változáson megy keresztül — bár kétségtelenül ez is előfordul —, hanem arról, hogy egy adott honlap internetes címén (pl. egy hírújság portálján) a látható tartalom meglehetősen gyorsan változik. Ennek megfelelően a statikus oldalakat könnyű bejárni, keresni, hiszen hosszú ideig változatlanok, akár lassú algoritmusokkal is könnyedén hozzáférhetünk szinte minden oldalhoz. Ugyanakkor a dinamikus oldalak mögött jellemzően programozási eszközök és különböző adatbázisok, adattárak vannak. Ha egy látogató „későbbi" időpontban tekinti meg az adott honlapot, akkor az adatbázis más elemei jelennek meg számára; közvetlenül nem képes hozzáférni a korábbi tartalmakhoz. Pedig azok is rendelkezésre állnak — a legtöbb szolgáltató külön erre a célra keresőoldalakat tart fenn az oldalain. Ezzel viszont máris kialakulnak az első értéknövelt tartalomszolgáltatások, a portálok, ami jó a felhasználóknak — ugyanakkor a keresőmotorok számára az információk összegyűjtése egyre reménytelenebbé válik. Fontos kiemelni és látni, hogy a dinamikus oldalak háttértartalmát egy keresőmotor akkor és csakis akkor dolgozhatja fel, ha erre a kiszolgáló tulajdonosa explicit lehetőséget biztosít.

A kereshetőségi, ha úgy tetszik a *keresőmotorok láthatósági szintjének* megfelelően tehát megkülönböztetünk ún. *felszíni* (hagyományos keresőmotorokkal kereshető) és ún. *mély-* (hagyományos keresőmotorokkal nem kereshető) hálót. Az internetnek van két további jellegzetes rétege, amelyek ritkán, vagy elvétve kapcsolhatóak be az általános keresési folyamatokba. Az egyik ilyen réteg az ún. *láthatatlan háló*, amelynek tartalma nem mindenki számára hozzáférhető — vagy azért, mert korlátozott számú felhasználóra van szabva, vagy mert pl. tűzfal védi. Ide sorolhatóak az amúgy közcélú, és sok esetben nyilvános adatokat tartalmazó kormányzati dokumentumtárak is. A másik ilyen réteg az ún. intim réteg, a

személyes felhasználók háttértárait és dokumentumait tartalmazó számítógépek. Mivel a személyeket a világ minden demokratikus államában adatvédelmi törvények védik az illetéktelen hozzáférések megakadályozása érdekében, így ezek a tartalmak a technikai korlátok mellett csak az önkéntesség és a megfontolás, a visszalépés szabad biztosításának elve szerint és mellett gyűjthetőek össze, valamint nem kereshetőek.

A mélyháló kereshetővé tételének alapvetően az adja a jelentőségét, hogy az interneten található összes tartalomnak csak mintegy 10%-a érhető el hagyományos, jól ismert keresőmotorok (pl. a Google, Yahoo, MSN) segítségével, a többi rejtve marad. A rejtett tartalom feltárása máig nyitott kérdés, általános megoldás ugyanis még nem született. A következőkben bemutatjuk a legsikeresebb elképzeléseket, illetve megmutatjuk, hogy a válaszkereső rendszerek és az NLIDB esetében már látott hordozhatóság hogyan segíthet idővel orvosolni a fennálló problémát.

9.2.1. Keresés metakeresővel

*Metakereső*nek nevezünk minden olyan keresőmotort, amely már létező keresőmotorokra „ül rá”, azaz a bemenetet valamilyen transzformáció után más keresőmotoroknak továbbítják, és azok eredményeit mutatják meg a felhasználónak. A metakeresőknek az elmúlt években két fajtáját fejlesztették ki. Ezeket az különbözteti meg egymástól, hogy a metakeresést végző szoftveralkalmazás egy központi, mindenki számára egységes helyen és felületen keresztül, vagy a böngészőben, esetleg annak valamilyen csatolt alkalmazásaként (plugin) érhető el. Előbbit *szerver oldali*, utóbbit *ügyfél oldali* metakeresőnek szokták nevezni.

A metakeresők általános sémáját az alábbiakban lehet meghatározni:

1. A felhasználó számára kínálj fel a metakereső által használt keresőkhöz (továbbiakban *célkereső*k) hasonló űrlapot (nyomtatvány, angolul: form). Ha a metakereső több célkeresőt is használ egyidejűleg, akkor érdemes a legkisebb szemcsézettséget választani az adatokból, és a célkeresők által támogatott keresési lehetőségek unióját vagy metszetét felkínálni. Előbbi lehetőség esetében a célkereső által nem támogatott elemek eldobandók a keresőkifejezés célkeresőhöz való továbbítása során. A legkisebb szemcsézettség választása azért indokolt, mert pl. nevek esetében a teljes név ismeretében a kereszt- és vezetéknevet utólag sokkal nehezebb meghatározni, mint a keresztnév és a vezetéknév segítségével a teljes nevet.

2. A metakereső űrlapjának elemeit képezzük le a célkereső űrlapjának elemeire – figyelembe véve a szemcsézettséget, illetve a támogatott keresési funkciók körét.
3. A válaszokat valamilyen egyszerű algoritmus szerint rendezzük.
4. A rendezett válaszokat tömbösítjük: vagy az azonos forráshoz tartozó elemeket fogjuk össze egyetlen tömbbe, vagy ugyanazon találatok különböző forrásait. Ezt úgy kell érteni, hogy általános keresés esetében pl. az a fontos, hogy a találatot ki szolgáltatta, így az adott szolgáltató válaszait érdemes aggregálni, míg pl. termékek keresése esetében a konkrét terméktípusokat érdemes összefogni, egy termékhez több árat és forgalmazót megjelenítve.
5. A felhasználó választásait, preferenciáit érdemes figyelemmel követni, amely a metakereső rangsorolási technikáját nagyban javíthatja, illetve jelentősen módosíthatja.
6. A válaszokat a metakereső külső hivatkozásokkal együtt jeleníti meg, egyetlen célkereső esetében annak oldalára irányítja át a felhasználót.

Ügyfél oldali metakereső ma már az összes népszerű böngészőben található, de az első a Macintosh-világban ismert, MacOS 8.5-ben elérhető Sherlock volt. Hozzá kell tennünk, hogy az ügyfél oldali keresőmotorok elsősorban az általános keresést támogatják, azaz közvetlenül a hagyományos keresőmotorokhoz továbbítják a bemenetként kapott keresőkifejezéseket. A mélyhálóban ezért — értelemszerűen — csak rossz hatásfokkal lehet velük keresni. A megoldás előnye, hogy egyszerű, a felhasználó számára kevesebb interakcióval elérhető szolgáltatást biztosít, és könnyen felismeri, tanulja a felhasználó szokásait, preferenciáit. Speciális, tématerületen belül összetett, esetlegesen szakértői keresést (pl. egy könyv minél olcsóbb megvásárlása is lehet ilyen) azonban csak nagyon nehézkesen tesz lehetővé. Az ügyfél oldali metakeresők ezt azért nem támogatják mert, minden tématerületre külön kell felkészíteni, címekkel, szolgáltatói listával feltölteni, bővíteni, kiegészíteni az alkalmazást. Problémát jelent az is, hogy bár a szokásainkhoz jól illeszkedik, más számítógépre átvinni a megfelelő beállításokat nehézkes, időigényes vállalkozás.

A *szerver oldali metakeresők* ezzel szemben folyamatosan fejleszthetők a felhasználók igényei szerint, tématerületei, a bekapcsolható szervezetek listái is rendre bővíthetőek. Éppen e sajátosságából következően a szerver oldali metakeresők már nem nagy, általános célú keresőmotorokhoz, hanem közvetlenül a tartalomszolgáltatók keresőmotorjaihoz csatlakoznak a mélyhálós tartalmakhoz való hatékony hozzáférés érdekében. Például a Magyarországon az egykori NeKeress

szolgáltatás a bemenetként kapott kifejezéseket a vele szerződéses viszonyban álló könyvesboltok keresőszolgáltatásai felé továbbította, majd az eredményeket cím és ár alapján, azon belül pedig a válasz megérkezésének idősorrendjében rendezte. Bár az ilyen jellegű keresők nagyon hatékonyak egy tématerületre, komoly gondot okoz azonban, hogy minden őket érdeklő területre a legjobb néhány szolgáltatás címét és nevét a felhasználók fejben tartsák. A megoldás egy általános, de legalábbis néhány száz tématerületre fókuszáló szerver oldali metakereső lehet. Ilyen megoldás például az angol nyelvterületre optimalizált CompletePlanet.[13]

A tématerületek számának növekedésével néhány új kihívással is meg kell küzdeni. Gondoljunk csak arra, hogy nem küldhetünk szét minden keresőkifejezést minden partneroldal felé, hiszen ez egyrészt nyilván értelmetlen, másrészt indokolatlanul komoly hálózati forgalmat generál. Érdemes a keresőkifejezést valamiféle osztályozásnak alávetni, hogy a megkérdezendő oldalak körét szűkítsük, és ténylegesen csak a releváns tartalomszolgáltatók felé küldjük el a keresendő kifejezést. Másik jellegzetes probléma, hogy a különböző oldalakról visszaérkező eredmények valamilyen rangsort már jeleznek, de ezek egymással nem hasonlíthatók össze. Sőt, az is probléma, hogy az egyes forrásoldalak üzemeltetői más-más szakértelemmel, rálátással rendelkeznek egy-egy adott témát tekintve, következésképp a témához jobban illeszkedő forrást célszerű előrébb rangsorolni, mint az azzal csak érintőlegesen foglalkozót. Itt szeretnénk emlékeztetni az Olvasót, hogy metakeresőről beszélünk, tehát a kereső számára az oldal tartalma a keresést követően válik ismertté. Az eredmények előzetes rangsorolása nem megoldható, a keresés kiszolgálásánál pedig erre sem idő, sem hálózati sávszélesség nem áll rendelkezésre.

A probléma súlyosságát érzékelhetjük, ha pl. a *target price* (célárfolyam) kifejezést adjuk a szerver oldali metakeresőnek. A célárfolyam kifejezés, kontextusát tekintve, majdnem bizonyosan tőzsdei információkra vonatkozik, de ha a szavakat külön-külön értelmezik (a partneroldalak keresőit nehéz befolyásolnia a metakeresőnek!), akkor a legtöbb forrás találatot fog jelezni. A metakereső csak annyit tehet, hogy a beérkezett válaszok, illetve azok metaadatai alapján, valamilyen technikával kiválaszt közülük néhányat. Például a CompletePlanet esetében a kiválasztási algoritmus a forrásoldal nevét rangsorolja magasabb szinten, emiatt az első 86 találat a `www.target.com`-ról származik, és csak a 97. az első relevánsnak tekinthető találat. Általános megoldás jelenleg nem ismert e problémákra.

[13] `www.completeplanet.com`

9.2.2. Kooperációs megoldások

A metakereső vagy egyetlen célkeresőhöz, vagy több, egymással közvetlenül nem kommunikáló szolgáltató keresőjéhez csatlakozik. Az együttműködés hiányából adódik, hogy az űrlapok jellemzően különbözőek, a metakereső bemeneti sémáit ezekhez kell igazítani. Ha a metakereső által megcélzott szolgáltatók között megegyezés van, akkor közös, integrált kereső létrehozása jóval egyszerűbb lehet, hiszen a szolgáltatók megegyezhetnek a keresési funkciók köréről. A megegyezésnek több formája is elképzelhető, így beszélhetünk

- *független adatforrás*ok esetén
  - lekérdezőnyelv alapú megegyezésről,
  - taxonómia alapú lekérdezésről,
  - teljes formai megegyezésről,
- illetve *adatszövetség*ről.

Független adatforrásokról akkor beszélhetünk, ha a lekérdezésre szánt információk autonóm szervezeteknél, adatgazdáknál találhatóak meg, a tárolt információk nem migrálódnak, illetve nem kerülhetnek át más szolgáltatókhoz. Ezek egységes lekérdezését vagy úgy tehetjük lehetővé, hogy a lekérdezéshez használt közös nyelvet (pl. SQL, Z39.50, XPath), vagy a lekérdezhető sémák alapszerkezetét (pl. OPAC, ETO), vagy mindkettőt (pl. Dublin Core, Open Archive Initiative Protocol) rögzítjük.

A közös taxonómia megteremtésével eleve kizárjuk annak a lehetőségét, hogy többféle rendszert könnyen csatlakoztatni tudjunk. Ilyenkor alapvetően a versengő szolgáltatók azonos tartalommal rendelkeznek. Ebben az esetben már másodlagos kérdés, hogy ezeket hogyan vonjuk össze közös gyűjtőrendszerbe: önálló szerver szolgálja ki a felhasználói kérdéseket és az csak a konkrét információkra vonatkozó adatokat gyűjti be az egyes szolgáltatóktól (vékony kliens megoldás); a keretrendszeren belül a szerver közvetíti a szolgáltatók felé a kéréseket, de azok önállóan dolgozzák fel azokat (vastag kliens megoldás); vagy teljes, egyenrangú erőforrás-elosztással (tipikusan ún. peer-to-peer technikával) közösen értékelik az eredményeket — kiemelten kezelve a kérdésfeldolgozást kezdeményező csomópontot.

Közös taxonómia hiányában általában szükség van egy kontextusazonosítóra, illetve az előző szakaszban ismertetett kiértékelő rendszerre. Előbbire megoldás, ha a témát egyetlen területre fókuszáljuk — ez a tipikus —, míg az utóbbiból esetlegesen adódó problémákat a kooperációs megoldások úgy kerülik meg, hogy

a potenciális forrásokat limitálni engedik — például egy könyvtárközi keresés esetén a megkérdezendő könyvtárak nevét vagy számát.

Ha az adatok nem függetlenek, azaz mozgathatóak, másolhatóak, akkor teljesen más technikákkal érdemes dolgozni. Érdemes egyetlen, univerzális nyelvet és feldolgozót kidolgozni, amelynek számtalan példánya segíti a rendszer hatékony működését. Ilyenek például a határokon átívelő nagyvállalati levelezőrendszerek is. A megoldás részletezése meghaladná a könyv kereteit — ráadásul igen kiterjedt szakirodalma is van, akár a szövetséges adatbázisok, akár a grid, akár az adatfarmok területére gondolunk —, emiatt ettől eltekintünk.

9.2.3. A mélyháló és a válaszkereső rendszerek

A mélyháló felderítésére alkalmazott válaszkereső rendszerek alapvetően nem sokban térnek el a természetes nyelvű adatbázis-interfészektől. Első ránézésre a két rendszer közötti különbség csak az, hogy az NLIDB minden esetben konkrét adatbázisokat, adatbázissémákat kérdez le, míg a mélyháló esetében oldalakra, oldalakon található információkra van szükségünk.

A különbség komoly technológiai problémákat is felvet. Gondoljunk csak arra, hogy egy adatbázisban egy konkrét rekord megtalálásánál nem okozott gondot a rekordhoz való hozzáférés, hiszen a válaszkereső rendszer ezirányú jogosultsága könnyen beállítható. A mélyháló esetében azonban ez nem így van. A tartalomszolgáltatók saját, közvetlenül nem hozzáférhető állományokkal dolgoznak, az elérhető információk pedig nem rekordok, hanem szöveges dokumentumok formájában állnak rendelkezésre. Ahhoz, hogy az NLIDB-alkalmazásunkat mélyhálókereső motorrá alakíthassuk, szükség van egyfajta szövegből adatbázissémákra (vagy fordítva) leképező algoritmusra úgy, hogy az egyes „adatbázis elemek” (rekordok) egyedi azonosítója mindig a forrásoldal címe legyen, ahonnan az ténylegesen származik. Másrészt valahogyan el kell érni a forráscím oldalait — láttuk, hogy ez a ma használatos webkeresők egyik fő problémája is.

A mélyháló oldalainak címkézése, illetve az oldalon található információk „rekordosítása” első hallásra kellemetlenül nagy feladatnak látszik. Vegyük azonban észre, hogy a mélyháló esetében eddig bemutatott megoldások részben-egészben megoldják a problémát, hiszen a forráscímhez kapcsolódó, valamilyen már meglévő (pl. éppen a szolgáltató által kifejlesztett) keresőmotorhoz kapcsolódnak űrlapokon keresztül. Az eddigi megoldások lényegében csak abban térnek el, hogy mennyire szabványos, nyílt űrlapokkal dolgoznak. Az űrlapok viszont nagyon emlékeztetnek az adatbázisok sémaszerkezetére, azok kitöltése pedig alapvetően keresőkifejezésekre, rekordokra. Ráadásul az űrlapok kiértékelésének eredmé-

nyeképpen éppen a releváns oldalak állnak elő, forráscímükkel azonosítva. Az űrlapok kitöltése pedig sokkal egyszerűbb feladat, mint a konkrét formális nyelvű lekérdezés előállítása, hiszen ezek szinte mindig egyszintűek (ebből következően bizonyosan veszítünk jelentéstartalmat), míg a formális nyelvű lekérdezés szemantikai megfeleltetése kell legyen a felhasználó által megadott természetes nyelvű kérdésnek. Következésképp az NLIDB-t a mélyhálóban való keresésre annyira kell átalakítani (helyenként egyszerűsíteni), hogy a különböző űrlapokhoz mint sémákhoz illeszkedjen, azok kitöltését megfelelően el tudja végezni. Itt kerül újra elő a 9.1. szakaszban sokat boncolgatott hordozhatósági kérdés.

Az NLIDB *alapú mélyhálókereső* (a továbbiakban NLIDW) tehát nem oldalakat kell, hogy szüreteljen és nyilvántartson, hanem a forrásoldalakon elérhető keresőfelületek, formanyomtatványok leírásait, illetve egy univerzális sémához való kapcsolatait, leképezéseit kell gyűjtenie, tárolnia. Az univerzális sémák használatát az egyszerűbb összefogás, a közös tématerülettel foglalkozó oldalak könnyebb csoportosítása teszi indokolttá. A megfeleltetést jelenleg még emberi erőforrással szokás megoldani — ezt mind a tartalomszolgáltató, mind a keresőmotor üzemeltetője megteheti.

A fenti megfontolások tükrében az NLIDW-rendszereknek az alábbi sémát érdemes követniük:

1. **Szintaktikai elemzés**. Ez megegyezik a 9.1. szakaszban tárgyalttal.
2. **Kérdésleképezés univerzális sémákra**. A szemantikai elemzéshez hasonlóan itt is egy formális résznyelvre képezzük le a szintaktikai elemzés után előállt mondatot, azonban a 9.1. szakasszal ellentétben ez a formális résznyelv sokkal korlátozottabb képességű (az űrlapok kitöltése ilyen): csak attribútumértékek megadását teszi lehetővé egy keresőkifejezésben.

   A kérdésleképezés során, a hordozhatóság növelése érdekében érdemes univerzális űrlapokat (sémákat) használni. Minden tématerületnek általában 10–15 univerzális sémája van, amelynek mentén tipikus felhasználói kérdéseket lehet feltenni. Az univerzális űrlapok száma a gyakorlatban azonban elsősorban attól függ, hogy a válaszkereső motort támogató oldalak milyen űrlapok kitöltését támogatják, hiszen végeredményben majd ezeket kell tudni megkérdezni. Egy tipikus forrásoldalon általában egy univerzális űrlap áll rendelkezésre a felhasználói keresések támogatására, ezért valójában elegendő minden tématerületre egyetlen egy részletes univerzális nyomtatvánnyal dolgozni.
3. **Célmeghatározás**. A kérdésleképezés során a potenciálisan szóba jövő űrlapokat kitöltöttük. Mivel feltehetjük, hogy a felhasználó által feltett kérdés-

ben szereplő elemek egyike sem elhagyható, csak azokat az űrlapokat tartjuk meg, amelyekhez a legtöbb, felhasználó által megadott támpontot felhasználtuk. Természetesen előfordulhat, hogy bizonyos elemeket mindenképpen el kell hagynunk. Az univerzális sémára illeszkedő kitöltött űrlapok alapján már meg tudjuk mondani, hogy mely forrásoldalak képesek potenciálisan megválaszolni a kérdést, hiszen csak azok adhatnak közelítő választ, amelyek az űrlapon megadott attribútumokat egyáltalán kereshetővé tették. Mivel az univerzális sémák és a forrásoldalak keresőfelületei közötti leképezés rendelkezésünkre áll, az ilyen oldalak meghatározása egyszerű.

4. **Metakeresés**. Ha rögzítettük a megkérdezendő oldalak körét, akkor az univerzális űrlapot az oldalak helyben használt űrlapjára át kell alakítanunk, majd a metakeresőknél látott módszer szerint továbbítani a megfelelő címre. Az eredmények értekelése, rendezése a 9.2.1. pontban látottal azonos módon mehet végbe.

Az NLIDW-megoldás sokkal rugalmasabb bármelyik más mélyhálókereső alternatívánál, ráadásul az ott tárgyalt problémákat, így pl. a releváns oldalak rangsorolását, a témafókusz korlátozását alapvetően megoldja. Előbbihez elegendő belátni, hogy az űrlapok (szemantikailag) helyes kitöltése lényegében garantálja, hogy ténylegesen csak olyan oldalakhoz juthassunk el, amelyek a kérdésre potenciálisan választ tudnak adni — de legalábbis a keresést lehetővé teszik. Az utóbbinak az NLIDW esetében csak az szab határt, hogy milyen univerzális sémákkal dolgozik a rendszer — ezeket viszont tetszőlegesen lehet bővíteni.

Az NLIDW a nemzetközi és a hazai szakirodalomban is egy új, kiforratlan terület. Az NLIDW eddigi tapasztalatairól a [188, 187] irodalmakból tájékozódhat a kedves Olvasó.

10. fejezet

Szövegbányász-szoftverek bemutatása

Ahogy azt könyvünkben bemutattuk, napjainkban egyre nagyobb piaci igény mutatkozik a szabadszöveges formátumban tárolt adatok feldolgozására is. Itt elegendő csak utalnunk néhány tipikus alkalmazásra, hogy megértsük a terület fontosságát: közvélemény-kutatási adatok automatikus kiértékelése; vevői panaszlevelek automatikus feldolgozása és osztályozása; weblapok automatikus feldolgozása; periratokban történő keresés egy adott esethez hasonló ügyek és ítéletek felkutatására. A példák is illusztrálják azoknak az alkalmazási területeknek a sokszínűségét, ahol igény van a szöveges adatforrásokra vonatkozó adatelemzési szolgáltatásokra.

A szövegbányászati szoftverek fejlesztésének két fő kiindulási pontja van: az adatbányászati és statisztikai elemző eszközök és az adatbázis-kezelők területe.

Az adatbányászati elemzők célja túlmutat az adatok megbízható, rendszerezett tárolásán és gyors visszakereshetőségén. Elsődleges céljuk a nemtriviális összefüggések feltárása nagy mennyiségű adathalmazon. Az adatbányászati szoftverek még nem terjedtek el a vállalati szférában olyan mértékben, mint az adatbázis-kezelők. A fejezet első két szakaszában az SPSS Clementine és a Statistica szoftvercsomagok szövegbányászati funkcióit tekintjük át.

Az adatbázis-kezelés elmélete és gyakorlata négy-öt évtizedes múltra tekint már vissza. Mára az adatbáziskezelő-rendszerek teljesen beépültek a vállalati információmenedzsment eszközeibe. Az adatbázis-kezelőkre építő szövegbányászati szoftverek a legtöbb esetben a szövegbányászat leghagyományosabb feladatára, a kulcsszó alapú keresésre fókuszálnak, és csak a legkifinomultabb megoldások tartalmaznak ennél bővebb szövegbányászati támogatást (osztályozás, csoportosítás). A 10.3–10.5. szakaszokban az adatbázis-kezelők szövegbányász kiegészítéseivel foglakozunk.

A magyar nyelv támogatásával kapcsolatban két szinten tapasztalhatunk nehézségeket az ismertetésre kerülő szoftvereknél. Az első kihívás a 35 betűs magyar karakterkészlet támogatása, melyet minden szoftver legalább egy karakterkódolás

esetén megold, de néha szükség lehet konverzióra. A második kérdés, hogy milyen szinten támogatják a magyar nyelvű dokumentumok feldolgozását. Egyelőre a bemutatott termékek egyike sem tartalmaz a magyar nyelvhez nyelvtechnológiai eszközöket, szótövezőt, szófajmeghatározót, morfológiai elemzőt, de még olyan egyszerűbb eszközök sem állnak rendelkezésre, mint a stopszavak vagy szinonimák szótára. Ez utóbbiak esetében lehetőség van a listák külső állományból való betöltésére, betölthető pl. a könyvünk honlapján található stopszólista

10.1. SPSS Clementine

Az SPSS cég *Clementine* terméke az egyik vezető adatbányászati eszköz különösen a piac ügyfélkapcsolat-kezelési (CRM) szegmensében.[1] A szoftver 11-es verziója 2007 elején jelent meg. A Clementine egyik legfontosabb tulajdonsága felhasználóbarát felépítése. A praktikus grafikus felületen hatékony támogatást kapunk a folyamatok lépéseinek rendszerezésére, az összetettebb algoritmusoknál is legfeljebb néhány paramétert szükséges (lehet) változtatnunk. Gyakorlati tapasztalatok alapján is állíthatjuk, hogy a Clementine grafikus interfésze ugrásszerűen megnöveli az adat- és szövegbányász szakemberek munkavégzésének sebességét.

A jó használhatóság jellemzi a szövegbányász modult is. A modul lelkét adó Textmining csomópont puritán volta már-már zavaró a gyakorlott szakember számára, de a különleges igényeket is jól kiszolgáló parancssoros paraméterezési lehetőségek alapján átlátjuk, hogy igen komplex „szív” dobog a csomópont belsejében. A Clementine egyszerűséget pártoló filozófiáját igazolja, hogy alig van olyan eset, amikor a csak kívülről elérhető paramétereken kell módosítani.

A szövegbányászati modul egy kisebb cég, a francia Lexiquest technológiája, amelyet az SPSS felvásárolt, de a fejlesztést továbbra is ugyanez a csapat végzi. A TextMining for Clementine 5.0 könyvünk kéziratának lezárásakor jelent meg. Fontosabb újdonságai a Textmining csomópontba integrált szótárkezelő, internetes források (RSS feedek) közvetlen beolvasását támogató WebFeed node, valamint a hatékony elemzést és az összefüggések vizuális felismerését segítő többfajta hálódiagram.

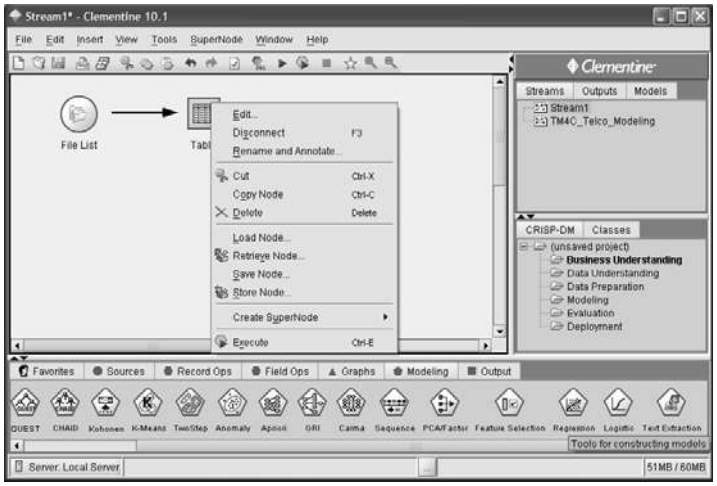
10.1.1. Kezelői felület, működés

Az adatok úgynevezett *feldolgozási folyam* (stream) mentén manipulálhatók, az egyes lépéseket *csomópont*ok (node) végzik, a sorrendet pedig az összekötő nyi-

[1] a Gartner Group 2007-es tanulmánya alapján

lak határozzák meg. Két szinten van lehetőség a csomópontok rendezésére. A szupercsomópontban több, gyakorlatilag egy logikai lépést megvalósító csomópont fedhető el, míg a 10.1. ábrán jobbra lent látható ablakban a CRISP-DM lépései szerint strukturálhatjuk az összetettebb feladat egyes lépéseihez létrehozott folyamokat. Az ábra jobb felső részén látható külön ablakban tárolja a program a megnyitott folyamok mellett az indítás óta keletkezett összes kimenetet, illetve a szövegbányászatban az átlagosnál többet használt modellek is itt találhatók.

Maguk a csomópontok az alsó sávban találhatóak, a funkcionális csoportokat megelőzően látható a kedvencek fül (Favorites), ami gyakorlatilag tartalmazza a folyamok kb. 80%-át adó elemeket. A tervezőpanelre drag and droppal helyezhetőek fel csomópontok. A csomópontra a jobb egérgombbal rákattintva szerkesztési, tárolási és futtatási menük jönnek elő — ez látható az ábrán is —, dupla kattintás esetén a szerkesztő ablak nyílik meg (Edit menü).



10.1. ábra. A Clementine kezelőfelülete

Még két hasznos funkciót tartunk fontosnak kiemelni. A Cashe, azaz csomópontra beállítható gyorstár nagyban gyorsítja az építkezést, ha a folyam vége fele kísérletezünk különböző alternatív megoldásokkal, ugyanis a gyorstár előtti rész ilyenkor nem hajtódik ismételten végre. A másik hasznos lehetőség az adatbázisokkal való integrálás: a Clementine az összes SQL parancsra lefordítható műveletet átadja az adatbázisnak, amelyek így az adat tárolási helyén az adott adatszerkezetre optimalizált műveletként fut le. Ezen túl képes az adatbázisszerverben megvalósított adatbányász-módszereket is használni.

10.1.2. Szöveges állományok kezelése

Mivel a szöveges állományok tárolása nagyon sokféle lehet, így a szövegbányász modulban a feldolgozandó dokumentumok kiválogatása, karakterhelyes beolvasása sem mindig egyszerű feladat.

A Clementine kétféle forrástípusból képes adatokat fogadni. A numerikus adatoknál megszokott adatmezős tárolás esetén az adatbázisból származó és a csv fájl esetén is egy-egy mezőben található a szöveg. Komplexebb feladat amikor különböző fájlformátumokból nyerjük ki a szövegeket. A szoftver képes kezelni doc, rtf, xls, ppt, txt, text, html, shtm, xml és pdf formátumú fájlokat. Minden karakterkódolás esetén karakterhelyesen beolvassa a szöveget, de ékezetes szövegnél érdemes az UTF-8 kódolás használni, mert egyébként a szó-dokumentum mátrix hibás lehet (10.1 verzió, TM 4.0).

A 10.1. ábra forráscsomópontja, a File List egy könyvtár (és opcionálisan) alkönyvtárainak feltérképezésére alkalmas. A feldolgozandó fájlokat csak kiterjesztés alapján lehet kiválogatni. A kimenet a feldolgozandó fájlok teljes elérési útvonalának listája, ez lesz a Text Extraction csomópont bemenete. Ez utóbbi lehetőséget nyújt egy fájlon belül több dokumentum értelmezésére azőket elválasztó szeparátorkarakter megadásával.

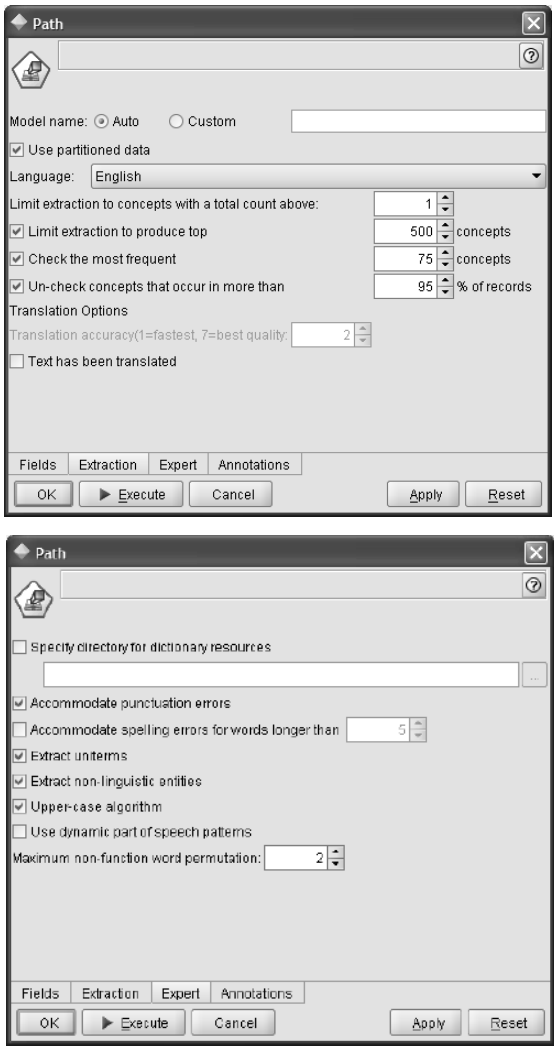
10.1.3. A korpusz szavainak feltérképezése

A szó-dokumentum mátrix építését a Clementine két lépésben végzi. Először a szótárt állítja elő, majd a második olvasással készül el a mátrix. Ennek legfőbb oka a teljes mátrix hatalmas mérete, ami miatt a Clementine alapesetben a teljes szótárnak csak egy töredékét ajánlja feldolgozásra.

A Text Mining for Clementine legkifinomultabb modulja a szótári elemek meghatározását végző modul. A szokásos szótár alapú szűréseken kívül a támogatott nyelvekre nyelvi feldolgozást (pl. szótövesítést) is végez, valamint meghatározza a korpuszban gyakran előforduló frázisokat, illetve többszavas kifejezéseket. Az utóbbi azért is fontos, mert így az egyszerű szó alapú szózsákmodellnél több tartalmi információ kerül kinyerésre.

A 10.2. ábrán látható a Text Extractor két legfontosabb ablaka. A nyelvbeállítással egy teljes szótárgyűjteményt aktivizálunk, de a magyar nyelv még nem szerepel a támogatott nyelvek között. A munkaigényes célszótáraknak más nyelveken való megvalósítását fordító modulokkal helyettesítik. Az alább megadott paraméterek szabályozzák a szótárkészítést. Az alsó ábrán látható lapon a speciális nyelvi beállítások adhatók meg.

- stopszólista,
- szótövezés (nyelv alapján),
- szinonimaszótár,
- szócsoportosítási jegyzék (hasonló a szinonimához, a leggyakoribb szóval helyettesítjük a csoport többi szavát),



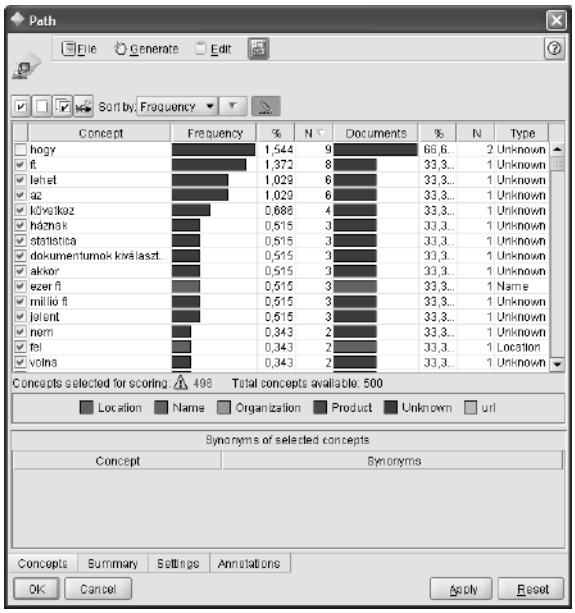
10.2. ábra. Text Extractor csomópont tulajdonságlapjai

- hibatűrő (fuzzy) illeszkedés (elírásokra),
- nyelvfüggő kifejezésfelismerés,
- XML-elemek feloldása.

10.1.4. Szavak szűrése, a szó-dokumentum mátrix létrehozása

Miután beállítottuk a szótárkészítés paramétereit, a Text Extractor modullal generáljuk a modellt, ami gyakorlatilag a vizsgált korpusz szótára. Ha ezt a modellt felhelyezzük a munkaterületre, akkor ennek segítségével készíthetjük el a szó-dokumentum mátrixot. A szótárba kerülő szavak kiválasztását a 10.3. ábra illusztrálja.

A túl gyakori és a küszöbértéknél ritkább szavak alapértelmezésben nincsenek bejelölve, de ezt a felhasználó felülbírálhatja. A gyakoriságokat jelölő oszlopok színe a szó típusától függ. Mivel nem áll rendelkezésre magyar szótár, így itt sok az ismeretlen típusú elem, de a megfelelő szótár alkalmazásával a szótípusra, ill. szófajra vonatkozó információt kapunk, ami igen hasznos az analízis fázisában.

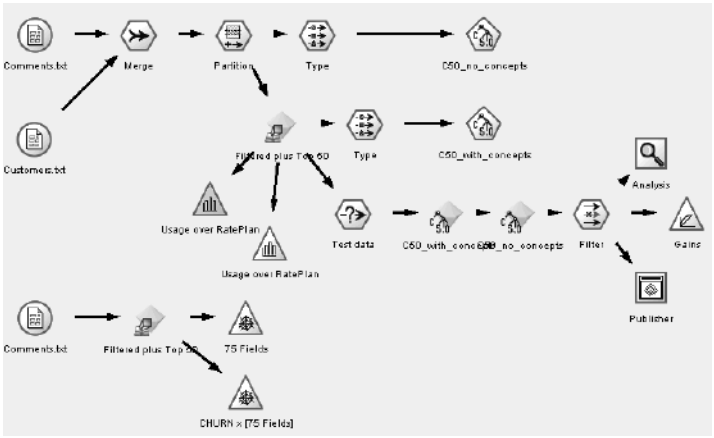


10.3. ábra. A szótár elemeinek kiválasztása

10.1.5. Analízis

A Clementine felépítéséből adódóan könnyen elemezhetőek az olyan adatstruktúrák is, ahol az entitásról nem csak szöveges, hanem kvantitatív információ is van (a gyakorlatban ez gyakran előfordul). A szó-dokumentum mátrix elkészítése után a feladat gyakorlatilag már elveszíti a szövegspecifikusságát.

A 10.4. ábrán látható elemzés egy telefontársaság ügyfélszolgálatára beérkezett üzenetekből és az előfizető egyéb adataiból kiindulva azt vizsgálja, hogy az ügyfelek milyen valószínűséggel hagyják el a szolgáltatót. A Text Extraction csomópont nem szerepel a munkalapon, csak a belőle származó Filtered plus Top 50, C5.0 modelleket generáló csomópontok.



10.4. ábra. Lemorzsolódás analízise telefonszolgáltatónál

10.2. Statistica Text Miner

A StatSoft cég *Statistica*[2] nevű terméke egyike a világ vezető általános statisztikai szoftvertermékeinek. A termék honlapja a `www.statsoft.hu`, ill. a `www.statsoft.com` címen érhető el. A Statistica szoftvercsomag elsődlegesen numerikus adatsorok, táblázatok adatbányászati feldolgozására készült[3], de a szö-

[2] A StatSoft dokumentációkban a szoftver nevét mindig nagybetűkkel és dőlten szedik, de ettől a formázási kiemeléstől könyvünkben eltekintünk.

[3] Az adatbányászati szegmensben a DM Review folyóirat 2006-os felmérése szerint a Statistica Data Miner nyújtja az egyik a legjobb szolgáltatást.

veges adatok elemzésének növekvő piaci igénye miatt ma már szövegbányászati támogatást is kínál.

10.2.1. A Text Miner modul áttekintése

A Statistica programcsomagban a szabadszöveges dokumentumok feldolgozását a *Text Miner* modul végzi. Ez a modul egy statisztikai előfeldolgozónak tekinthető, amelynek elsődleges célja a dokumentumok vektorreprezentációjának előállítása (ld. 2. fejezet). A vektortér dimenziói a dokumentumokban előforduló szavakat és kifejezéseket reprezentálják. Az elkészült dokumentumvektor már numerikus formában írja le a dokumentumot, és így elvégezhető már az osztályozás vagy a csoportosítás feladata. A Text Miner modul a következő nyelvi funkciókkal rendelkezik:

- szótövezés,
- szinonimák kezelése,
- stopszavak kezelése,
- vizsgálandó szavak kijelölése,
- kifejezések kezelése.

A szótövezést beépített szótárak segítségével végzi, amelyek jelenleg az angol, német, francia, olasz, spanyol, dán, holland, finn, norvég, portugál, orosz, svéd nyelveket támogatják. A program egyszerű szótövező algoritmust használ, nem végzi el például az igeidőt jelölő rag leválasztását.

A Text Miner modul használatának főbb lépései a következőkben foglalhatók össze. A rendszer főmenüjéből meghívható a Text mining dialógusablak, amelyben beállíthatók a szövegfeldolgozás paraméterei:

- a forrásdokumentumok elérése és formátuma,
- a dokumentumok nyelve,
- az elemzésbe bevont szavak száma,
- a szinonimák és a kifejezésszótár kijelölése,
- a szavak hosszára vonatkozó korlátok,
- a szeparáló jelek,
- a karakterkészlet,
- a stopszavak és bevont szavak,
- a tárolási adatbázis.

A paraméterek beállítása után elindítható a szövegfeldolgozás. Egyidejűleg több forrásdokumentum is feldolgozható. Ekkor a dokumentumgyűjtemény egy szokásos Statistica táblázatban is megadható, amely a dokumentumok elérési adatait és a leírására alkalmas többi adatmezőt tartalmazza.

A dokumentum feldolgozása magába foglalja a szavak indexelését és a szó–dokumentum-mátrix előfordulási értékekkel való feltöltését. Az eredménytáblázat tartalma:

- szó,
- szótő,
- szó relevanciája,
- tartalmazó dokumentumok száma (n_k).

A szó relevanciáját több módon is mérhetjük (a jelöléseket vö. a 2.2.3. ponttal):

- szóelőfordulás alapján, $d_{ki}^{(2)} = n_{ki}$,
- bináris értékkel, $d_{ki}^{(1)}$,
- logaritmikus súlyozással: $d_{ki}^{(3)} = 1 + \log(n_{ki})$,
- tf-idf súlyozással (a program a logaritmikus súlyozás alapján számolja): $1 + \log(n_{ki}) \cdot \log(\frac{N}{n_k})$.

Az eredménytáblázat már közvetlenül felhasználható statisztikai elemzésre, viszont rendszerint még elég sok dimenziót tartalmaz. A bonyolultabb elemzőalgoritmusok esetén ezért szükség lehet dimenzióredukciós lépésekre is, amelyhez a modulban az SVD alapú LSI áll rendelkezésre. Az eredmény egy szokásos táblázatba elmenthető, amely alapja lehet a további elemzéseknek.

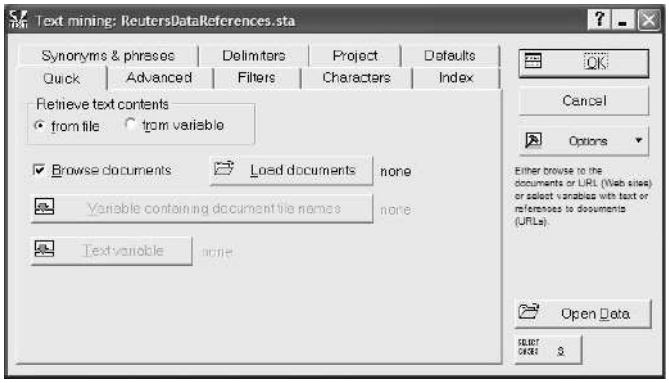
A Statistica rendszer kézikönyve szerint a Text Miner modul belső tárolási struktúrája sok kisméretű dokumentum esetére optimalizált. A rendszer tehát több, egyenként kisebb méretű szövegforrás statisztikai elemzésére használható fel hatékonyan. A fejezet elején (ld. a 237. oldalon) említett alkalmazási pédlák is mind ebbe a kategóriába esnek.

10.2.2. A Text Miner modul kezelőfelülete

A Text Miner modul főablaka a 10.5. ábrán látható. A főablak a főmenü Statistics menüpontján belül a Text & Document Mining, Web Crawling pont alatt érhető el.

A főablak egyes lapjai az alábbi szolgáltatásokat nyújtják:

- Quick és Advanced: forrásdokumentumok kijelölése, nyelv megadása;



10.5. ábra. A Text Miner modul főablaka

- Synonyms & phrases: szinonimaszótár és kifejezésszótár megadása;
- Index: stopszavak és vizsgált szavak meghatározása;
- Delimiters: szószeparáló jelek meghatározása;
- Filters: szavak vizsgált hosszának kijelölése;
- Characters: karakterkészlet meghatározása;
- Project: projekt beállítása.

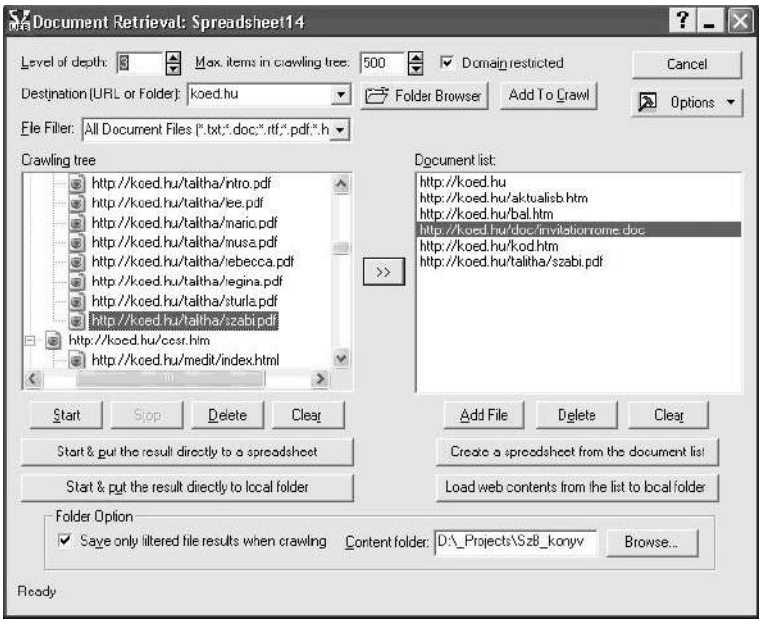
A rendszer a doc, rtf, txt, pdf, xml, html és a ps szöveges formátumokat közvetlenül támogatja. A főablak forrásdokumentum kijelölő részében meg kell adni a betöltendő dokumentumok elérési útvonalát. A vizsgálatoknál a forrásadatok más gépről, rendszerint egy webszerverről is származhatnak. Ekkor lehetőség van arra, hogy a Text Miner modul webrobotként begyűjtse a kért dokumentumokat. A begyűjtő robot paramétereit beállító ablak a 10.6. ábrán látható.

A dokumentumbegyűjtés fontosabb paraméterei:

- induló, gyökér URL,
- állománynévszűrő,
- letöltött dokumentumfa nagysága,
- mentési hely.

A letöltött dokumentumok elérési adatait fel lehet tölteni egy szabványos táblázatba, amely bemenő adata lehet az elemzés forráskijelölő almoduljának.

A rendszer a teljes dokumentumot egy egységnek tekinti, a dokumentumok formailag elkülönülő részeit (mint például a fejezet címe, a szerző adatai) nem kezeli



10.6. ábra. A webrobot paraméterbeállító ablaka

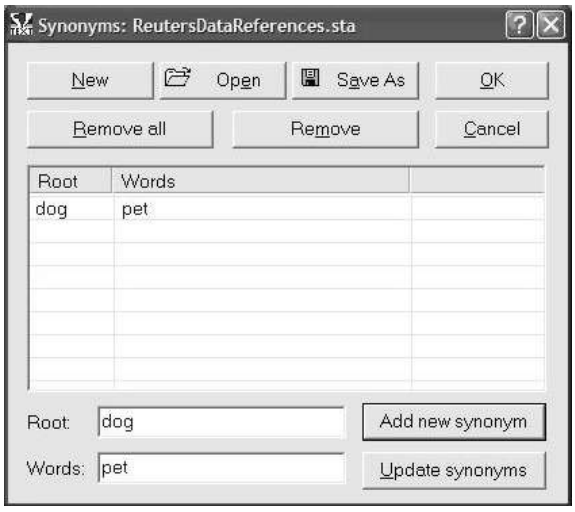
külön a rendszer. Magyar felhasználók részére az egyik alapvető kérdés mindig a magyar ékezetek helyes kezelése. Tesztjeink szerint a közép-európai kódolás kiválasztásával a beolvasás hibátlan volt, de ugyanez pl. nyugat-európai kódkészlet használatával is elérhető úgy, ha a Characters menüpontban a karakterkészletet kiegészítjük az összes magyar karakterrel.

A szinonimák megadása egyszerűen, a kapcsolódó szópárok vagy szó *n*-esek felsorolásával hajtható végre. A szópárok felvitelére szolgáló ablakot mutatja be a 10.7. ábra. Az első oszlop az alapszavakat tartalmazza, amelyek az indexben is szerepelni fognak. A második oszlop a kapcsolódó szinonimákat jelöli.

A stopszavakat egy egyszerű listában kell megadni. A listában szereplő szavak nem fognak megjelenni a generált indexben. A kifejezéseknél olyan több szóból álló sztringeket vihetünk be, amelyeket egy egységként szeretnénk kezelni. Ilyen kifejezés lehet pl. a *Magyar Köztársaság* kifejezés.

A dokumentum feldolgozása után feljön az eredményeket megjelenítő ablak, amelyet a 10.8. ábra mutat be. Az ablakon az alábbi funkciók érhetők el:

- Quick: eredmény megjelenítő táblázatok;



10.7. ábra. A szinonimák felviteli ablaka

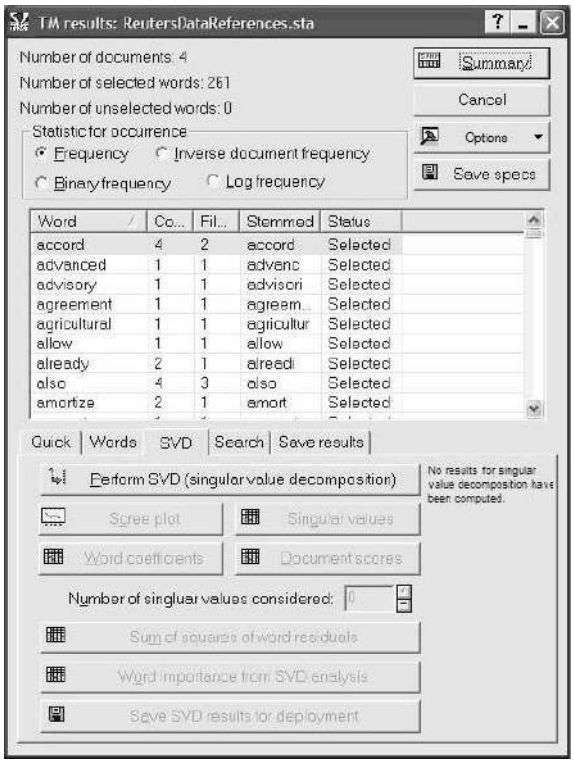
- Words: szavak kiemelése a vizsgálatból;
- SVD: az SVD alapú dimenzióredukciós eljárás elvégzése;
- Search: a keresési feltételt kielégítő dokumentumok listázása;
- Save result: az eredménymátrix mentése.

Az eredményablakban megjelenik a szavak gyakorisági listája és itt határozhatjuk meg, hogy milyen mérőszámot használunk a szavak fontosságának mérésére. A választható mérőszámok:

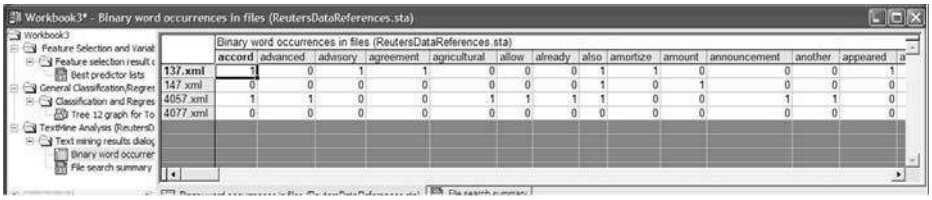
- Frequency: szóelőfordulás;
- Binary frequency: bináris;
- Inverse document frequency: tf-idf;
- Log frequency: logaritmikus számláló.

Az eredmény a szó-dokumentum mátrix lesz, amit a Summary feliratú gomb segítségével jeleníthetünk meg. Ez a táblázat az alapja a további statisztikai számításoknak. A táblázat megjelenítő ablakát a 10.9. ábra mutatja be.

A vektortér méretének csökkentése érdekében lehetőség van dimenzióredukcióra az SVD-módszer alkalmazásán keresztül. A kezelőpanel adott határok közt lehetőséget ad az új tér dimenziószámának szabályozására. A 10.8. ábra az SVD



10.8. ábra. Az eredménypanel ablaka



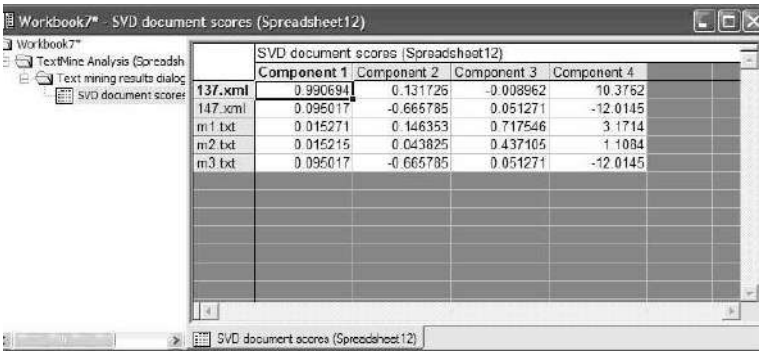
10.9. ábra. Az eredménymátrix ablaka

kezelő paneljét tartalmazza az ablak alsó részében (ld. a 2.3.2.2. alpontot). Az SVD-módszer futtatása után a következő adatokat kapjuk:

- Singular values: a sajátértékek listája,
- Screen plot: a sajátértékek relatív viszonyát mutató grafikon,

- Document scores: a redukált teret leíró mátrixot listázza ki (ld. 10.10. ábra),
- Word coefficients: a régi és új bázis közötti transzformációs mátrix,
- Word importance: az egyes szavak fontossága az új dimenziók kijelölésében.

Könyvünk honlapján egy mintapélda található a Statistica Text Miner moduljával történő dokumentumosztályozásra.



10.10. ábra. SVD eredmény mátrix

10.3. Oracle Text

10.3.1. Tipikus alkalmazások

Az *Oracle Text* az Oracle adatbázis-kezelő egyik komponense, amelynek célja a hatékony szövegkezelési funkciók biztosítása a szöveges formátumú adatelemeknél. Az Oracle Text ezen a néven az Oracle9i verzióban jelent meg először, de a modul gyökerei a Oracle8i verzióig nyúlnak vissza. Az Oracle Text-rendszert világszerte alkalmazzák. Az alábbiakban néhány prominens felhasználón keresztül bemutatjuk a rendszer jellemző alkalmazási területeit. További esettanulmányok és a kapcsolódó technológiai leírások elérhetők az Oracle cég honlapján.[4]

Az egyik ügyfélcég, az IronMountain számítógépes információfeldolgozást végez megrendelői részére. A feldolgozott anyagok zömében levelek és dokumentumok, méretük összességében mintegy 15 TB adatmennyiséget tesz ki. Ebből a hatalmas adatmennyiségből megközelítőleg 7 TB-ot tárolnak az Oracle Text-rendszerben. A szövegkezelő rendszert a dokumentumok indexelésére és hatékony keresésére használják.

[4] `www.oracle.com/technology/products/text/index.html`

A Motorola Printrak Biometrics Identification Solution terméke biometrikus adatok alapján működő azonosító rendszer, amely igen jelentős adathalmazra támaszkodik. A feldolgozott adatok ujjlenyomat-, hangminta- és aláírásadatok, melyeket XML-formátumban tárolnak. Az adatbázis több millió XML-állományt tartalmaz. Az Oracle Text-rendszert elsősorban a beérkező XML-dokumentumok automatikus osztályozására használják, de emellett a közelítő keresési feladatoknál is nagy hasznát veszik az Oracle Text szolgáltatásainak.

A rendszer felhasználói közé tartozik még a Világbank is, ahol az üzleti dokumentációk és a levelezés tárolására használják az Oracle Textet. A rendszer itt is milliós nagyságrendű dokumentumot tárol. Az Oracle Text feladata a szöveges formátumú iratok hatékony indexelése és a keresés gyors megvalósítása.

A következő neves felhasználó a CERN, ahol a projektek adminisztrálása során előálló dokumentumok kezelésére alkalmazzák az Oracle Textet. Az Oracle Text kiválasztása mellett szólt, hogy (1) támogatja a heterogén adatforrások kezelését, (2) integrálható az Oracle adatbázis-kezelővel, (3) támogatja a többnyelvűséget, (4) megvalósítható vele a szöveges adatok hatékony indexelése, ill. keresése.

A Der Spiegel német kiadónál működik a kiadó cikkeit tároló DIGAS elnevezésű digitális archívum, amely szintén az Oracle Texten alapszik. A rendszer adatbázisa mintegy 10 millió cikket tartalmaz. A szövegkezelő rendszert itt is a keresés, ill. indexelés hatékonysága miatt alkalmazzák.

10.3.2. A funkciók áttekintése

Ezek után nézzük meg részletesen, melyek a legfontosabb szövegfeldolgozási és -keresési funkciók az Oracle Textben:

- kulcsszó alapú keresés;
- szótő alapú keresés bizonyos nyelvekben;
- tezaurusz alapú keresés bizonyos nyelvekben;
- közelítő illesztésen alapuló keresés;
- illeszkedő részek rugalmas kiemelése;
- dokumentumszekció szerinti keresés;
- dokumentumok osztályozása;
- dokumentumok csoportosítása;
- dokumentumok kivonatolása;
- eredménymegjelenítési segédeszközök.

A kereséshez a vizsgált dokumentumokat egy dokumentumtáblába kell betölteni. A táblához egy speciális indexet használva hozzárendelhetjük a dokumentumgyűjteményben előforduló szavak listáját. A támogatott állományformátumok között megtalálható például az xml, a pdf, és a doc. A keresőkifejezésben alapvetően a keresett szavakat kell megadni, amelyekből logikai operátorok és speciális függvények segítségével komplex lekérdezéseket alakíthatunk ki. A lekérdezés eredményeképpen egy relevancia szerint rendezett dokumentumlistát kapunk. Bizonyos esetekben a kulcsszavak mellett struktúraspecifikus információk is megadhatók a keresési feltételben.

A dokumentumosztályozási feladatoknál az adatbázisban létre kell hozni egy szabályleíró táblát. Ez a tábla tartalmazza azon feltételeket, melyek segítségével eldönthető, hogy egy új dokumentum melyik osztályba kerüljön.

Az Oracle Text-rendszer parancsfelülete az Oracle SQL felületéhez illeszkedik. A lekérdezések végrehajtására az SQL nyelv SELECT parancsát lehet használni. Az ehhez szükséges háttérstruktúrák kezelésére az Oracle PL/SQL csomagokat biztosít, melyek a kezelőkörnyezet létrehozására és paraméterezésére szolgálnak. A konkrét példákat ld. pl. a 10.3.4. szakaszban.

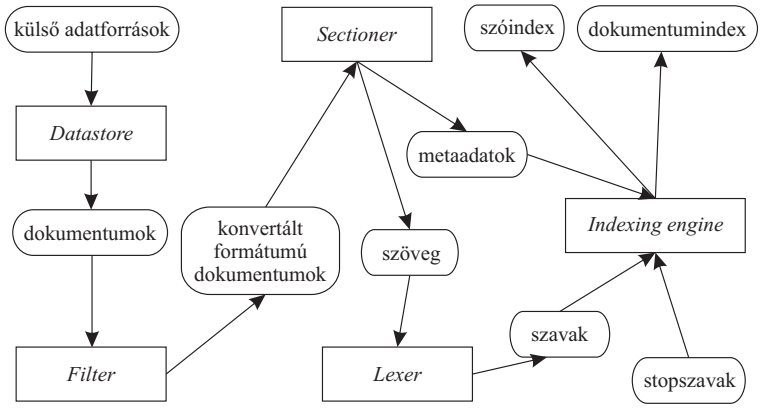
Az Oracle Text belső struktúrája több, egymástól jól elkülönülő modulra bontható. A rendszer főbb komponensei:

- Datastore: a nyers dokumentumok gyűjtő-, tárolóhelye;
- Filter: a nyers dokumentumok konvertálását végző modul, feladata a beolvasott állományok közös formára hozása;
- Sectioner: a dokumentumok belső struktúrájának feltárását végző egység;
- Lexer: a dokumentum szavainak feldolgozását végző modul;
- Indexing engine: a szavak és dokumentumok különböző típusú indexeit hozza létre;
- Dictionary: A modul segédszótárakat tartalmaz a szövegfeldolgozáshoz, ide tartozik többek között a stopszavak listája, ill. a tezaurusz.

A rendszer logikai felépítését a 10.11. ábra mutatja be. A komponensek működését a következő pontban ismertetjük.

10.3.3. Feldolgozási lépések

A szöveges adatok forrása lehet lokális külső állomány, távoli csomóponton elhelyezkedő állomány vagy adatbázismező. A feldolgozás első lépéseként a források adatai fizikailag vagy logikailag beolvasásra kerülnek. A forrásszövegek egy TEXT típusú adatbázis-objektumba kerülnek, amelynek tartalma lehet



10.11. ábra. Oracle Text belső struktúra

- szöveg;
- állományazonosító;
- URL-azonosító.

A feldolgozás következő lépésében a szöveg egy szűrőn halad keresztül (ld. a 10.11. ábrát). A szűrő végzi a szöveg formátumának, ill. kódolásának módosítását, amit egy indexelési paraméterben írhatunk elő. A szűrés eredménye egy saját belső formátumra konvertált dokumentum. Ez a formátum a dokumentumok tömörebb leírását teszi lehetővé. A következő lépésben a dokumentum struktúrájának feltárása történik. A szekcióhatárok és szekciójellemzők meghatározása után következik a szöveg tokenizálása. Az átalakítás során a szöveget előbb szavakra bontják, eltávolítva a figyelmen kívül hagyandó jeleket és szavakat (*stopszavak*), majd a megmaradt szavakat szabványos formátumra hozzák. Így például az

```
Aha! It's the 5:15 train, coming here now!
```

szövegből az átalakítás után

```
aha * * * 5 15 train coming * now
```

lesz, ahol a * szimbólum az elhagyott szavakat jelöli. A tokenizálás pontos menetét a Lexer paraméterezésével lehet szabályozni. Így például lehetőség van arra, hogy az *It's* alakot ne az *It is* alak, hanem egy összevont *Its* alak helyettesítse.

Az *indexelő motor* (Indexing Engine) az előző lépésben feltárt tokenekhez egy invertált indexet hoz létre, amiben minden tokenhez egy dokumentumlista tartozik.

A szöveges dokumentumot az adatbázisbeli táblákban többféle módon tárolhatjuk:

- Direct Datastore: a teljes dokumentum tárolása egy mezőben valósul meg;
- Multi Column Datastore: több testvérmezőben szétosztva valósul meg a tárolás;
- Detail Datastore: a dokumentum tárolása hierarchikus mezőkben történik;
- File Datastore: a dokumentum tárolása külső állományban valósul meg;
- URL Datastore: az interneten keresztül érhető el a dokumentum.

Az angol és francia nyelvre az Oracle Text-rendszer témaindexelési lehetőséget is tartalmaz. A témakapcsolati adatokat a rendszer egy tudásbázisban tárolja. Erre a két nyelvre a rendszer kész tudásbázist tartalmaz, míg más nyelvekhez a felhasználónak kell ezt létrehoznia. A Oracle Text indexelése támogatja a közelítő kereséseket is, így pl. a hasonló alakú (fuzzy matching), illetve azonos szótövű szavak keresését.

10.3.4. Az Oracle Text CONTEXT indexelési eljárása

A szavak gyors megkeresése megfelelő indexek alkalmazásával oldható meg. Az Oracle Textben a szükséges speciális indexkészlet a CTX_DDL csomag segítségével hozható létre:

```
EXEC CTX_DDL.CREATE_INDEX_SET('elnevezés')
```

Az Oracle Text-rendszer négyféle speciális indextípust támogat. Az indexek létrehozása, típustól függetlenül, az SQL-szabvány indexkezelő parancsára épül:

```
CREATE INDEX inév ON tábla(mező) INDEXTYPE típus;
```

A nagy méretű szövegek indexelésére a CONTEXT indextípus alkalmazható. A létrejövő indexstruktúra az alábbi komponensekből épül fel:

- A tokenek B-fája (ld. a 191. oldalt) a tartalmazó dokumentumok listájával együtt. A listában a dokumentumok belső kóddal szerepelnek, és a tokenek dokumentumon belüli pozícióit eltárolják.
- A szövegforrások tárolási helyét megadó értékek hozzárendelése a dokumentumok belső kódjaihoz.
- A dokumentumok belső kódjainak hozzárendelése a szövegforrás tárolási azonosítóihoz.
- A törölt dokumentumazonosítók nyilvántartása.

Az Oracle Text-lekérdezések alapvető sémája az, hogy megadunk egy keresőkifejezést, és a rendszer visszaadja az erre illeszkedő dokumentumokat. A lekérdezésnél a CONTAINS operátort kell használni. Az operátorhoz több paraméter is kapcsolódik, amelyek segítségével beállítható a keresés módja. A lekérdezés eredménydokumentumaihoz a rendszer illeszkedési mértéket társít (score), amelynek segítségével a találatok rangsorolhatók:

```
SELECT SCORE(1), title from news WHERE
CONTAINS(text,'ora',1) > 0 ORDER BY SCORE(1) DESC;
```

A CONTAINS függvény több különböző szóilleszkedési módot is támogat. Az operátorok listája:

- szó: pontos illeszkedés;
- szó1 AND szó2: együttes előfordulás;
- szó1 OR szó2: vagylagos előfordulás;
- NOT szó: a szó nem fordulhat elő;
- szó1, szó2: legalább az egyik szó forduljon elő;
- BT(szó): a megadott szó általánosításait keresi a tezaurusz alapján;
- NT(szó): a megadott szó specializációit keresi a tezaurusz alapján;
- REL(szó): kapcsolódó szavakat keres a tezauruszban;
- NEAR(szó1, szó2): a szavak egymáshoz közeli előfordulásait keresi;
- SYN(szó): a megadott szóval szinonim szavak keresése;
- !szó: hasonló hangzású szavak keresése;
- $szó: azonos szótővel rendelkező szavakat keres;
- ABOUT: a megadott témához kapcsolódó szavakat tartalmazó dokumentumok kikeresése;
- FUZZY(szó): hasonló alakú szavak keresése.

Például, az *étel* szótól egy távolságra eső specializációit kereső lekérdezés alakja:

```
SELECT szoveg from konyvek WHERE CONTAINS(szoveg, 'NT(étel,1)') > 0;
```

A szelekciós részben a minta mellett szerepelhet egy WITHIN tag is, amellyel a keresés a dokumentum egy adott szekciójára szűkíthető le.

10.3.5. További indextípusok

A rövidebb szövegek indexelésénél, amikor rendszerint nemszöveges mezők is szerepelnek a keresési feltételben, a CTXCAT indextípust kell használni. Ez az indextípus a CATSEARCH operátorhoz kapcsolódik. Ekkor egy B-fa jellegű indexstruktúra jön létre, amelyben a szöveges és hagyományos mezők vegyesen szerepelhetnek. Az indexek hierarchikus struktúrába szervezhetők. Ennél az indextípusnál a közelítő keresési módszerek nem alkalmazhatóak.

A dokumentumok osztályozásához a CTXRULE indextípus használható, amely az osztályozási szabályok halmazán értelmezett. A szabálytábla előállítható kézzel és automatikusan is, ez utóbbi esetben a CTX_CLS.TRAIN metódust kell meghívni. Az osztályozási lekérdezésekben a mintamondatra illeszkedő szabályok a MATCHES operátor segítségével kaphatók meg. A dokumentumok indexelésénél elvégezhető a stopszavak szűrése. Ezek listáját a felhasználó is szerkesztheti a CTX_DDL csomag segítségével. Ha a keresési kifejezésben egy stopszó szerepel, akkor arra a rendszer az összes szót illeszkedőnek tekinti.

Az Oracle Text a szavak közvetlen keresése mellett téma alapú keresést is biztosít. Ehhez a rendszer nyilvántartja a kulcsszavak hierarchikus rendszerét, amely rugalmasan bővíthető a dokumentumok feldolgozása során. A téma alapú lekérdezéshez az ABOUT függvényt kell a keresési operátor paramétereként megadni. A következő példában mindazon dokumentumokat visszakapjuk, amelyekben a *politics* témához kapcsolódó szavak előfordulnak:

```
SELECT SCORE(1), title FROM news WHERE
CONTAINS(text, 'about(politics)', 1) > 0 ORDER BY SCORE(1) DESC;
```

Az XML struktúrájú dokumentumokban a mintakeresést a CTXPATH indextípus segíti, felhasználása az EXISTNODE függvényen alapuló lekérdezéseknél jelent előnyt. Az indexelés több különböző karakterkódolást ismer, valamint egyes támogatott nyelvek esetén a szavak ragozott alakját is felismeri. Az Oracle Text az Oracle rendszer összes NLS karakterkódolásával tud dolgozni. Ez jelenleg mintegy 70 nyelvre terjed ki, köztük van a magyar, a japán és a kínai nyelv is.

10.3.6. Megjelenítési lehetőségek

A találatoknál lehetőség van arra, hogy a dokumentumon belül kiemelten jelenítsük meg a keresőkifejezésre illeszkedő részeket. A kiemelés az alábbi eljárásokkal valósítható meg, amelyek mindegyike a CTX_DOC csomag része:

- Markup: a megadott jelölőelemet a rendszer beteszi a kiemelt szó minden előfordulása elé és mögé.

- Highlight: ezzel az eljárással lehet lekérdezni az illeszkedő szavak pozícióit.
- Snippet: a kulcsszavakkal együtt az előfordulási szövegkörnyezetet is visszaadja;
- Themes: a szöveghez tartozó releváns témák, kulcsszavak kigyűjtését végzi. A mintapélda a tíz legfontosabb kulcsszót emeli ki:

```
declare
   the_themes ctx_doc.theme_tab;
begin
   ctx_doc.themes('myindex','1',the_themes, numthemes=>10);
   for i in 1..the_themes.count loop
     dbms_output.put_line(the_themes(i).theme||':'||the_themes(i).weight);
   end loop;
end;
```

- Gist: a dokumentumot legjobban jellemző szöveg előállítása.

10.3.7. A dokumentumok particionálása

Az Oracle Text háromféle particionálási módszert támogat a hasonló tartalmú dokumentumok feltárására:

- *Manuális osztályozás* esetén minden működési szabályt a felhasználó ad meg. A szabály a kulcsszavakat osztálykódokkal rendeli össze. A módszer előnye a kiszámítható működés, hátránya viszont, hogy a szabályok megalkotása időt rabló folyamat.
- *Automatikus osztályozás* esetén a felhasználónak elegendő csak egy tanítókörnyezetet létrehoznia (ld. az 5.3. szakaszt), amelynek alapján a rendszer megalkotja a teljes szabályrendszert. Ez lényegesen gyorsabb, mint a manuális osztályozás, viszont az osztályozás minősége a tanítókörnyezetet jóságától függ.
- *Csoportosítás* esetén a rendszer a dokumentumok hasonlósága alapján csoportokat képez.

A manuális osztályozás során előbb az alap adattáblákat hozzák létre. Ide tartoznak a dokumentumleíró, a szabályleíró és az eredményt tároló táblák. A második lépés a forrástáblák feltöltése. A szabálytáblában a kategóriák és kulcsszavaik összerendelését tároljuk le. Ezután következik a szabályok indexelése a CTXRULE indextípussal. Végül a dokumentumokat a kategóriákhoz rendeljük a kulcsszavak alapján. Ehhez sorba vesszük a dokumentumokat, és minden do-

kumentumhoz a MATCHES operátor segítségével megkerestetjük az illeszkedő kulcsszavak és kategóriák halmazát.

Az Oracle Text két osztályozási algoritmust ismer: döntési fák és az SVM módszerét. A tanítást a CTX_CLS csomag TRAIN metódusa segítségével lehet végrehajtatni. A döntési fák módszere esetén minden osztályhoz külön döntési fa készül. A tanítóhalmaz alapján a döntési fákhoz egy megbízhatósági (konfidencia) szintet rendel. A döntésifa módszerének előnye, hogy a kapott eredmények szemléletesek, könnyen értelmezhetők. A döntési fákon alapuló osztályozás alaplépései:

- a dokumentum, a kategória-, a hozzárendelési és az eredmény-adattáblák létrehozása;
- forrásadatok feltöltése a kategória- és a tanítóminta-táblákba;
- szabályok indexelése, osztályozási algoritmus és paramétereinek beállítása;
- az osztályozó betanítása, a döntési fák felépítése;
- a kapott szabálytábla indexelése;
- a dokumentumok kategorizálása az elkészült index alapján.

A csoportosítást a CTX_CLS csomag CLUSTERING eljárása végzi, amely a *k*-átlagot (ld. 6.4.1. szakasz) vagy egy hierarchikus csoportosító algoritmust valósít meg. Az utóbbi esetben eredményként a dokumentumok hierarchikus csoportosítását kapjuk, ahol az egyes dokumentumok csoporthoz való tartozási mértéke is lekérdezhető. A levéldokumentumok lesznek az elemi csoportok.

Az Oracle Text további érdekes lehetőségeiről bővebb ismertetés található a könyv honlapján, ahol néhány mintapéldán keresztül közvetlenebb tapasztalatokat is nyerhet az Olvasó.

10.4. Microsoft SqlServer szövegkezelő modulja

10.4.1. Áttekintés

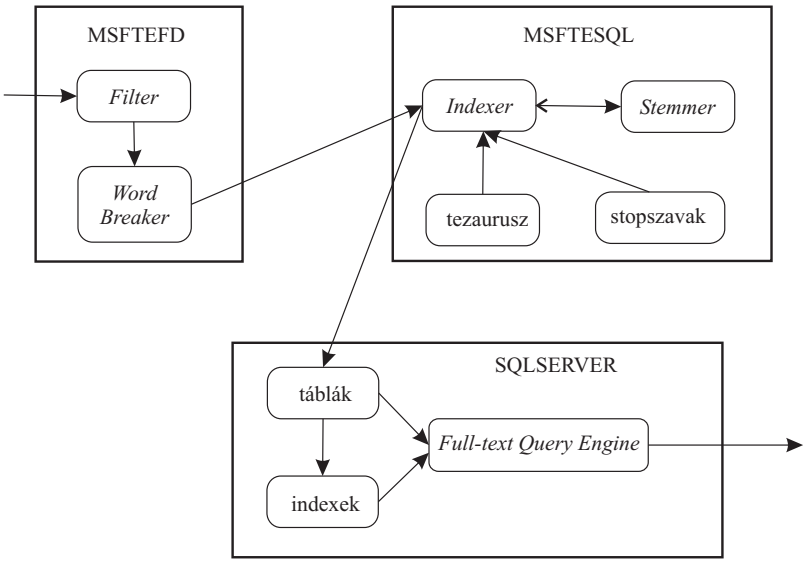
A *Microsoft* SQLSERVER adatbázis-kezelő rendszere az SQLSERVER 7.0 verziótól kezdve tartalmaz szövegkezelő modult. A modul célja, hogy a szöveges formátumú adatokon hatékony kulcsszavas lekérdezéseket lehessen végrehajtani, illetve lehetőség legyen a közelítő és nyelvi elemeket tartalmazó keresések végrehajtására is. Az SQLSERVER Fulltext lehetőségeiről részletes ismertetőt találhatunk a Microsoft kapcsolódó honlapján.[5]

[5] msdn2.microsoft.com/en-us/library/ms142571.aspx

AZ SQLSERVER szövegkezelő modulja struktúrájának egyik lényeges eleme, hogy a szövegek indexelését, és az index alapján való keresést nem valamelyik belső modul végzi, hanem külső programot hív meg ezen funkciók végrehajtására. Ez a program a *Microsoft Search Service*, amely maga tárolja és adminisztrálja a feldolgozott szövegeket. A keresőmotor az alábbi szolgáltatásokat nyújtja:

- indexelés;
- keresés, ezen belül
  - szóra vagy kifejezésre lehet keresni,
  - előfordulási pozícióra lehet szűrni,
  - nyelvi alapon történő szűrést (pl. szótő) lehet elvégezni.

A külső modulos megoldásból következik, hogy a szövegekhez készült indexek nem az adatbázisban találhatóak, hanem azokat a Search Service önálló külső állományban tárolja egy elkülönített katalógusban, amit indexkatalógusnak is szokás nevezni. A szövegkezelő rendszer átfogó struktúráját a 10.12. ábrán mutatjuk be.



10.12. ábra. SQL Full-Text Modul struktúrája

A szövegkezelő rendszer három fő modulja az

- SQLSERVER: a DBMS végrehajtó motor;
- MSFTESQL: a szövegfeldolgozó Fulltext Search motor, amely az SQLSERVER-en kívül más alkalmazásokat is kiszolgálhat (Microsoft Search Service);
- MSFTEFD: a dokumentumformátum konverzióját végző motor.

10.4.2. Feldolgozási lépések

Az SQLSERVER hatáskörében található a dokumentumhalmazt tároló adattábla. Az indexek karbantartására szolgáló indexkatalógust az SQLSERVER *Gatherer* modulja felügyeli és kezeli. Ez a modul felelős a feladatok ütemezéséért is. Az ugyancsak az SQLSERVER-ben található *Key map* modul a dokumentumok logikai és belső fizikai azonosítóinak összerendelését tartja nyilván. A *Query* modulok az SQL parancsok végrehajtásáért felelősek.

A *Filter* modul a különböző bináris formában tárolt dokumentumokat alakítja át szöveges alakra. A *Filter Daemon Manager* felügyeli az egyes *Filter* folyamatok szabályos lefutását. A konverzió során a *Word Breaker* modul állítja elő a szavakat, melyekhez egy belső kódot rendel. A szótövek meghatározása után, amit a *Stemmer* modul végez el, kapjuk meg a tokeneket. Az *Indexer* modul építi fel az egyes tokenekhez az invertált indexet. Ez tartalmazza az előfordulási helyeket, megadva a dokumentumazonosítót és a dokumentumon belüli pozíciót is. A felesleges stopszavak kiszűrése a *Noise (stop) words* lista alapján történik. A fogalmak közötti hasonlóságokat, általánosítási, specializációs vagy szinonimakapcsolatokat a *Thesaurus* modul tárolja. A szövegfeldolgozás specifikus operátorainak értelmezést a *Full-text Query Engine Processor* modul ad.

Az SQLSERVER jelenlegi változata 23 nyelvre ad támogatást, melyek között a magyar nyelv nem szerepel.

10.4.3. Indexelés

A dokumentumok tartalmának indexeléséhez a dokumentumokat egy tábla megadott mezőjébe kell összegyűjteni. A táblában minden rekord egy új dokumentumot jelent. A kulcsszavak szövegben való kereséséhez az SQLSERVER egy speciális indexstruktúrát alkalmaz. A szövegindex- (Fulltext Index) szerkezet eltér a hagyományos B-fa felépítéstől, mivel itt egy szóhoz tetszőleges sok dokumentumelőfordulás tartozhat. A szövegindex egy invertált lista, amelyben a kulcsszavakat tokenek reprezentálják, és minden tokenhez a tartalmazó dokumentumok azono-

sítóinak listája kapcsolódik. A listát a program tömörített formában tárolja. Az SQLSERVER az index létrehozásának háromféle módját támogatja:

- teljesindex-létrehozás: a teljes dokumentum halmaz indexálásra kerül;
- növekményesindex-létrehozás: csak a megváltozott tartalmú dokumentumokat indexeli;
- időbélyeg alapú indexlétrehozás: a dokumentumok változási időbélyege alapján indexel.

A rendszerben párhuzamosan több dokumentumgyűjtemény is létezhet, amelyek kapcsolódhatnak is egymáshoz. A logikailag összetartozó szövegindexek szövegindex-katalógust alkotnak. A szövegindex logikai felépítését az alábbi példával illusztrálhatjuk. Legyen a dokumentumhalmazt leíró tábla a következő:

DocID	szöveg
1	Péter iskolába megy
2	Anna alszik
3	Itt van újra Péter

A generált invertált index az alábbi fontosabb bejegyzéseket tartalmazza:

Kulcsszó	Table	DocID	Pos
Anna	1	2	1
Péter	1	1	1
Péter	1	3	4
megy	1	1	3

Az indextáblában a Table mező a dokumentumokat tároló adattáblát azonosítja, a DocID a dokumentumok azonosítókódját jelenti, míg a Pos mező a szónak a dokumentumon belüli pozícióját adja meg.

Mint már említettük, a program a dokumentumokat egy tárolótábla mezőiben helyezi el. Mivel a dokumentumok eredeti formátuma sokféle lehet, a mező típusaként egy semleges, mindent befogadó adattípust szokás választani. A leggyakoribb adattípus az IMAGE és a VARBINARY típus. A különböző formátumú dokumentumok feldolgozásához a bináris alakból szöveges (ASCII) alakra kell konvertálni. A szöveg formátumát a tárolótábla erre szolgáló típusleíró mezőjében tárolják. Ez alapján választja ki a beépített konverziós lehetőségek közül a megfelelőt az ASCII formára történő átalakítást végző Filter modul.

Az indexállomány mérete csökkenthető a stopszavak kiszűrésével. Ezek nyelvspecifikus listáját egy felhasználó által is módosítható szöveges állományból olvassa be a rendszer.

10.4.4. Kezelőfelület

A Fulltext indexek az SQLSERVER adatbázison kívül tárolódnak úgy, hogy egy táblához csak egy Fulltext-index hozható létre. Az indexek frissítésének módját az index létrehozásakor kell kijelölni.

A Fulltext Search modul használatának első lépése az indexkatalógus előállítása:

```
CREATE FULLTEXT CATALOG indexkatalógus
```

Az indexeléshez az adattáblában léteznie kell egy nem üres, egyedi értékű mezőnek. Ez az index azonosítja a dokumentumot tartalmazó rekordokat.

```
CREATE UNIQUE INDEX indexnév ON tábla (egyedi_mező);
```

Az index tényleges létrehozását az alábbi parancs végzi el:

```
CREATE FULLTEXT INDEX ON tábla (szövegmező
  TYPE COLUMN típusmező LANGUAGE nyelvkód )
  KEY INDEX indexnév ON indexkatalógus
  WITH CHANGE_TRACKING AUTO
```

A parancsban szövegmező jelöli a szöveges információt tartalmazó adattáblamezőt, a típusmezőben pedig a dokumentum formátumának kódja szerepel. A parancsban szereplő indexnév a példa előző lépésében létrehozott mező. Az indexkatalógus arra a Fulltext-katalógusra utal, amelyben helyet kap az így létrejövő index. A nyelvkód segítségével lehet opcionálisan megadni a szöveg nyelvét. A Fulltext alapú keresésnél a CONTAINS a legalapvetőbb művelet. Segítségével pontos és közelítő szókeresést, szótő-meghatározást, pozíció és szinonima alapú keresést lehet végrehajtani. Az operátor alakja:

```
CONTAINS (mező, feltétel, LANGUAGE nyelvkód)
```

A nyelvkód mező itt is opcionális. Az eredménybe azon rekordok kerülnek be, melyeknél a megadott mező teljesíti a második paraméterként szereplő illesztési feltételt. Ez lehet elemi, prefix, származtatási (ragozott és tezaurusz alapú), közelség alapú és súlyozási kifejezés. Az elemi kifejezés szavak listáját jelenti. Erre azok a mezők illeszkednek, amelyekben az összes szó pontosan a megadott alakban szerepel. Az illeszkedésvizsgálat során az írásjeleket a keresőmotor nem veszi figyelembe. A könyv honlapján néhány egyszerű példán keresztül bemutatjuk a keresési lehetőségeket.

A CONTAINS operátor rendszerint a SELECT parancs WHERE részében szerepel. A bináris visszatérési értéke alapján kerül be az aktuális rekord az eredményhalmazba. A Fulltext lekérdezésnek van egy másik alakja is, amikor eredményként a teljes illeszkedő találati halmazt megkapjuk. Ez az operátor olyan helyen fordulhat elő, ahol táblahivatkozás szerepelhet, mint például a FROM kifejezésnél. Alakja:

CONTAINSTABLE (tábla, mező, feltétel, LANGUAGE nyelvkód, méret)

A középső három paraméter jelentése megegyezik a CONTAINS operátornál ismertetett paraméterekkel. Az első paraméter azt a táblát jelöli ki, ahol a vizsgált szövegmező szerepel. Az ötödik opcionális argumentummal az eredményül kapott rekordok számára lehet felső korlátot szabni. A rekordok kiválasztása súlyérték alapú rangsorolás alapján történik. A súlyértékek a rank mezőbe kerülnek. Ahhoz, hogy a megjelenítési sorrend ezt a rangsorolást kövesse, az ORDER BY utasítást kell használni a SELECT parancs végén, ellenkező esetben a megjelenési sorrend különbözhet a súlyérték alapján kialakult sorrendtől. Az eredményül kapott tábla két speciális mezőt tartalmaz:

- key: a rekordon belüli szöveg Fulltext kulcsértéke;
- rank: a rekord súlyértéke.

A key mezőben lévő érték megegyezik a szövegmezőt tartalmazó rekord egyedi indexértékével. Ugyanezt az indexet kellett megadni a Fulltext-index létrehozásakor is. Egy illusztráló példa található a könyv honlapján.

A CONTAINS operátor működésének egy speciális esetét valósítja meg a FREETEXT operátor. Alakja:

FREETEXT (mező, feltétel, LANGUAGE nyelvkód)

A nyelvkód itt is opcionális. A feltétel részben csak elemi kifejezések lehetnek — ld. a CONTAINS operátornál leírtakat — értelmezése viszont másként valósul meg. A keresőmotor azon szövegeket is illeszkedőnek találja, amelyek a megadott szavaknak valamely szótövét vagy szinonimáját is tartalmazza. Vagyis a rendszer egy implicit nyelvtan alapú keresést hajt végre.

A logikai értékkel visszatérő FREETEXT operátoralak mellett itt is létezik egy eredménytáblát szolgáltató változat, melynek formátuma:

FREETEXTTABLE (tábla, mező, feltétel, LANGUAGE nyelv, méretkorlát)

Az operátor argumantumaira a CONTAINSTABLE esetén ismertetettek érvényesek. Használatára ismét a honlap mutat példát.

10.5. Egyéb adatbáziskezelő-rendszerek szövegbányászati elemei

10.5.1. mySQL Fulltext Search

A mySQL adatbáziskezelő-rendszer szintén rendelkezik szövegben való keresési lehetőségekkel, de a korábban bemutatott két adatbázis-kezelőéhez képest ez a lehetőség még viszonylag szűkös. Az illeszkedésvizsgálatnál az alábbi Fulltext Search [6] funkciók állnak rendelkezésre:

- elemi minta szerinti keresés;
- összetett minta szerinti keresés;
- kiterjesztett keresés.

Az illesztés során a stopszavakat nem veszi figyelembe a rendszer. Ezek közé tartoznak:

- az angol nyelvben gyakori szavak;
- a túl rövid szavak (4 karakternél rövidebbek);
- és a mintában túl gyakori szavak (legalább a dokumentumok 50%-ában előforduló szavak).

Az illesztés során a vizsgált elemekhez egy súlytényezőt rendel, amely azt mutatja, milyen mértékben illeszkedik az elem a megadott keresőkifejezésre. Az eredményeket az így kapott rangsor alapján rendezve jeleníti meg.

Az elemi mintaillesztéseknél a keresőkifejezést egy szólista alkotja. Az összetett keresések az alábbi operátorok segítségével adhatók meg:

- +: kötelező előfordulás;
- -: tiltás, a szó nem fordulhat elő;
- csak a szó: a szó opcionális;
- >: megnöveli a szó súlyát;
- <: csökkenti a szó súlyát;
- *: tetszőleges szórészlet helyettesítése;
- "': pontos illeszkedés.

A kifejezések megadásánál használhatjuk a zárójelet az operátorok precedenciasorrendjének felülírására. Így például a

```
+alma +(>fa <kert)
```

[6] dev.mysql.com/doc/refman/5.0/en/fulltext-search.html

kifejezésnek azon dokumentumok felelnek meg, amelyekben az alma szó mindenképpen, és emellett a fa vagy kert szó is szerepel, ahol a fa nagyobb prioritással rendelkezik, mint a kert.

A rendszer egyik érdekes lehetősége a kiterjesztett keresés. Ekkor a megadott keresőkifejezésre illeszkedő releváns szavakat felhasználva a rendszer megismétli a keresést, s az így kapott találati listát kapjuk vissza. A módszer előnye, hogy nem kell ismernünk a pontos szóelőfordulásokat a dokumentumokban, elég egy általános fogalmat megadni, legyen ez például az adatbázis. Ezzel az általános fogalommal elindítva a lekérdezést, az első fázisban előbb visszakapjuk a hozzá kapcsolódó és a dokumentumokban előforduló releváns szavakat, majd a keresés második fázisában az első fázisban kapott szavak is keresőszavak lesznek, így bővül az eredményhalmaz. Mint látható, a módszer a tezaurusz alapfunkcióit próbálja meg helyettesíteni. A módszer előnye, hogy nem igényli a tezaurusz előzetes felépítését, hátránya viszont, hogy a második körbe bevont szavak halmaza igen véletlenszerű, nagyban függ a vizsgált korpusztól.

10.5.2. DB2 Text Extender

Az IBM DB2 adatbáziskezelő-rendszeréhez is tartozik egy szabadszöveges keresőmodul. A DB2 termékhez több kiegészítő modul is elkészült, amelyből egyik a Text Extender [7] modul. A Text Extender is biztosítja a leggyakoribb szövegkeresési szolgáltatásokat. Ez magába foglalja a

- szó- és kifejezéskeresést;
- a dokumentumszekció alapján történő szűrést;
- az előfordulási közelség alapján való szűrést;
- a maszk használatát az illesztésnél;
- a mintához hasonló szavak keresését (fuzzy matching);
- a hasonló hangzású szavak keresését;
- a szinonimák keresését;
- a tezaurusz és a szótő alapú keresést.

A rendszerben háromféle indextípus értelmezett:

- *nyelvi index*: szótő alapú kereséshez használható;
- *normálindex*: a szavak pontos egyezőségének vizsgálatára szolgál;

[7] `www-306.ibm.com/software/data/db2/extenders/text/`

- *n-gramm alapú index*: a szavakat betűcsoportjaival azonosítja, a közelítő kereséseknél alkalmazható.

A nyelvi index természetesen csak bizonyos nyelveknél támogatott. A magyar itt sem szerepel ezek között. A szöveg indexelése során több előfeldolgozási lépés van. Először végrehajtódik a szöveg feldarabolása kisebb egységekre (mondat, szó). A rendszer elvégzi a szükséges karakterkonverziót a formátum egységesítése érdekében. Ezt követi a stopszavak eltávolítása és az összetett szavak felbontása (nyelvi index esetében). Nyelvi index esetében a feldolgozás része még a szótövek meghatározása is.

A DB2 is lehetővé teszi a rugalmas, testre szabott tezaurusz kialakítását. A szövegindexelési környezet kialakítása a DB2 esetén alapvetően az operációs rendszer parancsfelületén futó segédprogramokkal történik, míg a keresési szolgáltatások itt is az SQL nyelvi környezetbe ágyazva használhatók. A keresés beépített függvényeken keresztül valósul meg. A szóillesztéshez például a CONTAINS függvény használható, ahogy azt az alábbi példa is mutatja:

```
SELECT * FROM doksik WHERE
DB2TX.CONTAINS(szöveg,'"'car'"') = 1;
```

A példában szereplő szöveg azonosító ebben a környezetben nem mezőnév, hanem egy mezőt azonosító objektum (handler). A kapcsolódó mező tárolja a vizsgált szöveget. Ha a nyelvi, szótő alapú keresést kívánjuk használni, akkor a CONTAINS-ben megadott illesztési feltételnek a STEMMED hivatkozást kell használni, valamint meg kell adni a nyelvet is:

```
SELECT * FROM doksik WHERE DB2TX.CONTAINS(szoveg,
'STEMMED FORM OF GERMAN "'gefahren'") = 1;
```

A közelítő keresésnél a FUZZY kulcsszó szerepel. A hasonló hangzású szavakat a SOUNDS opcióval kereshetjük. A szinonimák keresésénél a SYNONYM kulcsszót kell alkalmazni:

```
SELECT * FROM doksik WHERE DB2TX.CONTAINS(szoveg,
'SYNONYM FORM OF "'gefahren'") = 1;
```

Mint látható, a DB2 rendszer Text Extender modulja is hasonló elvek alapján dolgozik, mint az Oracle Text, habár a parancsok szintaktikájában jelentős eltérések vannak.

10.5.3. Sybase Verity Full Text Search Engine

A Sybase adatbázis-kezelőbe a Verity cég által elkészített szövegkereső motor került beépítésre. A Sybase cég honlapján a rendszerről részletes tájékoztatást

kaphatunk.[8] A modul az adattáblamezőkben tárolt szöveges adatokra az alábbi speciális szövegkeresési szolgáltatásokat kínálja:

- eredménydokumentumok rangsorolása a keresőkifejezés szavainak előfordulási gyakorisága alapján;
- egymáshoz közel előforduló mintaszavak keresése;
- keresés leszűkítése megadott dokumentumszekcióra;
- szinonima alapú keresés;
- saját tezaurusz létrehozása;
- saját témakörök létrehozása;
- dokumentumok klaszterezése;
- adott dokumentumhoz hasonló dokumentumok keresése;
- különböző forrástípusok támogatása.

A szövegmezőkhöz elkészített indexek fizikailag az adatbázison kívül, szöveges állományokban helyezkednek el. Az adatbázisban a TEXT_DB adatbázis tárolja a szövegkereséshez szükséges metaadatokat. Mivel a szöveges keresés nem az adatbázis-kezelőben történik, hanem egy külön indexkezelő modulban, a lekérdezések megadásakor is tükröződik ez a kettősség. A lekérdezés alapvetően ugyanis a speciális indexállományra vonatkozik, és eredményül az illeszkedő dokumentumok kulcsértékeit kapjuk meg. Ha az alaptábla szövegértékeit szeretnénk elérni, akkor ezen eredménytáblát egy belső összekapcsolási művelet segítségével illeszteni kell az alaptáblához. A keresési feltétel megadása is sajátos módon történik. Az illesztési feltételt ugyanis a speciális index egyik pszeudomezőjéhez, az INDEX_ANY mezőhöz kell hozzárendelni. A pszeudómezők szolgálnak a feltétel megadása mellett a rangsor kijelölésére, az eredménylista elemszámának megadására, a relevanciaszűrés kijelölésére és az eredménydokumentumok csoportosítására is.

Az INDEX_ANY pszeudomezőnél megadott illesztési feltételben több speciális, az illesztés jellegére utaló kulcsszó szerepelhet:

- WORD: a szó pontos illesztése;
- PHRASE: kifejezés keresése;
- THESAURUS: a szó szinonimáinak keresése;
- STEM: szótő alapú nyelvi keresés;

[8] `manuals.sybase.com/onlinebooks/group-as/asg1250e/verity/`

- TOPIC: adott témához kapcsolódó szavak keresése;
- NEAR: előfordulási közelség szerinti szűrés;
- LIKE: adott dokumentumhoz hasonló dokumentumok keresése;
- IN: dokumentumszekcióra történő szűkítés;
- AND, OR: logikai operátorok az összetett keresésekhez.

A szintaktika bemutatására példánkban a football témához kapcsolódó dokumentumokat keressük. A lekérdezést itt is a SELECT SQL-paranccsal kell végrehajtani:

```
SELECT t2.szoveg FROM index_doksik t1, doksik t2 WHERE t1.id = t2.id AND
t1.INDEX_ANY = ' <TOPIC> (football)';
```

Mint látható, a Sybase/Verity Text Search Engine működése az SQLSERVER Full-Text moduljához áll a legközelebb.

Irodalomjegyzék

[1] L. Aas and L. Eikvil. Text categorisation: A survey. Raport NR 941, Norwegian Computing Center, 1999.

[2] Abonyi János, szerk. *Adatbányászat – a hatékonyság eszköze. Gyakorlati útmutató kezdőknek és haladóknak*. ComputerBooks, Budapest, 2005.

[3] Z. Alexin, J. Dombi, K. Fábricz, T. Gyimóthy, and T. Horváth. CONSTRUCTOR: A natural language interface based on attribute grammars. *Acta Cybernetica*, 9(3):247–255, 1990.

[4] J. Allen. *Natural Language Understanding*. The Benjaming/Cummings Publishing, Redwood City, USA, 2nd edition, 1995.

[5] H. Alshawi. *The Core Language Engine*. MIT Press, Cambridge, USA, 1992.

[6] H. Alshawi and J. van Eijck. Logical forms in the core language engine. In *Proc. of ACL-89, 27th Annual Meeting on Assoc. for Computational Linguistics*, pp. 25–32, Vancouver, Canada, 1989.

[7] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. Masque/SQL – an efficient and portable natural language query interface for relational databases. In *Proc. of IEA/AIE-93 Conf.*, pp. 327–330, Edinburgh, UK, 1993.

[8] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. Natural language interfaces to databases – an introduction. *J. of Natural Language Engineering*, 1(1):29–81, 1995.

[9] C. Apte, F. J. Damerau, and S. M. Weiss. Automated learning of decision rules for text categorization. *ACM Trans. Information Systems*, 12(3):233–251, 1994.

[10] S. E. Arnold. The Google Legacy, 2005.

[11] D. M. Ayuso. Discourse entities in janus. In *Proc. of ACL-89, 27th Annual Meeting on Assoc. for Computational Linguistics*, pp. 243–250, Vancouver, Canada, 1989.

[12] M. Bacchin, N. Ferro, and M. Melucci. University of Padua at CLEF 2002: Experiments to evaluate a statistical stemming algorithm. In *Working Notes for the CLEF 2002 Workshop*, Roma, Italy, 2002.

[13] Bach Iván. *Formális nyelvek*. TypoTEX, Budapest, 2. kiadás, 2002.

[14] L. D. Baker and A. K. McCallum. Distributional clustering of words for text classification. In *Proc. of SIGIR-98, 21st ACM Int. Conf. on Research and Development in Information Retrieval*, pp. 96–103, Melbourne, Australia, 1998.

[15] V. Balasubramanian. State of the art review on hypermedia issues and application. Report, E-Papyrus, 1993.

[16] B. W. Ballard, J. C. Lusth, and N. L. Tinkham. LDC-1: a transportable, knowledge-based natural language processor for office environments. *ACM Trans. on Information Systems*, 2(1):1–25, 1984.

[17] M. Bates, M. G. Moser, and D. Stallard. The IRUS transportable natural language database interface. In *Proc. of the 1st Int. Workshop on Expert Database Systems*, pp. 617–630, Kiawah Island, USA, 1984.

[18] F. Beil, M. Ester, and X. Xu. Frequent term-based text clustering. In *Proc. of SIGKDD-02, 8th Int. Conf. on Knowledge Discovery and Data Mining*, pp. 436–442, Edmonton, Canada, 2002.

[19] A. Berger, R. Caruana, D. Cohn, D. Freitag, and V. Mittal. Bridging the lexical chasm: Statistical approaches to answer-finding. In *Proc. of SIGIR-98, 23rd ACM Int. Conf. on Research and Development in Information Retrieval*, pp. 192–199, Athens, Greece, 2000.

[20] M. K. Bergman. The deep web: surfacing hidden value. *J. of Electronic Publishing*, 7(1), 2001.

[21] T. Berners-Lee. Information management: A proposal, 1989. CERN.

[22] M. W. Berry. *Survey of Text Mining*. Springer Verlag, New York, 2004.

[23] M. W. Berry, S. T. Dumais, and G. W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Review*, 37(4):573–595, 1995.

[24] G. L. Berry-Rogghe and H. Wulz. An overview of PLIDIS, a problem solving information system with german as query language. In *Natural Language Communication with Computers*, number 63 in Lecture Notes in Computer Science, pp. 87–132. Springer, London, UK, 1978.

[25] A. Blum. Learning boolean functions in an infinite attribute space. *Machine Learning*, 9(4):373–386, 1992.

[26] R. Blumberg and S. Arte. The problem with unstructured data. *DM Review*, February 2003.

[27] L. Bolc, K. Kochut, A. Leśniewski, and T. Strzałkowski. Natural language information retrieval system dialog. In *Proc. of ACL-83, 1st Conf. on European Chapter of the Assoc. for Computational Linguistics*, pp. 196–203, Pisa, Italy, 1983.

[28] K. Bontcheva, M. Dimitrov, D. Maynard, V. Tablan, and H. Cunningham. Shallow methods for named entity coreference resolution. In *Proc. of TALN-02, Chaînes de références et résolveurs d'anaphores*, Nancy, France, 2002.

[29] L. Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[30] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[31] P. D. Bruza and Th. P. van der Weide. Assessing the quality of hypertext views. *ACM SIGIR Forum*, 24(3):6–25, 1990.

[32] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[33] V. Bush. As we may think. *The Atlantic Monthly*, 176(1):101–108, 1945.

[34] W. B. Canvar and J. M. Trenkle. N-gram-based text categorization. In *Proc. of SDAIR-94, 3rd Annual Symp. on Document Analysis and Information Retrieval*, pp. 161–175, Las Vegas, USA, 1994.

[35] J. Carbonell and J. Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proc. of SIGIR-98, 21 st ACM Int. Conf. on Research and Development in Information Retrieval*, pp. 335–336, Melbourne, Australia, 1998.

[36] B. Carpenter. Phrasal queries with LingPipe and Lucene: ad hoc genomics text retrieval. *NIST Special Publication*, 2004.

[37] P. Cedermark. Swedish noun and adjective morphology in a natural language interface to databases. Master's thesis, Uppsala University, Department of Linguistics, 2003.

[38] S. Chakrabarti, B. Dom, R. Agrawal, and P. Raghavan. Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies. *The VLDB Journal*, 7(3):163–178, 1998.

[39] N. A. Chinchor. MUC-7 named entity task definition. In *Proc. of MUC-7, 7th Message Understanding Conference*, 1998.

[40] W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. *Proc. of IJCAI-03, Int. Joint Conf. on Artificial Intelligence, Workshop on Information Integration on the Web*, pp. 73–78, 2003.

[41] W. W. Cohen and Y. Singer. Context sensitive learning methods for text categorization. *ACM Trans. Inform. Syst.*, 17(2):141–173, 1999.

[42] A. Copestake and K. Sparck-Jones. Natural language interfaces to databases. *Knowledge Engineering Review*, 5(4):225–249, 1990.

[43] F. Crestani and S. Wu. Testing the cluster hypothesis in distributed information retrieval. *Inf. Processing & Management*, 42(5):1137–1150, 2006.

[44] D. Crystal. *A nyelv enciklopédiája*. Osiris, Budapest, 2003.

[45] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A framework and graphical development environment for robust NLP tools and applications. In *Proc. of ACL-02, 40th Annual Meeting of the Assoc. for Computational Linguistics*, pp. 168–175, Philadelphia, USA, 2002.

[46] D. R. Cutting, J. O. Pedersen, D. Karger, and J. W. Tukey. Scatter/Gather: A cluster-based approach to browsing large document collections. In *Proc. of SIGIR-92, 15th ACM Int. Conf. on Research and Development in Information Retrieval*, pp. 318–329, Copenhagen, Denmark, 1992.

[47] Csendes Dóra, Hatvani Csaba, Alexin Zoltán, Csirik János, Gyimóthy Tibor, Prószéky Gábor és Váradi Tamás. Kézzel annotált magyar nyelvi korpusz: a Szeged Korpusz. In *I. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY-03)*, pp. 238–245, Szeged, 2003.

[48] I. Dagan, Y. Karov, and D. Roth. Mistake-driven learning in text categorization. In *Proc. of EMNLP-97, 2nd Conf. on Empirical Methods in Natural Language Processing*, pp. 55–63, Providence, USA, 1997.

[49] S. D'Alessio, K. Murray, R. Schiaffino, and A. Kershenbaum. The effect of using hierarchical classifiers in text categorization. In *Proc. of RIAO-00, 6th Int. Conf. Recherche d'Information Assistee par Ordinateur*, pp. 302–313, Paris, France, 2000.

[50] J. L. Dawson. Suffix removal for word conflation. *Bulletin of the Assoc. for Literary & Linguistic Computing*, 2(3):33–46, 1974.

[51] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *J. of the Am. Soc. for Information Science*, 41(6):391–407, 1990.

[52] H. Deitel, P. Deitel, T. Nieto, T. Lin, and P. Sadhau. *XML: How to program*. Prentice-Hall, 2001.

[53] P. Domingos and M. J. Pazzani. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, 29(2–3):103–130, 1997.

[54] S. T. Dumais. Improving the retrieval information from external sources. *Behaviour Research Methods, Instruments and Computers*, 23(2):229–236, 1991.

[55] S. T. Dumais, J. Platt, D. Heckerman, and M. Sahami. Inductive learning algorithms and representations for text categorization. In *Proc. of CIKM-98, 7th ACM Int. Conf. on Information and Knowledge Management*, pp. 148–155, Bethesda, USA, 1998.

[56] H. P. Edmundson. New methods in automatic extracting. *J. of the ACM*, 16(2):264–285, 1969.

[57] H. P. Edmundson and R. E. Wyllys. Automatic abstracting and indexing-survey and recommendations. *Communications of the ACM*, 4(5):226–234, 1961.

[58] J. Edwards, K. McCurley, and J. Tomlin. An adaptive model for optimizing performance of an incremental web crawler. In *Proc. of WWW-01, 10th Int. Conf. on World Wide Web*, pp. 106–113, Hong Kong, 2001.

[59] J. Eisenstein and R. Davis. Gesture improves coreference resolution. In *Proc. of HLT-NAACL, Human Language Technology Conf. of the NAACL*, pp. 37–40, New York, USA, 2006.

[60] B. Endres-Niggemeyer. Human-style WWW summarization. Technical report, University for Applied Sciences, Department of Information and Communication, 2000.

[61] K. Fábricz, Z. Alexin, T. Gyimóthy, and T. Horváth. THALES: a software package for plane geometry constructions with a natural language interface. In *Proc. of COLING-90, 13th Int. Conf. on Computational Linguistics*, pp. 44–46, Helsinki, Finland, 1990.

[62] C. J. Fall, A. Törcsvári, K. Benzineb, and G. Karetka. Automated categorization in the international patent classification. *ACM SIGIR Forum Archive*, 37(1):10–25, 2003.

[63] C. J. Fall, A. Törcsvári, P. Fievét, and G. Karetka. Additional readme information for WIPO-de autocategorization data set, March 2003.

[64] C. J. Fall, A. Törcsvári, and G. Karetka. Readme information for WIPO-alpha autocategorization training set, December 2002.

[65] W. Fan, L. Wallace, S. Rich, and Z. Zhang. Tapping the power of text mining. *Communications of the ACM*, 49(9):76–82, 2006.

[66] Farkas Richárd és Szarvas György. Nyelvfüggetlen tulajdonnév felismerő rendszer, és alkalmazása különböző domainekre. In *IV. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY-03)*, pp. 22–31, Szeged, 2006.

[67] P. P. Filipe and N. J. Mamede. Databases and natural language interfaces. In *Proc. of JISBD-00, 5th Jornada de Ingeniería de Software y Bases de Dados*, pp. 321–332, Valladolid, Spain, 2000.

[68] W. B. Frakes and C. J. Fox. Strength and similarity of affix removal stemming algorithms. *ACM SIGIR Forum*, 37(1):26–30, 2003.

[69] A. Franz and B. Milch. Searching the web by voice. In *Proc. of COLING-02, 19th Int. Conf. on Computational Linguistics*, pp. 1213–1217, Taipei, Taiwan, 2002.

[70] Füstös László, Meszéna György és Simonné Mosolygó Nóra. *A sokváltozós adatelemzés statisztikai módszerei*. Akadémiai Kiadó, Budapest, 1986.

[71] N. Fuhr. Probabilistic models in information retrieval. *The Computer Journal*, 35(3):243–255, 1992.

[72] B. Galitsky. *Natural Language Question Answering System*. Advanced Knowledge International, Adelaide, Australia, 2003.

[73] M. K. Ganapathiraju. Relevance of cluster size in MMR based summarizer. Technical Report 11-742, Language Technologies Institute, Carnegie Mellon University, Pittsburgh, USA, 2002.

[74] J. M. Gawron, J. King, J. Lamping, E. Loebner, E. A. Paulson, G. K. Pullum, I. A. Sag, and T. Wasow. Processing english with a generalized phrase structure grammar. In *Proc. of ACL-82, 20th Conf. on Assoc. for Computational Linguistics*, pp. 74–81, Toronto, Canada, 1982.

[75] C. F. Goldfarb. *The SGML Handbook*. Oxford University Press, 1990.

[76] J. Goldstein, M. Kantrowitz, V. Mittal, and J. Carbonell. Summarizing text documents: Sentence selection and evaluation metrics. In *Proc. of SIGIR-99, 22nd ACM Int. Conf. on Research and Development in Information Retrieval*, pp. 121–128, Berkeley, USA, 1999.

[77] J. Goldstein, V. Mittal, and J. Carbonell. Creating and evaluating multi-document sentence extract summaries. In *Proc. of CIKM-00, 9th Int. Conf. on Information Knowledge Management*, pp. 165–172, McLean, USA, 2000.

[78] Y. Gong and X. Liu. Generic text summarization using relevance measure and latent semantic analysis. In *Proc. of SIGIR-01, 24th ACM Int. Conf. on Research and Development in Information Retrieval*, pp. 19–25, New Orleans, USA, 2001.

[79] G. Grefenstette and P. Tapanainen. What is a word, what is a sentence? Problems of tokenization. In *Proc. of COMPLEX-94, 3rd Int. Conf. on Computational Lexicography*, pp. 79–87, Budapest, Hungary, 1994.

[80] R. Grisham. Information extraction. In *The Oxford Handbook of Computational Linguistics*, pp. 545–559. Oxford University Press, 2004.

[81] F. Halasz and M. Schwartz. The Dexter hypertext reference model. *Communications of the ACM*, 37(2):30–39, 1994.

[82] P. Halácsy. Benefits of deep NLP-based lemmatization for information retrieval. In *Working Notes for the CLEF 2006 Workshop*, Alicante, Spain, 2006.

[83] P. Halácsy, A. Kornai, L. Németh, A. Rung, I. Szakadát, and V. Trón. Creating open language resources for Hungarian. In *Proc. of LREC-04, 4th Int. Conf. on Language Resources and Evaluation*, pp. 203–210, Lisbon, Portugal, 2004.

[84] E.-H. Han, D. Boley, M. Gini, R. Gross, K. Hastings, G. Karypis, V. Kumar, B. Mobasher, and J. Moore. WebAce: A web agent for document categorization and exploration. In *Proc. of Agents-98, 2nd Int. Conf. on Autonomous Agents*, pp. 408–415, Minneapolis, USA, 1998.

[85] G. G. Hendrix, E. D. Sacerdoti, D. Sagalowicz, and J. Slocum. Developing a natural language interface to complex data. *ACM Trans. on Database Systems*, 3(2):105–147, 1978.

[86] R. L. Herschman, R. T. Kelly, and H. G. Miller. User performance with a natural language query system for command control. Technical Report NPRDC-TR-797, Navy Personnel Research and Development Center, San Diego, USA, 1979.

[87] W. Hersh, C. Buckley, T. Leone, and D. Hickman. OHSUMED: an interactive retrieval evaluation and new large text collection for research. In *Proc. of SIGIR-94, 17th ACM Int. Conf. on Research and Development in Information Retrieval*, pp. 192–201, Dublin, Ireland, 1994.

[88] W. Howe. A brief history of the internet, 2005.

[89] D. A. Hull. Improving text retrieval for the routing problem using latent semantic indexing. In *Proc. of SIGIR-94, 17th ACM Int. Conf. on Research and Development in Information Retrieval*, pp. 282–289, Dublin, Ireland, 1994.

[90] H. Jing and K. R. McKeown. The decomposition of human-written summary sentences. In *Proc. of SIGIR-99, 22nd ACM Int. Conf. on Research and Development in Information Retrieval*, pp. 129–136, Berkeley, USA, 1999.

[91] T. Joachims. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In *Proc. of ICML-97, 14th Int. Conf. on Machine Learning*, pp. 143–151, Nashville, USA, 1997.

[92] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. Technical Report, University of Dortmund, Dept. of Informatics, Dortmund, Germany, 1997.

[93] H.-M. Jung and G. G.-B. Lee. Multilingual question answering with high portability on relational databases. In *Proc. of COLING-02, Workshop on Multilingual Summarization and Question Answering*, pp. 1–8, Taipei, Taiwan, 2002.

[94] Zs. T. Kardkovács, D. Tikk, and Z. Bánsághi. The Ferrety algorithm for the KDD cup 2005 problem. *ACM SIGKDD Explorations Newsletter*, 7(2):111–116, 2005.

[95] B. Katz. *Using English for Indexing and Retrieving*, volume 1 of *Artificial Intelligence at MIT: Expanding Frontiers*, pp. 134–165. MIT Press, 1990.

[96] B. Katz, S. Felshin, D. Yuret, A. Ibrahim, J. J. Lin, G. Marton, A. J. McFarland, and B. Temelkuran. Omnibase: A uniform access to heterogeneous data for question

answering. In *Proc. of NLDB-02*, volume 2553 of *Lecture Notes in Computer Science*, pp. 230–234. Springer, Stockholm, Sweden, 2002.

[97] B. Katz and J.L. Lin. Start and beyond. In *Proc. of SCI 2002*, volume XVI of *World Multiconference on Systemics, Cybernetics and Informatics*, 2002.

[98] J. Kim, T. Ohta, Y. Tsuruoka, Y. Tateisi, and N. Collier. Introduction to the bio-entity recognition task at JNLPBA. In *Proc. of JNLPBA, Int. Joint Workshop on Natural Language Processing in Biomedicine and its Applications*, pp. 70–75, Geneva, Switzerland, 2004.

[99] D. Koller and M. Sahami. Hierarchically classifying documents using a very few words. In *Proc. of ICML-97, 14th Int. Conf. on Machine Learning*, pp. 170–178, Nashville, USA, 1997.

[100] Korcsmáros István. Szövegbányászat (text mining) — új fogalom az üzleti intelligencia témakörében, 2003.

[101] Kornai András, Rebrus Péter, Vajda Péter, Halácsy Péter, Rung András és Trón Viktor. Általános célú morfológiai elemző kimeneti formalizmusa. In *II. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY-04)*, pp. 172–176, Szeged, 2004.

[102] M. Koskela, J. Laaksonen, and E. Oja. Entropy-based measures for clustering and SOM topology preservation applied to content-based image indexing and retrieval. In *Proc. of ICPR-04, 17th Int. Conf. on Pattern Recognition*, volume 2, pp. 1005–1008, 2004.

[103] Kovács László. *Adatbázisok tervezésének és kezelésének módszertana*. Computerbooks, 2004.

[104] M. Krier and F. Zaccà. Automatic categorization applications at the European Patent Office. *World Patent Information*, 24:187–196, 2002.

[105] R. Krovetz. Viewing morphology as an inference process. In *Proc. of SIGIR-93, 16th ACM Int. Conf. on Research and Development in Information Retrieval*, pp. 191–203, Pittsburgh, USA, 1993.

[106] D. Küpper, M. Strobel, and D. Rösner. NAUDA: A cooperative natural language interface to relational databases. *ACM SIGMOD Records*, 22(2):529–533, 1993.

[107] D. Kuropka. *Modelle zur Repräsentation natürlichsprachlicher Dokumente. Ontologie-basiertes Information-Filtering und -Retrieval mit relationalen Datenbanken*. Logos Verlag, Berlin, 2004.

[108] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. of ICML-01, 18th Int. Conf. on Machine Learning*, pp. 282–289, Williamstown, USA, 2001.

[109] W. Lam and C. Y. Ho. Using a generalized instance set for automatic text categorization. In *Proc. of SIGIR-98, 21st ACM Int. Conf. on Research and Development in Information Retrieval*, pp. 81–89, Melbourne, Australia, 1998.

[110] T. K. Landauer and S. T. Dumais. A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction and representation of knowledge. *Psychological Review*, 104:211–240, 1997.

[111] K. Lang. Newsweeder: learning to filter netnews. In *Proc. of ICML-95, 12th Int. Conf. on Machine Learning*, pp. 331–339, Lake Tahoe, USA, 1995.

[112] L. S. Larkey. A patent search and classification system. In *Proc. of DL-99, 4th ACM Conf. on Digital Libraries*, pp. 179–187, Berkeley, USA, 1999.

[113] H.-D. Lee and J. C. Park. Interpretation of natural language queries for relational database access with combinatory categorial grammar. *Int. J. of Computer Processing of Oriental Languages*, 15(3):281–303, 2002.

[114] Lejtovicz Katalin és Kardkovács Zsolt T. Anaforafeloldás magyar nyelvű szövegekben. In *IV. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY-06)*, pp. 362–363, Szeged, 2006.

[115] D. D. Lewis. An evaluation of phrasal and clustered representations on a text categorization task. In *Proc. of SIGIR-92, 15th ACM Int. Conf. on Research and Development in Information Retrieval*, pp. 37–50, Copenhagen, Denmark, 1992.

[116] D. D. Lewis. Naive (Bayes) at forty: The independence assumption in information retrieval. In *Proc. of ECML-98, 10th European Conference on Machine Learning*, pp. 4–15, Chemnitz, Germany, 1998.

[117] D. D. Lewis, Y. Yang, T. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *J. of Machine Learning Research*, 5:361–397, 2004.

[118] Y. H. Li and A. K. Jain. Classification of text documents. *The Computer Journal*, 41(8):537–546, 1998.

[119] R. Liere and P. Tadepalli. Active learning with committees for text categorization. In *Proc. of AAAI-97, 14th Conf. of the Am. Assoc. for Artificial Intelligence*, pp. 591–596, Providence, USA, 1997.

[120] N. Littlestone. Learning quickly when irrelevant attributes around: A new linear-threshold algorithm. *Machine Learning*, pp. 285–318, 1988.

[121] N. Littlestone. Comparing several linear-threshold learning algorithm on tasks involving superfluous attributes. In *Proc. of ICML-95, 12th Int. Conf. on Machine Learning*, pp. 353–361, Tahoe City, USA, 1995.

[122] P. C. Lockemann and F. B. Thompson. REL: a rapidly extensible language system. In *Proc. of Conf. on Computational Linguistics*, pp. 1–32, Sång-Säby, Sweden, 1969.

[123] J. B. Lovins. Development of a stemming algorithm. *Mechanical Translation and Computation Linguistics*, 11(1):23–31, 1968.

[124] I. Mani. *Automatic Summarization*. John Benjamin's Publishing Company, 2001.

[125] Ch. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2007.

[126] Ch. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, 1999.

[127] A. McCallum. MALLET: A machine learning for language toolkit, 2002.

[128] A. Mccallum. Information extraction: Distilling structured data from unstructured text. *ACM Queue*, 3(9), 2005.

[129] A. McCallum, D. Freitag, and F. C. N. Pereira. Maximum entropy Markov models for information extraction and segmentation. In *Proc. of ICML-00, 17th Int. Conf. on Machine Learning*, pp. 591–598, Stanford, USA, 2000.

[130] A. McCallum, R. Rosenfeld, T. Mitchell, and A. Ng. Improving text classification by shrinkage in a hierarchy of classes. In *Proc. of ICML-98, 15th Int. Conf. on Machine Learning*, pp. 359–367, Madison, USA, 1998.

[131] C. T. Meadow, B. R. Boyce, and D. H. Kraft. *Text Information Retrieval Systems*. Academic Press, Orlando, 2nd edition, 2000.

[132] X.-F. Meng and S. Wang. Nchiql: The chinese natural language interface to databases. In *Proc. of DEXA-01, 12th Int. Conf. on Database and Expert Systems Applications*, volume 2113 of *Lecture Notes in Computer Science*, pp. 145–154. Springer, London, UK, 2001.

[133] T. M. Mitchell. *Machine Learning*. McGraw Hill, New York, 1996.

[134] R. Mitkov, editor. *The Oxford Handbook of Computational Linguistics*. Oxford University Press, Gosport, Hampshire, 2004.

[135] J. L. Neto, A. D. Santos, C. A. A. Kaestner, and A. A. Freitas. Document clustering and text summarization. In *Proc of PADD-00, 4th Int. Conf. on Practical Applications of Knowledge Discovery and Data Mining*, pp. 41–55, London, UK, 2000.

[136] Németh László. Huntoken 1.4 kézikönyv, 2003.

[137] W. C. Ogden and P. Bernick. Using natural language interfaces. Technical Report MCCS-96-229, New Mexico State University, 1996.

[138] S. Olsen. IBM sets out to make sense of the Web, 2004.

[139] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford University, 1998.

[140] C. D. Paice. Another stemmer. *SIGIR Forum*, 24:56–61, 1990.

[141] L. A. F. Park. *Spectral Based Information Retrieval*. PhD thesis, The University of Melbourne, 2003.

[142] W. H. Paxton. *A Framework for Speech Understanding*. PhD thesis, SRI Artificial Intelligence Center, Menlo Park, USA, 1977.

[143] Ch. R. Perrault and B. J. Grosz. Natural-language interfaces. *Annual Review of Computer Science*, 1:47–82, 1986.

[144] J. L. Peterson. *Petri Net Theory and the Modelling of Systems*. Prentice-Hall, 1981.

[145] B. Pinkerton. Finding what people want: Experiences with the WebCrawler. In *Proc. of 2nd Int. WWW Conf.*, pp. 708–716, 1994.

[146] Pléh Csaba. Mondatmegértés a magyar nyelvben. In *Mondatközi viszonyok feldolgozása: az anafora megértése a magyarban*, pp. 164–195. Osiris, 2004.

[147] S. P. Ponzetto and M. Strube. Exploiting semantic role labeling, WordNet and Wikipedia for coreference resolution. In *Proc. of HLT-NAACL, Human Language Technology Conf. of the NAACL*, pp. 192–199, New York, USA, 2006.

[148] A.-M. Popescu, A. Armanasu, O. Etzioni, D. Ko, and A. Yates. Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In *Proc. of COLING-04, 20st Int. Conf. on Computational Linguistics*, pp. 141–147, 2004.

[149] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[150] M. F. Porter. Snowball: A language for stemming algorithms, 2001.

[151] Prószéky Gábor. *Számítógépes nyelvészet*. Számalk, Budapest, 1989.

[152] Prószéky Gábor. NewsPro: automatikus információszerzés gazdasági rövidhírekből. In *I. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY-03)*, pp. 161–167, Szeged, 2003.

[153] D. Radev, S. Blair-Goldensohn, and Z. Zhang. Experiments in single and multidocument summarization using MEAD. In *Proc. of DUC-01, Document Understanding Conf., Workshop on Text Summarization*, New Orleans, USA, 2001.

[154] R. A. P. Rangel, A. F. Gelbukh, J. J. G. Barbosa, E. A. Ruiz, A. M. Mejía, and A. P. D. Sánchez. Spanish natural language interface for a relational database querying system. In *Proc. of TSD-02, 5th Conf. on Text, Speech, and Dialogue*, volume 2448 of *Lecture Notes in Computer Science*, pp. 123–130. Springer, Brno, Czech Republic, 2002.

[155] P. Reis, J. Matias, and N. Mamede. Edite: A natural language interface to databases – A new perspective for an old approach. In *Proc. of ENTER-97, Information and Communication Technologies in Tourism*, pp. 317–326, Edinburgh, UK, 1997.

[156] J. J. Rocchio. Relevance feedback in information retrieval. In G. Salton, editor, *The SMART Retrieval System — Experiments in Automatic Document Processing*, pp. 313–323. Prentice-Hall, Englewood Cliffs, 1971.

[157] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in brain. *Psychological Review*, pp. 386–407, 1958. (Újranyomva: *Neurocomputing*, MIT Press, 1988).

[158] D. Roth. Learning to resolve natural language ambiguities: a unified approach. In *Proc. of AAAI-98, 15th Conf. of the Am. Assoc. for Artificial Intelligence*, pp. 806–813, Madison, USA, 1998.

[159] J. Ruppenhofer, M. Ellsworth, M. R. L. Petruck, Ch. R. Johnson, and J. Scheffczyk. *FrameNet II: Extended Theory and Practice*. International Computer Science Institute, Berkeley, 2006.

[160] G. Salton, editor. *The SMART Retrieval System — Experiments in Automatic Document Processing*. Prentice-Hall, Englewood Cliffs, 1971.

[161] G. Salton and C. Buckley. Term weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1998.

[162] G. Salton and M. J. McGill. *An Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[163] E. F. T K. Sang and F. De Meulder. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proc. of CoNLL-03, 7th Conf. on Computational Natural Language Learning*, pp. 142–147, Edmonton, Canada, 2003.

[164] R. E. Schapire and Y. Singer. BoosTexter: a boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168, 2000.

[165] R. E. Schapire, Y. Singer, and A. Singhal. Boosting and Rocchio applied to text filtering. In *Proc. of SIGIR-98, 21st ACM Int. Conf. on Research and Development in Information Retrieval*, pp. 215–223, Melbourne, Australia, 1998.

[166] B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2001.

[167] P. Schönhofen and H. Charaf. Sentence-based document size reduction. In *Proc. of ClusWEB-04, Clustering Information over the Web*, pp. 77–86, Heraklion, Greece, 2004.

[168] H. Schütze, D. A. Hull, and J. O. Pedersen. A comparison of classifiers and document representations for the routing problem. In *Proc. of SIGIR-95, 18th ACM Int. Conf. on Research and Development in Information Retrieval*, pp. 229–237, Seattle, USA, 1995.

[169] F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002.

[170] F. Sebastiani, A. Sperduti, and N. Valdambrini. An improved boosting algorithm and its application to automated text categorization. In *Proc. of CIKM-00, 9th ACM Int. Conf. on Information and Knowledge Management*, pp. 78–85, McLean, USA, 2000.

[171] S. Sekine, K. Sudo, and C. Nobata. Extended named entity hierarchy. In *Proc. of LREC-02, 3rd Int. Conf. on Language Resources and Evaluation*, pp. 1818–1824, 2002.

[172] D. Shen, R. Pan, J.-T. Sun, J. J. Pan, K. Wu, J. Yin, and Q. Yang. Q^{2}C@UST: Our winning solution to query classification in KDD cup 2005. *ACM SIGKDD Explorations Newsletter*, 7(2):100–110, 2005.

[173] T. Sibanda and Ö. Uzuner. Role of local context in automatic deidentification of ungrammatical, fragmented text. In *Proc. of HLT-NAACL, Human Language Technology Conf. of the NAACL*, pp. 65–73, New York, USA, 2006.

[174] C. Silverstein, H. R. Henzinger, H. Marais, and M. Moricz. Analysis of a very large web search engine query log. *ACM SIGIR Forum*, 33(1):6–12, 1999.

[175] J. Slocum. A practical comparison of parsing strategies. In *Proc. of ACL-81, 19th Annual Meeting Assoc. for Computational Linguistics*, pp. 1–6, Stanford, USA, 1981.

[176] L. I. Smith. A tutorial on principal component analysis. Technical report, University of Otago, New Zealand, 2002.

[177] W. M. Soon, H. T. Ng, and D. C. Y. Lim. A machine learning approach to coreference resolution of noun phrases. *Computational Linguistics*, 27(4):521–544, 2001.

[178] K. Sparck Jones, S. Walker, and S. E. Robertson. A probabilistic model of information retrieval: development and comparative experiments. *Information Processing and Management*, 36(6):779–808, 2000.

[179] M. Steinbach, Karypis G., and V. Kumar. A comparison of document clustering techniques. In *Proc. of Text Mining Workshop at SIGKDD-00, 6th Int. Conf. on Knowledge Discovery and Data Mining*, Boston, USA, 2000.

[180] P. D. Stotts and R. Furuta. Petri-net-based hypertext: Document structure with browsing semantics. *ACM Trans. on Information Systems*, 7(1):3–29, 1989.

[181] Gy. Szarvas, R. Farkas, L. Felföldi, A. Kocsor, and J. Csirik. A highly accurate named entity corpus for Hungarian. In *Proc. of LREC-06, 5th Int. Conf. on Language Resources and Evaluation*, Genoa, Italy, 2006.

[182] M. Templeton and J. Burger. Problems in natural-language interface to DBMS with examples from EUFID. In *Proc. of the 1st Conference on Applied Natural Language Processing*, pp. 3–16, Santa Monica, USA, 1983.

[183] Tikk Domonkos, Biró György, Szidarovszky Ferenc P., Kardkovács Zsolt T., Héder Mihály és Lemák Gábor. Magyar internetes gazdasági tematikájú tartalmak keresése. In *IV. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY-06)*, pp. 3–14, Szeged, 2006.

[184] D. Tikk, Gy. Biró, and A. Törcsvári. A hierarchical online classifier for patent categorization. In H. A. do Prado and E. Ferneda, editors, *Emerging Technologies of Text Mining: Techniques and Applications*. Idea Group Inc., 2007. (nyomdában).

[185] D. Tikk and Gy. Biró. Experiments with multilabel text classifier on the Reuters collection. In *Proc. of ICCC-03, 1st Int. Conf. on Computational Cybernetics*, pp. 33–38, Siófok, Hungary, 2003.

[186] D. Tikk, Gy. Biró, and J. D. Yang. Experiments with a hierarchical text categorization method on WIPO patent collections. In N. O. Attok-Okine and B. M. Ayyub, editors, *Applied Research in Uncertainty Modelling and Analysis*, number 20 in Int. Series in Intelligent Technologies, pp. 283–302. Springer, 2005.

[187] D. Tikk, Zs. T. Kardkovács, Z. Andriska, G. Magyar, A. Babarczy, and I. Szakadát. Natural language question processing for Hungarian deep web searcher. In *Proc. of ICCC-04, 2nd IEEE Int. Conf. on Computational Cybernetics*, pp. 303–309, Vienna, Austria, 2004.

[188] D. Tikk, Zs. T. Kardkovács, G. Magyar, A. Babarczy, and I. Szakadát. Natural language module of a Hungarian deep web searcher. In *Proc. of ISDA-04, 4 th IEEE Int. Conf. on Intelligent Systems Design and Applications*, pp. 73–77, Budapest, Hungary, 2004.

[189] D. Tikk, Zs. T. Kardkovács, and F. P. Szidarovszky. Voting with a parameterized veto strategy: Solving the KDD cup 2006 problem by means of a classifier committee. *ACM SIGKDD Explorations Newsletter*, 8(2):53–62, 2006.

[190] Tikk Domonkos, Törcsvári Attila, Biró György és Bánsághi Zoltán. Szótövező eljárások hatása magyar szövegek automatikus kategorizálásánál. In *III. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY-05)*, pp. 430–434, Szeged, 2005.

[191] S. Tomlinson. Finnish, Portugese and Russian retrieval with Hummingbird SearchServer at CLEF 2004. In *Working Notes for the CLEF 2004 Workshop*, Bath, UK, 2004.

[192] A. Tordai and M. de Rijke. Hungarian monolingual retrieval at CLEF 2005. In *Working Notes for the CLEF 2005 Workshop*, Vienna, Austria, 2005.

[193] Trón Viktor. HunLex – morfológiai szótárkezelő rendszer. In *II. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY-04)*, pp. 177–182, Szeged, 2004.

[194] Trón Viktor, Halácsy Péter, Rebrus Péter, Rung András, Simon Eszter és Vajda Péter. Morphdb.hu: magyar morfológiai nyelvtan és szótári adatbázis. In *III. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY-05)*, pp. 169–179, Szeged, 2005.

[195] V. Trón, L. Németh, P. Halácsy, A. Kornai, Gy. Gyepesi, and D. Varga. Hunmorph: open source word analysis. In *Proc. of ACL Workshop on Software at the 43rd Annual Meeting of ACL*, 2005.

[196] K. Tumer and J. Ghosh. Error correlation and error reduction in ensemble classifiers. *Connection Science*, 8(3–4):385–403, 1996.

[197] L. Underwood. A brief history of search engines, 2005.

[198] C. J. van Rijsbergen. *Information Retrieval*. Butterworths, London, 2nd edition, 1979.

[199] V. N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998.

[200] Varasdi Károly. Koreferenciák feloldása, 2006. Magyar WordNet Projekt, projekt beszámoló.

[201] Vivisimo.com. How the Vivísimo clustering engine works, 2002.

[202] D. L. Waltz. An english language question answering system for a large relational database. *Communications of the ACM*, 21(7):526–539, 1978.

[203] D. H. D. Warren and F. C. N. Pereira. An efficient easily adaptable system for interpreting natural language queries. *Computational Linguistics*, 8(3–4):110–122, 1982.

[204] R. M. Weischedel, R. J. Bobrow, D. Ayuso, and L. Ramshaw. Portability in the janus natural language interface. In *Proc. of HLT-89, Workshop on Speech and Natural Language*, pp. 112–117, Philadelphia, USA, 1989.

[205] R. M. Weischedel, E. Walker, D. Ayuso, J. de Bruin, K. Koile, L. Ramshaw, and V. Shaked. Out of the laboratory: a case study with the IRUS natural language interface. In *Proc. of HLT-86, Workshop on Strategic Computing Natural Language*, pp. 44–61, Marina del Rey, USA, 1986.

[206] S. M. Weiss, C. Apte, F. J. Damerau, D. E. Johnson, F. J. Oles, T. Goetz, and T. Hampp. Maximizing text-mining performance. *IEEE Intelligent Systems*, 14(4):2–8, 1999.

[207] S. M. Weiss, N. Indurkhyra, T. Zhang, and F. J. Damerau. *Text Mining: Predictive Methods for Analyzing Unstructured Information*. Springer, 2004.

[208] W. Wibovo and H. E. Williams. Simple and accurate feature selection for hierarchical categorisation. In *Proc. of DE-02, ACM Symposium on Document Engineering*, pp. 111–118, McLean, USA, 2002.

[209] E. D. Wiener, J. O. Pedersen, and A. S. Weigend. A neural network approach to topic spotting. In *Proc. of the SDAIR-95, 4th Annual Symposium on Document Analysis and Information Retrieval*, pp. 317–332, Las Vegas, NV, 1995.

[210] W. Winiwarter. *The Integrated Deductive Approach to Natural Language Interfaces*. PhD thesis, Department of Information Engineering, University of Vienna, Austria, 1994.

[211] M. J. Witbrock and V. Mittal. Ultra-summarization: A statistical approach to generating highly condensed non-extractive summaries. In *Proc. of SIGIR-99, 22nd ACM Int. Conf. on Research and Development in Information Retrieval*, pp. 315–316, Berkeley, USA, 1999.

[212] M. B. Wolfe, M. E. Schreiner, B. Rehder, D. Laham, P. W. Foltz, W. Kintsch, and T. K. Landauer. Learning from text: Matching readers and texts by Latent Semantic Analysis. *Discourse Processes*, 25(2/3):309–336, 1998.

[213] W. A. Woods. Context-sensitive parsing. *Communications of the ACM*, 13(7):437–445, 1970.

[214] W. A. Woods, R. M. Kaplan, and B. L. Nash-Webber. The lunar sciences natural language information system. Technical Report BBN Report No. 2378, Bolt, Beranek, and Newman Inc., Cambridge, USA, 1972.

[215] Y. Yang. Expert network: effective and efficient learning from human decisions in text categorisation and retrieval. In *Proc. of SIGIR-94, 17th ACM Int. Conf. on Research and Development in Information Retrieval*, pp. 13–22, Dublin, Ireland, 1994.

[216] Y. Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval*, 1(1–2):69–90, 1999.

[217] Y. Yang and C. G. Chute. An example-based mapping method for text categorization and retrieval. *ACM Trans. Inform. Syst.*, 12(3):252–277, 1994.

[218] Y. Yang and X. Liu. A re-examination of text categorization methods. In *Proc. of SIGIR-99, 22nd ACM Int. Conf. on Research and Development in Information Retrieval*, pp. 42–49, Berkeley, USA, 1999.

[219] Y. Yang and J. P. Pedersen. Feature selection in statistical learning of text categorization. In *Proc. of ICML-97, 14th Int. Conf. on Machine Learning*, pp. 412–420, Nashville, USA, 1997.

[220] O. Zamir, O. Etzioni, O. Madani, and R. M. Karp. Fast and intuitive clustering of web documents. In *Proc. of SIGKDD-97, 3rd Int. Conf. on Knowledge Discovery and Data Mining*, pp. 287–290, Newport Beach, USA, 1997.

[221] T. Zhang and F. J. Oles. Text categorization based on regularized linear classification methods. *Information Retrieval*, 4(1):5–31, 2001.

Tárgymutató