# További klaszterező módszerek

Ispány Márton és Jeszenszky Péter

2016. november 8.

# Tartalom

# Klaszterező algoritmusok (1)

Nagyszámú klaszterező algoritmust fejlesztettek ki különböző alkalmazási területekre.

Olyan módszerek vannak, amelyek jól működnek bizonyos helyzetekben.

# Klaszterező algoritmusok (2)

Sokszor szubjektív értelmezés kérdése, hogy mit tekintünk jó klaszterhalmaznak.

Ha egy objektív mérőszámot alkalmazunk a klaszter precíz definíciójaként, akkor az optimális klaszterezés megtalálásának problémája gyakran megvalósíthatatlanul számításigényes.

# A $K$-közép és DBSCAN összehasonlítása (1)

- A DBSCAN és a $K$-közép egyaránt felosztó klaszterező algoritmusok, amelyek minden objektumot egy klaszterhez rendelnek, de a $K$-közép tipikusan minden objektumot klaszterez, míg a DBSCAN kihagyja azokat az objektumokat, amiket zajnak osztályoz.
- A $K$-közép prototípus-alapú, míg a DBSCAN sűrűség-alapú klaszterfogalmat használ.
- A DBSCAN különböző méretű és alakú klasztereket is kezelni tud, és nem különösebben érzékeny a zajra vagy a kiugró értékekre. A $K$-középnek nehézséget okoznak a nem gömb alakú klaszterek és a különböző méretű klaszterek. Mindkét algoritmus rosszul tud teljesíteni, ha a klaszterek nagyon különböző sűrűségűek.

# A $K$-közép és DBSCAN összehasonlítása (2)

- A $K$-közepet csak olyan adatokra lehet használni, amelyeknek jól definiált középértéke van, mint például az átlag vagy a medián. A DBSCAN megköveteli, hogy értelmes legyen az adatokra a hagyományos euklideszi sűrűségfogalomra épülő sűrűség definíciója.
- A $K$-közép alkalmazható ritka, sokdimenziós adatokra, mint például a dokumentum adatok. A DBSCAN tipikusan gyengén teljesít ilyen adatokon, mert a hagyományos euklideszi sűrűségfogalom nem működik jól sokdimenziós adatokra.
- A $K$-közép és DBSCAN eredeti változatait euklideszi adatokra dolgozták ki, de mindkettőt kiterjeszették más típusú adatok kezelésére is.

# A $K$-közép és DBSCAN összehasonlítása (3)

- A DBSCAN nem feltételez semmit az adatok eloszlásáról. Az elemi $K$-közép algoritmus ekvivalens azzal a statisztikai klaszterező megközelítéssel (keverék modell), ami feltételezi, hogy minden klaszter különböző várható értékű, de azonos kovarianciamátrixú szferikus normális eloszlásból származik.
- A DBSCAN és a $K$-közép is úgy keres klasztereket, hogy minden attribútumot felhasználnak, azaz nem keresnek olyan klasztereket, amelyek csak az attribútumok egy részhalmazán alapulnak.
- A $K$-közép meg tud találni olyan klasztereket, amelyek nem különülnek el jól, még akkor is, ha átfedőek, azonban a DBSCAN összevonja az átfedő klasztereket.

# A $K$-közép és DBSCAN összehasonlítása (4)

- A $K$-közép algoritmus időbonyolultsága $O(m)$, míg a DBSCAN $O(m^2)$ időt igényel, eltekintve olyan speciális esetektől, mint például az alacsony dimenziós euklideszi adatok.
- A DBSCAN futásról futásra ugyanazokat a klasztereket állítja elő, viszont a $K$-közép, amit tipikusan a középpontok véletlenszerű inicializásával használunk, nem.
- A DBSCAN automatikusan meghatározza a klaszterek számát, a $K$-középnél a klaszterek számát paraméterként kell megadni. A DBSCAN-nek viszont két másik paramétere van (*Eps* és *MinPts*), amelyeket meg kell adni.

# A $K$-közép és DBSCAN összehasonlítása (5)

- A $K$-közép klaszterezés tekinthető optimalizációs problémaként is – azaz minimalizáljuk a pontoknak a legközelebbi középponttól vett négyzetes hibaösszegét –, és egy statisztikai klaszterező megközelítés különleges eseteként is (keverék modellek). A DBSCAN nem alapul semmilyen formális modellen.

# Statisztikai modelleken alapuló klaszterezés

Feltevés, hogy az adatok generálása egy statisztikai folyamat eredményeként történt.

Az adatokat a legjobban illeszkedő statisztikai modellel írjuk le.

A modellt egy eloszlással és annak paramétereivel adjuk meg.

Úgynevezett **keverék modell** használata, mely az adatokat több statisztikai eloszlással modellezi. Minden eloszlás egy klaszternek felel meg.

# Ütemterv

1. Keverék modellek
2. Statisztikai modellek paraméterbecslése
   - 2.1 Egyszerű statisztikai modellek paramétereinek **maximum likelihood becslés**e
   - 2.2 A maximum likelihood becslés általánosítása keverék modellek paramétereinek becslésére: **EM algoritmus**
3. Az EM algoritmus használata klaszterezésre

# Keverék modellek (1)

Az adatokra úgy tekintünk, mint különböző valószínűségeloszlások keverékéből származó megfigyeléshalmazra.

Tetszőleges eloszlásokat használhatunk, de a leggyakrabban többdimenziós normális eloszlásokat tekintünk, amelyek ellipszoid alakú klasztereket tudnak modellezni.

# Keverék modellek (2)

Adatgeneráló folyamat:

1. Legyenek adott valószínűségeloszlások, általában ugyanabból a típusból, de különböző paraméterekkel.
2. Válasszunk véletlenszerűen az eloszlások közül egyet és generáljunk belőle egy objektumot.
3. Ismételjük meg a folyamatot $m$-szer, ahol $m$ az objektumok száma.

# Keverék modellek (3)

Jelölés:

- $m$: az objektumok száma
- $\mathbf{x}_i$: az $i$-edik objektum ($i = 1, \ldots, m$)
- $\mathcal{X}$: az objektumok halmaza, azaz $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$
- $K$: az eloszlások száma
- $\theta_j$: a $j$-edik eloszlás paraméterei ($j = 1, \ldots, K$)
- $\Theta$: az összes paraméter halmaza, azaz $\Theta = \{\theta_1, \ldots, \theta_K\}$
- $P(\mathbf{x}_i \mid \theta_j)$: az $i$-edik objektum valószínűsége, ha a $j$-edik eloszlásból származik ($i = 1, \ldots, m$, $j = 1, \ldots, K$)
- $w_j$: annak a valószínűsége, hogy a $j$-edik eloszlást választjuk egy objektum generálására ($j = 1, \ldots, K$)

## Megjegyzés

Feltevés, hogy a $w_j$ súlyokra $\sum_{j=1}^{K} w_j = 1$ teljesül.

# Keverék modellek (4)

Ekkor az $\mathbf{x}$ objektum valószínűsége:

$$P(\mathbf{x} \mid \Theta) = \sum_{j=1}^{K} w_j P(\mathbf{x} \mid \theta_j).$$

Ha az objektumokat egymástól függetlenül generáljuk, akkor a teljes objektumhalmaz valószínűsége éppen az egyes $\mathbf{x}_i$ objektumok valószínűségeinek szorzata:

$$P(\mathcal{X} \mid \Theta) = \prod_{i=1}^{m} P(\mathbf{x}_i \mid \Theta) = \prod_{i=1}^{m} \sum_{j=1}^{K} w_j P(\mathbf{x}_i \mid \theta_j).$$

# Keverék modellek (5)

Keverék modelleknél minden egyes eloszlás egy különböző csoportot, azaz különböző klasztert ír le.

Statisztikai módszerekkel becsülhetjük az eloszlások paramétereit az adatokból.

Azonosítani tudjuk, hogy mely objektumok mely klaszterekhez tartoznak.

A keverék-modellezés azonban nem állítja elő az objektumok éles hozzárendelését a klaszterekhez, hanem inkább az objektumok egy bizonyos klaszterhez tartozásának a valószínűségét adja meg.

# Példa keverék modellre (1)

## Példa (Egydimenziós normális eloszlások keveréke)

Tegyük fel, hogy két egydimenziós eloszlásunk van, melyek szórása egyformán 2, várható értékük pedig $-4$ és 4. Tegyük fel továbbá, hogy a két eloszlást azonos valószínűséggel választjuk, azaz

$$w_1 = w_2 = 0{,}5.$$

Az egydimenziós normális eloszlás valószínűségi sűrűségfüggvénye egy $x$ pontban

$$P(x \mid \Theta) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

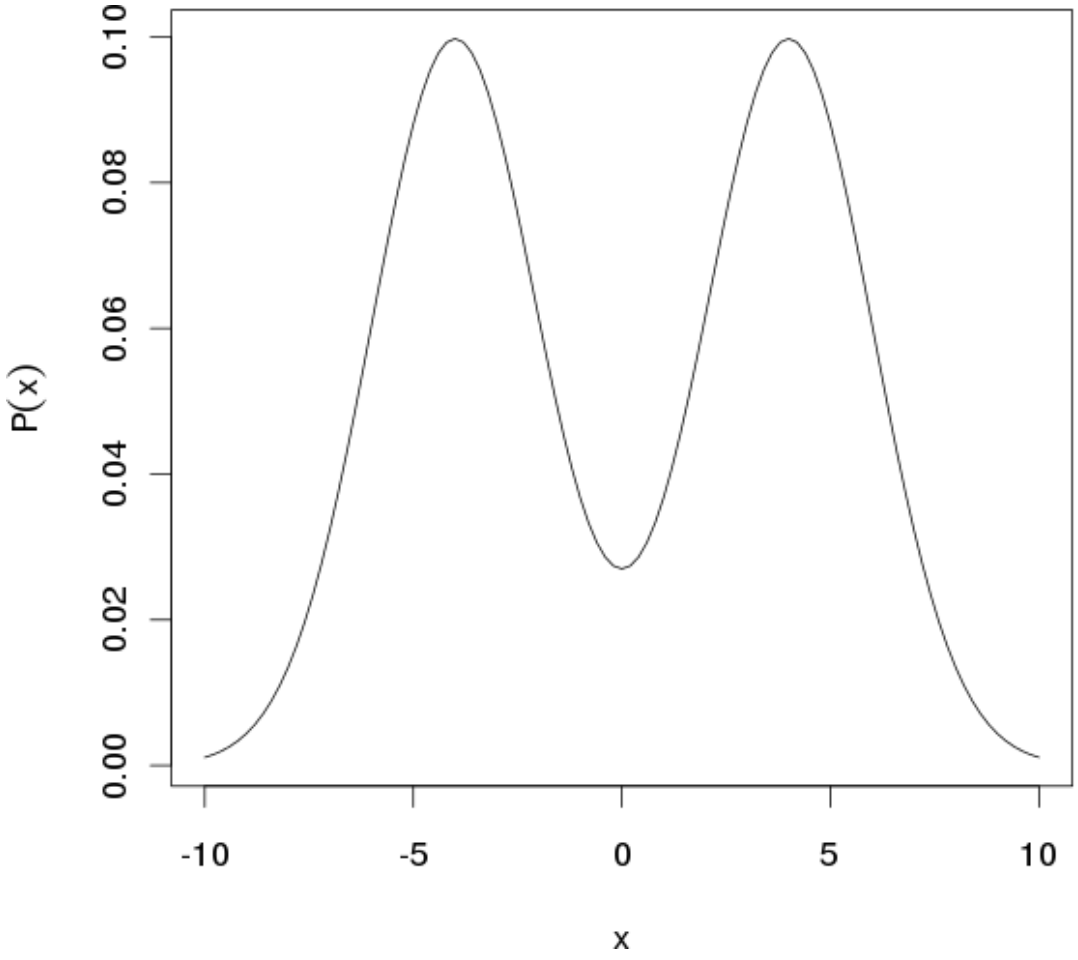
ahol $\Theta = (\mu, \sigma)$ az eloszlás paraméterei: $\mu$ az eloszlás várható értéke, $\sigma$ pedig a szórása.

# Példa keverék modellre (2)

Ekkor

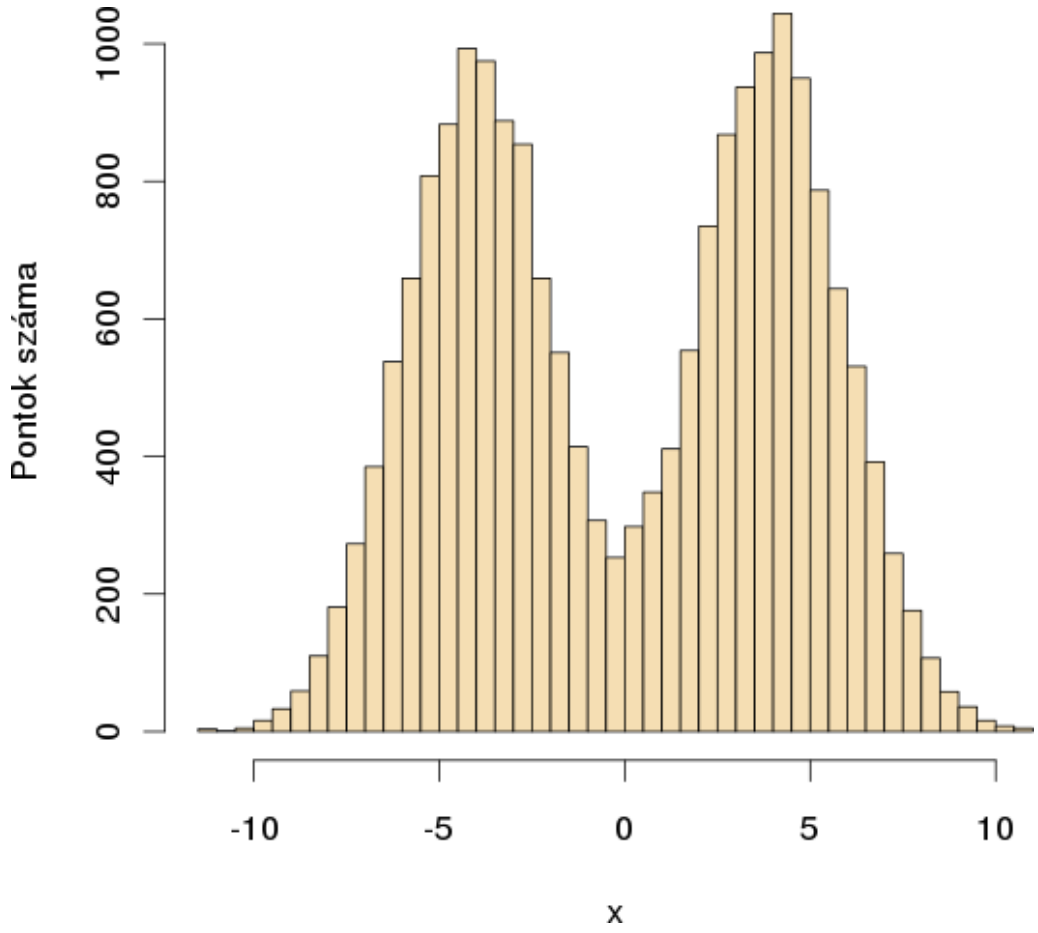
$$P(x \mid \Theta) = \frac{1}{4\sqrt{2\pi}}\, e^{-\frac{(x+4)^2}{8}} + \frac{1}{4\sqrt{2\pi}}\, e^{-\frac{(x-4)^2}{8}}.$$

# Példa keverék modellre (3)




1. ábra. A keverék modell valószínűségi sűrűségfüggvénye

# Példa keverék modellre (4)




2. ábra. A keverék modellből generált 20 000 pont hisztogramja

# Paraméterbecslés maximum likelihood módszerrel (1)

Ha adott egy statisztikai modell az adatokra, meg kell becsülni a modell paramétereit.

Erre a standard megoldás a **maximum likelihood becslés**.

# Paraméterbecslés maximum likelihood módszerrel (2)

Tekintsünk egy $m$ pontból álló halmazt, melyet egydimenziós Gauss-eloszlásból generáltunk.

A pontok valószínűsége, ha a pontokat egymástól függetlenül generáltuk:

$$P(\mathcal{X} \mid \Theta) = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} \; e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}. \qquad (1)$$

Mivel ez a valószínűség egy nagyon kicsi szám lenne, tipikusan a valószínűség (természetes) logaritmusával dolgozunk:

$$log \; P(\mathcal{X} \mid \Theta) = -\sum_{i=1}^{m} \frac{(x_i-\mu)^2}{2\sigma^2} - 0{,}5m \log 2\pi - m \log \sigma. \qquad (2)$$

# Paraméterbecslés maximum likelihood módszerrel (3)

Adjunk eljárást az ismeretlen $\mu$ és $\sigma$ paraméterek becslésére!

Egy megközelítés azon paraméterértékek választása, amelyekre az adatok a legvalószínűbbek.

Válasszuk az (1) képlet értékét maximalizáló $\mu$ és $\sigma$ értéket.

Ez a megközelítés a **maximum likelihood elv**, a **maximum likelihood becslés (MLE)** pedig az a folyamat, ami ezt az elvet alkalmazza a statisztikai eloszlás paramétereinek a becslésére az adatok alapján.

# Paraméterbecslés maximum likelihood módszerrel (4)

**Likelihood függvény**nek nevezzük az adatok a paraméterek függvényeként tekintett valószínűségét.

# Paraméterbecslés maximum likelihood módszerrel (5)

Az (1) képletet az alábbi alakba írjuk át, hogy kihangsúlyozzuk azt, hogy a $\mu$ és $\sigma$ paramétereket tekintjük változóknak, és hogy az adatokat konstansként kezeljük:

$$\mathit{likelihood}(\Theta \mid \mathcal{X}) = L(\Theta \mid \mathcal{X}) = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}. \qquad (3)$$

A loglikelihood függvényt a valószínűség logaritmusának (2) képletéből számoljuk:

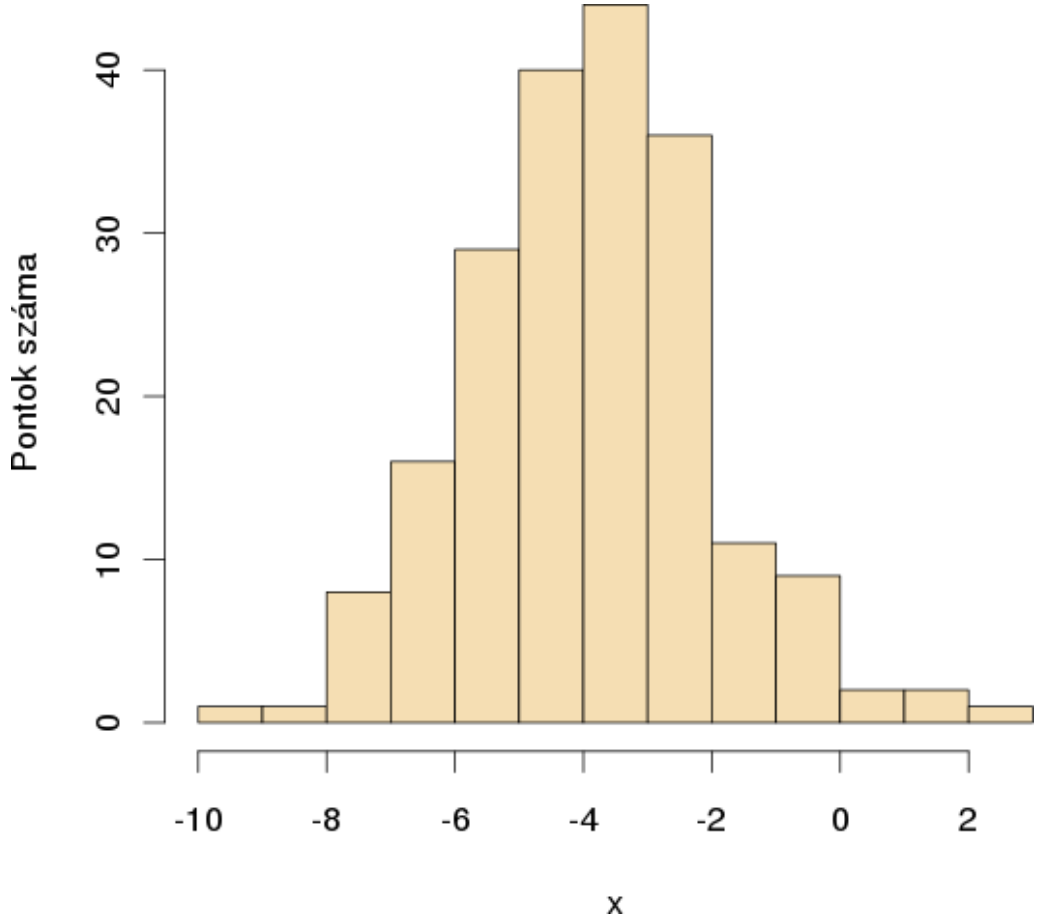
$$\mathit{loglikelihood}(\Theta \mid \mathcal{X}) = \ell(\Theta|\mathcal{X}) = -\sum_{i=1}^{m} \frac{(x_i-\mu)^2}{2\sigma^2} - 0{,}5m\log 2\pi - m\log\sigma. \qquad (4)$$

## Megjegyzés

Praktikus okokból gyakrabban használják a loglikelihoodot.

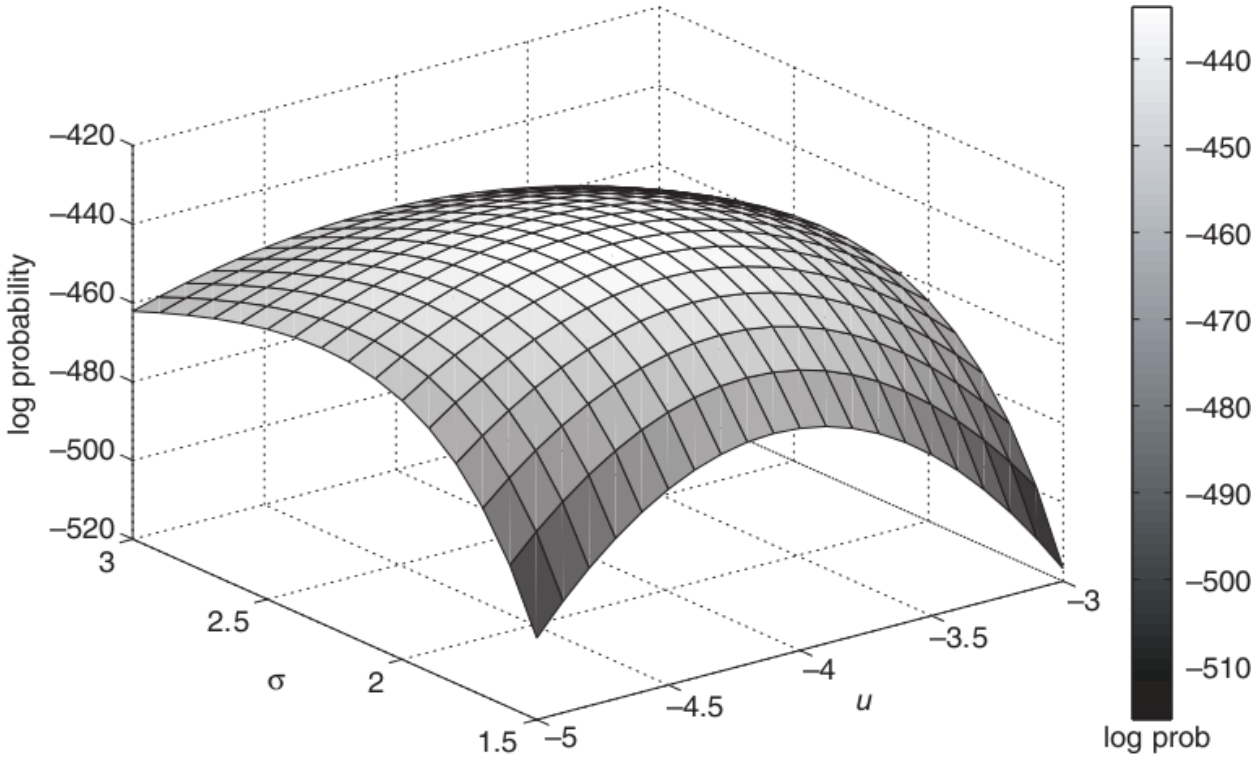
A loglikelihoodot maximalizáló paraméterek egyben a likelihoodot is maximalizálják, mert a logaritmus monoton növekvő függvény.

# Példa maximum likelihood paraméterbecslésre (1)




3. ábra. 200 pont hisztogramja

# Példa maximum likelihood paraméterbecslésre (2)




4. ábra. A 200 pont loglikelihood függvénye

A $\mu = -4{,}1$ és a $\sigma = 2{,}1$ paraméterekre maximális a loglikelihood, amelyek közel vannak az adatokat generáló normális eloszlás paraméterértékeihez ($\mu = -4$ és $\sigma = 2$).

# Paraméterbecslés maximum likelihood módszerrel (6)

Az adatok valószínűségét nem praktikus a paraméterek különböző értékeire ábrázolni kettőnél több paraméter esetén.

Egy statisztikai paraméter maximum likelihood becslésének kiszámításához a standard eljárás a loglikelihood függvény deriváltját venni a paraméter szerint, az eredményt egyenlővé tenni 0-val, és a kapott egyenletet megoldani.

## Megjegyzés

A normális eloszlásra meg lehet mutatni, hogy a mintaelemek átlaga és szórása az eloszlás paramétereinek maximum likelihood becslése.

# Keverék modellek paraméterbecslése maximum likelihood módszerrel

Ha tudjuk, hogy melyik adatobjektum melyik eloszlásból származik, akkor a probléma visszavezetődik arra, amikor egy eloszlás paramétereit kell becsülnünk az ebből az eloszlásból származó adott adatok alapján.
A legtöbb tipikus eloszlásra a paraméterek maximum likelihood becslését egyszerű képletek alapján számolhatjuk az adatokból.

Általában nem tudjuk, hogy melyik pontot melyik eloszlás generálta. Így nem tudjuk közvetlenül számolni az egyes adatpontok valószínűségét, ezért úgy tűnik, hogy a maximum likelihood elvet sem tudjuk használni a paraméterek becslésére.

A fenti problémára az EM algoritmus kínál megoldást.

# EM algoritmus (1)

**1. algoritmus.** EM algoritmus

1: Válasszunk kezdeti értékeket a modell paramétereire
(ahogy a $K$-középnél, ez történhet a véletlen segítségével vagy több más módon is)
2: **repeat**
3: **E (expectation, várható érték) lépés** Minden objektumra számoljuk ki annak valószínűségét, hogy az objektum az egyes eloszlásokhoz tartozik, azaz számoljuk ki a $P(j.\ \text{eloszlás} \mid \mathbf{x}_i, \Theta)$ valószínűségeket
4: **M (maximalizáló) lépés** Az E-lépésből adódó valószínűségekkel keressük meg az új paraméterbecsléseket, amelyek maximalizálják a várt likelihoodot
5: **until** a paraméterek nem változnak
(vagy megállás akkor, ha a paraméterek változása egy adott küszöbértéknél kisebb)

# EM algoritmus (2)

Ha van egy becslésünk a paraméterértékekre, az EM algoritmus kiszámolja annak a valószínűségét, hogy a pontok az egyes eloszlásokhoz tartoznak, majd ezeket a valószínűségeket használja a paraméterek új becsléséhez. (Ezek a paraméterek maximalizálják a likelihoodot.)

Ezt addig folytatjuk, míg a paraméterbecslések vagy nem változnak, vagy csak nagyon kicsit.

Továbbra is a maximum likelihood becslést használjuk, de egy iteratív keresésen keresztül.

# EM algoritmus (3)

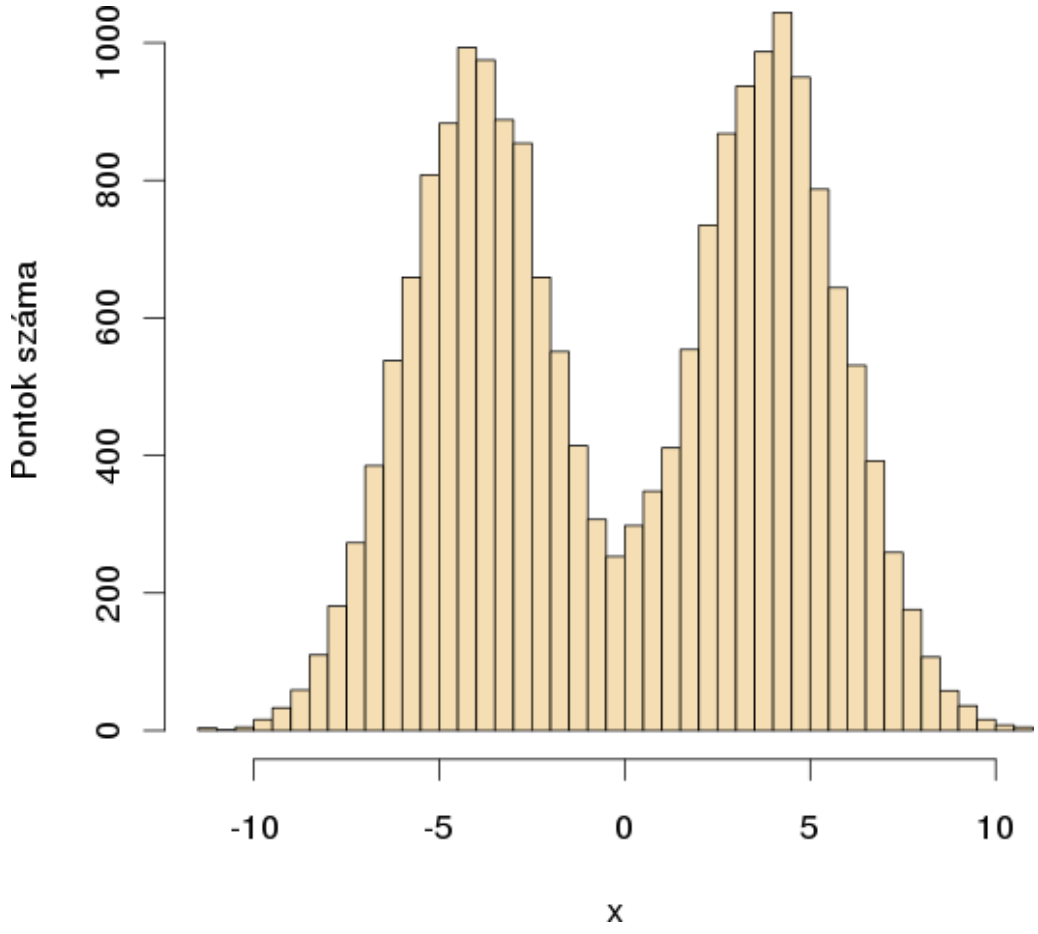
Az EM algoritmus hasonló a $K$-közép eljáráshoz.

Euklideszi adatokra a $K$-közép algoritmus speciális esete az EM algoritmusnak, térbeli, azonos kovarianciamátrixú, de különböző várható értékű normális eloszlásokkal.

- Az E lépés megfelel annak a $K$-közép lépésnek, amelyben minden objektumot egy klaszterbe sorolunk. Ehelyett minden objektumot valamilyen valószínűséggel rendelünk az egyes klaszterekhez (eloszlásokhoz).
- Az M lépés megfelel a klaszterközéppontok kiszámításának. Ehelyett az eloszlások paramétereit és a súlyparamétereket is úgy választjuk, hogy maximalizálják a likelihoodot.[1]

[1] A paramétereket tipikusan a maximum likelihood becslésből származó képletekkel lehet számolni. Egyetlen normális eloszlásra például a várható érték ML becslése az eloszlásból származó objektumok átlaga.

# Egyszerű példa az EM algoritmusra (1)




5. ábra. Az EM algoritmus működésének szemléltetéséhez használt adatok hisztogramja

# Egyszerű példa az EM algoritmusra (2)

Az egyszerűség kedvéért tegyük fel, hogy tudjuk, hogy mindkét eloszlás szórása 2, és hogy a pontokat a két eloszlásból egyenlő valószínűséggel generáltuk.

A baloldali és a jobboldali eloszlásra 1-es és 2-es eloszlásként hivatkozunk.

# Egyszerű példa az EM algoritmusra (3)

Az EM algoritmust a $\mu_1$ és $\mu_2$ paraméterekre vonatkozó kezdeti becslések megadásával kezdjük, legyen például $\mu_1 = -2$ és $\mu_2 = 3$.

Így a kezdeti $\theta = (\mu, \sigma)$ paraméterek a két eloszlásra

$$\theta_1 = (-2, 2) \quad \text{és} \quad \theta_2 = (3, 2).$$

Az egész keverék paraméterei

$$\Theta = \{\theta_1, \theta_2\}.$$

# Egyszerű példa az EM algoritmusra (4)

Az E lépésében ki szeretnénk számolni annak a valószínűségét, hogy egy pont adott eloszlásból származik, azaz

$$P(1.\text{ eloszlás} \mid x_i, \Theta) \quad \text{és} \quad P(2.\text{ eloszlás} \mid x_i, \Theta)$$

értékét.

Ezeket az értékeket a következő egyenletből fejezhetjük ki:

$$P(j.\text{ eloszlás} \mid x_i, \theta) = \frac{0{,}5P(x_i \mid \theta_j)}{0{,}5P(x_i \mid \theta_1) + 0{,}5P(x_i \mid \theta_2)}, \qquad (5)$$

ahol 0,5 az egyes eloszlások valószínűsége (súlya), $j$ pedig 1 vagy 2.

# Egyszerű példa az EM algoritmusra (5)

## Megjegyzés

Az (5) formulát a közismert Bayes-tételből kapjuk:

$$P(Y \mid X) = \frac{P(X \mid Y)P(Y)}{P(X)},$$

valamint felhasználtuk a teljes valószínűség tételét, miszerint ha $\{X_1, \ldots, X_k\}$ az $X$ valószínűségi változó egymást kölcsönösen kizáró és a teljes eseményteret lefedő kimenetelei, akkor a nevező kifejezhető a következőképpen:

$$P(X) = \sum_{i=1}^{k} P(X, Y_i) = \sum_{i=1}^{k} P(X \mid Y_i)P(Y_i).$$

# Egyszerű példa az EM algoritmusra (6)

Tegyük fel például, hogy az egyik pont a 0.

A normális eloszlás sűrűségfüggvény alapján ki tudjuk számolni, hogy

$$P(0 \mid \theta_1) = 0{,}12 \quad \text{és} \quad P(0 \mid \theta_2) = 0{,}06.$$

Ezekből és az (5) képletből azt kapjuk, hogy

$$P(1.\ \text{eloszlás} \mid 0, \Theta) = 0{,}12/(0{,}12 + 0{,}06) = 0{,}67$$
$$P(2.\ \text{eloszlás} \mid 0, \Theta) = 0{,}06/(0{,}12 + 0{,}06) = 0{,}33.$$

Azaz a paraméterértékekre vonatkozó aktuális feltételezések mellett a 0 kétszer akkora eséllyel tartozik az 1-es eloszláshoz, mint a 2-eshez.

# Egyszerű példa az EM algoritmusra (7)

A klaszterhez tartozási valószínűséget mind a 20 000 pontra kiszámolva új becsléseket számolunk $\mu_1$-re és $\mu_2$-re[2]:

$$\mu_1 = \sum_{i=1}^{20\ 000} x_i \frac{P(\text{1. eloszlás} \mid x_i, \Theta)}{\sum_{j=1}^{20\ 000} P(\text{1. eloszlás} \mid x_j, \Theta)}$$

$$\mu_2 = \sum_{i=1}^{20\ 000} x_i \frac{P(\text{2. eloszlás} \mid x_i, \Theta)}{\sum_{j=1}^{20\ 000} P(\text{2. eloszlás} \mid x_j, \Theta)}$$

[2] Az eloszlás várható értékére az új becslés éppen a pontok súlyozott átlaga, ahol a súlyok a pontok adott eloszláshoz tartozásának a valószínűségei.

# Egyszerű példa az EM algoritmusra (8)

Addig ismételjük a két lépést, amíg a $\mu_1$ és $\mu_2$ becslései vagy nem változnak, vagy csak nagyon kicsit.

# Egyszerű példa az EM algoritmusra (9)

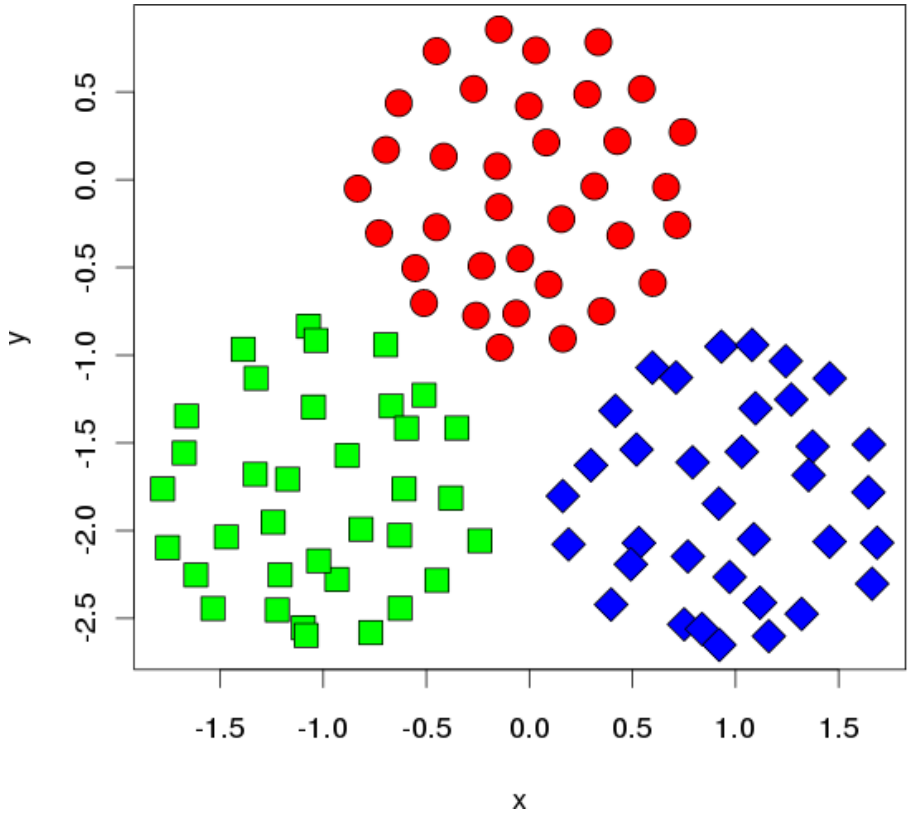
Az EM algoritmus első néhány lépése az adatokra:

| Iteráció | $\mu_1$ | $\mu_2$ |
|---|---|---|
| 0. | $-2{,}00$ | 3,00 |
| 1. | $-3{,}74$ | 4,10 |
| 2. | $-3{,}94$ | 4,07 |
| 3. | $-3{,}97$ | 4,04 |
| 4. | $-3{,}98$ | 4,03 |
| 5. | $-3{,}98$ | 4,03 |

Erre az adatsorra tudjuk, hogy melyik eloszlás melyik pontot generálta, így mindkét eloszlásra ki tudjuk számolni a pontok átlagát is:

$$\mu_1 = -3{,}98 \quad \text{és} \quad \mu_2 = 4{,}03.$$

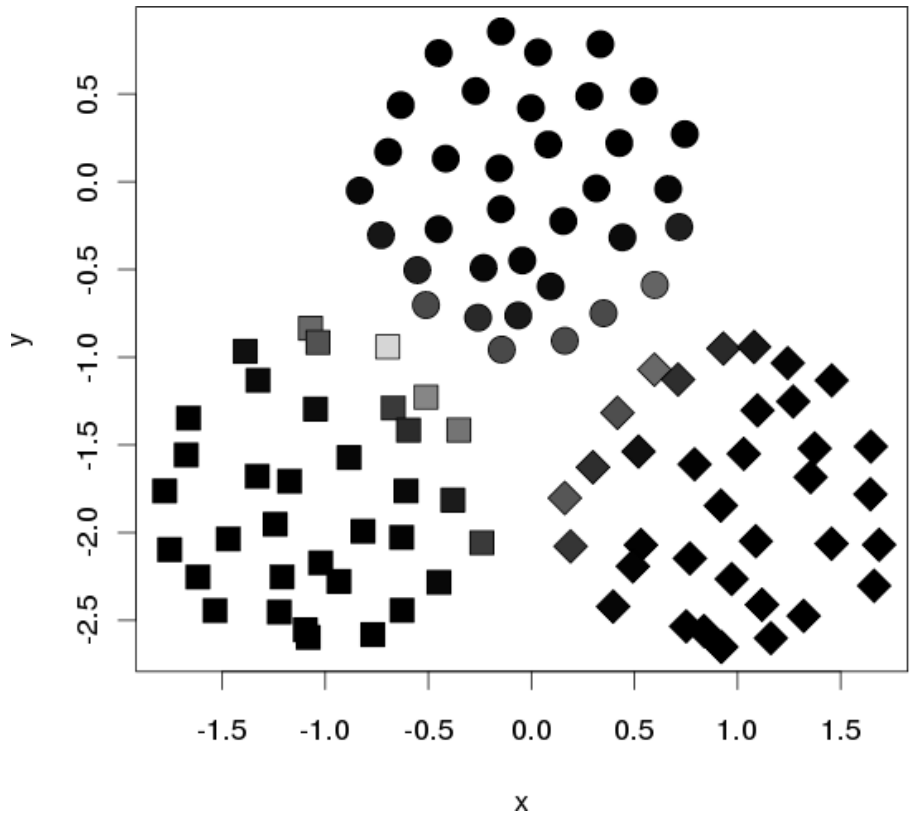
# Példa EM klaszterezésre (1/1)




6. ábra

Modellezzük az adatokat három, különböző várható értékű és megegyező kovarianciamátrixú kétdimenziós normális eloszlás keverékével!

# Példa EM klaszterezésre (1/2)




7. ábra. Az EM algoritmussal történő klaszterezés eredménye. Mindegyik pontot ahhoz a klaszterhez rendeltük hozzá, amelynek a legnagyobb a valószínűsége. A színezés a klaszterhez való tartozás fokát mutatja.

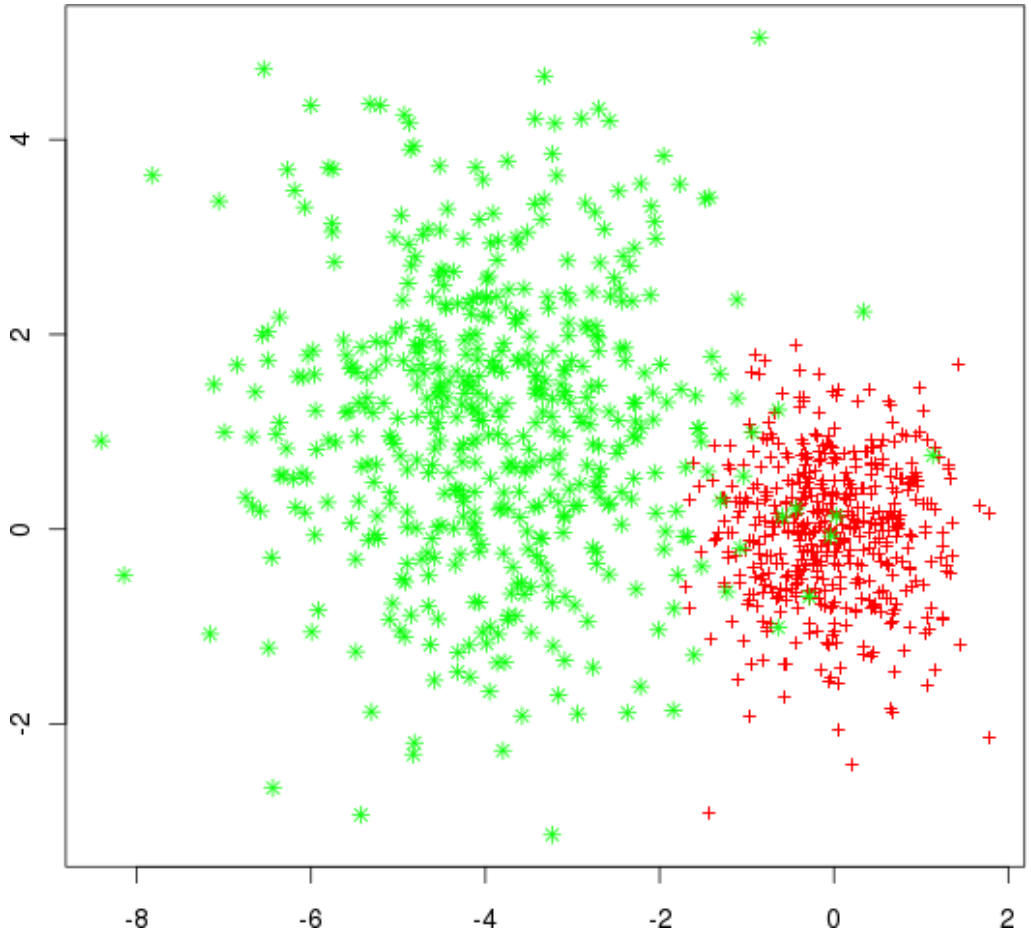
# Példa EM klaszterezésre (2/1)

Különböző sűrűségű klasztereket tartalmazó adatok klaszterezése.

Az adatok két természetes klaszterből állnak, mindegyik nagyjából 500 pontot tartalmaz.

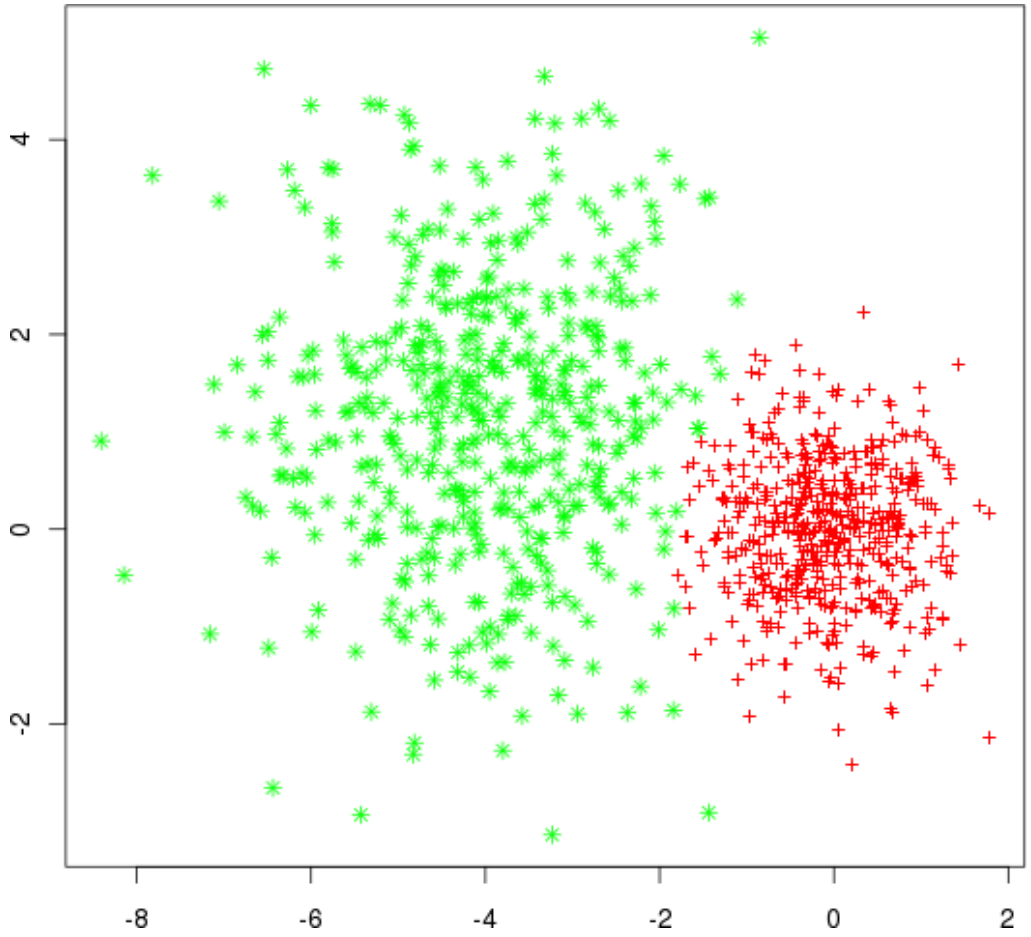
Az adatokat két kétdimenziós normális eloszlás kombinálásával állítottuk elő: az egyik várható értéke $(-4, 1)$, szórása 2, a másik várható értéke $(0, 0)$, szórása pedig 0,5.

# Példa EM klaszterezésre (2/2)



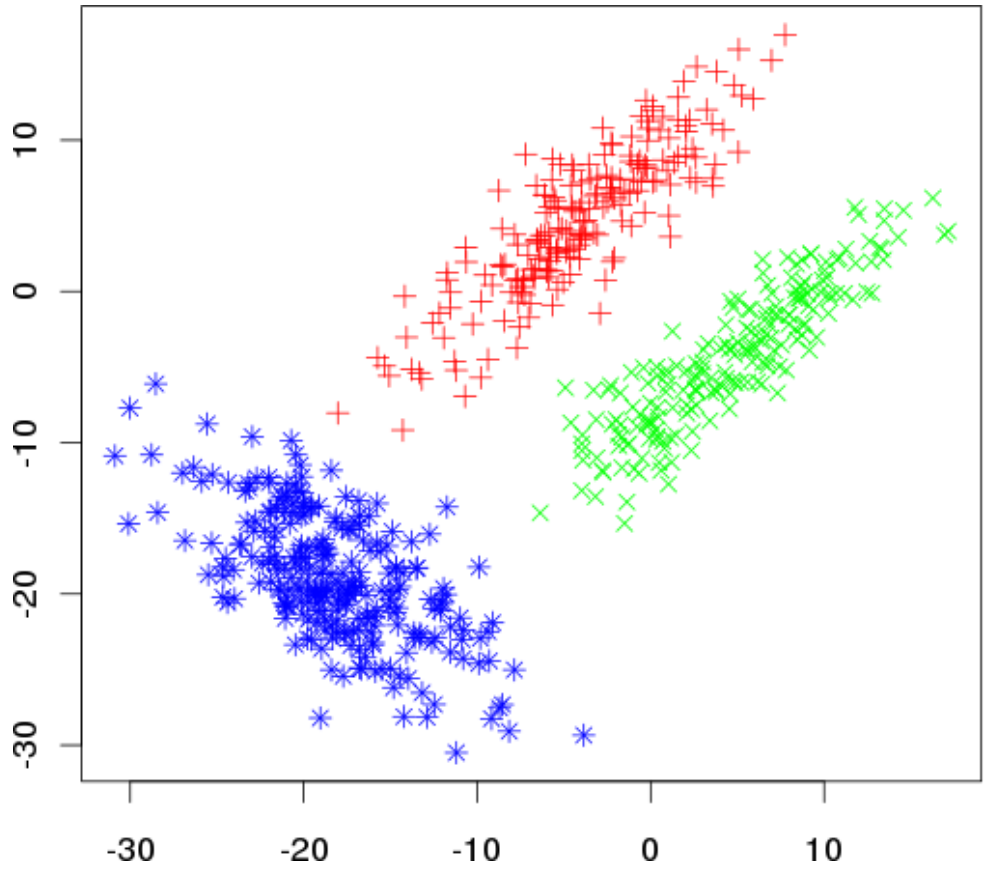

8. ábra. Két kétdimenziós normális eloszlásból származó adatok

# Példa EM klaszterezésre (2/3)



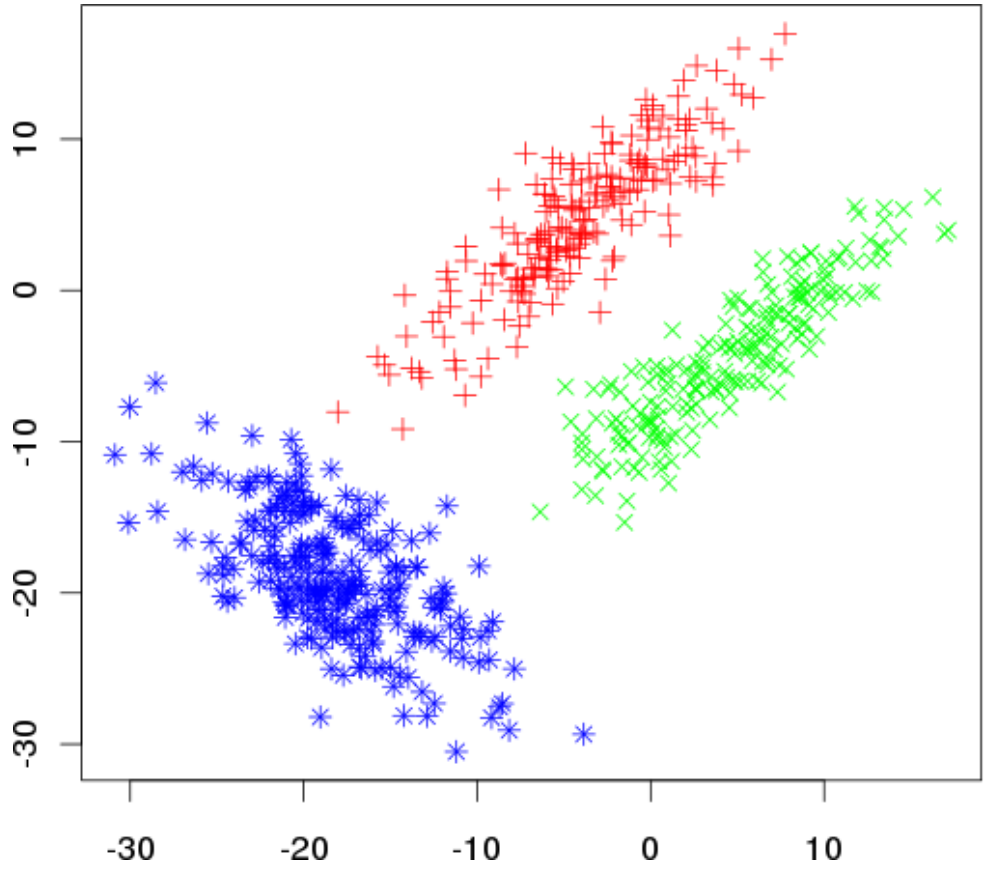

9. ábra. Az EM algoritmussal történő klaszterezés eredménye

# Példa EM klaszterezésre (3/1)



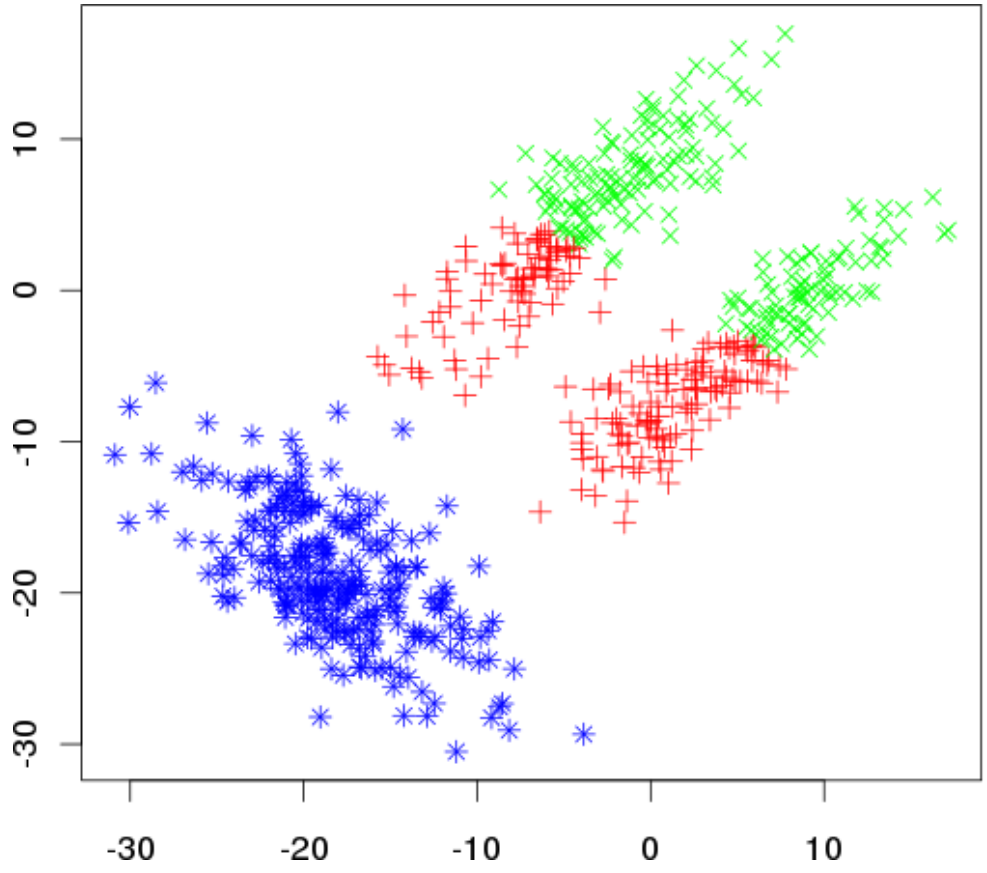

10. ábra. Olyan adatok, melyeket a $K$-közép módszer nem tud megfelelően kezelni

# Példa EM klaszterezésre (3/2)




11. ábra. Az EM algoritmussal történő klaszterezés eredménye

# Példa EM klaszterezésre (3/3)




12. ábra. A $K$-közép módszerrel történő klaszterezés eredménye

# A keverék modellekkel történő klaszterezés előnyei

- A keverék modellek általánosabbak, mint például a $K$-közép, mert különböző típusú eloszlásokat tudnak használni.
  A (normális eloszláson alapuló) keverék modellek meg tudják találni a különböző méretű és tojásdad alakú klasztereket.
- Könnyű jellemezni a kapott klasztereket, mivel néhány paraméterrel írhatóak le.
- Az adatok egy megfelelő modellhez való illesztése egyszerűsíti az adatok kezelését.
- Sok adatállomány valóban véletlen folyamat eredménye, így ki kell, hogy elégítsék a modellek statisztikai feltevéseit.

# A keverék modellekkel történő klaszterezés hátrányai

- Az EM algoritmus lassú lehet, nem praktikus nagy számú komponenst tartalmazó modellek esetén.
- Nem működik jól, amikor a klaszterek csak néhány adatpontot tartalmaznak, vagy amikor az adatpontok közel kollineárisak.
- Problémát jelenthet a klaszterszám becslése, vagy általánosabban a használt modell pontos formájának meghatározása.
- A keverék modelleknek nehézséget okozhatnak a zaj és a kiugró értékek is.

# Önszervező háló

A **Kohonen-féle önszervező háló** (**SOM – self-organizing map**) vagy **önszervező jellemzőháló** (**SOFM – self-organizing feature map**) egy neurális háló nézőpontú klaszterező és adatvizualizációs módszer.

Neurális hálós eredete ellenére tárgyalható a prototípus-alapú klaszterezés egy változataként.

# Az önszervező háló célja

Más középpont-alapú klaszterező algoritmusokhoz hasonlóan a SOM célja a középpontok (**referencia vektorok**) egy halmazának kijelölése és minden egyes objektum hozzárendelése ahhoz a középponthoz, amelyik az adott objektum legjobb approximációját adja.

# Összehasonlítás más prototípus-alapú módszerekkel

A növekményes $K$-középhez hasonlóan egyesével dolgozzuk fel az adatobjektumokat és a legközelebbi középpontot frissítjük.

A $K$-középpel ellentétben a SOM a középpontok egy topografikus rendezését írja elő és a közeli középpontokat is frissíti.

A SOM nem tartja nyilván egy objektum aktuális klaszterét.

A $K$-középpel ellentétben nem frissíti explicit módon a régi klaszterközéppontot, ha egy objektum klasztert vált. (A régi klaszter lehet az új klaszter szomszédságában, ezért frissítésre kerülhet.)

# Eredmény

A pontok feldolgozása addig folytatódik, amíg el nem érünk valamilyen előre meghatározott határértéket, vagy a középpontok már nem változnak sokat.

A SOM módszer végső eredménye a középpontok halmaza, amely implicit módon definiálja a klasztereket.

Minden egyes klaszter egy adott középponthoz legközelebbi pontokból áll.

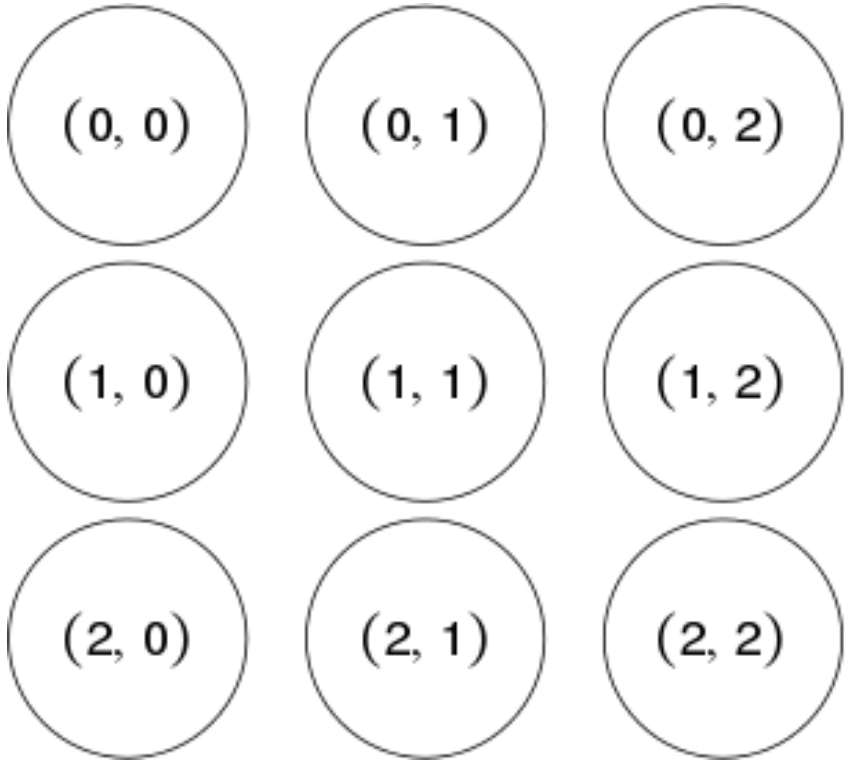
# Az önszervező háló felépítése (1)

A középpontokat négyzetrács szerkezetbe rendezett csomópontok reprezentálják.

Neurális háló terminológiában fogalmazva minden egyes középponthoz egy neuron tartozik.

Kétdimenziós, a középpontokat négyzetrácsba vagy hatszögrácsba szervező SOM-okat vizsgálunk.

# Az önszervező háló felépítése (2)




13. ábra. Kétdimenziós SOM, ahol a középpontokat négyzetrács szerkezetbe rendezett csomópontok reprezentálják. Minden középpontot egy $(i, j)$ koordinátapár azonosít.

# Az önszervező háló felépítése (3)

Egy ilyen hálózatot néha úgy rajzolnak fel, hogy összekötik a szomszédos csomópontokat.

Ez félrevezető lehet, mert egy középpont hatása egy másikra a koordináták, nem pedig a kapcsolatok segítségével van definiálva.

# Az önszervező háló működése

Lényeges eltérés a $K$-középhez vagy más prototípus-alapú módszerekhez képest, hogy a középpontoknak egy előre meghatározott topográfiai rendezési kapcsolatrendszere van.

A tanulási folyamat során a SOM minden adatpontot felhasznál a legközelebbi középpont és a topográfiai rendezés szerint közeli középpontok frissítésénél.

Azok a középpontok, amelyek egymáshoz közel vannak a SOM rácsában, szorosabb kapcsolatban vannak egymással, mint a távolabbi középpontok.

A SOM ilyen módon középpontok egy rendezett halmazát állítja elő minden adatállományra.

# SOM alapalgoritmus

**2. algoritmus.** SOM alapalgoritmus

1: Inicializáljuk a középpontokat
2: **repeat**
3: Válasszuk ki a következő objektumot
4: Határozzuk meg az objektumhoz legközelebbi középpontot
5: Frissítsük a középpontot és a hozzá közeli, azaz adott környezetbe eső középpontokat
6: **until** a középpontok nem változnak sokat vagy elérünk egy küszöbértéket
7: Minden objektumot rendeljünk hozzá a legközelebbi középponthoz és adjuk vissza a középpontokat és a klasztereket

# Kezdőértékadás

- Egy középpont minden egyes koordinátáját válasszuk véletlenszerűen az illető koordináta az adatokban megfigyelt tartományából.
  Nem szükségszerűen a legjobb megoldás, főleg akkor nem, ha gyors konvergenciát szeretnénk.
- A $K$-közép módszer véletlenszerű középpont-választásához hasonló megoldás: a kezdeti középpontokat válasszuk véletlenszerűen az adatpontok közül.

# Az objektum kiválasztása

Felvetődő nehézségek:

- Mivel a konvergenciához sok lépésre lehet szükség, lehet, hogy minden adatobjektumot többször kell használni (különösen akkor, ha kicsi az objektumok száma).
- Ha viszont nagy az objektumok száma, akkor nem minden objektumot szükséges használni.

# Besorolás

Egy távolság-metrika választása szükséges.

Leggyakrabban az euklideszi távolságot vagy a belső szorzatot használjuk.

Belső szorzat alkalmazása esetén az adatvektorokat előzőleg normáljuk, a referenciavektorokat pedig minden egyes lépésben.[3]

[3] Ekkor a belső szorzat alkalmazása ekvivalens a koszinusz mérték használatával.

# Frissítés (1)

Jelölés:

- $k$: a középpontok száma
- $\mathbf{m}_1(t), \ldots, \mathbf{m}_k(t)$: középpontok a $t$-edik lépésben
- $\mathbf{p}(t)$: az aktuális objektum a $t$-edik lépésben
- $\mathbf{m}_j(t)$: a $\mathbf{p}(t)$ objektumhoz legközelebbi középpont

A $(t+1)$-edik időpillanatban az alábbi módon frissítjük a középpontokat:

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + h_{ij}(t)\big(\mathbf{p}(t) - \mathbf{m}_i(t)\big),$$

ahol $i = 1, \ldots, k$.

A frissítés valójában a $j$-edik középpont rácsbeli helyének egy kis környezetére korlátozódik.

# Frissítés (2)

A $\mathbf{p}(t) - \mathbf{m}_i(t)$ különbség hatását határozza meg $h_{ij}(t)$, amelyet úgy kell megválasztani, hogy

1. csökkenjen az idő függvényében,
2. kikényszerítse a szomszédsági hatást, azaz azt, hogy egy objektum hatása az $\mathbf{m}_j(t)$ középponthoz legközelebbi középpontokra legyen a legerősebb. Itt a rácsbeli távolságról van szó!

# Frissítés (3)

Tipikus választás $h_{ij}(t)$-re:

$$h_{ij}(t) = \alpha(t) \exp\left(-\frac{d(\mathbf{r}_i, \mathbf{r}_j)^2}{2\sigma(t)^2}\right)$$

és

$$h_{ij}(t) = \begin{cases} \alpha(t), & \text{ha } d(\mathbf{r}_i, \mathbf{r}_j) \leq \textit{küszöb}, \\ 0, & \text{egyébként.} \end{cases}$$

# Frissítés (4)

- $\alpha(t)$ úgynevezett tanulási tényező paraméter $0 < \alpha(t) < 1$ és értéke monoton csökken az idő függvényében (a konvergencia sebességét szabályozza)
- $\mathbf{r}_i = (x_i, y_i)$ az $i$-edik, $\mathbf{r}_j = (x_j, y_j)$ pedig a $j$-edik középpont rácskoordinátáit megadó kétdimenziós pont
- $d(\mathbf{r}_i, \mathbf{r}_j)$ a két középpont rácsbeli helyének euklideszi távolsága, azaz

$$d(\mathbf{r}_i, \mathbf{r}_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

- $\sigma(t)$ a tipikus Gauss-féle szórás paraméter, ami a szomszédság szélességét szabályozza, azaz kis $\sigma$ kicsi környezetet eredményez, míg egy nagy $\sigma$ széles környezetet

# Frissítés (5)

$\sigma(t)$-re egy tipikus választás

$$\sigma(t) = \sigma_0 \exp\left(-\frac{t}{\tau_1}\right),$$

ahol $\sigma(t)$ kezdeti értéke az algoritmus indításakor $\sigma_0$, $\tau_1$ pedig egy további paraméter.

Hasonlóan választható az $\alpha(t)$ tanulási tényező:

$$\alpha(t) = \alpha_0 \exp\left(-\frac{t}{\tau_2}\right),$$

ahol $\alpha_0$ az $\alpha(t)$ kezdeti értéke az algoritmus indításakor, $\tau_2$ pedig paraméter.

# Leállás

Az iterációnak elméletileg addig kellene folytatódnia, amíg be nem következik a konvergencia, azaz addig, amikor a referencia vektorok már nem változnak, vagy csak nagyon keveset.

A konvergencia sebessége több tényezőtől is függ, így például az adatoktól és $\alpha(t)$-től.

A konvergencia lassú lehet és nem garantált.

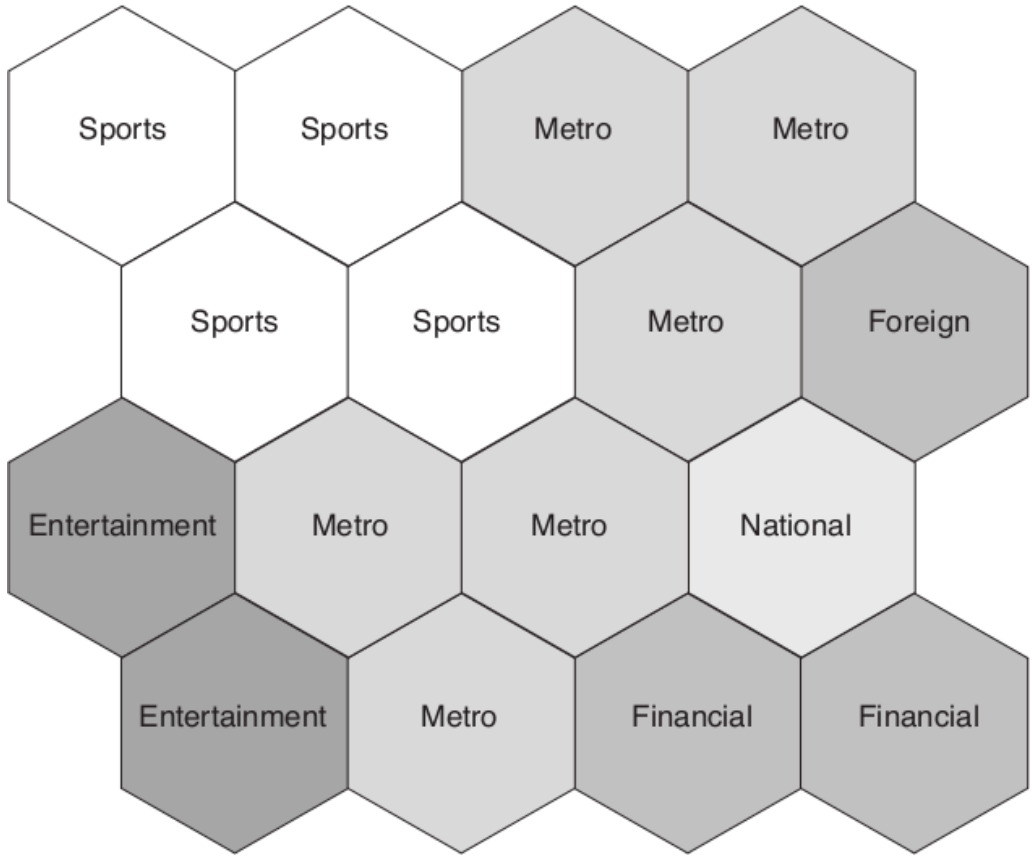
# Példa: dokumentum-adatok (1)

## Példa

SOM alkalmazása egy $4 \times 4$-es hatszögráccsal dokumentum adatokra.

A *Los Angeles Times* 3204 újságcikkének klaszterezése, melyek 6 különböző rovatból származnak: szórakozás (*Entertainment*), pénzügyek (*Financial*), külföld (*Foreign*), helyi hírek (*Metro*), belföld (*National*) és sport (*Sports*).

# Példa: dokumentum-adatok (2)

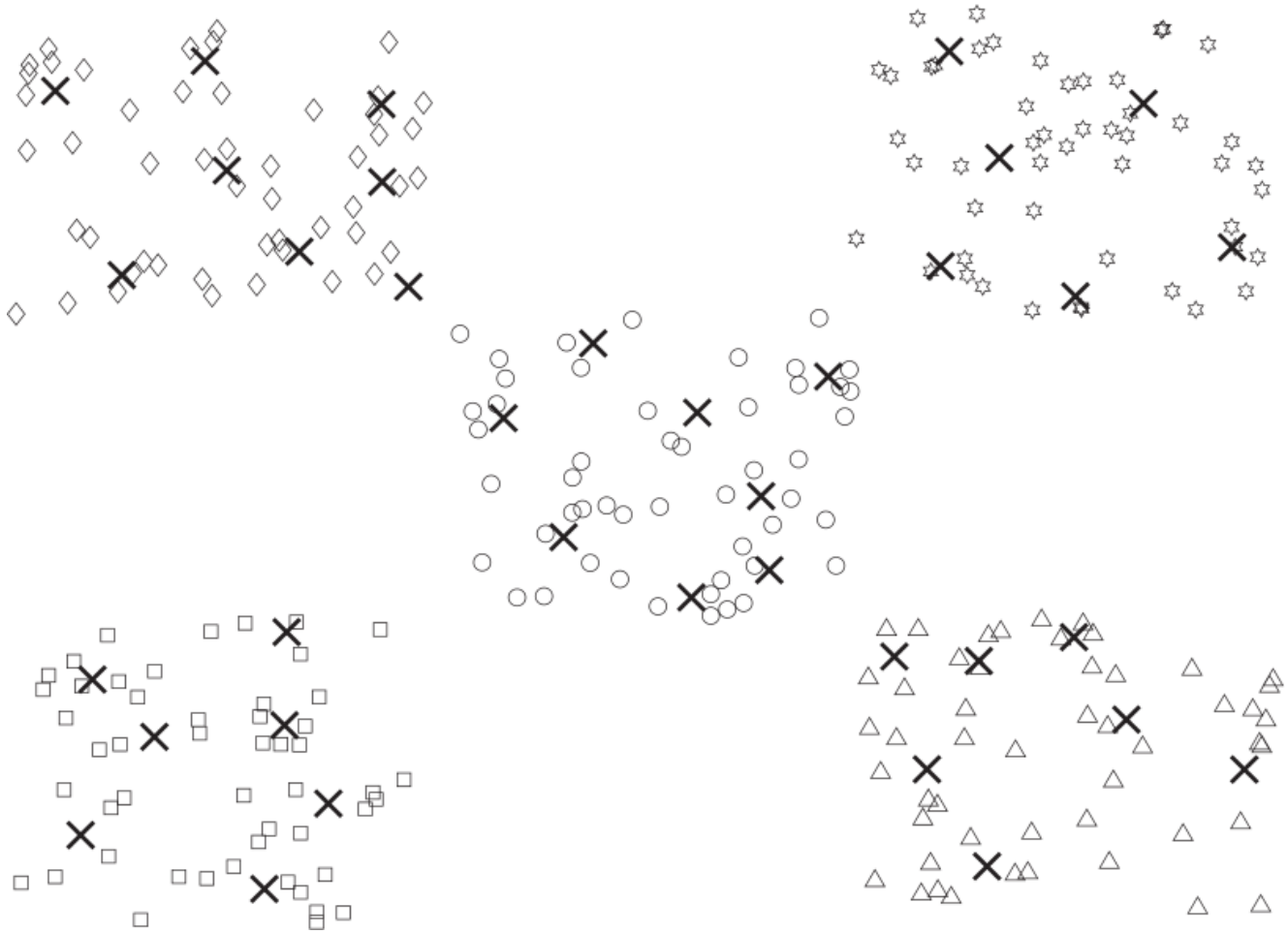



14. ábra. Minden SOM rácscellában a hozzárendelt pontok többségi osztályának címkéje látható

# Példa: kétdimenziós pontok (1)

$6 \times 6$-os négyzetrács alapú SOM alkalmazása és egy kétdimenziós ponthalmazra.

# Példa: kétdimenziós pontok (2)



15. ábra. Sakktábla-szerűen elhelyezkedő és öt osztályba tartozó pontok. A keresztek a SOM referencia vektorait ábrázolják.

# Példa: kétdimenziós pontok (3)

| | | | | | |
|---|---|---|---|---|---|
| rombusz | rombusz | rombusz | hatszög | hatszög | hatszög |
| rombusz | rombusz | rombusz | kör | hatszög | hatszög |
| rombusz | rombusz | kör | kör | kör | hatszög |
| négyzet | négyzet | kör | kör | háromszög | háromszög |
| négyzet | négyzet | kör | kör | háromszög | háromszög |
| négyzet | négyzet | négyzet | háromszög | háromszög | háromszög |

16. ábra. A középpontokhoz tartozó pontok többségi osztálya

# A SOM előnyei

Olyan klaszterező módszer, amely szomszédsági kapcsolatokat kényszerít ki az eredményül kapott klaszterközéppontok között.

Emiatt azok a klaszterek, melyek szomszédok, szorosabb kapcsolatban állnak egymással, mint azok, amelyek nem.

Az ilyen kapcsolatok megkönnyítik a klaszterezés eredményének értelmezését és megjelenítését.

# A SOM korlátai (1)

A következő korlátok némelyike csak akkor érvényes, ha a SOM-ot standard klaszterező módszernek tekintjük, amelynek az adatok valódi klasztereinek megtalálása a célja, nem pedig a klaszterezést az adatok szerkezetének feltárásához felhasználó módszernek!

A felmerülő problémák némelyikével a SOM kiterjesztései vagy a SOM által ihletett klaszterező algoritmusok foglalkoznak.

# A SOM korlátai (2)

- A felhasználónak kell megválasztania a paraméterbeállításokat, a szomszédsági függvényt, a rács típusát és a középpontok számát.
- Egy SOM klaszter gyakran nem felel meg egy természetes klaszternek.
  Egy SOM klaszter bizonyos esetekben magában foglalhat több természetes klasztert, míg más esetekben egy természetes klaszter több SOM klaszterre bomlik fel. Ennek okai:
  - Középpontokból álló rács használata.
  - A SOM más prototípus-alapú klaszterező módszerekhez hasonlóan hajlamos szétvágni vagy egyesíteni a természetes klasztereket, ha azok különböző méretűek, alakúak vagy sűrűségűek.

# A SOM korlátai (3)

- Nincs konkrét célfüggvény.
  A SOM középpontok egy olyan halmazát próbálja megkeresni, amely a legjobban közelíti az adatokat a középpontok közötti topográfiai feltételek figyelembe vétele mellett, de a sikeresség ezen a téren nem fejezhető ki egy függvénnyel.
  Ez megnehezítheti különböző SOM klaszterezési eredmények összehasonlítását.
- A konvergencia nem garantált, bár a gyakorlatban tipikusan konvergál a módszer.

# Tartóvektor klaszterezés (1)

Alapötlet:

- Transzformáljuk az adatokat RBF kernelfüggvénnyel egy nagy dimenziószámú tulajdonságtérbe.
- Határozzuk meg azt a legkisebb olyan gömböt a tulajdonságtérben, mely tartalmazza valamennyi adat képét.
- A tulajdonságtérbeli gömb felszínének visszavetítése az adattérbe olyan kontúrokat eredményez, melyek bekerítik az adatokat. A kontúrokat tekintjük klaszterhatároknak.

# Tartóvektor klaszterezés (2)

Az RBF kernelfüggvény szélesség paraméterével szabályozható a kontúrok száma.

A puha margó megfogalmazás $C$ paramétere teszi lehetővé kiugró értékek kezelését.

# A feladat megfogalmazása optimalizálási problémaként (1)

Jelölés:

- $\{ \mathbf{x}_1, \ldots, \mathbf{x}_N \} \subset \mathbb{R}^n$: a klaszterezendő adatok
- $\mathbf{\Phi}(\mathbf{x})$: az $\mathbf{x} \in \mathbb{R}^n$ vektor az RBF kernelfüggvény által meghatározott tulajdonságtérbeli képe
- $R$: az adatok képét tartalmazó tulajdonságtérbeli gömb sugara
- $\mathbf{a}$: az adatok képét tartalmazó tulajdonságtérbeli gömb középpontja

# A feladat megfogalmazása optimalizálási problémaként (2)

Feltételes optimalizálási probléma:

$$\min_{R} R^2$$

$$\text{feltéve, hogy} \qquad \|\mathbf{\Phi}(\mathbf{x}_i) - \mathbf{a}\|^2 \leq R^2, \qquad i = 1, \ldots, N.$$

# A feladat megfogalmazása optimalizálási problémaként (3)

Feltételes optimalizálási probléma (puha margó):

$$\min_{R} R^2 + C \sum_{i=1}^{N} \xi_i$$

feltéve, hogy $\|\mathbf{\Phi}(\mathbf{x}_i) - \mathbf{a}\|^2 \leq R^2 + \xi_i, \quad \xi_i \geq 0, \quad i = 1, \ldots, N.$

A célfüggvényben szereplő $C$ egy olyan pozitív értékű paraméter, melyet a felhasználó kell, hogy meghatározzon.

# Az optimalizálási probléma megoldása (1)

A probléma primál Lagrange-függvénye:

$$L_P = R^2 - \sum_{i=1}^{N} \lambda_i \left( R^2 + \xi_i - \|\mathbf{\Phi}(\mathbf{x}_i) - \mathbf{a}\|^2 \right) - \sum_{i=1}^{N} \mu_i \xi_i + C \sum_{i=1}^{N} \xi_i,$$

ahol $\lambda_i$ és $\mu_i$ paraméterek, úgynevezett Lagrange-multiplikátorok, melyekre $\lambda_i \geq 0$ és $\mu_i \geq 0$ minden $i = 1, \ldots, N$ esetén.

# Az optimalizálási probléma megoldása (2)

Teljesülnek az alábbi, ún. Karush-Kuhn-Tucker feltételek:

$$\lambda_i \left( R^2 + \xi_i - \|\mathbf{\Phi}(\mathbf{x}_i) - \mathbf{a}\|^2 \right) = 0, \tag{6}$$

$$\mu_i \xi_i = 0, \tag{7}$$

ahol $i = 1, \ldots, N$.

# Az optimalizálási probléma megoldása (3)

Az $L_P$ Lagrange-függvény $R$, $\mathbf{a}$ és $\xi_i$ szerinti elsőrendű deriváltjait nullává téve a következő egyenleteket kapjuk:

$$\frac{\partial L_P}{\partial R} = 0 \Longrightarrow \sum_{i=1}^{N} \lambda_i = 1 \tag{8}$$

$$\frac{\partial L_P}{\partial \mathbf{a}} = 0 \Longrightarrow \mathbf{a} = \sum_{i=1}^{N} \lambda_i \mathbf{\Phi}(\mathbf{x}_i) \tag{9}$$

$$\frac{\partial L_P}{\partial \xi_i} = 0 \Longrightarrow \lambda_i + \mu_i = C \tag{10}$$

# Az optimalizálási probléma megoldása (4)

Az előbbiek felhasználásával az alábbi duális Lagrange-függvény alkotható:

$$L_D = \sum_{i=1}^{N} \lambda_i \mathbf{\Phi}(\mathbf{x}_i)^{\mathrm{T}} \mathbf{\Phi}(\mathbf{x}_i) - \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j \mathbf{\Phi}(\mathbf{x}_i)^{\mathrm{T}} \mathbf{\Phi}(\mathbf{x}_j).$$

A korábbiakhoz hasonlóan $L_D$ maximalizálása a feladat, feltéve hogy teljesülnek az alábbiak:

1. $\sum_{i=1}^{N} \lambda_i = 1$,
2. $0 \leq \lambda_i \leq C$ minden $i = 1, \ldots, N$ esetén.

# Az optimalizálási probléma megoldása (5)

Ha

$$K(\mathbf{u}, \mathbf{v}) = \mathbf{\Phi}(\mathbf{u})^{\mathrm{T}} \mathbf{\Phi}(\mathbf{v})$$

kernelfüggvény, akkor az $L_D$ duális Lagrange-függvény a következő alakba írható:

$$L_D = \sum_{i=1}^{N} \lambda_i K(\mathbf{x}_i, \mathbf{x}_i) - \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j K(\mathbf{x}_i, \mathbf{x}_j).$$

## Megjegyzés

A továbbiakban a

$$K(\mathbf{u}, \mathbf{v}) = \exp\Big(-\gamma \left\|\mathbf{u} - \mathbf{v}\right\|^2\Big),$$

radiális bázisfüggvény kernelfüggvényt használjuk, ahol $\gamma$ parméter.

# Tartóvektorok (1)

A (6) KKT feltételből következik, hogy minden olyan $\mathbf{x}_i$ vektor esetén, amelyre $\xi_i > 0$ és $\lambda_i > 0$ teljesül, a vektor $\mathbf{\Phi}(\mathbf{x}_i)$ képe a tulajdonságtérbeli gömbön kívül helyezkedik el.

A (6) KKT feltétel szerint egy ilyen $\mathbf{x}_i$ vektorhoz tartozó $\mu_i$ Lagrange-multiplikátor értéke 0. Ekkor viszont a (10) egyenlet miatt $\lambda_i = C$ teljesül.

Egy ilyen $\mathbf{x}_i$ vektort **korlátos tartóvektor**nak (**BSV – bounded support vector**) nevezzük.

# Tartóvektorok (2)

A tulajdonságtérbeli gömb felszínén vagy belsejében helyezkedik el minden olyan $\mathbf{x}_i$ vektor képe, amelyre $\xi_i = 0$ teljesül.

A (6) KKT feltételből következik, hogy minden olyan $\mathbf{x}_i$ vektor esetén, amelyre $\xi_i = 0$ és $0 < \lambda_i < C$ teljesül, a vektor $\mathbf{\Phi}(\mathbf{x}_i)$ képe a tulajdonságtérbeli gömb felszínén helyezkedik el. Egy ilyen vektort nevezünk **tartóvektor**nak.

# Tartóvektorok (3)

Összefoglalva:

- A tartóvektorok pontosan a klaszterhatárokon – azaz a tulajdonságtérbeli gömb felszínén – helyezkednek el.
- A korlátos tartóvektorok a klaszterhatárokon kívül – azaz a tulajdonságtérbeli gömbön kívül – helyezkednek el.
- Az összes többi pont a klaszterhatárokon belül – azaz a tulajdonságtérbeli gömb belsejében – található.

## Megjegyzés

Ha $C \geq 1$, akkor $\sum_{i=1}^{N} \lambda_i = 1$ miatt nincs egyetlen korlátos tartóvektor sem.

# A klaszterek határvonalai (1)

Vezessük be az alábbi jelölést egy $\mathbf{x} \in \mathbb{R}^n$ vektor a tulajdonságtérbeli képének és a gömb $\mathbf{a}$ középpontjának távolságára:

$$R^2(\mathbf{x}) = \|\mathbf{\Phi}(\mathbf{x}) - \mathbf{a}\|^2 .$$

Ekkor a (9) egyenlet felhasználásával

$$R^2(\mathbf{x}) = K(\mathbf{x}, \mathbf{x}) - 2\sum_{i=1}^{N} \lambda_i K(\mathbf{x}_i, \mathbf{x}) + \sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i \lambda_j K(\mathbf{x}_i, \mathbf{x}_j).$$

# A klaszterek határvonalai (2)

A gömb sugara[4]:

$$R = \{ R(\mathbf{x}_i) \mid \mathbf{x}_i \text{ tartóvektor} \}.$$

Az adattérbeli klaszterek határvonalait az alábbi halmaz definiálja:

$$\{ \mathbf{x} \mid R(\mathbf{x}) = R \}. \tag{11}$$

## Megjegyzés

A határvonalak adattérbeli alakja a $\gamma$ és $C$ paraméterekkel szabályozható.

[4] Ne feledjük, hogy minden tartóvektor pontosan a tulajdonságtérbeli gömb felszínén helyezkedik el.

# Pontok hozzárendelése klaszterekhez (1)

A (11) képlet a klaszterek határvonalait definiálja.

Hogyan lehet meghatározni, hogy egy vektor ténylegesen melyik klaszterhez tartozik?

# Pontok hozzárendelése klaszterekhez (2)

Ehhez egy geometriai megközelítést használunk, amely az alábbi megfigyelésen alapul:

Tekintsük két olyan pontot, melyek különböző klaszterekhez tartoznak! Bármely ezeket összekötő útvonal tulajdonságtérbeli képe kilép a gömbből.

Ez azt jelenti, hogy egy ilyen útvonalon vannak olyan $\mathbf{y}$ pontok, amelyekre $R(\mathbf{y}) > R$ teljesül.

# Pontok hozzárendelése klaszterekhez (3)

Az alábbi szomszédsági mátrixot azokra az $\mathbf{x}_i$ és $\mathbf{x}_j$ pontpárokra definiáljuk, melyek tulajdonságtérbeli képei a gömb felszínén vagy belsejében helyezkednek el:

$$A_{ij} = \begin{cases} 1, & \text{ha az } \mathbf{x}_i \text{ és } \mathbf{x}_j \text{ pontokat összekötő szakasz minden } \mathbf{y} \text{ pontjára } R(\mathbf{y}) \leq R \text{ teljesül,} \\ 0, & \text{egyébként.} \end{cases}$$

Ezután az $A$ szomszédsági mátrix által meghatározott gráf összefüggő komponenseit tekintjük klasztereknek.

# Pontok hozzárendelése klaszterekhez (4)

## Megjegyzés

A szomszédsági mátrix megadható az alábbi alakban is:

$$A_{ij} = \begin{cases} 1, & \text{ha } R(\mathbf{x}_i + \delta(\mathbf{x}_j - \mathbf{x}_i)) \leq R \text{ minden } \delta \in [0,1] \text{ esetén,} \\ 0, & \text{egyébként.} \end{cases}$$

## Megjegyzés

Egy szakasz ellenőrzése adott számú pont mintavételezésével implementálható (ehhez választható például 20 pont a szakaszon).

# Pontok hozzárendelése klaszterekhez (5)

## Megjegyzés

A korlátos tartóvektorokat nem osztályozza a bemutatott eljárás, mivel ezek tulajdonságtérbeli képei a gömbön kívül helyezkednek el.

A felhasználó dönthet úgy, hogy a korlátos tartóvektorokat a legközelebbi klaszterhez rendeli hozzá.

# Klaszterezés korlátos tartóvektorok nélkül

Ha $C = 1$, akkor egyetlen adatvektor sem korlátos tartóvektor.

Ez akkor akkor ad jó eredményt, ha nincsenek kiugró értékek az adatokban.

# Klaszterezés korlátos tartóvektorokkal (1)

A korlátos tartóvektorok számát a $C$ paraméter szabályozza.

Ha $\mathbf{x}_s$ egy korlátos tartóvektor, akkor definíció szerint $\lambda_s = C$ teljesül.

Mivel $\sum_{i=1}^{N} \lambda_i = 1$ és $0 \leq \lambda_i \leq C$ minden $i = 1, \ldots, N$ esetén, a korlátos tartóvektorok számára ($n_{BSV}$) az alábbi felső korlát adható:

$$n_{BSV} < 1/C.$$

A gyakorlatban a korlátos tartóvektorok arányára adnak felső korlátot, amelyet $p$ jelöl:

$$p = \frac{1}{NC}.$$

# Klaszterezés korlátos tartóvektorokkal (2)

Akkor szükséges korlátos tartóvektorok használata:

- ha az adatokban jelen vannak kiugró értékek,
- ha a klaszterek átfedőek.

# Klaszterezés korlátos tartóvektorokkal (3)

## Megjegyzés

Ha a klaszterek nagymértékben átfedik egymást, akkor az eredményt némileg eltérően kell értelmezni.

Nagymértékű átfedés esetén nagyszámú korlátos tartóvektor szükséges, a kontúrok ekkor kisszámú adatpontot kerítenek be.

A kontúrok ilyenkor a klaszterek magjait azonosítják, a pontok – köztük a korlátos tartóvektorok – klaszterekhez való hozzárendelése a klasztermagoktól számított távolságuk alapján történhet.

# Megfelelő $\gamma$ és $p$ választása (1)

A paraméterek iteratív beállítása javasolt.

# Megfelelő $\gamma$ és $p$ választása (2)

Kiindulásként $\gamma$ legyen kicsi és legyen $C = 1$ (azaz $p = 1/N$).

Alkalmas kezdőérték $\gamma$-ra

$$\gamma = \frac{1}{\max_{i,j} \|\mathbf{x}_i - \mathbf{x}_j\|^2}.$$

Ekkor egyetlen, az összes pontot tartalmazó klasztert kapunk eredményül.

# Megfelelő $\gamma$ és $p$ választása (3)

$\gamma$ értékének növelésével a klaszterek szétválása figyelhető meg.

Ha egyetlen vagy néhány pontból álló klaszterek válnak le, vagy pedig nagyon egyenetlenné válnak a klaszterhatárok, akkor $p$ növelése szükséges, amely lehetővé teszi korlátos tartóvektorok használatát.

# Megfelelő $\gamma$ és $p$ választása (4)

Általában az a jó, ha minél kisebb a tartóvektorok száma: kevés tartóvektor esetén a klaszterhatárok simák lesznek.

Ha túl sok a tartóvektor, akkor $p$ növelése során sok tartóvektor korlátos tartóvektorrá válhat, amely simább klaszterhatárokat eredményez.

Úgy kell tehát $\gamma$ és $p$ értékét beállítani, hogy a lehető legkevesebb tartóvektor legyen, ugyanakkor a klaszterek megfelelően szétváljanak.

# A tartóvektor klaszterezés előnyei

- Nem él feltevéssel a klaszterek számával és alakjával kapcsolatban.
- Képes a zaj és a kiugró értékek kezelésére.
- Képes átfedő klaszterek kezelésére. (Nagymértékben átfedő klaszterek esetén azonban inkább csak viszonylag kisméretű klasztermagok azonosítására használható.)
- Tetszőleges alakú klaszterhatárokat képes előállítani.

# BIRCH – Balanced Iterative Reducing and Clustering using Hierarchies[5]

Nagyon hatékony klaszterező módszer euklideszi adatokra.

Különösen alkalmas nagyon nagyméretű adatállományokhoz.

Egy menetben hatékonyan tudja klaszterezni az adatokat, ez a klaszterezés további menetekben javítható.

Hatékonyan tudja kezelni a kiugró értékeket.

Növekményes módszerként használható adatfolyamok (*streaming data*) feldolgozásához. Ilyenkor egyetlen menetben történik az adatok beolvasása és klaszterezése.

[5] Kiegyensúlyozott iteratív csökkentés és klaszterezés hierarchiákkal

# Klaszterező jellemző fogalma

Minden egyes klaszter jellemzése egy $(N, LS, SS)$ rendezett hármassal történik, ahol

- $N$ a klaszter pontjainak a száma,
- $LS$ a klaszter pontjainak összege,
- $SS$ a klaszter pontjainak négyzetösszege.

A fenti rendezett hármasokat nevezzük **klaszterező jellemző**nek (**CF – clustering feature**).

# A klaszterező jellemző tulajdonságai

**Tétel (A klaszterező jellemző additivitása)**

*Ha* $CF_1 = (N_1, LS_1, SS_1)$ *és* $CF_2 = (N_2, LS_2, SS_2)$ *két diszjunkt klaszter klaszterező jellemzői, akkor a két klaszter egyesítésével kapott klaszter klaszterező jellemzője*

$$CF_1 + CF_2 = (N_1 + N_2, LS_1 + LS_2, SS_1 + SS_2).$$

A klaszterező jellemző komponensei közönséges statisztikai mennyiségek, melyeket növekményesen is ki lehet számolni.

# A klaszterező jellemző felhasználási lehetőségei

A klaszterező jellemző alapján számos fontos mennyiséget ki lehet számolni, mint például egy klaszter középpontját és varianciáját (szórását). A varianciát a klaszterek átmérőjének mérőszámaként használjuk.

A klaszterező jellemzők alapján kiszámítható klaszterek távolsága is. A BIRCH a klaszterek számos távolságmértékét definiálja, de mindegyiket ki lehet számolni a klaszterező jellemzőt alkotó alapstatisztikák segítségével.

# CF fa

A **CF fa** (**CF tree**) a klaszterező jellemzőket tároló kiegyensúlyozott fa, a B-fához hasonló adatszerkezet.

# CF fa: belső csúcsok

Minden belső csúcs legfeljebb $B$ számú, $[CF_i, \text{gyerek}_i]$ alakú bejegyzést tartalmaz, ahol

- $\text{gyerek}_i$ egy mutató az $i$-edik gyerek csúcsra,
- $CF_i$ az ezen gyerek csúcs által reprezentált alklaszter klaszterező jellemzője.

Egy belső csúcs a bejegyzései által reprezentált alklaszterekből álló klaszternek felel meg.

# CF fa: levélcsúcsok

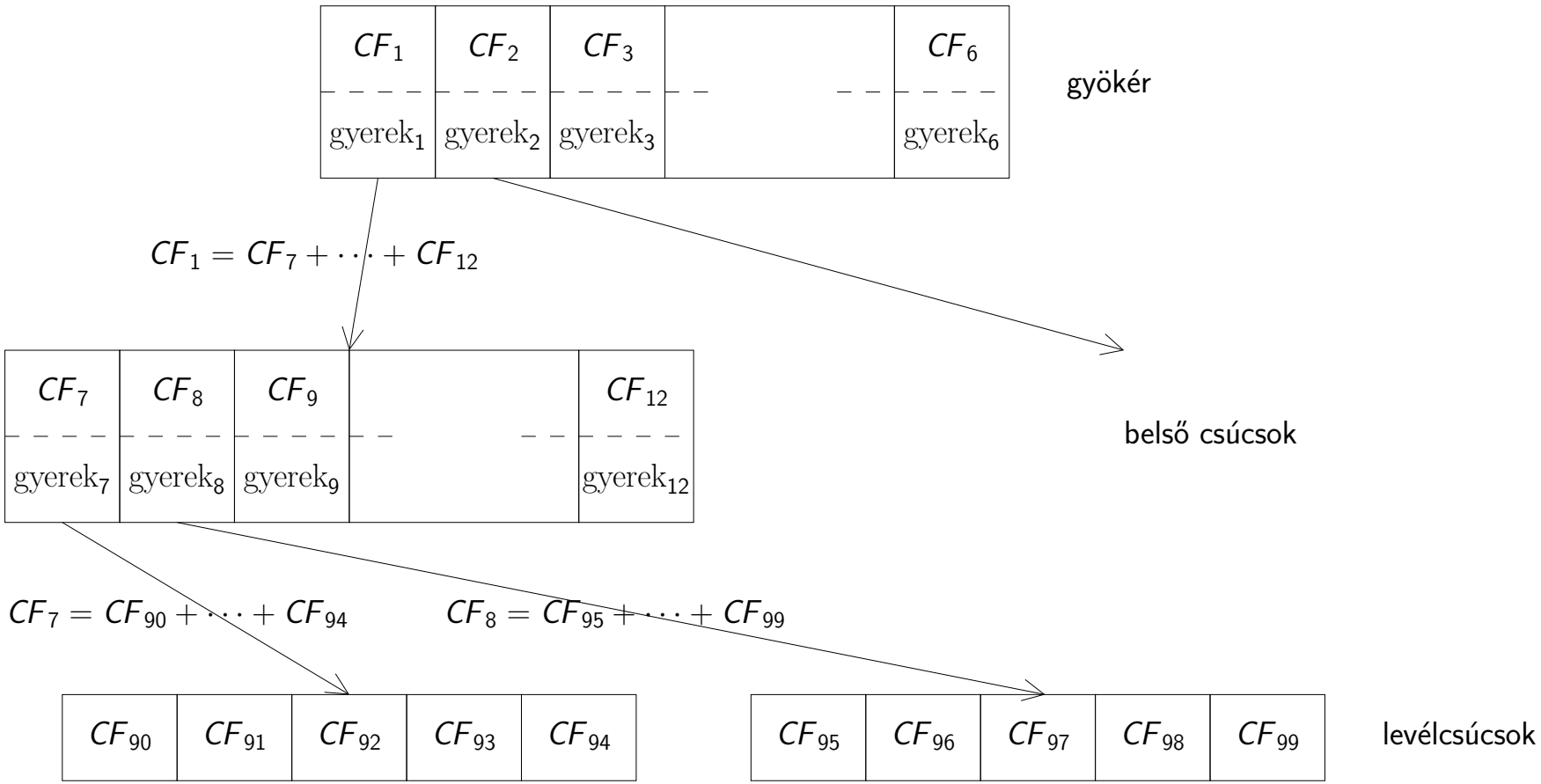
Minden levélcsúcs legfeljebb $L$ számú, $[CF_i]$ alakú bejegyzést tartalmaz.

Minden klaszterező jellemző egy korábban már megvizsgált ponthalmazt ábrázol.

Egy levélcsúcs a bejegyzései által reprezentált alklaszterekből álló klaszternek felel meg.

A levélcsúcsok minden egyes bejegyzésére az a megszorítás vonatkozik, hogy a reprezentált klaszter átmérője kisebb egy $T$ küszöbértéknél.

# Példa CF fára




17. ábra. Egy CF fa részlete ($B = 6$ és $L = 5$)

# CF fa: paraméterek (1)

Követelmény, hogy minden csúcs elférjen a memóriában egy $P$ méretű lapon.

Az adatok dimenziószáma (és típusa) alapján határozható meg adott lapméret esetén $B$ és $L$ értéke.

# CF fa: paraméterek (2)

A $T$ küszöbérték paraméter módosításával szabályozható a fa magassága: minél nagyobb $T$ értéke, annál kisebb a magasság.

$T$ a klaszterezés finomságát szabályozza, azaz annak mértékét, hogy mennyire csökkentjük az eredeti adatállomány adatait.

A cél az, hogy a CF fát a $T$ paraméter megfelelő beállításával a központi memóriában tudjuk tárolni.

# CF fa: az adatok tömör reprezentációja

A CF fa az eredeti adatállomány egy nagyon tömör reprezentációja, mert a levélcsúcsok bejegyzései nem egyetlen adatpontnak felelnek meg, hanem több pontot magukban foglaló, a $T$ küszöbértéknél kisebb átmérőjű alklaszteréknek.

# CF fa építése

Egy CF fát építünk az adatok beolvasása során.

Minden egyes adatpont esetén az alábbi lépésekben történik a CF fa módosítása:

1. A megfelelő levélcsúcs meghatározása
2. A levélcsúcs módosítása
3. A levélcsúcsba vezető út módosítása
4. Egyesítés

# CF fa építése: levélcsúcs meghatározása

Bejárjuk a CF fát a gyökérből indulva és minden szinten a legközelebbi gyerek csúcsot választva. Ne feledjük, hogy minden gyerek csúcs egy klaszternek felel meg!

# CF fa építése: levélcsúcs és a levélcsúcsba vezető út módosítása (1)

Ha az aktuális adatponthoz azonosítottuk a megfelelő levélcsúcsot, akkor keressük meg a levél az adatponthoz legközelebbi alklaszterét reprezentáló bejegyzését.

# CF fa építése: levélcsúcs és a levélcsúcsba vezető út módosítása (2)

Vizsgáljuk meg, hogy az aktuális adatpont hozzáadása a kiválasztott klaszterhez a $T$ küszöbértéknél nagyobb átmérőjű klasztert eredményez-e.

- Ha nem, akkor az adatpontot hozzáadjuk a kiválasztott klaszterhez a CF információk frissítésével, majd a levéltől a gyökérig minden csúcs klaszterinformációját frissítjük.
- Ha igen, akkor:
  - Ha a levélcsúcs még nem telt meg, akkor egy új bejegyzést hozunk létre, majd a levéltől a gyökérig minden csúcs klaszterinformációját frissítjük (pontosan úgy, mint az előző esetben).
  - Egyébként a levélcsúcsot fel kell bontani.

# CF fa építése: levélcsúcs és a levélcsúcsba vezető út módosítása (3)

A levélcsúcsot úgy bontjuk fel két levélcsúcsra, hogy a két legtávolabbi bejegyzést (alklasztert) tekintjük magnak, a fennmaradó bejegyzéseket pedig aszerint osztjuk el a két új levélcsúcs között, hogy melyikük tartalmazza a közelebbi mag-klasztert.

Ha felbontottuk levélcsúcsot, a szülő csúcsot is frissítjük, valamint felbontjuk, ha szükséges, azaz ha megtelt. Ez a folyamat egészen a gyökérig folytatható.

# CF fa építése: egyesítés (1)

A csúcsok felbontását a lapméret okozza, amely független az adatoktól.

A BIRCH ezért minden felbontást egy egyesítési lépéssel folytat.

# CF fa építése: egyesítés (2)

Annál a belső csúcsnál, ahol a felbontás megállt, megkeressük a két legközelebbi bejegyzést. Ha ez a pár nem a felbontásból származó két bejegyzés, akkor kísérletet teszünk a bejegyzések és a hozzájuk tartozó gyerek csúcsok egyesítésére.

Ha az egyesítés során kapott új csúcs bejegyzései nem férnek el egy lapon, akkor az új csúcsot ismét fel kell bontani. Ha a felbontásnál valamelyik csúcs megtelik a bejegyzések elosztás során, akkor a maradékot rendeljük a másik csúcshoz.

Az egyesítés jobb helykihasználást eredményez, illetve javítja a bejegyzések eloszlását a két legközelebbi gyerek között.

# A CF fa problémái

- Mivel a csúcsok bejegyzéseinek számát a lapméret korlátozza, egy csúcs nem mindig felel meg egy természetes klaszternek.
- A csúcsok felbontása azt eredményezheti, hogy különböző csúcsokhoz kerülhet két olyan alklaszter, amelyeknek egy klaszterben lenne a helyük.
- Ugyanabba a csúcsba kerülhet két olyan alklaszter, amelyeknek nem egy klaszterben van a helyük.
- Ha ugyanaz az adatpont többször is beszúrásra kerül, különböző levélcsúcs bejegyzésekhez kerülhet.

# Kiugró értékek kezelése

Ha a fát újra kell építeni, mert megtelt a memória, akkor a kiugró értékeket opcionálisan ki lehet írni lemezre.

Kiugró értéknek egy olyan csúcsot tekintünk, amelynek az átlagosnál sokkal kevesebb adatpontja van.

A folyamat bizonyos pontjain átvizsgáljuk a kiugró értékeket, hogy beépíthetők-e a fába anélkül, hogy annak mérete növekedjen. Ha igen, akkor beépítjük, ha nem, akkor pedig töröljük őket.

# BIRCH algoritmus

**3. algoritmus.** BIRCH

1: **Töltsük be az adatokat a memóriába az adatokat összefoglaló CF fa létrehozásával.**
2: **Építsünk egy kisebb CF fát, ha ez szükséges a 3. fázishoz.** Növeljük $T$-t, majd szúrjuk be újra a levélcsúcs bejegyzéseket (klasztereket). Mivel $T$-t növeltük, néhány klasztert egyesítünk.
3: **Végezzük el a globális klaszterezést.** Különböző globális klaszterező módszereket (a klaszterek közötti páronkénti távolságokon alapuló klaszterezéseket) lehet használni. Egy összevonó hierarchikus módszert választottunk azonban. Mivel a klaszterező jellemzők olyan összefoglaló információkat tárolnak, amelyek bizonyos fajta klaszterezésekhez szükségesek, a globális klaszterező algoritmust úgy lehet használni, mintha a CF által reprezentált klaszter minden pontjára alkalmaztuk volna.
4: **Rendezzük át az adatpontokat a 3. lépésben talált klaszterközéppontok között és tárjunk fel így egy új klaszterhalmazt.** Ez megold bizonyos problémákat, amelyek felléphetnek a BIRCH első fázisában. Ezt a fázist többször ismételve a folyamat egy lokálisan optimális megoldáshoz konvergál.

# Hivatkozások


Tartóvektor klaszterezés:

- Asa Ben-Hur, David Horn, Hava T. Siegelmann, and Vladimir Vapnik. "Support Vector Clustering". In: *Journal of Machine Learning Research* 2 (Dec. 2001), pp. 125–137. URL: http://www.jmlr.org/papers/volume2/horn01a/rev1/horn01ar1.pdf

BIRCH:

- Tian Zhang, Raghu Ramakrishnan, and Miron Livny. "BIRCH: an efficient data clustering method for very large databases". In: *SIGMOD96*. ACM Press, June 1996, pp. 103–114. DOI: 10.1145/233269.233324. URL: http://dx.doi.org/10.1145/233269.233324