Együttes módszerek

Ispány Márton és Jeszenszky Péter

2016. október 18.

Tartalom

Bevezetés

Zsákolás (bagging)

Gyorsítás (boosting)
- AdaBoost

Véletlen erdők (random forests)

Hibajavító kimenet kódolás (error-correcting output coding)

Hivatkozások

Alapötlet

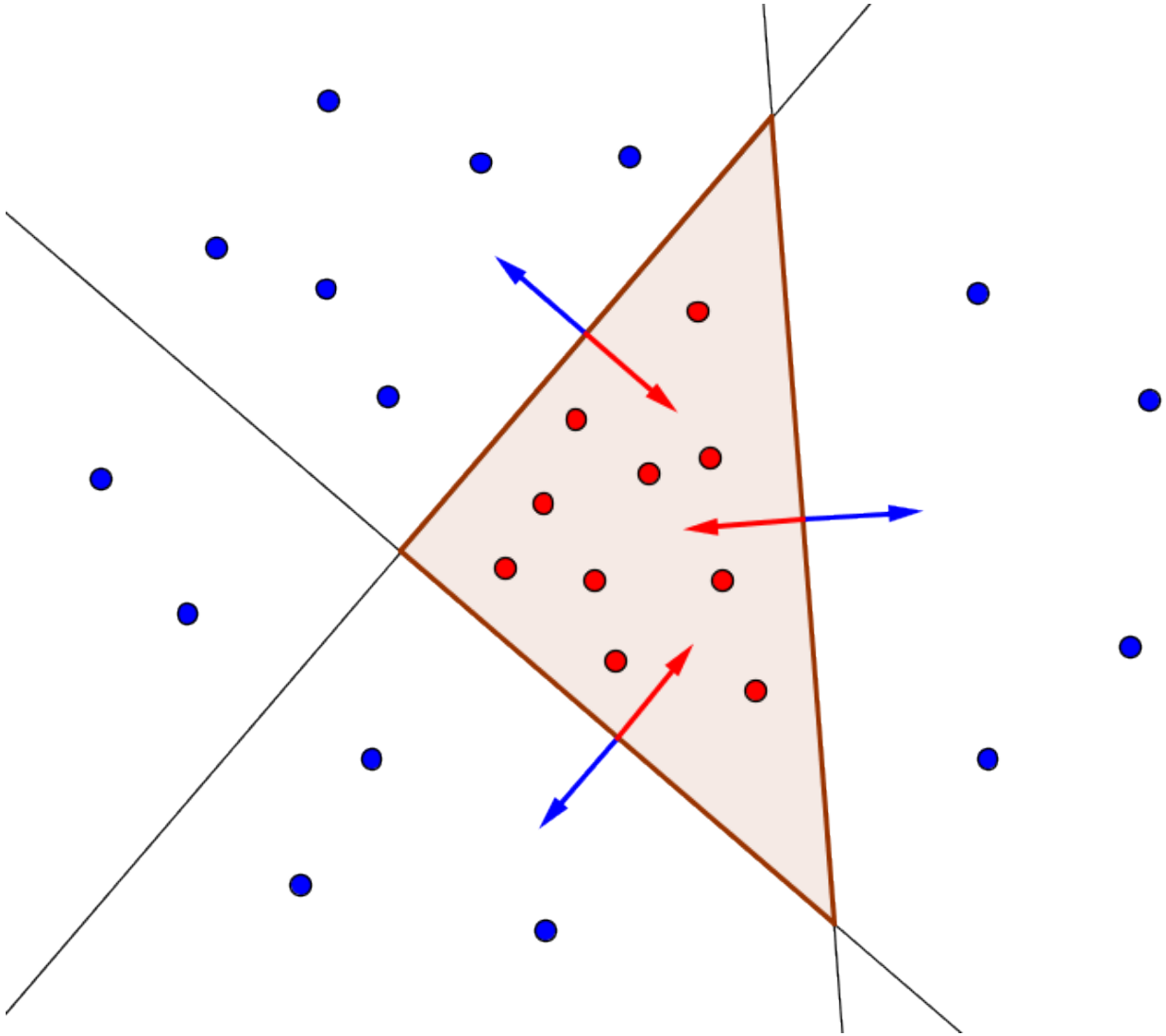
Az **együttes** (**ensemble**) vagy **kombinált osztályozó** (**classifier combination**) módszerek több osztályozó előrejelzéseit összesítik az osztályozás pontosságának növelése érdekében.

Alaposztályozók (**base classifiers**) egy halmazának létrehozása a tanulóadatokból.

Úgy történik az osztályozás az együttessel, hogy szavazást tartunk az egyes alaposztályozók előrejelzésein.[1]

[1] A szavazás lehet például **többségi szavazás**.

Együttes osztályozó döntési határa



Eredményesség (1)

Példa

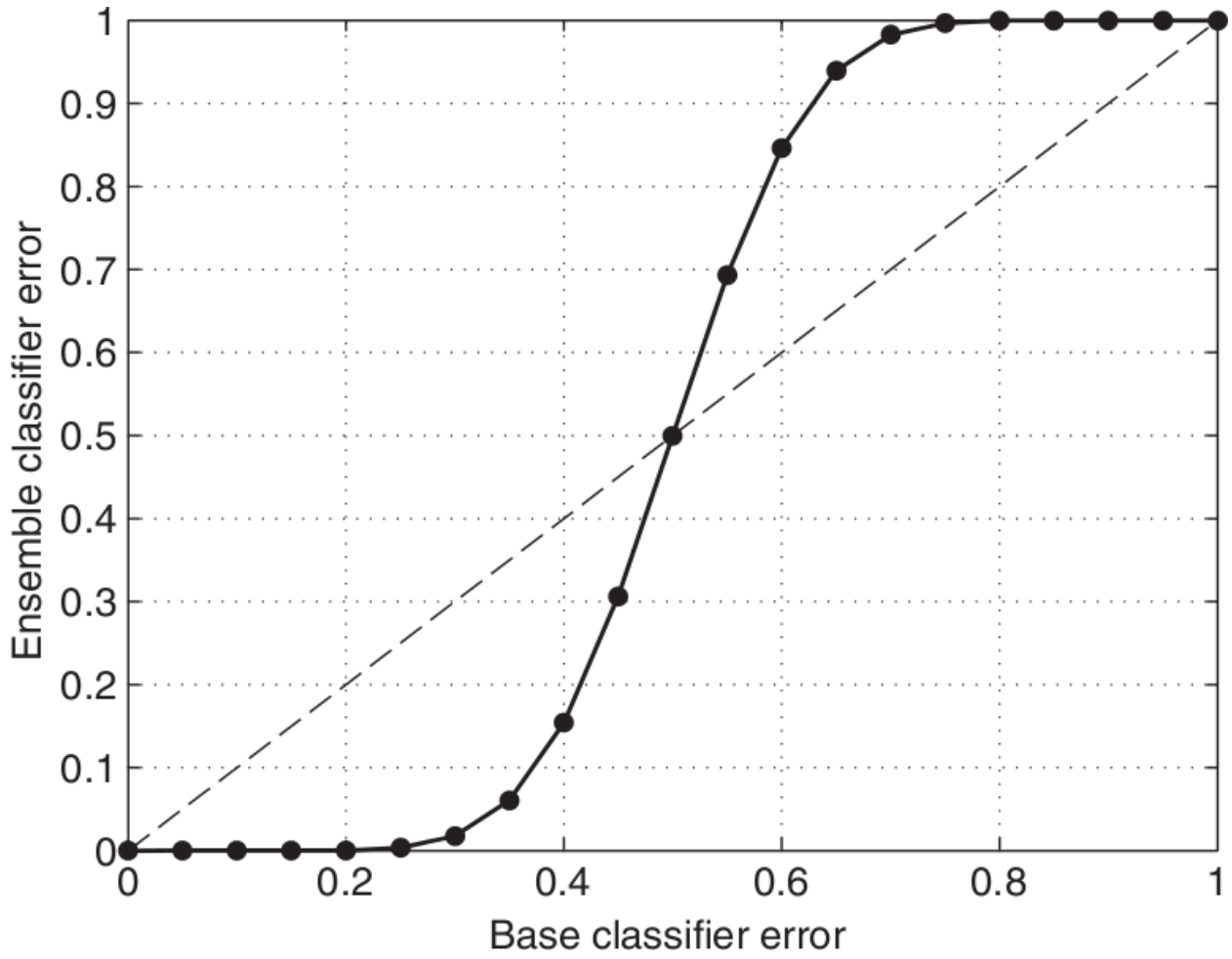
Vegyük huszonöt bináris osztályozó egy együttesét, amelyek mindegyikének $\epsilon = 0{,}35$ a hibaaránya.[2]

Ha az alaposztályozók függetlenek – azaz a hibáik korrelálatlanok –, akkor az együttes csak akkor ad hibás előrejelzést, ha az alaposztályozók több mint fele helytelenül prediktál. Ekkor az együttes osztályozó hibaaránya

$$\epsilon_{\text{együttes}} = \sum_{i=13}^{25} \binom{25}{i} \epsilon^i (1-\epsilon)^{25-i} = 0{,}06.$$

[2] Egy osztályozási modell hibaaránya a hibás előrejelzések számának és az összes előrejelzés számának hányadosa.

Eredményesség (2)



A 25 alaposztályozóból álló együttes hibaaránya az alaposztályozók hibaarányának függvényében.

Eredményesség (3)

Két szükséges feltétel ahhoz, hogy egy együttes osztályozó jobb eredményt érjen el, mint egyetlen osztályozó:

- az alaposztályozók függetlenek kell, hogy legyenek egymástól,
- az alaposztályozók jobb eredmény kell, hogy elérjenek, mint egy véletlen találgatást végző osztályozó.

A gyakorlatban bonyolult az alaposztályozók közötti teljes függetlenség biztosítása, de a gyakorlati tapasztalat azt mutatja, hogy gyengén korrelált alaposztályozók mellett is jobb osztályozási pontosságot kapunk.

Eredményesség (4)

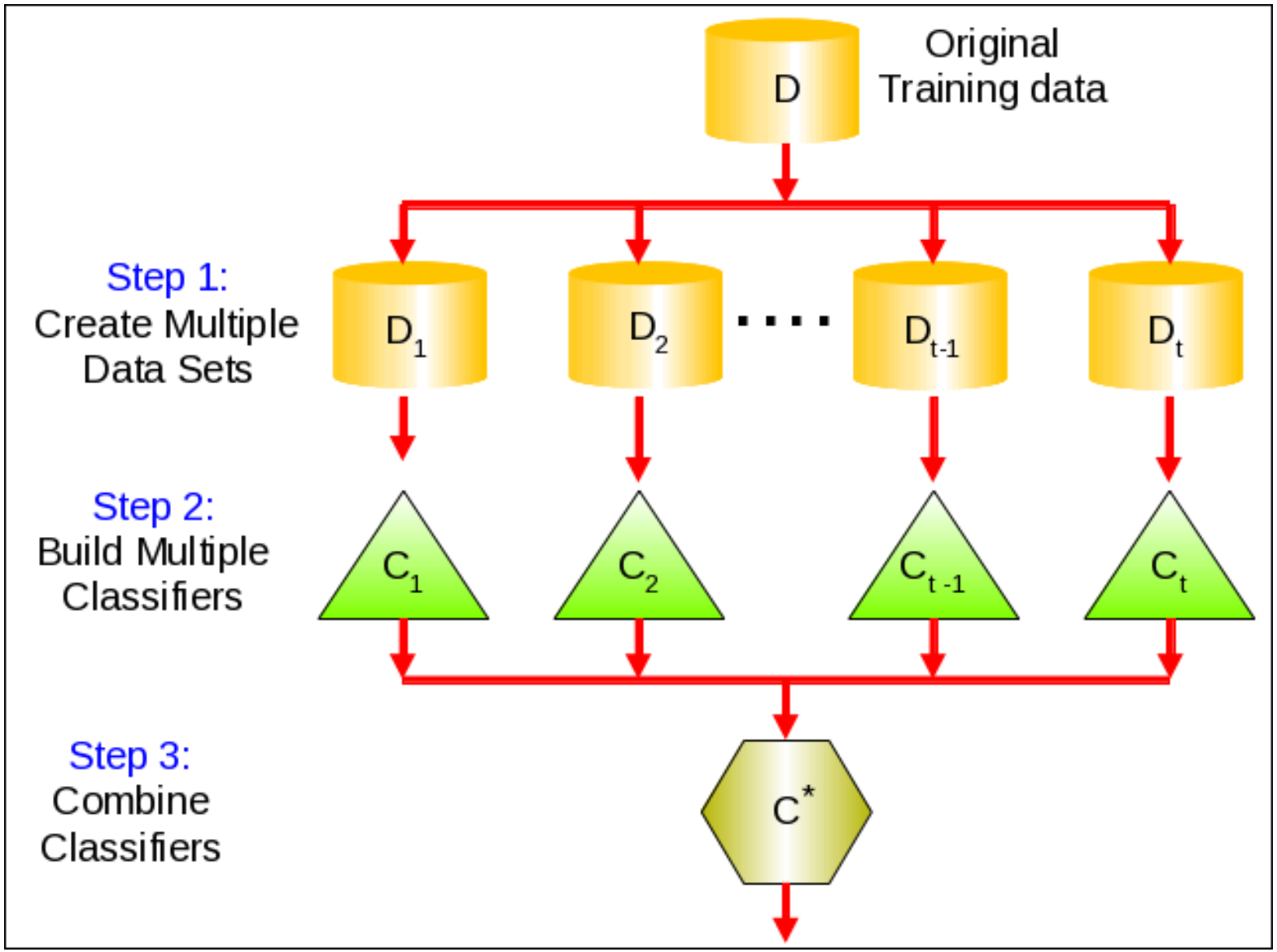
Az együttes módszerek jobban működnek **instabil osztályozókkal**, azaz olyan alaposztályozókkal, amelyek érzékenyek a tanulóhalmaz kis perturbációira (ilyenek például a döntési fák, a szabályalapú osztályozók és a mesterséges neurális hálók).

Módszerek együttes osztályozó építésére

Együttes osztályozó alkotható:

- A tanulóhalmaz manipulálásával (zsákolás és gyorsítás)
- A bemeneti jellemzők manipulálásával (véletlen erdő)
- Az osztálycímkék manipulálásával (hibajavító kimeneti kódolás)
- A tanuló algoritmus manipulálásával

Működés


D
Original
Training data
Step 1:
Create Multiple
Data Sets
D1
D2
Dt-1
Dt
Step 2:
Build Multiple
Classifiers
C1
C2
Ct -1
Ct
Step 3:
Combine
Classifiers
C*

Algoritmus

1. algoritmus. Az együttes módszer általános eljárása

1: Jelölje D az eredeti tanulóadatokat, k az alaposztályozók számát és T a tesztadatokat
2: **for** $i = 1$ **to** k **do**
3: Hozzuk létre a D_i tanulóhalmazt D-ből
4: Építsünk egy C_i alaposztályozót D_i-ből
5: **end for**
6: **for** minden $x \in T$ tesztrekordra **do**
7: $C^*(x) = \texttt{Szavazás}(C_1(\mathbf{x}), C_2(\mathbf{x}), \ldots, C_k(\mathbf{x}))$
8: **end for**

Zsákolás

A bemutatásra kerülő módszert **bootstrap aggregálásnak** (**bootstrap aggregating)** is nevezik.

Az alapötlet az, hogy hozzunk létre az eredeti tanulóhalmaz mintavételezésével vele azonos méretű bootstrap mintákat. A mintavételezés egyenletes eloszlás szerint történik visszatevéssel.

Bootstrap minták jellemzői

Mivel a mintavételezés visszatevéssel történik, az eredeti tanulóhalmaz bizonyos esetei többször is szerepelhetnek egy bootstrap mintában, míg mások lehet, hogy egyszer sem.

Egy D_i bootstrap minta átlagosan az eredeti tanulóadatok 63%-át tartalmazza, mert minden egyes eset D_i-be kiválasztásának $1 - (1 - 1/N)^N$ a valószínűsége. Ha N elég nagy, akkor a valószínűség az $1 - 1/e \approx 0{,}632$ értékhez tart.

Algoritmus

2. algoritmus. A zsákolás algoritmusa

1: Legyen k a bootstrap minták száma
2: **for** $i = 1$ **to** k **do**
3: Hozzunk létre egy N méretű D_i bootstrap mintát
4: Tanítsunk egy C_i alaposztályozót a D_i bootstrap mintán
5: **end for**
6: $C^*(x) = \underset{y}{\text{argmax}} \sum_i \delta\big(C_i(x) = y\big)$

$\{\delta(\cdot) = 1$, ha igaz az argumentuma, 0 egyébként$\}$

Példa zsákolásra (1)

Példa

Tekintsük az alábbi adatokat, ahol x egy egydimenziós attribútum y pedig az osztálycímke:

x	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

Példa zsákolásra (2)

Tételezzük fel, hogy egy osztályozót alkalmazunk, amely csak egyszintű bináris döntési fákat származtat, egy $x \leq k$ tesztfeltétellel, ahol k egy vágási pont, amelyet úgy határozunk meg, hogy minimalizálja a levélcsúcsok entrópiáját.

Egy ilyen döntési fát **döntési tönk**nek (**decision stump**) nevezünk.

Zsákolás nélkül a legjobb döntési tönk, amit előállíthatunk $x \leq 0{,}35$-nél vagy $x \leq 0{,}75$-nél osztja ketté a rekordokat. A fa pontossága mindenképpen legfeljebb 70%.

Példa zsákolásra (3)

Alkalmazzunk zsákolást 10 bootstrap mintával.

Példa zsákolásra (4)

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.8	0.9	1	1	1
y	1	1	1	-1	-1	1	1	1	1	1

x <= 0.65 ==> y = 1
x > 0.65 ==> y = 1

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.3 ==> y = 1
x > 0.3 ==> y = -1

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Példa zsákolásra (5)

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

x <= 0.05 ==> y = -1
x > 0.05 ==> y = 1

Példa zsákolásra (6)

Az együttes osztályozó tökéletesen osztályozza az eredeti adatok mind a tíz esetét.

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1
True Class	1	1	1	-1	-1	-1	-1	1	1	1

Gyorsítás (1)

Iteratív eljárás, ami a tanulóesetek eloszlásának adaptív módosítására szolgál, hogy az alaposztályozók a nehezen osztályozható esetekre összpontosítsanak.

A zsákolástól eltérően minden egyes tanulóesethez egy súly hozzárendelése, amely minden gyorsítási menet végén adaptív módon módosul.

Gyorsítás (2)

A tanulóesetekhez hozzárendelt súlyok lehetséges felhasználásai:

- Felhasználhatóak mintavételezési eloszlásként az eredeti adatokból egy bootstrap mintahalmaz választásához.
- Felhasználhatja őket az alaposztályozó egy a nagyobb súlyú esetek felé hajló modell megtanulásához.

Működés (1)

A tanulóhalmaz mintavételezési eloszlásának meghatározásához az esetek súlyainak felhasználása.

Kezdetben minden eset súlya azonosan $1/N$, hogy egyforma valószínűséggel legyenek kiválaszthatók.

Működés (2)

Egy mintát választunk a tanulóesetek mintavételezési eloszlása szerint, hogy egy új tanulóhalmazt kapjunk.

Egy osztályozót származtatunk a tanulóhalmazból és az eredeti adatok minden esetét osztályozzuk vele.

A tanulóesetek súlyait minden egyes gyorsítási menet végén módosítjuk: a helytelenül osztályozott esetek súlyait növeljük, míg a helyesen osztályozottak súlyait csökkentjük.

Példa gyorsításra (1)

Példa

Az egyes gyorsítási menetek során kiválasztott esetek:

Gyorsítás (1. menet):	7	3	2	8	7	9	4	10	6	3
Gyorsítás (2. menet):	5	4	9	4	2	5	1	7	4	2
Gyorsítás (3. menet):	4	4	8	10	4	5	4	6	3	4

Példa gyorsításra (2)

Kezdetben minden mintához ugyanazt a súlyt rendeljük. Bizonyos esetek azonban – például a 3. és 7. – egynél többször kerülhetnek kiválasztásra, mert a mintavételezés visszatevéssel történik.

Ezután egy, az adatokból épített osztályozót használunk az összes eset osztályozásához.

Tételezzük fel, hogy a 4. eset nehezen osztályozható. Ennek a mintának a súlyát növelni fogjuk a jövőbeli iterációkban, amennyiben ismételten tévesen osztályozzuk.

Eközben nagyobb esélye van a következő menetben kiválasztásra azoknak az eseteknek is, amelyek az előző menetben nem lettek kiválasztva – például az 1. és 5. eseteknek –, mivel az előrejelzéseik valószínűleg rosszak voltak az előző menetben.

A gyorsítási menetek előrehaladtával a legnehezebben osztályozható esetek még inkább túlsúlyba kerülnek. A végső együttest az egyes gyorsítási menetekből nyert alaposztályozók összességeként kapjuk meg.

Megvalósítás

A megvalósítások az alábbiak tekintetében térnek el:

- Hogyan történik a tanulóesetek súlyainak módosítása az egyes gyorsítási menetek végén.
- Hogyan történik az egyes osztályozók előrejelzéseinek kombinálása.

AdaBoost

Legyen $\left\{ (\mathbf{x}_j, y_j) \mid j = 1, \ldots, N \right\}$ tanulóesetek egy halmaza, ahol y_j egy osztálycímke minden $j = 1, \ldots, N$ esetén.

AdaBoost: alaposztályozó hibaaránya

Egy C_i alaposztályozó hibaarányát az alábbi módon definiáljuk:

$$\epsilon_i = \frac{1}{N} \sum_{j=1}^{N} w_j \, \delta\big(C_i(\mathbf{x}_j) \neq y_j\big)$$

ahol

$$\delta(p) = \begin{cases} 1, & \text{ha a } p \text{ predikátum igaz,} \\ 0, & \text{egyébként.} \end{cases}$$

AdaBoost: alaposztályozó fontossága (1)

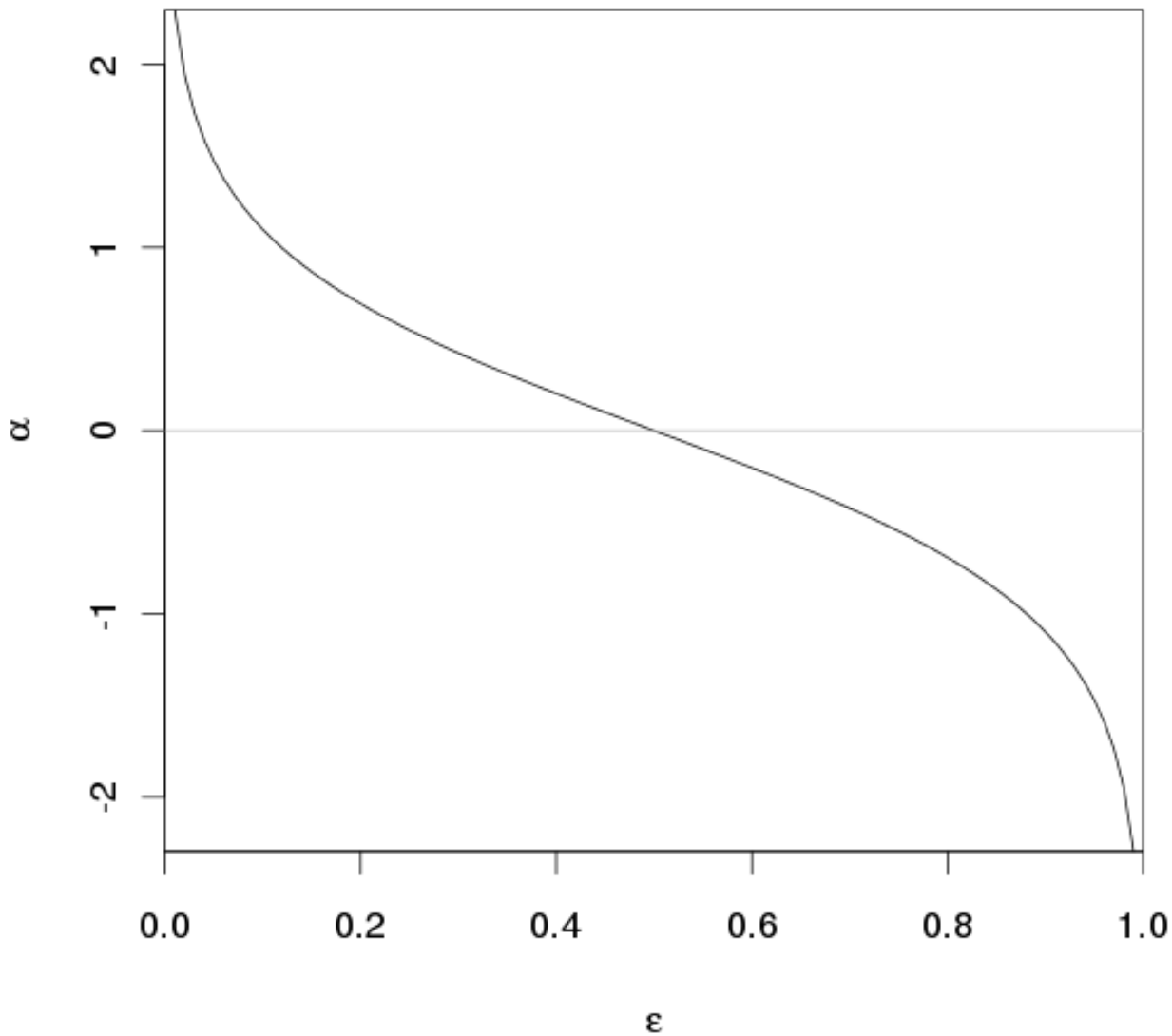
Egy C_i alaposztályozó fontosságát az alábbi módon definiáljuk:

$$\alpha_i = \frac{1}{2} \ln \left(\frac{1 - \epsilon_i}{\epsilon_i} \right).$$

Egy alaposztályozó fontossága tehát a hibaarányától függ.

AdaBoost: alaposztályozó fontossága (2)

$$\alpha = \frac{1}{2}\ln\left(\frac{1-\varepsilon}{\varepsilon}\right)$$



AdaBoost: súlymódosítás

Jelölje $w_i^{(j)}$ az $(\mathbf{x}_i, y_i)$ esethez a j-edik gyorsítási menet során hozzárendelt súlyt.

A súlymódosítás az alábbi egyenlet szerint történik:

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j} \times \begin{cases} e^{-\alpha_j}, & \text{ha } C_j(\mathbf{x}_i) = y_i, \\ e^{\alpha_j}, & \text{ha } C_j(\mathbf{x}_i) \neq y_i, \end{cases} \tag{1}$$

ahol Z_j normalizáló tényező, amely a $\sum_i w_i^{(j+1)} = 1$ feltételt biztosítja.

A fenti formula növeli a hibásan osztályozott esetek súlyait és csökkenti azokét, amelyek helyesen osztályozottak.

AdaBoost: szavazás

Többségi szavazási séma helyett minden egyes C_j osztályozó előrejelzését α_j-nek megfelelően súlyozzuk.

Ha bármelyik közbenső menetben 50%-nál nagyobb hibaarányt kapunk, akkor a súlyokat visszaállítjuk az eredeti egyforma értékeikre és megismételjük az újramintavételezési eljárást.

AdaBoost: algoritmus

3. algoritmus. AdaBoost algoritmus

1: $\mathbf{w} = \{ w_j = 1/N \mid j = 1, \ldots, N \}$ {az N eset súlyainak inicializálása}
2: Legyen k a gyorsítási menetek száma
3: **for** $i = 1$ **to** k **do**
4: Hozzuk létre a D_i tanulóhalmazt D-nek a $\mathbf{w}$ szerint történő (visszatevéses) mintavételezésével
5: Tanítsunk egy C_i alaposztályozót D_i-n
6: Alkalmazzuk C_i-t az eredeti D tanulóhalmaz valamennyi esetére
7: $\epsilon_i = \frac{1}{N} \sum_j w_j \, \delta\big(C_i(\mathbf{x}_j) \neq y_j\big)$ {a súlyozott hiba kiszámítása}
8: **if** $\epsilon_i > 0{,}5$ **then**
9: $\mathbf{w} = \{ w_j = 1/N \mid j = 1, \ldots, N \}$ {az N eset súlyainak visszaállítása}
10: Menjünk vissza a 4. lépésre
11: **end if**
12: $\alpha_i = \frac{1}{2} \ln \frac{1-\epsilon_i}{\epsilon_i}$
13: Módosítsuk minden eset súlyát az (1) egyenlet szerint
14: **end for**
15: $C^*(\mathbf{x}) = \operatorname{argmax}_y \sum_{j=1}^{k} \alpha_j \, \delta\big(C_j(\mathbf{x}) = y\big)$

AdaBoost: példa (1)

Példa

Tekintsük az alábbi adatokat, ahol x egy egydimenziós attribútum y pedig az osztálycímke:

x	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

AdaBoost: példa (2)

Alaposztályozókként használjunk döntési tönköket $x \leq k$ tesztfeltétellel, ahol k egy vágási pont, amelyet úgy határozunk meg, hogy minimalizálja a levélcsúcsok entrópiáját.

A döntési tönkök osztályozási pontossága mindenképpen legfeljebb 70%.

AdaBoost: példa (3)

Boosting Round 1:

x	0.1	0.4	0.5	0.6	0.6	0.7	0.7	0.7	0.8	1
y	1	-1	-1	-1	-1	-1	-1	-1	1	1

Boosting Round 2:

x	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
y	1	1	1	1	1	1	1	1	1	1

Boosting Round 3:

x	0.2	0.2	0.4	0.4	0.4	0.4	0.5	0.6	0.6	0.7
y	1	1	-1	-1	-1	-1	-1	-1	-1	-1

(a) Training records chosen during boosting

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
2	0.311	0.311	0.311	0.01	0.01	0.01	0.01	0.01	0.01	0.01
3	0.029	0.029	0.029	0.228	0.228	0.228	0.228	0.009	0.009	0.009

(b) Weights of training records

AdaBoost: példa (4)

Round	Split Point	Left Class	Right Class	α
1	0.75	-1	1	1.738
2	0.05	1	1	2.7784
3	0.3	1	-1	4.1195

(a)

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	-1	-1	-1	-1	-1	-1	-1	1	1	1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
Sum	5.16	5.16	5.16	-3.08	-3.08	-3.08	-3.08	0.397	0.397	0.397
Sign	1	1	1	-1	-1	-1	-1	1	1	1

(b)

Az AdaBoost tökéletesen osztályozza az eredeti adatok mind a tíz esetét.

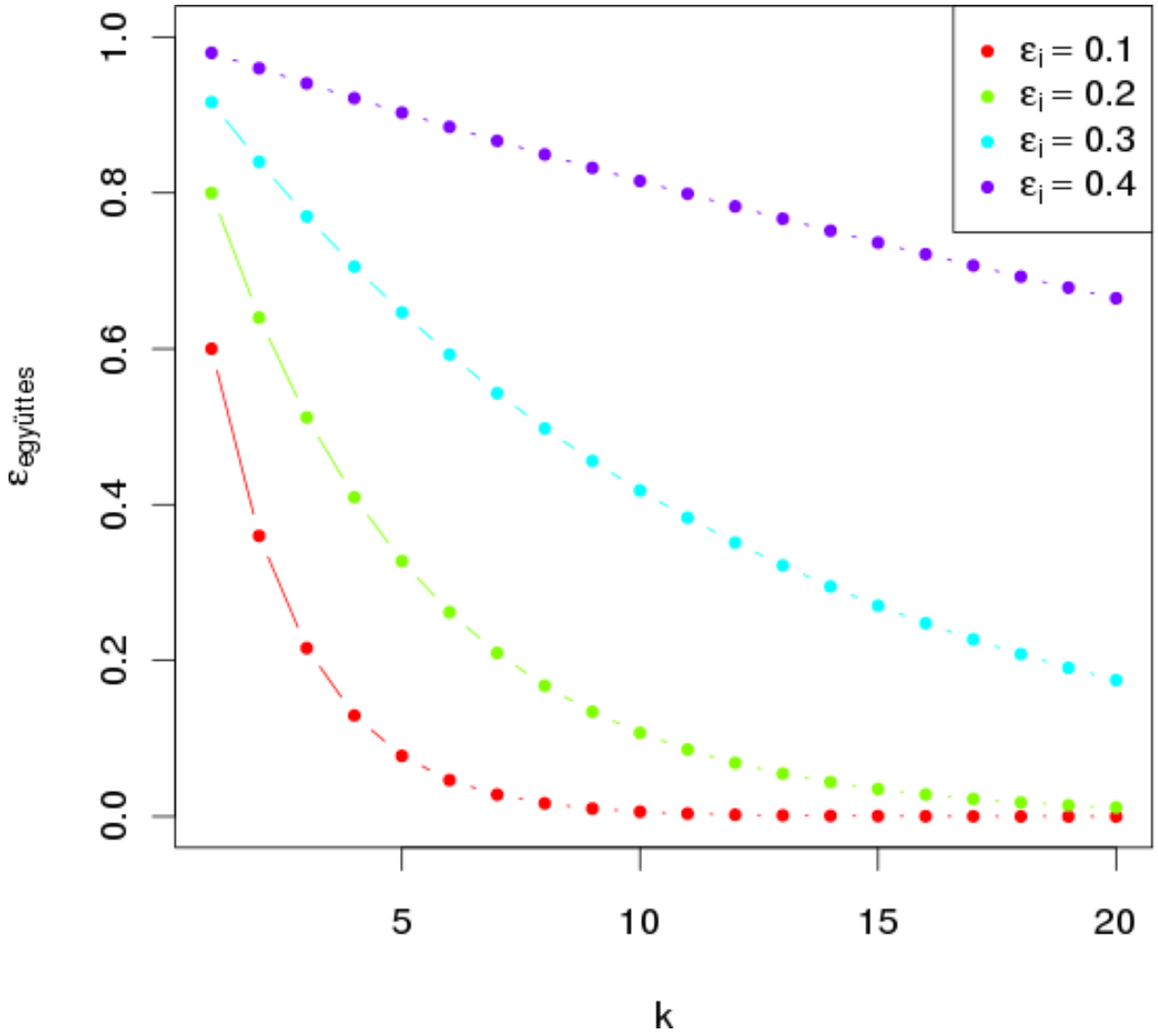
AdaBoost: hibakorlát (1)

Az együttes tanulóhalmazon mért hibaarányát a következő kifejezés korlátozza:

$$\epsilon_{\text{együttes}} \leq \prod_i 2\sqrt{\epsilon_i(1-\epsilon_i)},$$

ahol ϵ_i az i-edik alaposztályozó hibaaránya.

AdaBoost: hibakorlát (2)



Az együttes hibaarányára vonatkozó korlát azonos ϵ_i hibaarányú alaposztályozók esetén.

AdaBoost: hibakorlát (3)

Ha a hibaarány kisebb 50%-nál, akkor $\epsilon_i = 0{,}5 - \gamma_i$ írható, ahol γ_i azt méri, hogy mennyivel jobb az osztályozó a véletlen találgatásnál. Ekkor az együttes osztályozó hibaarányára a korlát:

$$\epsilon_{\text{együttes}} \leq \prod_i \sqrt{1 - 4\gamma_i^2} \leq \exp\left(-2 \sum_i \gamma_i^2\right).$$

Ha minden i esetén $\gamma_i \geq \gamma > 0$, akkor tehát

$$\epsilon_{\text{együttes}} \leq \exp\left(-2k\gamma^2\right),$$

azaz a hibaarány exponenciálisan csökken.

AdaBoost: megvalósítás

Bizonyos osztályozási módszerek képesek olyan tanulóhalmazok kezelésére, amelyekben a tanulóesetekhez súlyok tartoznak. Ilyen alaposztályozók használata esetén az algoritmus 4–5. lépései helyett az alábbi kell végrehajtani:

4–5: Tanítsunk egy C_i alaposztályozót D-n $\mathbf{w}$ felhasználásával

A módszernek ezt a változatát **újrasúlyozás általi gyorsítás**nak (**boosting by reweighting**) nevezik.[3]

A bemutatott algoritmus az alaposztályozók alkotásához D-nek $\mathbf{w}$ szerint történő mintavételezésével létrehozott D_i tanulóhalmazokat használ. A módszernek ezt a változatát **újramintavételezés általi gyorsítás**nak (**boosting by resampling**) nevezik.

[3] Ilyen az algoritmus eredeti változata.

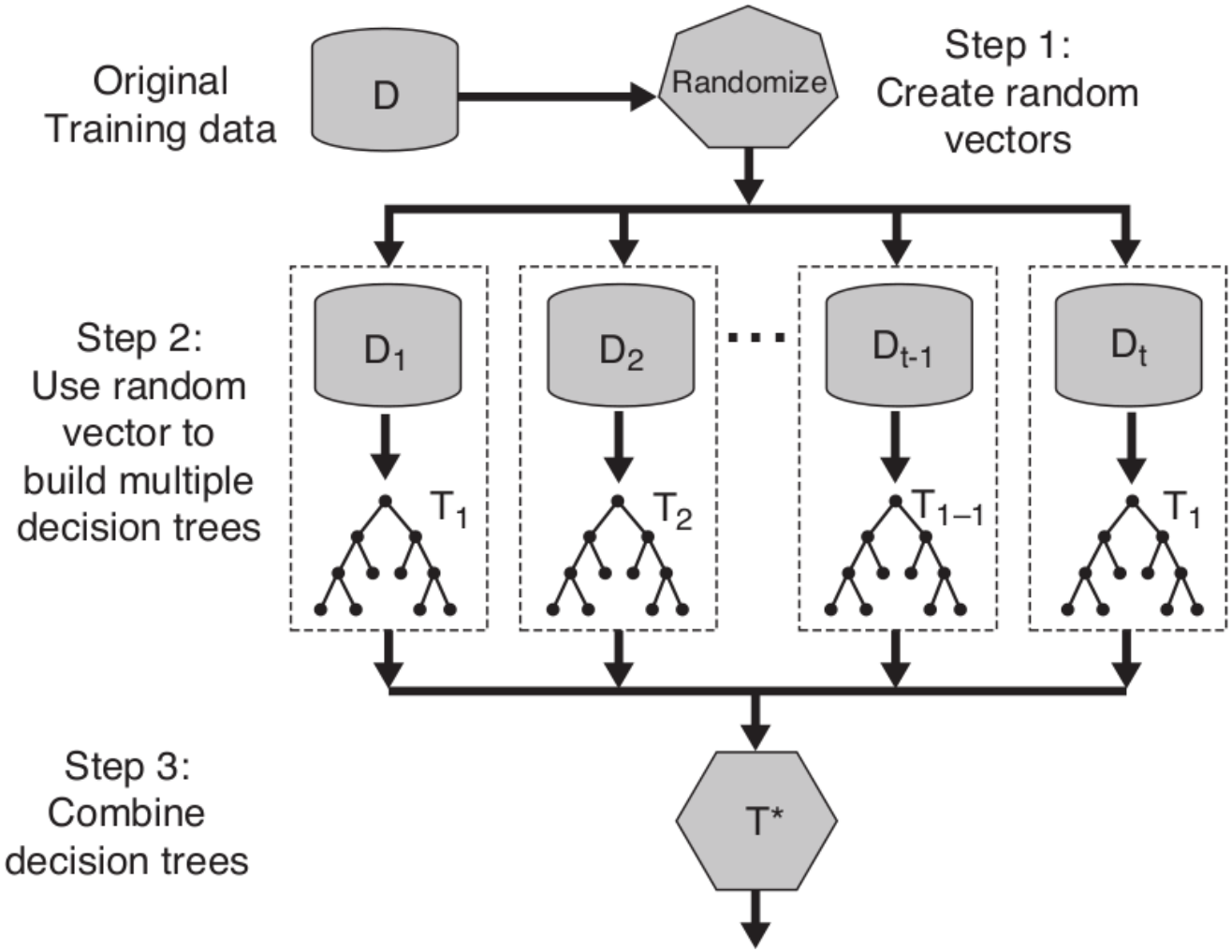
Véletlen erdők (1)

Olyan együttes módszerek, amelyeket kifejezetten döntési fa osztályozókhoz terveztek.

Több döntési fa által adott előrejelzéseket kombinálnak, ahol mindegyik fa véletlen vektorok egy független halmazának értékei alapján kerül létrehozásra.

A véletlen vektorok egy rögzített valószínűségi eloszlásból jönnek létre.

Véletlen erdők (2)



A zsákolás, mint a véletlen erdők speciális esete

A döntési fákat felhasználó zsákolás a véletlen erdők egy speciális esete, ahol úgy viszünk véletlenszerűséget a modellépítő eljárásba, hogy az eredeti tanulóhalmazból véletlenszerűen választunk visszatevéssel N mintát.

A zsákolás a teljes modellépítő eljárás során ugyanazt az egyenletes valószínűségi eloszlást használja a bootstrap minták előállításához is.

Véletlen erdő formális definíciója

Definíció

Véletlen erdőnek nevezzük döntési fák egy olyan $\{ h(\mathbf{x}, \Theta_i) \mid i = 1, \ldots, k \}$ együttesét, ahol a $\{\Theta_i\}$-k független, azonos eloszlású véletlen vektorok, és amely többségi szavazási sémát használ az osztályozáshoz.

Véletlen erdők pontossága

A pontosság az alábbi tényezőktől függ:

- Az egyes döntési fák számától és minőségétől.
- A döntési fák közötti korrelációtótól.

Felső korlát az általánosítási hibára

Elméletileg bizonyított, hogy a véletlen erdők általánosítási hibájának felső korlátja a következő kifejezéshez konvergál, amikor a fák száma elég nagy:

$$\text{általánosítási hiba} \leq \frac{\overline{\rho}(1-s^2)}{s^2},$$

ahol $\overline{\rho}$ a fák közötti átlagos korreláció, s pedig egy a fa osztályozók „erejét” mérő mennyiség.

Azaz amint a fák erősebben korrelálttá válnak vagy csökken az osztályozó-együttes ereje, nő az általánosítási hiba korlátja.

Véletlen erdő margója

Osztályozók egy halmazának ereje az osztályozók átlagos teljesítményét jelenti, ahol a teljesítményt valószínűségileg mérjük az osztályozók margói szerint:

$$\text{Margó:} \quad M(\mathbf{X}, Y) = P(\hat{Y}_\Theta = Y) - \max_{Z \neq Y} P(\hat{Y}_\Theta = Z),$$

ahol $\hat{Y}_\Theta$ az $\mathbf{X}$ egy valamilyen Θ véletlen vektorból épített osztályozó szerint prediktált osztálya.
Minél nagyobb a margó, annál valószínűbb, hogy az osztályozó helyesen prediktál egy adott $\mathbf{X}$ esetet.

Véletlen erdők megvalósítása

Sokféle módon építhető be a faépítő folyamatba egy véletlen vektor. Az alábbi megvalósításokat vizsgáljuk:

- Forest-RI
- Forest-RC
- Randomizálás

Forest-RI[4] (1)

Válasszunk véletlenszerűen F számú bemeneti jellemzőt a döntési fa minden csúcsának vágásához (F paraméter).

Egy csúcs vágásához nem vizsgáljuk meg az összes jellemzőt, a döntés meghatározása az F kiválasztott jellemzőből történik.

A fát ezután nyesés nélkül építjük teljes egésszé.

A fák megalkotását követően az előrejelzéseket egy többségi szavazási séma segítségével kombináljuk.

[4] Az eljárás nevében RI a véletlenszerű bemenet kiválasztást (*random input selection*) jelenti.

Forest-RI (2)

A véletlenszerűséget növelhetjük úgy, hogy zsákolással generált bootstrap mintákat használunk.

Forest-RI (3)

A véletlen erdők ereje és korrelációja F nagyságától függhet.

Ha F elég kicsi, akkor a fák általában gyengébben korrelálttá válnak.

Másrészt, a fa osztályozó ereje hajlamos javulni a jellemzők F számának növekedésével.

Kompromisszumként a jellemzők számára szokásos választás $F = \log_2 d + 1$, ahol d a bemeneti jellemzők száma.

Forest-RI (4)

Jelentősen csökkenti az algoritmus futásidejét, hogy minden csúcsban csak a jellemzők egy részhalmazát kell megvizsgálni.

Forest-RC[5] (1)

Ha az eredeti jellemzők d száma túl kicsi, akkor nehéz a döntési fák építéséhez véletlen jellemzők egy független halmazát kiválasztani.

A jellemzők terének növelésének egy módja a bemeneti jellemzők lineáris kombinációinak képzése.

[5] Az eljárás nevében RC a véletlenszerű lineáris kombinációt (*random linear combination*) jelenti.

Forest-RC (2)

Minden egyes csúcsban egy új jellemzőt generálunk véletlenszerűen kiválasztva a bemeneti jellemzők közül L számút (L paraméter).

A bemeneti jellemzőket a $[-1, 1]$ tartományon egyenletes eloszlásból generált együtthatókkal kombináljuk lineárisan.

Minden csúcsban F ilyen véletlenszerűen kombinált új jellemzőt generálunk (F paraméter), közülük a legjobbat választjuk a csúcs vágásához.

Randomizálás (1)

Döntési fa minden csúcsában az F legjobb vágás egyikének véletlenszerű választása (F paraméter).

Randomizálás (2)

A módszer erősebben korrelált fákat generálhat potenciálisan, mint a Forest-RI és Forest-RC, hacsak nem megfelelően nagy F.

Nem rendelkezik a Forest-RI és Forest-RC futásidő-megtakarításaival sem, mivel egy döntési fa minden egyes csúcsában meg kell vizsgálni az összes vágási jellemzőt.

A véletlen erdők jellemzői

Tapasztalatilag igazolható, hogy a véletlen erdők osztályozási pontosságai elég hasonlóak az AdaBoost algoritmushoz.

Robusztusabbak a zajra és sokkal gyorsabbak is, mint az AdaBoost algoritmus.

Hibajavító kimenet kódolás

A hibajavító kimenet kódolás robusztusabb módját biztosítja többosztályos problémák kezelésének, mint a korábban tárgyalt *one-against-one* és *one-against-rest* módszerek.

A módszert egy zajos csatornákon keresztüli üzenetküldés információelméleti megközelítése ihlette.

Az *one-against-rest* módszer érzékenysége az osztályozási hibákra (1)

Példa

Tekintsünk egy olyan osztályozási feladatot, ahol 4 osztály van, és jelölje $Y = \{ 1, 2, 3, 4 \}$ az osztálycímkék halmazát.

Tételezzük fel, hogy egy tesztvektort a következőképpen osztályozunk a *one-against-rest* módszer alkalmazása során:

Bináris osztálypár	$+$: 1 $-$: $\{2, 3, 4\}$	$+$: 2 $-$: $\{1, 3, 4\}$	$+$: 3 $-$: $\{1, 2, 4\}$	$+$: 4 $-$: $\{1, 2, 3\}$
Osztályozás	$+$	$-$	$-$	$-$

A predikciók kombinálása után az 1. osztály négy szavazatot kap, míg az összes többi osztály csak két szavazatot. A tesztvektort ezért az 1. osztályhoz tartozóként osztályozzuk.

Az *one-against-rest* módszer érzékenysége az osztályozási hibákra (2)

Ha az előbbi tesztvektort a 3. osztályozó tévesen osztályozza, akkor az alábbi eredményt kapjuk:

Bináris osztálypár	$+$: 1 $-$: $\{2,3,4\}$	$+$: 2 $-$: $\{1,3,4\}$	$+$: 3 $-$: $\{1,2,4\}$	$+$: 4 $-$: $\{1,2,3\}$
Osztályozás	$+$	$-$	$+$	$-$

A predikciók kombinálása után az 1. és 3. osztály három szavazatot kap, míg a másik két osztály csak egyet-egyet. A tesztvektor osztályaként az 1. vagy 3. osztály közül kell valamelyiket választani.

Az *one-against-rest* módszer érzékenysége az osztályozási hibákra (3)

Ha az egyik bináris osztályozó egy hibát vét az előrejelzéseiben, akkor az együttes végül az osztályok közötti döntetlent jelenthet vagy hibás előrejelzést adhat.

Hibajavító kimenet kódolás használata

Minden $y \in Y$ osztályt egy **kódszó**nak nevezett egyedi, n hosszú bitsorozat reprezentál.

n bináris osztályozót tanítunk a kódszó bitsorozat minden egyes bitjének előrejelzéséhez.

Egy tesztpéldány prediktált osztályát az a kódszó adja meg, amelynek Hamming-távolsága a legkisebb a bináris osztályozók által előállított kódszóhoz.[6]

[6] Két bitsorozat közötti Hamming-távolság az eltérő bitek száma.

A hibajavító kimenet kódolás hibatűrése

Ha bármely két kódszó közötti Hamming-távolság legalább d, akkor a kimeneti kód bármely $\lfloor (d-1)/2) \rfloor$ számú hibája javítható a legközelebbi kódszó segítségével.

Példa hibajavító kimenet kódolásra (1)

Példa

Tekintsünk egy többosztályos problémát, ahol $Y = \{1, 2, 3, 4\}$.

Kódoljuk az osztályokat például a következő 7-bites kódszavakkal:

Osztály	Kódszó						
1.	1	1	1	1	1	1	1
2.	0	0	0	0	1	1	1
3.	0	0	1	1	0	0	1
4.	0	1	0	1	0	1	0

A kódszó minden egyes bitjét használjuk fel egy bináris osztályozó tanításához.

Példa hibajavító kimenet kódolásra (2)

Tegyük fel, hogy egy tesztpéldányt $(0, 1, 1, 1, 1, 1, 1)$ módon osztályoznak a bináris osztályozók.

	0	1	1	1	1	1	1
1.	**1**	1	1	1	1	1	1
2.	0	**0**	**0**	**0**	1	1	1
3.	0	**0**	1	1	**0**	**0**	1
4.	0	1	**0**	1	**0**	1	**0**

A tesztpéldányt tehát az 1. osztályhoz tartózként osztályozzuk.

Példa hibajavító kimenet kódolásra (3)

Mivel a minimális Hamming-távolság bármely kódszópár között 4, az együttes elviselheti a hét bináris osztályozó egyike által vétett hibákat.

Ha egynél több osztályozó vét hibát, akkor az együttes nem minden esetben képes a hiba kompenzálására.

Kódszóhalmaz tervezése hibajavító kimenet kódoláshoz

Kérdés, hogy a különböző osztályokhoz hogyan válasszunk kódszavakat.

A kódelméletben hatalmas számú algoritmus került kifejlesztésre korlátos Hamming-távolságú n-bites kódszavak generálására.

Megjegyzés

Szignifikáns különbség van a kommunikációs feladatokhoz használt hibajavító kódok és a többosztályos tanuláshoz használtak tervezése között.

Hivatkozások (1)

Zsákolás:

- L. Breiman. "Bagging Predictors". In: *Machine Learning* 24.2 (1996), pp. 123–140. DOI: 10.1023/A:1018054314350. URL: http://dx.doi.org/10.1023/A:1018054314350

AdaBoost:

- Y. Freund and R. E. Schapire. "A decision-theoretic generalization of on-line learning and an application to boosting". In: *Journal of Computer and System Sciences* 55.1 (1997), pp. 119–139. DOI: 10.1006/jcss.1997.1504. URL: http://cseweb.ucsd.edu/~yfreund/papers/adaboost.pdf

Hivatkozások (2)

Véletlen erdők:

- L. Breiman. "Random Forests". In: *Machine Learning* 45.1 (2001), pp. 5–32. DOI: 10.1023/A:1010933404324. URL: http://dx.doi.org/10.1023/A:1010933404324

Hibajavító kimenet kódolás:

- T. G. Dietterich and G. Bakiri. "Solving Multiclass Learning Problems via Error-Correcting Output Codes". In: *Journal of Artificial Intelligence Research* 2 (1995), pp. 263–286. DOI: 10.1613/jair.105. URL: http://dx.doi.org/10.1613/jair.105