EÖTVÖS LORÁND TUDOMÁNYEGYETEM

KLASSZIFIKÁCIÓ AZ ADATBÁNYÁSZATBAN

SZAKDOLGOZAT

Készítette: Bényász Melinda
Matematika Bsc
Matematikai elemző szakirány

Témavezető: Kósa Balázs
Informatikai Kar
Információs Rendszerek Tanszék

Budapest
2010

Tartalomjegyzék

1. fejezet

Bevezetés

John Naisbitt:
„We are drowning in information, but starving for knowledge."
(Belefulladunk az információba, miközben tudásra éhezünk.)

Az utóbbi évtizedekben az adatok előállításának és gyűjtésének lehetőségei nagymértékben növekedtek, ezáltal hatalmas mennyiségű adat halmozódott fel. Adatokat generálunk mi magunk is például vásárlásaink alkalmával, vagy ha részt veszünk valamilyen kórházi vizsgálaton, ha hivatalos papírokat töltünk ki, stb.
Egyre több területen merült fel az igény, hogy a meglévő adathalmazokból értékes, használható információkat nyerjenek ki. Ez egy új tudományág, az adatbányászat születéséhez vezetett.
Az adatbányászat az 1980-as évektől tekinthető önálló tudományterületnek, melyben kezdetben nagyrészt a különböző heurisztikák domináltak, ezek viszont számos érdekes kérdést vetettek fel, melyek a kutatók körében egyre népszerűbbé tették ezt a területet.
Napjainkban már a nagy cégeknél, vállalatoknál az adatbányászat nélkül szinte elképzelhetetlen a munka, a hatalmas adattömegekből kinyert információtöbblet nélkülözhetetlen a fejlődéshez. A rivális cégek közötti egyre kiélezettebb versenyhelyzetnek, valamint a további megválaszolatlan kérdéseknek köszönhetően népszerűsége még igen hosszú ideig garantált.

2. fejezet

A tudásfeltárás folyamata

Sokan az adatbázisokban végzett tudásfeltárást (KDD) értik adatbányászat alatt, ám valójában az adatbányászat a KDD egy meghatározó része. A tudásfeltárás folyamatának főbb lépései a következők:

1. *Adattisztítás*
2. *Adatintegrálás*
3. *Adatkiválasztás*
4. *Adat-transzformáció*
5. *Adatbányászat*
6. *A kapott minta kiértékelése*
7. *Tudásmegjelenítés*

2.1. Adattisztítás

Az adattisztítás lényege, hogy javítja az adatok minőségét azáltal, hogy kiszűri és eltávolítja az adatokban fellépő hibákat és inkonzisztenciát. Az adataink *rekordokban* vannak, melyek *attribútumokból* (változókból) állnak.
A következő attribútum típusokat használjuk a továbbiakban:

- *kategorikus:* csak a felsorolt értékek (például: alma, körte, szilva)valamelyikét veheti fel;
- *numerikus:* szám típusú, például paraméterek mérési eredményei;

- *bináris:* csupán két értéket vehet fel, például 0 vagy 1, igen vagy nem, stb.

2.1.1. Hiányzó értékek

Ahhoz hogy egy rekord használható legyen egy adatbányászati algoritmushoz, sokszor szükséges, hogy minden attribútumát ismerjük. A valós életben ez azonban nincs mindig így. Mit tehetünk ilyenkor?

- *Nem vesszük figyelembe azt a sort, melyben hiányosság van*:
  ez a módszer nem túl előnyös (hiszen esetenként számos hasznos információról is lemondunk), csak akkor hatékony, ha sok hiányzó elem van az adott sorban.

- *Manuálisan töltjük ki a hiányzó értéket*:
  ez azonban igen időigényes, bár ha fontos információról van szó, elképzelhető hogy elkerülhetetlen. Sok hiányzó érték esetén nem használható.

- *Pótolhatjuk a hiányzó értéket az attribútum átlagértékével*:
  numerikus értékek esetén ez könnyű. Átlag helyett használhatunk esetleg mediánt vagy móduszt is.

- *Hiányzó értékünket pótolhatjuk a legvalószínűbb értékével*:
  meglévő adataink segítségével meghatározzuk, hogy melyik az az érték, amelyet az adott mező a legnagyobb valószínűséggel vesz fel. Ezt megtehetjük például Bayes–döntéssel, vagy regresszióval.

- *Esetleg reprezentálunk minden lehetséges értéket*:
  a hiányzó értéket sorban az összes lehetséges értékkel behelyettesítjük. Ez akkor hatásos, ha közel azonos az értékek eloszlása.

2.1.2. Zajos adatok

Mi is az a zajos adat?

Általában egy adat információból és zajból áll, ez utóbbi egyfajta hibát vagy ingadozást jelent a mért értékeken, de tekinthetünk rá úgy is, mint az adatok egyfajta torzítására. Ilyen esetekben hasznos lehet, ha „simítjuk” az adatokat. Ezt megtehetjük például egy kosarazási módszerrel. A kosarazási módszerek a rendezett adatok értékeit a körülöttük lévő értékekkel összevetve simítják. Feltesszük, hogy egy

kosárba legfeljebb n darab adat fér. A rendezett adatokból az egymás mellett lévő n darab értéket egy-egy kosárnak feleltetjük meg (ezek természetesen diszjunktak). Ezek után végezzük el a simítást, mégpedig úgy, hogy vesszük az egyes kosarakban lévő értékek mediánját vagy átlagát, majd ezek alapján írjuk felül a kosárban lévő adatokat. Ez a simítás lokális, hiszen közeli szomszédokat vizsgálunk.

Példa kosarazásra: A rendezett adataink: 4, 8, 15, 21, 21, 24, 25, 28, 34
Partícionáljuk az adatainkat azonos mélységű kosarakba:

1. kosár:4, 8, 15
2. kosár: 21, 21, 24
3. kosár: 25, 28, 34

Simítás a kosárátlagok szerint:

1. kosár: 9, 9, 9
2. kosár: 22, 22, 22
3. kosár: 29, 29, 29

A kosarazáson felül használhatunk még az adatok simítására regressziót is, mely során az adatok értékeire egy egyenest illesztünk.

2.1.3. Inkonzisztens adatok

Az inkonzisztencia szó jelentése: ellentmondás, következetlenség. Inkonzisztencia felléphet például az adatok integrálása közben, ha egy adott attribútum különböző néven szerepel a különböző adatbázisokban. Az inkonzisztenciák bizonyos esetekben kézzel kijavíthatók külső források alapján, vagy használhatunk az ismeretanyagot rendszerező, szervező eszközöket is.

2.2. Adatok integrálása

Az adatok integrálása alatt az adatok különböző adatbázisokból való egyesítését értjük. Közben azonban problémák merülhetnek fel. *Hogyan feleltethetjük meg egymásnak a különböző forrásokban lévő ekvivalens egyedeket?* Ez az egyedazonosítási probléma.
A *redundancia* szintén fontos kérdés. Akkor mondjuk hogy egy attribútum redundáns,

ha kiszámítható egy másik táblából. Korrelációanalízissel mérhetjük annak a valószínűségét, hogy az A és B attribútumok valamelyikéből következtethetünk a másiknak az értékére. Az alábbi képlettel határozhatjuk meg a két attribútum korrelációját:

$$r_{A,B} = \frac{\sum(A - \overline{A})(B - \overline{B})}{(n-1)\sigma_A\sigma_B},$$

ahol $\overline{A}$ és $\overline{B}$ az átlaga, σ_A és σ_B a szórása az A-nak és a B-nek, n pedig a sorok száma. Ha a 0-t kapjuk eredményül, akkor A és B függetlenek, nagy korreláció esetén pedig A vagy B redundáns mennyiségként eltávolítható, hacsak nem épp ezekre vadászunk, mint például *asszociációs szabályok* keresésekor.
Az *ellentmondó adatértékek felderítése és kijavítása* jelenti a harmadik fontos problémát. A nem egységes reprezentáció, skálázás vagy kódolás eredményeként a különböző forrásokban egy adott egyed attribútumainak értékei eltérhetnek egymástól.

2.3. Adatok kiválasztása

Gyakran előfordul, hogy hatalmas mennyiségű adattal találjuk szemben magunkat, ezt nemcsak a rekordok, de az attribútumok száma is okozhatja. Ez nagy dimenziójú adathalmazt eredményez, mely sok esetben hátrányos lehet, elég ha csak a tárolási költségre, vagy az ilyen adathalmazokon végzett algoritmusok futási idejére gondolunk.

2.3.1. Számosságcsökkentés

Ebben a részben a rekordok számának csökkentésével, azaz a számosságcsökkentéssel foglalkozunk. Erre szolgál például a *mintavételezés*, melynek segítségével egy nagyméretű adathalmazt reprezentálunk egy annál sokkal kisebb véletlen mintával.

- *Visszatevés nélküli M elemű egyszerű véletlen mintakiválasztás*: ez a legegyszerűbb és leggyorsabb módja az adatok kiválasztásának. Ugyanolyan valószínűséggel választunk minden elemet, de nem vehetjük a részhalmazhoz kétszer ugyanazt a rekordot.

- *Visszatevéses M elemű egyszerű véletlen mintakiválasztás*: akkor használhatjuk, ha nem számít a redundancia az elemző módszernél. Ekkor ugyanazt az elemet

többször is választhatjuk.

- *Klaszterminta*: ha az adathalmazt automatikusan csoportosítani (klaszterezni) tudjuk, akkor klaszterenként egyenletes eloszlással véletlenszerűen veszünk elemeket (visszatevéssel vagy anélkül).

- *Rétegzett minta*: önkényesen kijelölünk bizonyos csoportokat, ezekből kell véletlenszerűen mintákat választani (ezzel elérhető, hogy a ritka csoportok is reprezentálva legyenek).

2.3.2. Dimenziócsökkentés

Nagy dimenziójú adathalmazok esetén említést kell tennünk a *curse of dimensionality* (azaz a dimenziók átka) kifejezésről, amely *Richard Bellman* nevéhez fűződik. Számos szemléltetése létezik, nézzük ezek közül az alábbit:
Tetszőleges pont környezetében elég tanítópontnak kell lennie. Egy x pont esetén a legfeljebb ϵ távolságra lévő pontokat nevezzük az x pont környezetének. Ez egy dimenziós esetben egy 2ϵ hosszúságú szakaszt jelent, két dimenzióban ϵ sugarú kört, három dimenzió esetén pedig egy ϵ sugarú gömböt. Ha azt szeretnénk, hogy rögzített legyen az adott térben a tanítópontok sűrűsége, akkor a dimenzió számának növelésével a tanítpontok számának exponenciálisan kell nőnie. A tanítópontok a gyakolatban adottak, ez általában behatárolja a dimenziók számát, ezzel együtt pedig a figyelembe vehető attribútumok számát is.

Hamar felmerül a kérdés: *vajon a rendelkezésünkre álló számos paraméterből mely attribútumokra, dimenziókra lesz ténylegesen szükségünk?*
Amennyiben kisebb adathalmazokkal kell dolgoznunk esetleg manuálisan is kiválaszthatjuk a számunkra fontos attribútumokat, persze ehhez az kell, hogy ismerjük a köztük lévő összefüggéseket. Ez azonban nem működne nagyméretű adathalmazoknál, hiszen egyáltalán nem biztos, hogy minden attribútum tulajdonságát ismerjük, így tévedés esetén, fontos attribútumok elhagyásával, vagy lényegtelenek megtartásával az algoritmus alkalmazása során nem azt az eredményt kapjuk, amit vártunk volna. Ez rossz minőségű mintákat eredményezhet. Erre a problémára az *attribútumok egy részhalmazának kiválasztására* szolgáló módszereket használhatunk.

Hogyan találhatunk egy „jó" részhalmazt az attribútumok között? Legyen az attribútumok száma d. Ekkor 2^d lehetséges részhalmazunk van. *Kimerítő keresés*:

az összes lehetséges részhalmazt végigvizsgálja, garantáltan megtalálja közülük az optimálisat. Ha N-ből M elemű részhalmazt keresünk, akkor a módszer nem alkalmazható nagy N és M esetén, hiszen exponenciális lépésszámú az algoritmus. Ezért alkalmazzunk olyan heurisztikus módszereket, melyek egy csökkentett keresési téren dolgoznak. Stratégiájuk lényege, hogy lokális optimális választást végeznek annak reményében, hogy ez majd a globálisan optimális megoldáshoz vezet. Tapasztalatok szerint ezek a módszerek igen hatékonynak bizonyultak a gyakorlatban, gyakran kapunk alkalmazásukkal az optimális megoldásra jó közelítést. Melyek ezek a heurisztikus módszerek?

- *Előrelépéses kiválasztás*: üres attribútumhalmazzal indul, majd minden lépésnél a legjobb attribútummal bővíti a részhalmazt.
- *Visszalépéses eliminálás*: a teljes attribútumhalmazból indul, minden lépésnél a halmaz legrosszabb elemét hagyja el.
- *Az előző kettő kombinációja*: úgy kombináljuk össze a fentieket, hogy az aktuális halmazunkból elhagyjuk a legrosszabbat, és hozzávesszük a maradék attribútumok közül a legjobbat.
- *Hozzávesz j-t, eldob k-t*: előrelépéssel j darab attribútumot hozzávesz, majd k darabot visszalépéssel eldob. Ezzel a korábban kidobott attribútumok újravizsgálata is lehetséges.

Szinguláris felbontás

További lehetőség a dimenziócsökkentésre a szinguláris felbontás (singular value decomposition, SVD).

1. Definíció. *Egy $M \in R^{m \times n}$ mátrix szinguláris érték felbontásán az olyan*

$$M = U \Sigma V^T$$

szorzattá bontást értjük, ahol $U \in R^{m \times m}$, $V \in R^{n \times n}$ ortogonális mátrixok (oszlopaik ortogonális rendszert alkotnak), továbbá a Σ mátrix M-mel megegyező méretű és a főátlóban elhelyezkedő

$$\sigma_1 \geq \sigma_2 \cdots \geq \sigma_r > 0$$

pozitív számokat csupa 0 *követ és a többi elem szintén* 0. *A* σ_i *számokat szinguláris értékeknek nevezzük, és a* $\sigma_i = 0$ *választással terjesszük ki az* $i > r$ *esetre.*

A szinguláris felbontás sematikus vázlata:

$$M_{m\times n} = \begin{pmatrix} | & & | \\ u_1 & \dots & u_m \\ | & & | \end{pmatrix} \cdot \begin{pmatrix} \sigma_1 & & & & & \\ & \ddots & & & & \\ & & \sigma_r & & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \end{pmatrix} \cdot \begin{pmatrix} - & v_1^T & - \\ & \vdots & \\ - & v_n^T & - \end{pmatrix}$$

2.4. Adatok transzformálása

Az adatok transzformálásával a kiinduló adatokat a bányászat céljainak megfelelő alakra hozzuk. A legfontosabb dolog amire oda kell figyelnünk az, hogy közben ne veszítsünk információt. Néhány módszer a transzformálásra:

- *simítás* (korábban foglalkoztunk vele)
- *összevonás*:
  összegzési vagy összevonási műveleteket hajtunk végre az adatokon. Például: napi adatokat szeretnénk összevonni havi vagy éves adatokká.
- *általánosítás*:
  szintén egyfajta összesítést, összevonást jelent. Például: a lehetséges érdemjegyeket (1-5) leképezzük megfelelt, illetve nem megfelelt minősítésekre.
- *normalizálás*:
  úgy skálázzuk át az adatokat, hogy egy előre meghatározott, kis méretű tartományba essenek. Ez különösen fontos lehet összehasonlításokat használó algoritmusoknál (pl.: távolság alapú módszerek esetében kiküszöbölhetjük, hogy az eredetileg nagy értékkészlettel rendelkező attribútumok mellett eltörpüljenek a kis értékkészletűek).

A *min–max normalizálás* lineárisan transzformálja az adatokat, megőrzi az adatok közti összefüggéseket:

$$\nu' = \frac{\nu - min_A}{max_A - min_A}(newmax_A - newmin_A) + newmin_A,$$

ahol min_A és max_A az A attribútum minimális és maximális értéke.

Példa: Tegyük fel, hogy a vizsgált attribútumunk minimális értéke 12000, a maximális értéke 96000. Szeretnénk az attribútum értékét a $[0, 1]$ intervallumra képezni. Ekkor min-max normalizálással egy 76300 érték a következőképp transzformálódik:

$$\frac{73600 - 12000}{96000 - 12000}(1 - 0) + 0 = 0.733.$$

A *standardizálás*t akkor érdemes alkalmazni, ha nem ismerjük a korlátokat, de az átlagot és az értékek szórását igen. Kiszámítása:

$$\nu' = \frac{\nu - \overline{A}}{\sigma_A},$$

ahol $\overline{A}$ az A attribútum átlaga, σ_A a szórás értéke.

A *decimális skálázású normalizálás* a tizedesvesszőt helyezi át, melynek mértéke az attribútum legnagyobb abszolút értékétől függ:

$$\nu' = \frac{\nu}{10^j},$$

ahol j a legkisebb olyan egész szám, melyre

$$Max(|\nu'|) < 1.$$

- *új attribútumok konstrukciója*:
  olyan adatokat alkotunk, melyek a korábbiakból valamilyen művelettel megkaphatók. Például: ismert magasság és szélesség attribútumok alapján kiegészíthetjük adatainkat a terület attribútummal.

3. fejezet

Osztályozás

Most, hogy a KDD alapján előírt első négy lépésen túl vagyunk, rátérhetünk a tényleges adatbányászatra, azonbelül is az osztályozással foglalkozunk a dolgozat további részében.

Az osztályozás az adatbányászat egyik leggyakrabban alkalmazott területe. Az osztályozási feladatok során előre definiált osztályokba soroljuk a különböző mintákat. Annak az attribútumnak az értékei határozzák meg ezeket az előre definiált osztályokat, amely szerint osztályozni szeretnénk az adatainkat. Ezt a kiválasztott attribútumot nevezzük *osztálycímkének*. Ha ez folytonos, akkor előrejelzésről, egyébként *klasszifikációról* beszélünk. A klasszifikáció *felügyelt tanulás*, hiszen a modell tanítása során tudjuk, hogy az egyes tanulóminták (azaz a rendelkezésre álló adatok olyan részhalmazai, ahol az osztálycímke minden rekord esetén ismert) melyik osztályba tartoznak. Ezzel szemben klaszterezés (csoportosítás) során nem ismerjük előre a csoportokat, ezt *felügyelet nélküli tanulásnak* nevezzük.

A gyakorlatban számos alkalmazása ismert az osztályozási módszereknek. Használható például hitelkockázat becslésére: egy bank korábbi tapasztalatai alapján felépíthet egy olyan modellt, melynek segítségével osztályozni tudja ügyfeleit a hitelfolyósítás kockázatának függvényében. A modell segítségével új ügyfél esetén a bank meg tudja állapítani, hogy esetében mire számíthatnak, fizetné-e rendesen a törlesztőrészleteket vagy sem, ezáltal mérlegelni tud, folyósítja-e a hitelt vagy sem.

Az osztályozás alkalmazható még például adócsalás vagy biztosítási csalás felderítésére, kéretlen levelek kiszűrésére, biztosítási kockázatbecslés és díjszabás kialakítására, hálózati betörés detektálására, meghibásodás előrejelzésére, stb.

Az osztályozási módszerek rendszerint három lépésből állnak:

1. *Modellkészítés:*
   először meghatározzuk az osztálycímkét. Feltételezzük, hogy az attribútumok függnek az osztálycímkétől, hiszen ha ez nem így lenne, akkor nem tudnánk előrejelezni az új minták ismeretlen címkéjének értékét. Ez az osztályozás *alapvető feltételezése.* Adott lesz objektumok sorozata, melyet *tanító pontoknak* nevezünk, ezekkel tanítjuk a modellt valamilyen osztályozási algoritmus szerint.

2. *A modell ellenőrzése:*
   ennek során ellenőrizzük a pontosságát az elkészített modellünknek, azaz azt vizsgáljuk, hogy a modell milyen arányban osztályozta jól a teszthalmaz elemeit. (A kezdetben rendelkezésünkre álló ismert osztálycímkéjű adatok egy részét használjuk tanításra, a többin pedig tesztelünk.) A tanulás és tesztelés során egyre javítjuk a modellünket, míg az az alkalmazáshoz mérten elfogadható pontosságú nem lesz.

3. *A modell felhasználása:*
   ha sikerült elfogadható pontosságú modellt alkotnunk az előzőekben, akkor már alkalmazhatjuk *előrejelzésre* is. Az előrejelzés az új minta ismeretlen osztálycímkéjének meghatározását jelenti a többi, ismert attribútum értékének függvényében.

Egyes esetekben – például ha kevés adat áll rendelkezésünkre – az ellenőrzést nem tudjuk végrehajtani. Modellt persze ekkor is készíthetünk, de tesztelés hiányában lehetséges, hogy ennek minősége gyenge lesz (ezen lehet segíteni például keresztvalidálással, melyet később részletezünk.)
Amikor osztályozó módszert választunk, célszerű figyelembe vennünk a következő tulajdonságait:

- *Gyorsaság:* a modell felépítésének és használatának költségeire vonatkozik.
- *Robosztusság:* érzékeny-e a modell hiányzó, vagy outlier (kiugró) adatokra?
- *Skálázhatóság:* használható-e a modell nagy adathalmazokra is?

- *Értelmezhetőség:* kinyerhetünk-e az emberek számára értelmezhető tudást a modell belső szerkezetéből?
- *Skála-invariancia:* egy osztályozó eljárást skála-invariánsnak nevezünk, ha a módszer kimenete nem változik abban az esetben, ha tetszőleges intervallum típusú magyarázó változó helyett annak α–szorosát vesszük (ahol α pozitív).
- *Pontosság:*

3.1. Lineárisan szeparálható osztályok

Akkor mondjuk hogy két osztály lineárisan szeparálható, ha pontjaikat egy hipersík segítségével el tudjuk különíteni. Tegyük fel, hogy a tanuló mintákat n-dimenziós numerikus attribútumok írják le, és tekintsünk minden mintát egy-egy n-dimenziós térbeli pontnak. Ekkor a hipersík képlete:

$$\omega_1 a_1 + \omega_2 a_2 + \cdots + \omega_n a_n = 0.$$

Ebből meghatározzuk az ω súlyokat, majd ezekkel az ω–kal súlyozzuk az osztályozandó elem attribútumait. Ha a súlyozott összeg 0–nál nagyobb, akkor az első osztályba tartozik, ha nem, akkor a másodikba.
Adott tanítóponthoz több hipersík is létezhet, amellyel kettéválaszthatjuk az osztályokat.

3.1.1. Perceptron tanulási szabály

Minden attribútumnak valósnak kell lennie. Fel kell vennünk egy extra attribútumot (*bias*), ennek értéke minden tanítópont esetén 1. Kezdetben a súlyvektorunk $((\omega_1,\ldots,\omega_n))$ a konstans 0 vektor. A továbbiakban ezt módosítjuk a tanító pontok függvényében addig, míg végül a súlyvektorunk minden pontot jól szeparál.

Az algoritmus:

Require: τ: tanítópontok halmaza

$$\vec{\omega} = (0, 0, \ldots, 0)$$

while van rosszul osztályozott $\vec{t} \in \tau$ **do**
  for all minden $\vec{t} \in \tau$ **do**
    if $\vec{t}$ rosszul van osztályozva **then**
      if $\vec{t}$ az első osztályba tartozik **then**

$$\vec{\omega} = \vec{\omega} + \vec{t}$$

      else

$$\vec{\omega} = \vec{\omega} - \vec{t}$$

      end if
    end if
  end for
end while

Ha az algoritmus során rosszul osztályozott ponttal találkozunk, akkor módosítjuk a hipersíkot, mégpedig úgy, hogy a problémás tanítópont közelebb kerül hozzá, vagy akár át is kerülhet a sík másik oldalára.
Tegyük fel, hogy a rosszul osztályozott tanítópont az első osztályba tartozik! Ekkor a módosítás után az attribútum értékeinek a súlyozott összege:

$$\sum(\omega_i + t_i)t_i$$

lesz. Emlékezzünk vissza, hogy korábban (a módosítás előtt) ez az érték

$$\sum \omega_i t_i$$

volt. A különbség biztosan pozitív, hiszen négyzetöszeg, vagyis a hipersík a módosítás során jó irányba mozgott.

Lemma: *A perceptron tanulási algoritmus véges lépésben leáll, amennyiben az osztályok lineárisan szeparálhatók.*

Ha a tanítópontok nem lineárisan szeparálhatóak, akkor az algoritmus nem áll le. Ebben az esetben érdemes megadni egy maximális iterációs számot, melynek elérése után az algoritmus sikertelen üzenetet ad.

3.1.2. A Winnow módszer

Ezt az eljárást bináris attribútumok esetén használhatjuk. Nagyon hasonlít az imént tárgyalt perceptron tanuláshoz, az eltérés annyi csupán (leszámítva azt, hogy a kezdeti súlyvektorunk most a konstans 1 lesz), hogy rossz osztályozás esetén a súlyvektor bizonyos elemeit megszorozzuk vagy elosztjuk – attól függően, hogy melyik csoportba tartozik – egy α konstanssal, melynek értéke 1- nél nagyobb. Azokat a pozíciójú elemeket változtatjuk, melyekre a tanítópont vektora egyest tartalmaz. Egy $\vec{a}$ pontot az első osztályba sorol az osztályozó, ha

$$\omega_1 a_1 + \omega_2 a_2 + \cdots + \omega_n a_n > \theta,$$

ahol θ egy előre meghatározott konstans. Súlyvektorunk minden eleme mindig pozitív marad, hiszen kezdetben a konstans 1 vektor alkotja, a későbbiekben változást okozó α értéke pedig pozitív.

Előfordulhat olyan eset is, amikor negatív súlyokat is meg kell engednünk. Ekkor alkalmazhatjuk a Winnow módszer egy változatát, a *kiegyensúlyozott* (balanced) Winnow módszert.

3.1.3. A Rocchio-eljárás

A módszer feltételezi, hogy minden attribútum valós típusú. Minden c kategóriához megalkotunk egy *prototípusvektort*, majd ehhez hasonlítjuk az ismeretlen minta vektorát. A prototípusvektort a $\mathcal{D}_c$ tanulópéldák átlagaként kapjuk. A távolságot egy prototípusvektor és egy osztályozandó objektum között valamilyen távolságmértékkel számolhatjuk. Az eljárás előnye, hogy kicsi a számításigénye, ezért nagyon gyors a tanulás, hátránya viszont, hogy ha az egy osztályba tartozó pontok nem jellemezhetőek egyetlen vektorral, akkor rossz eredményt ad.

Lényegesen javítható a módszer hatékonysága, ha a negatív tanuló adatokat is figyelembe vesszük a prototípusvektorok megalkotásánál. Ekkor c prototípusvektora az alábbi képlettel számolható:

$$\vec{c} = \beta \cdot \sum_{j \in \mathcal{D}_c} \vec{d_j} - \gamma \cdot \sum_{j \notin \mathcal{D}_c} \vec{d_j}$$

A pontok centroidjaként számolt prototípusvektort abban az esetben kapjuk meg, ha a β értékének az 1-et, a γ értékének pedig a 0-t választjuk.

Még további javulás érhető el a hatékonyság terén, amennyiben a második tagban az összes negatív tanulópélda helyett csak azoknak az átlagát vesszük, melyek közel vannak a pozitívakhoz.

3.2. A k-NN algoritmus

A k–NN (k-*nearest neighbours*) vagy k-legközelebbi szomszéd módszer is egy „lusta" klasszifikáló eljárás (az előzőekhez hasonlóan), hiszen nem épít modellt. Azon az elgondoláson alapszik, hogy a hasonló attribútumú objektumok hasonló tulajdonságokkal rendelkeznek. A hasonlóság nagyságát távolságfüggvénnyel mérjük.
Továbbra is tegyük fel, hogy a tanuló mintákat n-dimenziós numerikus attribútumok írják le, és tekintsünk minden mintát egy-egy n-dimenziós térbeli pontnak. Egy új minta osztályozásához az említett n-dimenziós térben megkeressük azt a k tanuló mintát, melyek a legközelebb vannak az ismeretlen mintához. A „közelséget" definiáljuk az euklideszi távolság segítségével:

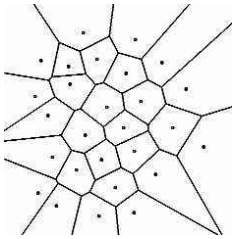
$$d(X,Y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}.$$

A fenti képlettel kapjuk tehát az X és Y n–dimenziós pontok euklideszi távolságát. Ha megtaláltuk a k–legközelebbi szomszédot, akkor az ismeretlen mintát abba az osztályba soroljuk, amely a k szomszéd között a leggyakoribb.
A legegyszerűbb esetben, azaz ha k értéke 1, akkor az ismeretlen minta abba az osztályba fog tartozni, amelyikbe a legközelebbi szomszédja is van. Ezt a módszert szokták 1-NN algoritmusnak is nevezni.
Abban az esetben, ha k értéke nagyobb 1-nél, valamint az osztálycímke csupán két értéket vehet fel, akkor adjunk k-nak páratlan értéket, hiszen így minden esetben kialakul a k-legközelebbi szomszéd osztálycímke értékei közt egy többség, melyet használhatunk.

A k-NN technika alkalmazása során mindig fontos kérdés a k érték megválasztása. Amennyiben ez lehetséges, célszerű többféle k értékkel is megvizsgálni az eredményt.
A módszer ábrázolásánál (k=1 esetén) kedvelt eszköz a *Voronoi–diagram*:
tartományokra osztjuk a felületet úgy, hogy minden ilyen tartományba pontosan egy

tanító pont essen, továbbá teljesülnie kell annak a feltételnek is, miszerint bármely tartományon belüli pont a tanítópontok közül a tartomány tanítópontjához van a legközelebb.

Az alábbi ábrán láthatunk példát Voronoi–diagramra.



3.1. ábra. Voronoi-diagram

A módszer egyik nagy hátránya, hogy érzékeny a független attribútumokra. Ennek kiküszöbölésére megoldást jelenthet a következők valamelyike:

- használjunk több tanítópontot, ha tehetjük;
- kérdezzük meg az alkalmazási terület egy szakértőjét, hogy mely attribútumokat érdemes figyelembe venni a távolság meghatározásánál;
- a függetlenség megállapítására alkalmazzunk statisztikai tesztet;
- ha nincs sok attribútumunk, akkor az összes attribútum részhalmaz esetén határozzuk meg az osztályozás pontosságát, majd válasszuk ezek közül a legjobbat;
- alkalmazhatunk egy mohó algoritmust, amely egyesével bővítené a tesztelendő attribútumhalmazt úgy, hogy az a legjobb osztályozást adja, leállunk, ha az osztályozás minősége már nem javul;
- használjunk csökkentő módszert, mely a teljes attribútumhalmazból indul ki, majd egy attribútumot dob ki minden lépésnél.

Mivel a módszer érzékeny a távolság definíciójára, így természetesen érzékeny a mértékegységekre is. A k-NN módszer nem skálainvariáns.

A k-legközelebbi szomszéd egy módosítása adja a megoldást egy másik problémára, miszerint az ismeretlen minta osztálycímkéjének meghatározásához az alap módszer mind a k szomszédot ugyanolyan fontosnak tekinti. Érezzük hogy ez nem jó, hiszen a távolabbi szomszédok kevésbé, a közelebbiek pedig jobban „hasonlítanak" az új mintához. A *súlyozott legközelebbi módszer* esetén a k szomszéd minden tagjának akkora a súlya, amekkora az osztályozandó ponttól mért távolságának az inverze, azaz:

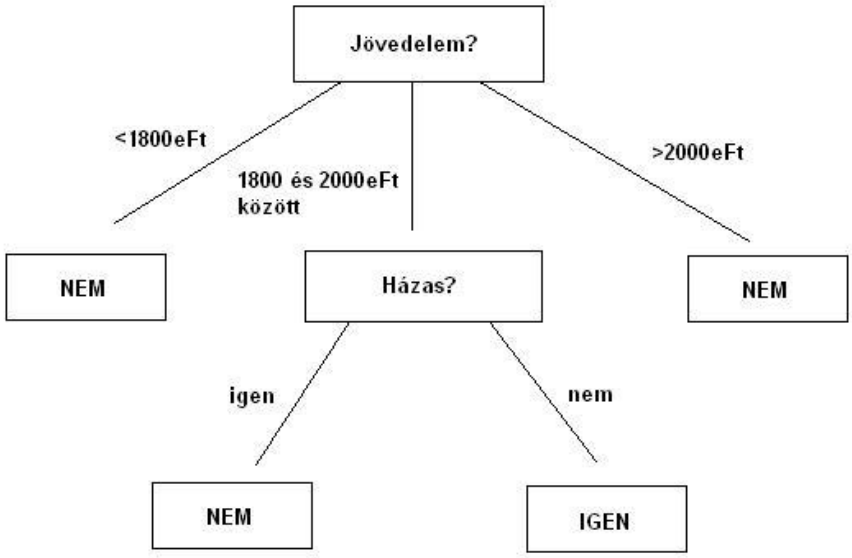
$$w = \frac{1}{d(y, x_i)^2}.$$

Tehát az osztályozandó ponthoz közelebb eső tanítópontoknak nagyobb szavuk van a végső osztály meghatározásában, mint a távolabbiaknak.

3.3. Döntési fák

Az osztályozási feladatok egyik legismertebb eszköze a *döntési fa*, melynek alapötlete, hogy bonyolult összefüggéseket egyszerű döntések sorozatára vezet vissza. A döntési fa egy fa formájú folyamatábra, ennek gyökeréből indulunk egy ismeretlen minta klasszifikálásakor. Fentről lefelé haladva a belső csúcsok egy-egy attribútum értékének vizsgálatát jelentik, a csúcsok közötti élek e vizsgálat eredményével (az adott attribútum megfelelő értékével) vannak megcímkézve, míg a levélcsúcsok magát a döntést (vagyis a megfelelő osztályt reprezentáló attribútum értékét) ábrázolják.

A döntési fákból könnyedén nyerhetünk *osztályozási szabályok*at, melyek összességét *szabálybázis*nak nevezzük. Az egyes szabályokat a gyökértől egy-egy levél felé vezető útvonal feltételeinek összeolvasásával kapjuk. Klasszifikáció esetén azt is láthatjuk, hogy milyen szabályok vezetnek a végső döntéshez, azaz hogy a döntési fa az új minta osztálycímkéjének értékét mi alapján határozza meg. A döntési fák igen népszerűek. Nem véletlenül, hiszen számos további jó tulajdonsággal rendelkeznek. Például: automatikusan felismerik a lényeges változókat, ezeket a gyökér közelében, míg a kevésbé fontosakat a levelekhez közel tesztelik. Előfordulhat, hogy egyes attribútumok nem jelennek meg a fában, hiszen nem befolyásolják a döntést. Ezeket *irrelevánsnak* tekintjük, a fában szereplő attribútumok pedig a *csökkentett attribútumhalmazt* adják. Ez alapján dimenziócsökkentésre is használhatjuk. Egy másik



3.2. ábra. Példa döntési fára (potenciális adócsalók kiszűrése)

nagy előnyük a *skálázhatóság*, vagyis hogy hatékonyan alkalmazhatóak nagyszámú adatminta esetén is.
A gyakorlatban sokszor használunk *bináris döntési fákat*, melyek sajátossága, hogy minden csomópontnak két gyermeke van. Mivel tetszőleges nem bináris döntési fa könnyedén átalakítható binárissá, így sok algoritmus csak bináris döntési fát tud előállítani.

3.3.1. A döntési fák elkészítése

A fa generálása egy partíciós folyamat. Kezdetben egyetlen csúcsunk van, a gyökér, ez tartalmazza a teljes tanító adathalmazt. Válasszuk ki azt az attribútumot, amely a legjobban szeparálja a halmazt, ez lesz a gyökérben az „ellenőrző" attribútum. Az attribútum minden lehetséges értékéhez egy-egy elágazást készítünk, majd ezek szerint osztjuk szét a mintákat. Az algoritmus a minták minden egyes partíciójánál rekurzívan ugyanezt az eljárást alkalmazza. Ha egy vizsgált csúcsban szereplő minden minta ugyanabból az osztályból való, akkor az a csúcs levél lesz. Amennyiben egy csúcsnál kiválsztottunk egy attribútumot, akkor azt a továbbiak-

ban már egyetlen leszármazottjánál sem kell figyelembe vennünk.

Mikor ér véget a rekurzió?
Amennyiben az alábbi feltételek valamelyike teljesül:

- Egy adott csomópont minden eleme ugyanabba az osztályba tartozik.
- Nem maradt olyan attribútum, ami alapján a mintát tovább bonthatnánk. Ebből a csúcsból ilyen esetben levél lesz, melynek osztálycímkéje annak az osztálynak a címkéjével egyezik meg, melyhez a legtöbb tanítópont tartozik (többségi szavazás elve).
- Nem tartozik az adott csomóponthoz tanítópont (már az összes adatpontot osztályba soroltuk).
- A fa mélysége elért egy előre megadott korlátot.
- A csomópontban szereplő pontok száma kevesebb egy adott korlátnál.

Hogyan válasszuk ki az adott csúcsnál azt az attribútumot, amely alapján a tesztet (azaz az adatok felosztását) végezzük?
Általában az *információnyereség elvét*, vagy a *Gini indexet* használjuk. Mindkettővel bővebben foglalkozunk a következőkben.

3.3.2. Az ID3 algoritmus

Az ID3 (Interactive Dichotomizer 3) az egyik legismertebb és legrégebbi (1960-as évek) osztályozó algoritmusunk. A tesztattribútum kiválasztásához az információnyereség elvét alkalmazza (entrópia csökkenés), vagyis azt az attribútumot választja, amely a legnagyobb információnyereséggel bír.
Vezessük be az *entrópia* fogalmát!
Az entrópia az Y-nal kapcsolatos bizonytalanságunkat fejezi ki.

2. Definíció. *Ha Y egy l lehetséges értéket p_i valószínűséggel felvevő valószínűségi változó, akkor Y Shannon-féle entrópiáján a*

$$H(Y) = H(p_1, \ldots, p_k) = -\sum_{j=1}^{l} p_j \log_2 p_j$$

számot értjük.

Tegyük fel, hogy lehetőségünk volt egy X változót megfigyelni, ennek értéke tapasztalataink szerint x_i. Ekkor az Y-nal kapcsolatos bizonytalanságunk nagysága:

$$H(Y|X = x_i) = -\sum_{j=1}^{k} \mathbb{P}(Y = y_j|X = x_i) \log_2 \mathbb{P}(Y = y_j|X = x_i)$$

Azaz a várható bizonytalanságunk abban az esetben, ha lehetőségünk van X-et megfigyelni:

$$H(Y|X) = \sum \mathbb{P}(X = x_i)H(Y|X = x_i)$$

Eszerint X megfigyeléséből adódó információnyereség:

$$I(Y, X) = H(Y) - H(Y|X),$$

azaz ennyi információt hordoz X az Y-ról (ennyivel csökken az entrópia).
Az algoritmus minden attribútum esetén kiszámítja az entrópia csökkenését, majd végül azt a változót választja ki, melyre ez az érték a legnagyobb volt, azaz amelyre $I(Y, X)$ maximális (vagyis $H(Y|X)$ minimális). Ezen attribútum szerint ágazik szét az ID3 algoritmus annyi felé, ahány értéke van a kiválasztott attribútumnak.

Probléma: Előfordulhat, hogy a kiválasztott attribútumok sok értéket vehetnek fel, ezáltal terebélyes fát kapunk. *Mit tehetünk ilyenkor?*
Használhatunk például *nyereségarány mutatót* (gain ratio):

$$gain\_ratio(X) = \frac{I(Y, X)}{H(X)}$$

Ez azért jó, mert figyelembe veszi hogy mennyi tanítópont kerül a gyerek csomópontokba és „bünteti" azokat az attribútumokat, amelyek túl sok gyereket hoznak létre.

Az ID3 nem alkalmaz visszalépést a keresés közben, azaz nincs lehetőség rá, hogy ha egy adott pontnál kiválasztottunk egy attribútumot, akkor utólag megváltoztassuk a döntést. Az algoritmus lokális optimumok alapján dolgozik, emiatt nem biztos, hogy megtalálja a globálisan optimális megoldás(oka)t, ugyanakkor nagy előnye, hogy minden lépésben statisztikán alapuló döntést hoz, ezáltal sokkal kevésbé érzékeny a tanulási példák egyedi hibáira (zajos tanulás).

Az ID3 algoritmus egy továbbfejlesztett változata a C4.5 algoritmus. Nagy előnye az ID3 algoritmussal szemben, hogy már nemcsak kategorikus, hanem numerikus értékeket is képes kezelni, automatikusan diszkretizálja az értékeket.

3.3.3. A CART algoritmus

Döntési fák előállítására az ID3 algoritmus mellett gyakran használják a CART módszert (Classification And Regression Trees).
A két algoritmus között a legfőbb eltérések a következők:

- A döntési fa felépítésénél a CART algoritmus bináris döntéseket használ az egyes csomópontokban.
- Még lényegesebb eltérés, hogy míg az ID3 algoritmus az információnyereség elvét használja a tesztattribútum kiválasztásához, addig a CART algoritmus a Gini indexet veszi alapul.

A következőkben nézzük meg hogyan lehet felépíteni egy döntési fát a Gini index segítségével!

3. Definíció. *Legyen T egy adathalmaz, mely N mintát tartalmaz. Ezeket a mintákat soroljuk n különböző osztályba. Ekkor a T adathalmaz Gini indexe az alábbi öszefüggés alapján számolható:*

$$gini(T) = 1 - \sum_{j=1}^{n} p_j^2,$$

ahol p_j a j. osztály relatív előfordulását jelöli a T halmazban.

Célunk továbbra is azon attribútum kiválasztása az egyes csomópontokban, melynek segítségével legjobban szeparálhatjuk az adatpontjainkat, azaz amely választás mellett a leginkább növekszik a következő szinten az adatok homogenitása.

Osszuk fel az aktuálisan vizsgált T adathalmazt az N_1 elemszámú T_1 és az N_2 elemszámú T_2 diszjunkt halmazokra. Ekkor a Gini indexet az alábbi összefüggésből kaphatjuk:

$$gini(T) = \frac{N_1}{N} \times gini(T_1) + \frac{N_2}{N} \times gini(T_2).$$

(A gini(T_1) és a gini(T_2) értékeket a Gini index definíciójából számíthatjuk.)

Határozzuk meg egy adott csomópontban minden attribútum szerinti felosztáshoz a Gini indexet, majd válasszuk ki ezek közül azt a változót, amelynél a kapott érték a legkisebb volt.

3.3.4. Fametszés

A fametszés, vagy a felépített fa tisztítása során az algoritmus törli azokat az ágakat, melyek *túltanítás* miatt jöttek létre. Akkor beszélünk túltanításról, ha a modellünk túlságosan illeszkedik a tanuló mintákra, megtanulja az azokban található zajokat és hibákat is, ezáltal az ismeretlen minták osztályozásánál gyenge pontosságot kapunk. Elkerülésének két megközelítése lehet:

- *előmetszés*: hamarabb leállítjuk a fa építését. Például egy adott csúcsnál úgy döntünk, hogy ennél tovább már nem partícionáljuk a tanuló minták részhalmazát (mondjuk mert nem eredményezne jelentős javulást). Ez a csúcs levél lesz, osztálycímkéje lehet a részhalmaz mintáinak osztályai közül a leggyakoribb, vagy azok valószínűségi eloszlása.

- *utómetszés*: felépítjük a teljes fát, majd eltávolítjuk a felesleges ágakat. Ezt tehetjük például a *költség bonyolultsága* szerint, lényegében az algoritmus a fa összes nem levél csúcsára kiszámítja abban az esetben a várható hibaarányt, ha lemetszenénk a csúcshoz tartozó részfát, majd abban is ha nem. Ha nagyobb hibaarányt eredményez a csúcs megmetszése, akkor a részfát megtartjuk, ha nem, akkor levágjuk.

Az előmetszés hátránya, hogy nehéz megtalálni azt az értéket, ami alatt már nem vágunk tovább. Hiszen ha túl kicsire választjuk, akkor igen nagy fát kaphatunk, ellenkező esetben viszont fontos információkat veszíthetünk. A gyakorlatban általában az utómetszés a megbízhatóbb, ezért sok fát építő algoritmus ezt preferálja (köztük a fent bemutatott ID3 és CART is).

3.4. Metaklasszifikátorok

Osztályozók kombinálásával, úgynevezett *metaklasszifikátorokkal* javíthatunk az osztályozóinkon, és csökkenthetjük a túltanulás mértékét. A következőkben a Bagging és a Boosting módszerekkel foglalkozunk a metaklasszifikátorok közül.

3.4.1. Bagging

A Bagging (**B**ootstrap **agg**regat**ing**) algoritmus a *„bootstrap"* adatbázisokon tanított osztályozók alapján választja ki az osztálycímkét. A bootstrap a tanító adatot ismétléses rekordkiválasztással választja egyenletes eloszlás szerint. Annak a valószínűsége, hogy egy adott rekordot kiválasztunk:

$$1 - \left(1 - \frac{1}{N}\right)^m,$$

ahol N a rekordok száma, m pedig a választásoké.

$$1 - \left(1 - \frac{1}{N}\right)^m \to 1 - \frac{1}{e} \approx 0.632 \qquad \text{ha} \quad m \approx N \quad \text{és} \quad N \to \infty.$$

Nagy előnye, hogy nem kell számontartanunk az eddigi választásainkat, ez sok választás esetén főleg hasznos.

A Bagging futtatása során T darab bootstrap adatbázist generál, és minden adatbázison egy C osztályozót készít. A kombinált osztályozó ezekből épül fel, kimenete pedig az az osztálycímke lesz, amely a legtöbbször előfordult az osztályozóknál.
A fent kapott 0.632-es érték miatt az így tanított osztályozók különbözőek lesznek nem stabil osztályozók esetén (mint például a döntési fák), ezek kombinálásával növelhető a rendszer hatékonysága.

3.4.2. Boosting

Boosting esetén a bootstrap mintát nem egyenletes eloszlás szerint választjuk az ismert célváltozójú adatokból, nagyobb valószínűséggel választjuk bele a súlyozott bootstrap mintába azt, amit éppen rosszul klasszifikálunk.

A Boosting lépései (egy fázis):

1. Mintavételezés
2. Klasszifikátor tanítása
3. Kiértékelés
4. Átsúlyozás

+Aggregáció

A Boosting egy konkrét megvalósítása az **AdaBoost** (Adaptív Boosting) algoritmus. Adaptív abban az értelemben, hogy az új osztályozó hipotézist az előzőekben meghatározott hipotézisek hibái alapján konstruáljuk meg.

A továbbiakban legyenek (x_j, y_j) adatok (j=1,...,N).

A c_i klasszifikátor hibája:

$$\varepsilon_i = \sum_{j=1}^{N} \omega_j \delta(c_i(x_j) \neq y_j).$$

A „fontosság":

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

A t. fázis után a súlyozás újraszámolása:

$$\omega_j^{(t+1)} = \frac{\omega_j^{(t)}}{z_t} \cdot \begin{cases} e^{-\alpha_t}, & \text{ha } c_t(x_j) = y_j \quad \text{(jó a klasszifikáció)} \\ e^{\alpha_t}, & \text{ha } c_t(x_j) \neq y_j \quad \text{(nem jó a klasszifikáció)} \end{cases},$$

ahol z_t szerepe a normalizálás:

$$\sum_{j=1}^{N} \omega_j^{(t+1)} = 1.$$

Az alábbiakban nézzük meg hogy működik az algoritmus!

1. *Inicializálás:*
$$\omega_j^{(1)} = \frac{1}{N} \qquad k \text{ fázis} \quad i : 1 \to k;$$
2. *Mintavételezés:* Elkészítjük D_i-t D-ből a ω_j^i súlyokkal;
3. Tanítjuk c_i-t D_i-n;
4. Teszteljük c_i-t az egész D-n, ebből nyerjük ε_i-t

5. Ha $\varepsilon_i > 0.5$, nem fogadjuk el, visszatérünk a (2)-es lépéshez úgy, hogy i értékét nem növeljük.

6. α_i és updateljük a súlyokat.

+Aggregálás:

$$c^*(x) = argmax \sum_{i=1}^{k} \alpha_i \delta(c_i(x) = y)$$

3.5. Osztályozók hatékonyságának mutatószámai

A legfontosabb az osztályozó pontossága, mely során összehasonlítjuk a teszt minták valós osztálycímkéit az osztályozó által jósolt osztálycímkékkel. A kapott eredményeket ún. *confusion* (tévesztési vagy keveredési) *mátrixban* rögzíthetjük. A mátrix annyi sorból és oszlopból áll, amennyi osztály van. Az i-edik sor j-edik eleme adja meg azoknak a pontoknak a számát, amelyeket az osztályozó a j-edik osztályba sorol, holott azok az i-edik osztályba tartoznak. A főátlóban található a helyesen osztályozott pontok száma.

A pontosság értéke azonban megtévesztő lehet. Például: nézzünk egy bináris esetet, tegyük fel, hogy az egyik változó 90%-os valószínűséggel fordul elő. Ekkor egy 80%-os osztályozó rossznak mondható, hiszen nagyobb pontosságot ad a véletlen osztályozó, melynek várható pontossága 82%.
Erre jelent megoldást az osztályozó *kappa statisztikája*, mely az osztályozó pontosságát a véletlen osztályozóéhoz hasonlítja.

4. Definíció. *Egy osztályozó kappa statisztikája:*

$$\frac{T - M}{N - M}, \qquad ahol \quad N = \sum_{i=1}^{k} n_i, \quad M = \sum_{i=1}^{k} n_i p_i,$$

T a helyesen osztályozott pontokat, p_i az egyes osztályok relatív gyakoriságait jelöli, n_i pedig az osztályok előfordulása a tanítóhalmazon. A kappa statisztika értéke 0 *és* 1 *közé esik (véletlen osztályozó esetén* 0*, tökéletes esetén* 1*).*

Bináris esetben a keveredési mátrix a következő:

	1	0
1	TP	FN
0	FP	TN

A mátrixban TP (True Positive) és TN (True Negatív) jelöli a helyesen osztályozott pontokat attól függően, hogy melyik osztályba tartoznak. A keveredési mátrix segítségével könnyen számítható a *modell becslési pontossága* az alábbi szerint:

$$\frac{TP + TN}{TP + FN + FP + TN}$$

További mutatók:

- *Hibaarány:*
$$error - rate = 1 - \frac{TP + TN}{TP + FN + FP + TN}$$
- *Szenzitivitás:*
$$TPR = \frac{TP}{TP + FN}$$
- *Specificitás:*
$$TNR = \frac{TN}{TN + FP}$$
- *Precision:*
$$P = \frac{TP}{TP + FP}$$
- *Recall (Felidézés):*
$$r = \frac{TP}{TP + FN}$$
- *F-mérték:* az utóbbi két érték parametrikus harmonikus közepe:
$$F = \frac{1}{\alpha \frac{1}{P} + (1 - \alpha)\frac{1}{r}}.$$

A modell pontossága nagyban függhet az ellenőrző minták megválasztásától. Ezért az osztályozók pontosságának tárgyalásakor fontos kérdés az is, hogy az ismert osztálycímkéjű adatokat hogyan válasszuk szét tanító és tesztelő mintákra. A következőkben áttekintjük a leggyakrabban használt módszereket.

1. *Véletlen vágás:* az ismert osztálycímkéjű adatokat két diszjunkt halmazra osztjuk fel, a felosztást véletlenszerűen végezzük, a kapott halmazok közül az egyik (általában a nagyobbik) lesz a tanító adatok halmaza, a másikat pedig tesztelésre használjuk.Többször megismételhetjük, a kapott pontossági értékek átlagát tekintjük a modell pontosságának.

2. *n-szeres keresztvalidáció:* az S adathalmazunkat véletlenszerűen n darab, közel azonos részre osztjuk, majd $S \setminus S_i$-n tanítunk, S_i-n pedig tesztelünk. A részekre osztást n-féleképpen tehetjük meg. Minden lehetséges esetben felépítjük a modellt, majd a modell végső pontosságának meghatározásához vesszük a kapott pontosságok átlagát.

3. *Leave-on-out:* az n-szeres keresztvalidáció egy speciális esete. Ekkor n megegyezik az ismert minták számával, vagyis minden lépésben egy mintával tesztelünk, a többin pedig tanítunk. Előnye a keresztvalidációval szemben, hogy nincs benne véletlenszerűség, ezért minden futtatáskor ugyanazt az eredményt adja, hátránya viszont, hogy nem használható nagy számú minta esetén (hiszen például 10000 ismert minta esetén 10000-szer kellene lefuttatni, ez azonban elfogadhatatlan).

4. *Bootstrap*: korábban már foglalkoztunk vele.

3.6. Osztályozó módszerek összehasonlítása

A kutatók tudományos munkáikban általában keresztvalidációt használnak. Vesznek n darab adatbázist, ezek mindegyikét véletlenszerűen felosztják r diszjunkt részre, majd ezt a véletlenszerű felosztást ismétlik meg k-szor. Összesen végeznek nrk tanítást, majd tesztelést, végül az ezekből kapott pontosságot használják a *student t-próba* során.

Az osztályozók student próbán alapuló összehasonlítása

Legyen adott N darab teszthalmaz. Jelölje a két összehasonlítandó osztályozó i. teszthalmazon mért pontosságát b_i és l_i, a hozzájuk tartozó pontosságok átlagát

pedig b és l, továbbá legyen

$$d_i = b_i - l_i.$$

Tegyük fel, hogy egy ν pontosságú osztályozó pontossága egy adott teszthalmazon egy ν várható értékű és ismeretlen szórású normális eloszlású valószínűségi változó egy megfigyelésével. Azt kellene eldöntenünk, hogy az eredeti pontosságok statisztikailag eltérnek-e egymástól (a megfigyelt pontosságokat ismerjük). Tehát a nullhipotézisünk a következő:

$$\nu_B = \nu_L \quad \text{vagy} \quad \nu_D = 0, \qquad \text{ahol} \quad D = B - L.$$

A student t-próba (vagy egymintás t-próba) egy ismeretlen várható értékű és szórású normális eloszlású valószínűségi változó várható értékére tett feltételt próbál eldönteni. Számítsuk ki a következő értéket:

$$\frac{\overline{d} - 0}{\sqrt{\sigma_d^{*2} \setminus N}},$$

ahol $\overline{d}$ a mintaátlagot,a σ_d^* az empirikus korrigált szórást jelöli. A kapott értéket vessük össze azzal, ahol a student eloszlás megegyezik $1-\alpha$-val (a próba szintjével). A nagyobb átlaghoz tartozó osztályozó statisztikailag is jobb a másiknál, ha a teszt elutasítja a nullhipotézist.

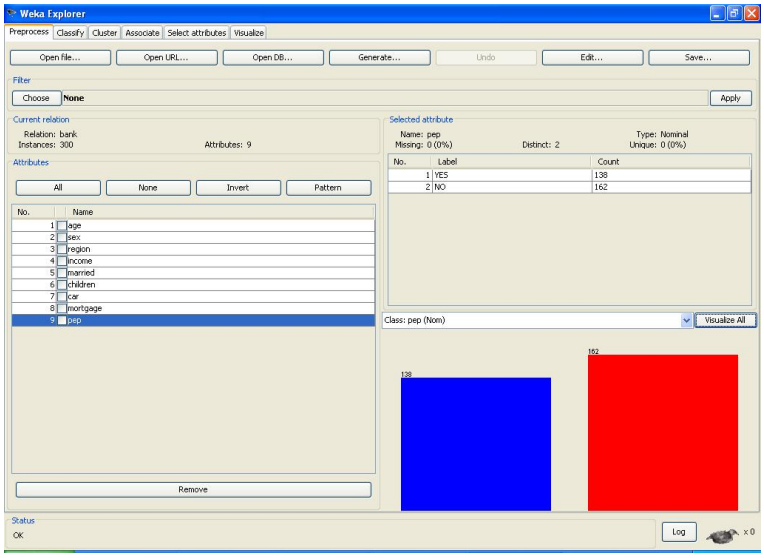
4. fejezet

Gyakorlat

A továbbiakban nézzük meg hogyan is működnek a fentiek a gyakorlatban egy konkrét adathalmazon. Ehhez segítségül hívom a „bank.arff” adathalmazt, amely a

$$http://maya.cs.depaul.edu/classes/ect584/WEKA/classify.html$$

honlapon érhető el. A példa feldolgozása a *WEKA* (Waikato Environment for Knowledge Analysis) 3.6.2-es verziójával történik.



4.1. ábra.

A betöltött adatokat láthatjuk a 4.1-es ábrán.

A vizsgált fájlunk 9 attribútumot tartalmaz, melyek rendre a következők:

- kor (numerikus)
- nem (FEMALE/MALE)
- terület (BELSŐ VÁROS,VÁROS,VIDÉKI,KÜLVÁROSI) (kategorikus)
- jövedelem (numerikus)
- családi állapot (nős/nem nős)
- gyerekek (igen/nem)
- autó (igen/nem)
- jelzálog (igen/nem)
- „fizetési hajlandóság" (igen/nem)

Azaz az adathalmaz egy sora az alábbi alakú:

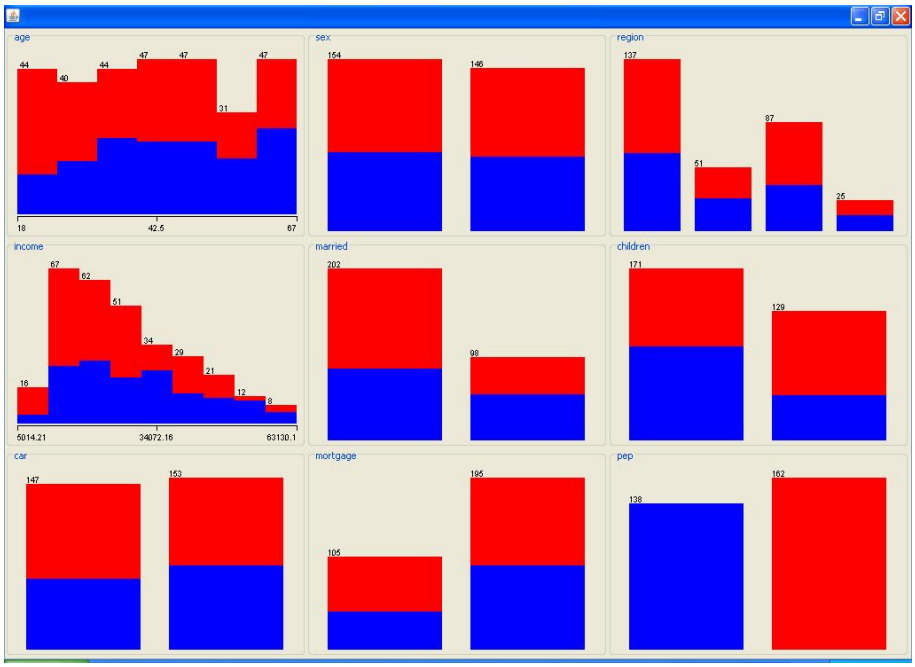
48, FEMALE, INNER CITY, 17546, NO, YES, NO, NO, YES

Adathalmazunk 300 sorból áll, azaz 300 ismert célváltozójú adat áll rendelkezésünkre. Ebben a konkrét esetben az osztályozó attribútum a fizetési hajlandóság lesz, azaz eszerint soroljuk osztályokba az adatokat. Az ábráról leolvashatjuk, hogy 138 adat rendelkezik az igen osztálycímkével (kék szín), a másik 162 pedig a nemmel (piros szín).

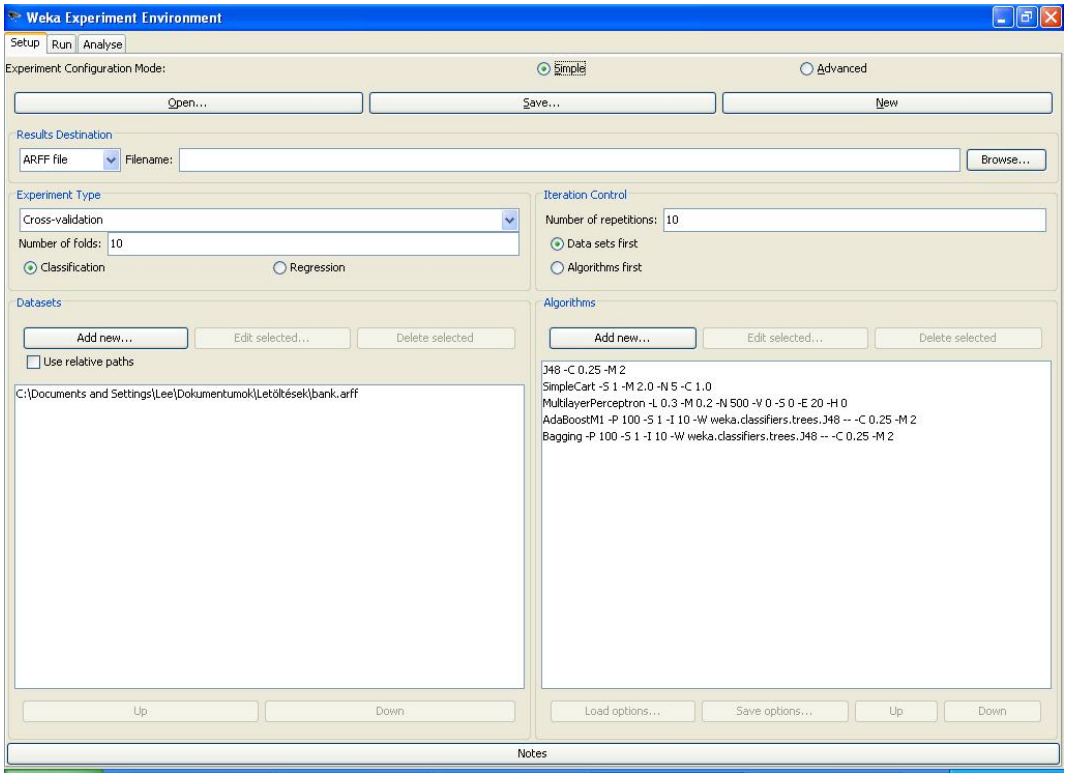
A 4.2 ábrán láthatjuk sorban a különböző attribútumokra lebontva az egyes osztályok eloszlását.

Az ábra alapján egyik attribútum sem tűnik perdöntőnek, hiszen közel azonos a színek aránya, talán a 2. sor utolsó eleme dominánsabb a többinél, ez pedig a „van-e gyereke" változó.

Próbáljuk ki, hogy a „nyers" adathalmazon milyen eredményt kaphatnánk egy osztályozás során. Ehhez hasonlítsuk össze az osztályozó módszereinket, a szignifikánsan legjobbat fogjuk kiválasztani (4.3 ábra).

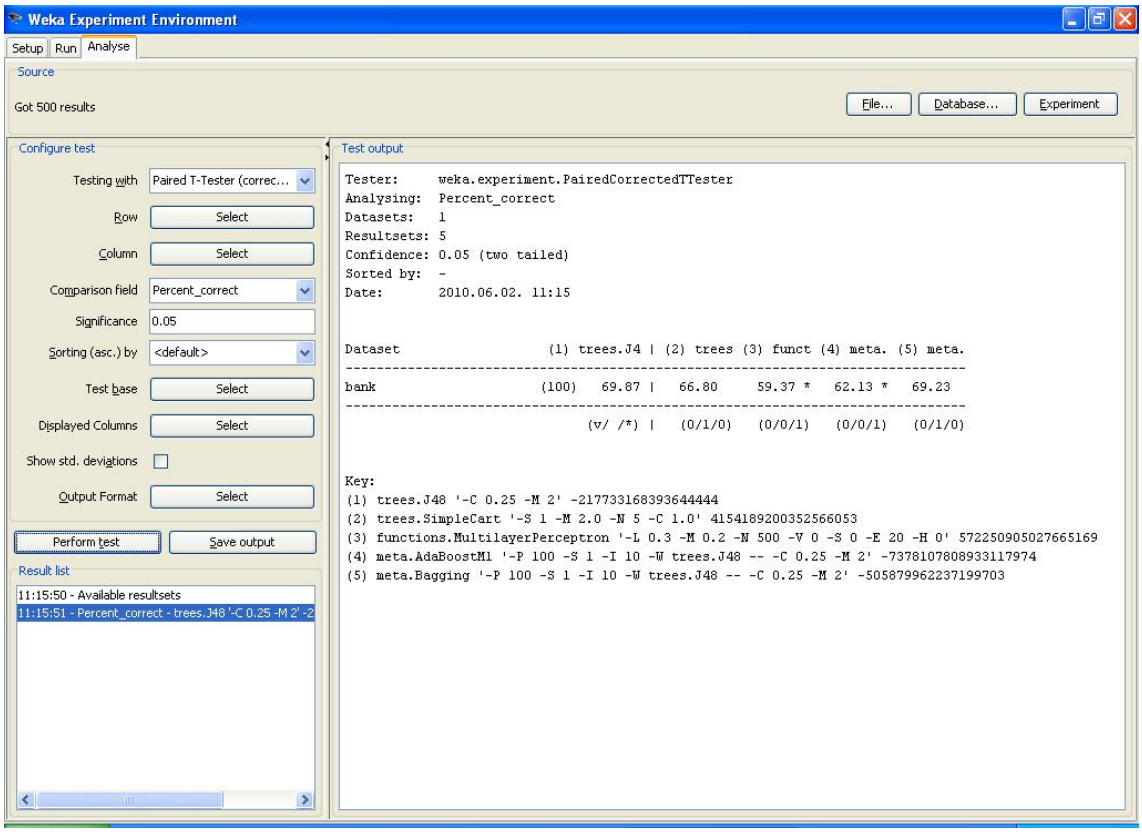


4.2. ábra.



4.3. ábra.

Kiválasztottam a $J48$ (a $C4.5$ algoritmus egy Java nyelven implementált változata), a CART, a Perceptron, a Bagging és az AdaBoost algoritmusokat (a Winnow, a k-NN, és az ID3 módszereket a nekik nem megfelelő attribútum típusok miatt nem választottam). Amennyiben a student t-próbát elvégezve a kapott érték mellett $*$ szerepel, akkor a student próba alapján az osztályozó szignifikánsan rosszabb, mint a legelső, ha pedig v, akkor szignifikánsan jobb az elsőnél. A kapott eredmény:

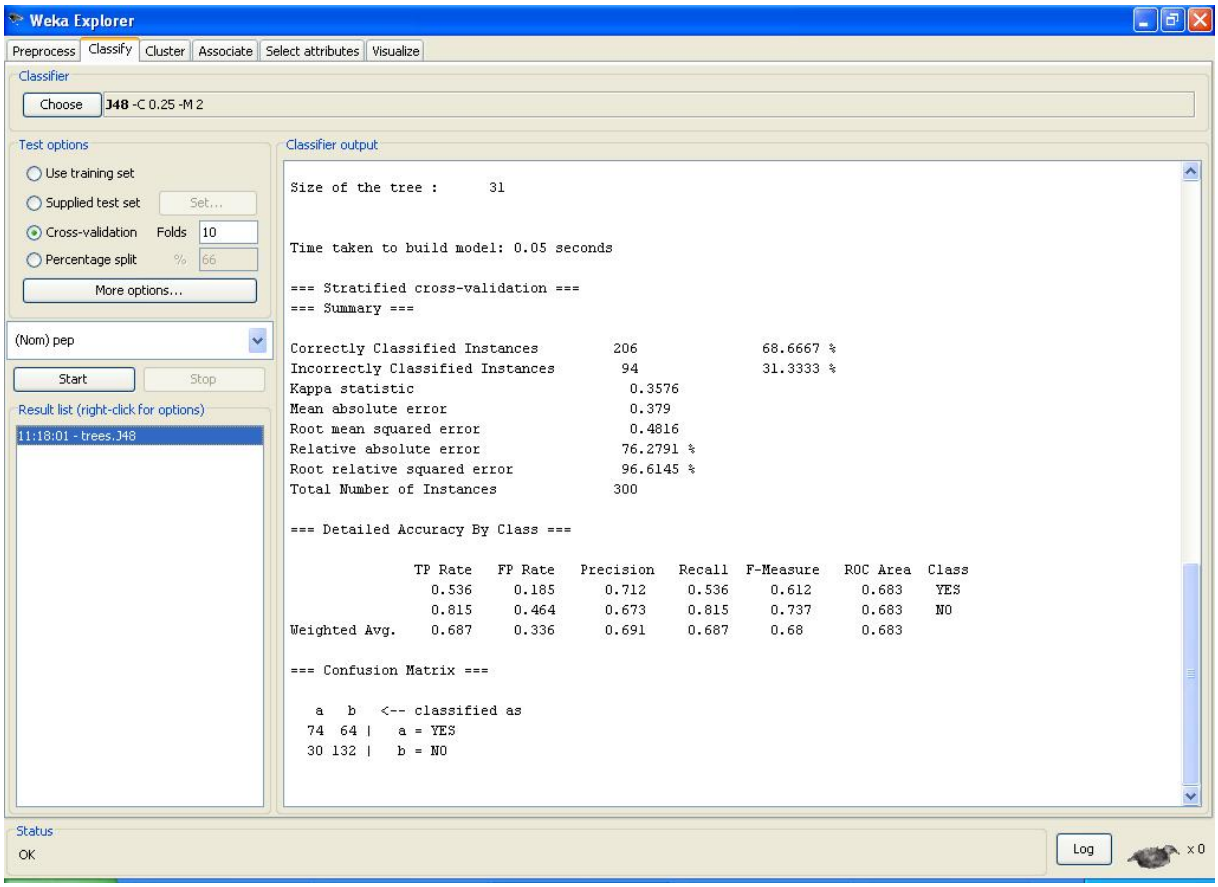


4.4. ábra.

Azt kaptuk tehát, hogy a $J48$, a Cart, és a Bagging algoritmusok szignifikánsan nem különböznek, a többi pedig rosszabb náluk.

Próbáljuk ki az egyik legjobbként kapott $J48$ algoritmust az adathalmazunkon! Az eredmény a 4.5-os ábrán látható (az adatok felosztásához a 10-szeres keresztvalidációt használtam).

Azt kaptuk, hogy az osztályozó módszerünk pontossága mindössze 68.6667%, a kappa statisztika értéke pedig 0.3576.



4.5. ábra.

Az eredményt meg is jeleníthetjük (4.6-os ábra).

A döntési fa alapján (ahogy azt a korábbiakban megsejtettük) a gyerekekre vonatkozó attribútum rendelkezik a legtöbb információtartalommal, hiszen ez a változó került a gyökérbe. Amennyiben az ügyfélnek van gyermeke, akkor a következő lépésben (a 2. szinten) a fizetésének nagyságát vizsgáljuk, ellenkező esetben az számít, hogy házas-e. Később az ábra szerint következnek a további attribútumok.

A jobb eredmény reményében kezdjük meg az adatok előfeldolgozását!

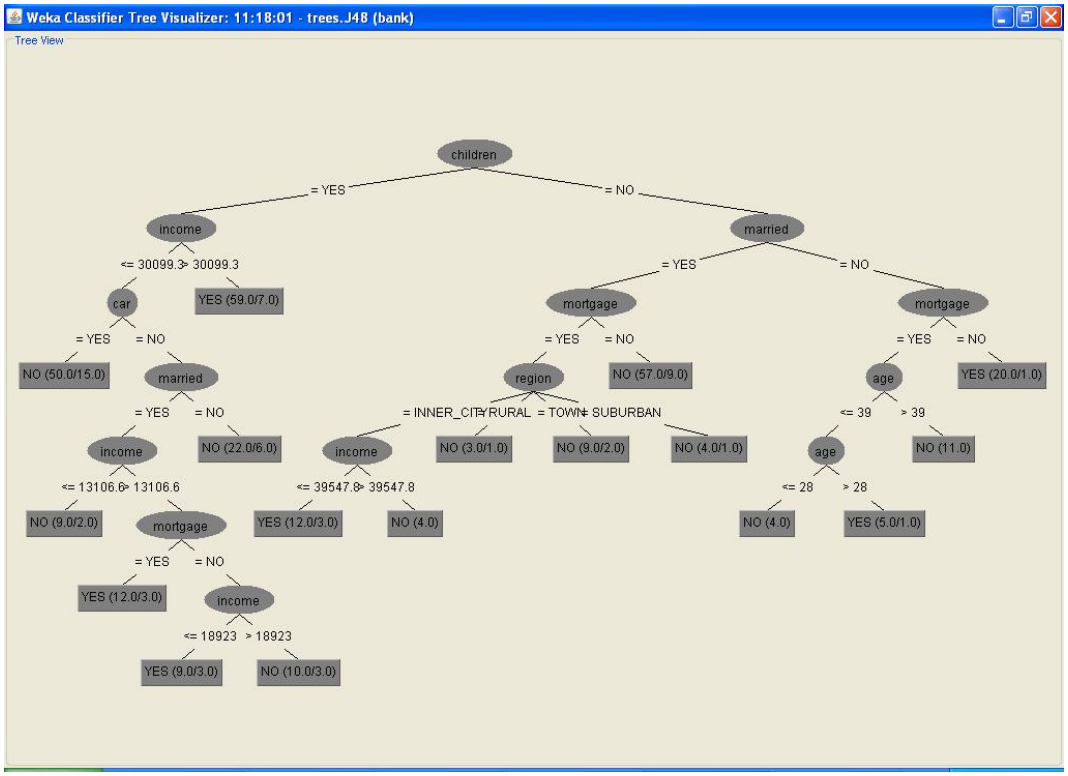
- *Hiányzó értékek:* ezzel most nem kell foglalkoznunk, hiszen az adathalmazunk nem tartalmaz ilyeneket. (Abban az esetben, ha ez a probléma jelentkezne

a WEKA kategorikus attribútumoknál a leggyakrabban előforduló értékkel, szám típusúaknál átlaggal helyettesít).

- *Zajos adatok:* Felderítjük a különc pontokat és az extrém értékeket.
  Jelölje $Q1$, $Q3$ a 25% és 75%-hoz tartozó kavantiliseket. IQR=$Q3$-$Q1$, OF az *Outlier Factor* és EVF az *Extreme Value Factor* (ezeket tetszés szerint változtathatjuk, mi most az alapértelmezett 3 és 6 értékekkel dolgoztunk). *Extrémnek* nevezünk egy értéket ha az nagyobb mint Q3+EVF*IQR, vagy kisebb, mint Q1-EVF*IQR.
  Különc pontok azok az értékek, melyek nem extrémek és nem esnek a

$$[Q1 - OF * IQR, Q3 + OF * IQR]$$

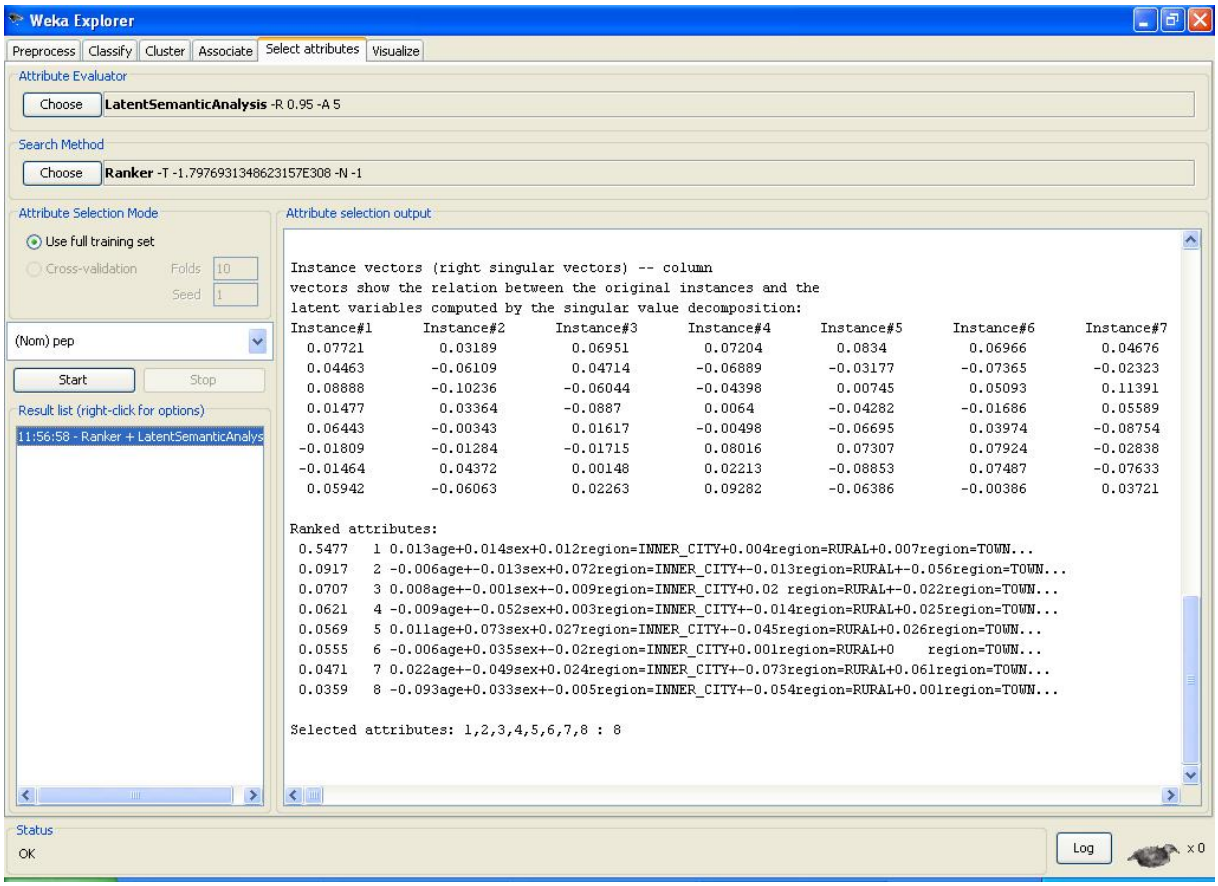
  intervallumba sem. Új attribútumot hozhatunk létre, melynek (A-medián)/IQR lesz az értéke. (A mostani adathalmazunkon végzett eljárások pontosságát azonban ez nem javítja, hiszen csak normalizálás előtt minősíti a WEKA a kor és a fizetés attribútumokat extrém és különc értékeknek).



4.6. ábra.

- *Mintavételezés*: 150%-os visszatevéses véletlen mintavételezés esetén jelentős javulás érhető el.

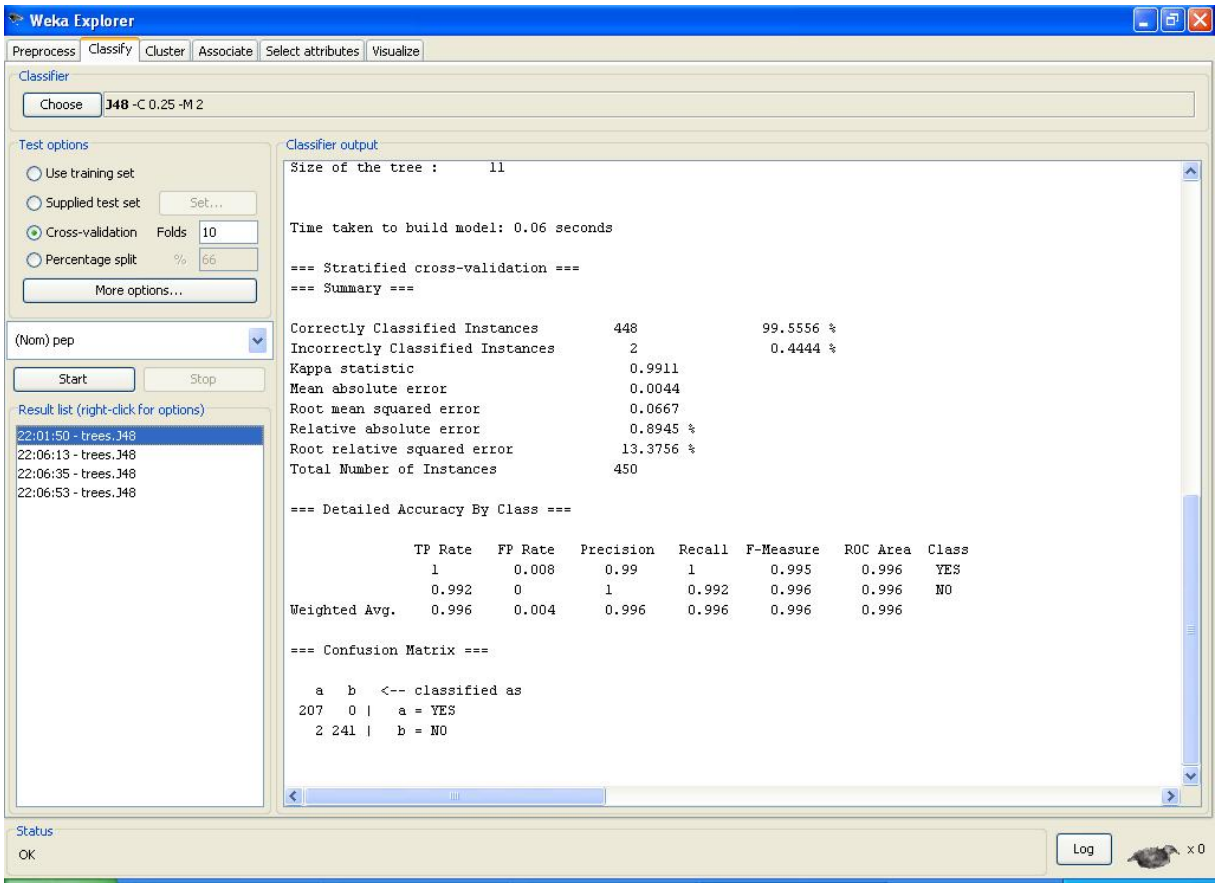
- *Dimenziócsökkentés:* használjuk a szinguláris felbontást (4.7-es ábra). A WEKA az SVD elvégzése előtt normalizálni fogja az attribútumokat (ez javítja majd a pontosságot, hiszen enélkül például a fizetés attribútum elnyomná a kor attribútumot).



4.7. ábra.

Nézzük meg az átalakítások után jobb eredményeket kapunk-e!
A szignifikánsan legjobb próbák egyike még mindig a $J48$ algoritmus, ezt használjuk a következőkben is.

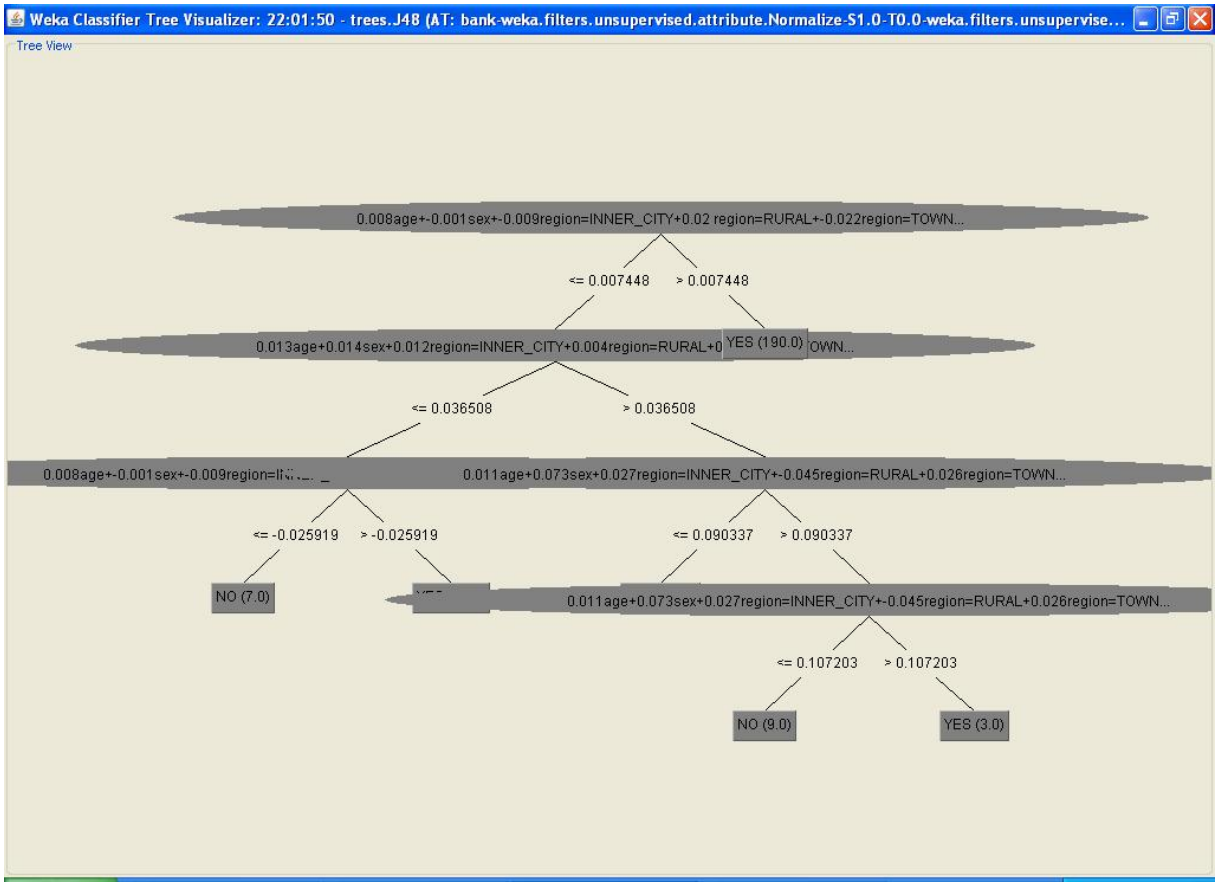
Az elmentett adathalmazunkon hajtsunk végre mindent ugyanúgy, mint ahogy a nyers halmazunkon tettük. A kapott eredményeket 4.8-as ábrán láthatjuk.



4.8. ábra.

Azt látjuk, hogy a 450 adatból összesen 2 olyan volt, amit nem megfelelően osztályoztunk. Vagyis az osztályozónk pontossága ezen az adathalmazon 99.5556%, a kappa statisztika értéke pedig 0.9911, ami már igen közel van a tökéleteshez. (Az elméleti részben tárgyalt további mutatók – mint például a TPR, az FPR, az F-érték, vagy a felidézés – szintén leolvashatóak az ábráról.)

Az eredményhez tartozó döntési fát jelenítjük meg a 4.9-es ábrán. Erről leolvashatóak az adott csomópontoknál végrehajtott tesztek. Azt látjuk, hogy (a szinguláris felbontás alkalmazása miatt) az eredeti attribútumokból új változók lettek, ezek az egyes csomópontokhoz tartozó ellenőrző attribútumok.



4.9. ábra.

Tehát ha egy új hitelkérelem beérkezik, nincs más dolgunk, mint behelyettesíteni a megfelelő attribútumainak értékeit az adott csomópontokban, és az egyes tesztek kimeneteleinek függvényében haladni a fában, míg el nem érkezünk egy levélhez. Amennyiben az itt található osztálycímke az „igen”, akkor a modellünk azt mondja, fogadjuk el a kérelmet, ellenkező esetben visszautasítást javasol (természetesen a végső döntés meghozatala előtt a javaslat felülbírálható, lehetnek például rendkívüli körülmények vagy egyéb módosító tényezők, amit esetleg érdemes figyelembe venni).

5. fejezet

Összegzés

Az akár többezer soron és oszlopon keresztül felsorolt adatok első ránézésre igen ijesztőnek tűnhetnek, különösebb összefüggéseket, szabályszerűségeket „szabad szemmel" aligha olvashatunk ki belőlük. Ám ha közelebbről szemügyre vesszük, megfelelően feldolgozzuk, átalakítjuk ezeket az adatokat, majd adatbányászati algoritmus(ok) bemeneteiként használjuk ezeket, számos érdekes és értékes információt nyerhetünk belőlük. A szakdolgozat célja kezdetben e probléma elméleti bemutatása, mely során először az előfeldolgozás legfontosabb lépéseit próbáltam bemutatni, majd az adatbányászat egy konkrét területével, a klasszifikációval foglalkoztam. Végül az előzőekben ismertetett lépéseket és eljárásokat próbáltam végigvezetni egy konkrét feladat feldolgozásával, melyből az is kiderült, hogy – bár viszonylag kis méretű adathalmazt használtam – néhány egyszerű átalakítással igen pontos és informatív eredményekhez juthatunk.

Irodalomjegyzék

[1] Pieter Adriaans–Dolf Zantinge: *Adatbányászat*, Budapest, Panem (2002)

[2] Dr. Abonyi János (szerk.): *Adatbányászat.A hatékonyság eszköze*, Budapest, ComputerBooks (2006)

[3] Jiawei Han–Micheline Kamber: *Adatbányászat.Koncepciók és technikák*, Budapest, Panem (2004)

[4] Dr. Bodon Ferenc: *Adatbányászati algoritmusok*, tanulmány (2009)

[5] Molnár Csaba–Matolcsi Zoltán: *Adatbányászati technológiák* (2006)

[6] $http://www.pml.co.jp/ghl/2D-fig/voronoi1.jpeg$

[7] $http://maya.cs.depaul.edu/classes/ect584/WEKA/classify.html$

[8] Lukács András: *Adatbányászat – órai jegyzet*