Dombi József

Mesterséges intelligencia II. jegyzet

2007. június 27.

Köszönetnyilvánítás

Köszönetet szeretnék mondani Jász Juditnak a jegyzet első változatának elkészítésére, Dezső Editnek, Diószegi Editnek, Megyeri Gabriellának, Ormándi Róbertnek, Busa-Fekete Róbertnek és Danyi Kornéliának, a jegyzet kibővítésére, valamint Horváth Virágnak, a jegyzet végső formájának elérésére tett erőfeszítéseikért. Szeretnék köszönetet mondani továbbá Gera Zsoltnak és Cséfán Györgynek, a jegyzetben lévő hibák felkutatásában, kijavításában nyújtott segítségükért.

Tartalomjegyzék

Bevezetés

A mesterséges intelligenciáról általában

A bennünket körülvevő világ problémáit az alábbiak alapján sorolhatjuk kategóriákba:

- Az ember által konstruált problémák:
  az emberi agyhoz kapcsolódnak → **mesterséges intelligencia**
  pl. betűfelismerés, sakk feladványok megoldása, közlekedési lámparendszerek intelligens működtetése, nyelvtanokhoz és nyelvekhez kapcsolódó kérdések, logikai programok, stb.

- Természetes problémák:
  a természetben, állatvilágban ugyanúgy megtalálhatók → **természetes intelligencia**
  pl. annak eldöntése, hogy az adott arc férfi vagy női, látás, járás, sportoló döntéseinek folyamata, stb.

A mesterséges intelligencia alapvetően a természetes intelligenciai problémákra, azok megoldására épül. Sokáig tartotta magát az a szemlélet, miszerint maga az *intelligencia* egy mentális képesség, mellyel kizárólag az ember rendelkezik. A kérdés, hogy helyes-e ez a meghatározás? Hasonlítsuk össze például egy szúnyog repülési képességeit egy helikopter működésével! A szúnyog manőverezési képességei jobbak, energiafelhasználása nagyságrendekkel jobb, rendelkezik reprodukciós képeségekkel, stb. Az eltérés rendkívül jelentős, illetve felveti a gondolatot, hogy pontosan mi is az *intelligencia*.

A mesterséges intelligencia fejlődése

A mesterséges intelligenciához tartozó témakörök az idők folyamán mindig változtak. Kezdetben minden olyan dolog, ami embert tudott helyettesíteni (pl. szövőgép), intelligensnek számított.

Korábban, az 1960-70-es években az olyan témakörök, mint a képfeldolgozás, a hangfelismerés, programozási nyelvek vagy az automaták a mesterséges intelligencia tárgykörébe tartoztak. Ezen témakörök azonban idővel különálló tudományágakká fejlődtek, és kiváltak a mesterséges intelligenciából. A mesterséges intelligencia tárgya tehát permanensen változott az idők folyamán, ami fejlődését mindig fenntartotta. Ugyanis ha létrejött egy-egy új tudományterület, először bekerült a mesterséges intelligencia tárgykörébe, és amint továbbfejlődött, megerősödött, úgy kivált és független tudományágként elszakadt a mesterséges intelligenciától. Ez a folyamat természetesen állandó megújulást hozott ennek a tudománynak.

Az 1980-as évek során azonban elkezdtek kialakulni az egyetemeken a mesterséges intelligenciával foglalkozó tanszékek. Ez a folyamat maga után vonta, hogy a tudósok pontosan meghatározzák, mely tudományágak is tartoznak a mesterséges intelligenciához. Ennek következtében nem fogadtak új területeket, megmaradtak alapvetően a logikához kapcsolódó témáknál. Ez visszavetette az ezidáig töretlen fejlődést. A mesterséges intelligencia így elvesztette a dinamizmusát a rögzítettség miatt (pl. nehezen fogadta be a neurális hálók vagy a fuzzy rendszerek témakörét is).

Napjainkra némileg változott ez a felfogás, és egyre inkább belátják, hogy a fejlődés érdekében nem lehet lezárni a mesterséges intelligenciát, mindig nyitni kell az új irányzatok felé. Azonban alapvetően még most is nehezen fogadnak be új diszciplínákat.

Az utóbbi időben a mesterséges intelligencia egyre inkább az elővilágra fókuszál. Az elővilágot megfigyelve ugyanis számos pozitív megoldással találkozhatunk különböző problémákra. Ilyen például egy adott légyfaj. Az egyedek 5 nap alatt válnak fogamzóképessé, és mindössze 30 napig élnek (évente 12 generáció). Minden 48-adik generáció nyomorék, mivel egy öröklődő genetikai információ miatt ez a 4 évente megjelenő generáció szárny nélkül születik. Valószínűleg sokunknak reflexszerűen ez genetikai hibának tűnik, de egészen más megvilágításba kerül, ha figyelembe vesszük a környezetet is. A legyek egy olyan szigeten élnek, ahol 100 évente vihar pusztít, ami elsodorja a repülő legyeket. Kérdéses, hogy a „génhiba" kijavítása hosszú távon előnyös lenne-e a legyeknek. Az élőlények természethez és egymáshoz való alkalmazkodása a gépektől távol álló tulajdonság. Ezekre a tényekre felfigyelve a mesterséges intelligencia is mindinkább próbál ezekből a „megoldásokból" ötleteket ellesni illetve felhasználni a kutatásban.

1. fejezet

Alakfelismerés és azonosítás

A képfeldolgozás illetve az alakfelismerés témakörei gyakran egybemosódnak. A képfeldolgozáshoz leginkább az objektumok, struktúrák lokalizálása, kontúrok detektátálása, alakzatok vékonyítása, stb. feladatkörök tartoznak. Az alakfelismerés nem más, mint objektumok osztályokba való besorolása. Egy klasszikus példával élve, ha az input az alakfelismerő számára egy kép, melyen írott betű található, el kell dönteni, hogy melyik „betűosztályba" tartozik. Az osztálybasorolást azonban nem csak alakfelismerésnél használjuk, hanem azonosításnál (identification) is.

1.1. Azonosítás

Az azonosítást elsősorban biztonsági rendszereknél alkalmazzák. Természetesen minden módszer megkerülhető, kijátszható, azonban több módszer együttes alkalmazása már jelentősen megnehezítheti a hamisítást. Az azonosítás folyamata kötődhet vizualitáshoz, illetve hangokhoz is. Vizualitáshoz köthető azonosítási folyamatok például az ujjlenyomat levétele, az aláírás, az írisz- és szemfenékvizsgálatok, az arcfelismerés. Hanghoz köthető azonosítási folyamat például a beszédhang vagy a lépések ritmusának figyelése, vagy akár követhetjük a jelszó begépelésének ritmusát is.

1.1.1. Vizualizáció

A vizualizáció arra törekszik, hogy a bejövő adatokat képpé alakítsa, és ezáltal megkönnyítse az információ felhasználását. Adott esetben a vizualitás problémák megoldására is szogálhat. Ha egy struktúra bonyolult, érdemes vizualitássá alakítani segítve ezzel áttekinthetőségét. Ezek a módszerek nagyon hatékonyak lehetnek, mivel az emberi szem párhuzamos idegcsápjainak köszönhetően a vizuális információt nagy sebességgel tudjuk feldolgozni.

Napjainkban a felhalmozódott adatok tárolása és feldolgozása elengedhetetlenné teszi a számítógépek használatát. Az információk sokasága egyre inkább kezelhetetlenné válik az emberi agy számára, a numerikus adatokból nehéz kiszűrni a lényeget, a fennálló összefüggéseket vagy változásokat elemezni. Az emberi szem jó feldolgozóképességének kihasználásával ezeket az adatokat vizuálisan ábrázolva sokkal gyorsabb és

pontosabb következtetéseket vonhatunk le az adott információ halmazról.

Egy bizonyítékul szolgáló példa a vizualitás hatékonyságára: adott egy 50×50-es mátrix, melynek minden eleme egy pozitív egész szám. A feladat a mátrixban egy 20 hosszú elemlánc kijelölése úgy, hogy a lánc értékeinek összege maximális legyen. A feladat nyertese az elemeket a számok alapján kiszínezve gyakorlatilag egy térképet kapott, amin a hegyvonulatot a legmagasabb értékekhez rendelt szín alapján lehet könnyen meghatározni. Így a hegygerincen végighaladó elemlánc adja a maximális értéket (lásd bővebben a Hegymászó algoritmus alfejezetet).

Megfigyelhetjük, hogy a szem illetve a látás milyen kitüntetett szerephez jut az állatok esetében is. A rovarok szeme és a virágszirmok színe például egymáshoz alkalmazkodva úgy alakultak, hogy a rovarok könnyen lássák őket. A virágszirom úgymond „világító kifutópálya" a rovarok számára. A sas szeme komoly területi scan funkcióval rendelkezik. Például repülés közben érzékeli a fű mozgását, ahogyan a szél fújja a fűszálakat. „Kiszámítja", hogy normál esetben hogyan kell a fűnek mozognia, és ha ettől eltérő mozgást érzékel, akkor ott a „zsákmány". Másik funkciója a teleobjektív látás: az áldozatainak ürülékét illetve vizeletét piros pontokként érzékeli.

1.1.2. Akusztika

A biztonsági rendszerek gyakran ezeket a módszereket használják. Előnyük, hogy hang útján kapott információ mellett tudunk másra is figyelni, például vezetés közben rádiót hallgatni, telefonálni, tehát képesek vagyunk másodlagos információ befogadására is. A látásra ez nem igaz, például vezetés közben hiába próbálunk filmet nézni, nem tudunk egyszerrre két vizuális csatornára figyelni. Ezért számos esetben érdemes hanggá konvertálni az információt, például operáció közben a szívritmus kijelzését hangeffekt is követi, így az orvos műtét közben is nyomon tudja követni a beteg állapotát. Más területen is alkalmazható a hanggá való konverziót. A kémiában, ahol az egyes molekulákhoz a mágneses rezgéseikből kapott spektrumot rendelik, mely egyértelműen azonosítja az adott molekulát. Ez a folyamat bár vizualizáció, a rezgésekhez hang is rendelhető és azáltal is felismerhetők a molekulák atomjai.

Néha talán túlzottan is bízunk a vizualizációban, pedig a hangok alkalmazása kézenfekvőbb bizonyos esetekben. Például, ha egy atomerőműben körülbelül 1000 műszer van, melyek közül mondjuk 100-at figyelnek folyamatosan, akkor ezeket figyelhetik akusztikusan is. A 100 műszert 100 zenésznek feleltetjük meg, melyek alap esetben (ha nincs semmilyen hiba) Beethoven 5. szimfóniáját játsszák. Vagyis, ha a műszer jó adatokat szolgáltat, akkor a hozzárendelt zenész jól játszik, különben pedig hamisan. Ezeknek a műszereknek az adatait folyamatosan vizuálisan követni sokkal megterhelőbb lenne számunkra, míg a hamis muzsikát gyorsan kiszűri egy jó hallással rendelkező ember akkor is, ha egyébként a műszerekhez nem is ért.

1.1.3. Azonosítás jellegzetességek alapján

Az azonosítás egyik ágánál sem lényeges maga a tartalom, vagyis, hogy milyen bemeneti információt kapunk, csak az számít, hogy maga az azonosítás a jellegzetességek alapján megtörténhessen, tehát egy

kulcsot tudjunk az inputhoz rendelni (pl. az aláírásnál sem az a lényeges, hogy hogyan hívják az embert). Az adott minta alapján történő azonosítás egyik fajtája az úgynevezett matching (illesztés, fedésbe hozás).

$$\left.\begin{array}{ll} M: & \\ F: & \end{array}\right\} (M-F)^2,$$

ahol M a mintát, F a megtanulandó objektumot jelöli.

A gyakorlatban nem szükséges a teljes egyezés, csupán a tulajdonságok (feature) egyezésének mértékét vizsgáljuk. A feature meghatározása, vagyis annak eldöntése, hogy milyen tulajdonságok alapján vizsgáljuk az egyezést, korántsem egyértelmű. A tulajdonságok lehetnek előre meghatározottak, illeve lehetséges, hogy a program dönti el, mik legyen a vizsgálandó tulajdonságok. Ehhez szükségünk lesz tanulóalgoritmusra, ami egy olyan metaalgoritmus, mely algoritmust állít elő, vagyis megkeresi az adott probléma megoldására használni kívánt módszert.

Az azonosítási folyamat lépései tehát: „feature” (tulajdonság) kiválasztása és „matching” (fedésbe hozás). A folyamatot *identifikációnak* nevezzük.

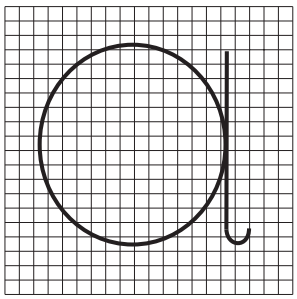
1.2. Alakfelismerés

Az alakfelismerés egyik legalapvetőbb feladata az írott betűk felismerése. Térjünk vissza ehhez a példához, és ezen keresztül határozzuk meg az alakfelismerési problémát! Például fel szeretnénk ismertetni az 1.1. ábrán látható betűsorozatot.

1.1. ábra. Felismerendő alakzatok – írott betűk

Első lépésként digitalizálni kell az ábrákat, mely ebben az esetben az alakzatok „rácsra” való elhelyezését jelenti. A digitalizálásnak több módja is van.

Egyik az 1.1. táblázatban látható 20×20-as rács, ami egy 0-1 reprezentációt szemléltet. Egy másik megközelítés a mátrixban szürkeségi értékeket tárol, illetve akár színinformációval is feltölthetjük. A lényeg mindhárom esetben az alakzat rácson való elhelyezése (lásd 1.2. ábra).

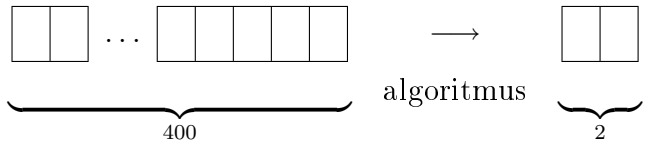


1.2. ábra. *a* betű rácson való elhelyezése

0	0	0	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0	0	0	0
0	0	0	0	0	**1**	**1**	0	0	0	0	0	**1**	**1**	0	**1**	0	0	0	0
0	0	0	0	**1**	**1**	0	0	0	0	0	0	0	**1**	0	**1**	0	0	0	0
0	0	0	**1**	**1**	0	0	0	0	0	0	0	0	**1**	**1**	**1**	0	0	0	0
0	0	**1**	**1**	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0
0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0
0	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0
0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0
1	**1**	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	0	0	0	0	0	**1**
1	**1**	0	0	0	0	0	0	0	0	0	**1**	0	**1**	0	0	0	0	**1**	0
0	**1**	0	0	0	0	0	0	0	0	**1**	**1**	0	**1**	0	0	0	0	**1**	0
0	**1**	**1**	0	0	0	0	0	0	0	**1**	0	0	**1**	0	0	0	**1**	0	0
0	0	**1**	**1**	0	0	0	0	0	**1**	0	0	0	**1**	**1**	0	**1**	0	0	0
0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0	0	**1**	**1**	0	0	0	0

1.1. táblázat. Kis *a* betű 0-1 reprezentációja

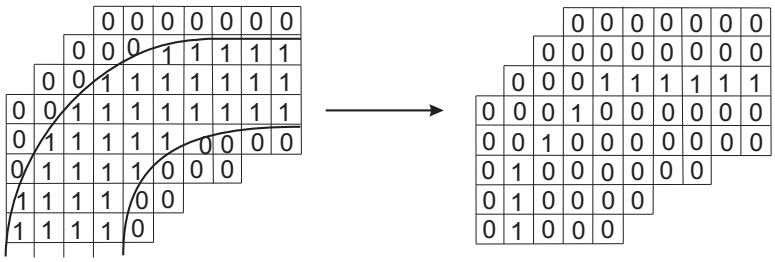
Így az alakzatainkból egy-egy 400 (20×20 - a rácsméret) hosszú $\underline{v}$ vektort készítünk. Azonban maga az információtartalom elférne mindössze 2 biten is, vagyis az információ reprezentálható lenne egy 2 hosszú vektorral is (csupán 4 betűt kell megkülönböztetni). Az alakfelismerési problémát a fentiek alapján tehát a következőképpen határozhatjuk meg: keresnünk kell egy algoritmust, mely sematikusan egy vektor-vektor leképezés:



Megközelíthetjük a problémát úgy is, hogy adott egy $\underline{v}$ vektor, és keresünk egy $F(\underline{v})$ függvényt, melynek 4 visszatérési értéke lehet: *a, o, u* vagy *v*, aszerint, hogy minek ismerte fel az algoritmus a $\underline{v}$ vektort. A felismerés ezáltal a tulajdonságvektor tömörítése információtartalommá. A tömörítés itt 200-szoros.

Természetesen felmerül a kérdés, hogy mekkora rácsméretet érdemes a reprezentáció során választani. Általában nagyobb felbontással szokás dolgozni, mint amekkora a betű meghatározásához kell. Érdemes azonban „arany középutat" találni a túl sűrű és túl ritka rácssorozat között, ugyanis ha túl ritka a rácssorozat, nehezen vagy egyáltalán nem lehet majd felismerni az alakzatot (vagyis az F leképezés nehezen megadható), illetve ha túl sűrű a rács, túl nagy tömörítést kell végrehajtani. Vannak olyan esetek, amikor érdemes a kép különböző részeiben más és más felbontást használni.

Amennyiben a mintánk vonalvastagsága túl nagy volt, úgy vékonyító eljárásokra lehet szükség az adatmennyiség csökkentése érdekében. A vékonyítás egy képfeldolgozásból ismert párhuzamosítható algoritmus, mely a váz meghatározására szolgál (1.3. ábra). A vékonyító eljárások a zajra igen érzékenyek.



1.3. ábra. Alakzat vékonyítása

Az alakzat vastag vonalát az 1-et tartalmazó mezők alkotják, és azokat kell módosítanunk. Ha 0-át tartalmazó mezőhöz érkezünk, annak értékét nem változtatjuk meg. Ha 1-et tartalmazó mezőhöz érkezünk, úgy meg kell vizsgálnunk a mező környezetét. Ha a környezetében csupa 1 van, akkor 1 marad. Amennyiben olyan az 1-es mező, hogy a környzetében 0 is és 1 is található, új értéket az alakzat folytonosságától függően kap. Arra kell ügyelnünk, hogy ne szakadjon meg a 0-ra váltása által az alakzat vonala. Vagyis akkor váltjuk csak 0-ra az ilyen tulajdonságú 1-est, ha az alakzat folytonossága így is megmarad. (Azonban vannak esetek, amikor még így is megváltozik az alakzat szerkezete, például eltűnhet az i betűről a pont.) A vékonyítás művelettel egyfajta szabványosítást (normalizálást) hajtottunk végre a mintán, a vonalvastagságot csökkentve.

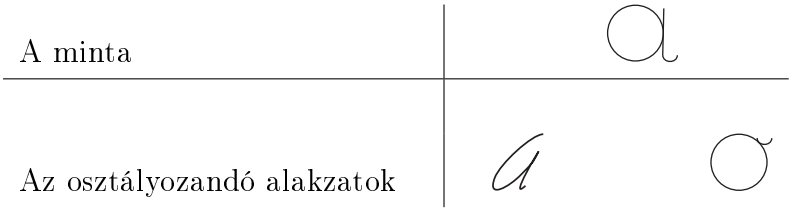
A „matching” tehát az alábbi folyamatokból áll: először megnézzük, hogy ekvivalens-e a mintánk valamely felismerendő alakzattal. Ha nem tudjuk egyérteműen felismerni, úgy megpróbáljuk szabványosítani a mintát, és újra vizsgáljuk az ekvivalenciát.

A kicsinyítést (nagyítást), illetve a vékonyítás technikáját használjuk leggyakrabban az alakzatok normalizálsára. Nem mindig célravezető azonban a normalizálás, így a vékonyítási művelet sem. Gondoljunk csak a japán vagy kínai karakterekre, melyek esetében akár meg is változhat az adott karakter jelentése, amennyiben vékonyítást alkalmazunk rajtuk, mert ezen jelek esetében a kalligráfia a lényeges. Felmerül a kérdés, hogy mikor is alkalmazható vékonyítás?

A vékonyításnak invariánsnak kell lennie az alakzat jelentésére nézve. A latin betűk azonosítása a vastagsággal szemben invariáns, de kínai és japán betűk felismerése a vékonyítással szemben nem invariáns. (A matematikában is találkozunk az invariancia fogalmával, pl. egybevágóság, hasonlóság, adott gráf struktúrái, stb.) Ebben a tekintetben az alakfelismerés során olyan transzfomációkat kell leírni, melyek invariánsak az alakzattal szemben. A felismerés pedig nem más, mint ezen transzformációk megtalálása.

Felmerül a kérdés, hogy pontosan mi teszi az adott mintát (ábrát) betűvé, és hogy milyen transzformációkkal szemben őrzi meg ezt a tulajdonságát? Az alakzatoknak vannak „lényeges” és „lényegtelen” részei, tulajdonságai. Lényeges résznek most azt a részletét nevezzük, ami alapján jól elkülöníthető a többi alakzattól, lényegtelennek pedig azt tekintjük, ami nem specifikusan rá jellemző. Például, ha az a és o betűket tekintjük, és ezeket akarjuk jól megkülönböztetni egymástól, akkor nem érdemes a kör meglétét

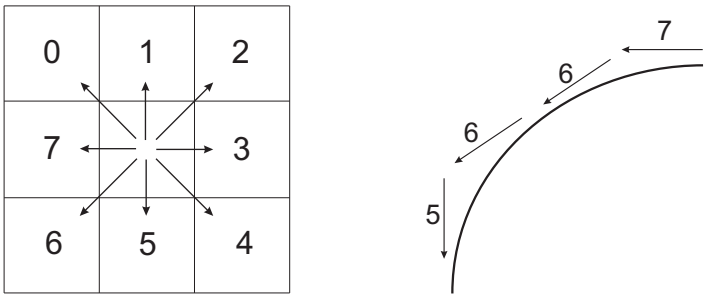
lényegesnek tekintenünk, mert az alábbi hibába eshetünk:



A mintától való eltérés az „o” betűnél kisebb lesz egy ilyen esetben, és így nem helyesen ismerjük fel az alakzatot. Ezért itt például célravezető lett volna a szár hollétét figyelembe vennünk, és azáltal helyes besorolásra juthattunk volna. Ezeket a lényeges és lényegtelen tulajdonságokat azonban közel sem triviális meghatározni, nehéz lenne például egy programmal megkerestetni a lényeges tulajdonságokat.

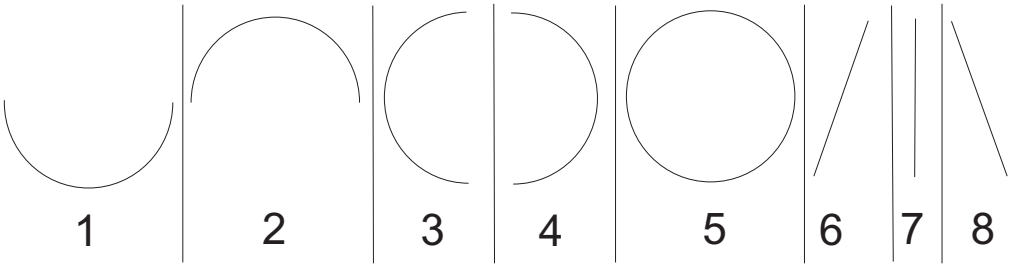
A megfelelő megoldáshoz az a hétköznapi példa vezetett, amikor azt vizsgálták, hogy az ember hogyan is tanulja meg a betűket leírni, és felismerni. A kisiskolás gyermek, amikor elsajátítja az egyes betűk leírásának módszerét, tulajdonképpen egy utasítássorozatot tanul meg. Például, ha az *a* betűt tanulja, akkor elmondják neki, hogy rajzoljon egy kört az óramutató 2-es irányából kiindulva, és rajzoljon mellé egy vonalat fentről lefelé, végül a vonal végére tegyen egy ívet. Ez a megközelítés adta az ötletet arra, hogy a mátrixos reprezentációból irányokat kell kinyerni. Tehát ne csak olyan információt tároljunk az alakzatról, hogy az adott helyen van-e pontja vagy sem, helyette tároljunk el irányokat is, például hogy az adott vonal merre megy.

Ezen alapul a Freeman-kódolás. A Freeman-kód irány szerinti kódolást jelent. Az eredeti vektort kódolja az alapján, hogy adott alakzathoz tartozó ponthoz viszonyítva a következő alakzathoz tartozó pont milyen irányban van, vagyis az adott irányokba történő elmozdulást adja meg. Például egy 3×3-as rács esetében az 1.4. ábra bal oldalán látható irányok lehetségesek, és az adott görbét a 7665 számsorral (vektorral) tudjuk kódolni. Jól látható, hogy a fenti 400 hosszú vektort sokkal tömörebb formában kaptuk meg. Irány szerinti kódolással a rácsméret természetesen lehet nagyobb vagy kisebb is, azt a problémától függően választhatjuk meg. Nagyobb környezet vizsgálatával pontosabb eredményhez jutunk. A Freeman-kódolás és a tömörítés is invariáns az alakzatra nézve.



1.4. ábra. Körív Freeman–kódolása az adott irányok szerint

További tömörítésre ad módot, ha felveszünk primitíveket (standard jelek, alakzatok), és ezekből kiindulva kódoljuk a mintákat. A kódot az adott betűt felépítő primitívek határozzák meg (1.5. ábra). Így az *a* betű kódja a következő lehet: 571.

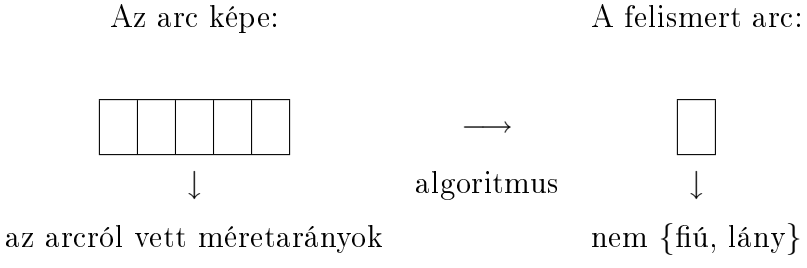


1.5. ábra. Összetett alakzatot felépítő primitív alakzatok

Természetesen a primitíveket kicsinyíthetjük illetve nagyíthatjuk, amennyiben szükséges. A betűk osztályozásánál azt figyeljük, hogy megtaláljuk-e benne a megfelelő primitíveket a megfelelő kapcsolatban, vagyis hogy mennyire egyezik az adott lánccal a primitívhalmaz (a hasonlóság mértékét kell figyelembe venni). Ez a transzformáció is invariáns az alakzatokra nézve.

Azonban egy betűnél több lánc is kapcsolódhat egymáshoz, például a K betű esetén. Ezért fontos az a tulajdonság, hogy ezek a láncok strukturálisan hogyan kapcsolódnak egymáshoz. Ennek leírására szolgálnak a gráfnyelvtanok. Minden betűhöz nyelvet rendelünk, mely leírja annak struktúráját, a nyelvekhez pedig automatát, ami azt felismeri. A leírt folyamatot *szintaktikus alakfelismerésnek* nevezzük. A szintaktikus alakfelismerés esetén arra is kell ügyelnünk, hogy valamilyen módon az automata konstrukciója során figyelembe vegyük a hasonlósági mértéket is, vagyis nem csak a teljesen egyező mintákat kell tudnunk felismerni, hanem a „tűréshatáron” belülre eső̈ket is.

Az alakfelismerés egy másik alapvető problémája az arcfelismerés. Vizsgáljuk csak azt az esetet, hogy az adott arc női vagy férfi. A megoldást egy hétköznapi életből vett példa révén lehet megadni. Tekintsük ugyanis egy emberi arc lerajzolásának folyamatát. A rajzoló megfigyeli az arc arányait (például a szem nagyságát, elhelyezkedését az orrhoz képest, stb.), ezeket az információkat rögzíti és igyekszik visszaadni a vásznon. (A megközelítést leginkább a karikatúrakészítés szemlélteti, mivel a karikatúra az arc eltúlzott arányait rögzíti.) Hasonlóan kell eljárnunk az algoritmus készítésénél: elemeznünk kell az arcot, és mérőszámokkal jellemezni azt. Megmérjük például az orr hosszát, a száj szélességét, az arc magasságát, stb. Majd képezzük a mért számok arányait. A folyamat az adott mintából való információ kinyerése, ami egy tömörítési mód, és a tömörített kód itt is invariáns a mintaosztályra nézve. Egy vektort képzünk, melynek hossza attól függ, hogy hány arányt képeztünk. A következő sematikus ábra az arcfelismerési problémát reprezentálja:



Kérdés, hogy hogyan határozunk meg ebből a vektorból 1 bitnyi információt, nevezetesen azt, hogy férfi vagy női arcról van-e szó? (Pontosabban: hogyan határozzuk meg a fenti ábra leképezését?) A problémát

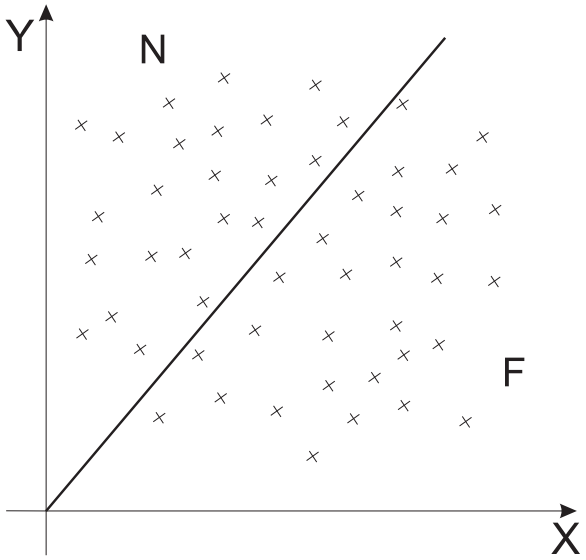
logikai formulakiértékeléssel lehet megoldani úgy, hogy beállítunk bizonyos küszöbszámokat az adott mérési adatokra, és eszerint hozunk döntéseket. Például a fentiek alapján kapott vektort $\underline{v}$-vel jelöljük, és $\underline{v} = \{a_1, a_2, \ldots, a_n\}$, akkor az alábbi feltételeket fogalmazhatjuk meg:

$$if \quad (a_1 < 0,7) \vee (a_1 > 0,8) \wedge (a_2 < 0,06) \wedge \cdots \wedge (a_n > 0,5) \quad then \quad WOMAN$$

Nyilvánvaló, hogy ez az algoritmus csak úgy működhet, ha meghatározzuk, hogy milyen tulajdonságokat mérünk, illetve megadjuk a megfelelő küszöbszámokat is.

A fenti probléma az emberi agy számára egyszerű, hiszen a legtöbb esetben a másodperc töredéke alatt eldöntjük, hogy férfi vagy női arcot látunk magunk előtt. Azonban nem lehet minden döntést szavakkal jól leírni. Azt mondhatjuk, hogy a valóság szavakkal való reprezentációja nem más, mint a minket körülvevő világ digitalizálása, leképezése általunk ismert fogalmakra. Ebből is látszik, hogy a természetes intelligenciai feladatok rendkívül összetettek, és hogy az arcfelismerést is nehéz algoritmizálni.

A legnagyobb nehézséget természetesen az jelenti, hogy milyen tulajdonságokat is mérjünk, vagyis hogy pontosan milyen mérőszámok teszik az adott arcot „nőivé" illetve „férfivá". Jelölje **N** a női arcra vonatkozó tulajdonságokat, és **F** a férfi arcra jellemző értékeket! Arra vagyunk kíváncsiak, hogy tudunk-e konstruálni egy szeparáló felületet, mely elkülöníti a két halmazt egymástól (1.6. ábra).



1.6. ábra. Szeparáló felület a női és férfi arcokat reprezentáló pontok elválasztására

Mivel a tulajdonságok meghatározása sem könnyű feladat, csakúgy, mint olyan algoritmus konstruálása, mely olyan határvonalat (szeparáló hipersíkot, a fenti esetben speciálisan egyenest) keres, amely pontosan elválasztáná **N**-et **F**-től, ezért a nehéz feladatok közé tartozik a természetes intelligencia hatékony modellezése.

Van egy másik megoldási lehetőség is az arcfelismerésre, méghozzá „erőből" megoldani a feladatot. Ekkor veszünk egy nagy számú mintát, melyek besorolását már tudjuk (tudástár, adatbázis), és amikor egy új mintát kapunk, hasonlóság alapján döntünk a besorolásáról. Ez a megoldási mód rendkívül számításigényes, és nagy tárolókapacitást igényel. Egy ilyen program kihasználja a számítógép azon előnyét az emberi aggyal szemben, hogy a gép tárolóorientált, az emberi agy pedig algoritmusorientált.

Most térjünk át egy másik lényeges problémára, a képek vizuális tartalma szerinti rendezésére. Digitális fényképezőgépek által egyszerűen és rövid idő alatt tudunk felhalmozni sok nagy felbontású képet. A több ezer, vagy akár több tízezer képből álló képarchívumban igen nehézkes a keresés. Erre nyújtana megoldást a tartalom szerinti keresés algoritmusa. Itt természetesen nem arról van szó, hogy valamiféle szöveges információt a képekhez illesztve, ezen „digitalizált" információ alapján történik a lekérdezés, hanem a mesterséges intelligencia módszereinek segítségével a képek tartalma (vizuális információ) alapján szeretnénk keresni. Alapvetően az információ mennyisége nem más, mint a képen található pixelek és azok színe. Egyértelmű tehát, hogy egyfajta tömörítésre lesz szükség ahhoz, hogy lekérdezéseket hajthassunk végre.

A megoldást ezúttal a festészet módszereinél találjuk, ugyanis mikor a festő megfesti a valóság egy részletét, tulajdonképpen tömörít. Sokkal kevesebb színt használ, mint amennyi valójában jelen van, illetve leegyszerűsítve adja vissza a dolgokat, ecsetvonások sorozatával reprezentálva a világot. A festő tehát „átlagos" színeket fest, illetve foltokat készít, melyek mérete függ az ecset vastagságától, illetve vonalakat húz különböző irányokban. Vagyis adott egy paletta korlátozott mennyiségű színnel, és adottak különböző méretű ecsetek, melyek a paletta színeivel vonalakat tudnak húzni. Tulajdonképpen jelen esetben az algoritmus sem más: felveszünk egy korlátos számú színt tartalmazó palettát, mely a valóságban előforduló színektől sokkal kevesebb színt tartalmaz, illetve definiálunk „ecseteket" különböző mérettel, és eltároljuk minden képről, hogy „hol van rajta vonal", milyen ecsettel kell azt meghúzni, és hogy milyen színt használunk. Ezáltal jóval kevesebb információt kell eltárolunk, és ezek alapján lehetségessé válik a keresés. Például, ha túlnyomórészt zöld szín van a képen, valószínű, hogy tájképről van szó, illetve ha függőleges fehér ecsetvonások vannak többségben, esetleg vitorláskikötőt ábrázolhat a kép. Természetesen az invariancia itt is lényeges, ügyelnünk kell rá, hogy a kapott adathalmaz invariáns legyen a mintával szemben.

A képek ilyenfajta tömörítését vitte véghez az impresszionizmus. Mint maga a szó is mutatja, az az impresszión, benyomáson alapszik, vagyis a művészek azt a benyomást ábrázolták, amit rájuk a megfestendő dolog tett. Erősen elnagyolt képek jellemzik ezt az irányzatot, vagyis igen erősen tömörítették az ábrázolni kívánt dolgokat. Csak a „lényeget" festették meg, egyfajta vázlatot adtak. A festő mindent a saját aspektusából kiindulva ábrázolt. (Például Rembrandt: Önarckép - nem magát festette meg a festő, hanem azt, ahogy önmagát látta.) A kubizmus szintén eléggé leegyszerűsítve festette meg a világot. Az irányzat képviselői tisztán geometriai formákkal ábrázolták a dolgokat, ez is egyfajta tömörítés. Tehát a festészet is egy matematikai algoritmust (a vizuális információ tömörítésének algoritmusát) nyújt ezen irányzatok által. Jól látszik, hogy a humán tudományok eredményei milyen mély befolyással vannak más tudományágakra, illetve hogy hogyan is lehet felhasználni ezeket az eredményeket. A fent leírt módszer alapján *statisztikus alakfelismerést* is lehet végezni, hiszen a simított képről statisztikát készítve azonosíthatjuk azt. Számos területen alkalmazhatjuk a statisztikus alakfelismerést, így például az orvostudományban az agyhullámok görbéinek felismertetésére is. Az, hogy az alakfelismerés melyik ágát, a szintaktikus vagy a statisztikus módszert alkalmazzuk, a felismerendő jelektől függ.

2. fejezet

Fuzzy logika

1965-ben az iráni származású, Berkeley Egyetemen oktató matematikus, Lotfi A. Zadeh egy reptéren várakozva a hangosbemondón a következő mondatot hallotta: „A repülőgép előreláthatólag kissé késik." Eltűnődött, hogy mit is jelent ez pontosan. A „kissé" nem jól meghatározott, matematikailag addig megfoghatatlannak tűnt. Tehát vannak szavak, melyeket a matematika nem tud igazán jól kezelni. Ez a kiindulópontja a fuzzy elméletnek. A „fuzzy" angol szó jelentése: elmosódott, határozatlan. A fuzzy célja, hogy matematikai leírást adjon az olyan fogalmakra, melyek nem pontosan meghatározottak (ezek általában mérésekhez kapcsolódnak). Ilyenek például:

fiatal	-	öreg
gyakran	-	ritkán
néha	-	gyakran
körülbelül		
könnyű	-	nehéz (súlymérés)
rövid	-	hosszú (hosszmérés)

Véges sok szót használunk, ellentétben a világ dolgainak végtelenségével. A szavak digitalizálják a valóságot, így tartalmazniuk kell pontatlanságot. Alig van olyan természetes nyelvű szó, aminek meg lehetne adni az egzakt definícióját. Szavaink többsége főnév, ebből is adódik a pontatlanság.

A példákból jól látható, hogy az emberi gondolkodásmód sokkal jobban reprezentálható egy olyan rendszerrel, mely figyelembe veszi ezt a bizonytalanságot, melynek nincsenek éles korlátai. A fuzzy logika alapvetően azért született meg, hogy pontatlan vagy bizonytalan információkat, ismereteket, vagy rugalmasan kezelendő határfeltételeket is matematikai formába lehessen önteni, azokat kvantitatív módon kezelni lehessen.

2.1. Fuzzy pragmatizmus

A fuzzy alapgondolata, hogy egy tulajdonság megléte és nem megléte közötti átmenet folytonos legyen. A koncepciót ki lehet terjeszteni fogalmakat reprezentáló szavakra is, mint pl. asztal, szék, toll, stb. A tárgyak definiálását két fő irányzatra különíthetjük el: az egyik a definíciós, a másik pedig a pragmatikus.

A definíciós irányzat leginkább a német (porosz) nyelvterületeken terjedt el. Itt a tárgyak, fogalmak definíciójából indulunk ki: pl. a szék egy olyan tárgy, melynek támlája, lábai, és ülőrésze van, lehetnek karfái is, ekkor karosszékről beszélünk. A pragmatikus (funkcionalitás-elvű) megközelítés pedig abból indul ki, hogy az adott tárgy mire való, pl. szék az, amire leülhetünk (→ egyfajta megszemélyesítése a dolgoknak). A pragmatizmus főként az angolszász országokban elterjedt. Jó példa erre az angol KRESZ könyv, ami kevesebb, mint 20 oldal. A tolatás szabálya például: „Ha nem lehet rendesen hátra látni, akkor kérj meg valakit, hogy irányítson!"

Nagyon sok fuzzy fogalommal találkozhatunk mindennapjaink során: például az időjárásjelentésben: „Várhatóan meleg időnk lesz, kevés csapadékkal, délután a szél megélénkül". Ha azt mondjuk, hogy 23 °C lesz a hőmérséklet, akkor ez mást jelent számunkra télen, és mást nyáron. Nyáron ezt nem mondhatnánk melegnek, télen viszont egyenesen kánikula. Az időjárásjelentések szabványok alapján készülnek (amennyiben az adott mérési adat egy adott intervallumba esik, úgy előre meghatározott, hogy milyen szót kell használni - pl. hőmérséklet - 20 °C felett - meleg), és nem veszik figyelembe az emberek hőérzetének változásait.

A jogalkotásban is rengeteg gondot okoz a fogalmak pontos definíciójára való törekvés, azaz a fogalmak definíciós megközelítése. Például amikor a MÁV definiálja, hogy pontosan mit vihet magával egy-egy utas poggyászként a következő paramétereket szabják meg: maximum 1,5 m hosszú lehet, maximum 25 kg súlyú lehet, illetve maximum 4 darabot lehet az utasnak magával vinnie. Ekkor felmerült a probléma: ha egy rúdugró utazik vonaton, és a rúdja túllépi a hosszúsági korlátot, akkor nem viheti a rudat magával. Ugyanakkor egy rúdugró számára a rúd a személyes poggyászát képezi. Az építőipari szakember szintén nem viheti magával a betongerendát vagy a vascsöveket. Ebből a példából jól látszik, hogy sok esetben az egzakt definíciók nem jól alkalmazhatók, mert folyamatosan bővíteni kell őket a kivételek kezelése miatt.

A pragmatizmus hasznosságára jó példa, mikor favágók leülnek a levágott fatönkökre ebédelni, illetve kiválasztanak egyet asztalnak, nem feltétlen érezzük helyesnek, hogy a fatönköt asztalként vagy székként is definiálják (szék: amire le lehet ülni, asztal: amin ehetünk). A pragmatikus megközelítés megoldhatja a rúdugrók gondját, amennyiben olyan szabályt alkotunk, miszerint mindenki a személyes poggyászát magával viheti. Ekkor a fenti probéma megoldott. Az angolszász jogrendszer pragmatikus. Ez a megközelítés sokkal rövidebb, tömörebb leírását adja a dolgoknak. (A programozási nyelvek nagy részének értelmezője funkcionalitás-elvű, a szubrutinok funkciójuk szerint kapnak besorolást és így értelmezzük őket.)

A fuzzy elméletnek kezdetben sok ellenlábasa volt, de fokozatosan elismerték létjogosultságát, és mára már teljesen elfogadott tudományterületté vált.

2.2. Fuzzy halmazok

A következőkben a fuzzy elmélet első megjelent cikkének elemeit ismertetjük. A fuzzy halmaz a klasszikus halmaz („éles" halmaz, matematikai halmaz) „általánosítása". Minden klasszikus X halmaz reprezentálható egy karakterisztikus függvénnyel, melynek jele $\chi_X(x)$. Egy éles halmaznak egy objektum vagy eleme, vagy

nem. Ezt a relációt jellemezhetjük a *karakterisztikus függvénnyel*:

$$\chi_X(x) = \begin{cases} 1, & ha \quad x \in X \\ 0, & ha \quad x \notin X \end{cases}$$

A fuzzy elmélet alapja a karakterisztikus függvénnyel való leírási mód megváltoztatása. Ennek alapján kézenfekvő a halmazfogalom általánosítása.

Egy $F = \{(x, \mu_F(x)) : x \in X\}$ halmazt X *feletti fuzzy halmaznak* nevezünk, ahol az x elemek egy 0 és 1 közötti számmal jellemezhetőek, vagyis egy elem minél közelebb van az 1-hez, annál inkább tartozik a halmazhoz. Ez a *halmazhoztartozási vagy tagsági (membership) függvény*:

$$\mu_F(x) : X \to [0, 1].$$

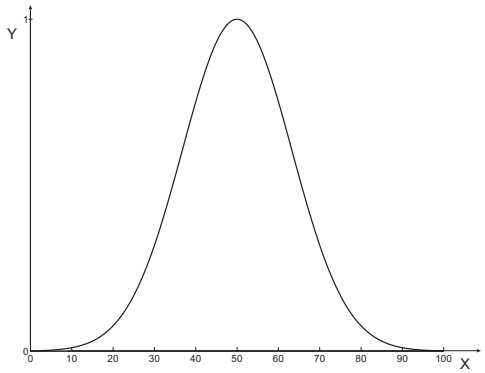
Zadeh példája: Vegyük a *fiatal*, *középkorú*, *idős* fogalmakat. Ezeket a tulajdonságokat klasszikus halmazként értelmezhetjük: $F(x)$ – fiatal, $K(x)$ – középkorú, $I(x)$ – idős. x-re az életkor függvényében vizsgáljuk ezeket. Minden halmazhoz hozzá tudjuk rendelni a karakterisztikus függvényét $(\chi_F(x), \chi_K(x), \chi_I(x))$:

$$\chi_F(x) = \begin{cases} 1, & \text{ha} \quad x \leq 30 \\ 0, & \text{különben} \end{cases}$$

$$\chi_K(x) = \begin{cases} 1, & \text{ha} \quad 30 < x \leq 50 \\ 0, & \text{különben} \end{cases}$$

$$\chi_I(x) = \begin{cases} 1, & \text{ha} \quad x > 50 \\ 0, & \text{különben} \end{cases}$$

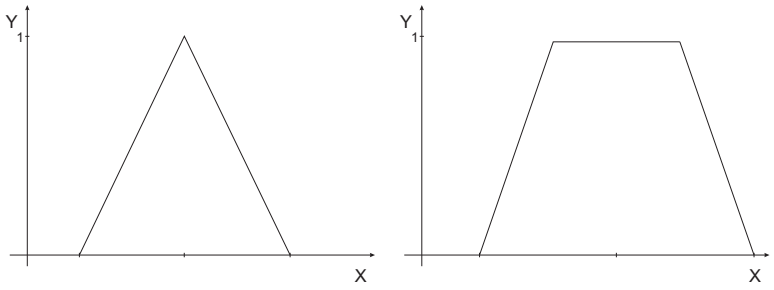
A fuzzy elméletben a tagsági függvényt rendeljük a fogalmakhoz. Tehát amíg a klasszikus halmaz karakterisztikus függvénye 0-t vagy 1-et rendel x-hez attól függően, hogy x eleme-e a halmaznak, a fuzzy tagsági függvény folytonos értékeket határoz meg x halmazhoztartozási mértékének kifejezésére (2.1. ábra).



2.1. ábra. Középkorú halmazhoztartozási függvénye–$\mu_K(x)$

Halmazhoztartozási függvények típusai

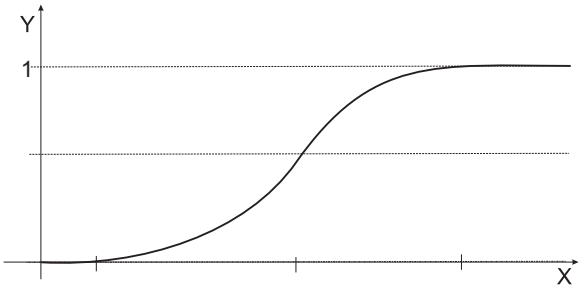
1. Trianguláris és trapéz



2.2. ábra. Trianguláris és trapéz függvény

2. Szigmoid

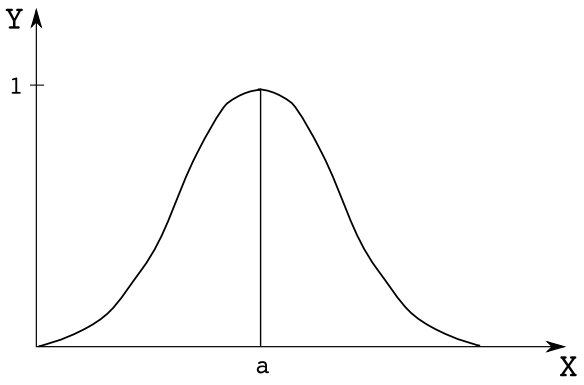
$$\sigma_a^\lambda = \frac{1}{1 + e^{-\lambda(x-a)}}$$



2.3. ábra. Szigmoid függvény (λ – meredekség, a – eltolás)

3. Haranggörbe

$$e^{-\lambda(x-a)^2}$$



2.4. ábra. Harang függvény.

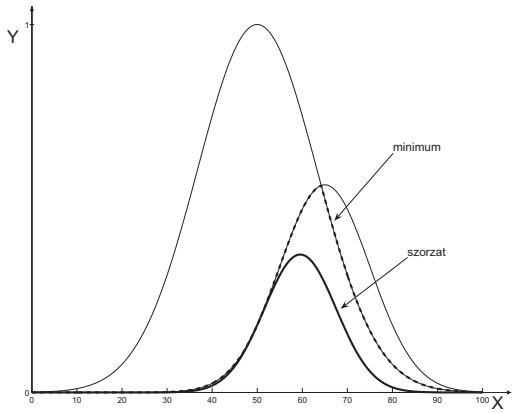
2.2.1. Fuzzy halmazműveletek

Az ilyen fuzzy halmazokon értelmeznünk kell az unió, a metszet, illetve a komplementer fogalmát. Zadeh első cikkében a következő két műveletet javasolta a metszetre:

$$\chi_{A\cap B}(x) = \chi_A(x) \cdot \chi_B(x), \quad \text{kiterjesztve} \quad \mu_{A\cap B}(x) = \mu_A(x) \cdot \mu_B(x).$$

$$\chi_{A\cap B}(x) = \min(\chi_A(x), \chi_B(x)), \quad \text{kiterjesztve} \quad \mu_{A\cap B}(x) = \min(\mu_A(x), \mu_B(x)).$$

Így rögtön két metszetműveletet is kaptunk: a szorzatot és a minimumot. A kapott eredmény eltérő, ahogy az a 2.5. ábrán látható. Innen is látszik, hogy nem triviális meghatározni a fuzzy műveleteket, a fuzzy halmazokon a műveletek többféleképpen is kiterjeszthetők.



2.5. ábra. Fuzzy halmazok metszetműveleteinek eredménye: szorzat és minimum

A következőkben a leginkább alkalmazott fuzzy műveleteket ismertetjük.

Alapvető fuzzy halmazműveletek

1. Unió műveletek:

$$\mu_{A\cup B}(x) = \max(\mu_A(x), \mu_B(x)) \tag{2.1}$$

$$\mu_{A\cup B}(x) = \min(1, \mu_A(x) + \mu_B(x)) \tag{2.2}$$

$$\mu_{A\cup B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x) \cdot \mu_B(x) \tag{2.3}$$

2. Metszet műveletek:

$$\mu_{A\cap B}(x) = \min(\mu_A(x), \mu_B(x)) \tag{2.4}$$

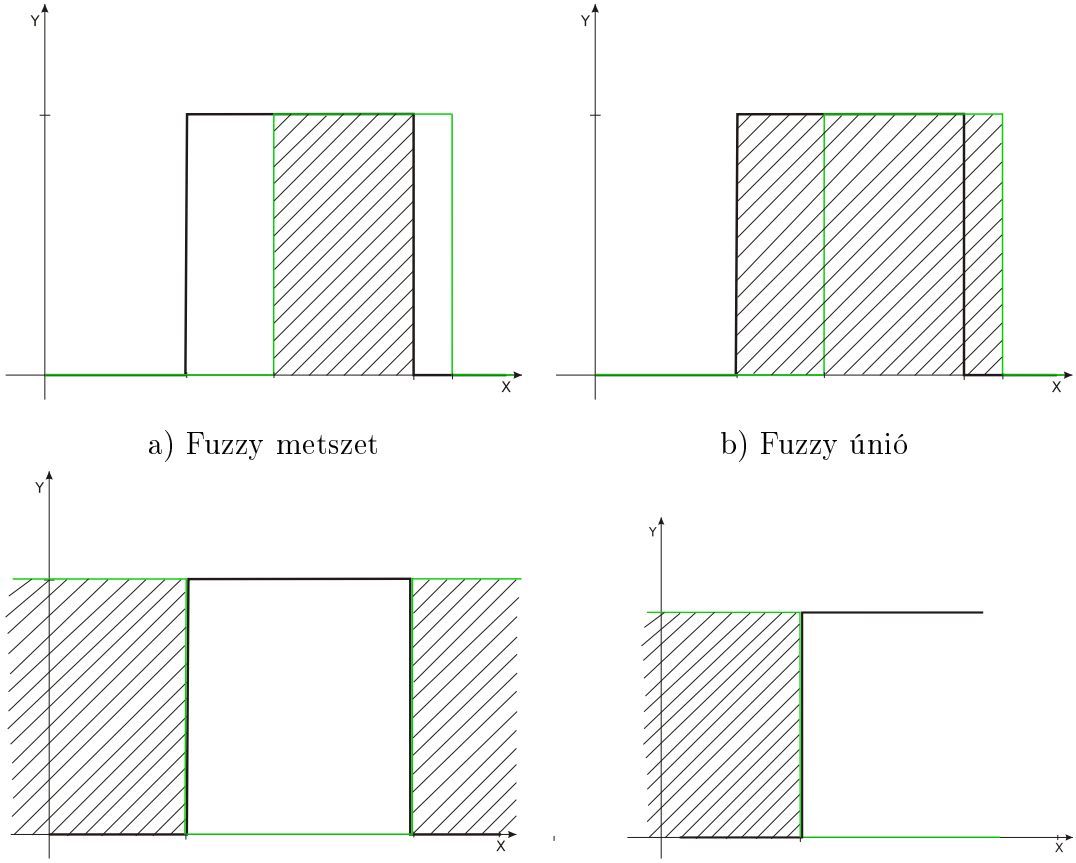
$$\mu_{A\cap B}(x) = \mu_A(x) \cdot \mu_B(x) \tag{2.5}$$

$$\mu_{A\cap B}(x) = \max(\mu_A(x) + \mu_B(x) - 1, 0) \tag{2.6}$$

3. Komplementer műveletek: (általában az $1 - x$ függvény a halmaz komplementere)

$$\mu_{\neg A}(x) = 1 - \mu_A(x) \tag{2.7}$$

$$\mu_{\neg A}(x) = \sqrt[p]{1 - \mu_A^p(x)} \quad p > 0 \tag{2.8}$$



a) Fuzzy metszet b) Fuzzy únió

c) Fuzzy komplementer d) Fuzzy komplementer

2.6. ábra. Fuzzy műveletek

A különféle megközelítéseket, műveleti struktúrákat szokás „világoknak” nevezni. A fuzzy fő kutatási területe, hogy milyen műveleti struktúrák vannak. Napjainkban fele annyi kutató foglalkozik már fuzzy elmélettel, mint a mesterséges intelligenciával. Az 1920-30-as években volt fő kutatási irányzat a többértékű (folytonos) logika. Az irányzatnak jeles lengyel képviselője volt Łukasiewicz. A folytonos logika törekvése az, hogy a hagyományos logikában megismert igaz és hamis értékeket kiterjesztve legyenek köztes értékek is, annak reprezentálására, ha nem tudjuk egy adott kijelentés igazságértékét egyértelműen eldönteni. Tehát a 0 és 1 diszkrét értékeket terjesszük ki a $[0, 1]$ intervallumra. A kérdés az volt akkoriban, hogy lehetséges-e ez a kiterjesztés. A Zadeh-i gondolat teljesen hasonló reprezentációt adott: a fentiekben láthattuk, hogy az $x \in A$ kifejezés egy többértékű függvénnyel írható le. Ha azonban a logikai igaz és hamis fogalmat fuzzy halmazokkal reprezentáljuk, kérdés, hogy hogyan fogjuk megkapni a többértékű logikai értékeket, pl. az $x \in A \wedge y \in A$ kifejezés értéket, amennyiben például az $(x \in A) = 0,7$ és az $(y \in A) = 0,8$? A fuzzy elmélet egyik fő irányzata, amely a logikával, állítások eldöntésével foglalkozik, pontosan ezen kérdés megválaszolására épül.

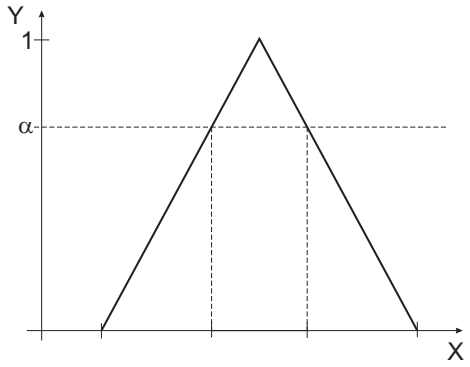
1. Fuzzy logikának (fuzzy következtetésnek) hívjuk a következő „világot”: $x \wedge y = \min(x, y)$. Egy $F = \{(x, \mu_F(x)) : x \in X\}$ fuzzy halmazt tekinthetünk úgy, mint egy X halmaz elemeire megfogalmazott állítást, ahol a μ_F tagsági függvény megadja az állítás igazságtartalmának a mértékét.

2. A fuzzy másik meghatározó irányzata a tolerancia-jellegű fuzzy (a „körülbelül” megfelelője). Például

adott egy körülbelül 2 méter hosszú zsinór, a teherbírása körülbelül 2 kg, így hány kg-ot bír el? Vagy mennyi a körülbelül 2 méter plusz körülbelül 5 méter mennyiség? Itt az aritmetikai műveletek a folytonosak, és általában mért értékekhez kötődnek a műveletek.

2.3. Fuzzy aritmetika

A függvényeken végrehajthatunk úgynevetett α-vágásokat, mely a következőt jelenti: egy $\alpha \in [0,1]$ szinttel elmetsszük a halmazhoztartozási függvényt. Az α-metszés (α-vágás) gyakorlatilag a mérés pontosságát reprezentálja.

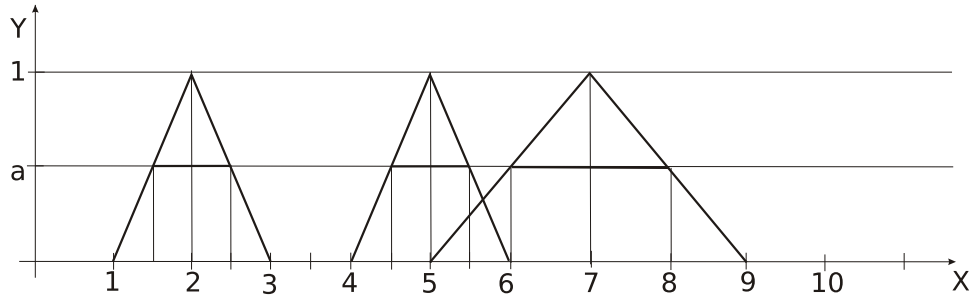


2.7. ábra. α metszés

Az α–vágás tehát a következőképpen történik: a $[0,1]$ intervallumon kijelölünk egy α számot és húzunk egy egyenest az α–szinten az x tengellyel párhuzamosan (2.7. ábra). Ez megfelel az $y = \alpha$ egyenletnek. Így a halmazhoztartozási függvényből egy intervallumot jelöltünk ki, és visszavezethetjük a fuzzy aritmetikát az intervallumaritmetikára.

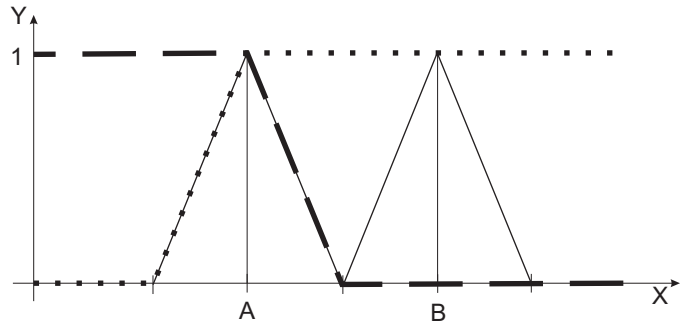
2.3.1. Összeadás és kivonás

Példánkat, miszerint mennyi lehet a körülbelül 2 és körülbelül 5 méter összege, a 2.8. ábra szemlélteti.



2.8. ábra. Körülbelül 2 és körülbelül 5 összegének visszavezetése intervallumaritmetikára

Tekintsünk egy példát az összeadás és kivonás gyors elvégzésére a fenti halmazokon! Első lépésként különválasztjuk a jobb és bal részeket a 2.9. ábrán látható módon. Majd képezzük a bal oldali kijelölt rész



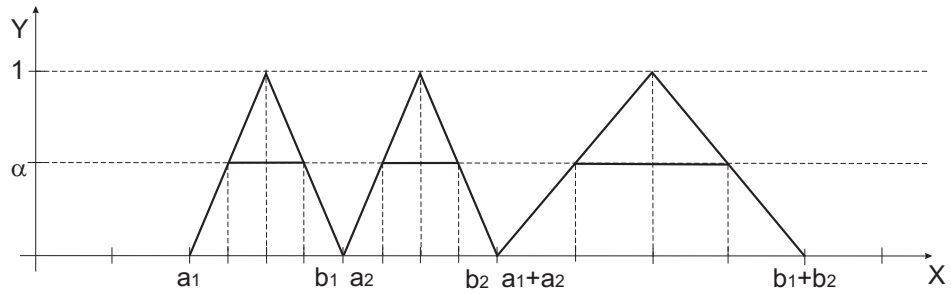
2.9. ábra. Összeadás és kivonás gyors elvégzése

inverzfüggvényét, és ezen végezzük el a műveletet! Az így kapott függvény inverzét jelölje s! Most pedig vegyük s bal oldalát, mely maga az összeg.

A kvadratikus halmazhoztartozási függvény inverze gyökös lesz, illetve lehet olyan halmazhoztartozási függvény is, melynek egyszerűen nem létezik az inverze. Ebből is látható, hogy az inverzképzés nem triviális művelet, nehézségekbe ütközhetünk.

Az első jelentős fuzzy tétel pontosan a fuzzy halmazokon végzett műveletekkel volt kapcsolatos. Ez az úgynevezett sup–min megközelítés, mely az alábbi módon definiálja az összeadást (2.10. ábra):

$$\mu_{A+B}(z) = \sup_{x+y=z} (\min(\mu_A(x), \mu_B(y)))$$



2.10. ábra. Összeadás sup–min közelítéssel

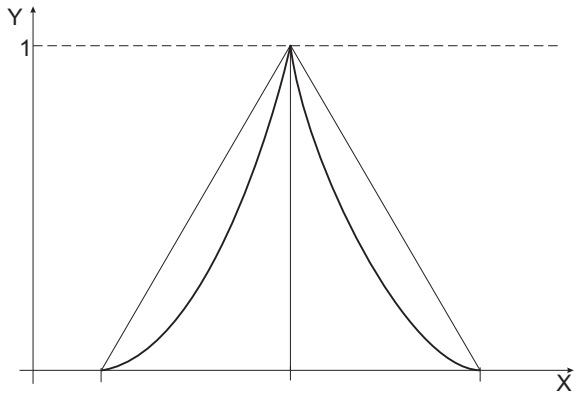
2.3.2. Szorzás

A trianguláris halmazok esetében a szorzat nem trianguláris lesz. Egy ilyen függvényhez általában megfelelő közelítést adunk például a 2.11. ábrán látható módon, amivel tovább tudunk dolgozni.

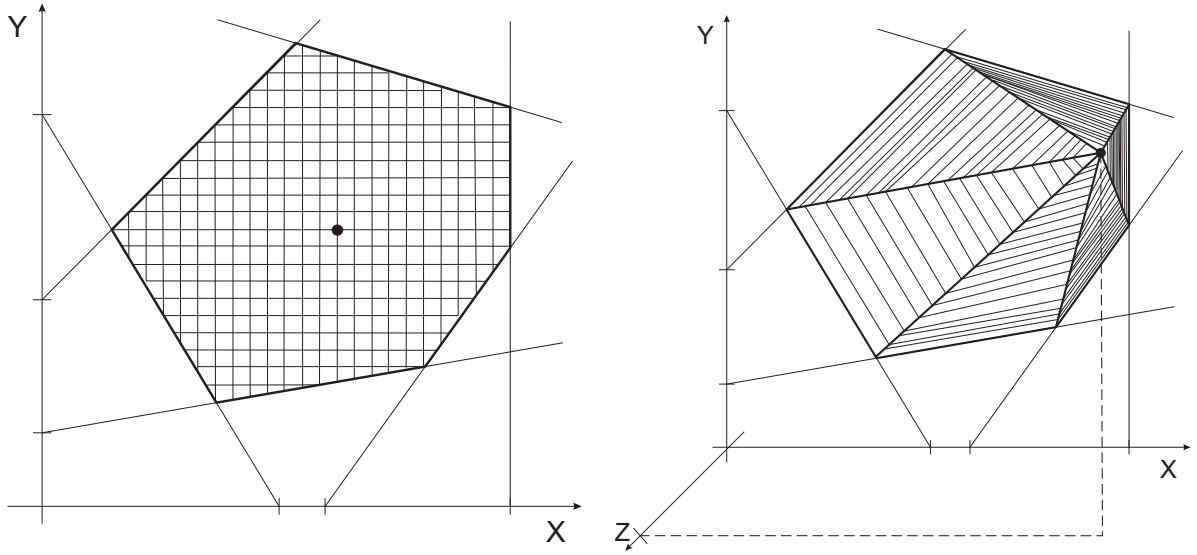
A min és max idempotens, asszociatív és disztributív. Bizonyítható, hogy a min–max szabállyal le lehet írni a fuzzy aritmetikát intervallum–aritmetikával.

A fuzzy aritmetika egyik alkalmazási területe az optimalizálás. Az optimalizálási feladatban adott egy lineáris feltételrendszer és egy célfüggvény az alábbi módon:

$$\begin{array}{rcl} \sum a_{ij}x_j & \leq & b_i \\ \hline c^T x & \rightarrow & \max \end{array}$$



2.11. ábra. Trianguláris halmazok szorzata nem trianguláris



a) b)

2.12. ábra. Optimalizálás: a) adott korlátok közé eső megoldás keresése; b) fuzzyfikált változat

b_i előjelétől függően adott egy alsó és/vagy egy felső korlát. A korlátok egy része $A_i \leq x_i \leq B_i$ alakú.

Tegyük fel, hogy ez a fenti feladat egy konzervgyár termelését reprezentálja, ahol zöldborsó, eper, illetve cseresznye konzerveket állítanak elő. Meg szeretnénk határozni, hogy miből mennyit érdemes készíteni, hogy maximális profitot érjünk el (lásd 2.12.a ábra). A lineáris programozás eredménye x_i változókra A_i és B_i, melyek nem tükrözik reálisan az elvárásokat, például ha azt kapjuk, hogy eperkonzervet egyáltalán ne készítsünk, az irreális. Ebből az következik, hogy egyszerűen nem megfelelő a célfüggvény, nem csak a profitot kell figyelembe vennünk.

Ha megadnánk az összes lehetséges célfüggvényt, akkor a kompromisszumok szerinti feltételrendszer belsejében lenne az optimum. A fuzzy elméleten alapuló optimalizálás célja, hogy stabil megoldást találjon a problémára, mely a gyakorlatban is elfogadható és kivitelezhető. Legtöbbször egy optimalizálási feladat során meg kell határozni az A és a B korlátokat. Például, ha egy ládában maximum 5 mázsa teher fér el, az nem feltétlen jelent szigorú korlátot, hiszen valószínűleg elfér még 5 mázsa és 10 kg is. Tehát érdemes B-re nem szoros korlátot alkalmazni, vagyis fuzzyfikálni (fuzzy halmazként értelmezni). Ezáltal még 1 dimenzióval bővül a feladat (lásd 2.12.b ábra).

Az így kapott csonkagúlánk élei a feltételrendszertől függően futnak össze. A fuzzyfikáció által keletkező új feltétel:

$$u_i = \alpha B + \gamma$$

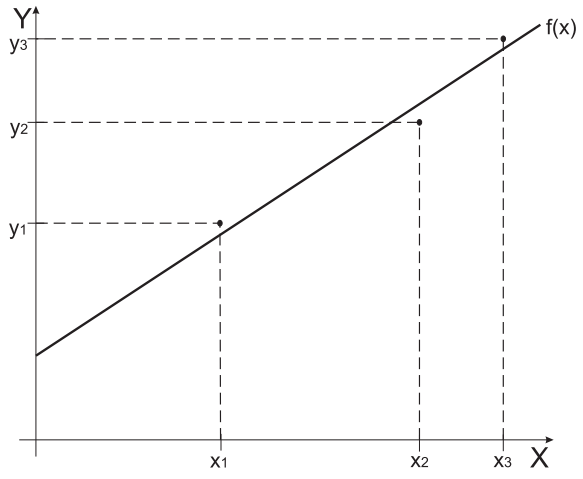
Az optimalizálás másik megközelítése azon a gondolaton alapszik, miszerint ha a B-t fuzzyfikáltuk, akkor értelmezhetnénk az A-t is fuzzy-ként. (Pl. az adott árut nem csak 4000 Ft-ért lehet eladni, hanem 5000 Ft-ért is → ez sem feltétlenül szigorú korlát.) Ez viszont egy másik, az előzőnél bonyolultabb fuzzy programozási feladathoz vezet. A fuzzy modell a parametrikus programozással ekvivalens.

2.4. Fuzzy regresszió

A regresszió két (vagy több) mennyiség közti összefüggés mérése, mely fogalommal a statisztika témakörében találkozhatunk. Legyenek adottak x_i és y_i mért adatok, ekkor olyan f függvényt keresünk, mely megadja az összefüggést x_i és y_i között. Az f függvény ismeretében leolvasható egy nem ismert x^* pontra, hogy mi az y^* érték. A regresszió során tehát mintákból kell következtetnünk egy függvényre, így a regresszió gyakorlatilag a legelemibb tanulási modell:

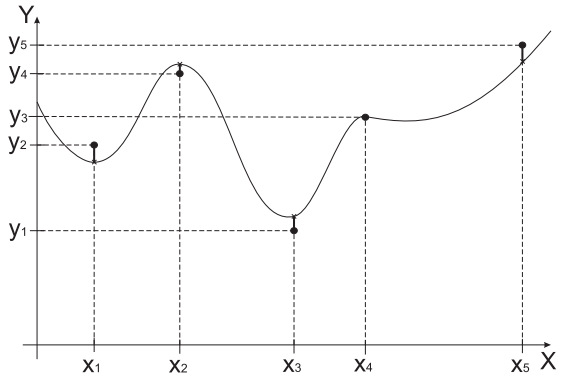
2 dimenzióban			**Több dimenzióban**		
x_1	→	y_1	$\underline{x_1}$	→	y_1
x_2	→	y_2	$\underline{x_2}$	→	y_2
	...			...	
x_n	→	y_n	$\underline{x_n}$	→	y_n

Amennyiben a függvényt lineáris alakban keressük, lineáris regresszióról beszélünk (2.13. ábra). A problémát az okozza, hogy a pontokra nagyon sok függvényt illeszthetünk. Egy lehetséges megoldás a polinomillesztés. Az interpolációs feladat esetén $p_n(x_i) = y_i$, ahol n a polinom fokszámát jelenti, ami az (x_i, y_i) párok darabszáma.



2.13. ábra. Lineáris regresszió

Ahhoz, hogy le tudjuk kérdezni az adatokat, el kell tárolnunk az $\underline{x}$ vektorokat és az y értékeket. Az iterpolációs polionom együtthatóinak száma szintén n. Így vagy az együtthatókat, vagy a vektorokat kell eltárolni, így nincs érdemi különbség a két megközelítés között (2.14. ábra).



2.14. ábra. Interpolációs polinommal való közelítés

Tehát n darab adat eltárolására lenne szükség (amennyi mintánk volt). Azt mondhatjuk, hogy az interpoláció nem tanul, hanem „magol". Emlékezzünk, hogy a karakterfelismerésnél tömörítést kell végrehajtanunk és így az algoritmus ténylegesen tanul. Az interpolációs polinom tulajdonsága, hogy „rezeg" a pontok között, és nem egyenletesen konvergál a közelítendő függvényhez.

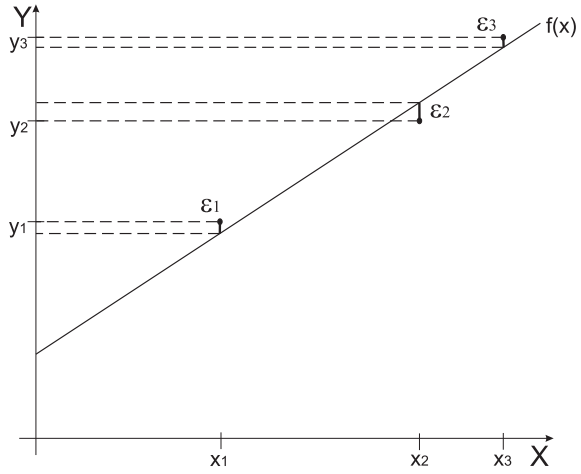
$\mathbf{x_i}$ pontok $\mathbf{F(x_i)}$ értékeinek közelítése esetén:

$$\left.\begin{array}{rcl} y_1 & \to & F(x_1) \\ y_2 & \to & F(x_2) \\ & \dots & \\ y_i & \to & F(x_i) \\ & \dots & \\ y_n & \to & F(x_n) \\ \hline y_1 & \to & p_n(x_1) \\ y_2 & \to & p_n(x_2) \\ & \dots & \\ y_i & \to & p_n(x_i) \\ & \dots & \\ y_n & \to & p_n(x_n) \end{array}\right\} \text{viszonylag nagy eltérés lesz a valós pontokhoz képest}$$

A tanulásnál tehát nem érdemes n-ed fokú polinomot használni. Szükség van tömörítésre is, például egyenessel való közelítéshez összesen 2 paraméter kell n helyett. Megjegyezzük, hogy a regresszió alatt általában regressziós egyenest szokás érteni.

A 2.15. ábrán látható regressziós egyenes esetén a hiba $\underline{\varepsilon}$, és az egyenlet $ax+b$ alakú. Ekkor minimalizálni szeretnénk a hibát, vagyis:

$$\sum_{i=1}^{n} \varepsilon_i^2 \to \min \qquad = \qquad \min_{a,b} \left(\sum_{i=1}^{n} (y_i - (\underbrace{ax_i + b}_{y^*}))^2 \right),$$



2.15. ábra. Lineáris regresszió – az $f(x)$ egyenes ε hibával közelíti a pontokat

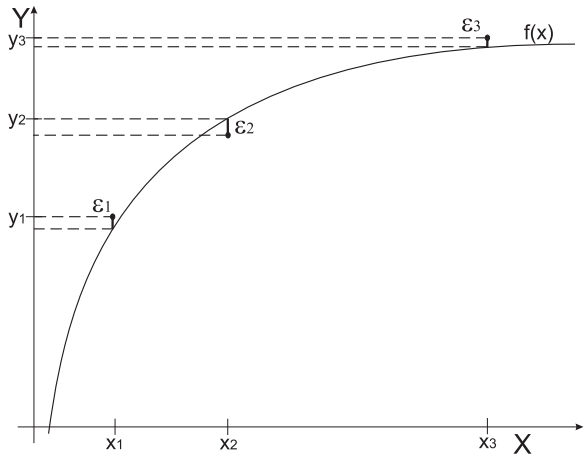
ahol adottak az x_i és y_i értékek, a illetve b ismeretlenek. A minimalizálandó célfüggvény a-nak és b-nek egy másodfokú célfüggvénye. Kereshetnénk más alakban is a közelítő függvényt, pl. $a \log x + b$ alakban:

$$\min_{a,b} \left(\sum_{i=1}^{n} (y_i - (a \log x_i + b))^2 \right).$$

A képlet hasonló, mint az előző esetben, csupán annyiban tér el, hogy az x helyett $\log x$ áll. Az $X_i = \log x_i$ transzformáció bevezetésével ugyanazt a feladatot kell megoldani, mint az előbb. Így:

$$\min_{a,b} \left(\sum_{i=1}^{n} (y_i - (aX_i - b))^2 \right).$$

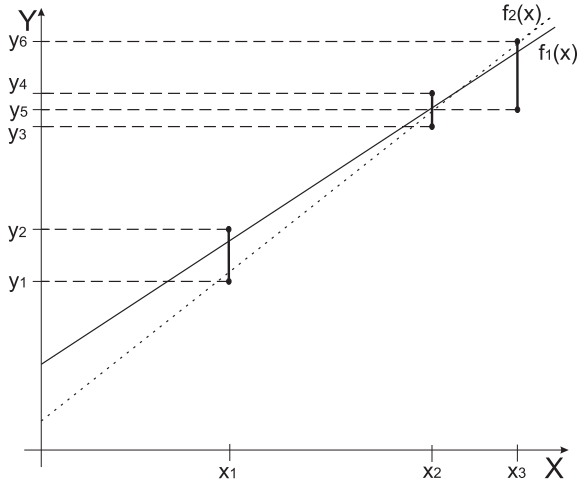
A kapott minimalizálást elvégezve ugyanarra az eredményre jutunk. Viszont, ha ellenőrizzük a hibát, láthatjuk, hogy a transzformáció nélküli esetben nem ugyanakkora, mivel a transzformáció megváltoztatja a hiba nagyságát. A hiba mértéke a transzformáció által a logaritmusfüggvény linearizálása miatt változott meg (lásd 2.16. ábra). Ebből következik, hogy a hagyományos regresszió egy ilyen transzformációval szemben nem őrzi meg az optimumot.



2.16. ábra. Regresszió logaritmusfüggvénnyel

A regresszió a legtöbb esetben mérési adatokhoz kötődik. Minden méréshez érdemes lenne mérési hibát megadni, amit a hagyományos regresszió (szemben a fuzzy regresszióval) nem kezel. A fuzzy regresszió azon

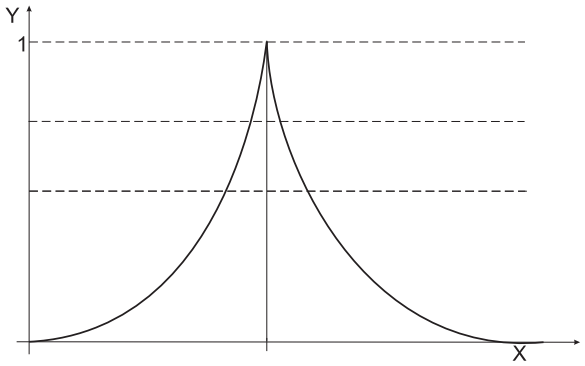
alapszik, hogy minden méréshez adott egy ε_i mérési hiba, és ha ezen hibákon belül halad az egyenes (az egyenes metszi a hibákat), akkor elfogadjuk az adott egyenest.



2.17. ábra. Fuzzy regressziós egyenes

Fuzzy regressziós egyenesnek nevezzük azt az egyenest, ami eleget tesz a hibafeltételnek (2.17. ábra). Legtöbbször végtelen sok ilyen tulajdonságú egyenes létezik. Az egyeneseknek egyfajta „kilengése" vagy „billegése" lehet, ami a hibahatároknak eleget tesz. A hibákhoz tehát halmazhoztartozási függvényeket rendelünk. A halmazhoztatozási függvény alakja legtöbbször a hiba valószínűségi jellegétől függ (megjegyezzük, hogy ez a gyakorlatban nehezen megadható). Az így kapott halmazhoztartozási függvényeken α-vágásokat hajtunk végre. Ha növeljük α értékét, kisebbek lesznek a hibaintervallumok, és ezáltal kisebb lehet az egyenesek „kilengése". Addig növeljük α értékét, amíg az egyenes egyértelmű nem lesz, vagyis keressük $\max \alpha$-t. Az algoritmus szerint az egyenes paramétereinek a tartományát (lehetséges megoldások) szűkítjük addig, amíg egyetlen pontot nem kapunk, mely pont fuzzy optimum az adott α-vágásra nézve.

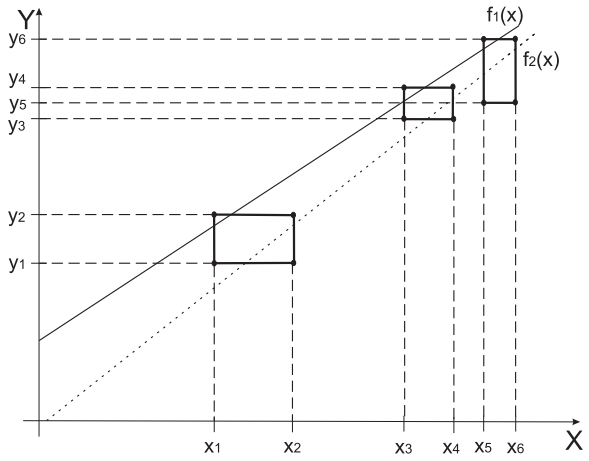
Az algoritmus nem alkalmazható a gyakorlatban, ha nincs olyan egyenes, ami a hibafeltételeknek eleget tesz, illetve nem mindig találunk megoldást. Ebben az esetben a 2.18. ábrán látható halmazhoztartozási függvénnyel közelítjük az optimumot.



2.18. ábra. Halmazhoztartozási függvény közelítése – x–tengelyre asszimptotikus

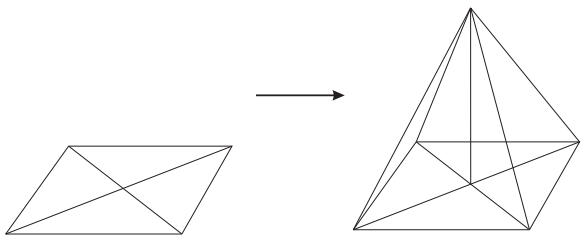
Az ilyen alakú függvénynek az az előnye, hogy asszimptotikusan közelíti az x-tengelyt, így mindig tudunk olyan α-szintet választani, mely megoldást ad. A problémát kiterjeszthetjük az alábbi módon: nem

pontosan beállított x értékekkel dolgozunk. Ekkor a regressziós egyenesnek az alábbi határokon belül kell mozognia (lásd 2.19. ábra).



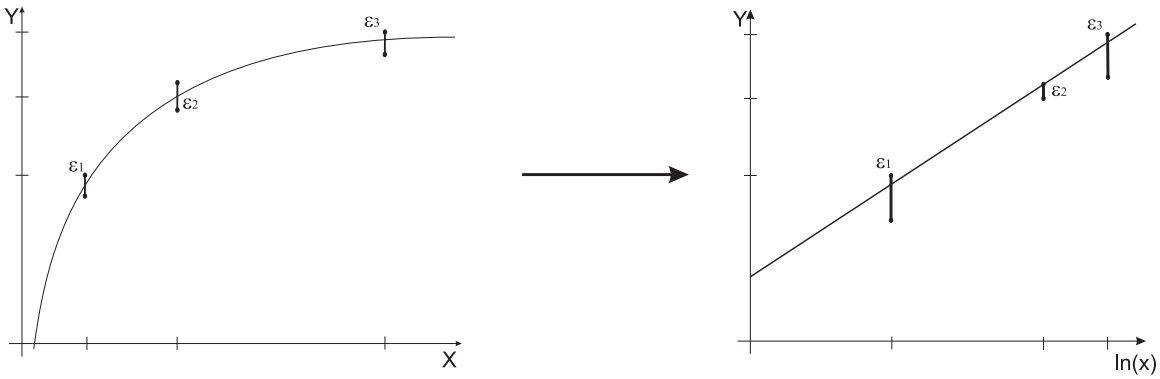
2.19. ábra. Fuzzy regressziós egyenesek mozgása a hibahatárokon belül

A legtöbb esetben a halmazhoztartozási függvények triangulárisak, így tulajdonképpen gúlákat definiáltunk a hibák felett és a gúlákat metsszük el a megoldás megtalálása érdekében (2.20. ábra).



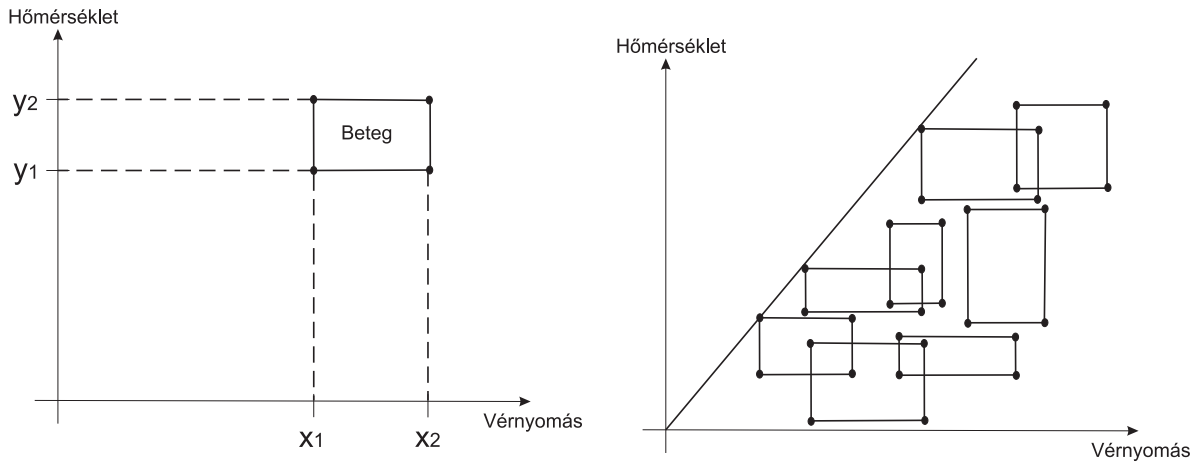
2.20. ábra. A hibák felett gúlákat definiálunk a trianguláris halmazhoztartozási függvényekkel

Az eddigi esetekben lineáris alakban kerestük a regressziót, és láttuk, hogy a hagyományos regresszió nem őrzi meg az optimumot, ha nemlineáris térből lineárisba transzformáljuk a függvényt. A fuzzy regresszióra érvényes, hogy megőrzi a terek transzformációi közt a *fuzzy értelemben vett optimumot* (2.21. ábra).



2.21. ábra. Fuzzy regresszió megőrzi a terek transzformációi közt a fuzzy értelemben vett optimumot.

Logaritmikus alakban keressük a függvényt, ahol adottak a hibák. A függvényt, illetve a hibákat is áttranszformáljuk a lineáris térbe. Ezáltal a visszafelé vett transzformáció során megőrződik az optimum.



a) b)

2.22. ábra. Szeparálósíkok

A fenti algoritmusok mind a 2 dimenziós esetet tárgyalták. A fuzzy regressziós algoritmus magasabb dimenziókra bonyolult számítást igényel.

2.5. Fuzzy lekérdezés

A szavainkat tekinthetjük a bennünket körülvevő világ digitalizálásának, ugyanis a világ dolgait előre meghatározott kategóriákba soroljuk. A szavak alapján történő kommunikációval információt veszítünk, hiszen dolgokat diszkrét kategóriákba konvertáljuk, „intervallumokra" bontjuk. Például, ha azt szeretnénk 2 mért értékből kiszűrni, hogy az adott páciens beteg-e vagy sem, akkor bizonyos esetekben találnánk olyan szeparálósíkot, mely elkülönítené a beteg és nem beteg pácienseket. A szeparálósíknak ebben az esetben mindössze 2 paramétere van (2.22.a ábra). Szavak esetén a mért tulajdonságokat kategóriákba osztjuk és két dimenzióban téglalapokat adunk meg. Ha pontosan akarnánk a betegeket leírni, végtelen sok téglalapra lenne szükség (2.22.b ábra).

Ebből is látszik, hogy a szavakkal történő leírás csak körülírás, és bár a tanulás szavakkal történik, a tudás maga sokkal több annál, mint amit szavakban ki tudunk fejezni (a többéves tapasztalat szavakkal nem átadható, a férfi és nő megkülönböztetetését, ami az emberi agynak triviális, nehéz leírni szavakkal).

A fuzzy tulajdonképpen a természetes szavak összekapcsolása a matematikával. A fuzzy elméletben csak bizonyos szavakat modellezünk:

1.) Mérésekhez rendelhető fogalmak: mérésekhez kapcsolódik a jelentésük. Pl.
   életkor → fiatal, középkorú, idős
   hosszúság → rövid, hosszú
   súly → könnyű, nehéz

2.) Poláris fogalmak: páronként ellentétes dolgokat fejeznek ki, de ha tagadjuk az egyiket, nem a másikat kapjuk vissza. Pl.
   alacsony - magas → nem alacsony $\neq$ magas

illetve

nem magas $\neq$ alacsony

3.) Kontextusfüggő fogalmak: a jelentésük erősen függ attól, hogy milyen környezetben használjuk őket. Pl.

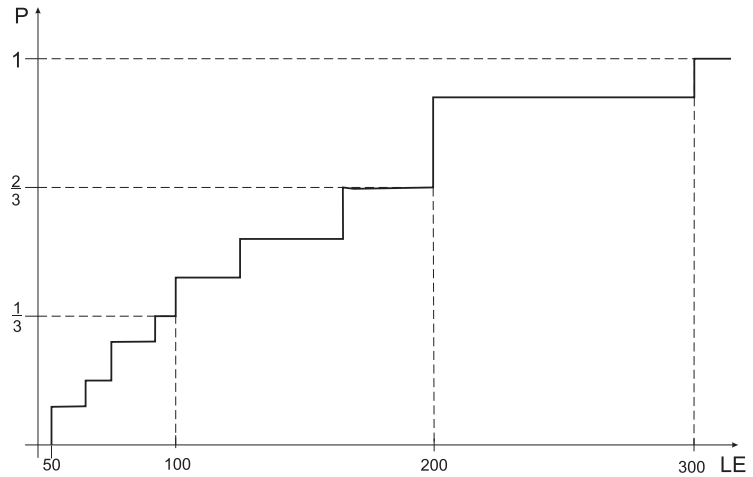
fiatal akadémikus	$\rightarrow$	kb. 45-50 éves
fiatal munkatárs	$\rightarrow$	kb. 25 éves
fiatal műkorcsolyázó	$\rightarrow$	kb. 12 éves

Tekintsük a kontextusfüggőség modellezését! Vegyük az alábbi példát: autót szeretnénk vásárolni, és szeretnénk sorrendet állítani a lehetséges alternatívák között. A következő szempontokat vizsgáljuk: lóerő (továbbiakban: LE, jelen esetben a 50-300 LE teljesítményű gépjárműveket vizsgálunk), illetve benzines vagy dízel. Teljesítmény szerint az autókat felosztjuk 3 kategóriára: kis teljesítményű (50-100 LE), közepes teljesítményű (101- 200 LE), illetve nagy teljesítmányű (201-300 LE). Ezzel a megközelítéssel az a baj, hogy például a 201 és 300 LE közötti autóból csak egy van az adabázisunkban, míg 50-100 LE között 652 darab. A következő eljárás szerint lehet jól kategorizálni: adjunk meg egy rangsort az alternatívák között a LE szerint!

alternatívák		LE
a_1	$\rightarrow$	x_1
a_2	$\rightarrow$	x_2
	$\dots$	
a_n	$\rightarrow$	x_n

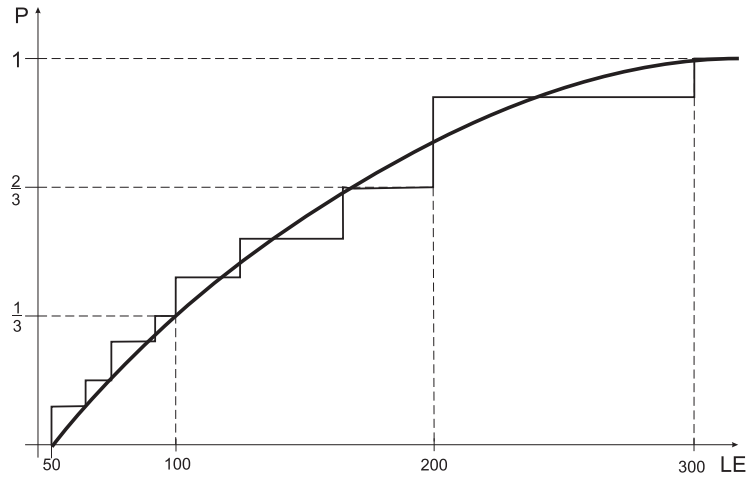
LE szerinti rendezettség: $x_i \leq x_j$, ahol $i < j$

Készítsük el az autók LE szerinti empirikus eloszlásfüggvényét (lépcsős függvény – 2.23. ábra)!



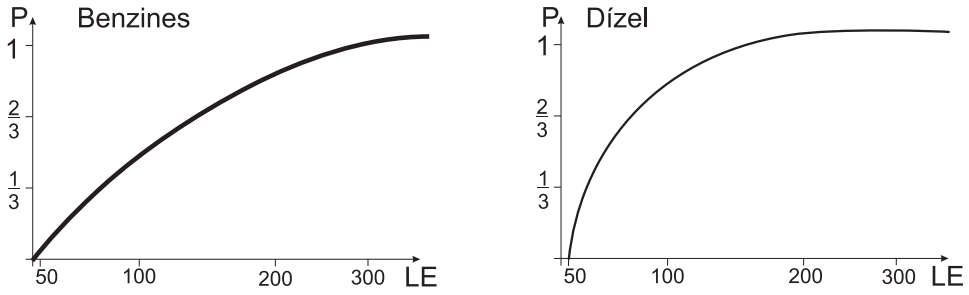
2.23. ábra. Az autók LE szerinti empirikus eloszlásfüggvénye

A függvényértékek 0 és 1 közé esnek. A felvett függvényértékek szerint határozzuk meg a kis, közepes, és nagy teljesítményű autókat. Ez azt jelenti, hogy a kategoriákba közel azonos számú alternatíva tartozik. A lépcsőzetességet egy függvénnyel közelítjük (lásd 2.24. ábra).



2.24. ábra. Empirikus eloszlásfüggvény közelítése.

Ha a benzines és dízel tulajdonság szerint osztjuk ketté az autók halmazát és mindkét halmazra megkonstruáljuk az empirikus eloszlásfüggvényt, és megkonstruáljuk a kategóriákat, akkor kontextusfüggő értelmezést kapunk. Dízel esetben 60-90 LE a kis teljesítményű, míg benzinesnél 50-100 LE (2.25. ábra).



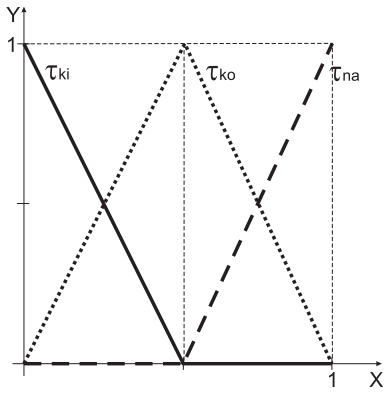
2.25. ábra. Benzines és dízel autók empirikus eloszlássfüggvényének közelítése.

A gyakorlatban azonban a fenti módszert módosítani kell, mert például adózási határok miatt nagy ugrások lehetnek az eloszlásfüggvényben, ekkor ugyanis az ugrás határa a kategóriahatár. (Pl. törvény írja elő a károsanyag-kibocsátást, az adózás mértékét a teljesítmény határozza meg, stb.) Reprezentáljuk a kontextusfüggőség halmazhoztartozási függvényét a 2.26. ábrán látható módon! A 3 halmazhoztartozási függvény a $[0, 1]$ intervallumon értelmezett.

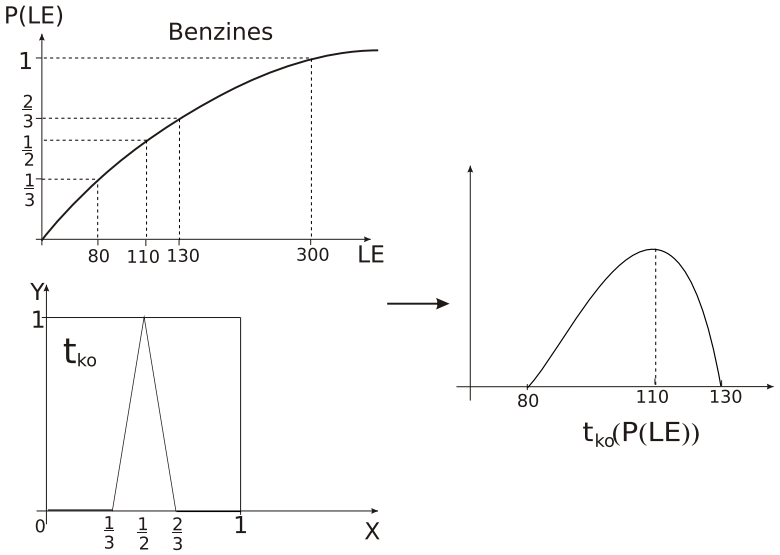
$$\begin{aligned} \tau_{ki} &\rightarrow \text{kis tejesítményű} \\ \tau_{ko} &\rightarrow \text{közepes teljesítményű} \\ \tau_{na} &\rightarrow \text{nagy teljesítményű} \end{aligned}$$

Alkalmazzuk a 3 függvényt az adott tulajdonság eloszlásfüggvényére, például a közepes teljesítményre, akkor megkapjuk a közepes teljesítmény kontextusfüggő fuzzy leírását ($\tau_{ko}(F(x))$ - közepes teljesítményű autók). Ezt szemlélteti a 2.27. ábra.

A fenti eljárás megfelel egy fuzzy adatbázis-lekérdezésnek. A hagyományos adatbázis-lekérdezésekkel az a baj, hogy nem kezelik a kontextusfüggőséget, illetve hogy éles határokkal dolgoznak. Ezért nem tudunk



2.26. ábra. Kontextusfüggőség halmazhoztartozási függvénye



2.27. ábra. Közepes teljesítmény kontextusfüggő leírása.

olyan lekérdezéseket végrehajtani, hogy például „Szeretném lekérni a vizsonylag nagy teljesítményű kombi autók listáját!", ezt például az SQL nyelv nem tudja lekezelni. Csak olyan SQL lekérdezést tudunk készíteni, amely éles határokon belül szűri az adatokat. Természetesen felmerül egy ilyen esetben az igény, hogy egy lekérdzés végeredménye egy sorrend legyen az alternatívák között. A hagyományos adatbáziskezelő rendszerek ezt sem biztosítják (pl. egy SQL SELECT utasítás eredménye egy lista, mely elemei megfelelnek a feltételeknek). Egy fuzzy lekérdezés végeredménye viszont egy rendezés az alternatívákon a megadott szempontok szerint, figyelembe véve a kontextusfüggőséget is. A fuzzy lekérdezések leírásához többértékű logika szükséges.

2.6. Többértékű logika

Jelölje $c(x, y)$ az „és" műveletet (konjunkció), $d(x, y)$ a „vagy" műveletet (diszjunkció), $n(x)$ pedig a negáció műveletet. A többértékű logika, Zadeh definíciója szerint lehet:

			Duálisok
$c(x,y)$	$=$	$\min(x,y)$	$d(x,y) = \max(x,y)$
$c(x,y)$	$=$	$x \cdot y$	$d(x,y) = x + y - x \cdot y$
$n(x)$	$=$	$1 - x$	

A min(x, y) tulajdonságai:

1. $c(x, y) \to$ folytonos
   $[0, 1] \times [0, 1] \to [0, 1]$
2. monoton (nem szigorúan)
3. $c(0, 0) = 0$
   $c(1, 1) = 1$
   $c(0, 1) = c(1, 0) = 0$
4. asszociatív: $c(c(x, y), z) = c(x, c(y, z))$
5. idempotens: $c(x, x) = x$

Tétel: A fenti 5 tulajdonság akkor és csak akkor teljesül, ha $c(x, y) = \min(x, y)$.

A fuzzy elmélet nagyon gyakran használja a $\min(x, y)$, $\max(x, y)$ műveletet, de a fuzzy alkalmazásokban inkább az $x \cdot y$ művelet a preferált. Ez utóbbi azért előnyös, mert a $min(x, y)$ esetében az egyik argumentum tulajdonképpen nem befolyásolja a végeredményt.

Fontos szerepet játszik még az alábbi logika, mely Łukasiewicz-logika, korlátos összeg vagy nilpotens néven is ismert:

$$\begin{aligned} c(x,y) &= \max(x + y - 1, 0) \\ d(x,y) &= \min(x + y, 1) \end{aligned}$$

A nilpontens operátorok esetében egy új Boole-azonosság is teljesül:

6. osztályozó tulajdonság:

$$\begin{aligned} x \wedge n(x) &= 0 \\ x \vee n(x) &= 1 \end{aligned}$$

A Łukasiewicz-logika nagyon jó operátorstruktúrát alkot, mert teljesíti a 6. tulajdonságot:

$$\begin{aligned} \max(x + (1 - x) - 1, 0) &= 0 \\ \min(x + (1 - x), 1) &= 1 \end{aligned}$$

A Łukasiewicz-operátor nem idempotens operátor, mert:

$$\max(x + x - 1, 0) \neq x$$

Ezért a fent ismertetett 5 feltételhez vegyünk fel még a 6. tulajdonságot (az utolsó Boole-azonosságnak megfelelőt):

$$c(x, 1 - x) = 0$$

Tétel: Nem létezik olyan operátorcsalád, amely mind a 6 feltételt teljesíti.

A különböző többértékű és folytonos logikák aszerint térnek el egymástól, hogy milyen Boole-azonosságot teljesítenek.

Térjünk rá a gyakorlat által gyakran alkalmazott $x \cdot y$ konjunkciós műveletre! Ebben az esetben módosítanunk kell a feltételrendszeren. Egyrészt ki kell kötni a szigorú monotonitást (2. feltétel), illetve módosítanunk kell az 5. feltéltelt is, a következőképpen:

$$c(x, x) < x,$$

amit **arkhimédesziségnek** nevezünk.

Mielőtt rátérnénk a 4. feltétel függvényegyenletének megoldására, nézzünk további operátorokat!

2.6.1. Fuzzy operátorok

Az előzőekben megismert arkhimédeszi világban műveleteket fogunk definiálni.

1. Valószínűségi operátor

$$c(x, y) = x \cdot y$$

2. Hamacher-operátor

$$c_p(x, y) = \frac{p \cdot x \cdot y}{1 - (1 - p)(x + y - x \cdot y)} \quad (p > 0)$$

Kizárólag ez az operátor írható fel racionális tört alakban, azaz két polinom hányadosaként, a p az operátor paramétere.

3. Frank-operátor

$$c_s(x,y) = \log_s\left(1 + \frac{(1-s^x)(1-s^y)}{1-s}\right) \quad (s>0)$$

Összefüggések elemzéséhez használják, az s paraméter az összefüggésekre jellemző (azt jellemzi, hogy az egyik változó értéke mennyire függ a többitől).

4. Dombi-operátor

$$c_\alpha(x,y) = \frac{1}{1+\left(\left(\frac{1-x}{x}\right)^\alpha + \left(\frac{1-y}{y}\right)^\alpha\right)^{\frac{1}{\alpha}}} \quad (\alpha>0)$$

Magába foglalja a diszjunkció és a konjunkció műveletét is, konjunkció esetén $\alpha > 0$, diszjunkció esetén $\alpha < 0$. Az α paraméter megadja, hogy mennyire arkhimédeszi a függvény (ha $\alpha \to \infty$, akkor a min operátort kapjuk).

5. Einstein-operátor

$$c_E(x,y) = 1 - \frac{(1-x)(1-y)}{1-(1-x)(1-y)}$$

Az Einstein-diszjunkció alakja az alábbi:

$$d_E(x,y) = \frac{x+y}{1+xy}$$

Az Einstein-operátor egy speciális alakja a Hamacher-operátornak. Az elnevezést a következő indokolja: Einstein relativitás elméletének alapgondolata az, hogy a testek mozgása viszonylagos egymáshoz képest, és az elmélet másik fő tétele, hogy nem létezik a fénysebességnél nagyobb sebesség. Ha például vonaton sétálunk, Newton szerint az eredő sebességünk egy külső szemlélő számára:

$$v_e = v_1 + v_2,$$

ahol v_e az eredő sebesség, v_1 a vonat sebessége, és v_2 a mi sebességünk. Kérdés, hogy mennyi a fény sebessége akkor, ha egy mozgó vonaton felkapcsolunk egy lámpát? A fenti, egyszerű összeadáson alapuló képlettel nem jó eredményhez jutunk, hiszen nincs a fénysebességnél nagyobb sebesség. Einstein szerint:

$$v_e = \frac{v_1+v_2}{1+\frac{v_1 v_2}{c^2}}$$

Ez a képlet valóban helyes, mert ha például $v_2 = c$, akkor is c-t ad eredősebességnek:

$$v_e = \frac{v_1+c}{1+\frac{v_1 c}{c^2}} = c$$

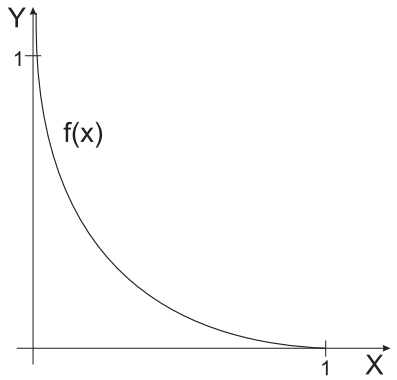
A képletből származik az Einstein-operátor a következő módon: c-vel osztjuk mindkét oldalt, és $x = \frac{v_1}{c}$, $y = \frac{v_2}{c}$, akkor

$$\frac{v_e}{c} = \frac{x+y}{1+xy}$$

Tétel: Az asszociatív függvényegyenlet az 5 feltétel mellett a következő alakban adható meg:

$$c(x,y) = f^{-1}(f(x) + f(y)),$$

ahol $f(x)$ az operátor generátorfüggvénye és konstans szorzó erejéig meghatározott. Az f olyan függvény, mely szigorúan monoton csökkenő, $[0,1]$-en értelmezett, és 1-ben 0-t vesz fel, a 0-ban aszimptotikus, alakját lásd a 2.28. ábrán.



2.28. ábra. Konjunkció operátor generátorfüggvénye

Az asszociatív függvényegyenlet működése

1. Ha $f(x) = -\ln x$.

   Képezzük az inverzfüggvényt!

   $$f^{-1}(x) = e^{-x}$$

   A tétel szerint: $(-\ln x + -\ln y) = -\ln xy$ -t kell $f^{-1}(x)$-be helyettesíteni:

   $$f^{-1}(f(x) + f(y)) = e^{-(-\ln xy)} = xy$$

   Tehát $f(x) = -\ln(x)$ esetén operátor az xy.

2. Ha $f(x) = (\frac{1-x}{x})^{\alpha}$

   inverzfüggvény:

   $$f^{-1}(x) = \frac{1}{1 + x^{\frac{1}{\alpha}}}$$

   Ha alkalmazzuk a tételt, épp a Dombi-operátorhoz jutunk.

Minden arkhimédeszi operátor egy generátorfüggvény segítségével áll elő.

Az asszociativitás igazolása:

$$c(c(x,y),z) = f^{-1}(f(\underbrace{c(x,y)}_{f^{-1}(f(x)+f(y))}) + f(z)) = f^{-1}(f(x) + f(y) + f(z))$$

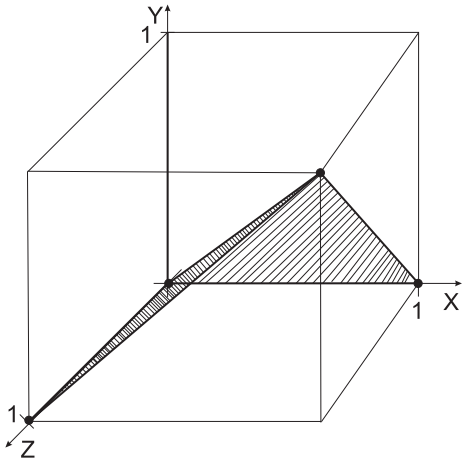
Hasonló eredmény kapható a $c(x, c(y,z))$ képletből kiindulva, így igazolható az asszociativitás.

Állítás: A generátorfüggvény konstanssal való szorzás erejéig egyértelmű.
Tehát az xy-nak generátorfüggvénye a $-5\ln x$ is. (Ennek a megállapításnak a bizonyítását az olvasóra bízzuk.)
Az asszociatív függvényegyenlet megoldása kommutatív megoldást ad.

Hilbert problémája: Adott egy többváltozós függvény. Előállítható-e ez a függvény folytonos egyváltozós függvények kompozíciójaként?

$$F(x,y,z) = g(f_1(x) + f_2(y) + f_3(z))$$

Kolmogorov kompozíciós tétele kimondja, hogy ez lehetséges. Az asszociatív többváltozós függvények esetén $f_1(x) = f_2(x) = f_3(x)$ és $g(x) = f^{-1}(x)$.
Az operátorok szemléltethetők egy 3 dimenziós egységkockával (2.29. ábra).



2.29. ábra. $\min(x,y)$

Vizsgáljuk meg a Łukasiewicz-logikát és általánosításait!

$$c(x,y) = \max(0, x+y-1), \tag{2.9}$$

$$d(x,y) = \min(1, x+y). \tag{2.10}$$

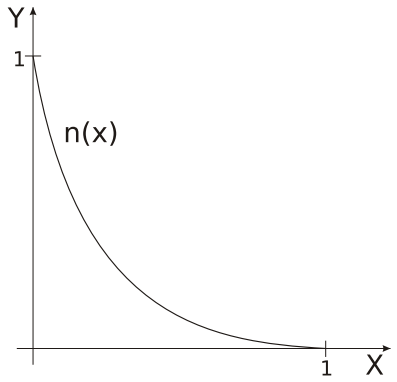
A nilpotens operátorok nem szigorúan monotonok, de arkhimédesziek. Vizsgáljuk meg, hogy hogyan lehet ezeket felírni!

Pseudo–inverz

Vezessük be a következő jelölést, ahol a [ ] a pszeudo-inverzet jelenti, és:

$$[x] = \begin{cases} 1, & \text{ha} \quad x > 1, \\ 0, & \text{ha} \quad x < 0, \\ x, & \text{különben.} \end{cases}$$

Így $c(x, y)$ generátorfüggvényének alakja: eddig aszimptotikus volt, most érinti a tengelyeket (lásd 2.30. ábra).



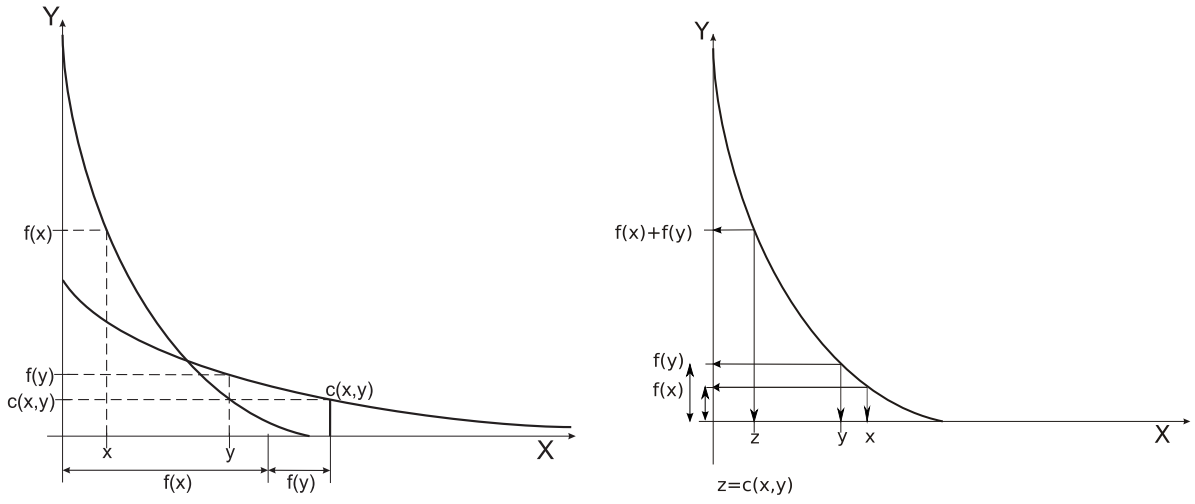
2.30. ábra. Konjunkció pszeudo–inverz generátorfüggvénye

Így a Lukasiewicz-operátor:

$$\begin{aligned} c\,[x, y] &= [y + x - 1] \\ d\,[x, y] &= [x + y] \end{aligned}$$

Tétel: Az összes nilpotens operátor felírható az alábbi alakban:

$$c\,[x, y] = f^{-1}\,[f(x) + f(y)]$$



a) b)

2.31. ábra.

Most azonban mivel a generátorfüggvény érinti a tengelyeket, egy pontnál tovább negatív értékeket venne fel a függvény. A negatív értékek helyett végig 0-t veszünk. Ezért *pszeudo-inverzről* beszélünk. Nilpotens operátorok tulajdonságai:

$c[x, n(x)] = 0$

$d[x, n(x)] = 1$

2.6.2. A fuzzy operátorok gyakorlati alkalmazása: összefüggéselemzés

A 3 legnevezetesebb operátor legegyszerűbb alakja:

$c(x,y)$	$d(x,y)$
$\min(x,y)$	$\max(x,y)$
$x \cdot y$	$x+y-x \cdot y$
$[x+y-1]$	$[x+y]$

A fenti három operátornak van egy közös tulajdonsága:

$$\begin{array}{ccccc} \min(x,y) & + & \max(x,y) & = & x+y \\ x \cdot y & + & x+y-x \cdot y & = & x+y \\ [x+y-1] & + & [x+y] & = & x+y \end{array}$$

Mindhárom esetben az összeg $x+y$ lesz. Vizsgáljuk meg, hogy csupán ezekre az operátorokra igaz ez a tulajdonság, vagy másokra is! A következő egyenletet kell megoldani, amely a **mértékazonosság** tulajdonság:

$$c(x,y)+d(x,y)=x+y \tag{2.11}$$

Tétel: A Frank-operátor szükséges és elegendő feltétele 2.11 teljesülésének, ahol c és d szigorúan monoton, asszociatív operátorok.
A De-Morgan azonosságokat felhasználva:

$$\left.\begin{array}{rcl} \overline{x \wedge y} & = & \overline{x} \vee \overline{y} \\ \overline{x \vee y} & = & \overline{x} \wedge \overline{y} \end{array}\right\} \quad \text{alapján} \quad 1-c(1-x,1-y)=d(x,y)$$

$c(x,y)+1-c(1-x,1-y)=x+y$ függvényegyenletet kell megoldani.

Tétel: Az asszociatív függvényegyenlet megoldása, amely eleget tesz 2.11-nek, a Frank-operátor.

A Frank-operátor képlete tartalmazza mindhárom operátort a következőképpen: az s paraméter 3 helyen nincs értelmezve (a logaritmus miatt) $s=0$, $s=1$, illetve $s=\infty$ helyeken. Ezeken a határértékeken éppen a 3 nevezetes operátort kapjuk:

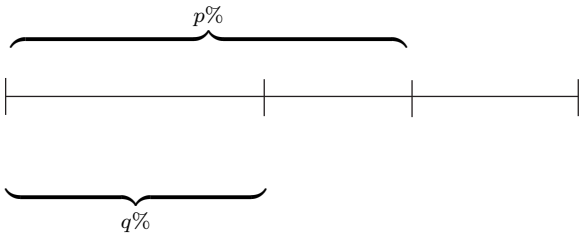
$s \to \infty$	$min(x,y)$
$s \to 1$	xy
$s \to 0$	$[x+y-1]$

Számos esetben szembekerülünk azzal a problémával, hogy kategóriákba sorolt dolgok közötti összefüggést kell vizsgálni. Például: a jó tanulmányi eredmény és a jólöltözöttség közti kapcsolat vizsgálata. Jó tanulónak minősul az a tanuló, akinek a tanulmányi átlaga 4,1 felett van, illetve jól öltözött az a tanuló, aki 7000 Ft-nál többet költ havonta ruházkodásra. A Frank-operátor segítségével az összefüggés mértékét

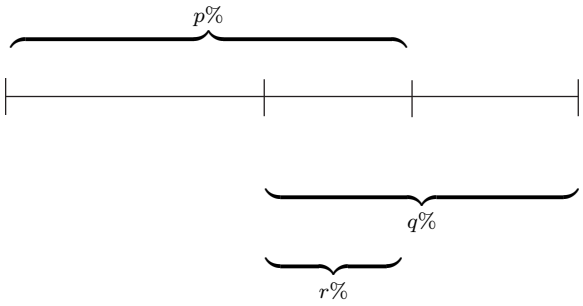
meg lehet határozni. Tegyük fel, hogy $q\%$ -a a tanulóknak jó tanuló, $p\%$-a a tanulóknak jól öltözött, és $r\%$ pedig azon tanulók aránya, akik mindkét tulajdonsággal bírnak.

Jó tanuló	Jól öltözött	Együtt
1	0	0
1	1	1
0	1	0
0	0	0
1	1	1
...	...	...
$p\%$	$q\%$	$r\%$

1. Ha a két tulajdonság nem zárja ki egymást. Azaz, ha az egyik tulajdonság megvan, akkor a másik is, akik mindkét tulajdonsággal bírnak: $r = \min(p, q)$ (min művelet).



2. Ha a két tulajdonság kizárja egymást: $r = p - (1 - q) = p + q - 1 \qquad ([x + y - 1])$



3. Ha a két tulajdonság egymástól statisztikailag független, akkor a szorzatuk adja az eredményt($x \cdot y$ művelet): $r = p \cdot q$

Transzformáljuk a Frank-operátor s paraméterét a [0,1] intervallumba! $\Rightarrow s = \frac{1}{1+s}$
Jelentse a transzformált értéket az s!

$$r = \log_s(1 + \frac{(1 - s^p)(1 - s^q)}{1 - s})$$

Vizsgáljuk meg, hogy milyen s-re lesz ez igaz!

$$\begin{aligned} s \to 0 &\Rightarrow \text{a két tulajdonság kizárja egymást} \\ s \to \frac{1}{2} &\Rightarrow \text{a két tulajdonság statisztikailag független} \\ s \to 1 &\Rightarrow \text{a két tulajdonság egymás következménye} \end{aligned}$$

A Frank-operátor gyakorlati alkalmazási területei:

- vezetői információs rendszerekben
- tanulóalgoritmusoknál a komponensek közötti összefüggések előzetes vizsgálatához
- adatbányászatban tudásbázis készítéshez, amely összefüggések keresésével kezdődik.

2.7. Módosító szavak modellezése

Tegyük fel, hogy például a *hosszú* poláris szó elé szeretnénk illeszteni a *nagy* illetve *kevésbé* módosító szavakat (angolul hedges=módosító szavak). Hogyan formalizálhatnánk az ezzel történő változásokat:

$$\text{nagyon hosszú: } x^2$$
$$\text{kevésbé hosszú: } \sqrt{x}$$

Ez a heurisztika nem túl működőképes. Modalitások alapján történő megközelítés: a mondatokra két másik szó, a *lehetséges* és a *szükségszerű* bevezetése ($\exists$, $\forall$), melyek megváltoztatják a logikai kiértékelést.

Diodoros-i paradigma (i.e. 4. század): 3 igaz állítás, melyek ellentmondanak egymásnak. Altrichter Ferenc rekonstruálta Észérvek című könyvében.

Konfirmáció paradoxona: $\forall$ ember halandó, $\forall$ holló fekete
Hogyan lehet ezeket igazolni? Ha találok egy pozitív példát, akkor a megerősítés növekszik. (Az élet sokkal bonyolultabb, mint a matematikai formulák.)

$$x \to y \equiv \bar{x} \vee y$$

$$\text{holló} \to \text{fekete} \equiv \text{nem holló} \vee \text{fekete}$$

Jelölések: SZÜKSÉGSZERŰ : $\Box$, LEHETSÉGES : $\Diamond$

$$\Box(nem(x)) = lehetetlen(x)$$
$$nem(\Diamond(x)) = lehetetlen(x)$$

$$\Box(\underbrace{n_{\nu_1}(x)}_{z}) = \underbrace{n_{\nu_2}(x)}_{szigorubb} \qquad \nu_1 > \nu_2$$

$$z = n_{\nu_1}(x) \rightarrow x = n_{\nu_1}^{-1}(z) \quad x = n_{\nu_1}(z)$$

$$\Box z = n_{\nu_2}(n_{\nu_1}(z))$$

A „szükségszerű” operációt így megkaphatjuk a két negáció egymásra alkalmazásával („nagyon”). Szükséges még:

$$\Box(n_{\nu_1}(x)) = n_{\nu_2}(\Diamond(x))$$

$$\Diamond(x) = n_{\nu_1}(\Box(n_{\nu_1}(x))$$

$$\Box x = n_{\nu_2}(n_{\nu_1}(x))$$

$$\Diamond x = n_{\nu_1}(n_{\nu_2}(n_{\nu_1}(n_{\nu_1}(x))))$$

$$\Diamond x = n_{\nu_1}(n_{\nu_2}(x))$$

Fordítva alkalmazva a két negációt a „lehetséges” operációt kapjuk meg („kevésbé”).

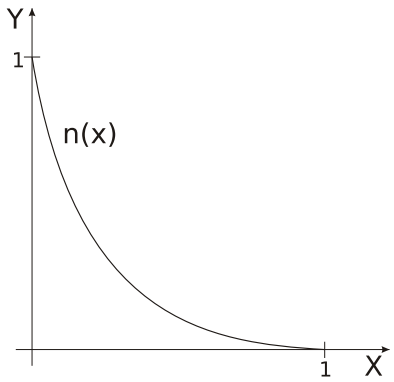
2.8. Fuzzy logikai műveletek

1. **Negáció:**

   A negációnak számos alakja van, például:

   - $\mu_{\neg A}(x) = 1 - \mu_A(x)$
   - $n(x) = \neg x = 1 - x$
   - $n(x) = (1 - x^n)^{\frac{1}{n}}$ (Yager)
   - $n(x) = \frac{1-x}{1+ax}$, ha $a > -1$ (Hamacher)

   A negációs operátor a 2.32. ábrán látható.



2.32. ábra. Negációs operátor

Tulajdonságai:

- folytonos leképezés a $[0,1]$ intervallumból a $[0,1]$ intervallumba
- szigorú monoton csökkenő
- kompatibililis a klasszikus logikával:

$$n(0) = 1 \qquad n(1) = 0$$

- involutív: $n(n(x)) = x$

Tétel: Minden negáció előállítható $n(x) = f^{-1}(1 - f(x))$ (*Trillas-képlet*) alakban, ahol $f(x)$ a negáció generátorfüggvénye és $f : [0,1] \to [0,1]$ szigorúan monoton, folytonos függvény, amelyre $f(0) = 0$ és $f(1) = 1$.

Példák:

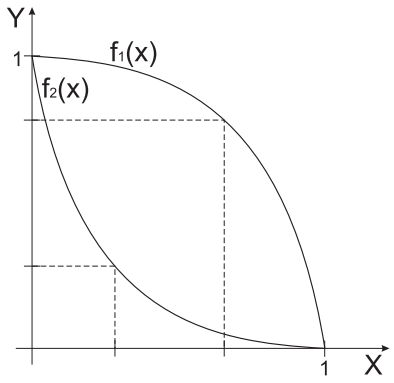
- Milyen alakú a negáció, ha az $f(x) = x^n$ a generátorfüggvénye?
  A Trillas-képlet alapján:
  $$n(x) = f^{-1}(1 - f(x)) = (1 - x^n)^{\frac{1}{n}}$$
- A negáció a Trillas-képlet alapján involutív.

$$x = n(n(x)) = f^{-1}(1 - f(n(x))) = f^{-1}(1 - (1 - f(x))) = f^{-1}(f(x)) = x$$

Szigorú a negáció, ha az $(1,0)$ ponthoz „gyorsan” közelít, illetve nem szigorú a negáció, ha a $(0,1)$ ponthoz „lassan”’ közelít (2.33. ábra).

Minden negációnak létezik fixpontja, azaz $\exists \nu : n(\nu_*) = \nu_*$. Döntéseknél a fixpont a neutrális érték (döntési szint). A fixpont az alábbi tulajdonságokkal rendelkezik:

$$\begin{aligned} \nu_* &= f^{-1}(1 - f(\nu_*)) \\ f(\nu_*) &= 1 - f(\nu_*) \\ 2f(\nu_*) &= 1 \\ \nu_* &= f^{-1}(\tfrac{1}{2}) \end{aligned}$$



2.33. ábra. Negáció szigorúságának szemléltetése.

2. **Konjunkció:**

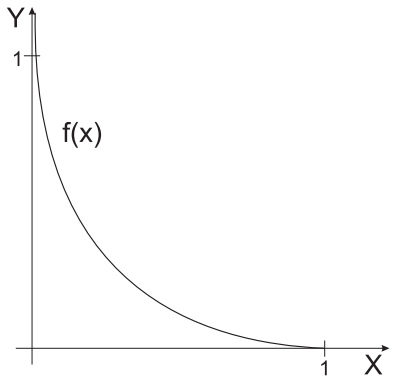
   Definiáló tulajdonságai:

   - monoton
   - folytonos a $[0, 1]$ intervallumon
   - kompatibilis a klasszikus (kétértékű) logikával:

$$\begin{aligned} c(0,0) &= 0 \\ c(0,1) &= c(1,0) = 0 \\ c(1,1) &= 1 \end{aligned}$$

   - asszociatív

A konjunkciós generátorfüggvény aszimptotikus az y-tengellyel és az 1-ben metszi az x-tengelyt, valamint konstans szorzó erejéig egyértelműen meghatározott (2.34. ábra).



2.34. ábra. Konjunkciós operátor generátorfüggvénye.

Példák:

- Adjuk meg az $f(x) = -\ln(x)$ generátorfüggvényhez tartozó konjunkciót!
  Az inverzfüggvény: $f^{-1}(x) = e^{-x}$

$$c(x,y) = e^{-(-\ln(x)-\ln(y))} = e^{\ln x + \ln y} = e^{\ln(xy)} = xy$$

– Adjuk meg az $f(x) = \frac{1}{1+x^{\frac{1}{\lambda}}}$ generátorfüggvényhez tartozó konjunkciót! Az inverzfüggvény: $f^{-1}(x) = (\frac{1}{x} - 1)^{\lambda}$
Ha $\lambda = 1$, akkor $c(x, y) = \frac{xy}{x+y-xy}$

3. **Diszjunkció:**
Definiáló tulajdonságai:

– monoton

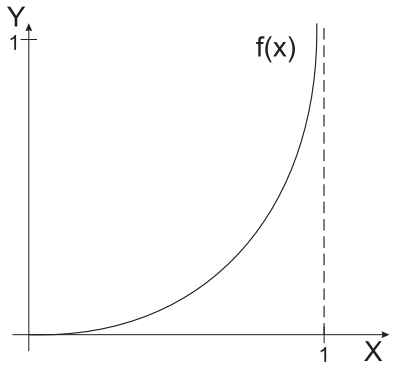
– folytonos a $[0, 1]$ intervallumon

– kompatibilis a klasszikus (kétértékű) logikával:

$$\begin{aligned} d(0,0) &= 0 \\ d(0,1) &= d(1,0) = 1 \\ d(1,1) &= 1 \end{aligned}$$

– asszociatív

A diszjunkció generátorfüggvénye : $x = 1$-ben aszimptotikus, $x = 0$-ban 0-t vesz fel, szigorúan monoton növő.



2.35. ábra. Diszjunkciós operátor generátorfüggvénye.

Példák:

– Mi a diszjunkció képlete, ha a generátorfüggvény $f(x) = -\ln(1 - x)$?
Az asszociatív függvényegyenlet alapján:

$$f_d^{-1}(x) = 1 - e^{-x}$$

$$d(x, y) = 1 - e^{-(-ln(1-x)-ln(1-y))} = 1 - e^{ln(-x-y+xy+1)} = x + y - xy$$

– Mi a diszjunkció képlete, ha a generátorfüggvény $f(x) = \frac{1}{1+x^{\frac{-1}{\lambda}}}$?
Az asszociatív függvényegyenlet alapján:

$$f_d(x) = \frac{1}{1 + x^{\frac{-1}{\lambda}}}$$

Az inverzfüggvény:

$$f_d^{-1}(x) = (\frac{1}{x} - 1)^{-\lambda}$$

Ha $\lambda = 1$, akkor $d(x,y) = \frac{x+y-2xy}{xy}$

- Bizonyítsuk be, hogy az alábbi műveletek asszociatívak!

$$x \vee (y \vee z) = d(x, d(y,z)) \Rightarrow f^{-1}(f(x) + f(d(y,z))) = f^{-1}(f(x) + f(y) + f(z))$$

4. **Implikáció:**

Ha az $\wedge$ és $\vee$ és $\neg$ műveletek adottak, akkor az implikáció nem más, mint:

$$i(x,y) = x \rightarrow y = \neg x \vee y$$

Tulajdonságai:

- Hamisság elve: i(0,x)=1
- Semlegesség elve: i(1,x)= x
- Igazság elve: i(x,1)= 1
- Negációval való kompatibilitás: i(x,0)=n(x)
- i(x,x)=1

Az azonosság elve (*identity principle*) összefoglalja az implikáció több tulajdonságát:

$$x \rightarrow y = 1 \Leftrightarrow x \leq y$$

Az implikáció képezhető diszjunkció és negáció segítségével, de az így kapott implikáció nem teljesíti az azonosság elvét.

$$\begin{aligned} \overline{x} \vee y = 1 \quad &\Leftrightarrow \quad x \leq y \\ \max(1-x, y) = 1 \quad &\not\Leftrightarrow \quad x \leq y \end{aligned}$$

A természetes kiterjesztés nem működik minden esetben: $(x \rightarrow y = \overline{x} \vee y)$

- min, max műveletekre:
  $max(1-x,y) = 1 \Leftrightarrow x \leq y$ és $x = y = \frac{1}{2}$ esetében nem jó.
- Szigorúan monoton osztályok esetén:
  $1 - x + y - (1-x) \cdot y = 1 \Leftrightarrow x \leq y$, ami nem igaz

Nilpotens műveletek esetén az implikáció teljesíti az azonosság elvét:

$$[1 - x + y] = 1 \Leftrightarrow x \leq y$$

A fentiekből jól látható, hogy új művelet bevezetésekor nem támaszkodhatunk az eddigi műveletekre.

Az implikáció bevezetése azért fontos, mert a logikai levezetések alapja, és ezáltal következtetéseket tudunk levonni. Erre jó a *modus ponens* szabály:

$$\left.\begin{array}{l} x \\ x \rightarrow y \\ \hline y \end{array}\right\} \Rightarrow x \wedge (x \rightarrow y) \neq y \quad \textit{helyett}: \quad (x \wedge (x \rightarrow y)) \rightarrow y = 1$$

Fuzzy operátorokhoz implikáció rendelése:

Induljunk ki a *modus ponens*-ből és az *identity principle*-ből!

$$(x \wedge (x \rightarrow y)) \rightarrow y \equiv 1 \Rightarrow x \wedge (x \rightarrow y) \leq y$$

Legyen $x \rightarrow y$, amit keresünk. Az $x = 0$ mindig megoldás. Az implikáció legyen a legnagyobb olyan érték, melyre $x \wedge y \leq y$. Az így kapott implikációt nevezzük *reziduális implikációnak.*

Az $\wedge$ művelet inverze (majdnem) az implikáció. Példa: „Nem zörög a haraszt, ha a szél nem fújja."

Ha $x \wedge z = y$:

$$\begin{array}{l} x + z = y \\ \quad \downarrow \\ y - x = z \quad \rightarrow \quad \textit{inverz} \end{array}$$

Ha $x \wedge z = y$:

$$\begin{array}{l} x \cdot z = y \\ \quad \downarrow \\ \frac{y}{x} = z \quad \rightarrow \quad \textit{inverz} \end{array}$$

Ahhoz, hogy az azonosság elve érvényes maradjon, a következő változtatást kell végrehajtanunk:

$$\begin{array}{lll} x \wedge \underbrace{(x \rightarrow y)}_{=z} & \leq & y \\ x \wedge z & \leq & y \\ & \downarrow & \\ & z = ? & \end{array}$$

Ha $z = 0$, akkor igaz. Növeljük z-t!

$\max_z(x \wedge z) \leq y \rightarrow$ **reziduális implikáció** (jele: $i(x, y)$, vagy $x \rightarrow y$)

$$\frac{\left.\begin{array}{r} x \\ x \rightarrow y \end{array}\right\}\text{meta „és"}}{y}$$

A min művelet reziduális implikációja:

$$i_{\min}(x,y) = \sup\{z | \min(x,z) \leq y\}$$

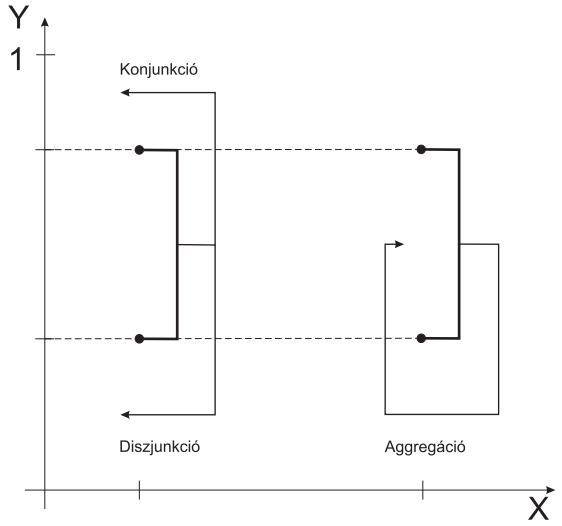
$$i_{\min}(x,y) = \begin{cases} 1, & \text{ha} \quad x \leq y \\ y, & \text{különben} \end{cases}$$

Az $[x+y-1]$ művelet reziduális és nem reziduális implikációja azonos (és teljesül az azonosság elve is).

5. **Aggregáció**

A kétértékű logika nem teszi lehetővé, hogy köztes értékeket kapjunk eredményül, pedig a mindennapi életben legtöbbször erre lenne szükség. Az ilyen művelet már nem kompatibilis a klasszikus logikával, egyetlen klasszikus logikai operátor sem képes arra, hogy a két argumentuma közé eső értéket adjon eredményül. Ezt a műveletet nevezzük aggregációnak (eredmény), jele: $a(x,y)$. Az aggregáció „sarkítása" a klasszikus logika.

$$\exists x, y \quad \min(x,y) \leq a(x,y) \leq max(x,y)$$



2.36. ábra. Aggregáció eredményének a konjunkció illetve a diszjunkció eredményéhez való viszonya.

A konjunkciót szokás még *triangluláris normának* (*t-norm*), a diszjunkciót *triangluláris konormának* (*t-conorm*), az aggregáció pedig az *uninormának* (*uninorm*) nevezni.

Tétel: Az összes konjunkciós operátorra teljesül, hogy $c(x,y) \leq \min(x,y)$ és az összes diszjunkciós operátorra teljesül, hogy $d(x,y) \geq \max(x,y)$.

Egy döntési szituációt a következőképpen írhatunk le. Legyenek adottak a következő jó tulajdonságok: $x_1, x_2, \ldots, x_n$, és $a(x_1, x_2, \ldots, x_n) = A$. Ekkor $A > \delta$ valamilyen δ-ra. Vegyük a $y_1, y_2, \ldots, y_n$ rossz tulajdonságokat, melyek az előbbi x_i tulajdonságok negáltjai ($y_i = n(x_i)$). Ezeket az $A^* = B$

tulajdonságban foglaljuk össze ($B = a(n(x_1), n(x_2) \dots n(x_n)) = (a(y_1, \dots y_n))$. A és B viszonya ekkor $B < n(\delta)$, tehát

$$\begin{array}{ccc} x_1, x_2, \cdots, x_n & \rightarrow & A > \delta \\ y_1, y_2, \cdots, y_n & \rightarrow & B < n(\delta) \\ \hline & y_i = n(x_i) & \end{array}$$

A fentiekből belátható, hogy:

$$a(x, y) = n(a(n(x), n(y)))$$

Az aggregáció elvárt tulajdonságai:

1. folytonos $[0, 1]$-en
2. szigorúan monoton
3. $a(0, 0) = 0$ illetve $a(1, 1) = 1$ (kompatibilitás szűkítése)
4. asszociatív
5. $\underline{a}(n(x), n(y)) = n(a(x, y))$ (ön-DeMorgan azonosság)

   Tétel: $a(n(x_1), n(x_2), \cdots, n(x_n)) = n(a(x_1, x_2, \cdots, x_n))$

   Az asszociativitás miatt: $a(x, y) = f^{-1}(f(x) + f(y))$
6. kommutativitás (neutrális érték meghatározása) $\rightarrow n(\nu_*) = \nu_*$

$$n(\underbrace{a(x, n(x))}_{=z}) = a(n(x), \underbrace{n(n(x))}_{=x}) = a(x, n(x))$$

$$n(z) = z$$

$$z = \nu_*$$

$$a(x, n(x)) = \nu_*$$

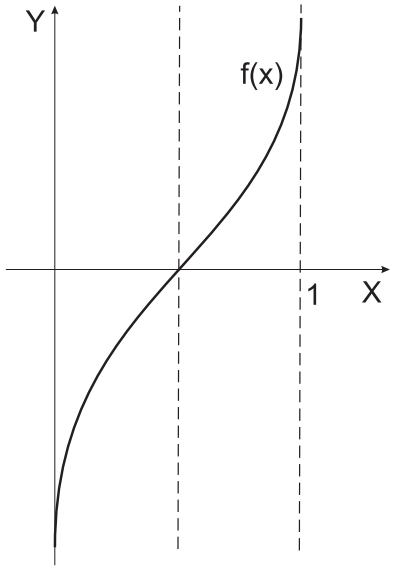
Az egyik legegyszerűbb aggregációs modell:

$$a(x_1, \dots x_n) = \frac{\prod x_i}{\prod x_i + \prod(1 - x_i)}$$

Tétel: $a(x, y) = f^{-1}(f(x) + f(y))$, ahol f az aggregáció generátorfüggvénye, amely konstans szorzó erejéig egyértelmű (2.37. ábra).

A neutrális érték hatása az aggregációra: az aggregációnál igen fontos szerepet tölt be a negáció művelet. A neutrális érték tulajdonságai:

- $\nu_* = n(\nu_*)$
- $n(a(x, n(x))) = a(n(x), n(n(x))) = a(n(x), x) \;\Rightarrow\; n(A) = A \Rightarrow a(x, n(x)) = \nu_*$
- $a(\nu_*, x) = x$



2.37. ábra. Aggregáció generátorfüggvénye.

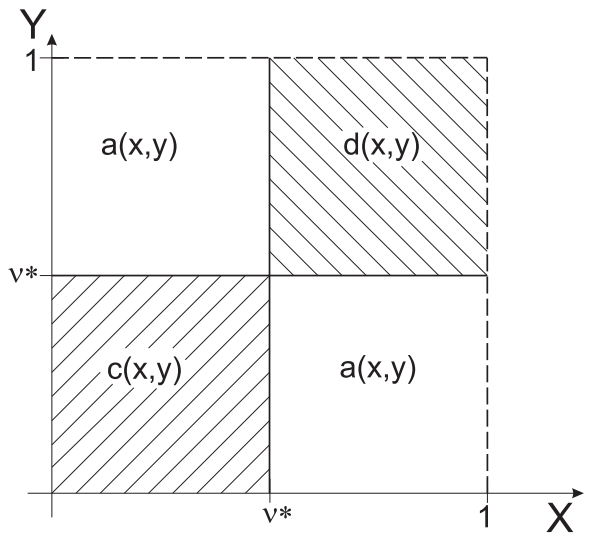
Tehát két rossz tulajdonság aggregációjából egy még rosszabb lesz, illetve két jó tulajdonság aggregációja egy még jobb lesz. Még több is igaz:

- Ha $x, y < \nu_*$, akkor $a(x, y) \leq \nu_*$, azaz $a(x, y) \leq min(x, y)$. Ha két rossz tulajdonságot aggregálok, akkor a rosszabbiknál is rosszabbat kapok.
- Ha $x, y > \nu_*$, akkor $a(x, y) \geq \nu_*$, azaz $a(x, y) \geq max(x, y)$ aggregált érték még a jobbiknál is jobb lesz.

$$a(x_1, x_2, \cdots, x_n) = \frac{\Pi x_i}{\Pi x_i + \Pi(1 - x_i)} \quad \Rightarrow \quad \nu = \frac{1}{2}$$

Az aggregáció értelmezése a többtényezős döntések szempontjából:

Az aggregáció magába foglalja a konjunkció és diszjunkció műveleteket $\rightarrow a(x, y) \leq \min(x, y)$ illetve $a(x, y) \geq \max(x, y)$ (lásd 2.38. ábra).



2.38. ábra. Aggregáció értelmezése a többtényezős döntések szempontjából.

$$\text{Ha } \nu_* \sim 1, \text{ akkor } a(x,y) = c(x,y).$$
$$\text{Ha } \nu_* \sim 0, \text{ akkor } a(x,y) = d(x,y).$$
$$\text{Ha } \nu_* \in (0,1), \text{ akkor aggregálunk.}$$

Uninorma abban az értelemben is, hogy konjunkció és diszjunkció is egyben. A ν_* egy elvárási érték, amihez viszonyítunk, ami alapján döntünk. A klasszikus logikában a ν_* csak 0 vagy 1 lehet. A ν_* alapján hozunk döntéseket:

Példa az aggregáció értelmezésére:

- Konjunkciós döntés ($\nu_* = 1 \rightarrow$ minden alternatíva rossz): házimozirendszer vásárlása. Ebben az esetben megpróbáljuk meghatározni a legideálisabb alternatívát, ez tulajdonképpen annak a megközelítésnek a megfelelője, miszerint minden alternatíva valamilyen mértékig rossz, és lehető legkevésbé rosszat szeretnénk kiválasztani. Pl. a Sony ugyan többe kerül, mint a Panasonic, de a Sonynak szebb a képe, így a Sonyt választjuk. (Mindkét alternatívának vannak hátrányai, de a számunkra lehető legkevésbé rosszat választjuk.)
- Diszjunkciós döntés ($\nu_* = 0 \rightarrow$ minden alternatíva jó): születésnapi ajándék. („Ajándék lónak ne nézd a fogát!") Itt minden alternatívát elfogadunk, legyen az bármilyen rossz, minden ajándéknak örülni kell, mint ahogy egy versnek csak jó tulajdonságai vannak esztétikai szempontból.

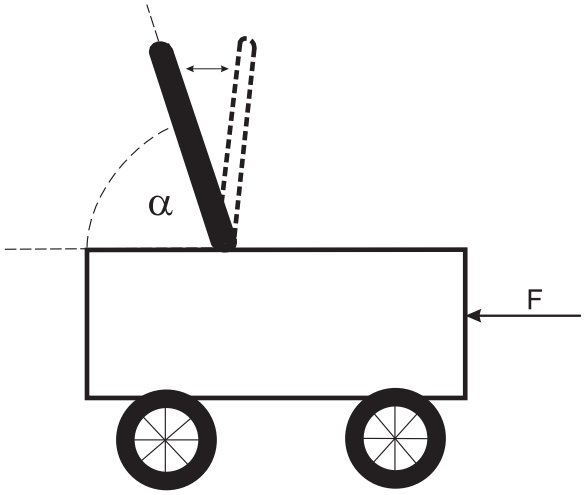
Amikor az alternatívák tulajdonságain egyfajta elvárást határozunk meg, akkor konjunkciós döntést hozunk ($\nu_* = 1$), és amikor esztétikai értékítéletet kell mondanunk, az mindig diszjunkciós döntés ($\nu_* = 0$), de a gyakorlatban és a mindennapokban aggregációt végzünk ($\nu_* \in (0,1)$).

2.9. Fuzzy irányítás

Alapvetően azzal foglalkozunk, hogy hogyan lehet egy olyan műveletet „megtanítani" egy számítógépnek (robotnak), mint például az autóvezetés, vagy a biciklizés, ahol az adott szituációban döntések sorozatát kell meghozni a világ reakcióitól függően. A fuzzy irányításban (fuzzy control) nem csak a fizikai tulajdonságokat, hanem a szabályokat is figyelembe kell venni (pl. KRESZ).

A fizika ezeket a helyzeteket csak bonyolult differenciálegyenletekkel tudja leírni. Az egyenletek megoldásához természetesen szükség van a paraméterek ismeretére. Egy valós szituáció leírásához azonban nagyon sok paraméterre lenne szükség, illetve a probléma megoldása igen sok időt venne igénybe, ami például biciklizés közben gondot jelent. Nem lehet percekig várni, hogy merre és milyen szögben fordítsuk el a kormányt. A paramétereket a legtöbb esetben egyébként sem tudjuk mérni, ezért becsülni kell. Ezért sem célravezető differenciálegyenletek segítségével keresni a megoldásokat. Más megoldást kínál a fuzzy vezérlés.

Ha adott egy kiskocsi, melyet úgy szeretnénk mozgatni, hogy a kocsin lévő bot ne dőljön el, akkor a feladatot az alábbiak szerint modellezhetjük. Tegyük fel, hogy α ismert, $\dot{\alpha}$ az α idő szerinti deriváltja és azt az F erőt kell meghatározni, amivel a kocsit mozgatjuk (2.39. ábra).



2.39. ábra. A példát szemléltető kiskocsi

Az alábbi a szabályrendszer:

1. $if \quad \underbrace{\alpha \in A_1 \wedge \dot{\alpha} \in B_1}_{\mathcal{L}_1} \quad \text{then } F_1 \quad (= 5N)$
2. $if \quad \underbrace{\alpha \in A_2 \wedge \dot{\alpha} \in B_2}_{\mathcal{L}_2} \quad \text{then } F_2 \quad (= -1N)$
3. $\vdots$

A környezetből különböző értékeket kapunk, melyekre reagálni kell: pl. $\alpha_x = 0,3$ és $\dot{\alpha}_x = 0,2$. A fenti szabályrendszer áll a rendelkezésünkre, és ha α_x illetve $\dot{\alpha}_x$ értéke szerepel a szabályrendszerben, akkor azonnal meg lehet határozni, hogy mit kell tenni, viszont ha nincs a szabályrendszerben ez az eset, akkor valamilyen algoritmussal kell meghatározni azt. Ez a feladat igen összetett. A súlyozott összeg szabályt alkalmazva adjuk meg, hogy egy ilyen érték esetén hogyan kell reagálni:

$$F_x^* = \frac{\sum \mathcal{L}_i(\alpha_x) F_i}{\sum \mathcal{L}_i(\alpha_x)}$$

Elérhető, hogy példák alapján tanulja meg a rendszer, hogy mit kell tenni a különböző helyzetekben. Olyan szabályrendszert keresünk, melynek a reakciói a lehető legközelebb állnak az elvárt reakciókhoz. Olyan F^*-ot kell keresni, ahol a négyzetes hiba minimális, azaz

$$\sum \left(\underline{F_i} - F_i^*\right)^2 \to min$$

Felül kell viszgálni ezen kívül magát a szabályrendszert is, hogy mennyire helyes:

1. $\mathcal{L}_1(\alpha_x) = 0,2 \qquad A_1 = 5N$
2. $\mathcal{L}_2(\alpha_x) = 0,9 \qquad A_2 = -1N$
3. $\mathcal{L}_3(\alpha_x) = 0,3 \qquad A_3 = -30N$

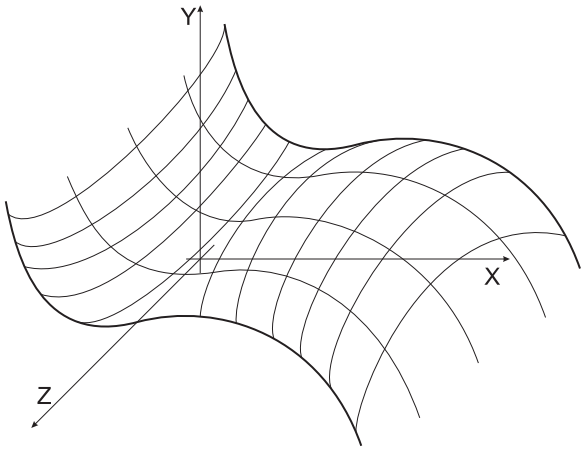
$$\frac{0.2 \cdot 5 + 0.9 \cdot (-1) + 0.3 \cdot (-30)}{0.2 + 0.9 + 0.3} \approx 34,286$$

A fentiek kifejezések kombinációiból kell előállítani a megfelelő eredményt.
A fuzzy vezérlést a fentiek alapján következőképp modellezzük:

$$\begin{array}{lllll} if & x \in A_1 & \wedge & y \in B_1 & then \quad a_1 \\ if & x \in A_2 & \wedge & y \in B_2 & then \quad a_2 \\ & & \ldots & & \end{array}$$

Itt a jobb oldalon álló a_i-k konkrét értékek. A bal oldalt definiáljuk halmazhoztartozási függvényekkel, így az if utáni részt egy vektorral reprezentáljuk $\rightarrow \alpha(\underline{x})$, ami egy fuzzy logikai kifejezés.

Amennyiben 2 változónk van, úgy 2 dimenziós a keresési tér, azonban ha több változónk van, úgy több dimenzióban kell keresnünk. Így tehát a fuzzy irányítás nem más, mint többdimenziós felületek konstruálása (2.40. ábra).



2.40. ábra. Példa 3D-s felület konstruálása a fuzzy irányításhoz.

A kérdés, hogy hogyan konstruáljuk meg az adott felületet.

$$\begin{array}{llllll|lll|lll|l} if & x \in A_1 \wedge y \in B_1 & then & a_1 \rightarrow & L_1 & = & \mathcal{L}_1(x^*, y^*) & = & 0,3 \\ if & x \in A_2 \wedge y \in B_2 & then & a_2 \rightarrow & L_2 & = & \mathcal{L}_2(x^*, y^*) & = & 0,9 \\ & \vdots & & & \vdots & & \vdots & & \vdots \\ if & x \in A_n \wedge y \in B_n & then & a_n \rightarrow & L_n & = & \mathcal{L}_n(x^*, y^*) & = & 0,01 \end{array}$$

A fuzzy logikai formula kiértékelése elvégezhető operátorokkal, csupán a kifejezés implementálásához kell választani egy operátort. A kimenő érték a következő lesz:

$$\frac{\sum L_i \cdot a_i}{\sum L_i} = a$$

Tegyük fel, hogy adott egy n dimenziós tér, vagyis n különböző dolgot mérünk. Ha minden tartományt csupán 2 részre (jó és rossz) osztunk fel, akkor is 2^n darab szabályunk lesz, vagyis a fuzzy szabályrendszer mérete exponenciálisan nő a változók számában. Ha gyors megoldásra van szükség, akkor lineáris módon kell növelni a szabályrendszert, és az eljárás megkonstruálja a felületet.

Felmerül a probléma, hogy milyen halmazhoztartozási függvényt válasszunk. Mint már a korábbiakban láthattuk, 3 fő irány létezik ezeknél a függvényeknél: trianguláris, harang görbe, illetve szigmoid függvény. A halmazhoztartozási függvényeket 90%-ban önkényesen adják meg.

A fuzzy irányításnak két jelentős modellje van: a Takagi-Sugeno illetve a Mamdani-modell.

2.9.1. Takagi-Sugeno-modell

$$\begin{aligned} a_1 &= w_1^{(1)} x + w_2^{(1)} y + c^{(1)} \\ a_2 &= w_1^{(2)} x + w_2^{(2)} y + c^{(2)} \\ &\vdots \\ a_n &= w_1^{(n)} x + w_2^{(n)} y + c^{(n)} \end{aligned}$$

Itt a w_i és c értékeket kell megtanulni. A Takagi-Sugeno modellben a konstansok helyére számokat írva lineáris kifejezéseket kapunk, és megnézzük azt, hogy egyes esetekben milyen érték lenne helyes. Ezeket a szituációkat standard tanuló egyenleteknek tekintve egy többváltozós, nemlineáris, minimalizációs eljárással megkereshetők a paraméterek. Ezen lineáris együtthatók meghatározása:

$$F(x^*) = \left(F_{x_j^*}(w_0, w_1, c) - A\left(x_j^*\right)\right)^2$$

$$\sum \left(F_{x_j^*}(w_0, w_1, c) - A\left(x_j^*\right)\right)^2 \rightarrow \min_{a_0, a_1, b_1},$$

ahol $A\left(x_j^*\right) \rightarrow$ az elvárt érték

Ha kvadratikus kifejezéssel próbálnánk megadni az együtthatókat az alábbi egyenlethez jutnánk:

$$z = a_1 + a_2 x + a_3 y + a_4 x^2 + a_5 y^2$$

Így csupán a paraméterek számát növeltük, de a paraméterekben való linearitás megmaradt, ugyanis ha egy adott x és y értéket behelyettesítünk, akkor az a értékek lineáris kifejezését kapjuk. Csak akkor érdemes kvadratikus kifejezéseket alkalmazni, ha ez jelentős pontosságot eredményez.

Ezt a modellt igen sokszor használják a gyakorlatban egyszerűsége miatt, pl. fényképezőgépek, metro vezérlés, mosógépek (Dombi-operátor segítségével működnek Japánban), stb.
Megjegyzés: a Dombi-operátor felhasználása a fuzzy irányításban:

$$\left(\frac{1}{1+\sum\left(\frac{1-\mu_i}{\mu_i}\right)^{\alpha}}\right)^{\frac{1}{\alpha}},$$

ahol ha $\alpha < 0$, akkor konjunkció, ha $\alpha > 0$, akkor diszjunkció, illetve a μ_i egy halmazhoztartozási függvény és szigmoidnak kell választani:

$$\mu_i = \frac{1}{1+e^{-\lambda_i(x_i-a_i)}} \quad \rightarrow \quad \frac{1-x_i}{x_i} = \frac{1-\frac{1}{1+e^{-\lambda_i(x_i-a_i)}}}{\frac{1}{1+e^{-\lambda_i(x_i-a_i)}}} = e^{-\lambda_i(x_i-a_i)}$$

Ezt visszahelyettesítve a képletbe:

$$\left(\frac{1}{1+\sum e^{-\lambda_i'(x_i-a_i)}}\right)^{\frac{1}{\alpha}},$$

ahol $\lambda_i' = \alpha\lambda_i$. Ez a képlet könnyen kezelhető, ezért igen jelentős a fuzzy irányításban.
Japánban a növekvő népességgel járó óriási áramfogyasztásra nem felkészült, túl kis teljesítményű infrastruktúra miatt csak 30°-on szabad mosni, mivel a 90°-os mosással járó áramfogyasztást nem bírná el a rendszer. Ezért volt szükség olyan mosógépekre, melyek a 90°-os mosásnak megfelelő tisztítást biztosítják 30°-on.

2.9.2. Mamdani-modell

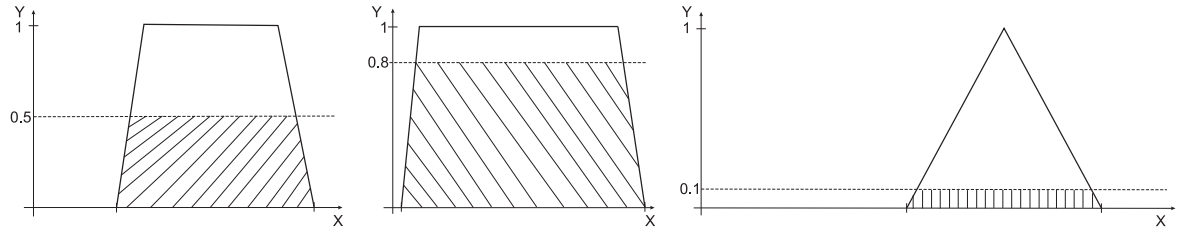
Az alapgondolata az, hogy ha a bal oldalt fuzzyfikáljuk, tegyük ezt a jobb oldallal is.

$$\text{if } x \in A \ \wedge \ y \in B \text{ then } z \in C$$

Vegyük végig az eljárás menetét 3 szabály esetén!

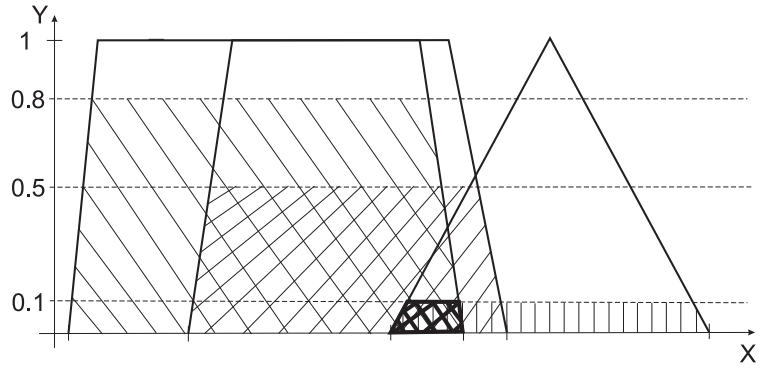
szabály		optimum
1. szabály	$\rightarrow$	$\mathcal{L}_1(\underline{x^*}) = 0,5$
2. szabály	$\rightarrow$	$\mathcal{L}_2(\underline{x^*}) = 0,8$
3. szabály	$\rightarrow$	$\mathcal{L}_3(\underline{x^*}) = 0,01$

Most az eredmény halmazhoztartozási függvényeken α_i-vágásokat hajtunk végre az $\mathcal{L}_i(x^*)$ értékekkel, a 2.41. ábrán látható módon.



2.41. ábra. α_i-vágások végrehajtása a szabályok halmazhoztartozási függvényén.

A fent kapott függvényeket egy koordináta–rendszerben ábrázoljuk, és megkeressük ezek metszetének *gravitációs középpontját* (súlypontját – lásd 2.42. ábra). Ez a művelt defuzzyfikálás abban az értelemben, hogy a fuzzy halmazokból egyetlen értéket generálunk.



2.42. ábra. A szabályok lemetszett halmazhoztartozási függvényének metszete – gravitációs középpont.

A Mamdani-modell lépései tehát:

1. lépés: fuzzyfikálás

2. lépés: eljárás végrehajtása

3. lépés: defuzzyfikálás

Ha abból a megközelítésből indulunk ki, miszerint a fuzzy irányítás nem más, mint többdimenziós felületek konstruálása, úgy a probléma a következőképpen fogalmazható meg:

- ha 1 dimenziós felület → függvénykonstrukció

- ha 2 dimenziós felület → felületkonstrukció

- ha több dimenziós felület → hiperfelületek konstrukciója

Matematikailag ez felületek approximációját (közelítését) jelenti. Rengeteg approximációs eljárás létezik (pl. regresszió, az interpoláció ugyan közelítés, de nem approximáció).
Megjegyzés: létezik fuzzy interpoláció is, ezzel a témakörrel foglalkozik például Kóczy László (könyvet is jelentetett meg a fuzzy irányításról).

2.10. Fuzzy klaszterezés

A klaszterező algoritmusok egymástól elkülönülő csoportosulásokat, „sűrűbb" részeket hivatottak felismerni egy címkézetlen adathalmazon belül. A klaszterezés és az alakfelismerés bizonyos mértékben összekapcsolódó fogalmak. Egy klaszter ugyanis tetszőleges alakú lehet: kör, egyenes, ellipszis, stb.. Az a kérdés, hogy képes-e az algoritmusunk felismerni az adott alakzatba rendeződő csoportosulását az elemeknek, kapcsolatot teremt a két problémakör között. A klaszterező algoritmusok teljesítményét a klaszterek alakja mellett erősen befolyásolja a klaszterek egymástól mért távolsága, térbeli elhelyezkedése.

A klaszterezés más megfogalmazásban egy halmaz partícionálását, felosztását jelenti olyan részhalmazokra, ahol az egy részhalmazban lévő elemek hasonlítanak egymásra, míg különböző részhalmazban lévő elemek nem hasonlítanak egymásra. Ennek végrehajtásához szükség van egy hasonlósági mérték bevezetésére. Ez a hasonlósági mérték leggyakrabban az elemek egymástól mért térbeli távolsága.

A klasszikus (nem fuzzy) klaszterezésben egy X halmaz klaszterezésével létrejön X-nek c darab páronként diszjunkt részhalmaza, ahol c a klaszterek száma. Minden egyes X-beli elem pontosan egy klaszterben lesz benne.

Fuzzy klaszterezés során ellenben az X halmaz c darab fuzzy részhalmazra kerül felbontásra, megengedve, hogy egy elem több klaszterhez is tartozzon, különböző hozzátartozási mértékkel.

A klaszterezés célja, hogy egy X halmazt c klaszterre bontson. Egyelőre feltételezzük, hogy c ismert. A $c \times N$ mátrix $\mathbf{U} = [\mu_{i,k}]$ reprezentálja a fuzzy partícionálását (klaszterre bontását) X-nek, ahol $\mu_{i,k}$ jelöli az $\underline{x}_k$ elem hozzátartozási mértékét az i. klaszterhez. Így az $\mathbf{U}$ i. sora tartalmazza az X i. fuzzy részhalmazához tartozó hozzátartozási értékeket, míg az $\mathbf{U}$ k. oszlopa az $\underline{x}_k \in X$ elem hozzátartozási értékeit az egyes fuzzy klaszterekhez. A fuzzy partíciónak teljesítenie kell az alábbi feltételeket:

$$\mu_{i,k} \in [0,1], \quad 1 \leq i \leq c, \; 1 \leq k \leq N, \tag{2.12}$$

$$\sum_{i=1}^{c} \mu_{i,k} = 1, \quad 1 \leq k \leq N, \tag{2.13}$$

$$0 < \sum_{k=1}^{N} \mu_{i,k} < N, \quad 1 \leq i \leq c. \tag{2.14}$$

Az első feltétel jelentése az, hogy minden egyes hozzátartozási értéke egy $[0,1]$ zárt intervallumbeli értéket vehet fel. A második feltétel jelentése az, hogy minden egyes elem klaszterekhez tartozásának összege 1 lesz. Ez azt jelenti, hogy ha egy elem nagy, például $0,9$-es halmazhoz tartozási értékkel tartozik egy klaszterhez, akkor a többi klaszterekhez összesen csak $0,1$-es halmazhoztartozási értékkel fog tartozni. Ennek segítségével nem jönnek létre elfajuló megoldások, és a halmazhoz tartozási mértékek egymással összehasonlíthatóak lesznek. A harmadik feltétel azt jelenti, hogy minden részhalmazhoz tartoznak valamilyen mértékben elemek, azaz nem megengedett az üres halmaz.

2.10.1. A Fuzzy C Means klaszterező eljárás

A klaszterező eljárás tekinthető egy optimalizációs problémának is. Definiálható egy célfüggvény, mely eleget tesz a 2.12-2.14 feltételeknek, és optimumának megtalálása a klaszterezés feladatának kielégítő megoldását is jelenti. Az egyik leggyakrabban használt klaszterező eljárás a Fuzzy C Means (FCM), mely az FCM célfüggvényére épül. Ez a célfüggvény a következő:

$$J(\mathbf{X}, \mathbf{U}, \mathbf{V}) = \sum_{i=1}^{c} \sum_{k=1}^{N} (\mu_{i,k})^m ||\underline{x_k} - \underline{v_i}||_A^2, \tag{2.15}$$

ahol

$$\mathbf{U} = [\mu_{i,k}] \tag{2.16}$$

X egy fuzzy partíciója,

$$D_{i,kA}^2 = ||\underline{x}_k - \underline{v}_i||_A^2 = (\underline{x}_k - \underline{v}_i)^T A (\underline{x}_k - \underline{v}_i) \tag{2.17}$$

egy alkalmazott távolság norma, ahol A a távolság mérték és

$$V = [\underline{v}_1, \underline{v}_2, \ldots, \underline{v}_c], \quad v_i \in \Re^n \tag{2.18}$$

a klaszter prototípusokból (középpontokból) álló mátrix, amelyek helyzetét ki kell számítani. Az elemek klaszter középpontoktól való távolsága fejezi ki a hasonlóságot, illetve a különbözőséget az elem és az egyes klaszter középpontok között. Minél távolabb van a térben egy elem egy klaszter középponttól, annál kevésbé hasonlít rá. Távolság normaként leggyarabban az euklideszi normát használják, ahol $A = I$.

A távolság, mint hasonlósági mérték súlyozásra kerül a halmazhoz tartozási mértékkel, ami azt jelenti, hogy a célfüggvény értékében azok az elem-középpont távolságok nyomnak nagyobb súllyal a latba, amelyeknél a klaszter középpont által meghatározott klaszterhez nagy mértékben tartozik az elem. Fordítva, a célfüggvény értékét nem rontja le, ha egy elem nagy távolságban helyezkedik el egy olyan klaszter középponttól, amely az általa meghatározott klaszterhez csak kis mértékben tartozik. A célfüggvény az $x_k,\ 1 \leq k \leq N$ elemek és a $v_i,\ 1 \leq i \leq c$ klaszter középpontok távolságának (vagy másképp fogalmazva: különbözőségük mértékének) a hozzátartozási mérték m-dik hatványával súlyozott összege. Az $m \in (1, \infty)$ paraméter egy súly kitevő, mely meghatározza az eredmény „fuzzyságát".

Az FCM algoritmus

Az FCM célfüggvényének minimalizálása egy nem lineáris optimalizációs probléma.

Adott X adathalmaz, a klaszterek c száma, egy m súly exponens, ϵ megállási feltétel vagy hibaküszöb, és a távolság normát meghatározó A mátrix. Inicializáljuk $\mathbf{U}^{(0)}$ 2.12- 2.14-nak megfelelően.

Ismételjük $l = 1, 2, . \cdots - ra$

1. Számítsuk ki a klaszter középpontok helyét:

$$v_i^{(l)} = \frac{\sum_{k=1}^{N} \left(\mu_{i,k}^{(l-1)}\right)^m x_k}{\sum_{k=1}^{N} \left(\mu_{i,k}^{(l-1)}\right)^m}, \quad 1 \leq i \leq c \tag{2.19}$$

2. Számítsuk ki az új távolságokat:

$$D^2_{i,kA} = (\underline{x}_k - \underline{v}_i)^T A(\underline{x}_k - \underline{v}_i), \quad 1 \leq i \leq c, 1 \leq k \leq N \tag{2.20}$$

3. Számítsuk ki a partíciós mátrixot:

$$\mu^{(l)}_{i,k} = \frac{1}{\sum_{j=1}^{c} \left(\frac{D^2_{i,kA}}{D^2_{j,kA}}\right)^{\frac{1}{m-1}}}, \quad 1 \leq i \leq c, 1 \leq k \leq N, \tag{2.21}$$

amíg $||\mathbf{U}^{(l)} - \mathbf{U}^{(l-1)}|| < \epsilon$

2.10.2. Az FCM tulajdonságai

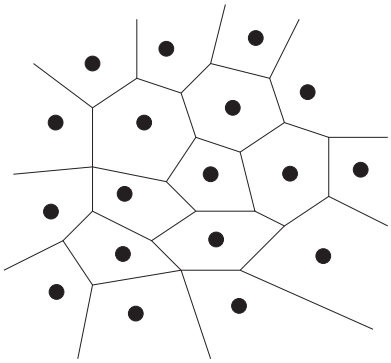
Az FCM algoritmus futási ideje $O(n)$. A felismerhető klaszteralakzat a használt A távolsági mértéktől függ. Az egységmátrixot felhasználva (amely euklideszi távolság normát eredményez) hipergömb alakú klaszterek felismerésére lesz képes az algoritmus. A klaszterek gyakran nem hipergömb alakúak, ezért ez a választás korlátozhatja az algoritmus képességeit. Emiatt használhatunk más távolsági normákat is, mint például a Mahalanobis normát, amelyekkel ellipszis alakú klaszterek is felismerhetők. Azonban általánosan igaz az FCM algoritmusra, hogy a távolsági norma és így a felismerhető klaszterek alakja előre rögzített.

Vannak olyan klaszterező eljárások, amelyek nem rendelkeznek a fenti korlátozással. Ilyen algoritmus a Gustaffson-Kessel, mely a Fuzzy C Means kibővítése, és képes adaptívan változtatni a használt távolság normát, és így előzetes beállítás nélkül felismerni a különböző elliptikus alakú klasztereket. Ennek az agloritmusnak a hátránya, hogy csak előre rögzített méretű klasztereket képes felismerni. Ez az adaptív távolság norma alkalmazásának a következménye. A Gath-Geva klaszterező algoritmus azonban képes mind alakban, mind méretben különböző klaszterek automatikus felismerésére. Ennek az algoritmusnak a hátránya, hogy nagy a számítás igénye, és olyan lineáris algebrabeli műveleteket alkalmaz, melyek érzékennyé teszik, így finomhangolás nélkül nem képes eredményt produkálni bizonyos adathalmazok esetén.

2.10.3. A klaszterezés eredményének megállapítása

Miután a klaszterező algoritmus futása véget ér, és létrejön a végleges fuzzy partíció, szükség lehet arra, hogy megállapítsuk, hogy az egyes elemek pontosan melyik részhalmazához tartoznak X-nek. Ekkor a klaszterezés eredményeként létrejövő fuzzy partíciót defuzzyfikálni kell. A legáltalánosabban használt technika a *voronoi diagram*. Ekkor elméletben megszerkesztésre kerülnek a klaszterközéppontokhoz tartozó Voronoi cellák.

A 2.43. ábrán látható egy Voronoi diagram és benne az egyes középpontokhoz tartozó voronoi cellák. A fuzzy partícióból pontosan azok az elemek kerülnek azonos halmazba, melyek ugyanabban a Voronoi cellában helyezkednek el. Egy elem adott Voronoi cellába tartozása ekvivalens azzal, hogy az elemhez az adott középpont helyezkedik el legközelebb az összes középpont közül. Emiatt gyakorlatban nem kerül megszerkesztésre a tényleges Voronoi diagram, hanem helyette megkeresésre kerül a legközelebbi klaszter középpont, és az adatelem annak klaszterébe lesz besorolva.



2.43. ábra. Voronoi diagram

2.10.4. Validitási mértékek

Eddigiekben végig feltételeztük, hogy a klaszterek száma ismert. Azonban gyakran fordul elő a gyakorlatban, hogy egy adathalmazról nem tudjuk, hogy tartalmaz-e klasztereket, illetve ha igen, akkor hányat. Ezt a problémát úgy szokták kiküszöbölni, hogy lefuttatják a klaszterezést különböző klaszterszámokra és az algoritmus eredményére kiszámítanak különböző mutatókat, amik jellemezhetik a létrejövő partícionálás minőségét. A létrejövő eredmények közül ezután kiválasztják a legjobb mutatóval rendelkező partíciót és ez lesz a klaszterezés végleges eredménye. Néhány gyakran használt validitási mérték következik:

1. Partition Coefficient (PC)

   A PC validitási mérték megméri, hogy mennyire átfedőek a klaszterek.

$$PC(c) = \frac{1}{N}\sum_{i=1}^{c}\sum_{k=1}^{N}(\mu_{i,k})^2 \tag{2.22}$$

2. Classification Entropy (CE)

$$CE(c) = -\frac{1}{N}\sum_{i=1}^{c}\sum_{k=1}^{N}\mu_{i,k}ln(\mu_{i,k}) \tag{2.23}$$

3. Partition Index (SC)

   Az SC az egyik leggyakrabban használt index.

$$SC(c) = \sum_{i=1}^{c}\frac{\sum_{k=1}^{N}(\mu_{i,k})^m\|x_k - v_i\|^2}{\sum_{k=1}^{N}\mu_{i,k}\sum_{j=1}^{c}\|v_j - v_i|^2} \tag{2.24}$$

4. Separation Index (S)

   A klaszterek elválasztottságának mértékét ebben az esetben a két legközelebbi klaszter-középpont távolsága adja meg. A S egy másik gyakran használt index.

$$S(c) = \frac{\sum_{i=1}^{c}\sum_{k=1}^{N}(\mu_{i,k})^2\|x_k - v_i\|2}{Nmin_{i,j}\|v_j - v_i\|^2} \tag{2.25}$$

5. Dunn's Index (DI)

   Ennek a validitási mértéknek a hiányossága a magas számítási igénye.

$$DI(c) = \min_{i\in c}\{\min_{j\in c, i\neq j}\{\frac{min_{x\in C_i, y\in C_j}d(x,y)}{max_{k\in c}\{max_{x,y\in C}d(x,y)\}}\}\} \tag{2.26}$$

6. Fuzzy hyper volume (hipertérfogat)

   Ez a mérték a klaszter „térfogatát” reprezentálja.

$$\nu = \sum_{i=1}^{c} det(\mathbf{F}_i) \tag{2.27}$$

2.11. K–means klaszterező algoritmus

A napjainkban nagyon népszerű K–means (K–közép) klaszterező algoritmus mondhatni az „egyszerűen nagyszerű” megoldások közé tartozik, mégis van néhány alapvető hiányossága, melyek a módszer továbbfejlesztésére késztetik a téma iránt érdeklődőket. Először, az algoritmus lassú és rosszul skálázható. Másodszor, a K értéket, vagyis a klaszterek számát a felhasználónak kell megadnia, ami negatívan befolyásolhatja a kapott eredményt. Harmadszor, a tapasztalatok azt mutatják, hogy ha a K értéke előre rögzített, az algoritmus rosszabb lokális optimumot talál, mint akkor, ha a K érték dinamikusan változik.

Ezen problémák kiküszöbölése érdekében Dan Pelleg és Andrew Moore több módosításra is javaslatot tett. Ilyen például a kd–fa (kd–tree) adatszerkezet alkalmazása, mely az eredeti algoritmus működésén nem változtat, viszont több szempontból is hatékonyabbá teszi az eljárást. Egy másik fontos bővítés a K érték dinamikus maghatározásával való kiegészítés, ami az X–means algoritmus nevet kapta.

2.11.1. K–means

Adott egy adathalmaz R darab rekordja, egyenként M darab attribútummal, valamint egy K konstans. A klaszterezési probléma az adatok K darab részhalmazba sorolása úgy, hogy minden részhalmaz „jól viselkedjen” bizonyos mérték szerint.

Az egyszerű K–means algoritmus K darab véletlenszerűen választott középponttal indulva iteratívan osztja fel az adathalmaz pontjait a hozzájuk legközelebb eső középpont alapján. Vagyis minden pont tartozni fog valamely középponthoz, és az azonos középponthoz tartozó pontok alkotják a klasztereket.

Legyen $C^{(i)}$ az i-edik iterációban lévő középpontok halmaza. Amíg $C^{(i)}$ és $C^{(i+1)}$ nem egyenlő, ismételjük a következőket:

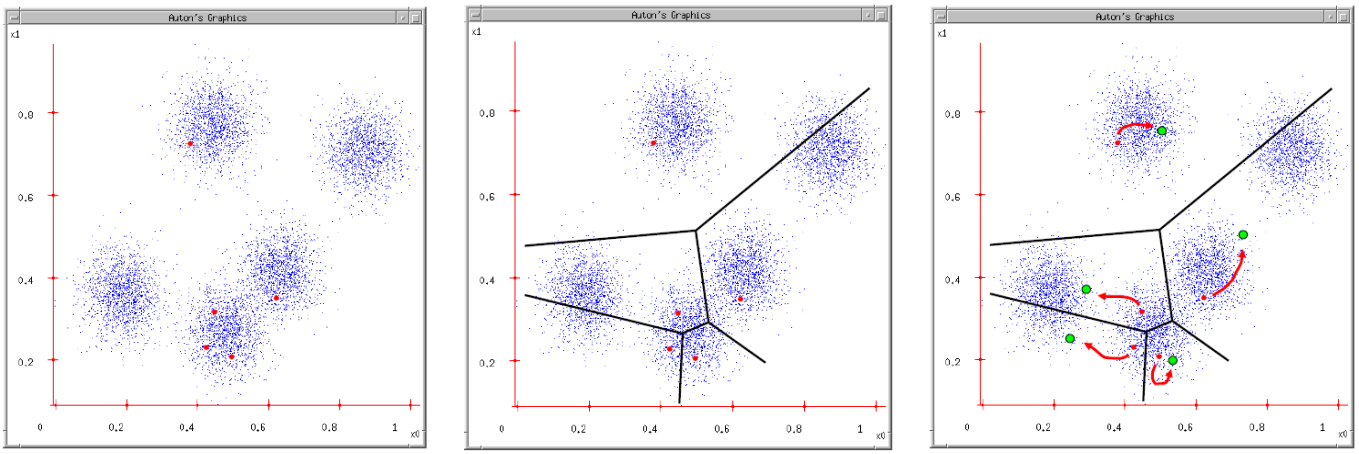
1. $\forall x$-re keressük meg a hozzá legközelebb eső $C^{(i)}$-beli középpontot, és társítsuk hozzá.
2. Számítsuk ki $C^{(i+1)}$-et úgy, hogy minden középpontot helyettesítsünk a hozzá társított pontok tömegközéppontjával.

Az 2.44. ábrán láthatóak a K–means lépései:

a) Adottak az aktuális középpontok, $C^{(i)}$

b) Az adathalmaz pontjait a hozzájuk legközelebb eső középponthoz rendelve a pontok K részhalmazra osztódnak.

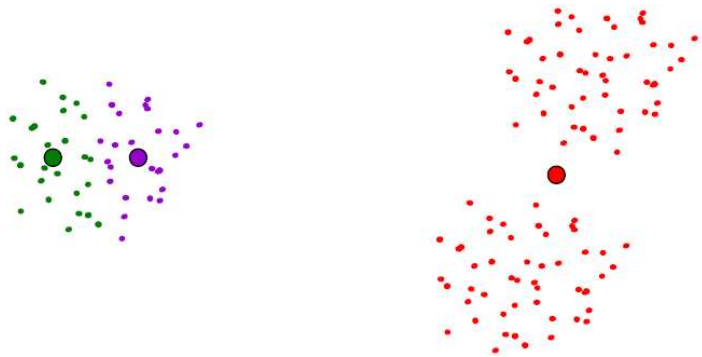
c) A klaszterekbe eső pontok tömegközéppontjának kiszámításával a középpontok elmozdulnak, $C^{(i+1)}$.



a) b) c)

2.44. ábra. Egy K–means iteráció lépései.

Tulajdonképpen a feladat célfüggvénye a középpont és a hozzá tartozó pontok közötti átlagos négyzetes távolságok összege. A K–means algoritmus ennek a függvénynek egy lokális minimumához konvergál. Emelett ismeretes, hogy adatbázisokra való gyakorlati alkalmazása nagyon lassú. Több munka született már a témához kapcsolódóan, melyeknek nagy része nem az algoritmus módosítását célozza meg, hanem különböző kiegészítéseket javasolnak az alap K–means algoritmus mellé, melyekkel javíthatnak az módszer eredményességén.



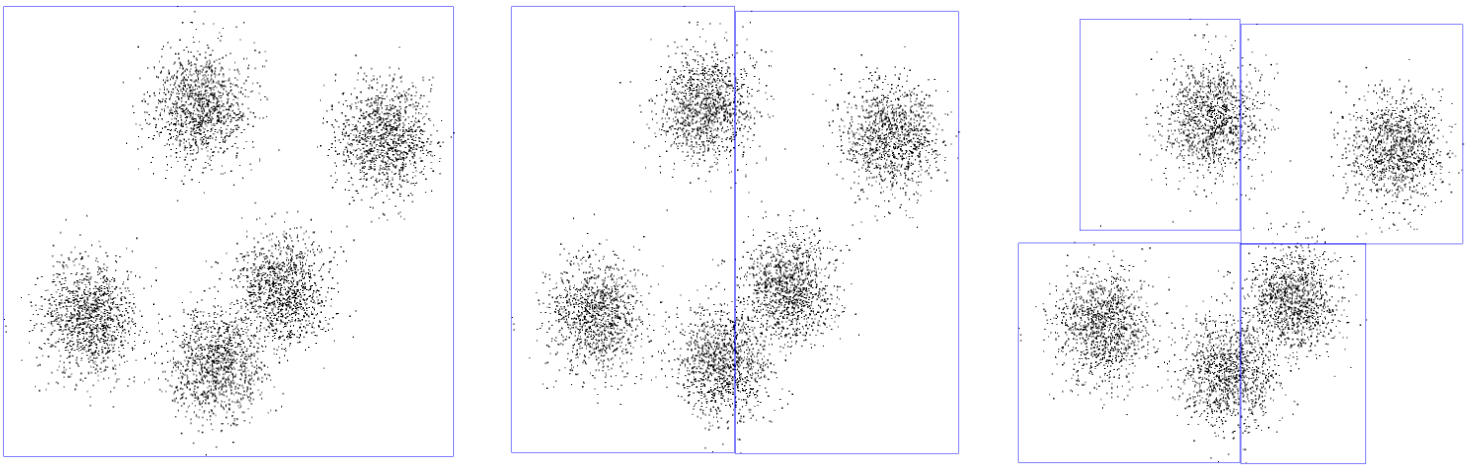
2.45. ábra. K–means – véletlen kezdőpontok hatása az eredményre.

Habár a kezdeti középpontokat tetszőlegesen, megszorítás nélkül választhatjuk ki, a K–means algoritmus teljesen determinisztikus adott kiindulási állapot esetén. A kezdeti középpontok rosszul választása nagy (és persze negatív) hatással lehet a teljesítményre és a célfüggvény értékre egyaránt. A 2.45. ábrán látható egy példa, hogy a rosszul megválasztott kiindulási állapot nem optimális klaszterezést eredményezett.

2.11.2. kd–fák

A hagyományos K–means algoritmus naív módon határozza meg a középponthoz való tartozást, vagyis minden iterációban az adathalmaz minden pontjára kiszámítja az összes középponttól való távolságot, és így választja ki a legközelebbit. Ez a megoldás sok újraszámítást igényel, melyek egy része felesleges. Innen

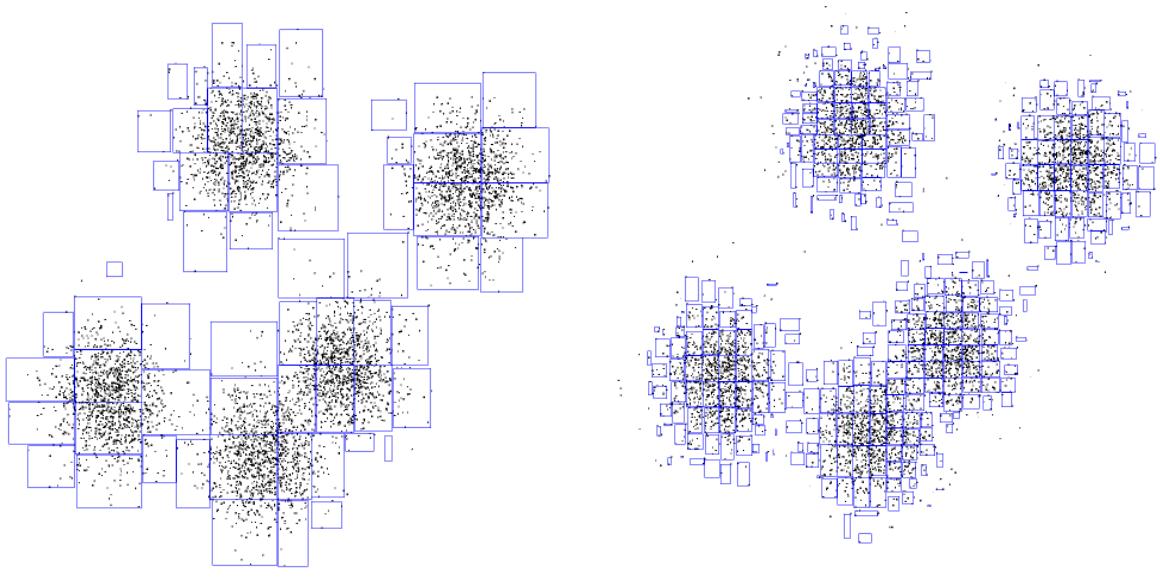
jött az ötlet, hogy olyan adatszerkezetet kellene használni, ami plusz információt tud tárolni egy adott ponthalmazról.



a) b) c)

2.46. ábra. a) kd–fa gyökere; b) gyökér gyerekei; c) gyökér unokái.

A kd–fa egy bináris fa adatstruktúra, melyben minden csúcs egy adott hipertéglalapba eső ponthalmazt tárol. A gyökér tárolja az összes pontot (2.46.a ábra). A szülő csúcshoz tartozó hipertéglalapot valamely tengellyel párhuzamosan két részre osztjuk. Az így kapott részeket tartalmazzák az adott csúcs gyerekei (2.46.b és 2.46.c ábra). Ezt a felosztást addig folytatjuk, amíg a hipertéglalapok több csúcsot tartalmaznak. A fa levelei magukat a pontokat tartalmazzák egyenként (2.47. ábra).



a) b)

2.47. ábra. a) kd–fa belső pontjai; b) kd– fa pontjai közel a levelekhez.

Tehát minden csúcs a fában az adathalmaz egy részhalmazát reprezentálja, melyről plusz információkat is tartalmaz, például az őket befoglaló tengelyekkel párhuzamos oldalú hipertéglalap paramétereit, a pontok számát, tömegközéppontját, stb. Emellett egy mutatót tartalmaznak mindkét gyermekükre, melyek a szülő által reprezentált ponthalmaz részhalmazait tárolják.

Abban az esetben, ha a kd–fában egy adott csúcshoz tartozó ponthalmaz minden tagja egy középponthoz tartozik, akkor a középpont következő mozgását a halmaz tömegközéppontja és a benne lévő pontok száma befolyásolja. Ezért minden kd–fa csúcsra a tömegközéppontot előre kiszámítva, és ezt az adott csúcsban eltárolva, nagy mennyiségű számítást spórolhatunk meg. Adott csúcs ponthalmazának adott középponthoz való tartozását szintén hatékonyan lehet eldönteni. A kd–fák ezen kívül a kezdeti középpontok meghatározását is hatékonnyá teszik

2.11.3. X–means

Ahogy azt a korábbiakban említettük, az eredeti K–means algoritmusnak szükséges egy konkrét K érték megadása, amely alapján kezdeti középpontokat generálhatunk, és amelyek konvergálása nyomán kialakul a K darab klaszter. Ezen értékek meghatározása a felhasználó számára nem mindig triviális feladat.

Az X–means eljárás azon az ötleten alapszik, hogy nem konkrét K érték adott, hanem egy tartomány, melyen belül dinamikusan kell megkeresni az optimális klaszterszámot. Vagyis a felhasználónak csak az alsó és felső korlátokat kell megadnia, melyből dinamikusan keressük meg a legjobb eredményt adó K értéket.

Az X–means algoritmus a teljes adathalmazra alkalmazza a K–means klaszterező eljárást a megadott legkisebb lehetséges K értékkel, és folyamatosan növeli a középpontok számát, ahol szükséges egészen addig, amíg nem éri el az adott felső korlátot. Az algoritmus a következő:

1. **Paraméterek javítása:** a hagyományos K–means iterációk futtatása.
2. **Struktúra javítása:** meghatározza, hogy kell–e és ha igen, akkor hol szükséges új középpontot létrehozni. Ez az adott középpont kettéosztásával valósul meg.
3. **Megállási feltétel:** Ha $K > K_{max}$, akkor megállunk, és az eddig talált legjobb modell a megoldás. Különben vissza az 1. ponthoz.

A kérdés az, hogy hogyan döntjük el, hogy melyik középpontot osszuk ketté? Erre két alapötlet adott:

- **Egyszerre egyet.** Válasszunk egy középpontot, osszuk ketté, majd futtassuk le a K–means algoritmust és nézzük meg, hogy jobb eredményt adott-e. Ha igen, fogadjuk el az új struktúrát, ha nem, térjünk vissza az előzőhöz. Ez a megoldás $O(K_{max})$ struktúra javítási lépést igényel, amíg az X–means lefut. Emiatt célszerű valamilyen módon kiválasztani azt a középpontot, amelyiket érdemes ketté osztani.

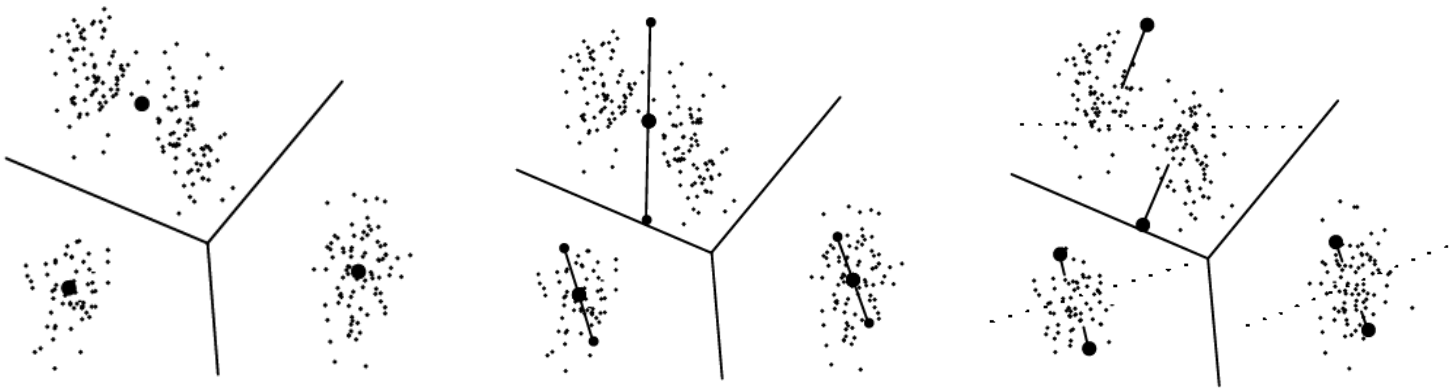
  Ebben az esetben is valószínű, hogy minden középpontot vizsgálnunk kell, ami nagyon időigényes, figyelembe véve, hogy egyesével adjuk hozzá az új középpontokat a modellhez.

- **Próbáljuk a felét.** Válasszuk ki a középpontok felét valamilyen heurisztikát alkalmazva arra, hogy mennyire ígéretes a kettéosztásuk. Osszuk ketté őket, futtassuk le a K–means algoritmust, és nézzük

meg, hogy az új struktúra jobb eredményt ad-e. Ha igen, fogadjuk el. Ez a módszer $O(logK_{max})$ struktúra javítási lépést igényel.

A heurisztika kiválasztása azonban nem triviális. Alapulhat például a középpontokhoz tartozó területen, vagy az adott középponthoz tartozó célfüggvényértéken. Hátránya ezen kívül, hogy bizonyos esetekben egy–két középpontot jó ketté osztani, míg a többit nem.

Pelleg és Moore ezt a két ötletet ötvöző megoldást dolgozott ki, mellyel kihasználják mindkét módszer jó tulajdonságait, ugyanakkor kiküszöbölik a hátrányaikat. Az 2.48.a ábrán látható egy stabil K–means megoldás, melyben a három középpont a határokkal jelölten osztja szét a pontokat. A struktúra javítása minden középpont kettéosztásával kezdődik (2.48.b ábra).



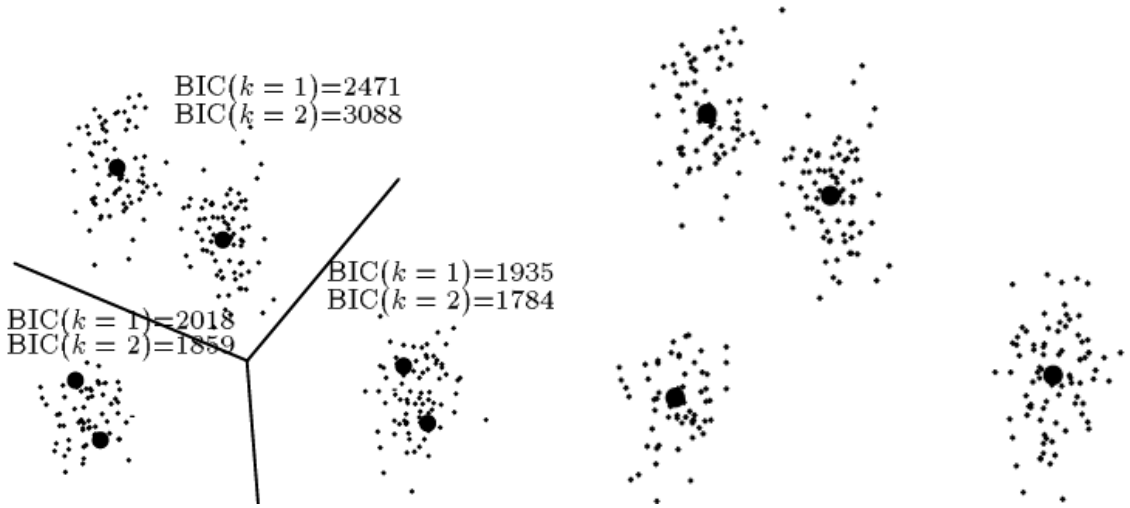
a) b) c)

2.48. ábra. Középpontok kettéosztása és párhuzamos 2–means futtatás.

A származtatott középpontok az adott tartomány nagyságának arányában egy véletlenszerűen választott vektor mentén lesznek elmozdítva ellentétes irányban az eredeti középponttól. Ez után minden tartományban helyileg párhuzamosan futtatjuk a K–means algoritmust (K=2 értékkel) a kapott származtatott középpontokat kezdeti értékeknek tekintve. Az 2.48.c ábrán látható a helyileg futtatott K–means eljárások első lépése, melyen a behúzott vonalak a középpontok elmozdulását mutatják.

A 2.49.a ábra szemlélteti, hogy a származtatott középpontok hova kerülnek a párhuzamosan futtatott 2–means eljárás végén. Ennél a pontnál döntést kell hoznunk arról, hogy az eredeti középpont vagy a belőle származtatott középpontok modellezik jobban a struktúrát, és ettől függően az eredeti vagy a származtatott középpontokat fogadjuk el, míg a másikat elvetjük. Ennek eldöntésére a Bayesian Information Criteria-t (BIC) használják, amely a pontok logaritmikus valószínűsége alapján számítható. A 2.49.a ábrán láthatóak az eredeti és a származtatott középpontok esetére számított BIC értékek, melyek közül a nagyobbat választva megkapjuk az új 2.49.b ábrán látható modellt, vagyis a 3 kezdeti középpont közül csak az egyik kettéosztását fogadjuk el a kapott BIC értékek alapján.

Ezzel a módszerrel minden lehetséges középpontfelezést megvizsgálunk, és a két alapötlettel összevetve így automatikusan választhatjuk ki, hogy egy körben nagyon kevéssel (ha már közel vagyunk az optimális K értékhez), vagy nagyon sokkal (ha a K értéke még nagyon alulbecsült) növeljük a klaszterek számát. Emellett a helyileg futtatott 2–means eljárások kevésbé érzékenyek a lokális optimumra.



a) b)

2.49. ábra. X–means.

Az imént ismertetett bővítések számottevő javulást eredményeztek az eredeti K–means algoritmussal összehasonlítva. A kd–fa adatszerkezet alkalmazásával 170-szeres gyorsulást tapasztaltak valós asztrofizikai adatokon, illetve fölényt a naív módszerrel szemben 5 vagy annál kevesebb dimenziós szimulált adatok esetén. Az optimális K érték meghatározására javasolt X–means eljárás szintén hatékonynak bizonyult. A $[2, \ldots, K_{max}]$ intervallumból dinamikusan választott K értékkel jobb eredményt szolgáltat a BIC értékeket tekintve, mint az iteráltan alkalmazott K–means. Emellett nagy problémákon kétszer olyan gyors, mint a kd–fákkal megvalósított K–means algoritmus.

3. fejezet

Többtényezős döntések

3.1. Bevezetés

A többtényezős döntések tárgykörével először az 1800-as években Marquis de Condorcet foglalkozott, aki a szavazásokat, és azok igazságosságát vizsgálta. A szavazás csoportos döntés, ahol azt fejtegette, hogy az egyes embereknek más súlyú a szavazatuk. A többtényezős döntések témakörébe tartozik még például egyes sportágak (pl. tenisz, öttusa) pontozási rendszere. Természetesen itt is felmerül az, hogy adekvátak-e (igazságosak-e) a döntések. Egy rendszer igazságosságának eldöntésére tanulóalgoritmus alkalmazható. Például az egyetemi felvételi rendszer adekvátságának eldöntésére lehetne tanulóalgoritmust alkalmazni, ha megvizsgálnánk, hogy a felvett hallgatók közül ki milyen sikerrel állt helyt az egyetem elvégzése után, elvégezte-e az egyetemet, illetve milyen sikeres lett. A tanulóalgoritmus alkalmazását azonban egy kutatási folyamat előzi meg, ilyen például fényképezőgép vásárlása előtt az alternatívák felderítése, és azután a számunkra lényeges kritériumok figyelembevétele. A többtényezős döntések esetében az alternatívák száma jóval nagyobb, mint kritériumok száma, a szavazásoknál pedig fordítva. Ott a szavazók (kritériumok) száma nagy, és csak néhány alternatíva között kell dönteni.

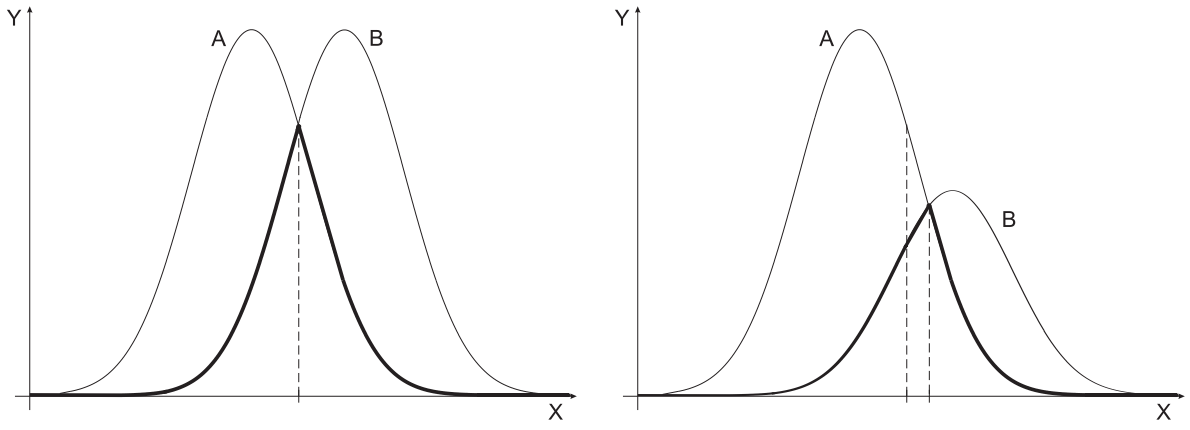
	c_1	c_2	$\cdots$	c_n
a_1	x_{11}	x_{12}	$\cdots$	x_{1n}
a_2	x_{21}	x_{22}	$\cdots$	x_{2n}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
a_n	x_{n1}	x_{n2}	$\cdots$	x_{nn}
	w_1	w_2	$\cdots$	w_n

A fenti táblázatban az a_i értékek az alternatívák, a c_i értékek a kritériumok, és a w_i értékek a súlyok (fontosság). Például az $a_1, a_2, \ldots, a_n$ alternatívákkal reprezentáljuk a pácineseket és $c_1, c_2, \ldots, c_n$ vizsgálatok vannak, melyeket el lehet végezni a pácienseken. Döntést akarunk hozni arról, hogy például tüdőbeteg-e egy páciens. Ekkor a $w_1, w_2, \ldots, w_n$ vektort kell meghatározni, hogy a döntés segítégével az illető tüdőbeteg-e. Természetesen például vesebetegség eldöntésére más vizsgálatok és más súlyok kellenek.

Ha a diagnózisnak több fajtája van azaz több diagnosztizálási osztály létezik $(o_1, o_2, \ldots, o_l)$, akkor a következőképp reprezentálhatjuk a problémát:

	c_1	c_2	$\cdots$	c_n
o_1	w_{11}	w_{12}	$\cdots$	w_{1n}
o_2	w_{21}	w_{22}	$\cdots$	w_{2n}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
o_l	w_{l1}	w_{l2}	$\cdots$	w_{ln}
$\cdots$	$\cdots$	$\cdots$		

ahol a w-k az individuumok „céljai”, ezért a táblázat szubjektív, mindenkinek különböző. A kritériumok között lehet fontossági sorrend, például ha két kritérium van, és az A alternatíva legyen fontosabb, mint a B! A két alternatíva között keressük a kompromisszumot. Természtesnek tűnik, hogy a fontossági transzformáció során növeljük A-t és csökkentjük B-t.



a) b)

3.1. ábra. a) A és B egyenlő fontosságú; b) B-t csökkentve a $\min(A, B)$ csúcspontja B felé tolódik.

Ekkor jól látható, hogy a $\min(A, B)$ csúcspontja eltolódott a B irányába (3.1. ábra), vagyis a transzformáció rossz, éppen fordítva kell végrehajtani, a fontosabbat csökkenteni, a kevésbé fontosat növelni kell. Ez a súlyozás monotonitásának paradoxona. Ebben az esetben a metszet szerint döntöttünk. Ha az unió szerint döntünk, akkor a fontosabb alternatíva növelése jó irányba változtatja a függvényt.

Ha súlyokat szeretnénk reprezentálni egy eljárás során, akkor fuzzy operátorokat is figyelembe kell venni. Ha a feladat $f^{-1}(\sum f(x))$ alakú, és súlyozni szeretnénk az operátort, akkor az alábbi alakot célszerű választani:

$$f^{-1}\left(\sum w_i f(x)\right)$$

Például:

- ha $f(x) = x^p$, akkor:

$$w_i = \frac{1}{n} \Rightarrow \left(\frac{1}{n} \cdot \sum x_i^p\right)^{\frac{1}{p}}$$

Ha $n = -1$ akkor a harmonikus közepet, és ha $n = 1$, akkor a számtani közepet kapjuk.

- ha $f(x) = \ln x$, akkor:

$$e^{\frac{\sum \ln x}{n}} = \sqrt[n]{\Pi x}$$

Ami a geometriai közép.

A fenti példából is jól látható, hogy ez egy igen általános képlet. Kérdés, hogy a két közép közül melyik a nagyobb?

$$\begin{array}{ccc} f^{-1}(\sum \underbrace{w_i f(x)}_{x}) & ? & g^{-1}(\sum w_i g(x)) \\ \underbrace{g(f^{-1}}_{F}(\sum w_i x_i)) & ? & \sum w_i \underbrace{g f^{-1}(x)}_{F} \\ \hline F(\sum w_i x_i) & ? & \sum w_i F(x_i) \end{array}$$

Milyen kapcsolatban vannak egymással f és g? Tegyük fel, hogy $\sum w_i = 1$!

$$F(\frac{x_1 + x_2}{2})$$

Ha F konkáv, $\Rightarrow$ $F(\sum w_i x_i) > \sum w_i F(x_i)$

ha F konvex, $\Rightarrow$ $F(\sum w_i x_i) < \sum w_i F(x_i)$

Az F függvény konvexitását a másodrendű deriváltjának segítségével dönthetjük el.

A többtényezős döntéseknek alapvetően kétféle megközelítése létezik:

- Hasznosságelmélet (angolszász iskola, képviselői: Neumann János, Oskar Morgenstern, Peter C. Fishburn)
- Preferencia alapú (európai, francia iskola, képviselői: Roy, Mareshal, Vinche, Rubens)

	c_1	c_2	$\cdots$	c_n
a_1	x_{11}	x_{12}	$\cdots$	x_{1n}
a_2	x_{21}	x_{22}	$\cdots$	x_{2n}
$\cdots$	$\cdots$	$\cdots$		
a_n	x_{n1}	x_{n2}	$\cdots$	x_{nn}
	w_1	w_2	$\cdots$	w_n

	c_1	c_2	$\cdots$	c_n
o_1	w_{11}	w_{12}	$\cdots$	w_{1n}
o_2	w_{21}	w_{22}	$\cdots$	w_{2n}
$\cdots$	$\cdots$	$\cdots$		
o_n	w_{n1}	w_{n2}	$\cdots$	w_{nn}
$\cdots$	$\cdots$	$\cdots$		

3.2. Az additív hasznosságelmélet

Hasznosság (utility, utilitás) függvény: szigorúan monoton növő vagy szigorúan monoton csökkenő.
Visszatérve a gépkocsivásárlási példára, egy preferenciasorrendet szeretnénk felállítani az autók között: Az addititív utilitási eljárás képlete:

$$u(\underline{x}) = \sum w_i u_i(x)$$

Egyike a gyakrabban alkalmazott eljárásoknak, főként egyszerűsége miatt használják. A problémát itt a súlyok, az utilitásfüggvények meghatározása jelenti. Az utilitásfüggvény meghatározása, vagyis a paraméterek kinyerése az úgynevezett assessment (=kinyerés) eljárás segítségével történhet.

Egy alkalmazási területe a többtényezős döntéseknek az orvosi diagnosztika. Az Egyesült Államokban például az orvosi kezelések felírása az alábbiak szerint történik: adottak betegségek, illetve terápiák, melyek bizonyos mértékig hatásosak, illetve melyeknek mellékhatásaik vannak. Arról hoznak döntést, hogy az adott betegnél melyik terépiát érdemes alkalmazni. Ilyenkor egy általánosabb modellt, a multiplikatív utilitásfüggvényt alkalmazzák:

$$u(x) = \frac{1}{k}(\Pi(1 + kw_i u_i(x)) - 1)$$

A fenti függvényt szokás „élet-halál" függvénynek is nevezni, mert a terápiák alkalmazásáról is dönthet. $u(x)$ egy polinom, melyre $u_i(x) = 1$, akkor $u(x) = 1$. Így:

$$1 = \frac{1}{k}(\Pi(1 + kw_i) - 1)$$

A fenti egyenlet k gyökét kell meghatározni. Ha $\sum w_i = 1$ -et feltesszük, akkor az additív utilitást kapjuk (az ennek speciális esete). A w_i és u_i mennyiségeket kérdőíven szokás szakértőktől megkérdezni.

Az UTA-módszer

Az UTA betűszó a utility assessment angol szavakból ered. Az UTA az 1980-as években jött létre. A módszer arra ad választ, hogy hogyan lehet a w_i és u_i paramétereket megkeresni. Tegyük fel, hogy nagyon sok alternatívát kell jóságuk szerint rendezni, például adott 100000 cég, és rangsorolni kell őket, vagy adott több ezer beadott pályamunka, és ezeket kell rangsorolni. Ilyen esetekben nehézkes lenne kérdőívek alapján az összes paraméter meghatározása és érdemes szakértőket megkérdezni néhány alternatíva rendezéséről.

Például:

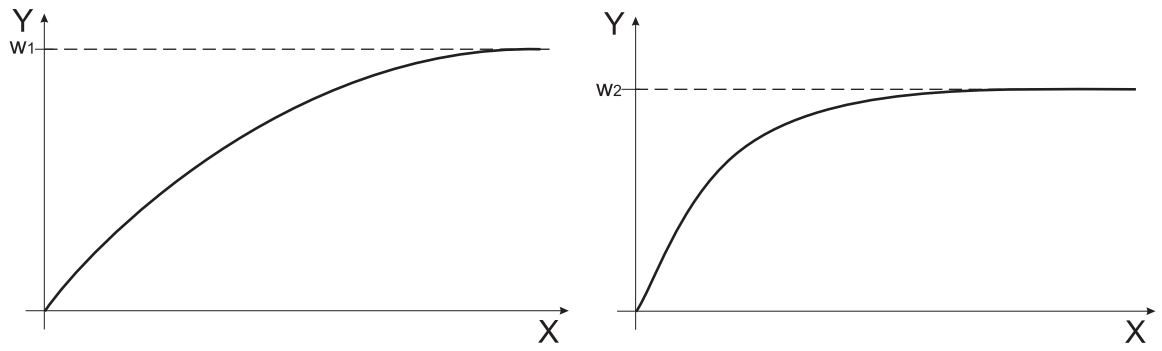
$$a_5 > a_7 > a_3 > a_2$$

$$a_1 > a_6 > a_8$$

$$\dots$$

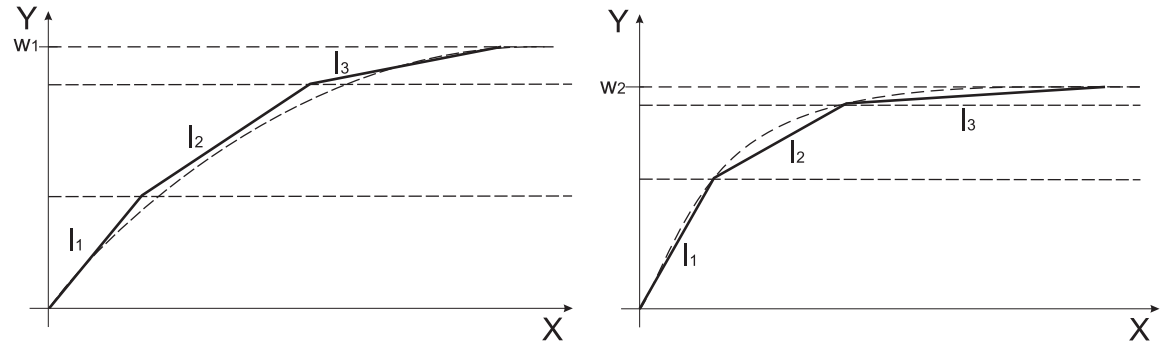
Az eljárás során feltesszük, hogy $\sum w_i = 1$

w_i beolvasztahtó $u_i(x)$ -be, legyen $u_i(x) = w_i u_i(x)$. Tehát $u_i(x)$ maximális értéke w_i.



3.2. ábra.

Az u_i-t lineáris szakaszokkal próbáljuk meg közelíteni:



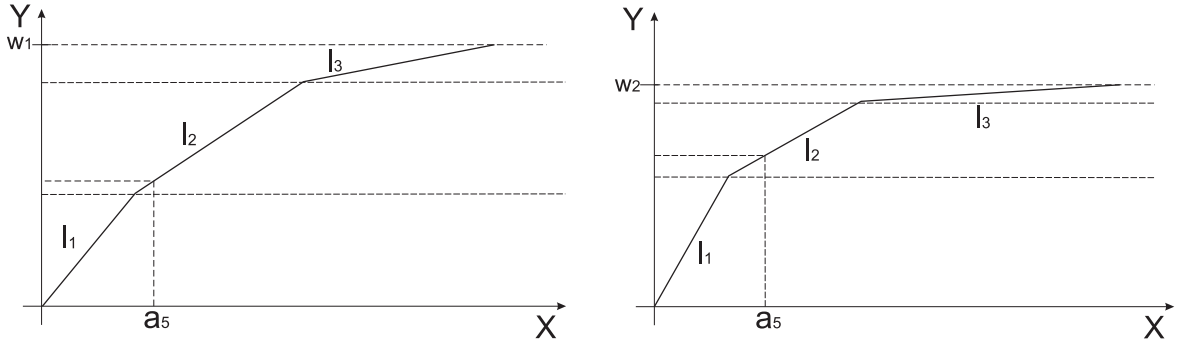
3.3. ábra.

$$l_i^{(j)}(x) = a_i^{(j)}(x) + b_i^{(j)}$$

Ebben az esetben a következőképpen fogjuk felhasználni a rendezési információt: pl. $a_5 > a_7$

$$u_1(a_5) + u_2(a_5) > u_1(a_7) + u_2(a_7)$$

Az a_5 pl az első illetve a második függvénynél:



a) b)

3.4. ábra.

$$l_1^{(1)}(x_5^{(1)}) + l_2^{(2)}(x_5^{(2)}) > l_1^{(1)}(x_7^{(1)}) + l_2^{(2)}(x_7^{(2)})$$

Behelyettesítve az $l_i^{(j)}$ értékeket:

$$a_1^{(1)}x_5^{(1)} + b_1 + a_2^{(2)}x_5^{(2)} + b_2 > a_1^{(1)}x_7^{(1)} + b_1 + a_2^{(2)}x_7^{(2)} + b_2$$

Azonban nem ismertek $a_i^{(j)}$ és b_i értékek. Átrendezve az egyenlőtlenséget kapjuk:

$$\sum X\underline{a} > 0,$$

ahol az $\underline{a}$ tartalmazza az egyes paraméterek értékeit $(a_i^{(j)}, b_i)$

Így egy egyenlőtlenségrendszert kaptunk. A legutolsó értékek összege 1, mivel $\sum w_i = 1$. Ez a konstrukció az LP feladathoz hasonlít, de még nincs célfüggvény, csupán a korlátok adottak. Konstruáljuk meg a célfüggvényt!

Azt tudjuk, hogy $a_5 > a_7$, de azt nem, hogy a_5 mennyivel jobb, mint a_7. Vezessük be ennek érdekében a következő mennyiségeket: $\delta_1, \delta_2, \cdots, \delta_n$, ahol $\delta_i > 0$ ha $0 < i \leq n$

$\sum \delta_i$ tehát azt jelenti, hogy mennyivel jobb az egyik alternatíva a másiknál. Ezek alapján a célfüggvény:

$$\sum \delta_i \rightarrow max$$

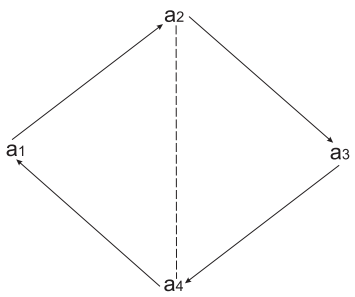
Az UTA-eljárás a tanulható preferencián alapszik, egy tanulóalgoritmus is egyben, mely a döntési eljárást tanulja. Az eljárásnak több módosítása létezik:

- Az egyenlőség reláció megengedése: két alternatíva között az egyenlőséget is megengedjük.
- Iteratív módosítás: az alapeljárás végeredményéből újabb egyenleteket adhatunk iteratív módon, ezzel pontosítva a végeredményt. Veszélye ennek a módosításnak a manipulálhatóság (pl. pályázat elbírálása során egy előre maghatározott pályázatot előre helyezve).
- Manipuláció kiküszöbölése: fiktív alternatívák generálásával lehetséges. Például ha cégeket akarunk rangsorolni profit és foglalkoztatott munkavállalók száma szerint, és adottak A, B, C fiktív cégek. Az A cég profitja évi 100 millió Ft, és 300000 embert foglalkoztat, a B cég profitja csupán évi 1 millió Ft, és 10 emberrel működik, és legyen a C fiktív cég profitja évi 50 millió Ft, és dolgozóinak kétszáma 1000 ember. A döntéshozók szerint a C vállalat jobb, mint a B, de roszabb, mint az A.

Az értékelő függvény monoton és a paraméterei szabadon változtathatók. A hasznossági függvényre további korlátozások érvényesek.

3.3. Outranking eljárások

(francia iskola) Az outranking eljárások preferenciaviszonyt állítanak fel az alternatívák közt, melyet egy preferenciamátrix reprezentál, aminek megfeleltethető az alternatívák feletti irányított gráf. Az angol *outrank* szó jelentése *rangban megelőz valakit*, mely az alternatívák felett értelmezett relációra utal.



3.5. ábra.

Látható, hogy vannak hiányzó élek, hiszen nem minden alternatíva között ismerjük a preferenciát.

1. Az ELECTRE-1 eljárás:

   Az ELECTRE-1 eljárás a ROY eljáráscsalád tagja, az elnevezés az angol *election* (szavazás) szóból ered. Az ELECTRE még az UTA előtt született meg. Az eljárás az alternatívák összehasonlíthatóságán alapszik. Fontos fogalom a diszkordancia (nem összehasonlíthatóság), és a konkordancia (preferáltság). Az eljárás lényeges eleme az alternatívák tulajdonságai közötti összefüggés mértékét leíró $\underline{p}$ vektor.

	c_1	c_2	$\cdots$	c_n
a	x_1	x_2	$\cdots$	x_n
b	y_1	y_2	$\cdots$	y_n
	w_1	w_2	$\cdots$	w_n
	p_1	p_2	$\cdots$	p_n

   A táblázatban c_i kritériumok, a és b alternatívák, w_i súlyok és p_i preferenciák ($0 < i \leq n$). Az alternatívákat függőlegesen oszlop szerint, $x_i \ ? \ y_i$ illetve vízszintesen soronként, $x_i \ ? \ x_{i+1}$ és $y_i \ ? \ y_{i+1}$ is összehasonlíthatjuk. Tekintsük a függőleges összehasonlítást, tehát a p_i preferenciák jelöljék az oszlop szerinti összehasonlításokat, ahol $p_i \in [0, 1]$. A preferenciákra egy $\underline{p}$ vektort kapunk aszerint, hogy az adott x_i érték milyen relációban állt a vele egy oszlopban lévő y_i értékkel, például:

$$\begin{aligned} x_1 > y_1 &\Rightarrow p_1 = 1 \\ x_2 < y_2 &\Rightarrow p_2 = 0 \\ \cdots &\qquad \cdots \\ x_n > y_n &\Rightarrow p_n = 1 \end{aligned}$$

   A fenti megadása $\underline{p}$-nek nem mond semmit arról, hogy mi történjen egyenlőség esetén. Most megállapodás szerint legyen: ha $x_i \geq y_i \Rightarrow p_i = 1$

   Tehát $\underline{p}$ bináris vektort kapjuk a preferenciákra. Ha a súlyok egyenlők voltak, akkor tekintsük a $\sum p_i$ értéket, amennyiben a súlyok különböznek, úgy vegyük a $\sum w_i p_i$ értéket. Például a $w_1 p_1$ jelentése az, hogy x_1 jobb y_1 -nél a 0,9 fontosság figyelembe vételével, amennyiben w_1 értéke 0,9. Egy súlyozott, irányított gráfot kapunk (3.6. ábra).

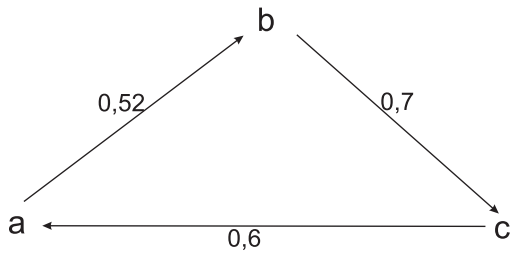
   A gráf irányítottsága fordított, ha (a, b)-t felcseréljük. Azt mondjuk, hogy $P(a, b)$ igaz, ha $\frac{1}{n} \sum p_i > \frac{1}{2}$

   Így feltéve, hogy nincs olyan tulajdonság, amiben a és b megegyezne:

$$\begin{array}{rl} & \frac{1}{n} P(a, b) \\ + & \frac{1}{n} P(b, a) \\ \hline & 1 \end{array}$$

   Ha a w_i fontosságok nem egyformák:

$$\sum w_i p_i > \frac{1}{2}$$



3.6. ábra.

Problémát jelent az egyenlőség, ugyanis ha egy él irányát meg akarjuk fordítani, akkor a $\underline{p}$ vektor elemeinek az ellentettjét kellene vennünk:

$$P(a,b): \quad 1 \quad 0 \quad 1 \quad \cdots \quad 1$$
$$P(b,a): \quad 0 \quad 1 \quad 0 \quad \cdots \quad 0$$

De $x_i = y_i$ esetén, ha egy értéknél 1 volt, akkor az összegben ez kétszer jelenik majd meg:

$$\frac{1}{n}(P(a,b) + P(b,a)) \neq 1$$

Ennek kiküszöbölésére egyenlőség esetén legyen $P(a,a) = \frac{1}{2}$, és így:

$$\frac{1}{n}(P(a,b) + P(b,a)) = 1$$

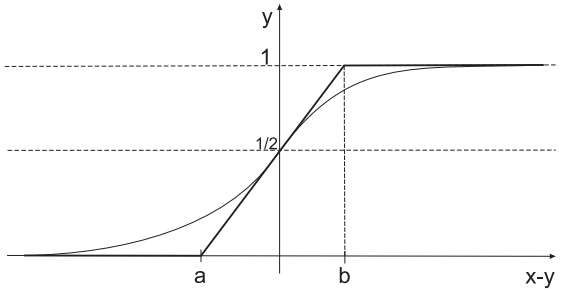
Így helyreáll a $P(a,b)$ és $P(b,a)$ közti összefüggés. A preferenciareláció tehát:

$$p_i(x,y) = \begin{cases} 0, & ha \quad x < y \\ \frac{1}{2}, & ha \quad x = y \\ 1, & ha \quad x > y \end{cases}$$

Felmerül a probléma, hogy a kapott gráf nem feltétlenül tranzitív. Szükség van a tranzitivitás helyreállítására. Ennek érdekében vizsgálnunk kell még egy tényezőt, méghozzá azt a tényt, hogy vajon minden alternatíva összehasonlítható-e minden tekintetben mindegyikkel. Vagyis a diszkordanciát, annak mértékét, hogy mennyire összehasonlítható két mennyiség. Ha túl nagy eltérés mutatkozik valamelyik tulajdonság szerint két alternatíva között, úgy azok nem összehasonlíthatóak. Tehát felvesszük a $d_1, d_2, \ldots, d_n$ értékeket, melyek a konkordancia küszöböket adják meg. Nem állítható fel preferenciareláció, amennyiben legalább 1 kritériumnál nem összehasonlíthatóak, azaz a küszöbeikben nagyobb eltérés van, és az irányított gráfban nem húzzuk be azt az élt.

Az eljárás hátránya az, hogy a preferencia érzéketlen a különbség mértékére, és ez nem reális abban a tekintetben, hogy amennyiben nagyon kis eltérés mutatkozott két mennyiség közt, úgy is 1-et ad értékül, csakúgy mint, amikor nagy az eltérés. Ennek a problémának a feloldásásra született meg az ELECTRE-1 fuzzysítása. Ennek a módszernek a kiindulópontja az, hogy legyen a preferencia a különbség függvénye, vagyis egy fuzzy halmaz:

$$p_i(x,y) = G(y-x)$$



3.7. ábra.

A halmazhoztartozási függvényeknek különböző alakja lehet (pl. lásd 3.7. ábra).

A diszkordancia értékeket is lehet aggregálni, fuzzysítani. Rendeljünk küszöbértékeket mind a konkordanciához, mind a diszkordanciához (δ_1, δ_2). Ez a meggondolás már közelít az ELECTRE-3 eljáráshoz, amelynél mindkét oldal fuzzy. Ennek a megközelítésnek több változata van, például amikor a két δ_1 és δ_2 küszöbből egy δ küszöböt generálunk (aggregálunk), melyet könnyebb mozgatni, mint kettőt.

Az ELECTRE eljárásokról 146 jelentős cikk jelent meg. Két magyar kutató is foglalkozott ezzel az eljáráscsaláddal: Kindler és Papp, ők alkották meg az úgynevezett KI-PA eljárást, ami szintén egy ELECTRE-variáns. Az ELECTRE esetében a súlyozási módokból rengeteg van.

2. PROMETHEE-eljárás:

x_1	x_2	x_3
y_1	y_2	y_3
$d(x_1, y_1)$	$d(x_2, y_2)$	$d(x_3, y_3)$

Itt a $d(x_i, y_i)$ értékek a különbségeket jelentik. A különbségfüggvények mindig két paraméteresek, és ezt a két paramétert kell megadniuk a kérdezetteknek. Ezen függvények alakja 6-8 féle lehet, és kérdések alapján ezeket kell meghatározni. Pl:

1. Kis különbségek vannak: rögtön preferált lesz, amint átlépi az értéket
2. Van „érzéketlenség”: adott egy küszöb, mely alatt „levágjuk” az értéket
3. Egyenletesen változó függvény
4. Eltolt egyenletes (ELECTRE-jellegű)
5. S-alakú függvény, ahol az élesség (meredekség) meghatározása a jelentős
6. Lépcsős függvény: egy határig 0, aztán $\frac{1}{2}$, majd egy határon túl 1

$$\sum w_i p(a,b) = P(a,b)$$

Minden alternatíva jellemezhető, és így köztük rangsor is megadható.

$U^-(a) = \frac{1}{n}\sum P(a,x)$

$U^+(a) = \frac{1}{n}\sum P(x,a)$ is számolható

A PROMETHEE is tartalmaz diszkordanciát. Meghatározása: az a és b alternatívák viszonyát vizsgáljuk a többi alternatívához képest.

$$U^+(a) < U^+(b) \quad \wedge \quad U^-(a) > U^-(b)$$

az összehasonlíthatóság feltétele.

3. AHP-eljárás:

Az AHP betűszó jelentése: Analytical Hierarchic Process, vagyis analitikus hierarchikus eljárás. Az eljárás megalkotója, Thomas L. Saaty statisztikával foglalkozott. Amikor megszületett ez az eljárás, minden korábbit (UTA, ELECTRE, stb.) „elsöpört", és ennek oka használatának egyszerűsége. Azóta számos kritika érte, de újabb módosításai miatt nem vesztett népszerűségéből. (Saaty eredményeit először egy pszichológiai lapban tette közzé.)

Az emberek általában nehezen tudják számokkal reprezentálni azt, hogy melyik alternatívát mennyivel tartják jobbnak egy másiknál. Azonban sokkal könnyebb, ha csak egy szűkített skálával kell ábrázolniuk a különbségeket. Az AHP-eljárás során a súlyok arányát kell megbecsülni, ami egy páros összehasonlítás, vagyis $\frac{w_i}{w_j}$ hányados. Az eljárásban csupán $1-9$-ig használhatók a számjegyek, így a lehetséges számjegyek $\frac{1}{9}, \frac{1}{8}, \cdots, 1, 2, \cdots, 9$.

Az összehasonlításokból egy mátrixot kapunk:

$$A = \begin{pmatrix} \frac{w_1}{w_1} & \frac{w_1}{w_2} & \frac{w_1}{w_3} & \cdots & \frac{w_1}{w_n} \\ \frac{w_2}{w_1} & \frac{w_2}{w_2} & \frac{w_2}{w_3} & \cdots & \frac{w_2}{w_n} \\ & \cdots & \cdots & & \cdots \\ \frac{w_n}{w_1} & \frac{w_n}{w_2} & \frac{w_n}{w_3} & \cdots & \frac{w_n}{w_n} \end{pmatrix}$$

Vizsgáljuk meg az A mátrix rangját! Ha adott az első sor $a_1 = (\frac{w_1}{w_1} \quad \frac{w_1}{w_2} \quad \cdots \frac{w_1}{w_n})$, akkor az összes többi elemet is meg tudjuk határozni, ugyanis:

$$\frac{w_2}{w_3} = \frac{\frac{w_1}{w_3}}{\frac{w_1}{w_2}} = \frac{w_3}{w_2}$$

A mátrix első sorából gyakorlatilag kiszámítható az egész mátrix, tehát a mátrix rangja 1. Mivel azonban nem konzisztens a becslés, a rang n legtöbbször. Szorozzuk be az A mátrixot a $\underline{w}$ vektorral:

$$\begin{pmatrix} \frac{w_1}{w_1} & \frac{w_1}{w_2} & \frac{w_1}{w_3} & \cdots & \frac{w_1}{w_n} \\ \frac{w_2}{w_1} & \frac{w_2}{w_2} & \frac{w_2}{w_3} & \cdots & \frac{w_2}{w_n} \\ & \cdots & \cdots & & \cdots \\ \frac{w_n}{w_1} & \frac{w_n}{w_2} & \frac{w_n}{w_3} & \cdots & \frac{w_n}{w_n} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \cdots \\ w_n \end{pmatrix} = n \begin{pmatrix} w_1 \\ w_2 \\ \cdots \\ w_n \end{pmatrix}$$

Tehát az egyenletünk alakja az alábbi:

$$A\underline{w} = n\underline{w}$$

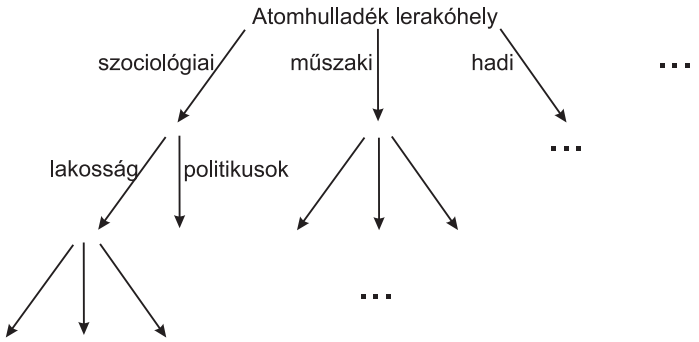
Ez a sajátvektoregyenlet. Tegyük fel, hogy a súlyokat normalizáltuk : $\sum w_i = 1$ legyen. A módszerünk a következő: határozzuk meg az A mátrix sajátvektorát (ez lesz a súlyvektor - $\underline{w}$), illetve a hozzá tartozó sajátértéket (n). Saaty megvizsgálta, hogy ha a kérdezettek nem konzisztensen töltik ki a kérdőíveket, nem következetesek a válaszaik, akkor ezt hogyan lehet mérni. Mivel számítható

$$A\underline{x} = \lambda_{max}\underline{x}$$

sajátérték, célszerű a $\frac{\lambda_{max}}{n-1}$-t a konzisztencia-mérőszámnak választani.

A következő problémák merültek fel az AHP-eljárással kapcsolatban:

1. Például azt a problémát szeretnénk megoldani, miszerint egy atomhulladék lerakóhelyet kell telepíteni. 5 különböző alternatíva jöhet szóba, és körülbelül 90 faktort kell figyelembe venni (pl. lakosság ellenállása, politikai megfontolások, mennyire védhető a hely, mennyire van távol a hulladék keletkezési helyétől, pénzügyi megfontolások, stb. – lásd 3.8. ábra). Ekkor a faktorokra vonatkozó kérdőívek $89 \cdot 90$-es méretűek, ami viszont óriási mennyiség, ennyi kérdést nem lehet szakértőknek feltenni, és valószínűleg nagyon nem konzisztens válaszokat kapunk. A különböző faktorokat szétválaszthatjuk kategóriákra, ezáltal fát építhetünk. Ennek az az előnye, hogy az egyes szakterületekre vonatkozó kérdéseket csak az adott szakembereknek tesszük fel, így kevesebb kérdést kell feltenni, és csak a témában kompetens embereknek.
2. Az eljárásban a $\frac{w_i}{w_j}$ hányadosokból $\sum w_i x_i$-vel szokás továbbszámolni. Felmerült a kérdés, hogy miért pont súlyozott összeggel számítjuk a súlyokat?



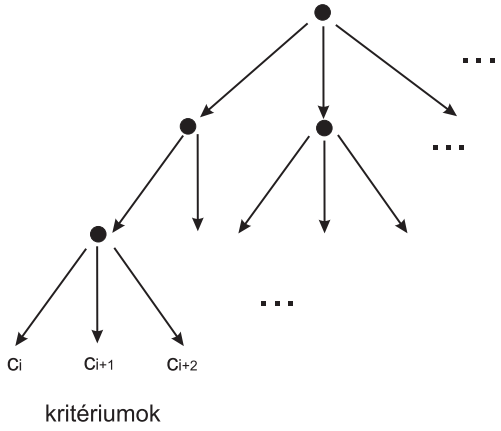
3.8. ábra. Atomhulladék lerakóhely probléma faktorai.

A 3.9. ábra szerint a 90 kritériumot lekezeltük 4 szinttel. A szintek közé súlyokat teszünk, és ezekre határozzuk meg a $w_1, w_2, \ldots, w_n$ értékeket. A legalsó szinten vannak a végső kritériumok, amik alapján dönteni kell.

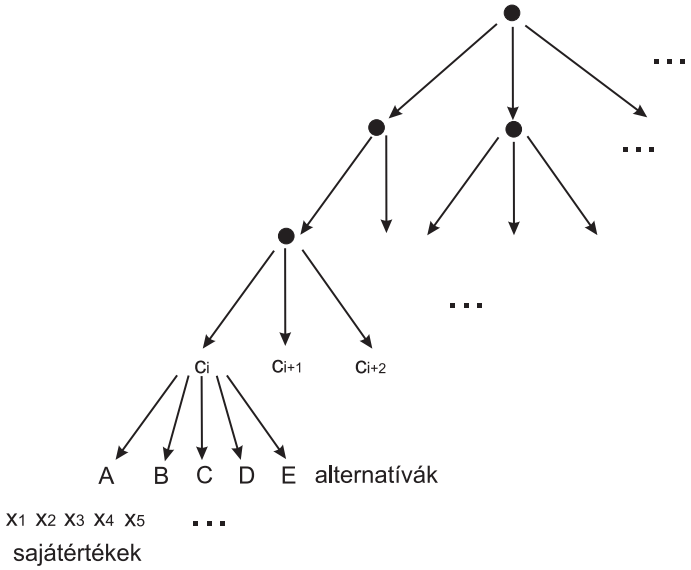
A c_i levelekben már ismertek a súlyok:

$$\Pi w_i = c_i$$

Ehhez a fához még hozzátesszük az alternatívákat (jelenleg az 5 helyet: A,B,C,D,E, így kapunk még egy 5×5-ös mátrixot, mely az alternatívákat és a hozzájuk tartozó sajátértékeket tartalmazza – 3.10. ábra):



3.9. ábra. Kritériumok lekezelése 4 szinttel; c_i-k végső kritériumok.



3.10. ábra. Fa kibővítése a helyek alternatíváival.

A $\sum w_i x_i$ kifejezésből megkapjuk a végső x_i értékeket.

A Microsoft cég is készített egy AHP-eljárást megvalósító szoftvert, neve SMART.

Az alábbi kritikák érték az eljárást:

- A konzisztencia-érték számítás nem jó.
- Önkényes az $1, 2, 3, \ldots, 9$ számok választása. (Más számstruktúrával más lesz a konzisztencia mértéke.)
- Az aránymérés után megfelelő-e a számtani középpel dolgozni? (A geometriai közép jobb választás.)

Az eljárás igen sikeres, mégis néha nem indokolja kellően az eredménye. Egy tudományos közleményben volt olvasható, hogy 5 éve Finnországban ezt az eljárást alkalmazták annak eldöntésére, hogy hová létesítsenek hockey-pályát. Finnországban mindössze 5 nagyobb város volt a jelölt. A pályát nagyvárosban akarták elhelyezni, ami 4-5 település közös pályája lehetett volna. A kérdés az volt, hogy pontosan hol legyen a beruházás. Számos kritériumot vetettek fel, például szociológiai szempontokat, parkolási lehetőségeket, a település(ek) befogadóképességeit, stb. Összesen mintegy 70-80 szempont jött össze. Az 5 lehetséges helyszínt rangsorolni kellett. Az AHP-eljárás eredménye a 2. alternatíva volt. Ekkor azonban közbeszólt a politika, és önkényesen a 4. alternatívát választották, melyet egyébként az AHP a legrosszabb helyre sorolt. Megépült a pálya, és pár év múlva megvizsgálták, hogy mennyire volt rossz a döntés. Kiderült, hogy valójában a döntés megfelelő volt. Az AHP tehát nem mindig írja le jól az adott döntési szituációt.

4. fejezet

Döntési fák

4.1. Bevezetés - ID3

Az ID algoritmusok egy elemhalmaz felhasználásával elemek egy osztályozására alkalmas *döntési fát* (*decision tree*) hoznak létre. Az elemeknek előre meghatározott, közös attribútumaik vannak, minden elem attribútumainak értéke ismert. Az algoritmus által létrehozott döntési fa bármely nem levél csomópontja egy attribútum alapján osztja szét az elemeket, az attribútum minden lehetséges értékéhez egy ágat rendelve. A fa leveleihez egy-egy osztályértéket rendelünk, amely az elem osztálya. Az algoritmus egy elemhalmazra eldönti a legalkalmasabb attribútumot, mely szerint az adott halmazt szétvágjuk. Így rekurzívan felépíti a döntési fát.

Az algoritmus működéséhez az alábbiakat kell meghatározni:

1. Attribútum-kiválasztó szabály: ennek segítségével a fa egy pontján meghatározzuk, hogy mely attribútummal érdemes a mintahalmazt felbontani.
2. Továbbontó szabály: ennek segítségével bontjuk tovább rekurzívan a mintahalmazt, vagyis további fapontokat határozunk meg.
3. Befejező szabály: ennek segítségével eldöntjük, hogy meddig kell bontani a mintahalmazt, vagyis mikor nevezünk el egy pontot levélnek.
4. Osztályozó szabály: ezzel minden levélhez egy osztályértéket rendelünk.

A négy szabály alapján egy általános faépítő eljárás a következő:

1. **lépés:** A befejező szabály alapján eldöntjük, hogy kell-e tovább bontani a fát. Ha nem, az aktuális ághoz az osztályozó szabály alapján rendeljünk osztályértéket, különben folytassuk az eljárást a 2. lépéssel!
2. **lépés:** Az adott mintahalmazra az attribútum-választó szabály szerint válasszunk egy attribútumot, és folytassuk a 3. lépéssel!

3. **lépés:** A kapott attribútummal a továbbontó szabály alapján bontsuk fel a mintahalmazt, és folytassuk a 4. lépéssel!

4. **lépés:** A kapott részhalmazok mindegyikéhez egy-egy ágat rendelve hajtsuk végre az eljárást mindegyik ágra!

Az algoritmus neve ID3, megalkotója Quinlan. Amennyiben nagy számú mintahalmazunk van, nem érdemes az algoritmust az összes elemre lefuttatni, hanem csak a mintahalmaz egy részhalmazára (ablakára). Ez alapján az ID3 általános leírása:

1. **lépés:** Válasszuk ki véletlenszerűen a mintahalmaz egy részhalmazát, és folytassuk a 2. lépéssel!

2. **lépés:** Ismételjük az alábbiakat, amíg a döntési fa tökéletesen nem működik a mintahalmazon! Tökéletesnek nevezünk egy döntési fát, amennyiben a mintahalamaz minden elemét helyesen osztályozza.

   a.) Az aktuális ablakra futtassuk le a faépítő eljárást, majd b.) lépés!

   b.) Keressük meg azokat az elemeket, melyekre a kapott döntési fa rosszul osztályoz, majd c.) lépés!

   c.) Hozzunk létre egy új ablakot a jelenlegiből, és a b.) lépésben kapott elemekből!

Az ID3 eljárásnak számos változata van, például olyanok, melyek kezelik a mintahalmaz zajosságát, vagy a hiányos attribútumértékeket is, stb. (Pl. ID3-IV, GID3-IV, CID3, kombinált folytonos ID3 → C4.5, stb.)

Egy döntési fával megoldható feladat: adottak páciensek, illetve vizsgálati eredményeik, és azt szeretnénk eldönteni, hogy melyik páciens milyen betegségben szenved. Ekkor a fa csúcsait az egyes vizsgálatok fogják képezni, és az adott csúcsnak annyi gyereke lesz, ahány féle eredménye lehetséges a viszgálatnak (pl. a vércukorszint lehet alacsony, normális és magas, tehát itt 3 gyereke lesz a vércukorszint csomópontnak), illetve a levelekben kétféle érték lehet: az adott betegségben szenved vagy nem.

4.2. A döntési fák tanulása, entrópia

A döntési fa építése során a megoldandó feladattípus nem más, mint egy fogalomtanulás diszkrét értékkészletű jellemzőkre. Az eljárás alapötlete William Ockham (egyes forrásokban Occam, továbbiakban itt Ockham) a 14. században élt angol szerzetes gondolata, mely Ockham-borotvája néven vált ismertté: „A konzisztens hipotézisek közül a legegyszerűbb a legjobb.” Vagyis a legegyszerűbb olyan fát keressük, mely jól osztályozza a mintákat. Ennek érdekében például felesleges attribútumokat (melyek nem befolyásolják a mintahalmaz felbontását) nem szükséges figyelembe vennünk.

Egy döntési fa felépítése egy adott mintahalmazon (adatbázison) tulajdonképpen egy elemi tanulási modell. A kapott döntési fa egy döntési eljárást ír le. A faépítés egy hierarchikus eljárás, hiszen az attribútumokon tulajdonképpen egy sorrendet állítunk fel. A minimális döntési fa készítése NP nehéz feladat, így heurisztikát kell alkalmazni. Nyilván az attribútum–választás lesz a kulcskérdés, hiszen arra törekszünk,

hogy úgy vágjuk szét a mintahalmazt, hogy utána a lehető legkisebb fát tudjuk felépíteni. A heurisztika alapja az **entrópia**. Az *entrópia* görög eredetű szó, melynek jelentése rendezetlenség. Ez a fogalom két területen is megjelenik: a fizikában, illetve az informatikában.

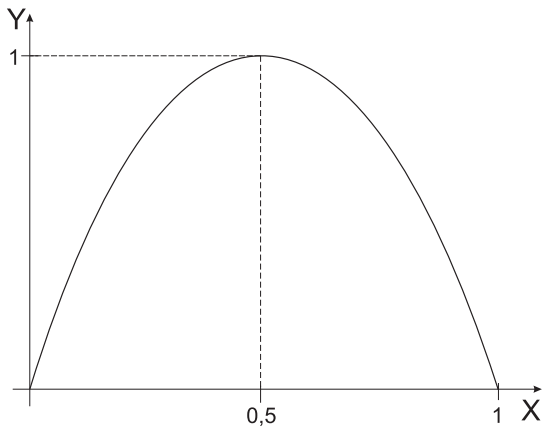
- Fizika: Itt az entrópiafogalom Boltzmann nevéhez fűződik. A hőmozgást végző részecskék elmozdulásának valószínűsége az entrópia. A hőtan egyik alapvető kérdése az volt, hogy ha két különböző hőmérsékletű gázt összeeresztünk, akkor mi történik az entrópiával? Arra az eredményre jutottak, hogy az entrópia mértéke növekszik, és hogy a hőkiegyenlítődés mindig egyirányú (ez a hőtan egyik alaptörvénye). Ezen jelenség által definiálható az idő iránya, ami zárt rendszerekben igaz. Ehhez kapcsolódik a hőhalál-elmélet, ami azt mondja ki, hogy a Világegyetem entrópiája folyamatosan nő, és így egyszer - körülbelül több 100 millió év múlva - eléri az egyensúlyi állapotot, mely ellehetetleníti majd az életet.

- Informatika: Körülbelül a '40-es évek vége óta használják ezt a fogalmat, Shannon nevéhez fűződik. Az entrópia itt a különböző csatornákban az átvett információ mértékét jelenti.

A döntési fa építés során az informatikából ismertté vált Shannon-féle entrópiát használják.

$$E_S = -k \sum p_i \log_2 p_i \tag{4.1}$$

A Shannon-féle entrópia (4.1), ahol p_i az információ mennyiség valószínűsége, $\log_2 p_i$ pedig az információ mértéke. Az entrópiafüggvény az információmérték várható értéke, maximuma ott van, ahol a valószínűségek egyformák, tehát $p_i = \frac{1}{n}$. Kétváltozós esetben: $E_S = -k \sum p_i \log_2 p_i = -(x \log_2 x + (1-x) \log_2 (1-x))$, tehát a maximum $\frac{1}{2}$-nél van (4.1. ábra).



4.1. ábra. Az entrópiafüggvény maximuma kétváltozós esetben $\frac{1}{2}$-nél van.

Megjegyzés: Nem csak Shannon alkotott entrópiára vonatkozó képletet, hanem például Rényi Alfréd matematikus is. A Rényi-féle entrópia (4.2) általánosabb, mint a Shannon entrópia, mivel (4.3).

$$E_R(p) = \frac{1}{\alpha - 1} \log \sum p_i^{\alpha} \tag{4.2}$$

$$\lim_{\alpha \to 1} E_R(p) = E_S(p) \tag{4.3}$$

4.3. Az algoritmus működése

Vezessük be az alábbi jelöléseket! Legyenek:

m	$\rightarrow$	az attribútumok száma
A	$\rightarrow$	$A = \{A_1, A_2, \cdots, A_m\}$ az attribútumhalmaz (m elemű)
$R(a_i)$	$\rightarrow$	az attribútumok lehetséges értékeinek száma
p	$\rightarrow$	az osztályok száma
C	$\rightarrow$	$C = \{C_1, C_2, \cdots, C_p\}$ az osztályhalmaz (p elemű)
S	$\rightarrow$	a fa egy pontjánál a mintahalmaz

A fentiek alapján bevezetjük még a $V_1, V_2, \cdots, V_p, C_k$ $m+1$ elemű jelsorozatot, mely segítségével minden S-beli elem leírható, és ahol $V_i \in R(A_i)$, $i = 1, 2, \cdots, m$, és $C_k \in C$ az elem osztályértéke.

- Az attribútum-választó szabály:
  A fentiek alapján $|C_k|_S$ kifejezés az C_k osztályértékkel rendelkező S halmazbeli elemek számossága. Ekkor az információ mértéke:
  $$I(S) = -\sum_{k=1}^{p} \frac{|C_k|_S}{|S|} \log_2 \frac{|C_k|_S}{|S|}$$
  Az algoritmus minden olyan a_i attribútumra, mely több, mint 1 értéket vehet fel S elemeiben az S halamazt $R(A_i)$ darab S_j részhalmazra bontja, ahol $S_j = \{e \in S| \quad ha \quad e - re: \quad A_i = V_j\}$, vagyis S_j minden olyan elemét tartalmazza S-nek, melyre e elem A_i attribútuma a $V_j \in R(A_i)$ értéket veszi fel. A kapott részhalmazok entrópiája:
  $$E(A_i, S) = \sum_{V_j \in R(A_i)} \frac{|S_j|}{|S|} I(S_j)$$
  A szabály azt az A_i attribútumot választja, melyre
  $$Gain(A_i, S) = I(S) - E(A_i, S)$$
  értéke a maximumot adja az összes S-beli attribútum közül.

- A továbbontó szabály:
  Az attribútum-kiválasztó szabály által kiválasztott attribútum értékei szerint részhalmazokat képzünk a mintahalmazból, a kiválasztott attribútumot kihagyjuk a részhalmazok elemeiből.

- A befejező szabály:
  Ha a mintahalmaz azonos osztályértékű elemeket tartalmaz, akkor a döntési fát nem bővítjük tovább, az ágat levélnek nevezzük.

- Az osztályozó szabály:
  Egy levélhez a hozzá tartozó mintahalmaz elemeinek osztályértékét rendeljük.

Az ID3-algoritmus nem képes visszalépni, mohó módon történik a választás, így nem garantált, hogy a globális optimumot megtalálja. A gyakorlat azonban azt mutatja, hogy az algoritmus számos esetben jó megoldást ad.

4.3.1. Egy példa az ID3 működésére

A következőkben Quinlan példáját ismertetjük. Veszünk egy embereket leíró adathalmazt, ahol minden embernek szerepeltetjük a magasságát (magas, alacsony), hajszínét (sötét, vörös, szőke), illetve szemszínét (kék, barna), valamint mindegyiket a „+” (pozitív) vagy a „-” (negatív) osztályba soroljuk. Az S halmaz a következő elemekből áll a magasság, hajszín, szemszín értékei szerint:

	magasság	hajszín	szemszín	osztály
1.	alacsony	szőke	kék	+
2.	magas	szőke	barna	-
3.	magas	vörös	kék	+
4.	alacsony	sötét	kék	-
5.	magas	sötét	kék	-
6.	magas	szőke	kék	+
7.	magas	sötét	barna	-
8.	alacsony	szőke	barna	-

Látható, hogy a befejező szabály szerint tovább kell még bontani a mintahalmazt, hiszen többféle osztályértékű elem van S-ben. Az attribútum-kiválasztó szabály alapján:

$$I(S) = -\frac{|C_+|}{|S|}\log_2\frac{|C_+|}{|S|} + -\frac{|C_-|}{|S|}\log_2\frac{|C_-|}{|S|} =$$

$$= -\frac{3}{8}\log_2\frac{3}{8} + -\frac{5}{8}\log_2\frac{5}{8} = 0,954434$$

A magasság attribútum vizsgálatához szükségesek a következő információmértékek:

$$I(S_{magas}) = -\frac{|C_+|_{S_{magas}}}{|S_{magas}|}\log_2\frac{|C_+|_{S_{magas}}}{|S_{magas}|} + -\frac{|C_-|_{S_{magas}}}{|S_{magas}|}\log_2\frac{|C_-|_{S_{magas}}}{|S_{magas}|} =$$

$$= -\frac{2}{5}\log_2\frac{2}{5} + -\frac{3}{5}\log_2\frac{3}{5} = 0,970950$$

$$I(S_{alacsony}) = -\frac{|C_+|_{S_{alacsony}}}{|S_{alacsony}|}\log_2\frac{|C_+|_{S_{alacsony}}}{|S_{alacsony}|} + -\frac{|C_-|_{S_{alacsony}}}{|S_{alacsony}|}\log_2\frac{|C_-|_{S_{alacsony}}}{|S_{alacsony}|} =$$

$$= -\frac{1}{3}\log_2\frac{1}{3} + -\frac{2}{3}\log_2\frac{2}{3} = 0,918295$$

Ezeket az értékeket behelyettesítve:

$$E(magasság, S) = \frac{|S_{magas}|}{|S|}I(S_{magas}) + \frac{|S_{alacsony}|}{|S|}I(S_{alacsony}) =$$

$$= \frac{5}{8} \quad 0,970950 + \frac{3}{8} \quad 0,918295 = 0,951204$$

Vagyis

$$Gain(magasság, S) = I(S) - E(magasság, S) = 0,954434 - 0,951204 = \underline{0,003230}$$

Viszgáljuk meg a hajszín attribútumot!

$$I(S_{sötét}) = -\frac{|C_+|_{S_{sötét}}}{|S_{sötét}|}\log_2\frac{|C_+|_{S_{sötét}}}{|S_{sötét}|} + -\frac{|C_-|_{S_{sötét}}}{|S_{sötét}|}\log_2\frac{|C_-|_{S_{sötét}}}{|S_{sötét}|} =$$

$$= -\frac{0}{3}\log_2\frac{0}{3} + -\frac{3}{3}\log_2\frac{3}{3} = 0$$

$$I(S_{vörös}) = -\frac{|C_+|_{S_{vörös}}}{|S_{vörös}|}\log_2\frac{|C_+|_{S_{vörös}}}{|S_{vörös}|} + -\frac{|C_-|_{S_{vörös}}}{|S_{vörös}|}\log_2\frac{|C_-|_{S_{vörös}}}{|S_{vörös}|} =$$

$$= -\frac{1}{1}\log_2\frac{1}{1} + -\frac{0}{1}\log_2\frac{0}{1} = 0$$

$$I(S_{szõke}) = -\frac{|C_+|_{S_{szõke}}}{|S_{szõke}|}\log_2\frac{|C_+|_{S_{szõke}}}{|S_{szõke}|} + -\frac{|C_-|_{S_{szõke}}}{|S_{szõke}|}\log_2\frac{|C_-|_{S_{szõke}}}{|S_{szõke}|} =$$

$$= -\frac{2}{4}\log_2\frac{2}{4} + -\frac{2}{4}\log_2\frac{2}{4} = 1$$

Ezeket az értékeket behelyettesítve:

$$E(haj, S) = \frac{|S_{sötét}|}{|S|}I(S_{sötét}) + \frac{|S_{vörös}|}{|S|}I(S_{vörös}) + \frac{|S_{szõke}|}{|S|}I(S_{szõke} =$$

$$= \frac{3}{8} \quad 0 + \frac{1}{8} \quad 0 + \frac{4}{8} \quad 1 = 0,5$$

$$Gain(haj, S) = I(S) - E(haj, S) = 0,954434 - 0,5 = \underline{0,454434}$$

A szem attribútumra számolandó értékek:

$$I(S_{kék}) = -\frac{|C_+|_{S_{kék}}}{|S_{kék}|}\log_2\frac{|C_+|_{S_{kék}}}{|S_{kék}|} + -\frac{|C_-|_{S_{kék}}}{|S_{kék}|}\log_2\frac{|C_-|_{S_{kék}}}{|S_{kék}|} =$$

$$= -\frac{3}{5}\log_2\frac{3}{5} + -\frac{2}{5}\log_2\frac{2}{5} = 0,970950$$

$$I(S_{barna}) = -\frac{|C_+|_{S_{barna}}}{|S_{barna}|}\log_2\frac{|C_+|_{S_{barna}}}{|S_{barna}|} + -\frac{|C_-|_{S_{barna}}}{|S_{barna}|}\log_2\frac{|C_-|_{S_{barna}}}{|S_{barna}|} =$$

$$= -\frac{0}{3}\log_2\frac{0}{3} + -\frac{3}{3}\log_2\frac{3}{3} = 0$$

Ezeket az értékeket behelyettesítve:

$$E(szem, S) = \frac{|S_{kék}|}{|S|}I(S_{kék}) + \frac{|S_{barna}|}{|S|}I(S_{barna}) =$$

$$= \frac{5}{8}0,970950 + \frac{3}{8}0 = 0,606843$$

Tehát:

$$Gain(szem, S) = I(S) - E(szem, S) = 0,954434 - 0,606843 = \underline{0,347590}$$

A számítások alapján az attribútum-kiválasztó szabály a haj attribútumot választja ki, mivel annak a legmagasabb a *Gain* értéke ($0,454434 > 0,347590 > 0,003230$). A továbbontó szabály a haj értékeinek megfelelően 3 részhalmazt képez:

sötét: alacsony, kék - magas, barna - magas, kék -
vörös: magas, kék +
szőke: alacsony, kék + magas, barna - alacsony, barna - magas, kék +

Rekurzívan a sötét ágra meghívva az algoritmus befejező szabálya nem bontja tovább a részhalmazt, ehhez a levélhez az osztályozó szabály szerint a „-” osztályértéket rendeli, majd visszalép. Hasonlóan a vörös ágra, de ott a „+” osztályértéket kapja a levél. A szőke ág viszont további attribútumbontást igényel. A szükséges számítások:

$$I(S) = -\frac{|C_+|_S}{|S|}\log_2\frac{|C_+|_S}{|S|} + -\frac{|C_-|_S}{|S|}\log_2\frac{|C_-|_S}{|S|} =$$

$$= -\frac{2}{4}\log_2\frac{2}{4} + -\frac{2}{4}\log_2\frac{2}{4} = 1$$

A magasság attribútum vizsgálatához szükségesek a következő információmértékek:

$$I(S_{magas}) = -\frac{|C_+|_{S_{magas}}}{|S_{magas}|}\log_2\frac{|C_+|_{S_{magas}}}{|S_{magas}|} + -\frac{|C_-|_{S_{magas}}}{|S_{magas}|}\log_2\frac{|C_-|_{S_{magas}}}{|S_{magas}|} =$$

$$= -\frac{1}{2}\log_2\frac{1}{2} + -\frac{1}{2}\log_2\frac{1}{2} = 1$$

$$I(S_{alacsony}) = -\frac{|C_+|_{S_{alacsony}}}{|S_{alacsony}|}\log_2\frac{|C_+|_{S_{alacsony}}}{|S_{alacsony}|} + -\frac{|C_-|_{S_{alacsony}}}{|S_{alacsony}|}\log_2\frac{|C_-|_{S_{alacsony}}}{|S_{alacsony}|} =$$

$$= -\frac{1}{2}\log_2\frac{1}{2} + -\frac{1}{2}\log_2\frac{1}{2} = 1$$

Ezeket az értékeket behelyettesítve:

$$E(magasság, S) = \frac{|S_{magas}|}{|S|} I(S_{magas}) + \frac{|S_{alacsony}|}{|S|} I(S_{alacsony}) =$$

$$= \frac{2}{4} \quad 1 + \frac{2}{4} \quad 1 = 1$$

Vagyis

$$Gain(magasság, S) = I(S) - E(magasság, S) = 1 - 1 = \underline{0}$$

Ez azt jelenti, hogy ezen a halmazon a magasság attribútumnak semmi információtartalma sincs. A szem attribútumra a számolandó értékek:

$$I(S_{kék} = -\frac{|C_+|_{S_{kék}}}{|S_{kék}|} \log_2 \frac{|C_+|_{S_{kék}}}{|S_{kék}|} + -\frac{|C_-|_{S_{kék}}}{|S_{kék}|} \log_2 \frac{|C_-|_{S_{kék}}}{|S_{kék}|} =$$

$$= -\frac{2}{2} \log_2 \frac{2}{2} + -\frac{0}{2} \log_2 \frac{0}{2} = 0$$

$$I(S_{barna} = -\frac{|C_+|_{S_{barna}}}{|S_{barna}|} \log_2 \frac{|C_+|_{S_{barna}}}{|S_{barna}|} + -\frac{|C_-|_{S_{barna}}}{|S_{barna}|} \log_2 \frac{|C_-|_{S_{barna}}}{|S_{barna}|} =$$

$$= -\frac{0}{2} \log_2 \frac{0}{2} + -\frac{2}{2} \log_2 \frac{2}{2} = 0$$

Ezeket az értékeket behelyettesítve:

$$E(szem, S) = \frac{|S_{kék}|}{|S|} I(S_{kék}) + \frac{|S_{barna}|}{|S|} I(S_{barna}) =$$

$$= \frac{2}{4} \quad 0 + \frac{2}{4} \quad 0 = 0$$

Vagyis

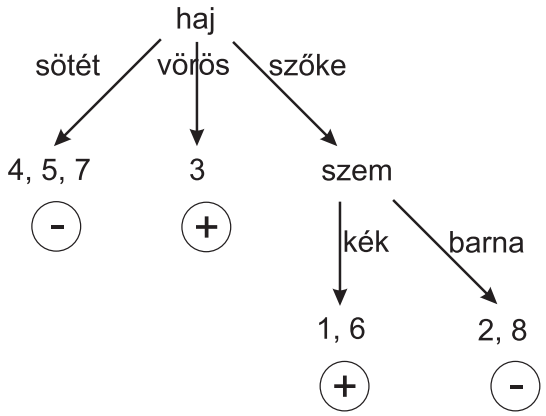
$$Gain(szem, S) = I(S) - E(szem, S) = 1 - 0 = \underline{1}$$

Ez azt jelenti, hogy ennek az attribútumnak a segítségével teljesen szét lehet választani a halamaz elemeit. Így az attribútum-kiválasztó szabály a szem attribútumot választja ki továbbontásra, a továbbontó szabály pedig ennek megfelelően két részhalmazt hoz létre:

kék: alacsony + magas +

barna: magas - alacsony -

Az algoritmust rekurzívan meghívva a részhalmazokra a befejező szabály nem bontja tovább egyiket sem, a kék részhalmazhoz az osztályozó szabály a „+”, a barna részhalmazhoz pedig „-” osztályértéket rendeli. A létrehozott döntési fa a 4.2. ábrán látható. Vegyük észre, hogy a magasság attribútumot egyáltalán nem kell megvizsgálni az elemek osztályozásakor.



4.2. ábra. A létrehozott döntési fa.

Az ID3 algoritmus tulajdonságai

1. Bármilyen ellentmondásmentes példahalmazhoz képes konzisztens hipotézist találni, azaz nem *biased* olyan értelemben, hogy nem szűkíti le eleve a hipotézisteret (restriction bias). A bias a következtetési mechanizmusában van, mivel a kisebb fájú hipotéziseket preferálja (preference bias).

2. Mivel az egyes csúcsokban az attribútumokat mohó módon választja, ezért nem képes visszalépésre, és nem garantált, hogy globálisan optimális fát talál.

3. Nem érzékeny a zajokra, ugyanis az egyes csúcsokban statisztikai alapon (az oda eső példák alapján) választ attribútumot. Mivel a fa építése során egyre lentebb haladva egyre kevesebb az egy csúcsra eső példa, ezért a zaj csak a fa alján fog zavart okozni. Ezt csökkentheti a fa csonkolása (lásd később). Vegyük továbbá észre, hogy az ID3 algoritmus az ellentmondó példákat is képes kezelni! (Ekkor többségi alapon hoz döntést).

4. A tanulás eredménye emberi szemmel is értelmezhető, például a fát IF-THEN szabályokká alakítva.

5. Kiterjeszthető úgy az algoritmus, hogy hiányzó attribútumok esetén is működjön.

A döntési fa tanulás kiterjesztése más feladatokra

a.) Kiterjesztés osztályozásra
Ekkor a fa levelein +,- helyett osztálycímkék lesznek. Az algoritmus kiterjesztése nagyon egyszerű, ugyanis az entrópia több érték esetére is definiált, ahol P tetszőleges diszkrét valószínűségi eloszlás a p_i értékekkel.

b.) Kiterjesztés folytonos változókra
Egy folytonos változó értékkészletét $<$, $>$ feltételekkel intervallumokra bonthatjuk. Ily módon a folytonos változók elvileg kezelhetők. A tanulóalgoritmus azonban többféle módosítást igényel (pl. az intervallumok automatikus kialakítására a példák alapján)
Megjegyzés: Folytonos térben a módszer tengelyekkel párhuzamos téglalapok uniójával osztályoz.

c.) Kiterjesztés regresszióra

A folytonos értékkészletű függvények tanulásának leggyakoribb esete valamely valószínűségi eloszlás tanulása. Leggyakrabban azt várjuk a döntési fától, hogy ne csak egyetlen osztálycímkét adjon vissza, hanem hogy az egyes osztályokhoz valószínűségeket rendeljen. Ez megoldható az alap ID3-algoritmus következő módosításával: Ahol eddig a leggyakoribb osztály-címkét adta vissza, ott e helyett az osztályétékek arányát adja vissza. (Ilyen döntési fák esetében gyakran a CART elnevezést használják.)

A döntési fa vágása (pruning)

Ellentmondásmentes példák esetében az ID3 algoritmus addig fut, amíg a befejező szabály alapján minden ág levél nem lesz. Kevés tanulópélda esetén ilyenkor könnyen előállhat egy olyan helyzet, melyben a levelek egy része csak egyetlen példa alapján kap címkét. Ennek eredményeként az alábbi problémák léphetnek fel:

- Túltanulás fellépése, vagyis a rendszer túlzottan precízen betanulta a tanulópéldákat, ennek következtében az általánosítási képesség roomlik.
- Zajérzékenység fellépése, mivel a levélcímke egyetlen, esetleg hibás példán alapszik.

Megoldást nyújthat a fenti problémákra, ha nem engedjük, hogy a döntési fa ennyire részletesen szétossza a tanulópéldákat, és az egyes ágakat magasabban elvágjuk. Ennek megvalósítására alapvetően kétféle módszer létezik:

- A faépítés korai leállítása.
- A fa utólagos csonkolása (ez a módszer a gyakoribb).

A döntési fák csonkolása

- Csonkolás validációs példaszett segítségével:
  Az utólagos csonkoláshoz az a legjobb, ha rendelkezésünkre áll egy validációs készlet, amelyen megbecsülhetjük a fa általánosítási képességét. A legegyszerűbb módszer, ha iteratívan eltávolítjuk (alulról felfelé) azokat a csúcsokat, amelyek kidobása a legnagyobb mértékben csökkenti a hibát a validációs szetten. Ezt addig végezzük, amíg a hiba nőni nem kezd.
- Csonkolás validációs példák nélkül:
  Például statisztikai vagy komplexitási feltételek alapján végezzük a csonkolást.

A döntési fák alkalmazási területei

- Hadiipar: A döntési fa gyorsabban tud válaszolni az adott kérdésekre, mint az emberi reakcióidő.

- Közgazdasági döntések: például vannak múltra vonatkozó döntési szituációk (adatok). Akkor a múltra vonatkozóan meg tudjuk mondani, hogy mikor kellet volna részvényt venni vagy eladni. Az osztályérték visszamenőleg megadható. Így az ID3-algoritmus alapján el tudjuk készíteni, hogy a múlttal konzisztens döntésekben milyen döntési fát kell konstruálni. Ha megvan ez a döntési fa, akkor az adott szituációban, a döntési fával ki tudjuk számolni, hogy hogyan kell dönteni. Fontos, hogy minden releváns információnak benne kell lennie. Például a politikai események is meghatározzák a közgazdasági folyamatokat.

5. fejezet

Adatbányászat és adatvizualizáció

5.1. Bevezetés

A világ adatmennyisége évente nagyjából megduplázódik, ezért egyre nehezebb megtalálni azokat az adatokat, melyekre szükség van. Sokszor az adatok önmagukban nem igazán hasznosak, csak a belőlük kinyerhető információ az. Az adatbázisok mérete több gigabájtos is lehet, amelyhez például az SQL lekérdező nyelv nem elég, hogy kinyerjük a hasznos információt (a szükséges információ nem elérhető egy egyszerű SELECT lekérdezés formájában). Így természetes az igény egy olyan eszköz iránt, amely nagy mennyiségű nyers adat elemzésére képes. Ez az eszköz az adatbányászat, mely az 1990-es években bontakozott ki. Az adatbányászat tehát „nyers" adatokból olyan információkat, illetve ismereteket állít elő, melyek lehetnek korábban nem ismert kapcsolatok, szabályok, esetek, ok-okozati összefüggések stb., és amelyek nagyban segíthetik, javíthatják az üzleti döntések eredményességét. Az adatbányászat tehát azzal foglalkozik, hogy hogyan lehet nagy adatbázisokban rejtett tudást, új összefüggéseket, szabályokat, mintákat találni.

5.2. Adatbányászat

Egy üzleti vállalkozás megfelelő működéséhez elengedhetetlen a döntések meghozatalát elősegítő információk megszerzése. Ilyen információ lehet például a vállalkozás szolgáltatásait igénybe vevő vásárlók életkora, a lefolytatott reklámkampány eredményessége, a bankhitelt igénylők családi állapota. Az ilyen információk megszerzéséhez hatalmas adatbázisok állnak rendelkezésre. Azonban ezek pusztán adatkezelési (pl. SQL) és statisztikai eszközökkel történő feldolgozása csak a „felszínt" tudja elérni, amely nem elégséges. Olyan automatikus eszközökre van szükség, melyek az adatbázisokban rejlő „mélyebb", rejtett kapcsolatok felkutatására képesek.

Az adatbányászatról, mint mesterséges intelligencia önálló területéről nem lehet beszélni. Ugyanis az adatbányászat az összes előzőleg tárgyalt tanuló módszert, a neurális hálózatokat, genetikus algoritmust, döntési fákat, keresési eljárásokat, klaszterzéseket, egyszóval a mesterséges intelligencia teljes fegyvertárát használhatja és ebben az értelemben a mesterséges intelligencia szinte minden része gyakorlati alkalmazást nyer. Így azt is mondhatnánk, hogy nincs mit írni róla. Mégis az adatbanyászat speciális feladatának

megértésével lehet csak az algoritmusokat jól kiválasztani. Ezért az első részben a problémát vázoljuk, megemlítve az alkalmazható eljárásokat. Mivel az algoritmusok alkalmazása önmagában nem nyújt megfelelő megoldást, a második részben részletesen tárgyaljuk a gyakorlatban használatos adatbányászati eszközökkel egyenrangú fontosságú fejlett grafikus felületet, amely a sokszempontú lekérdezést és döntéshozást támogatja.

5.2.1. Az információs túlterhelés

Az információs társadalom létrehozása szinte minden fórumon elsődleges téma. A hálózatok robbanásszerű terjedése miatt az információs korszak kezdetének nevezik korunkat, ugyanis a lokálisan rendelkezésre álló információk a hálózatok segítségével mindenütt elérhetővé válnak. A globális információáramlásnak a felállított biztonsági eljárások sem tudják útját állni. Mindenesetre a hálózaton elérhető információk olyan gazdagságával állunk szemben, ami a kezelhetőség szempontjából új problémákat vet fel. Ha ehhez hozzávesszük a nem ingyenes adatszolgáltatók által nyújtott információ-özönt, tényleg elveszett embernek hihetjük magunkat. Az információt nemcsak úgy lehet elrejteni, ha titkosítjuk, hanem ha mértéktelen mennyiségben áll rendelkezésre adat.

Egy történettel megvilágítva a fentieket, tegyük fel, hogy a király titkos üzenetet akar elküldeni. Ekkor folyamodhat ahhoz, hogy rébuszokban beszélve küldöncével egy étel elkészítésének módját taglalva adja tudtára tervét: "Ne bárányból készítsd az ebédet, elég, ha kappannal fogadod vendéged, amit kellőképpen fűszerezel és hozzá savanyúság helyett kompótot tálalsz ...". Az ellenfélnek - a küldöncöt elfogva és megtalálva az üzenetet - csak arra kell koncentrálnia, hogy a megfelelő értelmezést megadja. A helyzetet úgy jellemezhetnénk, hogy a küldönc receptek tucatjait esetleg egy szakácskönyvet visz magával és csak azt az információt tartja a fejében, hogy melyik receptet kell átadnia a király bizalmasának. Az ellenség megkaparintva a recepteket, szinte reménytelen helyzetbe kerül a megfejtést illetően.

A mai helyzet szerint a példában szereplő mennyiségeket nagyságrendekkel megnövelhetnénk. A király üzenete esetünkben valamilyen törvényszerűség (társadalmi, pl. fogyasztói szokás), aminek megfejtése a feladat. Az „adatbányászat” szó azért terjedt el, mivel a valódi bányászat esetén is az érték csak töredéke a megmozgatott anyag mennyiségének. Először is össze kell gyűjteni azokat az anyagokat, amikből kinyerhető az érték. Erre a fázisra jellemző a hálózatok építése, illetve az adatáruházak létrehozása. Ma mindenki az adatok felhalmozásának szükségességéről beszél és csak másodlagosnak tekintik a feltáró folyamatot, aminek az a veszélye, hogy a nem–feladatorintált megközelítés sok felesleges munkát eredményez. A felhalmozás még csak az előkészületi fázis, ettől még nem lesz meg az adatbányászat eredménye: az ásvány illetve a „kincs”. A gazdasági haszon miatt nem egyetlen ember, vagy szervezet indul az információ felderítésének, valamint megszerzésének versenyében.

5.2.2. Az adatelemzés fontossága

Az adatbányászat nem csak a kecsegtető haszon miatt fontos, hanem egyszerűen kényszer is, mert a világban felgyorsult folyamatokra adatbányászati eszközök használata nélkül a szervezetek már nem tudnak

megfelelően reagálni. Ismét hasonlattal élve olyan ez, mint amikor az utcán közlekedő autók sebességkorlátozása megszűnik és a gyalogosok közlekedése lehetetlenné válik, vagy mint amikor a számítógépes játékoknál a képernyőn mozgó objektumok sebességét többszörösére növeljük és nem lehetséges hatékony beavatkozás. Valahogy így kell elképzelni a társadalmi folyamatok felgyorsulása közepette való létünket is.

Az adatok közötti tájékozódás szinte minden vállalat és szervezet számára létszükségletté válik. Nemcsak a profit nagysága függ ettől, hanem sokkal több, az életben maradásé. Szükségük van szolgáltatásaik, termékeik, piacuk, pénzügyi helyzetük pontos meghatározására nemcsak lokális környezetükben, hanem sokkal szélesebb körben, a globalizálódó világban is. (Tipikusan ilyen feladat a termékmenedzseré, a vásárlói szokásokat feltáró elemzőé, a piackutatóé, a gazdasági stratégiát meghatározóé, stb.) Nemcsak a törvényszerűségeket kell feltárni, a vezető számára néha sokkal fontosabb a rendellenességek megtalálása a folyamatok megértése. A nagy szervezetek létben való fenyegetettsége azt is jelenti, hogy statisztikusokat alkalmaznak adatelemzés céljából, akik a statisztikához ugyan értenek, de a vállalat irányításához nem, és alkalmazásuk csak részben oldja meg a problémát. Az eredmények értelmezésében ez mindig gondot okoz, nem beszélve arról, hogy bizonyos fontos diszciplínák nem is tartoznak a statisztika területéhez, így például a tanulás, neurális hálók döntési fák stb. alkalmazása.

Az európai szervezetek (de nemcsak az európaiak) eddig képtelenek voltak felvenni a versenyt a kihívásokkal. Új típusú vezetői szemlélet kell ahhoz, hogy a helyzetet meg lehessen menteni és - a következő generáció színrelépéséig - a középvezető rétegnek kell ezt a problémát megoldani. ők csak közvetve válnak felelőssé a döntésért azzal, hogy elvégzik a döntési lehetőségek feltárását.

Az adatbányászat eszköz az információ-rengetegben való tájékozódáshoz. Segítségével gyorsabb, jobb javaslatok készíthetők elő és segíti az ügyintézést is. A legnagyobb probléma, hogy az alkalmazók nem adatelemző specialisták, ezért a jelenleg rendelkezésre álló eszközök alkalmatlanok számukra. Mint ahogy a rádióhallgatónak sem kell ismernie a rádió működését, ahhoz hogy használja, ezért arra van szükség, hogy elkészüljenek a felhasználót messzemenően figyelembe vevő újszerű programok, lehetőleg azt a fejlődési fázist is kihagyva, amikor az eszköz jóságát annak bonyolultsága igazolta. Jól ismert tény, hogy az egyszerűség marketing szempontból jelentős előny. (Az automata fényképezőgép sokkal nagyobb piaci szegmenst birtokol, mint a professzionális gépek.)

5.2.3. Mi az adatbányászat?

Az adatbányászat, mint önálló diszciplína a statisztika mellett úgy jöhetett létre, hogy könnyen használható elemző eszközöket kellett biztosítania az üzleti szakértőknek, a piackutatóknak, a gazdasági és stratégiai tervezésben érdekelteknek. Ezeknek az eszközöknek a közös lényegi vonása, hogy használóikat segítik az adatelemzésben és az adatelemzés részletei helyett inkább az üzleti problémákra lehet koncentrálni.

Az adatáruházak manapság az információfeldolgozás összes formájával foglalkoznak, különösen nagy hangsúlyt fektetve az adatbányászat irányzatára.

Az adatbányászatot nem könnyű definiálni: régebben tudás-kezelésnek (knowledge management), vagy tudás-technológiának nevezték. Az adatbányászat általánosan elfogadott definíciója: ismeretlen minták és

összefüggések keresési folyamata a teljes adatbázis alapján. A régi keresési módszerek, amelyek az adatbázis alapján bizonyos kritériumnak való megfelelést vizsgálnak (speciális tényre vagy tényekre vonatkoznak), nem azonosak a mai adatbányászatéval. Az adatbányászat eljárása sokkal inkább hasonlít egy törvényszéki nyomozó tevékenységéhez, aki minden lényegesnek tűnő tényt megvizsgál és keresi az összefüggő motívumokat, hogy felderítse a bűncselekményt. Nem használ lekérdező nyelvet a speciális adatok keresésére, hanem inkább megvizsgálja az összes tényezőt, hogy kiderítse, van-e olyan motívum, vagy összefüggés, aminek értelme (jelentése) van.

A bányászat helyett tehát a nyomozás, felderítés legalább olyan jó szó, mert a tevékenységre utal. Az adatbányászat feladata olyan eszközrendszer összeállítása, kifejlesztése, ami segíti a feltáró folyamatot. Az adatbányászat olyan keresési módszert alkalmaz, amellyel kialakul a minták egy bizonyos sorrendje. Az adatbányászat software-termékei speciális eszközök, amelyekkel a felvetett kérdésekre adhatunk választ azzal, hogy lehetővé tesszük a felhasználóknak, hogy kereső eljárásokat hajtsanak végre. Az adatbányászat így tartalmazza a lekérdező (SQL) nyelvek fejlesztését is.

Az adatbányászat során a nagy adatbázisokból olyan kapcsolatokat, motívumokat és jellegzetességeket keresnek, amelyekről előzően nem tudták, hogy léteznek, vagy nem is voltak láthatóak. A megtalált kapcsolatokat ugyan elfogadhatták a szakemberek vagy értékesítők, de mielőtt alkalmaznák, először ki kellett próbálni, esetleg pontosítani, finomítani kellett őket.

Az adatbányászat eredménye olyan új információ vagy tudás, amely lehetővé teszi a felhasználó közösségeknek, hogy hatékonyabbak legyenek. Az adatbányászat nehézsége, hogy néhány összefüggő tény feltárása miatt hatalmas adatbázist kell feldolgozni. Mint ahogy nincs két egyforma bűnügy, úgy ugyanazt az algoritmust és keresési kritériumot, amit egyszer használtunk, valószínűleg nem lehet már újra pontosan ugyanolyan módon használni. Összefoglalva: az adatbányászat egy olyan információ feldolgozó eljárás, amely megmutatja a válaszokat azokra a kérdésekre is, amelyeket gyakran még fel sem tudunk tenni. Ahelyett, hogy egy relációs adatbázisnak hagyományos lekérdező nyelven azt mondanánk: "Menj és keresd meg azokat az embereket, akik ebben az évben ablakredőnyt vásároltak és valamivel később ágyneműt is vettek!", az adatbányászatban a kérdés így hangzik: "Találd meg az összefüggő vásárlási mintákat!". A válasz pedig: "van egy minta, ami az idő x százalékában jelenik meg, mégpedig akkor, ha valaki ablakhoz szükséges alkatrészeket vásárol (nemcsak redőnyt) és 1-3 hónapon belül ágyneműt is vásárol, a következő négy hónapon belül még bútort is vesz". Ha egy speciális összefüggésre kérdezünk rá, akkor pontos, de használhatatlan információt kapunk. Ha viszont olyan kapcsolatokra kérdezünk, amelyeknek a létezéséről még nincs tudomásunk, sokkal jelentősebb összfüggésre találhatunk, aminek üzleti értéke van. Számos technológiát kell alkalmazni ahhoz, hogy az adatbányászat működőképes legyen.

Először is, az egyik legfontosabb feladat egy adatáruház létrehozásának vállalása. Az adatbányászatnak képesnek kell lennie az összes adat megvizsgálására, amihez egy adatraktárat kell készíteni. Az adatbázisok adatáruházzá való átalakítása csak az első, kezdő lépésnek lehet tekinteni az adatbányászat megvalósításában. Másodszor, léteznie kell egy jegyzéknek az adatbázisok tartalmáról, azaz egy metainformációs rendszert is létre kell hozni. Ez azért szükséges, hogy a használók (elemzők) tudják milyen

adatok állhatnak rendelkezésükre. Az olyan információs rendszer, ami az üzleti adatoknak csak egy részét teszi elérhetővé, valójában értéktelen. Az adatok hatékony feldolgozásához a legkülönbözőbb forrásból származó adatokat is ismernünk kell. Az adatbányászati eszközöknek ténylegesen képesnek kell lenniük az adatáruházi és bármilyen más, például a szerveren szétszórt adat megkeresésére.

Harmadszor, olyan eszközökkel kell rendelkezni, amelyek kivitelezhetővé teszik az adatbányászati technológiát. Számos eszköz áll rendelkezésre, amelyek különböző kategóriákba sorolhatók pl. a legközelebbi szomszéd algoritmus, a döntés-fák és az adatok vizualizációja. Néhány eszköz az ipar sajátos területeire specializálódik, mint például a pénzügyre vagy a biztosításra, kihasználva a sajátos üzleti modelleket és speciális, jól meghatározott összefüggéseket keresve.

Az általános adatbányászati eszközök is kezdenek megjelenni, amelyek ugyan nem speciális üzleti ágakra irányulnak, de meghatározott összefüggéseket keresnek. A neurális hálózaton alapuló technikák rendkívül hasznosnak bizonyulnak. A humán kérdésfelvetést meghaladó eljárás jött létre alkalmazásukkal, ami csupán a tanulás sikerességére koncentrál. De ebbe az osztályba tartozik a döntési fák konstruálása is táblázat alapján. Az adatbányászatban ezen technikák megvalósítása egy új információfeldolgozási eljárást jelent. A SQL és a statisztika elavultnak tűnik hatékonyságuk tükrében.

5.2.4. Adatbányászati technológiák

Az adatbányászat szempontjából a relációs adatbázisok jelentik a kiinduló pontot. Az adatbányászat sikere részben a már meglévő kapcsolatoktól függ, újabb kapcsolatok pedig könnyen beépíthetők. Az adatbázisokból visszanyerhető információ azonban nem szükségszerűen az adatbázis szerkezetéből származik, hanem az összefüggések elemzéséből. A tények felderítésére két különböző technikát alkalmaznak, a származtatást és a következtetést (dedukció, indukció).

A dedukciós módszerek általában az összes relációs adatbáziskezelő rendszerben rendelkezésre állnak (DBMS), míg az induktív módszerek nem használhatók fel közvetlenül. A dedukció mindig olyan információkat ad, amelyeknek be lehet bizonyítani a tényszerűségét, míg az indukció olyanokat, amelyekről elfogadható, hogy valamilyen valószínűséggel igazak, de nem szükségszerűen bizonyíthatóak. Az adatbányászatban induktív módszereket kell alkalmazni. Nagy mennyiségű adatot kell megvizsgálni, hogy eljussunk az eredményekhez, amit csak új technológiák alkalmazásával lehet végrehajtani. Ezek a technológiák részben a relációs DBMS rendszerek továbbfejlesztéseként állnak rendelkezésre. Az adatok típusától függően különböző eljárások léteznek. A legfőbb probléma az, hogy a hagyományos SQL módszerek nem használhatók adatbányászatra jelenlegi korlátaik következtében és így teljesen új fejlesztések szükségesek ennek a folyamatnak a végrehajtásához.

A feltárás során - az adatbányászat eszközeinek és módszereinek alkalmazásával - először megfelelő adatelemzésekkel leszűkítik a vizsgált adathalmazt, majd ezután történik a tényleges feltáró munka. Az adatbányászat egy részletesen kidolgozott folyamatnak, az Adatbázisok Tudásfeltárásának (Knowledge Discovery in Databases - KDD) az egyik kulcsfontosságú eleme. Az Adatbázisok Tudásfeltárása az alábbi fázisokra osztható:

1. Előkészítő fázis: az alkalmazási terület feltárása, előzetes ismeretek és igények begyűjtése, célok kijelölése.

2. Adatkiválasztás: ennek során hozzák létre a céladatbázist a felhasználandó adatbázisokból.

3. Adattisztítás: téves bejegyzések eltávolítása, hiányos mezők pótlása, stb.

4. Adattér meghatározása: egyrészt az adatbázisból csak a cél szempontjából érdekes attribútumokat emeli ki, másrészt további attribútumok szerezhetők más táblákból (pl. kiegészítő demográfiai adatok vásárlása).

5. Adattranszformáció: az adatok jobb összehasonlíthatósága érdekében szükséges lehet az adatok transzformációja. Ilyen transzformáció lehet például karakteres címek helyettesítése területi kódokkal, születési dátum helyett életkor megadása, különféle vetítések végrehajtása, (PCA, LDA), klaszterezés, stb.

6. Tudásbázis létrehozása: ebben tárolható a formalizálható tudás, a különböző paraméterek, küszöbértékek, mutatók, stb.

7. Adatbányászati algoritmus kiválasztása: a minták kinyerésére alkalmazandó algoritmus kiválasztása, elemzése (futásidő, memóriaigény, stb.).

8. Minta kinyerése Az alkalmazó számára értékes minták kinyerése a kiválasztott adatbányászati algoritmus segítségével.

9. Mintakiértékelés: a kinyert információ értelmezése. Minél jobban egybe tudjuk építeni az adatbányászati modult a mintakiértékeléssel, annál hatékonyabb és gyorsabb lehet a tudásfeltárás. Ez a lépés magában foglalhat adat-vizualizációt is a jobb megértés céljából.

10. Jelentés készítése: az algoritmus és a mintakiértékelés eredményeinek elemzése, illetve ezen eredmények további alkalmazása újabb adatokra.

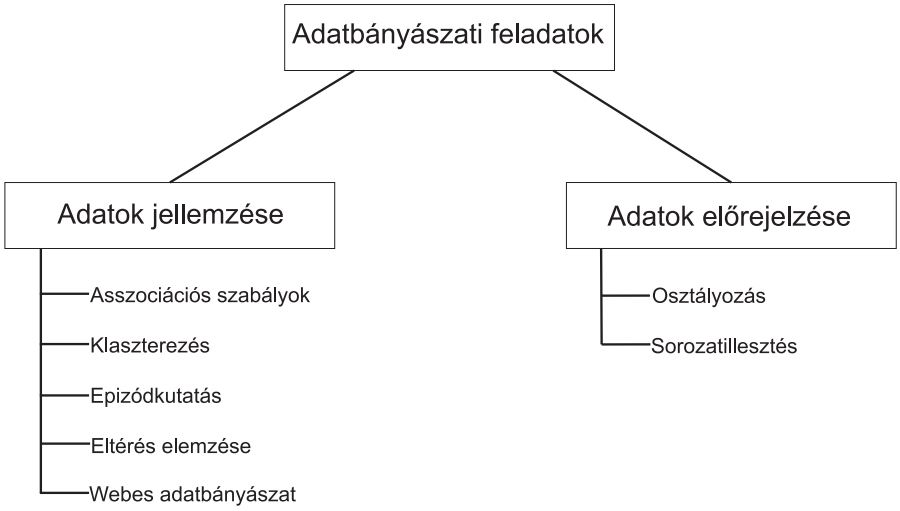
5.2.5. Adatbányászati feladatok csoportosítása

Az adatokból történő tudás kinyerésének alapvetően két célja van:

1. Adatok jellemzése: az adatokat jellemezni szeretnénk a feltárt tulajdonságaik alapján. Itt a cél az adathalmaz általános tulajdonságainak karakterizálása.

2. Adatok előrejelzése: előrejelzéseket, következtetési szabályokat szeretnénk a meglévő adatok alapján megfogalmazni.

Az összefüggések típusa szerint az adatbányászati feladatok az 5.1. ábrán látható csoportokba sorolhatók.



5.1. ábra. Adatbányászati feladatok felosztása.

1. **Adatok jellemzése** - Descriptive Data Mining

a.) Asszociációs szabályok (- association rules)

Ebbe a csoportba tartoznak az „X terméket vásárlók k%-a Y terméket is vásárol" típusú értékelések. Az $X \to Y$ alakú asszociációs szabályok felhasználhatók például áruházak, szupermarketek akcióinak tervezésére (X termék árát csökkentjük 10%-al, Y termékét pedig 20%-al növeljük), vagy nagy áruházak terméktérképeinek kialakításához (X terméket Y termék közelébe célszerű elhelyezni), vagy cégek alkalmazottainak, ügyfélkörének elemzésére, stb.

b.) Klaszterezés (- clustering)

A klaszterezés tulajdonképpen hasonlóságkeresés, melynek alapján az objektumok hasonlóságuk alapján különböző csoportokba sorolhatók. A csoportok tehát nincsenek előre definiálva, ezeket az adatbányászó algoritmusnak kell meghatároznia. Ilyen eljárást használnak például a csillagászatban a galaxisok, csillagok, egyéb égitestek szeparálására.

c.) Epizódkutatás (- time-series analysis)

Hosszú eseménysorozatokban gyakran előforduló elemeket, illetve elemhalmazokat keresnek. Az epizódkutatás az asszociációs szabályokból fejlődött ki. A különbség a kettő között az, hogy amíg az asszociációs szabályok a tranzakciókon belüli összefüggéseket vizsgálják, addig az epizódkutatatás a tranzakciók közötti összefüggéseket elemzi. Felhasználási területei például a direkt marketing (pl. a videomagnót vásárlók 40%-a 2-3 hónap múltán videokamerát is vásárolt, ezért érdemes a videomagnót vásárlóknak videokamera-prospektusokat küldeni), vagy az orvosi diagnosztika (epizódkutatás segítségével meghatározható, hogy egyes betegségeket milyen tünetek, illetve más betegségek előztek meg).

d.) Eltéréselemzés

Azon elemek vizsgálata az adatbázison belül, melyek tulajdonságai eltérnek az általános jellemzőktől. Egyéb adatbányászati módszerek ezeket az adatokat figyelmen kívül hagyják, illetve adattisztítás

során kiszűrik. Jellemző alkalmazási területei például a csalások, visszaélések kiszűrése (biztosítótársaságok, bankok, stb.), illetve vírusok detektálása, stb.

e.) Webes adatbányászat

A legnagyobb adattömeg az Interneten található, így az innen történő információ kinyerése is az adatbányászathoz sorolandó. Felhasználási területei például az intelligens keresés, oldalak rangsorolása, hasonló tartalmú oldalak megtalálása, stb.

2. **Adatok előrejelzése** - predictive data mining

a.) Osztályozás (- classification)

Az osztályozásnál az adatok csoportokba lesznek sorolva. Lényeges különbség azonban a klaszterezéssel szemben, hogy itt az elemek osztályai előre meghatározottak. A cél egy tanulóalgoritmus és egy előre osztályozott adathalmaz segítségével olyan modell létrehozása, melynek segítségével egy nem osztályozott objektumról nagy pontossággal eldönthető, hogy melyik osztályba tartozik. Az osztályozáshoz használt modellek például a döntési fák, neurális hálók, stb. Felhasználási területe például a hitelelbírálás bankoknál, illetve vírusirtó programok, ahol az ismert vírusszignatúrák alapján olyan modelleket állítanak fel, melyek jól leírják a vírusok tulajdonságait, és így egy még nem látott vírust is nagy valószínűséggel ki tudnak szűrni.

b.) Sorozatillesztés

Adott sorozatok egy halmaza. A cél egy új sorozat esetén a hozzá leginkább hasonlító sorozat illetve részsorozat meghatározása. Ez is egyfajta osztályozás.

5.2.6. Adatbányászati algoritmusok csoportosítása

Az adatbányászati eszközöket a feldolgozó algoritmusok és eljárások szerint négy fő típusba lehet sorolni:

- egyesítés vagy összekapcsoló elemzés asszociáció
- sorrendi minták
- csoportosítás (clustering)
- osztályozás

Az egyesítés vagy összekapcsoló elemzés

Az adatbányászati asszociációk általában arra irányulnak, hogy az adatbázisból kikeressék az összes olyan tranzakciót, amelyek nagy valószínűséggel ismétlődnek. A kérdésnek ehhez a típusához egy olyan algoritmus szükséges, amely megtalálja az összes olyan szabályt, ami az események vagy adatok egy készletét összefüggésbe hozza egy másikkal. Az ilyen típusú folyamatra jellemző eredményeket a kiskereskedelmi példák használata mutatja legjobban. Tipikus válasznak tekinthető a következő: Azok közül, akik hordozható számítógépet vásárolnak, 78 % további kiegészítő termékeket is vásárolni fog. Ezt a típusú feldolgozást

minden olyan relációs adatkészleteknél lehet használni, amelyek standard SQL állító logikát használnak. Az asszociatív algoritmusok célja, hogy bővítsék az SQL lehetőségeit. Az algoritmusoknak nagyon alkalmazkodóknak és dinamikusaknak kell lenniük. Szabályok szerint kell megtalálni a mintát, miközben változhat a megvizsgált adatkészlet és ennek megfelelően az asszociációs szabályok ill. a bennük előforduló százalékok is változnak. Ezeket a szabályokat aztán rendezni lehet a gyakoriságok szerint, hogy a felhasználó megtalálja a legjobb lehetséges jelenségeket az üzleti stratégiák és termékelhelyezések szempontjából. Pl. Az élelmiszeráruházakban meghatározó, hogy az italvásárlók nagy százalékban sós süteményt is vásárolnak, ha az üzletben ugyanazon az útvonalon van a két termék. Ma már sokrétű asszociációs algoritmusok és feldolgozó eszközök állnak rendelkezésre, amelyek mind az ipar, mind pedig az üzleti folyamatok területén használatosak. Az általánosított algoritmusok jók ugyan, de legtöbbször nem biztosítják a kívánt pontosságot a versenyképes üzleti gyakorlatban.

Sorrendi minták

A sorrendi minták ugyanazokra az alapadatokra támaszkodnak, mint az asszociatív eljárások, azzal a kiegészítéssel, hogy az adatokat egy meghatározott időtartam szerint kell összegyűjteni, ami a rendszerből kigyűjtött tételek és tranzakciók egy ún. történeti adatkészlete, az eredmény pedig az ismétlődésre legnagyobb valószínűségű mintákat tartalmazza. A sorrendi mintákat az üzleti életben leginkább mintaelemzésre használják. Ez a leggyakoribb példa arra, hogy az adatbányászat hatékonyságát bemutassák. A sorrendi mintákkal az a probléma, hogy nagyon sok használhatalan eredmény keletkezhet. Az események sorrendjében nagy valószínűsége van annak, hogy egy bizonyos idő elteltével az első és a második esemény egy olyan mintát mutat, ahol a harmadik esemény sohasem fordul elő. Az üzletelemző fő feladata a szabályok finomítása az ismétlődő végrehajtások során, beállítva a minimum és maximum százalékos küszöböket. A sorrendiségnek van még egy másik megközelítési módja is, ami az üzleti életre hatással van. Ezt a technikát hasonló sorozatoknak nevezik. A sorrendi mintáknál az eredmény az események időbeli lefolyása. A hasonló sorozatoknál az időbeli események sorrendje hasonlít egy másik sorrendiséghez. Például a kiskereskedelemben olyan üzleteket keresünk, ahol hasonlóak az eladási árképzési stratégiák, vagy olyan raktárakat, amelyeknél hasonló ármozgással dolgoznak.

Csoportosítás (Clustering)

Az adatbányászat során néha még hipotézisünk sincs arra, hogy miképp tegyük fel a kérdést. Ezekben az esetekben csoportosító algoritmust kell használni, hogy felfedezzük a jellegzetesség egy eddig ismeretlen, vagy csak sejtett formáját. A csoportosító példák hiányelemzések, vagy rokoni viszonyban álló csoportelemzések. Vannak eljárások, amelyek olyan csoportot találnak, ami néhány gyakori osztályt bont részekre. A csoportosító folyamatok néhány olyan sajátos eseményen alapulnak, mint pl. amikor egy jó vásárló megszünteti hitelkártyáját. Ebben az esetben a vevő típusát meghatározó szabályok ismeretlenek. Ahhoz, hogy ennek a vevőnek a típusát meghatározzák, a csoportosítás (clustering) képes olyan szabályokat létrehozni, amelyek lehetővé teszik a társaságnak, hogy megelőzze a fent említett hitelkártya törlését. A "clustering"-et

irányítatlan tanulásnak nevezik.

Osztályozás

Az osztályozás, vagy más néven (irányított) tanuló algoritmus, olyan eljárás, amelynél példák alapján kell az algoritmusnak a szabályokat megtalálni. A szabályok alapján pedig az adatállomány bővítése során létrejövő eseményeket lehet jellemezni, ill. események bekövetkezését lehet megjósolni. Az adatbányászatban legfőképpen az üzleti haszon alapján kell meghatározni a szabályokat. Pl. feladat a nyereség meghatározása a vállalat egyéb jellemzőinek felhasználásával, a megoldás pedig a szabályrendszer megalkotása. Az osztályozás a relációs adatbázisban nagyon összetett folyamattá válhat, mert gyakran sok táblázatot és tulajdonságot kell megvizsgálni. Így aztán nincs adott nevük vagy tulajdonságtípusuk sem. A leggyakoribb üzleti példa az osztályozásra a hitelkártya jóváhagyó eljárás. A legnevezetesebbek a perceptron, back-propagation és a döntési fákra (ID3, C4.5) vonatkozó algoritmusok.

5.2.7. Asszociációs szabályok

Mint az előzőekben már szerepelt, ebbe a csoportba tartoznak az „X terméket vásárlók k%-a Y terméket is vásárol” típusú feladatok. Az asszociációs algoritmusok megértéséhez elsőként néhány alapfogalmat kell tisztázni.

Adatbázis ábrázolása

Formálisan az adatbázistábla (jele: **D**) felfogható egy logikai relációs táblának. Az oszlopok (mezők) a különböző termékeket jelölik, a sorok (rekordok) pedig az egyes tranzakciókat, melyet a szakirodalmak *kosárként* (basket) definiálnak. Amennyiben az adott kosár egy terméket tartalmaz, a mező értéke 1, ha nem, akkor 0 lesz. A továbbiakban a termékek halmazát **A** (attributes), a kosarakat pedig **r** jelöli. Az alábbi példában tehát $\mathbf{A} = \{a, b, c, d, e\}$ az attribútumhalmaz.

Kosarak(**r**)	a	b	c	d	e
1	1	0	0	1	1
2	1	0	1	1	0
3	0	1	1	0	0
4	0	1	1	0	1
5	1	0	1	0	1
6	1	1	1	1	0

Az így létrejött logikai táblát fizikailag kétféleképpen rendezhetjük:

1. Horizontálisan: ekkor a kosarakhoz azon termékek halmazát rendeljük, melyek az adott kosarakba bekerültek, vagyis az adatbázisban értékük 1 volt. Az előző táblát felhasználva, az alábbi horizontális elrendezést kapjuk:

r	**cas**
1	*ade*
2	*acd*
3	*bc*
4	*bce*
5	*ace*
6	*abcd*

A táblázatban a **cas** (canonical attribute sequence) az adott kosarakban lévő termékek halmazát jelöli. Tehát a horizontális elrendezés $(\mathbf{r}, \mathbf{cas})$ párok halmazából áll.

2. Vertikálisan: ekkor a termékekhez rendeljük az őket tartalmazó kosarakat. Így a vertikális elrendezés $(\mathbf{a_i}, \mathbf{list})$ párokból áll, ahol $\mathbf{a_i} \in \mathbf{A}$, **list** pedig az $\mathbf{a_i}$-t tartalmazó kosarak (tranzakciók) rendezett listája. A fenti példa alapján az elrendezés a következő:

a	*b*	*c*	*d*	*e*
1	3	2	1	1
2	4	3	2	4
5	6	4	6	5
6		5		
		6		

Az asszociációs algoritmusok általában a vertikális elrendezésből indulnak ki.

Termékhalmaz előfordulása

Legyen $\mathbf{D} \subset \mathbf{N}^+ \times \mathcal{P}(\mathcal{A})$ és $\mathbf{s}, \mathbf{u} \subseteq \mathbf{A}$. Jelölje $\lambda(\mathbf{s})$ az adott $\mathbf{s}$ termékhalmaz előfordulását, ahol

$$\lambda(\mathbf{s}) = \{\mathbf{r} | \mathbf{s} \subseteq \mathbf{u}, (\mathbf{r}, \mathbf{u}) \in \mathbf{D}\}$$

Például:

$$\lambda(a) = \{1, 2, 5, 6\} \quad \lambda(bc) = \{3, 4, 6\}$$
$$\lambda(ab) = \{6\} \quad \lambda(a, b, c, d, e) = \oslash$$

Termékhalmaz támogatottsága (support)

Egy $\mathbf{s} \subseteq \mathbf{A}$ termékhalmaz támogatottsága (support) azon kosarak számának aránya az összes kosárszámhoz ($|\mathbf{D}|$), amelyek tartalmazzák **s**-et.

$$\mathbf{supp}(\mathbf{s}) = \frac{|\lambda(\mathbf{s})|}{|\mathbf{D}|}$$

Például:

$$\mathbf{supp}(a) = \frac{|\{1,2,5,6\}|}{6} = 66,7\% \quad \mathbf{supp}(b) = \frac{|\{6\}|}{6} = 16,7\%$$

Tétel : Legyen $\mathbf{s}, \mathbf{s}' \subseteq \mathbf{A}$. Ha $\mathbf{s}' \subseteq \mathbf{s}$, akkor $\mathbf{sup}(\mathbf{s}') \geq \mathbf{sup}(\mathbf{s})$, tehát bármely termékhalmaz tetszőleges részhalmazának támogatottsága nagyobb vagy egyenlő, mint az eredeti halmaz támogatottsága, mivel minden kosár, mely tartalmaz egy termékhalmazt, tartalmazza annak összes részhalmazát.

Gyakori termékhalmaz

Gyakori termékhalmaznak nevezzük azon termékhalmazokat, melyekre igaz:

$$\mathbf{supp}(\mathbf{s}) \geq \sigma,$$

ahol σ egy előre meghatározott támogatottsági küszöb. A fenti tétel alapján egy gyakori termékhalmaz minden részhalmaza gyakori.

Asszociációs szabály

Legyen:

- $\mathbf{s}, \mathbf{u} \in \mathbf{A}$, úgy hogy $\mathbf{s} \cap \mathbf{u} = \oslash$
- $\lambda(\mathbf{s} \rightarrow \mathbf{u}) = \lambda(\mathbf{s} \cup \mathbf{u})$
- $\sigma = [0, 1]$ és $\gamma = [0, 1]$, ahol σ a támogatottsági küszöb (support threshold), γ pedig a felhasználó által adott úgynevezett bizonyossági küszöb (confidence threshold).

Egy kosárhalmazban $\mathbf{s} \rightarrow \mathbf{u}$ -t $\mathbf{supp}(\mathbf{s} \rightarrow \mathbf{u})$ támogatottságú, $\mathbf{conf}(\mathbf{s} \rightarrow \mathbf{u})$ bizonyosságú asszociációs szabálynak nevezzük, ha:

$$\mathbf{supp}(\mathbf{s} \rightarrow \mathbf{u}) = \frac{|\lambda(\mathbf{s} \rightarrow \mathbf{u})|}{|\mathbf{D}|}$$

és

$$\mathbf{conf}(\mathbf{s} \rightarrow \mathbf{u}) = \frac{|\lambda(\mathbf{s} \rightarrow \mathbf{u})|}{|\lambda(\mathbf{s})|}$$

A szabályban **s**-t előzménynek vagy feltételnek, **u**-t pedig következménynek nevezzük. Egy asszociációs szabály $\sigma\gamma$-érvényes ($\sigma\gamma$-valid), ha az alábbi két feltétel egyszerre teljesül:

- $\mathbf{supp}(\mathbf{s} \rightarrow \mathbf{u}) \geq \sigma$, vagyis **s**-nek és **u**-nak elég a tranzakciók σ%-ában egyszerre szerepelni.
- $\mathbf{conf}(\mathbf{s} \rightarrow \mathbf{u}) \geq \gamma$, amennyiben $\gamma \neq 1$ az azt jelenti, hogy megengedett a 100%-tól való eltérés, tehát az asszociációs szabály csak „körülbelül" igaz.

Érvényes asszociációs szabály

Egy $\mathbf{s} \rightarrow \mathbf{u}$ szabály érvényes asszociációs szabály, amennyiben $\sigma\gamma$-érvényes. $\gamma = 1$ esetén egzakt asszociációs szabályról beszélünk. Az érvényes asszociációs szabály keresése két lépésből áll:

1. Gyakori termékhalmazok keresése: az $\mathbf{A}$ termékhalmazból kikeressük azon $\mathbf{y} \subseteq \mathbf{A}$ termékhalmazokat, melyekre igaz, hogy $\mathbf{supp}(\mathbf{v}) \geq \sigma$

2. Érvényes asszociációs szabályok kiválasztása a gyakori termékhalmazból. Ez az eljárás a következő: kiválasztom azon $\mathbf{s} \rightarrow \mathbf{u}$ szabályokat, melyekre $\mathbf{u} = \mathbf{v} \backslash \mathbf{s}$ és $\mathbf{u} \neq \oslash$ és $\mathbf{conf}(\mathbf{s} \rightarrow \mathbf{u}) \geq \gamma$

Amennyiben a fenti feltételek teljesülnek, $\mathbf{s} \rightarrow \mathbf{u}$ érvényes asszociációs szabály.
Például:

Legyen mindkét példánál: $\sigma = 50\%$ és $\gamma = 100\%$

1. Példa: legyen $\mathbf{s} = \{b\}$ és $\mathbf{u} = \{c\}$!

$$\lambda(\mathbf{s} \rightarrow \mathbf{u}) = \lambda(\mathbf{s} \cup \mathbf{u}) = \lambda(\{b\} \cup \{c\}) = \lambda(\{b, c\})$$

$$\mathbf{supp}(\mathbf{s} \rightarrow \mathbf{u}) = \frac{|\lambda(\{b\} \cup \{c\})|}{|\mathbf{D}|} = \frac{|\{3, 4, 6\}|}{6} = 50\%$$

$$\mathbf{conf}(\mathbf{s} \rightarrow \mathbf{u}) = \frac{|\lambda(\{b\} \cup \{c\})|}{\lambda(\{b\})} = \frac{|\{3, 4, 6\}|}{|\{3, 4, 6\}|} = 100\%$$

Tehát $\mathbf{s} \rightarrow \mathbf{u}$ érvényes asszociációs szabály, mivel $\sigma\gamma$-érvényes.

2. Példa: legyen $\mathbf{s} = \{c\}$ és $\mathbf{u} = \{b\}$

$$\mathbf{supp}(\mathbf{s} \rightarrow \mathbf{u}) = \frac{|\lambda(\{c\} \cup \{b\})|}{|\mathbf{D}|} = \frac{|\{3, 4, 6\}|}{6} = 50\%$$

$$\mathbf{conf}(\mathbf{s} \rightarrow \mathbf{u}) = \frac{|\lambda(\{c\} \cup \{b\})|}{|\{\lambda(\{c\})\}|} = \frac{|\{3, 4, 6\}|}{|\{2, 3, 4, 5, 6\}|} = 60\%$$

Mivel ennél a példánál $\mathbf{conf}(\mathbf{s} \rightarrow \mathbf{u}) \geq \gamma$ nem igaz, ezért a $\mathbf{c} \rightarrow \mathbf{b}$ szabály nem érvényes. Vagyis aki **c** terméket megvásárolja, az nem vesz feltétlenül **b** terméket is.

Tétel: Legyenek **s** és **u** gyakori termékhalmazok. Ha $\mathbf{s} \subseteq \mathbf{u}$ és **u** σ-gyakoriságú (σ-frequent), akkor **s** is az.
Bizonyítás:
Ha **u** σ-gyakoriságú, akkor $\mathbf{sup}(\mathbf{u}) = \frac{|\lambda(\mathbf{u})}{|\mathbf{D}|} \geq \sigma$. Az előzőekből következően $\mathbf{supp}(\mathbf{s}) \geq \mathbf{supp}(\mathbf{u})$, tehát ha **u** σ-gyakoriságú, akkor az összes részhalmaza is az.
Következmény:
Ha **s** nem σ-gyakoriságú, akkor az összes olyan halmaz, melynek **s** részhalmaza sem lehet az.

Az adatbányászat témaköre nagyon jelentős szerepet játszik például a villamosenergia-termelés során. Ebben az iparágban rendkívül fontos az áramfogyasztás megbecslése, milliárdokat lehet havonta spórolni egy pontos becslés által. Az ilyen problémákat megoldó szoftverek ára igen magas, viszont gyakran 1 hónap alatt már megtérül.

5.2.8. Az adatbányászatban alkalmazott technikák

Az adatbányászat alapfeladata:
Adott egy adathalmaz (adatáruház), amelyből számítógépes tanulás útján új információt, illetve új ismeretet, tudást kell előállítani, azaz olyan mintákat, kapcsolatokat, amelyekről előzően nem tudták, hogy

léteznek, vagy nem voltak láthatóak. (Információ például minta vagy döntési tábla; ismeret, tudás például döntési vagy osztályozó szabály.)

Az adathalmaz nagy méretű és jellemzően elosztott (általában több, egymástól független adatbázisra). A gépi tanulás célja az, hogy keressük azt a hipotézist, amely „legjobban illeszkedik" a kiindulásul vett adatokra - azzal az elvárással, hogy ez a hipotézis alkalmazható legyen előre nem látott adatok esetére is. Egy adatbázis felett megfogalmazott SQL lekérdezés során a feltett konkrét kérdésekre konkrét válaszokat kapunk (mégpedig az adott kérdésben megfogalmazott feltételeknek eleget tevő adathalmazt). Az adatbányászatban jellemzően nem megfogalmazható kérdésekre keressük a választ. Itt a munka két részből áll: az első rész a hipotézisek felállítása és tesztelése, a második a hipotézis alapján történő tényleges adatfeltárás.

1. Az első részben a szakember a kiinduló adatbázis előzetes elemzése révén („ránézésre") feltevéseket, hipotéziseket állít fel, és megfelelő adatbányászati eszközökkel megpróbálja azokat igazolni. A hipotéziskeresés és annak igazolása matematikai statisztikai módszerekkel és (az adatbányászati eszközök által erősen támogatott) különböző vizualizációs módszerek alkalmazásával történik. A felállított előzetes hipotézisek segítségével kezelhetővé válik az adatbányászati probléma. Ezen hipotézisek bizonyos összefüggések kézzel, a szakember által, való feltárását jelenti, melyek a későbbi, automatizált részben felhasználásra kerülnek.

2. A második részben a szakember az igazolt hipotézis alapján feltárást végez, melynek során a rendelkezésére álló különböző algoritmusok felhasználásával az (első részben előkészített) adatbázisból jellemző mintákat, összefüggéseket emel ki. A feltárás jellemzően a gépi tanulás módszereinek alkalmazásával történik a vizualizációs lehetőségek kihasználásával; az eredmény egy leíró és (táblázatokkal, grafikonokkal stb.) vizuálisan megjeleníthető modell.

Ez a két rész gyakran összefonódik: az adatfeltárás eredményeit elemezve esetleg újabb hipotéziseket készít a szakember, majd azok sikeres igazolása után újabb feltárást indít. Természetesen, az alkalmazott adatbányászati technikáktól, eszközöktől függően a kiindulási adathalmazt előfeldolgozásnak kell alávetni; ez az adatok megszűrését, tisztítását, kiválogatását, esetleg transzformációját jelenti.

A korszerű adatbányászati eszközökben felhasznált technikák:

1. Magasszintű statisztikai és egyéb elemző módszerek: például statisztikai próbák, klaszter-, faktor-, diszkriminancia analízis, többdimenziós skálázás, lineáris és nemlineáris modellek, kontingenciatáblák, conjoint elemzés, preferencia térképek, továbbá idősorok elemzése, lineáris és nemlineáris regresszió-analízis, lineáris és nemlineáris programozás, ökonometriai modellek, szimulációs és egyéb speciális eljárások

2. Vizualizációs technikák: két- és háromdimenziós grafikonok, tudományos és üzleti diagramok, speciális ábrázolások, térképek, térinformatikai eszközök, valamint multimédia

3. Következtető technikák: döntési fát vagy szabályhalmazt generáló induktív technikák, fuzzy technikák, genetikus algoritmusok, neuronhálózatok

A két utóbbi csoport az adatbányászat második részében alkalmazott „ismeretfeltáró" mesterséges intelligencia-technikákat fogja össze. A „feltáró" jellegű adatbányászati technikák tipikusan induktív jellegűek: egyedi esetekből jutnak általános érvényű következtetésre, szemben a logikai rendszerekben alkalmazott deduktív következtetéssel, amely az általánosból indul ki. (A relációs adatbázis-kezelő rendszerek már rendelkeznek bizonyos deduktív képességekkel, de induktív kiterjesztésük az adott technológián belül nem lehetséges.) Az induktív technikák az adatokból a gépi tanulás egyes módszereit alkalmazva különböző modelleket építenek, melyeket az adatbányászati eszközök a megértést segítendő, vizuálisan is megjelenítenek. Az adatbányászatban alkalmazott gépi tanulási módszerek végső célja az, hogy a megtanult minta vagy összefüggés alkalmazható legyen előre nem látott esetekre is. A gépi tanulásnak felügyelt és nem-felügyelt változatait szokás megkülönböztetni:

- A felügyelt tanulást (supervised learning) prediktív vagy következtető, jósló elemzésnek is nevezik. Ez az adatbányász szakember által manuálisan irányított tanulási folyamat, melynek során a szakember ismert (a tanulótól is elvárt helyes válasz ismert) adatokat prezentál a tanuló algoritmus számára, mely ezekből a tanító példákból állít össze olyan modellt, amelynek segítségével képes lesz ismeretlen példákon is helyesen működni.

- A nem-felügyelt tanulást (unsupervised learning) deszkriptív vagy leíró elemzésnek is nevezik. Ez egy olyan automatikus folyamat, melynek célja az adatbázisból (új) minták, összefüggések felfedezése.

A felügyelt tanulás

A felügyelt tanulás a meglévő adatok között összefüggést keresve egy új, „megjósolt" adatmezőt ad eredményként (melyet az angol terminológia label-nek nevez). A tanulás során a felügyelt tanulást végző algoritmusnak a szakember olyan példákat prezentál, melyeknek a label mezője helyesen ki van töltve. A létrejött modellt teszt példák felhasználásával tesztelik, azaz olyan példákkal, melyeknek a label mezője „letakarásra" kerül.

Formálisan a felügyelt tanulás algoritmusai egy függvényt határoznak meg, illetve közelítik azt: legyenek adottak $(\underline{x_j}, y_j)$ változó párok, ahol $j = 1, \ldots, m$. Előállítandó az az $y = f(x)$ függvény, amely minden változópárra teljesíti, hogy $y_j = f(\underline{x_j})$. Általában minden $\underline{x_j}$ érték valamely objektum vagy esemény leírása (pl. a beteg tünetei). Az y_j értékek az $\underline{x_j}$ értékekből történő következtetéseket reprezentálják (pl. a beteg diagnózisa).

Feltételezzük, hogy a tanulás során az y_j értékeket a „felügyeletet ellátó tanár" határozza meg. A bemenő adatok, azaz $\underline{x_j}$ értékei lehetnek számszerűek (pl. életkor, fizetett összeg), kategorizáltak (pl. nem, lakhely), de lehetnek valamely előfeldolgozás eredményeként kapott értékek is (pl. átlag, maximum, minimum). Ha az y változónak csak két lehetséges értéke van (pl. „influenzás"/„nem-influenzás"), akkor fogalom

tanulásról (concept learning) beszélünk. Ebben az esetben a tanító példákat két diszjunkt részhalmazra lehet bontani: a pozitív és a negatív példák halmazára. A felügyelt tanulás során két feltételezéssel élünk:

1. Az f függvény egyedi példákból általánosítással történő előállítására egyszerű számítógépes modellt tudunk adni - bizonyos pontosság mellett.

2. A tanító példahalmaz eléggé informatív az előbbi általánosítás elvégzéséhez.

A modell ezeknél az algoritmusoknál jellemzően nem teljesen pontos (csak hipotézist kapunk), ezért fontos kérdés a pontosság mértékének megállapítása. E módszerek között aszerint, hogy az új mező (label) diszkrét vagy folytonos értékkel rendelkezik megkülönböztetünk osztályozást, illetve regressziót.

- Az osztályozás vagy klasszifikáció tanító példák alapján tanulja meg az osztályozás szabályait, melyeket szabályok vagy döntési fa formájában fogalmaz meg. (A tanító példák rendre tartalmazzák a megfelelő osztályértéket). A kapott eredményt tesztpéldákkal szokás ellenőrizni. Az eredmény (vagyis az osztályozás) pontossága növelhető a betanítás során alkalmazott tanítópélda-halmaz méretének növelésével. Egyes algoritmusok kezelni tudják a hiányosan, illetve a zajjal terhelt példákat is. Jellemző osztályozó módszerek: döntési fákat generáló induktív módszerek (ID3 algoritmus és bővített változata a C4.5 rendszer), szabályhalmazok tanulása, tanulás neuronhálózatok (pl. Hopfield típusú hálózatok) felhasználásával. Az osztályozó eljárások alkalmazása főképpen az üzleti jellegű problémáknál történik. Tipikus eset az, amikor egy címjegyzék elemeiről kell eldöntenünk, hogy érdemes-e nekik termékmintát és/vagy prospektust küldeni, azaz fognak-e vásárolni az adott termékből (fogalom tanulás). Nagy haszna van a banki hitelkérelmet elbíráló rendszereknél történő „előrejelző" alkalmazásoknak: a korábbi ügyféladatok (a kérelemben szereplő fontosabb adatok, a visszafizetés pontossága, késedelmek stb.), mint tanító példák alapján kidolgozott osztályozó alkalmazása nagyban megkönnyíti - és biztonságosabbá teszi - a bejövő új kérelmek elbírálását. Ehhez hasonló problémakör például a hitelkártyákkal elkövetett csalások gyakoriságának vizsgálata.

- A regresszió esetében hasonló a helyzet, mint az osztályozásnál, azonban az új mező értéke nem diszkrét, hanem folytonos. (Mivel ez nem közvetlenül a mesterséges intelligencia területe, ezért ennek részletezésével itt nem foglalkozunk.) A regresszióval történő függvényrekonstrukció leggyakoribb felhasználása a gazdasági matematikában található, ahol idősorok (egy része) alapján kell előre jelezni az adott idősorok jövőbeni állapotát. Szintén alkalmazzák még termékek üzleti sikerességének előrejelzésére is. A hagyományos (köz)gazdaságtani matematikai módszerek egy úgynevezett sikerfüggvény alkalmazásával dolgoznak, amelynek argumentumai között szerepelhet például a termék reklámkampányára költött összeg, a megcélzott vásárlói réteg vagy a termék ára, míg kimenete a várható haszon.

A nem-felügyelt tanulás

Jóllehet a felügyelt tanulást alkalmazó adatbányászati szoftvermegoldások nagy hatékonyságra képesek, alkalmazásuk eléggé költséges és időigényes (informatív tanító- és tesztpéldák előállítsa, módosítása). Bi-

zonyos feladatokra egyszerűbb módszerek is elegendőek. A gépi tanulás ilyen egyszerűbb, automatikusan alkalmazható módszerei a nem-felügyelt tanuló technológiák. Itt a cél az, hogy az adatbázisban korábban nem ismert mintákat, összefüggéseket találjanak. Formálisan: az algoritmusok csak az $\underline{x_j}$ értékeket kapják meg ($i = 1, \ldots, m$), és azok elemzése révén szabályosságokat „fedeznek fel" e minta elemei között. A nem-felügyelt tanulásnak két jellemző módszere a csoportosítás és az asszociáció.

- A csoportosítás vagy klaszterezés (clustering) feladata az, hogy az elemzendő adathalmazokat homogén diszjunkt részhalmazokra bontsuk. A homogenitást ebben az esetben a csoport elemeinek hasonlóságára alapozzuk. Szemléltetésként képzeljünk el egy kupac labdát, pirosakat és fehéreket. Nyilvánvaló ekkor, hogy a labdák két homogén csoportra oszthatók, mégpedig a színük szerint. Amennyiben a labdák méretükben is különböznek (kicsik és nagyok), úgy már négy homogén csoportot nyerünk: piros kicsik, piros nagyok, fehér kicsik valamint fehér nagyok. Általánosítva a feladat nem egy függvény, hanem valamely homogenitási reláció definiálása. Ez a legegyszerűbben egy távolságmetrika segítségével oldható meg, vagyis egy csoportot homogénnek veszünk, ha bármely két eleme közt a távolság nem ér el egy küszöbszámot (ez a küszöbszám a rendelkezésre álló adatok alapján automatikusan is számolható). A probléma ezzel a megközelítéssel, hogy a távolságmetrikával definiált hasonlósági reláció nem kötelezően ekvivalenciareláció (abból hogy A hasonlít B-re, és hogy B hasonlít C-re nem feltétlen következik, hogy A is hasonlít C-re, azaz nem feltétlenül tranzitív a relációnk), ezért nem alkalmas a homogén ekvivalenciaosztályok meghatározására. Különböző módszerek léteznek a hasonlósági reláció megfelelő átalakítására, hogy ekvivalenciarelációt kapjunk. Osztályozási szabály kidolgozásával optimális partícionálási feladatot tudunk megoldani. Itt a hasonlóság mértékének meghatározása halmazfüggvények alkalmazásával történik: legyen a feladat egy A halmaz megfelelő diszjunkt A_j részhalmazokra bontása. Definiáljuk az A_j részhalmazok súlyát valamely $F(A_j)$ halmazfüggvény segítségével, az A halmaz súlyát pedig ezek összegével vagyis: $F(A) = \sum_j (F(A_j))$. Ekkor a partícionálási feladat a következőképpen határozható meg: vegyük azt a részhalmaz-rendszert, amely mellett $F(A)$ optimális (pl.: minimális). A csoportosításnak sokféle módszere van; ezek alkalmazása különösen akkor ajánlható, ha a vizsgált adathalmaz nagy és/vagy áttekinthetetlenül bonyolult. A módszer az adatok közötti rejtett összefüggések feltárásával hasonló karakterisztikájú csoportokba tudja rendezni az adatokat. (Egy-egy csoporton belül esetleg már rá lehet érezni valamilyen heurisztikára - így alkalmazható lehet ott a felügyelt tanulás valamely módszere.)

- Az asszociáció (egyesítés vagy összekapcsoló elemzés) a nem-felügyelt tanulás fontos módszere. Ennek célja általában az, hogy az adatbázisból kikeressék az összes olyan tranzakciót, amelyek nagy valószínűséggel ismétlődnek. Igen sok kiskereskedelmi alkalmazása van e módszernek (pl.: a tejet vásárlók 55 %-a egyéb tejterméket, 42 %-a pedig kenyeret is vásárol). Az asszociatív algoritmusok bővítik az SQL lehetőségeit változó adatállomány és változó „viselkedés" figyelembevételével. Sokféle asszociációs technika áll rendelkezésre az egyes adatbányászati eszközökben; ezeket mind az ipar, mind az üzleti folyamatok területén kiterjedten használják. Az asszociatív algoritmusokhoz hason-

lóak a sorrendi minták, amelyek meghatározott időtartam szerint kigyűjtött adatokkal dolgoznak, és e történeti adatok elemzésének eredményeként megadják a legnagyobb valószínűséggel ismétlődő mintákat (mintaelemzés). A sorrendi minták mellett a hasonló sorozatok technikáját is szokták alkalmazni az üzleti életben. Itt nem az események időbeli lefolyását elemzik (mint a sorrendi mintáknál), hanem az időbeli események sorrendjét. Például hasonló árképzéssel dolgozó kiskereskedelmi üzlet keresése vagy hasonló ármozgással dolgozó raktárak keresése.

5.2.9. A kereső eljárások

Az adatbányászati módszerek típusai után vizsgáljuk meg a kereső algoritmusokat. Hiába végzünk el egy részletes keresést, az adatbázis méreteinek köszönhetően ezt mégsem mindig célszerű megtenni a feladat komplexitása miatt. Hatékony kereső algoritmusokra van szükség. A legtöbb adatbányászati megoldás klasszikus logikai lekérdezést hajt végre, s aztán a lekérdezéseket módosítja iteratív módon, az üzleti célnak megfelelő határérték eléréséig. Ezekben a megközelítésekben számos különböző típusú stratégia létezik. Nincs legjobb megközelítési mód, a feladattól függ, mikor melyiket célszerű alkalmazni. A keresés kezdeti megközelítésének függvényében beszélhetünk felfelé vagy lefelé irányuló folyamatról. A felfelé irányuló megközelítést adatvezérelt megközelítésnek nevezzük. Mint a szakértői rendszereknél, ez egy adatbázissal és egy kezdeti szabállyal indul, majd finomító eljárást hajt végre, mindaddig míg a határértéknek megfelelően nem tartalmaz olyan adatot, amely megszegi a szabályt. Egyszerűbben kifejezve, megvizsgálja az adatot, megkeresi benne az összes egyedi elemet, amely megfelel a szabályoknak és aztán ezekből az elemekből meghatározza az eredményt. A lefelé irányuló megközelítés szabály-vezérelt keresés és néhány leíró művelet alkalmazásával kezdődik, ami elindít egy válogatást. Kezdetben, az összes adatból indulunk ki, a finomítási eljárások folyamán végül azokat az elemeket kapjuk, amik nem tesznek eleget a szabályoknak. Az eljárás addig folytatódik, amíg olyan adatmennyiséget kapunk, ami megfelel a kívánt értékhatároknak. A felfelé irányuló megközelítés csak a kriteriumoknak megfelelőket válogatja ki, hogy végül megtalálja az események mintáját és sorrendjét. A lefelé irányuló megközelítés pedig hatalmas készlettel indul és csak a kriteriumoknak nem megfelelőket válogatja ki. Mindkét megközelítésnél finomítási és műveleti eljárásokat hajtunk végre, amely új adatbázist, vagy adatbázis leírást eredményez. Aztán újrakezdődhet a folyamat egy másik adatbázisra ugyanazokat az eljárásokat alkalmazva. Ez az eljárás az új szabályok megtalálása mellett az adatbázist szem előtt tartva a keresés grafikonját is felépíti segítve ezzel az előírt határértékek figyelembe vételét. A folyamatot könnyebb megérteni egy valós példán. A szabályalkotás és finomítás bemutatásához próbáljuk meg elképzelni a következő műveleteket:

1. Vizsgáljunk meg minden vásárlást, ami egy élelmiszerüzletben történt és válasszuk ki bármelyik két elemet ebből az egyéni vásárlásból. 2. Vizsgáljuk meg az összes többi vásárlást. Nézzük meg, hogy a két kiválasztott kezdeti elemet ugyanakkor vásárolták-e. Ha ezen vásárlások száma az előírt határértéket eléri, elkezdhetünk egy eredménykészletet építeni. 3. Vegyük az egyik első elemet és párosítsuk össze egy másikkal az első vásárlásból. Ezután indítsuk újra az eljárást.

A kereső eszközök a mesterséges intelligencia kutatásának középpontjában állnak. Különböző megkö-

zelítéseket és stratégiákat alkalmaznak, hogy megelőzzék a részletező keresést és gyorsan egy konklúzióhoz érkezzenek.

5.2.10. Adatbányászati módszerek

Az előzőekben láttuk, hogy milyen színes az a technikai paletta, amelyből az adatbányászat jelenlegi eszközei építkeznek. Ezen technikák és a kereskedelemben kapható adatbányász eszközök ismerete azonban nem elég a sikeres ismeret-feltárási munkához. Tudni kell azt is, hogy hogyan formalizáljuk a megoldandó problémát, és hogy milyen lépéseket, lépés-sorozatot célszerű tenni a probléma megoldása érdekében. Az egyes adatbányász eszközök kínálnak ilyen fejlesztői módszertant az adatbányász szakember számára. Egy adatbányászati alkalmazás fejlesztésének módszertana a következő fázisokat írja elő:

1. Az üzleti probléma megismerése és fogalmazása.
2. A rendelkezésre álló adatok megértése.
3. Elemzés. Ez egy ciklus, melynek lépéseit többször hajthatjuk végre:
   a.) Adatelőkészítés a probléma és az alkalmazandó eszköz specifikumaira figyelve.
   b.) Modellezés.
   c.) Tesztelés.
   d.) Kiértékelés.
4. Alkalmazás.
5. Karbantartás (amennyiben szükséges).

E fázisokat nem taglaljuk részletesen, csupán néhány jellegzetes fejlesztői fogást vázolunk a következőkben. (Az IBM már 1996-ban kiadta saját adatbányászati módszertanát ismertető „fehér könyvét”.)

Adattisztítás

A hiányzó, vagy rossz adat kezelésének módszere. Nagy adatbázisokban gyakran előfordul, hogy egyes attribútumok rosszul kerülnek be az adatbázisba, vagy be sem kerülnek oda. Ennek több oka lehet: hiba vagy hanyagság történt az adatfelvitelben, vagy egyszerűen az adott attribútumról nem volt információ (nem adták meg, titkos, magánjellegű stb.) Ugyanakkor szeretnénk konzisztens, helyes ismereteket kinyerni az adatbázisból a hibák, hiányok ellenére is. A hibás adatok kezelésére kínálkozó módszerek például:

- Nem vesszük figyelembe a hibás adatot az adatbányászat során.
- Figyelembe véve az adott attribútumhoz tartozó többi adatot, az adatbázisban statisztikai eljárásokkal javítjuk a valószínűleg hibás adatot.
- Olyan tanulóalgoritmust használunk majd az adatbányászat során, amely kevésbé érzékeny a hibákra.

Adatintegrálás

Az adatintegritás több forrásból származó adatok integrálását jelenti. Elképzelhető olyan eset, amikor az adatbányászathoz szükséges adatok több különböző adatbázisban vannak. Lehetséges, hogy az egyes adatbázisokat különböző adatbázisszerverek tárolják, különböző formában. Ekkor az adatokat össze kell gyűjteni az egyes adatbázisokból, és egységes formátumra kell hozni. Erre a problémára nyújtanak megoldást az úgynevezett adattárházak (data warehouses), melyek egyszerű konzisztens hozzáférést nyújtanak szervezetek és intézmények adatbázisaihoz, egységesítve az egyes osztályok adatbázisait. Ezekben a tárházakban általában a cégek régebbi adataikat publikálják (tehát amennyiben aktuális adatokkal akarunk dolgozni, akkor az adatokat a megfelelő adatbázisokból kell kigyűjteni).

A tanulóalgoritmus megválasztása

Az adott feladatnál a tanulóalgoritmus megválasztásakor figyelembe kell vennünk, hogy a megszerzett ismeretet milyen formában akarjuk reprezentálni. Például egy osztályozó problémát viszonylag jól tud megoldani egy neuronháló alapú alkalmazás, ugyanakkor rejtve marad a döntési mechanizmus, hogy egy adott példányt mi alapján sorolt be az osztályba, illetve vetett el. Lehetséges, hogy ugyanerre a feladatra egy szabályalapú rendszer kevésbé jó megoldást ad, de a felhasználó sokkal több információt („magyarázatot") tud adni a felhasználónak. Az adatbányászat körében előnyösen alkalmazhatók a többféle tanuló (és további más) technikákat együttesen tartalmazó, úgynevezett hibrid eszközök.

Adatválasztás

Az adatválasztás arra a kérdésre keresi a választ, hogy hogyan kell a megfelelő tanítóhalmazt megválasztani. Az adatbányászat során nem használjuk az adatbázisban tárolt egész adatmennyiséget főként annak méretei miatt, csak annak egy jóval kisebb részét. Ez alapján fogjuk a mintákat, összefüggéseket megkonstruálni és elvárásaink szerint ezek az összefüggések az egész adatbázisra vonatkozni fognak.

Adattranszformáció

Az adattranszformáció során arra a kérdésre keressük a választ, hogy hogyan hozzuk az adatot olyan formára, hogy a tanulóalgoritmus használni tudja azt? Sok esetben az adatbányászatban használt tanulóalgoritmus adott formában várja a bemenő adatokat (pl. ha döntési fát kívánunk generálni, szimbolikus adatokból dolgozunk). A bemenet típusa lehet numerikus is ha neuronhálót, Support Vector Machinet vagy más függvényközelítő módszereket használunk. Ekkor az adatokat megfelelő formára kell hozni. Az utóbbi esetben ez azt jelenti, hogy kell találnunk egy olyan függvényt, amely az adat tulajdonságait egy olyan számra képezi le, amely megfelelően reprezentálja az adatot. A függvény megkonstruálása lehet könnyű, de igen bonyolult is; ez mindig a konkrét problémától függ. Általános recept nincs, ugyanakkor az információelmélet eszközeit sokszor sikeresen lehet alkalmazni (entrópia, kölcsönös információ stb.).

5.2.11. Az adatbányászat reprezentációs technikái

Lényeges kérdés a felderített eredmények megjelenítésének formája. Bármilyen értékes új felfedezésre jutottunk egy eszköz segítségével, ha azt nem lehet érthetően és látványosan megjeleníteni, haszna kevés

lesz. A kinyert eredmény reprezentálásának főbb fajtái: döntési táblák, döntési fák, osztályozó szabályok, társító szabályok, kivételek, példány-alapú reprezentáció, csoportok. Az alábbiakban ezek rövid informális bemutatását adjuk.

1. Döntési táblák (decision tables)
   Ez a legegyszerűbb, legalapvetőbb, közismert reprezentációs forma. A probléma itt az, hogy nem tudjuk a táblázat alapján „ránézésre" eldönteni, hogy mely attribútumok hagyhatók el anélkül, hogy a döntés eredménye ne változzon.

2. Döntési fák (decision trees)
   Ez a reprezentáció az oszd meg és uralkodj (divide-and-conquer) megközelítéssel konzisztens reprezentációt ad a megszerzett ismeretekről. A gyökértől a levélig vezető utakról leolvashatjuk az egyes döntési szabályokat. Ugyanakkor már nem túl nagy vizsgált attribútum halmazra is nehezen értelmezhető, hatalmas méretű és két dimenzióban nehezen ábrázolható fához juthatunk a különböző attribútum értékek vizsgálatakor.

3. Osztályozó szabályok (classification rules)
   Egy osztályozó szabály olyan tesztek sorozata, amely (mint a döntési fa egy ága) egy osztályt (vagy osztályokat) határoz meg. A szabály feltételrésze tartalmazza a teszteket, következményrésze pedig az osztályt/osztályokat. Előfordulhat azonban, hogy ezek a szabályok nem konzisztensek, ellentmondásra vezetnek, különösen, ha nem egy döntési fa alapján generálják őket. A problémára megoldást nyújthat, ha megszámoljuk, hogy egy adott példányt az algoritmus hányszor sorol az egyik, illetve a másik osztályba, és a nagyobb számosságút vesszük figyelembe. Másik megoldás, hogy az adott példányt nem soroljuk be egyik osztályba se. Előnye ennek a reprezentációnak, hogy egy új szabály az előzőek módosítsa nélkül hozzáadható a szabályhalmazhoz (a döntési fánál az egész fát át kell alakítani). Hátrány az, hogy a szabályhalmaz nem tartalmazhat egymásnak ellentmondó következményrészű szabályokat.

4. Társító szabályok (association rules)
   Ezek olyan osztályozó szabályok, melyeknek következményrésze attribútumokra vonatkozó értékadást is tartalmaz. A társító szabályok különböző törvényszerűségeket fejeznek ki, és sorrendjük kötött.

5. További, speciális szabályformák
   A szabályokban megfogalmazhatunk kivételeket (exceptions) is a speciális, kirívó esetek kezelésére. A kivételeket megfogalmazó szabályok egymásba is ágyazhatók. Bizonyossági tényezőt (certainty factor) is kapcsolhatunk a szabályokhoz, amely megmutatja, hogy az adott szabály mekkora bizonyossággal áll fenn.

6. Példány-alapú reprezentáció (instance-based representation)
   A tanulás legegyszerűbb formája az egyszerű memorizálás, melynek során jellemzően a legjobban hasonló példányokat vesszük segítségül. A „legjobban hasonló" definiálása sokféle módon történhet.

Lehet akár például osztályozó szabályok segítségével is. Ismertek az euklideszi távolságmértékkel operáló legközelebbi szomszéd, vagy k legközelebbi szomszéd eljárások. Ezek esetén az attribútumértékeket normalizálni szokás. Bizonyos esetekben a távolságmérték számításához csak az „érdekes" attribútumokat veszik figyelembe. Nem-numerikus attribútumok esetén az egybeesés, illetve a hasonlóság mértékének reprezentációját meg kell oldani. A példány-alapú reprezentációnál az egyes példányokat pontokként jelenítjük meg a képernyőn, ahol például az egy osztályba tartozó (ilyen értelemben hasonló) példányok egy alakzatba kerülnek. Más módszerek lehetőséget nyújtnak bonyolultabb döntési felület megjelenítésére.

7. Csoportok (clusters)
   Sokszor adott tulajdonság(ok) alapján csoportokba szeretnénk osztani a példányokat. Az ábrázolás módja igen sokféle lehet: az n dimenziós síkok felosztása $n-1$ dimenziós terekkel; halmazokkal történő ábrázolás (Venn-diagram); táblázatos ábrázolás (valószínűségekkel); fastruktúra-ábrázolások (dendogram), csoport-alcsoport ábrázolás (egy elem több csoportba is beletartozhat, ld. Venn-diagram) stb. A fastruktúra-ábrázolásokból könnyű osztályozó szabályokat kiolvasni. Itt minden levélhez egy szabály tartozik: a következményrész a levélhez kapcsolt csoport, míg a feltételrész a csomóponti feltételeknek felel meg.

5.2.12. Konkrét adatbányászati alkalmazások

Az első „intelligens" adatbányászati konferenciát 1995-ben tartották (KDD-95: Knowledge Discovery and Data Mining Conference). A következőkben kiemelünk a konferencián megismert, az adatbányászati eszközpiac, illetve az adatbányászati technikákat alkalmazó, sikeres alkalmazások közül néhány tanulságosat, megjelölve a fejlesztés során felhasznált adatbányászati eszközöket (a fejlesztés intézményét); az eszközbe integrált vagy más eszközből vett mesterséges intelligencia technikákat.

1. **Gazdasági alkalmazások**

   a.) Gazdasági adatok tisztítása (Lockheed)

   Eszköz: Recon (Lockheed); deduktív adatbázis, induktív szabálygenerálás. Az alkalmazás célja egy pénzügyi történeti adatállomány megtisztítása volt; az adatbázis több mint 2200 mexikói, brit és Euro kötvényről tartalmazott adatot. 10 tábla és 150 mező felhasználásával írta le az egyes kötvények feltételeit és további háttér-információkat. Ugyanakkor az adatbázis nem tartalmazott szigorú adatellenőrzési mechanizmust, így az adatok integritása kétséges volt. Ezért volt szükség a Recon eszköz alkalmazására. Elemzők különböző adatbázisokból a Recon segítségével információkat (különböző korlátozásokat) nyertek ki az egyes kötvényekkel kapcsolatban, amelyeket beleépítve a Recon-ba, futatták azt a kötvényeket tartalmazó adatbázisra. A kirívó esetek megtalálására természetesen felhasználták a Recon vizuális eszköztárát is. Ezt az eljárást alkalmazva sikeresen növelték az adatbázis konzisztenciáját, kiszűrve a hibás bejegyzéseket és a lejárt kötvényeket.

b.) Tőzsdei árfolyamok ellenőrzése (Reuters)

Eszköz: Clementine (Integral Solution Ltd.); neuronhálózatok, induktív fa- és szabálygenerálás. Ez egy olyan, külföldi tőzsdeadatokban hibát kereső alkalmazás, amely az aktuális árfolyam alakulás alapján durva előrejelzést is szolgáltat. Nagy előnye, hogy a rendelkezésére bocsátott adatokból automatikusan, (emberi) szakértő alkalmazása nélkül képes dolgozni változó árfolyam-alakulás feltételei mellett.

c.) Hátralék problémák előrejelzése (The Leeds)

Eszköz: XpertRule Analyzer (Attar Software); induktív szabálygenerálás. A Leeds célja az volt, hogy feltárja, milyen hátralék problémák léphetnek fel az általa kezelt 500000 szervezet jelzálog számláján. Kifejlesztettek több adatprofil-meghatározó szabályt: egészséges számla, hátralékban lévő számla, teljesíthetetlen kintlévőség, regionális hátralékok, és a hátralékokban lévő számlák egyéb karakterisztikájának a kiszűrésére. A Leeds ugyancsak adatbányászó algoritmust alkalmazott a hitelképesség kiértékelésére, illetve célorientált marketing-terv elkészítésére.

2. **Egészségügyi alkalmazások**

a.) Med-AI: Betegségek modellezése, súlyossági esetek kiszűrése, adattisztítás (Med-AI Inc.)

Eszköz: a Med-AI egyben eszköz is; neuronhálózatok, indukciós technikák. A Med-AI célja az volt, hogy nagy kórházi adatbázisokból olyan ismereteket nyerjen ki, amelyekkel javítani tudnák az ellátás hatékonyságát. Először különböző adatbázisokból összegyűjtötték két év kórházi bejegyzéseit. Ezek közül a hibás adatokat - neuronhálózatos és induktív technikák alkalmazásával - kiszűrték. Ennek során sikerült feltárni nem csak a hibás számlákat, hanem helytelen, illetve gazdaságtalan klinikai eljárásokat is. Fény derült többek között arra is, hogy a klinikán miért volt olyan sok operáció után fellépő megbetegedés. Ezeket a fertőzéseket emelt szintű antibiotikum adagolással kezelték, ami jelentősen megnövelte a gyógyítás költségeit. Korábban nem tudták kideríteni e rejtély okát. A Med-AI kimutatatta, hogy az érintettek zöme olyan helyen lakott, ahol csatorna vagy más mérgező anyagokat tároló hely található. Így az orvosok már specifikusabb gyógyszerekkel sikeresen vehették fel a harcot az operációk után fellépő fertőzésekkel szemben. Számos kórház alkalmazza a Med-AI szolgáltatásait. (Florida államban 25 millió beteg rekordot vizsgáltak meg felhasználásával.)

b.)KEFIR: Key Findings Reporter for Analysis of Healthcare Information (GTE Labs)

Eszköz: Information Harvester (Information Harvesting). A KEFIR-t arra fejlesztették ki, hogy automatikusan megtaláljon kirívó eltéréseket egy nagy, időben gyorsan változó kórházi adatbázisban, illetve annak jelentősebb attribútumaiban. Az eltéréseket a várt és múltbeli értékek alapján szűri ki. Ahol szükséges, a program egy megfelelő értéket generál és ajánl fel az általa hibásnak vélt érték helyére. A program további érdekessége egy Netscape-et használó grafikus felhasználói felület.

3. **Marketing alkalmazások**

a.) Vásárlási trendek feltérképezése (Dickinson Direct)

Eszköz: Information Harvester (Inf. Harvesting); induktív szabálygenerálás, fuzzy technika. A Dic-

kinson Direkt ugyan csak 230 főt foglalkoztató direkt marketing cég, de olyan nagy vállalat-óriásokkal áll kapcsolatban, mint az AT&T és az IBM. Az Information Harvester adatbányászó szoftvert arra használják, hogy a rendelkezésére álló történeti vásárlási adatok alapján kimutassák ügyfeleinek főbb vásárlási trendjeit, és szabályszerűségeket mutassanak ki a korábbi marketing tevékenység alapján. Adatbányászati módszerekkel meg tudják többek között állapítani a tipikus vásárló profilját, melynek alapján célorientált marketing stratégiát tudnak kidolgozni.

b.)Piackutatás (Reader's Digest Canada)

Eszköz: Knowledge Seeker (Angloss Software); döntési fa és induktív szabálygenerálás. A főbb piaci szegmensek meghatározása után meghatározták a profitábilis részeket, majd feltárták a változó kapcsolatokat (kölcsönhatásokat, értékcsoportokat, hierarchikus kapcsolatokat stb.). Ezzel erős piaci pozíciót tudtak szerezni.

4. **Kiskereskedelmi alkalmazások**

a.) Lottógépek felállítási helyének kijelölése (Automated Wagering, Inc.)

Eszköz: ModelMax (Advanced Software Applications); neuronhálózat. Egy kisvállalkozás megnyitásakor nagyon fontos a megfelelő helyszín kijelölése; ez olykor eldöntheti, hogy az adott vállalkozás sikeres lesz-e. Florida államban a ModelMax prediktív modellezését és a földrajzi elemzés módszereit együttesen alkalmazva sikeres előrejelzéshez jutottak lottógépek felállítási helyének meghatározásában.

b.) Gépkocsik sztereo berendezéseinek piackutatása (Washington Auto Audio, Inc.)

Eszköz: AIM - Abductive Information Modelling (AbTech Corp.); neuronhálózat és regressziós analízis helyett abduktív modellezés, fuzzy technika. Négy év (14000 ember) demográfiai adatait elemezve arra a kérdésre kerestek választ, hogy kik lehetnek a potenciális vásárlók - és ezeket célozták meg marketing akcióikkal.

5. **Bűnüldözési alkalmazás**

Pénzmosás kiszűrése - FAIS (US Dep. of Treasury, Financial Crimes Enforcement Network)

Eszközök: NetMap (Alta Analytics), és (2) Nexpert Object (Neuron Data); frame- és szabályalapú következtetés. Az USA nagy pénzügyi visszaéléseket felderítő hivatala a FAIS (FinCen AI System) alkalmazást abból a célból fejlesztette ki, hogy kiszűrje a gyanús pénzügyi tranzakciókat, a lehetséges pénzmosásokat. A FAIS egy (a Nexpert Object eszközzel készített) olyan szakértőrendszer, mely a NetMap fejlett adat vizualizációs eszközeivel erősen támogatja a döntéseket. Adatbányászati technikákat arra használták, hogy a legkülönbözőbb perspektívákból összefüggéseket találjanak, és hatékony formában jelenítsék meg a feltárt relációkat a felhasználó előtt, megkönnyítve a hatalmas adattömegben való tájékozódását.

6. **Tudományos alkalmazások**

a.) JarTool: a Vénusz krátereinek felderítése (JPL: Jet Propulsion Lab)

Eszköz: JPL Adaptive Recognition Tool (JPL and Caltech). A JPL adaptív felismerő eszköztárát alkalmazták a Vénusz SAR képeinek elemzésére, mely során sikeresen katalogizálták a Vénusz felszínén található kis vulkánokat.

b.) SKICAT: Az égi objektumok katalogizálása (JPL: Jet Propulsion Lab)
Eszköz: Sky Image Cataloging and Analysis Tool (nemzetközi fejlesztés); döntési fa. A SKICAT adatbányászó programban döntési fák alapján osztályozták égi objektumok millióit precízebben és gyorsabban, mint bármilyen (emberi) szakértő. Rövid idő alatt a SKICAT több százmillió égi objektumot katalogizált, és segítségével 10 új kvazárt fedeztek fel az univerzumban.

A mai adatbányászati eszközök nagy része különféle intelligens mintafelismerési technológiát használ, mint például neuronhálózatok, döntési fák, indukciós szabálykeresés, fuzzy logika. Ellentétben a tipikus algoritmusos adatbányászattal, ami a korábban említett AI technikákon alapul, könnyebben használhatók azok az adatbányász programok, amelyeknek jól meghatározott profiljuk van. Ilyen például az IVEE Development's Spotfire 1.0 rendszer, amely minden statisztikai változóhoz egy beállító csúszkát rendel, amelyekkel a változók egymásra hatása megvizsgálható és így az érdekes minták felfedezhetők. Arra is lehetőséget ad, hogy valamilyen háttérinformációval (pl. egy térképpel) együtt mutassa be az adatokat, hogy még könnyebb legyen felfedezni a mintákat. Az ilyen rendszerek mégis korlátozottabban használhatók, mint a hagyományosak – főként ha a neuron hálózatok flexibilitásához hasonlítjuk – mivel a használók a saját különleges adatbázisra épülő tudásukra korlátozottak. Másrészről, ahogy Chris Alberg az IVEE Development-től kimutatta, a legtöbb szakember jobban ismeri a saját üzletét, mint egy neurális hálózat. Valójában a neurális hálózat tanításának eredménye erősen függ attól az egyéntől, aki a tanítást végzi. A Cognos, az elemző folyamatok eszközeinek egyik vezető képviselője. A Right Information System egy neurális hálózaton alapuló software üzleti mintákra és előrejelzésekre. A SAS Intézet által kifejlesztett DMINE nevű termék a neurális hálózatok, a döntési fák és vizuális technikák kombinációja és az üzleti életrere irányul. Manapság a döntéskészítők sok időt töltenek az adatbázisok lekérdezésével. Annak érdekében, hogy az adatbányászatot elérhetővé tegyük szélesebb körű közönség számára és segítsük a döntéshozókat, hogy ezt a rendszert a saját PC-jükön futtathassák, a SAS Intézet egy olyan módszert támogat, amely öt alapvető lépésből áll: kiválasztás, megvizsgálás, kezelés, mintaelőállítás, becslés. A módszer az adatbázis egy részét használja, hogy csökkentse az eljárások futási idejét. Az adatok vizualizációja segíti a használót, hogy a helyes adatállományrészt válassza ki. Ha az adat túl összetett a grafikus bemutatásra, akkor a hagyományos statisztikai módszerek használhatók, mint pl. csoportosítás, összefüggés-elemzés. Az ilyenfajta adatvizsgálattal a felhasználó megtisztíthatja és aktuálissá teheti a kiválasztott részt, aztán futtathatja a kereső folyamatot, amely meghatározza azokat a legfontosabb ismertetőjeleket, amelyek a megadott adatállományhoz tartoznak. A folyamat utolsó lépéseként a használó értékeli a mintákat és a valósággal szembesítve ellenőrzi az értékét. Az Isoft az Alice nevű termékének egy leegyszerűsített változatát forgalmazza sikeresen. Az Alice széles körű statisztikai algoritmusokat használ. A program a minta eredményeit döntési fákban jeleníti meg, lehetőséget adva a használónak, hogy megértse az adatbázis belső viszonyait és könnyen ellenőrizhesse a feltevéseket. A döntési fák készítése nagyon vonzóvá teszi az adatbá-

nyászat folyamatát, ezzel az adatok nagy tömegét is eredményesen tudja kezelni. A döntési fák segítségével a felhasználó szét tudja választani, vagy egybe tudja olvasztani a csomópontokat, lecsökkentheti, vagy kibővítheti az ágakat és meghatározhatja a paraméterek számát egy fában. Ráadásul a felhasználó könnyen kiválaszthatja azokat az ágakat, amelyek érdeklik. A vizuális adatbányászat lehetővé teszi az algoritmusok összekapcsolását a személyes tapasztalattal és tudással, így a statisztikai eredményeket könnyebben le lehet fordítani életképes üzleti stratégiára. Ezen alapul a Cygron DataScope nevű rendszere, ami nem az algoritmusokra, hanem az emberi intuícióra koncentrálva interaktív játszadozást tesz lehetővé az adatokkal, mely közben értékes felfedezések születhetnek. A folyamatok megértése itt fontosabb, mint egy szabályrendszer létrehozása. A következő rész a DataScope vizualizációjával foglalkozik.

Az adatbányász rendszerek meg tudják mutatni az időbeli változásokat és azt, hogy néhány változó hogyan befolyásolja a többit. Mégis az a mód, ahogy ezeket az információkat kifejezik gyakran misztikus hatású és néhány üzletember számára nehezen érthető. Szükség van egy olyan nyelvre, amely közvetlenül az üzleti élet képviselői számára transzformálja a gépi tudást.

További adatbányász eszközökről tájékoztatók például az alábbi internet címeken találhatók:

- *http://www.kdnuggets.com/software/index.html*
- *http://www.datamining.hu*

5.3. Adatvizualizáció

Az adatbányászati eljárásokat két fő szempont szerint lehet értékelni: a törvényszerűségek megtalálása, ill. az eredmények értelmezhetősége szempontjából. Az utóbbi az eredmények cognitív aspekusa, amely a vezetők számára sokkal jelentősebb, mint pl. egy nagy pontosságú statisztikai mintavételen alapuló szabály. A vizualizáció éppen ezért került a középpontba. Több módszer alkalmazható az adatok vizualizálására, melyek közül megemlítünk néhány példát. Egy konkrét alkalmazás a DataScope szoftver, mely egyedülálló módon, vizuálisan valósítja meg a lekérdezést. 1997-ben Európai Információ Technológiai díjjal tüntették ki innovatív megoldásáért. A továbbiakban a DataScope főbb jellegzetességit ismertetjük.

5.3.1. Vizualizációs technikák

A statisztikát illetve a számítógépes grafikát széles körben alkalmazzák numerikus adatok vizuális ábrázolására, mint például diagramok készítésére vagy egyéb fejlettebb eljárásokkal történő ábrázolásra, többek között Andrews görbe, Chernoff arc vagy Korhonen harmonikus ház módszereinek felhasználásával. Ezek a technikák segíthetnek az adatok osztályozása (klaszterezése) során.

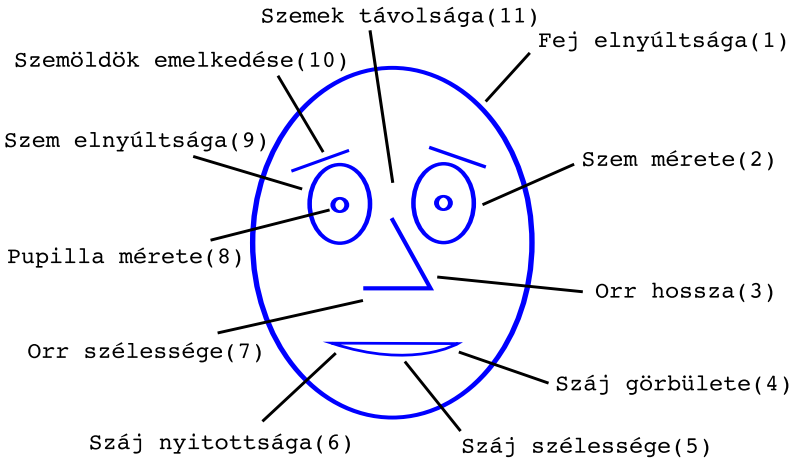
- Andrews görbéje:

  Ezzel a módszerrel minden adattényező egy kétdimenziós harmonikus görbeként ábrázolt, a változók száma korlátozatlan. A harmonikus görbe a változók alkalmazási sorrendjétől függ. Az (5.1) képlet trigonometrikus sor.

$$U(t) = c_1 \cdot \sin(1t) + c_2 \cdot \sin(2t) + \dots \tag{5.1}$$

- Chernoff Arc:

  Chernoff emberi arc formájában ábrázolt többtényezős adatokat. Az arc különböző részeinek megfeleltetett információ határozza meg az arcrész formáját, nagyságát, stb. (5.2. ábra).



5.2. ábra. Példa Chernoff arc részeire

A Chernoff arc jól definiált geometriai alakzatokból áll össze (körívek, ellipszis ívek, egyenes vonalak, stb.). Chernoff eredetileg 18 paramétert javasolt, de az arcokat kevesebb, körülbelül minimum 10 paraméter esetén már meg lehet rajzolni. Vegyünk például egy autós adatbázist, melyben minden autó–rekordhoz 11 numerikus érték tartozik.

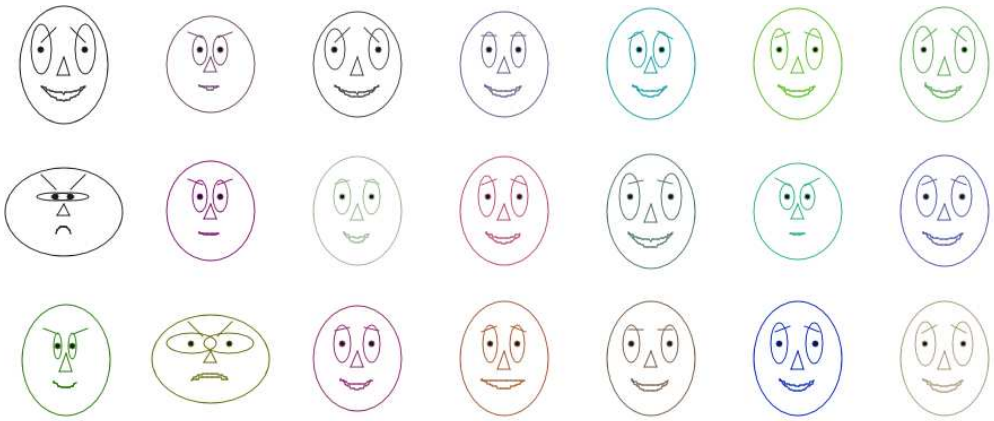
arcrész	3	5	2	6	8	1	10	
autó	ár	fogy.	lóerő	csomagt. nagysága	CO_2 kib.	személyek száma	gyorsulás	...
Audi	13	9	110	60	8	5	9	
Fiat	2	6	72	32	6	5	12	

Az értékek megfelelő arcrészhez rendelése után meghatározhatjuk a „szép” archoz tartozó értékeket, majd minden rekordhoz készíthetünk egy arcot, amelyeken az arcrészek „torzultsága” jeleníti meg az optimálisnak választott értékektől való eltérést. Különböző rekordokhoz rendelt Chernoff arcok közötti különbségek láthatók az 5.3. ábrán.

Chernoff módszerének legnagyobb előnye, hogy az arcon sok jellegzetességet egyidejűleg tudunk megjeleníteni. Természetesen a „szép” arc meghatározása egyáltalán nem triviális feladat. A szimmetria megsértése általában súlyos problémákat tükröz.

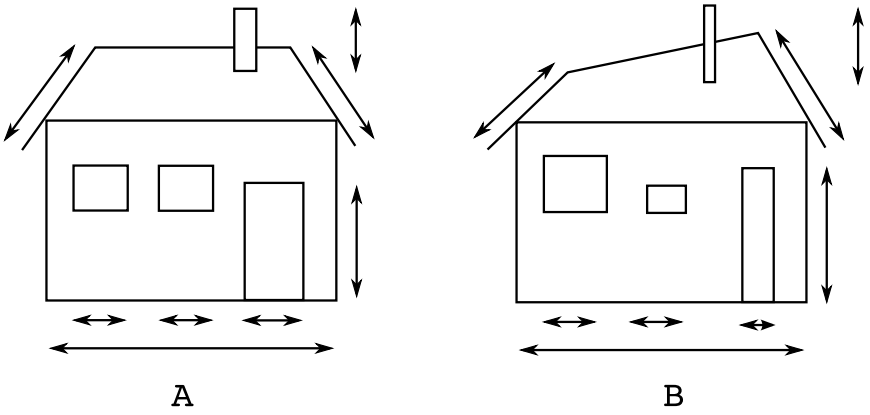
- Korhonen harmonikus ház:

  Korhonen a többtényezős információkat házak formájában vizualizálta (5.4. ábra). Ez a leképezés is alkalmas az adatok kiértékelésének megkönnyítésére. Különböző cégek adatait egy-egy házként



5.3. ábra. Különböző Chernoff arcok.

vizuálisan ábrázolva a megjelenített képi információk alapján a megkérdezettek 98%-ban meg tudták határozni azt az 5 céget reprezentáló házat, amelyek csődbe jutottak. Ugyanez a numerikus adatok alapján nem sikerült nekik.



5.4. ábra. Korhonen harmonikus ház: A-harmonikus, B-kevésbé harmonikus

Statikus változat: egy ház minden céghez. Pillanatnyi állapotot tükröz, változást nem.
Dinamikus változat: animáció a házon (cég mutatóin) történt változások szemléltetésére

A deformált házak rossz értékeket takarnak. Természetesen nem egyértelmű, sőt, általában kifejezetten nehéz kritériumokat úgy hozzárendelni a házakhoz, hogy a „szép” ház legyen a legjobb alternatíva. Többtényezős döntések esetén a legtöbb tényezőt úgy kezeljük, hogy minél nagyobb (vagy kisebb) az érték, annál jobbnak tekintjük a döntés szempontjából. A harmonikus ház azonban nem tesz eleget ezeknek az előfeltételeknek, mivel a „szépség” nem áll arányban a méretekkel, hanem inkább a különböző részek közötti arányosságtól függ.

5.3.2. Mi a DataScope?

A DataScope egy Microsoft Windows 3.1 alkalmazás, melynek segítségével adatbázisainkon hatékony elemzéseket végezhetünk. A szoftver grafikusan megjeleníti egy tetszőleges adatbázis tartalmát, és sokféle eszközzel támogatja az egyes adatbázis rekordok közötti viszonyok tanulmányozását, valamint a (pozitív

vagy negatív értelemben) kivételes tulajdonságokkal rendelkező alternatívák kiválasztását. A DataScope használható grafikus on-line lekérdező rendszerként is. Különböző szempontok szerinti szűréseket (lekérdezéseket) is végezhetünk vele és a munkát a leszűrt adatokon folytathatjuk. A rendszer segítségével az adatok közötti összefüggések sokkal jobban és gyorsabban feltérképezhetők, így hatékonyabb döntések hozhatók. Ezt egy példán keresztül mutatjuk meg. Tekintsünk egy adatbázist, amelynek rekordjai autók néhány adatát tartalmazzák. Az egyes autókról a következő adatok állnak rendelkezésre: név, ár, teljesítmény, fogyasztás, hengerűrtartalom és a használt üzemanyag fajtája. Ezekre a mezőkre a bemutatás során hivatkozunk. Felsorolunk néhány tipikus kérdést, amivel a vezetők az adatbázishoz fordulnak és eddig csak lekérdező nyelv segítségével voltak megvalósíthatók:

- Melyik a legdrágább/legolcsóbb autó?
- Melyik a legdrágább/legolcsóbb dízel autó?
- Drága-e az autó?
- Melyek azok az autók, amelyeknek a fogyasztása kisebb, mint 7 l/100 km és ára alacsonyabb 20000 márkánál?
- Hogyan lehet néhány autót sok tulajdonság szerint könnyen összehasonlítani?
- Mi az ára egy közepes árú autónak?
- Mi az ára egy közepes árú dízel autónak?
- Igaz-e, hogy a dízel autók általában drágábbak, mint a benzinesek?
- A 20000 márkánál olcsóbb autóknak hány százaléka dízel?
- Hány autó elégít ki egy bizonyos feltételt?
- Hány autónak nincs az áráról adat?
- Mi a dízel és a nem dízel autók aránya az adatbázisban?
- Van-e kapcsolat a fogyasztás és a teljesítmény között? Melyek a kivételek?
- Hogyan található olyan autó, amelynek ára 20000 márka körül van, és teljesítménye minél magasabb?
- Hogyan lehet egyszerűsíteni a munkát, ha csak a dízel autókkal akarunk foglalkozni?
- Hogyan jelölhető ki az összes olyan autó, amelynek áráról nincs adat?
- Hogyan jelölhető ki lokálisan a 10 legnagyobb teljesítményű autó?
- Ha van egy diszkrét mezőnk, és pontozni tudjuk az egyes kategóriák jóságát, hogyan készíthetünk olyan numerikus mezőt, amelyben ezek a jósági értékek szerepelnek?

A fenti kérdések DataScoppal vizuálisan, egér mozgatással megvalósíthatók és a válasz vizuálisan áll rendelkezésre. A program nemcsak az adatbázis egyes rekordjainak elemzésére alkalmas. Lehetőséget biztosít csoportos kiválasztásra, szűrésre is. Bármely ablakban (azaz bármelyik tulajdonság szerint) kiválaszthatunk tetszőleges rekordokat. Ezt lokális kijelölésnek nevezzük. A lokális kijelölések összességéből az unió (vagy), vagy a metszet (és) operátor segítségével képezhető a globális kijelölés (eredmény). Így például könnyen kijelölhetjük az olcsó és kis fogyasztású autókat. A többi ablak is mutatja, hogy melyek a kiválasztott elemek, így ezután megvizsgálhatjuk ezek teljesítménymutatóit, vagy a relációs diagramok alapján megkereshetjük a kivételes tulajdonságokkal rendelkezőket. Készíthetünk egy új projektet is, amely csak a globálisan kijelölt rekordokat tartalmazza, ezzel kiszűrve az érdektelen alternatívákat és áttekinthetőbbé téve adatainkat.

5.3.3. Kijelölések, szűrések

Mielőtt a mezőket megjelenítő ablakok használatával foglalkoznánk, meg kell ismernünk néhány fogalmat. Mint a bevezetőből tudjuk, a DataScope segítségével nemcsak egyes rekordok tulajdonságait vizsgálhatjuk, hanem rekordcsoportokét is. Ehhez ki kell jelölnünk a vizsgálandó rekordokat. A kijelölésnek két típusa létezik: a lokális (feltételrendszer) és a globális (eredmény) kijelölés.

Lokális kijelölés

Bármelyik mező szerint kiválaszthatunk bizonyos rekordokat. Például, ár szerint kijelölhetjük a 15000 és 20000 márka közötti autókat, vagy az azonosító mező alapján bizonyos autókat (pl. az összes Audit és BMW-t), vagy az Üzemanyag mező szerint az összes dízel autót. Mivel akár minden rekordról egyesével megmondhatjuk, hogy az kijelölt-e (kivéve a diszkrét mezőket, ahol csak egész kategóriákat jelölhetünk ki), ez a kijelölés tetszőlegesen bonyolult kritériumok szerint történhet. Ezt a kijelölési formát a mező szerinti lokális kijelölésnek nevezzük. Egyidejűleg több mező szerint is elvégezhetjük. Az egyes mezők szerinti lokális kijelölések függetlenek egymástól és bármikor módosíthatók. Minden mezőablak jelzi, hogy az általa képviselt mező(k) szerint milyen lokális kijelöléseket végeztünk. Az azonosító ablakoknak (ld. később) kitüntetett szerepük van, mivel ezek mindig az éppen aktív ablak szerinti lokális kijelölést jelzik. A státussorban megjelenik az aktív ablakban megjelenített mező lokálisan kijelölt rekordok száma ill. az, hogy ezek az összes rekordnak hány százalékát képviselik. A figyelmünket felkeltő rekordok megjelölhetők egy, legfeljebb kétbetűs azonosítóval. A jelek az összes diagramon megjelennek, így a továbbiakban a rekordok könnyen nyomon követhetők. A jelölés lehetőséget ad a rekordok tulajdonságainak összehasonlítására is (pl. néhány megjelölt autó közül azonnal látszik, hogy melyik a legolcsóbb).

Globális kijelölés vagy eredmény

Globális kijelölést a lokális kijelölések együttes figyelembe vételével az Unió vagy Metszet művelet végrehajtásával hozhatunk létre. Egyidejűleg csak egy globális kijelölés létezhet. A globális kijelölés használatával meghatározhatjuk azokat az adatbázisrekordokat, amelyek egy adott feltételkombinációnak eleget tesznek.

A metszetképzés az "és" típusú kapcsolatok megadására szolgál. Ha az előbb említett két lokális kijelöléssel rendelkezünk (15000 márkánál alacsonyabb árú autók és 7 liter alatti üzemanyagfogyasztású autók), akkor ezek metszetét véve megkapjuk az olcsó és kis fogyasztású autókat. (A metszetben szereplő autók mindkét feltételnek eleget tesznek.) Az unióképzés segítségével "vagy" típusú kapcsolatokat állíthatunk fel. Például, ha lokálisan kijelöljük a 15000 márka alatti autókat az árat képviselő ablakban, valamint a 7 liternél kisebb fogyasztásúakat a megfelelő ablakban, akkor az unióképzéssel megkapjuk az olcsó vagy kis fogyasztású autók csoportját. A lokális kijelöléseket bármikor módosíthatjuk és új globális kijelölést hozhatunk létre. Szükség lehet erre például, ha nincs olyan autó, amelyik megfelelne minden megadott feltételnek. Ekkor a metszetképzés üres halmazt eredményez. Ilyen esetekben megpróbálhatjuk bővíteni valamelyik lokális kijelölést, és újraképezni a metszetet. A státussorban megjelenik a globálisan kijelölt rekordok száma, és hogy ezek az összes rekordnak hány százalékát képviselik.

5.3.4. Adatbázismezők típusai

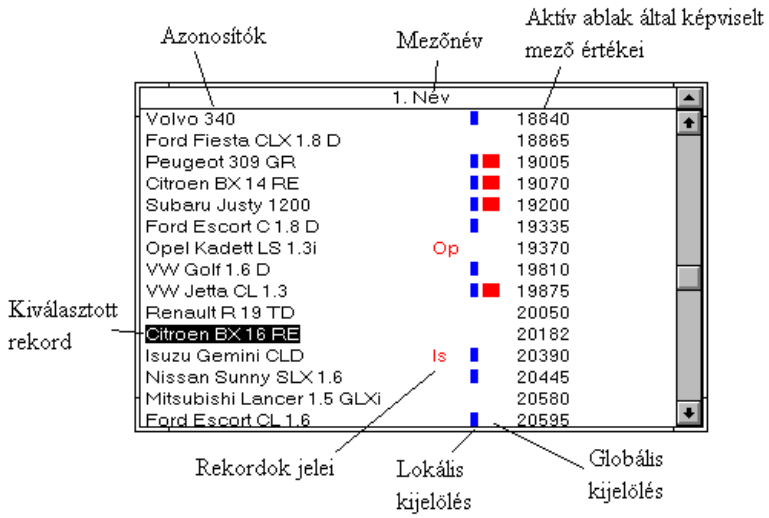
A DataScope segítségével megjeleníteni, vagy elemezni kívánt adatbázisok különféle mezőket tartalmazhatnak. A megjeleníteni kívánt adatbázis mezői három típusba sorolhatók: Azonosító, numerikus, diszkrét mező.

Azonosító mező

Ennek alapján történik az egyes rekordok azonosítása. (A példában ez az autók megnevezése.) Az azonosító mező az adatbázis rekordjainak egyedi azonosítására szolgál. Fontos, hogy értéke minden rekordnál különböző legyen, vagy legalábbis, csak igen ritka ismétlődések forduljanak elő. Egy személyek adatait tartalmazó adatbázisban a rekordok azonosíthatók a személyek nevével, vagy személyi számával, vagy akár mindkettővel (több azonosító mezőt is megadhatunk). A projektnek nem kötelező azonosító mezőt tartalmaznia, nélküle azonban a DataScope lehetőségei leszűkülnek tendenciák megállapítására. A konkrét rekordok vizsgálata nem lehetséges. Az azonosító mező tartalma listában jelenik meg (lásd az 1. ablakot az 5.5. ábrán).

A listában a mezőtartalom mellett az éppen aktív ablak által képviselt mező tartalma is szerepel és a lista ez utóbbi mező szerint rendezett. (A képen az autók ár szerint vannak rendezve, az árak az autók neve mellett láthatók.) Ha maga az azonosító ablak az aktív, akkor a lista ábécé sorrendbe rendezett; Ha valamely más mezőablak az aktív, akkor a lista az aktív ablak által képviselt mező értékei szerint rendezett. (Mivel a rendezések egy előfeldolgozó lépés során megtörténnek, ez nem igényel időt.) Így az adatokat tetszőleges szempont szerint rendezve vizsgálhatjuk.

Az éppen kiválasztott rekordot inverz sáv jelzi. Az azonosító ablakban minden rekord neve mellett két négyzet található. Az egyik négyzet a lokális, a másik a globális kijelölések jelzésére szolgál. Ha a jobb oldali (nagyobb) négyzet megjelenik, akkor az adott rekord beletartozik az éppen aktuális globális kijelölésbe. Hasonló ugyan a lokális kijelöltséget jelző kisebbik négyzet szerepe is, mégis van egy fontos különbség: ez a négyzet mindig az éppen aktív ablak szerinti lokális kijelölés állapotát mutatja. Ha egy



5.5. ábra. DataScope 1. ablak: az azonosító mező tartalma listában.

másik ablakot aktiválunk, akkor az abban érvényes lokális kijelölés jelenik meg az azonosító ablakban. (Ez azért hasznos, mert így láthatjuk az aktív mezőablak lokális kijelölésébe tartozó rekordokat azonosítójuk szerint is.) Az azonosítók mellett láthatjuk a rekordhoz rendelt betűjelet is, ha vannak ilyenek. Az utolsó oszlopban jelennek meg az aktív ablak által képviselt mező(k) értékei.

Ha egy rekordnak különös figyelmet kívánunk szentelni, és szeretnénk azt a továbbiakban egyszerűen nyomon követni, célszerű hozzárendelni egy azonosító jelet. Az azonosító jel minden ablakban megjelenik, így a rekord könnyen nyomon követhető. Ez nagyon hasznos lehet például akkor, ha néhány rekord viszonyát tanulmányozzuk: mindegyiket megjelölve anélkül láthatjuk elhelyezkedésüket, hogy meg kellene őket keresnünk a különböző ablakokban.

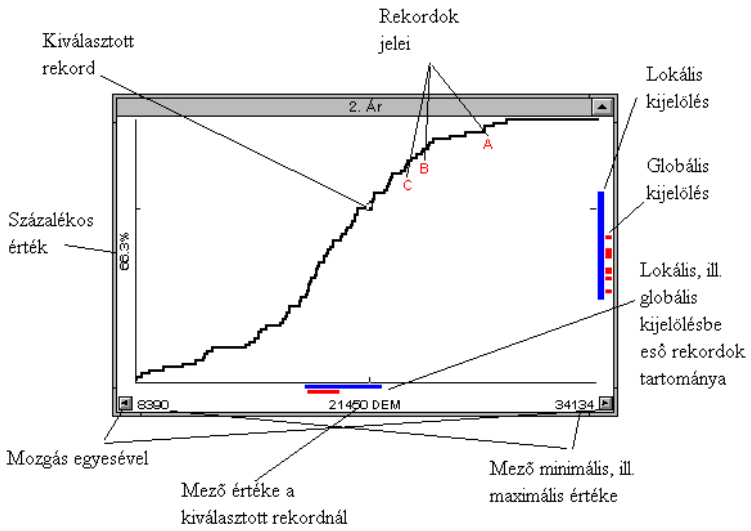
Az azonosító ablak általában a szűrési műveletek végén hasznos. Ha nem csak tendenciákat kívánunk megállapítani, hanem konkrétan kíváncsiak vagyunk, hogy melyek azok a rekordok, amelyek feltételeinknek eleget tesznek, akkor végiglépkedhetünk a globálisan kijelölt rekordokon, és az azonosító ablakban mutatja, hogy melyek ezek. Ha a relációs ablakban (lásd később) egy pontra rákattintunk, akkor szintén az azonosító ablakban állapíthatjuk meg, hogy melyik rekordot képviseli.

Numerikus mező

Értéke tetszőleges számadat lehet. (Ilyen például az ár vagy a fogyasztás.) A DataScope képes feldolgozni olyan adatbázist is, ahol nem áll minden rekordról rendelkezésre az összes adat ("nincs adat" mezőérték). Numerikus adatok esetén a DataScope az adatok eloszlásfüggvényét ábrázolja, grafikonja lépcsős szerkezetű.. (Ilyen például, az autók ára, fogyasztása; személyeket tartalmazó adatbázisnál a bér, az életkor, stb.) A képen ilyen a 2. ablak. Az X tengelyen az adatok a legkisebb értéktől a legnagyobbig vannak ábrázolva. Az Y tengelyen százalékos ábrázolást láthatunk, ahol a 0% képviseli a legkisebb adatot, a 100% pedig a legnagyobbat. A százalékos érték azt mutatja, hogy az adatbázis rekordjainak hány százaléka előzi meg sorrendben a rekordot (a sorrend a rendezettségtől is függ). Például, ha (csökkenő rendezettség esetén) egy autó árához 40% tartozik az Y tengelyen, akkor az adatbázis autói közül 40 százalék ára magasabb (a

csökkenő rendezettség miatt a legdrágább autó az első). Más szóval, az Y tengelyről a kiválasztott rekord helyezését olvashatjuk le.

Az 5.6. ábráról leolvasható az éppen kiválasztott rekord adott mezője értékének viszonya a többihez. (A képen az 1. ablakban az Audi 80 D a kiválasztott és a 2. ablak grafikonján a kis négyzet azt jelzi, hogy ennek az autónak az ára igen magas a többiéhez képest. Az X tengely alatt leolvasható a konkrét ár (27850 DEM), az Y tengely mellett pedig az, hogy az autók 95.2%-a ennél olcsóbb.) Fordítva, a grafikon tetszőleges pontjára rámutatva a névlista beáll a ponthoz legközelebb eső rekordra, a többi ablak pedig megmutatja a rekord egyéb tulajdonságait.



5.6. ábra. DataScope 2. ablak: a grafikon az ár szerinti empirikus eloszlásfüggvény.

Az éppen kiválasztott rekord helyét egy kis négyzet jelzi a függvénygörbén. Fontos, hogy több olyan rekord is lehet, amelynek az ablak által képviselt mezője azonos értékű, így egy bizonyos X értékhez több rekord is tartozhat. Így előfordulhat, hogy tovább mozgunk az adatbázisban, de a kis négyzet nem mozdul, ha a következő rekordnál sem változott a mező értéke. Az ablak alsó részén látjuk a kiválasztott rekord mezőjének értékét, valamint a legkisebb és a legnagyobb mezőértéket. A bal szélén a kiválasztott mező értékéhez tartozó százalékos érték (az eloszlásfüggvény értéke a kiválasztott pontban) látható. A kijelöléseket az ablak jobb szélén és alján lévő területeken láthatjuk. Az ablak jobb szélén két egymás melletti oszlop található. A bal oldali a lokális, a jobb oldali pedig a globális kijelölést jelzi.

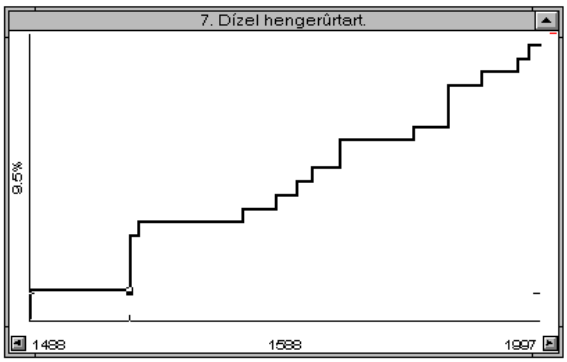
Az ablak alján lévő vízszintes téglalapok mindig folytonosak, és az első lokálisan (globálisan) kijelölt rekordtól az utolsóig tartanak. Segítségükkel leolvasható, hogy a kijelölt rekordok az adott mező szerint milyen értéktartományba esnek. Például, ha az 'Ár' ablakban a legolcsóbb és a legdrágább autó globálisan kijelölt, a felső vízszintes sáv az ablak bal szélétől a jobb széléig terjed. Az eloszlásfüggvény alatt lévő vízszintes sáv megmutatja, hogy az ablak által képviselt mező értékei milyen tartományba tartoznak a legutolsó szűrés szerint. Például, ha kiválasztjuk a dízel, 70 lóerőnél nagyobb teljesítményű autókat, akkor az árat képviselő ablak alján láthatjuk, hogy a feltételnek megfelelő autók árai milyen intervallumba esnek, és kiválasztható legolcsóbb.

A globálisan kijelölt rekordok helyét mutató függőleges sáv az utoljára megadott feltételt kielégítő

rekordok elhelyezkedését mutatja. Ha globálisan kijelöljük a nagy teljesítményű autókat, az 'Ár' ablakban láthatjuk, hogy ezek nagy része magas árú (a függőleges vonal felső részén több lesz a kitöltött terület).

Az eloszlásfüggvényről más információk is leolvashatók. Ha az eloszlásfüggvény szokatlanul hosszú vízszintes vonalat tartalmaz, az azt jelenti, hogy az X tengely ezen intervallumához nem tartoznak rekordok. Például a fenti képen a jobb felső sarokban látható egy hosszú egyenes szakasz, tehát itt egy nagyobb tartományba egyetlen autó ára sem esik.

A hosszú függőleges vonal azt jelzi, hogy az adott X értékhez nagyon sok rekord tartozik. Az 5.7. ábrán például a dízel autók hengerűrtartalmát ábrázoltuk. Láthatjuk, hogy nagyon sok az 1588 köbcentiméteres autó. Ez egy régebbi (nyugat-európai) korlátozás eredménye, amely az 1600 cm3 feletti dízel autókat megadóztatta. Utána egy hosszú vízszintes vonalat is látunk, amely mutatja, hogy egy nagyobb tartományba nem esnek autók.



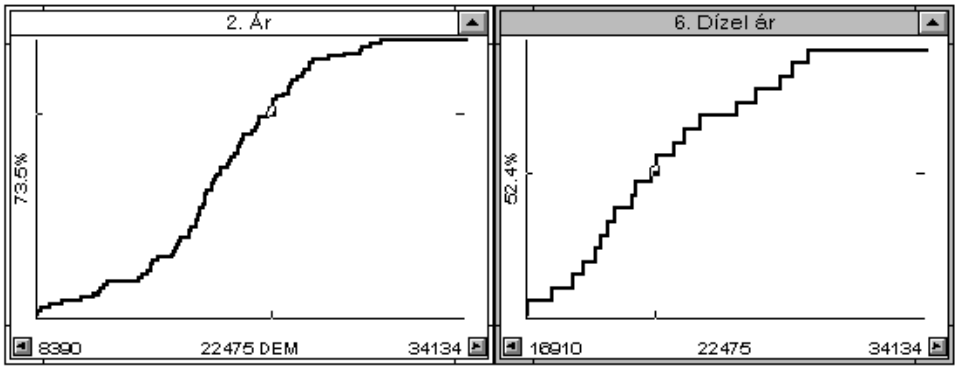
5.7. ábra. DataScope: a grafikon a Dizel hengerűrtartalom szerinti empirikus eloszlásfüggvény.

A vízszintes, illetve függőleges vonalat természetesen szabadabban is értelmezhetjük (főleg mivel sok rekord esetén az eloszlásfüggvény nem ilyen tagolt). Ha például sok 1580 és 1600 cm^3 közötti autó szerepelne, akkor a vonal nem lenne teljesen függőleges, de csak kis vízszintes irányú eltérések lennének. Más szóval, az eloszlásfüggvény azon a helyen meredekebben emelkedne. Ugyanez igaz a vízszintes esetre is: ha az eloszlásfüggvény nem, vagy csak alig emelkedik, akkor kevés rekord esik abba az értéktartományba. Az eloszlásfüggvény egy másik tulajdonsága: az a kijelentés, hogy közepes árú diezel autók drágábbak, mint a benzinesek, vizuálisan pontosan megmutatja. Válasszuk ki az 50%-hoz legközelebb eső árú autót a Dízel ár ablakban (5.8. ábra).

Amint látjuk, a dízel autók esetében az 52.4%-os ár 22475 DEM. A DataScope szinkronitási elve miatt most ugyanez a rekord kiválasztódott az Ár ablakban is, itt azonban százalékos értéke 73.5%. Ez azt jelenti, hogy ami egy dízel autónál közepes árnak számít, az az összes autó között már a magas árkategóriát képviseli. Vagyis a dízel autók drágábbak. Ehhez hasonló következtetések levonására alkalmas a feltételes mező.

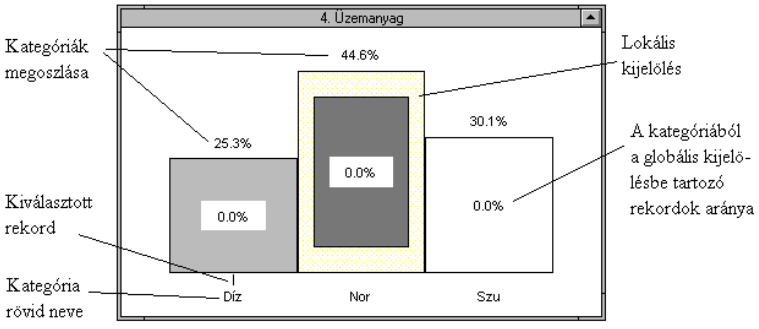
Diszkrét mező

Tartalma általában néhány különböző érték lehet. (A fenti példában a használt üzemanyag fajtáját – dízelolaj('D'), normál benzin('N'), szuperbenzin('S') – tartalmazó mező ilyen.) A diszkrét adatok különféle



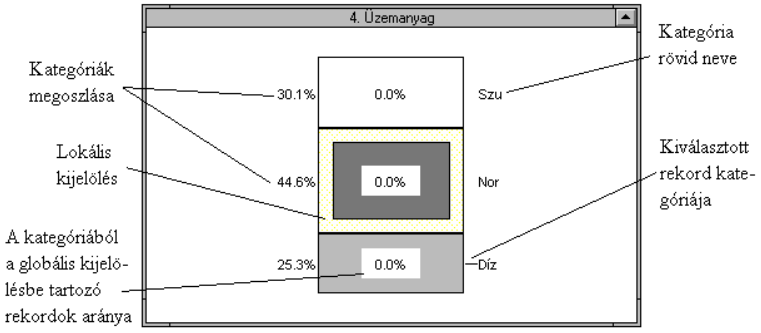
5.8. ábra. DataScope: szinkronitásnak köszönhetően mindkét ablakban kiválasztódik a rekord.

módokon ábrázolhatók (pl. kördiagram, oszlopdiagram, stb.). A DataScope itt is jelzi, hogy az éppen kiválasztott rekord melyik kategóriába tartozik. (Az 5. ablakban a diagram alatt a "D" (dízel) tartományban látható egy jelzővonalka.) A diszkrét kategóriákat össze is vonhatjuk. (Például a különféle benzintípusoknak (normál, szuper) megfelelő felosztás helyett tekinthetünk egyszerűen dízel- és benzinüzemű autókat.) Ha egy mező lehetséges értékei nem számok, vagy az előforduló értékek számok ugyan, de csak néhány különböző szám, akkor érdemes diszkrét típusúnak definiálni a mezőt (ha az értékek nem számok, akkor kizárólag diszkrét típusú lehet). Ilyenkor a DataScope kategóriákat képez az adatbázis rekordjaiból. Egy kategóriába azok a rekordok tartoznak, amelyeknél az adott mezőben szereplő érték azonos. Az autós adatbázisban diszkrét mező a használt üzemanyag típusa (Üzemanyag mező), amelynek lehetséges értékei a D, N, S betűk. Más adatbázisokban ilyen lehet mondjuk egy olyan mező, amely rekordokat osztályoz (pl. A, B, C osztályon dolgozó személy; I., II. vagy III. osztályú áru, stb.) A megjelenítésre több lehetőség van. Az oszlopdiagrammon az egyes kategóriák százalékos arányát láthatjuk (5.9. ábra). A kiválasztott rekord helyét itt is egy vonalka jelzi (a képen az D terület alatt). A lokálisan kijelölt kategóriákat vastagabb keret jelzi (D kategória). Az oszlopok belsejében látható, hogy az egyes kategóriákból mekkora rész tartozik a pillanatnyilag érvényes globális kijelölésbe.



5.9. ábra. DataScope: oszlopdiagramm az üzemanyag kategóriák százalékos arányát mutatja.

A másik megjelenítési mód az eloszlásdiagram (5.10. ábra). Ez egy állandó magasságú oszlop, amely az egyes kategóriák arányában van felosztva. A kiválasztott rekord és a lokális kijelölés jelzése az oszlopdiagramnál leírt módon történik. Hasonlóan az oszlopdiagramhoz, itt is látjuk, hogy az egyes kategóriák mekkora része esik az éppen aktuális globális kijelölésbe.



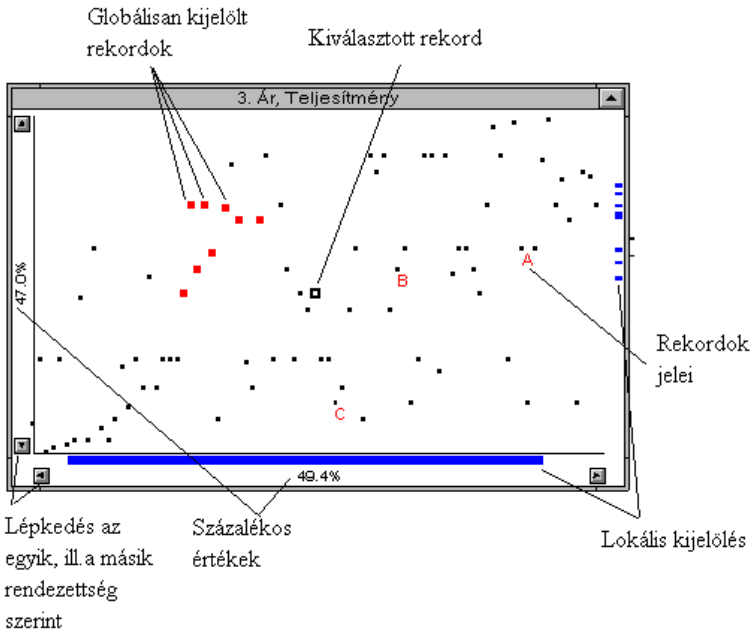
5.10. ábra. DataScope: eloszlásdiagramm az üzemanyag kategóriák arányában felosztva.

A diszkrét típusú mezőt megjelenítő diagramokat a szokásos módon használhatjuk statisztikai következtetések levonására. A diagramokon kényelmesen jelölhetünk ki kategóriákat (csak egész kategóriákkal dolgozhatunk) és végezhetünk velük szűrési műveleteket. Például egy kattintással kijelölhetjük a dízel autókat, és a Kijelölés menü segítségével módosíthatjuk globális kijelölésünket. Diszkrét mezők esetében a lokális kijelölés kategóriánként történik, vagyis egy egész kategóriát jelölünk ki egyszerre. Ez gyorsítja az egyes kategóriák szerinti szűréseket. Ha létrehoztunk egy globális kijelölést, az oszlopdiagramon láthatjuk, hogy az hogyan oszlik meg az egyes kategóriák között. Így láthatjuk, hogy egy bizonyos feltételrendszert kielégítő rekordokból mennyi esik az egyes kategóriákba. Például ha kiválasztjuk a 20000 márkánál olcsóbb autókat és belőlük globális kijelölést képzünk, akkor az Üzemanyag ablakban láthatjuk, hogy ezek az autók nagyrészt a benzines kategóriába esnek. Ha kívánjuk, a diszkrét kategóriánkat össze is vonhatjuk. Ha két kategóriának ugyanazt a rövid nevet adjuk, azok ideiglenesen összeolvadnak. Például megtehetjük, hogy a normál és szuper benzinnel üzemelő autókat egybevonjuk úgy, hogy mindkét kategóriának a 'B' rövid nevet adjuk. Ezután csak D és B jelű kategóriák lesznek.

Relációs diagram

A leghatékonyabb megjelenítési mód az ún. relációs diagram (3. ablak az 5.11. ábrán). Ez két numerikus adatmező viszonyának elemzését teszi lehetővé. Az egyik tengelyen az egyik adatmező értékeit ábrázoljuk a minimálistól a maximálisig, a másik tengely ugyanígy képviseli a másik adatmezőt. Az adatbázis rekordjait egy-egy pont jelzi, amely a rekord két mezője eloszlásfüggvény értékének metszéspontjában helyezkedik el. A relációs diagram lehetőséget ad a kivételek gyors kiszűrésére. Minél távolabb van egy pont az átlótól (az $y = x$ egyenestől), annál inkább eltér az adott rekord az átlagtól. A fenti példában a 3. ablak az ár (vízszintes tengely) és a teljesítmény (függőleges tengely) mezők kapcsolatát ábrázolja. A listából kiválasztott Audi 80 D helyét egy kettős négyzet jelzi. Az ábráról leolvasható, hogy ennél az autónál igen magas árhoz alacsony teljesítmény (14.5%) tartozik, ami kedvezőtlen tulajdonság. Azt is láthatjuk, hogy az autók túlnyomó részénél a két tulajdonság között lineáris a kapcsolat, és főképp negatív kivételek vannak. Ha a diagramon kiszemelünk egy számunkra érdekes pontot, rákattinthatunk az egérrel. A többi ablak (köztük a névlista) azonnal megmutatja, hogy melyik rekordról van szó és milyenek az egyéb tulajdonságai. Természetesen több különböző relációs diagramot is megnyithatunk és az egyiken egy kivételes rekordot kiválasztva,

megállapíthatjuk, hogy az más szempontok szerint is eltérő-e az átlagtól.

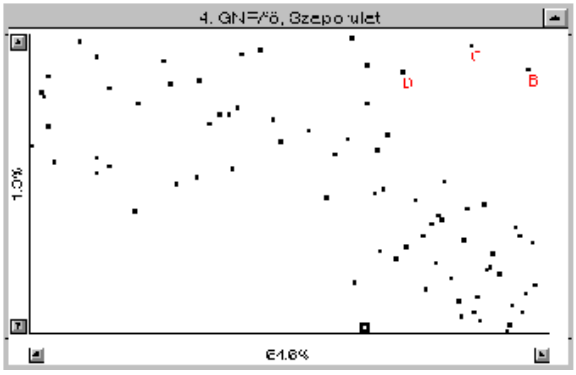


5.11. ábra. DataScope 3. ablak: két numerikus adatmező viszonyának elemzését teszi lehetővé.

Az ablak átlói fontos választóvonalak, ezek mentén helyezkednek el ugyanis azok a rekordok, amelyeknél a két tulajdonság szerint elért helyezés hasonló (pl. átlagos ár/teljesítmény viszony). Minél távolabb helyezkedik el egy pont ettől az egyenestől, annál inkább kivételes az általa képviselt rekord.

Ez az ablaktípus tehát jó lehetőséget ad a kivételek gyors kiszűrésére. Ha ábrázoljuk két mező viszonyát, akkor azonnal szembeötlenek a kiugró pontok. Egy pontra rákattintva kiválaszthatjuk a hozzátartozó rekordot, és megnézhetjük az azonosító ablakban, hogy név szerint melyik az. Más ablakokban pedig láthatjuk annak egyéb tulajdonságait. Ha több relációs ablakot is megnyitottunk, akkor ezekben azonnal láthatjuk azt is, hogy más szempontok szerint is kivételes-e az adott rekord.

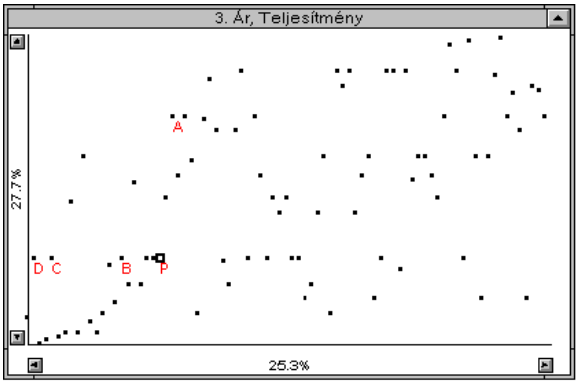
A relációs ablak a kivételek mellett a tendenciákat is mutatja. Egy pillantással megállapíthatjuk, hogy milyen jellegű kapcsolat van a két tulajdonság között, és hogy ez mennyire "erős", azaz hogy mennyi olyan pont van, amely nem követi ezt a tendenciát.



5.12. ábra. DataScope: országok nemzeti jövedelmének és szaporulatának aránya.

Az 5.12. ábra például a Világ országainál mutatja a nemzeti jövedelem és a szaporulat arányát (1988-

as adatok). Jól látható a tendencia, amely szerint minél gazdagabb egy ország, annál kevesebb gyermeket vállalnak. Megjelöltünk néhány kivételt is: B - Kuvait, C - Líbia, D - Venezuela. Középen alul a kiválasztott rekord Magyarország. A további következtetések levonását az Olvasóra bízzuk. Azt is könnyen megvizsgálhatjuk, hogy létezik-e egy bizonyos rekordnál jobb választás. Ábrázoljuk az autók ár/teljesítmény viszonyait egy relációs diagrammon, mindkettőt növekvő rendezettséggel (5.13. ábra).



5.13. ábra. DataScope: autók ár/teljesítmény viszonyainak relációs diagrammja növekvő rendezettséggel.

Vizsgáljuk meg a fenti képen a "P" betűvel jelölt autót. Tegyük fel, hogy szeretnénk egy hasonló árú, de nagyobb teljesítményű autót. Mivel az árat a vízszintes tengelyen ábrázoltuk, ezért az azonos árú autók azon a függőleges egyenesen helyezkednek el, amelyik átmegy a "P" ponton. Láthatjuk, hogy bár ezen az egyenesen nincs másik pont, jóval a "P" fölött, egy kicsivel jobbra van egy érdekes pont, amelyet "A"-val jelöltünk meg. Az "A" pont által képviselt autó egy kicsivel drágább ugyan, viszont a teljesítménye jóval nagyobb. Ezután eldönthetjük, hogy megéri-e nekünk az ártöbbletet ez a teljesítménynövekedés, és persze meg kell vizsgálnunk a másik autó egyéb tulajdonságait is. Azt is láthatjuk a képen, hogy a "P"-vel azonos teljesítményű autót olcsóbban is kapunk (pl. a "B", "C" és "D" jelű autók ilyenek). Természetesen itt is meg kell vizsgálnunk, hogy ezek megfelelnek-e egyéb elvárásainknak is.

5.3.5. A DataScope főbb tulajdonságai és további lehetőségei

Felismerhető adatbázisok

A DataScope az adatokat a Microsoft Open Database Connectivity (ODBC) szabványa segítségével olvassa be az adatbázisokból. Ez a szabvány lehetővé teszi tetszőleges típusú adatbázis kezelését egy meghajtó segítségével. A DataScope-hoz mellékelt meghajtók segítségével beolvashatók adatok szöveges, DBase, Excel, Paradox, Btrieve, MS-Access, FoxPro file-okból, és SQL szerverek adatbázisaiból. További meghajtók a Microsoft-tól szerezhetők be.

Szinkronitás

A program legfontosabb jellemzője a teljes szinkronitás. Az előbbiekben láttuk, hogy miközben az adatbázis valamely elemét egy bizonyos szempontból vizsgáljuk, a program automatikusan megjeleníti ugyanazon elem más tulajdonságait is. Kiválaszthatunk meghatározott szempont szerint is bizonyos tulajdonságú

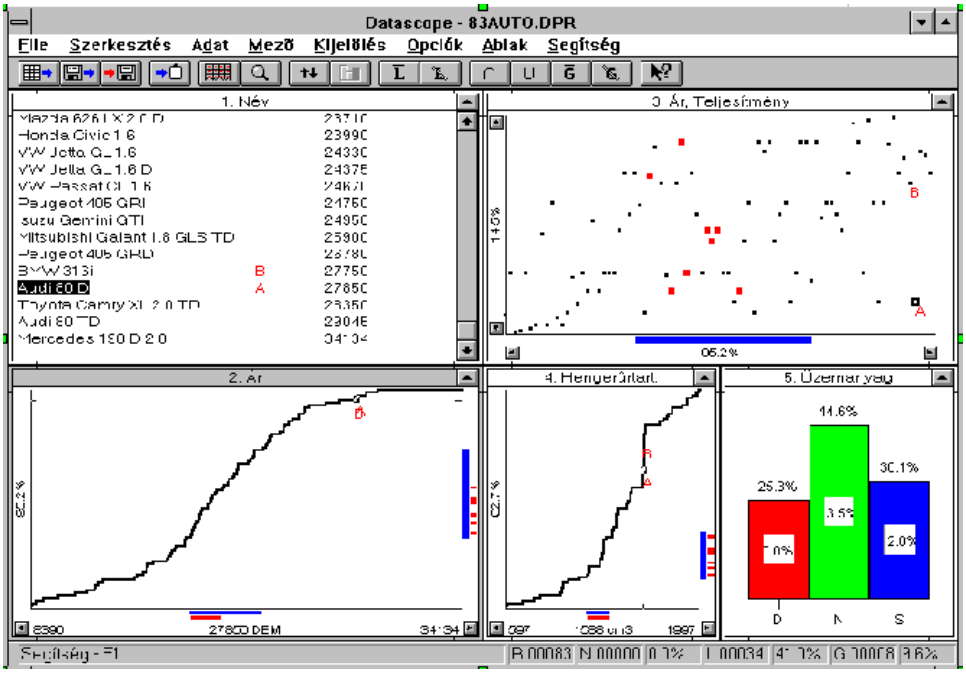
elemeket, és láthatjuk ezek egymáshoz való viszonyát más szempontok szerint. Egyidejűleg 16 ablakot nyithatunk meg, azaz egyszerre ennyi tulajdonság (vagy tulajdonságpár) szerint tanulmányozhatjuk az adatokat. Interaktivitás, grafikus lekérdezés Az adatbázis a megjelenő diagramokból interaktívan lekérdezhető. Ez szükségtelenné teszi parancsok begépelését. Egyszerűen csak ki kell jelölni egy pontot, vagy intervallumot az egérrel a kívánt információ megjelenítéséhez. A DataScope ezért tekinthető közvetlen vizuális lekérdező rendszernek is.

Kontextfüggő elemzés

A numerikus adatok elemzése sokkal hatékonyabban történik. Eddig nagyon sok időnkbe telt megállapítani egy rekord viszonyát a többiek között, az eloszlásfüggvény értéke azonban ezt azonnal mutatja. Például, ha azt hallottuk, hogy X autó Y litert fogyaszt, nem tudtuk, mennyire jó ez az érték, amíg nem néztünk át egy autókatalógust, vagy nem használtunk statisztikai módszereket, hogy képet kapjunk a jelenlegi helyzetről. Most, a DataScope-pal egyszerűen csak leolvassuk a százalékos értéket az Y tengelyről.

Interaktivitás, grafikus lekérdezés

Az adatbázist interaktív módon, közvetlenül az ábrázolt diagramokból kérdezhetjük le, így szükségtelenné válik szöveges parancsok kiadása. Egyszerűen csak rá kell mutatnunk a diagramok megfelelő pontjára vagy intervallumára, és azonnal megjelenik a kívánt információ (5.14. ábra). Így a DataScope on-line lekérdező rendszerként is jól alkalmazható.



5.14. ábra. DataScope: interaktív, grafikus lekérdezések.

Elemzés a DataScope-pal

A DataScope projekt fájlokkal dolgozik, amelyek tartalmazzák az adatbázis adatait, és minden beállítást, ami a munka későbbi folytatásához szükséges.

Bonyolultabb műveletek megvalósítása

Az Unió és a Metszet menüpontok mindig törlik az előző globális kijelölést, és újat hoznak létre az összes lokális kijelölésből. A DataScope egy speciális technikával lehetőséget biztosít bonyolult lekérdezések megvalósítására. Miután a lokális kijelölésekből létrehoztunk egy globális kijelölést, amelyen másféle műveletet is szeretnénk végezni, először rögzítenünk kell a globális kijelölést. Majd létrehozhatunk egy új diszkrét mezőt, amelynek két lehetséges értéke van: "Igen", ha a rekord beletartozik a globális kijelölésbe; "Nem", ha nem tartozik bele. Az új mezőnek nevet adva a későbbeik során ugyanúgy használható mint a diszkrét mező.

Numerikus mező létrehozása a globális kijelölésből

Lehetőségünk van arra is, hogy a globális kijelölésből egy új numerikus mezőt készítsünk valamelyik, már létező numerikus mező alapján. Ennek segítségével elérhetjük, hogy csak egy bizonyos feltételnek megfelelő rekordcsoport tulajdonságait vizsgáljuk

Új projekt létrehozása a globális kijelölésből

A globális kijelölésbe tartozó rekordokból új projektet hozhatunk létre a Kijelölés/Globálisból új projekt menüpont használatával. Az újonnan létrehozott projekt adatbázisa csak a kijelölt rekordokat fogja tartalmazni.

Kijelölések törlése, komplementálása

Egy adott mezőablakhoz tartozó lokális kijelölést megszüntethetünk a Kijelölés/Lokális törlése menüpont segítségével. Megtehetjük azt is, hogy a kijelöléseket ellenkezőjére fordítjuk (komplemens-képzés). Ezzel a kijelölt rekordokból jelöletlenek, az eddig nem jelöltekből pedig kijelöltek lesznek.

6. fejezet

Metaheurisztikák

6.1. Bevezetés

A kombinatorikus optimalizálás területén számos olyan probléma adódik, melyek nehezen megoldhatók, **NP**-teljesek (a feladat optimális megoldása legjobb esetben nem végezhető el polinomiális időben). Mivel az ilyen problémák megoldási ideje általában a feladat méretének exponenciális függvénye, ezért legtöbbször nem lehet hatékony, egzakt algoritmust alkalmazni. Az **NP**-teljes problémák esetében bevett gyakorlat olyan közelítő algoritmusok kidolgozása, melyek elfogadható időn belül produkálnak kielégítő megoldásokat, amik sok esetben nem optimális megoldások.

Az utóbbi években a nehéz optimalizációs problémák megoldására kifejlesztett közelítő algoritmusok a *metaheurisztikák* jutottak nagy szerephez. Az első metaheurisztikák az 1970-es években jelentek meg (pl. evolúciós algoritmusok), alkalmazásaik, illetve fejlesztéseikre irányuló kutatások a '90-es évektől egyre kiterjedtebbek lettek. A metaheurisztikus módszerek sikere főként annak köszönhető, hogy széles körben alkalmazhatók, rugalmasak, és számos problématípus esetében bizonyítottan kiemelkedő eredményeket produkálnak.

Az optimalizálás problémáinak megoldására léteznek „klasszikus" algoritmusok is, melyek sok esetben alapját képezték a metaheurisztikáknak. Az ilyen klasszikus algoritmusoknak alapvetően kétféle megközelítése létezik:

1. Konstruktív algoritmusok:
   E technika a közelítő megoldást inkrementális módszerrel állítja elő. Egy „üres" megoldásból kiindulva minden lépésben újabb elemet adunk hozzá a megoldásunkhoz, míg a teljes, megvalósítható megoldás fel nem épül. Egy tisztán konstruktív algoritmus e folyamat során nem használ visszalépést. A megoldás következő elemének meghatározásakor általában probléma-specifikus heurisztikákat szokás alkalmazni, és a választható elemeket a beillesztés hasznának becslése alapján szokás rangsorolni.

2. Lokális keresési algoritmusok:
   A lokális keresési megközelítés egy teljes megoldásból indul ki, és azt javítja lépésről lépésre. Az elnevezés onnan ered, hogy ezek az algoritmusok nem a teljes megoldásteret vizsgálják, hanem egy adott

megoldás jól definiált környezetében keresnek egyre jobb megoldásokat. Ezt a környezetet sokszor valamely egyszerűbb, a megoldást módosító operátor határozza meg. A lokális keresési algoritmus azokat a megoldásokat értékeli ki, amelyek az aktuális legjobb megoldásból a módosító operátor egyszeri alkalmazásával elérhetőek. Ha ezek között talál olyat, amely javítja az aktuális legjobb megoldást, akkor ennek a környezetében folytatja a keresést.

A gyakorlatban ritkán alkalmazzák a két megközelítés valamelyikét, ugyanis a két megközelítés általában jól kiegészíti egymást, így gyakran használnak hibrid algoritmusokat.

A legsikeresebb metaheurisztikák

1. Evolúciós algoritmusok (evolutionary algorithms, EA):
   Evolúciós algoritmusok közé soroljuk többek között a genetikus algoritmusokat, a genetikus programozást és az evolúciós stratégiát.

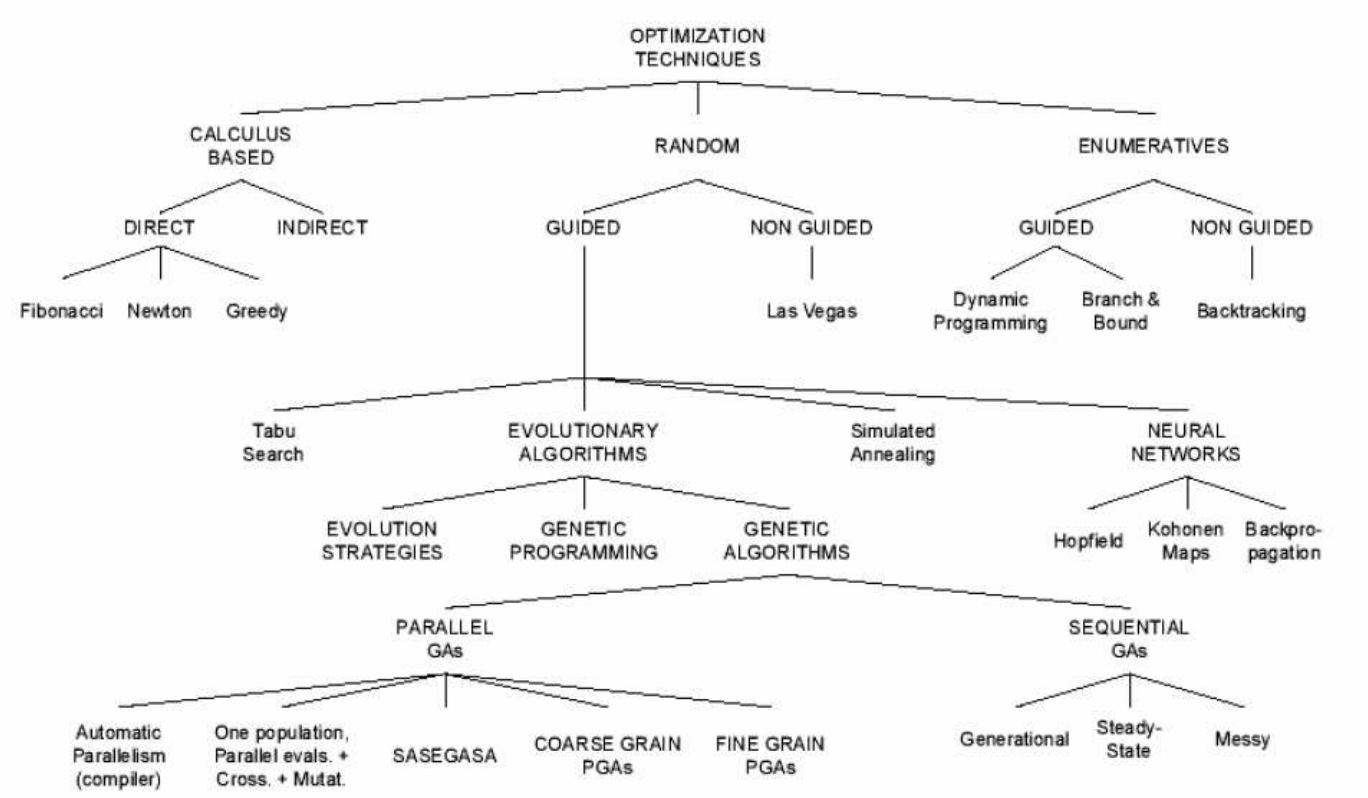
2. Szórt keresés (scatter search):
   A szórt keresés alapvető jellemzőit tekintve nagyban hasonlít az evolúciós algoritmusokra. Itt is a megoldások egy halmazával dolgozunk. A megoldások ezúttal vektorok. A megoldások több vektor lineáris kombinációjaként állnak elő. Az előállított vektort egy erre kialakított eljárás alakítja át a megoldástér részét képező megvalósítható megoldássá. A megoldások minőségének kiértékelésén alapuló selejtezési mechanizmusnak köszönhetően a megoldások halmazának mérete a szórt keresés esetében kisebb, mint az evolúciós algoritmusok esetén és nem változik.

3. Tabulistás keresési algoritmusok (taboo search):
   A tabulistás keresés alapötlete az, hogy a keresést a lokális optimum elérése után is mozgásban tartsuk, akkor is továbblépünk az aktuális legjobb megoldás szomszédságában található megoldás vizsgálatára, ha a kiválasztott megoldás rosszabb a jelenlegi legjobb megoldásunknál. Mivel a keresés könnyen ciklikussá válhatna tabu táblákkal jelöljük ki, hogy merre jártunk és hogy merre nem érdemes újra keresni.

4. Monte-Carlo algoritmus (Monte-Carlo algorithm):
   A Monte-Carlo algoritmus egy igen széles körben elterjedt algoritmus, mely a statisztikán alapszik. A neve is erre utal, ugyanis a módszer egy monte-carlo-i kaszinó ruletteredményeit is alapul vehetné. Véletlenszerű pontokban vizsgáljuk a feladatot és ezen véletlen pontokban kapott eredményekből vonunk le következtetéseket.

5. Hangyakolónia rendszerek (ant colony systems):
   A hangyakolónia koncepció a táplálékot kereső hangyák viselkedésének imitációján alapul.

A fenti felsorolás természetesen nem teljes, számos egyéb sikeres metaheurisztika létezik (pl. memetikus algoritmusok - memetic algorithms, mohó adaptív véletlen keresés - greedy randomized adaptive search, adaptív újraindítás - adaptive multisearch). A 6.1. ábrán látható optimalizálási technikák egy felosztása.



6.1. ábra. Optimalizálási technikák osztályozása.

A metaheurisztikák közös jellemzői

1. A keresés során megtalált megoldások legfőbb tulajdonságai valamilyen adatszerkezet segítségével memorizálásra kerülnek. Az evolúciós algoritmusok és a szórt keresés esetében a populációban megőrzött megoldások tárolják a keresés eddigi tapasztalatait; a tabulistás keresésnél maga a tabulista, a hangyakolónia rendszerekben pedig a feromon-mátrix szolgál memóriaként.

2. A memóriában felhalmozott információk felhasználásra kerülnek a további megoldások előállítása során.

3. A metaheurisztikus koncepciók önmagukban nem képesek jó minőségű közelítő megoldások előállítására, az ismert klasszikus módszerek irányított alkalmazásán alapulnak. Az eredményes implementációk általában több klasszikus módszert ötvöznek (hibrid algoritmusok).

A metaheurisztikus módszerek sikere több tényezőből adódik

- könnyű alkalmazhatóság
- a modellek bővíthetősége (a gyakorlati alkalmazás során felmerülő specifikus korlátozó feltételek a legtöbbször probléma nélkül beilleszthetők)
- a kapott megoldások jó minősége

6.2. Az evolúciós algoritmusok

A természetben lejátszódó evolúció igen összetett feladatokat oldott meg az idők folyamán. Ilyen feladatnak tekinthető például az emberi szem kifejlődése, melynek során többek között a szemet alkotó pálcikáknak, csapoknak és az azokat ellátó ereknek minél hatékonyabb konfigurációban, mennyiségben és arányban kellett kialakulniuk. Rengeteg szempont mellett született egy rendkívül eredményesen működő megoldás az emberi látás problémájára.

Az evolúció nem más, mint egy rendszer fokozatos (és véget nem érő) fejlődésének, átalakulásának a folyamata. Az evolúció a szelekciót, a keresztezést és a mutációt felhasználva egyedek egy adott generációjából egy újabbat generál úgy, hogy az utód generáció nem lesz tökéletes másolata az eredetinek. Emiatt időről időre felbukkan egy, vagy több utód, amely olyan génekkel rendelkezik, hogy az életben maradás tekintetében előnyt élvez más utódokkal szemben. Emiatt nagyobb eséllyel marad életben, és legtöbbször nagyobb eséllyel szaporodik, mint a többi utód. Így egy sikeres génállomány előbb-utóbb meghatározóbbá válik, mint egy kevésbé sikeres, és elterjed a populáción belül, míg a sikertelen háttérbe szorul.

A **szelekció** során a populációból kiválasztásra kerülnek az „erősebb” egyedek, melyek szaporodhatnak. Ez biztosítja, hogy csak a kellően erős, egészséges, életképes egyedek adhatják tovább a génjeiket. A **rekombináció** vagy keresztezés során a két szülő tulajdonságai keveredve átöröklődnek az utódokba. A **mutáció** során a szülő egyes génjei véletlenszerűen megváltozva kerülnek át az utódba.

Az egyes egyedeket a DNS kódolja, ez tartalmazza az egyed tulajdonságait. Az evolúció során a DNS-en mennek végbe a változások. Az evolúció során létrejövő egyedeknek két végső feladatuk, és hatékonysági mércéjük van, a túlélés és a reprodukció képessége.

6.2.1. Számítógépes megvalósítás

Az evolúciós algoritmusok (EA) alapötletét maga az evolúció adta. Meghatározásra kerül egy függvény, ami azt mutatja meg, hogy egy lehetséges megoldás mennyire jó megoldás. Ez a **fitnesz függvény** (fitness function). Minél jobb egy megoldás, annál jobb a fitnesz függvény-értéke.

Az algoritmus tervezése során egy adott problémára megkeressük, hogy egy lehetséges megoldást hogyan tudunk reprezentálni. A reprezentációnak olyannak kell lennie, hogy rajta el tudjuk végezni a rekombináció és a mutáció műveleteit, valamint lehetőség szerint minél gyorsabban megállapítható legyen az adott egyed fitnesz értéke.

Ezt követően legeneráljuk lehetséges megoldások egy halmazát a fenti reprezentációt használva. Ez lesz a kezdő populáció, a 0. generáció. A kezdő populáció generálása után kiértékelünk minden egyedet, azaz meghatározzuk a fitnesz értékeket. Ezek után kiválasztunk valamilyen stratégia alapján k elemet, és végrehajtjuk velük a rekombináció műveletét. Miután létrejöttek az új egyedek egy előre meghatározott stratégia szerint kialakítjuk a következő generációt, egy új populációt, mely tartalmazhat egyedeket az előző generációból is, de állhatnak kizárólag utódokból is. Az algoritmus tervezésekor szükség van egy megállási feltételre is, ezek közül az alábbiak a leggyakoriabbak:

- Előre meghatározott maximális generációszám elérése,
- Előre meghatározott fitnesz érték elérése,
- Előre beállított maximális futási idő elérése,
- A populáció változatosságának egy előre meghatározott szint alá süllyedése,
- A fitnesz érték változásának aránya egy előre meghatározott érték alá süllyedése.

Az algoritmus lépései sematikusan a következők:

Init lépés: Konstruáljunk egy véletlenszerű P_0 populációt, $i = 0$.

1. lépés: Értékeljük ki P_i-t! Ha a leállási feltétel teljesül, vége az algoritmusnak.

2. lépés: Hajtsuk végre P_i-n az alábbi lépéseket:

   a.) Szelekció

   b.) Rekombináció

   c.) Mutáció

   d.) Visszahelyezés, vagyis a P_{i+1}. generáció kialakítása

3. lépés: $i = i + 1$, és folytassuk a végrehajtást az 1. lépéssel, amíg a megállási feltétel nem teljesül.

Az evolúcióra épülő optimalizációs eljárások hatékonyak abban a tekintetben, hogy képesek megfelelő megoldást találni bonyolult optimalizációs problémákra is.

6.2.2. Az evolúciós algoritmusok típusai

Az evolúciós algoritmusok általános leírása nem határozza meg pontosan sem a reprezentációt, sem a műveleteket, sem az egyéb paramétereket. Így ezek megválasztásával igen sokféle evolúciós algoritmus alkotható meg. A leggyakrabban használt típusok a következők:

Evolúciós stratégiák (ES) A megoldásokat valós számokból álló vektorokkal ábrázolják, és gyakran adaptív mutációs rátát használnak (Rechenberg, Schwefel 1969).

Genetikus algoritmusok (GA) Ez a legnépszerűbb típusa az evolúciós algoritmusoknak. A megoldásokat bitekből, vagy egy véges ábécé elemeiből álló előre meghatározott hosszúságú sorozatok alkotják. Számos altípusa ismert (J. Holland 1962).

Genetikus programozás (GP) A megoldásokat számítógépes programok (tradicionálisan zárójelezett LISP kifejezések), illetve hasonló struktúrák alkotják. Fontos tulajdonság, hogy míg a korábban említett típusoknál a megoldások reprezentációit valamilyen algoritmussal ki kell értékelni, genetikus programozásnál az egyedek ezt az algoritmust magukban hordozzák (J. Koza 1994).

	ES	GA	GP
leggyakoribb reprerezentáció	valós értékű véges	bináris vagy ábécé feletti szavak	fa vagy programstruktúra
önadaptív	igen	nem	nem
szelekció	determinisztikus	sztochasztikus	sztochasztikus
problémák	folytonos optimalizálási problémák	diszkrét, kombinatorikus optimalizálási problémák	algoritmusok, függvények, logikai áramkörök opt.

6.1. táblázat. Az evolúciós algoritmusok típusainak összehasonlítása.

6.2.3. Műveletek

Az evolúciós algoritmusok során használt műveletek általában nagyban függnek a használt reprezentációtól. Az alábbiakban csak néhány példát adunk az evolúciós stratégiákban és a genetikus algoritmusokban használt operátorokra.

Rekombináció

Diszkrét egypontos keresztezés A két szülőt egy pontban két részre vágjuk, és a hátsó részeket megcserélve két új egyedet kapunk:

$$AB\mathbf{CDBA} \longrightarrow AB\mathbf{DCAD}$$
$$BB\mathbf{DCAD} \longrightarrow BB\mathbf{CDBA}$$

Diszkrét kétpontos keresztezés Hasonló az egypontos keresztezéshez, de itt csak a szekvencia egy közbülső részét cseréljük meg:

$$AB\mathbf{CDBA}BDA \longrightarrow AB\mathbf{DCAD}BDA$$
$$BB\mathbf{DCAD}AAB \longrightarrow BB\mathbf{CDBA}AAB$$

Valós értékű átlagolt rekombináció Az új egyedet a szülők (teljes, vagy részleges) átlagából kapjuk.

$$4,12;\ 5,90;\ \mathbf{6,98};\ \mathbf{1,73};\ 3,62 \longrightarrow 4,12;\ 5,90;\ \mathbf{4,15};\ \mathbf{2,48};\ 3,62$$
$$2,75;\ 5,27;\ \mathbf{1,32};\ \mathbf{3,23};\ 7,34 \longrightarrow 2,75;\ 5,27;\ \mathbf{4,15};\ \mathbf{2,48};\ 7,34$$

Mutáció

Diszkrét módosítás A megoldást reprezentáló vektor egy vagy több elemét lecseréljük:

$$ABE\mathbf{D}CABDE \longrightarrow ABE\mathbf{A}CABDE$$

Diszkrét csere A megoldást reprezentáló vektor két tetszőlegesen választott elemét megcseréljük:

$$AB\mathbf{E}DCA\mathbf{B}DE \longrightarrow AB\mathbf{B}DCA\mathbf{E}DE$$

Diszkrét áthelyezés A megoldást reprezentáló vektor egy elemét véletlenszerűen egy új pozícióba helyezzük:

$$ABED\mathbf{C}ABDE \longrightarrow AB\mathbf{C}EDABDE$$

Valós értékű additív A vektor egy eleméhez hozzáadunk egy (általában Gauss-eloszálsú) véletlenszámot:

$$4,12;\ 5,90;\ \mathbf{6,98};\ 1,73;\ 3,62 \underset{-0,12}{\longrightarrow} 4,12;\ 5,90;\ \mathbf{6,86};\ 1,73;\ 3,62$$

Valós értékű multiplikatív A vektor egy elemét megszorozzuk egy (általában Gauss-eloszlású) véletlenszámmal:

$$4,12;\ 5,90;\ 6,98;\ \mathbf{1,73};\ 3,62 \underset{\cdot 1,25}{\longrightarrow} 4,12;\ 5,90;\ 6,98;\ \mathbf{2,16};\ 3,62$$

Szelekció

A kiválasztást is a műveletek közé soroljuk, bár ez nem az egyedeken, hanem a populációkon működik. Éppen ezért legtöbbször független a reprezentációtól. Legismertebb fajáti a következők:

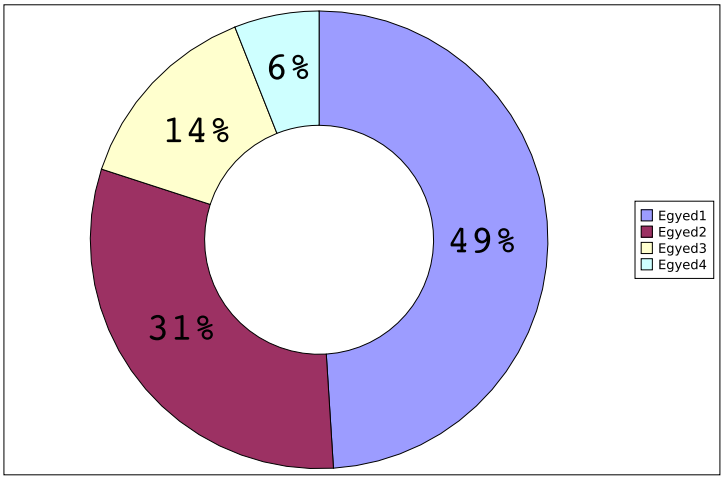
Legjobb egyed (best selection) Fitnesz érték alapján sorbarendezzük az egyedeket, és az első N darabot választjuk ki. Amennyiben $k > N$ szülőre van szükségünk, egy egyedet többször is kiválasztunk, mégpedig legalább $\lfloor k/N \rfloor$, legfeljebb $\lceil k/N \rceil$ alkalommal.

Rulettkerék (roulette wheel selection) Amikor a kiválasztás valószínűsége a fitnesz értékkel arányos, rulettkerék kiválasztásról beszélünk, ugyanis ez a kiválasztás pont úgy működik, mintha az egyedeket egy olyan rulettkerék cikkelyeihez rendelnénk, ahol a cikkelyek nagysága a fitnesz értékkel arányos (lásd 6.2. ábra).

$$P(e_i) = \frac{f(e_i)}{\sum_{e_i \in Pop} f(e_i)}$$

Párverseny (tournament selection) A kiválasztás úgy is történhet, hogy rendre kiválasztunk két egyedet, és a fitnesz értékük alapján versenyeztetjük őket, mégpedig úgy, hogy a nagyobb fitnesz értékű nagyobb eséllyel győz. Ez a módszer akkor is hasznos lehet, ha valami miatt nem tudunk fitneszt számolni, de két egyed jóságát össze tudjuk hasonlítani (például játékok nyerő stratégiáinak keresésekor a két stratégiát egymás ellen alkalmazva).

Rangsorolás (rank selection) Hasonló a rulettkerék algoritmushoz, de a kiválasztás valószínűségét nem a fitnesz érték határozza meg, hanem az egyed rangsorban elfoglalt helye.



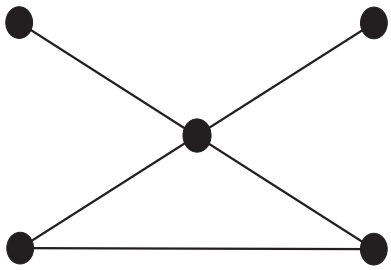
6.2. ábra. Rulettkerék szelekció 4 egyedre.

6.2.4. Példák

Számos probléma megoldható evolúciós algoritmusokkal. A következőkben ezekből mutatunk be négyet, és ezen keresztül szemléltetjük az evolúciós algoritmusok néhány lehetséges megvalósítását.

Gráfszínezés

Egy példa genetikus algoritmusra: gráfszínezés problémája. A gráfszínezés során adott egy G gráf, melyet optimálisan és jól kell kiszínezni, vagyis a lehető legkevesebb színt használva úgy, hogy a szomszédos csúcsok mást színt kapjanak. Például adott a 6.3. ábrán látható gráf.



6.3. ábra. Gráfszínezési probléma.

A csúcsokat megsorszámozzuk, a színeket pedig betűkkel (Pl. A - sárga, B - kék, C - zöld ,stb.) jelöljük. A színezés leírásához a csúcsok sorszámának megfelelő sorrendben írjuk le a színek betűkódjait. Így például az ABCDE számsor azt jelenti, hogy mind az 5 csúcs más színű, és a színeik sorban sárga, kék, zöld, stb. Nyilvánvalóan ez nem optimális színezést jelent. A $fitness$-érték pedig az azonos színű szomszédos csúcsok száma (pl. $fitness(ABCDEF) = 5$). Minél kevesebb azonos színű szomszédos csúcsot szeretnénk (lehetőleg egyet sem), tehát ez egy minimalizálási probléma. Operátorként a diszkrét kereszteződések és mutációk bármelyikét használhatjuk.

A szelekció művelet küszöbértékét önkényesen határozzuk meg, illetve a leállási feltétel attól függ, hogy mennyire "'jó"' megoldást szeretnénk. Természetesen a fenti gráf esetében a 0 $fitness$-érték elérése a cél, viszont nagyon nagy gráfok esetén elfogadható lehet ennél nagyobb (rosszabb) $fitness$-értékű megoldás is.

Egy ilyen kis gráf esetén a genetikus algoritmus elég gyorsan megtalálja az optimális megoldást (AACAB).

Nyolc királynő probléma

A nyolc királynő probléma esetén a feladat az, hogy egy sakktáblán 8 királynőt kell elhelyezni úgy, hogy ne üthessék egymást, azaz egy sorban és egy oszlopban pontosan egy királynő lehet. A nyolc királynő probléma reprezentációja például történhet úgy, hogy megadjuk az adott konfigurációban az egyes királynők melyik sorban és melyik oszlopban állnak. Például az alábbi táblázattal:

	sor	oszlop
1.	2	4
2.	5	6
3.	3	7
4.	5	4
5.	6	3
6.	3	5
7.	4	5
8.	1	3

Ezt akár ábrázolhatjuk egy 16 elemű vektorral is, amelyre a korábban említett diszkrét operátorokat alkalmazhatjuk, de magunk is definiálhatunk mutáció operátort, például egy királynő eltolása vízszintesen, függőlegesen vagy átlós irányban. A fitneszt az egymást ütő bábuk száma adja, az optimum a 0, tehát minimalizálási problémával állunk szemben.

Utazó ügynök probléma

Az utazóügynök probléma (TSP - Travelling Salesman Problem) során adottak városok, melyeket az ügynöknek be kell járni úgy, hogy az összes városban pontosan egyszer járjon, és az eljárás végén az ügynök visszatérjen a kiinduló városba. A cél egy olyan útvonal meghatározása, melynek a költségei a legkisebbek. Költség leggyakrabban az útvonal hossza.

Az egyedreprezentáció a TSP esetén egy vektor lesz, melyben a városok sorrendje szerepel úgy, hogy a városokat számokkal reprezentáljuk. A *fitness*-érték az utazás összköltsége. A TSP problémának az inputját szokás súlyozott gráffal megadni. Azonban probléma lehet a műveletek megvalósításánál abban a tekintetben, hogy mivel az egyed tulajdonképpen egy körút a gráfban, a mutációnál figyelni kell arra, hogy a módosítani kívánt csúcs ne jelentsen „szakadást” a körútban (vezessen bele él az őt megelőző csúcsból, illetve vezessen belőle él az őt követő csúcsba). A keresztezés szintén bonyolult egy ilyen esetben, hiszen ha két egyeden egypontos keresztezést hajtunk végre, akkor előfordulhatnak szakadások, illetve hogy csúcsok többször szerepelnek majd az út során. Megoldást jelenthet a problémára, hogy vagy csak „jó” változtatást engedélyezünk az egyedeken, vagy pedig a *fitness* függvényt kibővítve nagyon nagy büntetést adunk a nem megfelelő egyedeknek (szelekció így biztosan kiszórja őket).

Hátizsák probléma

A hátizsák probléma szintén megoldható evolúciós algoritmusokkal. A probléma során adott egy hátizsák, melynek van egy adott kapacitása (C), illetve van n darab tárgy, adott térfogattal (w_i), és az értékkel (p_i). A cél, hogy minél nagyobb értékben helyezzünk el tárgyakat a hátizsákban. A táskában elhelyezett tárgyak súlya:

$$C \geq w = \sum_{i=1}^{n} w_i x_i, \tag{6.1}$$

ahol $x_i \in \{0, 1\}$ jelentése, hogy az i-edik elem benne van-e a hátizsákban. A feladat célfüggvénye, amit jelen esetben maximalizálni szeretnénk a következő:

$$E = \sum_{i=1}^{n} p_i x_i, \tag{6.2}$$

ahol a p_i az i-edik elem értéke. A probléma átfogalmazható a következőképpen: határozzuk meg a tárgyak halmazának azon részhalmazát, melyre E maximális!

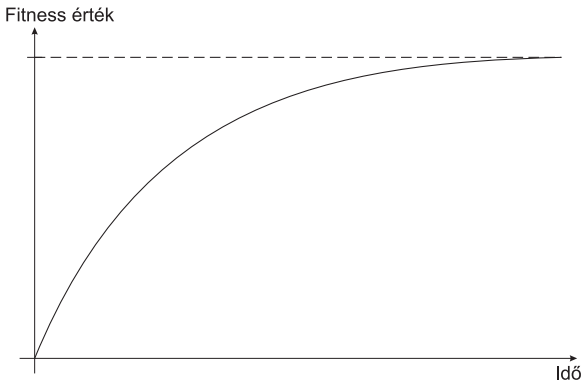
A algoritmus definiálásához rögzítsük a tárgyak sorrendjét. Ekkor a probléma egy lehetséges megoldása egy n hosszú bináris vektorral reprezentálható, melynek az i-dik elemének értéke 1, ha az i. tárgy ki lett választva, és 0, ha nem. Tehát egy egyed alakja:

1	0	1	1	$\cdots$	1

Ennél a feladatnál szintén a korábban felsorolt diszkrét keresztezést és mutációt használhatjuk. A maximális kapacitás betartását büntetőpontok levonásával kényszeríthetjük ki.

6.2.5. Az evolúciós algoritmusok tulajdonságai

Az evolúciós algoritmusok hatékonyságát úgy szokták szemléltetni, hogy az egyes populációkhoz hozzárendelik az adott populációban fellelhető egyedek fitneszértékének átlagát és szélsőértékeit, és ezt grafikonon ábrázolják. Egy maximalizálási probléma esetén leggyakrabban a 6.4. ábrán látható görbéhez hasonló grafikont kapunk.



6.4. ábra. Evolúciós algoritmusok hatékonyságának szemléltetése.

Az evolúciós algoritmusok előnyei:

- Általában egyszerűen kódolható.

- Igen széles körben használható (nagy keresési térben is alkalmazható, kevés információt igényel a célfüggvényről).

- Zajos fitneszfüggvény esetén is működik.

A felsoroltakon kívük fontos megemlíteni az alábbi (elméleti jelentőségű) tételt:
Tétel: A genetikus algoritmusok globális optimumhoz konvergálnak.
(*Ennek bizonyítását lásd: Futó Iván: Mesterséges intelligencia.*)
Gyakorlatban az evolúciós algoritmusoknak bizonyos hátrányaik is vannak:

- Lokális optimumot ad csak (véges lépésben).

- Vannak olyan problémák, melyek esetén nehézkes az egyedreprezentáció, illetve a műveletek megvalósítása.

- Az algoritmus futási ideje nagy.

- A peremfeltételeket nehéz kódolni.

A teljesség kedvéért megjegyezzük, hogy az evolúciós algoritmusok irodalma elég széleskörű, sok gyakorlati problémára létezik már megoldás, és az algoritmusok hatékonyságának növelése érdekében is számos technikát fejlesztettek ki. Ilyenek például a következők: Az eljárás hatékonyságának növelése érdekében:

1. Elitizmus bevezetése, azaz a legjobb egyedeket mindig megőrizzük.

2. Egy mutáció műveletet csak akkor végrehajtani, ha előnyös a mutáció.

3. Több populáció segítségével a keresési tér jobban lefedhető. Ha az egyes populációk találkozását is implementáljuk, akkor elkerülhető, hogy az algoritmus egy lokális optimumnál beragadjon.

4. Sok esetben léteznek járulékos információk, amelyet a kiválasztásnál vagy az operátorok alkalmazásánál felhasználhatunk.

6.3. Hegymászó algoritmus - hill climbing

A hegymászó stratégia egy igen egyszerű meggondoláson alapszik, miszerint ha egy diszkrét értelmezési tartományú függvény maximumát keressük, akkor ezt a legkönnyebben úgy tehetjük, hogy kiindulunk egy pontból és megvizsgáljuk, hogy a pont szomszédjai közül van-e olyan, aminek függvényértéke nagyobb mint az aktuális ponté. Ha találunk ilyen pontot, akkor válasszuk azt aktuális pontnak és ismételjük meg a vizsgálatot csakúgy, mint egy hegymászó, aki nem látja a csúcsot, és mindig fölfelé törekszik. (Egyfajta mohó stratégiát megvalósítva.)

A hegymászó algoritmus számos probléma optimumát jól közelíti, és egyszerű a kódolása is. Az algoritmus megállási feltétele az, hogy egy adott pont környezetében nem találunk olyan pontot, melynek függvényértéke nagyobb lenne, mint az aktuális ponté.

A hegymászó algoritmus nagy hátránya, hogy gyakran lokális optimumba ragad.

Példa a hegymászó algoritmus működésére: adott egy 50×50-es négyzetháló, melyben számok vannak elhelyezve. A cél az, a számokon haladva megtaláljuk azt a 20 hosszúságú utat, mely út mentén a számok összeges maximális, és amely szomszédos számokat köt össze (a számok között csak jobbról balra, és fentről lefelé lehet haladni).

Véletlenszerűen kiválasztunk egy pontot, és megvizsgáljuk például a 3×3-as környezetét, majd innen próbálunk magasabb értékre lépni. Ha sikerült magasabb értékre lépni, akkor a kapott magasabb értéket kiinduló pontnak tekintve vizsgáljuk a környezetet. Ehhez természetesen szükség van egy függvényre, mely megmondja egy megoldás „jóságát".

A feladat megoldható azonban egyszerűbben is: a számokat diszkretizáljuk nagyságuk szerint, és ennek megfelelően kiszínezzük a mezőket (így a számokat akár ki is törölhetjük). Ezek után a szemünkkel pontosan látjuk, hogy hol vannak a hegygerincek, és ez alapján könnyen meghatározhatjuk az optimumot. Tehát a szemünk jobb, mint számos heurisztikus algoritmus, mert a szem párhuzamosan képes feldolgozni a különböző információkat.

6.4. Szimulált hűtés - simulated annealing

A fent megismert hegymászó stratégia bár egyszerűen adoptálható számos probléma esetében, és széles körben használják, mégis javításra szorul, hiszen gyakran ragad lokális optimumba. Ennek a problémának a feloldására javasolt Kirkpatrick, Gelatt és Vecchi 1983-ban egy módszert, mely tulajdonképpen a hegymászó stratégia egy javítása. Az általuk javasolt Szimulált Hűtés (Simulated Annealing) egy véletlenszerűen választott megoldásból kiindulva keres annak környezetében egy új megoldást, viszont bizonyos valószínűségi feltételek teljesülése esetén megengedi a megoldás romlását is. Ezért ki tud lépni a lokális optimumból, melyet egy gradiens alapú vagy egy hegymászó típusú algoritmus nem képes.

A szimulált hűtés is alapvetően egy természetbeli (ipar eljárások során használt) termodinamikai folyamaton alapszik. Magas hőmérsékleten a folyadék molekulái egymáshoz viszonyítva szabadon mozognak. Ha a folyékony halmazállapotú anyagot lassan hűtjük, a hőenergiája, vagyis a molekulák mozgási energiája csökken. Ekkor az atomok megpróbálnak elrendeződni és tiszta kristályrácsot kialakítani, ami a rendszer legalacsonyabb energiaszintű szerkezetét jelenti. A fokozatos hűtés során a termikus zajnak köszönhetően néha növekszik a folyadék energiája, azonban kellően lassan hűtött rendszereknél a természet képes megtalálni a globálisan minimális energiaszintű állapotot. Ha a folyékony fémet gyorsan hűtjük vagy megedzük, az nem éri el ezt a szerkezetet, hanem polikristály vagy amorf állapotba kerül valamivel magasabb energiaszinten. Ezen a természeti jelenségen alapul a Szimulált Hűtés módszere is.

Az iteratívan javuló típusú eljárásoknál véletlen pontok sorozatát generáljuk, amíg a célfüggvényben

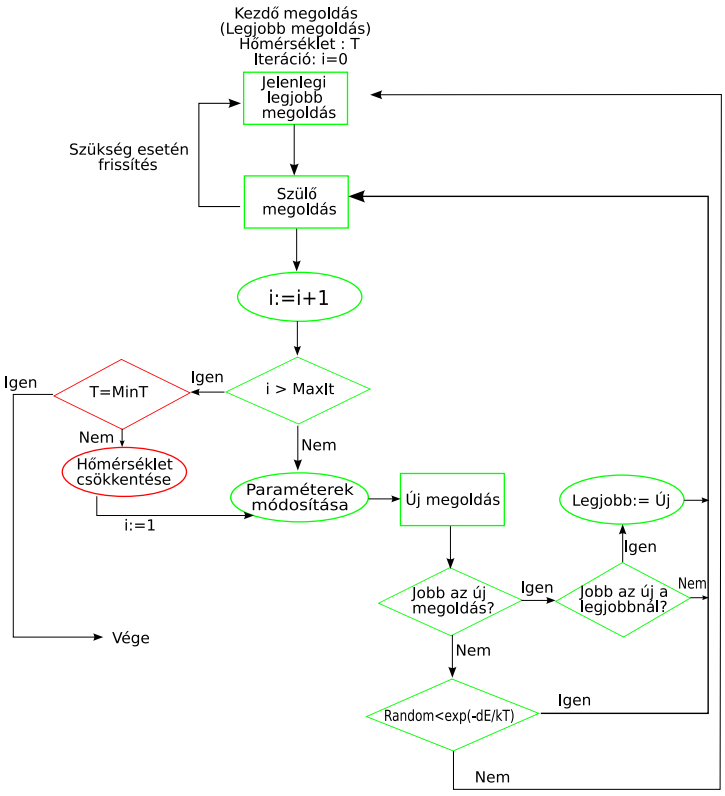
javulást figyelhetünk meg. Ebben az esetben a véletlen pontot elfogadjuk. Mivel ez az eljárás csak lefelé menetet enged meg a tartományon belül, így az optimalizálás könnyen megakadhat egy lokális optimumban. Ennek elkerülése érdekében a hűtési eljárás mellé egy másodlagos kritériumot kell adnunk a módszerhez. E szerint, ha a véletlen pont nagyobb célfüggvényértéket eredményez, mint az eddigi legjobb eredmény, akkor ennek a véletlen pontnak az elfogadási valószínűsége:

$$P = e^{-\frac{\Delta E}{T}}$$

Itt a ΔE az aktuális és a vizsgált megoldás energiájának különbsége, T pedig az aktuális hőmérséklet. Látható, hogy az elfogadási valószínűség függ attól, hogy mennyivel rosszabb a vizsgált megoldás az aktuálisnál, illetve, hogy mennyire magas a hőmérséklet (nagyobb hőmérsékleten nagyobb valószínűséggel fogadunk el rossz megoldásokat). Ekkor választunk egy véletlen számot a $[0, 1)$ intervallumból, és ezt a valószínűséget összehasonlítjuk vele.

$$P \geq random[0, 1)$$

Ha a feltétel teljesül, akkor a véletlen pontot elfogadjuk, ellenkező esetben pedig elutasjtjuk. Ez a véletlen számoktól való függőség teszi a szimulált hűtést *sztochasztikus* eljárássá.



6.5. ábra. A SA folyamatábrája.

Ennek a javítási iterációnak a többszöri végrehajtása minden adott T kontroll paraméterértékre, tulajdonképpen a fématomok hőtermikus újrarendeződésének a szimulációja az adott hőmérsékleten. Az adott

hőmérsékleten végzett iterációk képezik az eljárás belső ciklusát, a külső ciklus pedig a T kontrollparaméter (hőmérséklet) csökkentése. A T kezdő értékének megfelelően magasnak kell lennie, amit a meghatározott hűtési ütemezés szerint csökkentünk. Az eljárás folyamatábrája a 6.5. ábrán látható.

A SA stratégia alkalmazásához minden optimalizálási probléma esetén szükséges definiálni a következő 4 fő elemet:

1. *Probléma konfiguráció/Solution space.* A keresési tartomány megadása, amely felett az optimumot keressük.

2. *Környezet konfiguráció/Transiotion.* Új véletlen pont generálásának módja.

3. *Célfüggvény megadása/Fitness function.* Egy valós értékű függvény, amely méri minden lehetséges megoldás súlyát, így minden lehetséges megoldáshoz szolgáltat egy értéket attól függően, hogy az a pont mennyire jó.

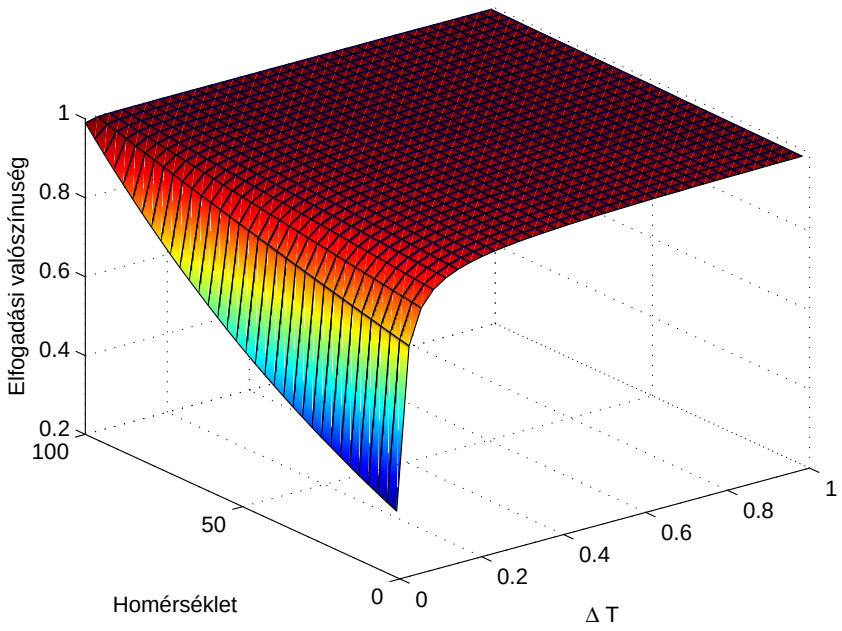
4. *Hűtési ütemterv/Annealing Schedule.* A belső ciklusban végrehajtandó iterációk száma, illetve a kontroll paraméter külső ciklusbeli csökkentésére használt módszer meghatározása.

Paraméterek: T_0 - kezdő hőmérséklet , $g()$ - hűtési függvény,
S - a megoldások halmaza, $N(x)$ - az x szomszédja
hűtési_feltétel:maximális iterációszám elérése
megállási_feltétel

procedure simulated_annealing
begin
$t := 0$;
inicializálás $T = T_0$;
random inicializálás $x \in S$;
repeat
repeat kiválaszt $y \in N(x)$;
if $f(y) < f(x)$ **then** $x := y$
else if $random[0,1) \leq e^{\left[\frac{f(x)-f(y)}{T}\right]}$
then $x := y$;
until (hűtési_feltétel);
$T := g(T,t)$;
$t := t + 1$;
until (megállási_feltétel);
end;

Az algoritmus egyik előnye, hogy egyszerűen megvalósítható. Továbbá az SA algoritmus megfelelő paraméterezéssel nem ragad lokális optimumba. Pontosabban, ha minden lehetséges megoldás elérhető a kiindulási állapotból (az egyes állapotok környezetkonfiguráció segítségével), akkor a globális optimumot el is érjük elegendő sok iteráció után. A legegyszerűbb hűtési ütemterv a lineáris: $(T_{t+1} = \alpha T_t, \alpha \in (0,1))$. Azonban a hűtési ütemezésre sűrűn alkalmaznak exponenciális és szigmoid hűtési függvényeket. A 6.6. ábrán az SA elfogadási valószínűsége van ábrázolva a hőmérséklet és a hőmérséklet-változás $(\Delta T = f(x) - f(y))$ függvényében. Jól látható, hogy minél rosszabb állapotba szeretnénk átlépni, annál kisebb valószínűséggel fogadjuk el az átmenetet. Illetve minél alacsonyabb a hőmérséklet, annál kevésbé fogadunk el egy rosszabb állapotot.

A szimulált hűtés sokváltozós optimalizálási problémákat is képes kezelni, illetve akkor is működik, ha sok lokális optimuma van az optimalizálandó függvénynek. Azonban nem mindig triviális az algoritmus paramétereinek beállítása (nincs egyértelmű szabály, hogy milyen feladattípusnál milyen paramétereket érdemes használni, ez általában tapasztalati úton kell meghatározni), illetve a futási idő nagy.



6.6. ábra. Az SA elfogadási valószínűsége.

Az algoritmus hatékonyságát javítani lehet például azzal, ha többször egymás után lefuttatjuk, akár véletlenszerűen választott pontból kiindulva, akár az előző végrehajtások során megtalált optimumból kiindulva. Számos területen használják a szimulált hűtést, például kombinatorikus optimalizálási problémák megoldásánál, vagy integrált áramkörök tervezésénél, illetve az orvostudományban tomográfiával készült felvételek feldolgozásánál.

6.5. Hangya kolónia algoritmus - ant colony system

A hangya algoritmus szintén a természetből vette az alapötletét. Mint már a bevezetőben említettük, a hangyakolónia koncepció a táplálékot kereső hangyák viselkedésén alapul. A hangyák igen kifinomult módszerekkel képesek az egymás közti kommunikációra, ugyanis a bolytól a különböző táplálékforrásokhoz vezető útvonalakat a bolyhoz visszavezető úton feromon hormon kibocsátásával jelölik meg. A feromonösvényeket a többi hangya érzékeli, és nagy valószínűséggel követni kezdi. Az élelmek felé vezető utak nagyon különbözőek lehetnek, akadályok is előfordulhatnak közöttük. A hangyák célja természetesen minél több élelem begyűjtése. Az egyes egyedek adottságai korlátozottak, viszont a kolónia egysége igen hatékonyan oldja meg ezt a problémát.

Az egyes hangyaegyedek elszigetelten gyakorlatilag véletlenszerűen mozognak, viszont felismerve a feromonnal megjelölt ösvényeket nagy valószínűséggel követni is kezdik azt. Mindeközben saját feromonkibocsátásukkal növelik az útvonal feromon koncentrációját, s ezáltal annak vonzerejét. Így a gyakran használt útvonalak feromon szintje egyre erősödik, míg az elhanyagolt ösvényeké csökken.

A hangyák által kibocsátott feromon folyamatosan párolog, azaz az adott útvonalon megerősítés (újabb feromon adag kibocsátása) nélkül folyamatosan csökken a feromon szintje (amíg tartalmaz feromont). Abban az esetben, ha két út áll rendelkezésre ami a táplálékhoz vezet, a rövidebb útvonalon gyakrabban képesek fordulni a hangyák (mivel hamarabb érnek oda és vissza). Így gyakrabban megerősítik a feromon szintet, ezáltal magasabb „vonzerőt” biztosítva az útvonal számára. Ám ez további hangyákat csábít át a rövidebb útra. Egy idő után a rövidebb úton fog a legtöbb hangya közlekedni, míg a hosszabb út feromon szintje minimális szintre csökken.

A hangyakolónia hatékonyságát növelni lehet azáltal, hogy ha a hangyák az élelemtől visszafelé vezető úton erősebb feromonokat hagynak maguk után abban az esetben, ha jobb élelmiszerforrást találtak. Egyes hangyafajokra jellemző ez a viselkedés.

Az optimalizációs problémák megoldására alkalmazott hangyakolónia metaheurisztika „hangyának” nevezett ágensek között osztja meg a keresési feladatokat. Ezek az ágensek nagyon egyszerű alapadottságokkal rendelkeznek, és bizonyos fokig a valódi hangyák viselkedését szimulálják. A hangyakolónia rendszerekben a „mesterséges hangyák” (ágensek) megfelelő probléma-specifikus konstruktív heurisztika alapján építenek fel megoldásokat. A megoldás építőelemeinek a jó megoldások felépítése során alkalmazott sorrendjét egy feromon-mátrix segítségével határozzuk meg. A feromon-mátrixban tárolt értékeket a többi ágens valószínűségi alapon figyelembe veszi a saját megoldásainak konstruálásakor. A mesterséges hangyák úgy kutatják fel a feladat megoldásterét, mint a valódi hangyák a környezetüket; a talált élelmiszerforrások minősége az előállított megoldások célfüggvény-értékének felel meg: a feromon-ösvények tekinthetők adaptív memóriának. A megoldás felépítése során a mesterséges hangyák a probléma-specifikus memórián túlmenően saját memóriával is rendelkeznek, ahol tárolják a felépített megoldás minden fontos tulajdonságát.

Az első hangyakolónia rendszert az utazóügynök probléma (TSP - travelling salesman problem) megoldására alkalmazták. Legyen a bejárandó városok száma n. A városok közti távolságokat egy $n \times n$-es D mátrixban tároljuk, melynek egy d_{ij}eleme az i-dik város j-diktől vett távolságát tartalmazza. Szükségünk

lesz egy F feromon-mátrixra is, amelynek f_{ij} eleme az i-dik és j-dik város közti út feromon-szintjét tárolja. Az F mátrixot egy kiinduló (alacsony) értékkel inicializáljuk. Virtuális hangyáink minden iterációs lépésben kiválsztják a következő meglátogatandó várost. A döntéshozatal előtt minden városhoz kiszámítunk egy valószínűséget, ami az aktuális város és a vizsgált város közti út hosszának és feromon-szintjének függvénye. A már látogatott városok újbóli választását természetesen nem engedjük meg, az ezekhez tartozó valószínűségeket zérusra állítjuk. A látogatható városokhoz tartozó valószínűséget számolhatjuk például a következő képlettel:

$$p_{ij} = \frac{f_{ij}^{\alpha} \frac{1}{d_{ij}}^{\beta}}{\sum_{k \in L} \left[f_{ik}^{\alpha} \frac{1}{d_{ik}}^{\beta} \right]}$$

Itt az i az aktuális város indexe, j egy választható város indexe, L a választható városok halmaza, α és β paraméterek, melyek segítségével beállíthatjuk a heurisztikus információk és a keresési tapasztalatokból származó információk egymáshoz viszonyított jelentőségét.

A hangya az így kapott valószínűségeken alapuló véletlen választással hozza meg a döntését a körútba következőként felvett várost illetően:

- Feromon-párolgás: a feromon-mátrix minden elemét beszorozzuk egy 0 és 1 közötti párolgási együtthatóval. A feromon párolgása megakadályozza a korai konvergenciát.

- Feromon-frissítés az előállított megoldás alapján: az előállított körút szomszédos városai közti feromon-szintet megnöveljük a körút teljes hosszának figyelembevételével:minél rövidebb körutat találunk, annál inkább megnöveljük a bejárt útvonal szakaszaihoz tartozó feromon-szintet. A fenti folyamat memorizálja a keresés tapasztalatait.

- Megoldás kiértékelése: ha az előállított körút jobb az eddigi legjobb megoldásnál, akkor eltároljuk.

A hangyakolónia-rendszerek általában kolóniákba szervezik a hangyákat, azaz egy keresési lépésben több hangya állítja elő megoldásait az aktuális feromon-értékek alapján. a gyakorlati tapasztalatok szerint ez nagyban növeli az algoritmus stabilitását. Az újabb és újabb lépéseket vagy előre meghatározott lépésszám eléréséig folytatjuk, vagy akkor állunk le, ha hosszabb ideig nem javít a legjobb megoldáson az algoritmus.

A következő szempontokat kell figyelembe venni egy tetszőleges problémára felépítendő hangyakolónia-rendszer implementálása esetén:

a.) Feromon definíció:

Az adaptív memóriát megvalósító adatszerkezet kiválasztása döntően befolyásolja implementációnk sikerességét. Az utazóügynök probléma megoldására bemutatott példában az f_{ij} feromon-szint megközelítőleg azt fejezi ki, hogy a teljes körút minősége szempontjából mennyire előnyös az i város után közvetlenül a j várost választjuk. A városok körúton belüli relatív pozíciójának beépítése az adaptív memóriába a későbbiekben valóban lényegi információt szolgáltat. Az f_{ij} egy másik értelmezése lehet,

hogy mennyire előnyös az i várost a j-dik állomásként meglátogatni. Mivel körutakkal dolgozunk, a $\pi = (1, 2, \cdots, n)$ megoldási permutáció megegyezik a $\pi' = (n, 1, 2, \cdots, n-1)$ megoldással, tehát a városok körúton belüli abszolút pozíciója teljességgel indifferens; az erre alapozott adaptív memória nem használható fel a keresés folyamán. Ugyanakkor más (elsősorban ütemezési) problémákban az abszolút pozíció memorizálása lehet a jó választás.

b.) Egyensúly a konzervatív és felfedező keresés között:

A konzervatív és felfedező keresés közti megfelelő egyensúly biztosítása minden metaheurisztikus módszernek alapvető problémája. A konzervatív szemlélet a keresés folyamán összegyűjtött tapasztalatokat használja ki, míg a felfedező hozzáállás a megoldástér mindeddig ismeretlen részei felé orientálja a keresést. Bármelyik szemlélet kizárólagos alkalmazása problémákat eredményez: a konzervatív keresés nagyon gyorsan bekerül a lokális optimum csapdájába, míg a felfedezés túlzott erőltetése megbízhatatlan keresési folyamatot ad (esetleg a megoldástér ígéretes részeit nem vizsgálja kellő alapossággal). Az ismertetett példában az egyensúlyt a mesterséges hangya-konstrukció során alkalmazott döntési mechanizmusa biztosítja: a feromon-szintek figyelembevételével konzervatív irányba visz, míg a valószínűségi választás a felfedező irányba.

c.) Lokális keresés alkalmazása:

Az ismertetett implementáció valószínűleg igen gyengén teljesítene az utazóügynök probléma megoldása során. A hangyakolónia-rendszerek általában felhasználnak valamilyen probléma specifikus lokális keresési módszert, ami a hangyák által konstruált megoldást eljuttatja a legközelebbi lokális optimumig. A feromonfrissítést pedig már a lokális kereséssel feljavított megoldás alapján végzik el.

d.) Heurisztikus információk felhasználása:

A probléma-specifikus heurisztikák felhasználása a megoldás felépítése során nagyban javítja a hangyakolónia-rendszer teljesítményét. Az ismertetett példákban heurisztikus információként a körútba következőként beilleszthető jelöltek közelsége szolgált, melyeket rendkívül egyszerűen a távolság reciprokaként számítottunk.

e.) A hangyakolónia mérete:

A gyakorlati tapasztalatok azt mutatják, hogy a több hangyából álló kolóniák jobban teljesítenek, mint egyetlen hangya. A hangyák száma erősen függ a probléma jellegétől, vagy akár a feladat egy-egy adott tulajdonságától. A megfelelő értéket általában a fejlesztés utolsó fázisában állítják elő, a tesztfuttatások alapján.

f.) Jelöltlisták alkalmazása:

A mesterséges hangyák konstruktív algoritmusának fejlesztése során találkozhatunk azzal a problémával, hogy a megoldás építésének egy korai állapotában túlságosan nagy számú továbblépési lehetőség közül kell választanunk. Ez egyrészt nyilvánvalóan jelentősen megnövelheti az algoritmus időigényét, másrészről túlságosan szétszórhatja nagyméretű megoldástérben a hangyákat, amik így nem tudnak

jól együttműködni. Ez a probléma felmerülhet például nagy méretű utazóügynök problémánál. Ilyen helyzetben érdemes a feladat előzetes elemzéséből származó információk alapján csökkenteni az adott állapotból való továbblépési lehetőségek számát. Az utazóügynök probléma esetében például minden városhoz kigyűjthetjük annak legközelebbi szomszédját, és ezek fogják alkotni a városhoz tartozó jelöltlistát. A következő állomás kiválasztásakor pedig csak a jelöltlistát vizsgáljuk, és ha lehet, onnan választunk. Egy adott város tágabb környezetét csak akkor vizsgáljuk ha a jelöltlista minden városa már szerepelt az útvonalon.

Korábban láttuk, hogy már a korai hangyakolónia implementációk túlléptek a valódi hangyák viselkedésének szimulációján. A koncepció fejlődése során megfigyelhető, hogy az újabb és sikeresebb alkalmazások egyre távolabb kerülnek a kiindulási analógiától. Tekintsünk néhány továbbfejlesztést!

1. Feromon-frissítés teljes megoldások alapján: a korai hangyakolónia-rendszerek a konstruktív algoritmus minden lépése után kiértékelték a csonka megoldást, és annak minősége alapján számították a megtett lépéshez kapcsolódó feromon-értéket. Mivel a csonka megoldások értékelése mindig „rövidlátó", az implementációk gyorsan áttértek a teljes megoldás alapján való, eredményesebb utólagos feromon-frissítésre.

2. Elitista stratégia: a kiindulási elgondolás szerint a feromon-szinteket minden előállított megoldás alapján növelni kell, az előállított megoldás minőségével arányosan. Ez a módszer sok helyen tartotta magasan a feromon-értékeket, és ezáltal nem eredményezett megfelelő megoldáshoz való konvergenciát. Az elitista stratégia szerint minden iteráció után (egy iteráció során a hangyakolónia minden tagja konstruál egy új megoldást) a legjobb megoldás alapján növeljük a feromon-szinteket. A legjobb megoldás jelentheti az adott iteráció legjobb megoldását (gyenge elitizmus), vagy a keresés eddigi legjobb megoldását (erős elitizmus). Az elitista stratégia általánosításában nem egyetlen megoldás mentén növeljük a feromon-szinteket, hanem több jó megoldás alapján.

3. Pszeudo-random választási szabály: a bemutatott rendszerben a hangyák a heurisztikus információ és a feromon-szintek alapján választották ki a valószínűségi alapon a konstrukció következő lépését. Mivel így, a tapasztalatok szerint, túl nagy szerepet kapott a véletlen, ezért bevezetésre került a pszeudo-random választási szabály, amely során egy új paraméter került bevezetésre. A paraméter által kifejezett valószínűséggel a hangya a heurisztikus információ és a feromon-szintek alapján a legelőnyösebbnek tűnő lépést fogja választani. Az új szabály bevezetése jelentősen növelte a közelítő algoritmus robosztusságát.

4. Feromon-párolgás helyett feromon-koptatás: a feromon párolgása minden iteráció után egységesen csökkenti az összes feromon-szintet. A feromom „koptatása" csak az adott iterációban létrehozott megoldások mentén csökkenti a feromon-szinteket (lokális feromon-frissítés). Ez a technika a megoldástér eddig fel nem derített részei felé orientálja a keresést, ezáltal fenntartva a keresés felfedező jellegét.

5. A feromon-szintek korlátozása: néhány implementáció a feromon-szintek kétoldali bekorlátozásával gátolja a túl korai konvergenciát. A feromon-szintek maximumát és minimumát meghatározó korlátok dinamikusan is változhatnak; a keresés korai szakaszában a felfedező jellegre helyezve a hangsúlyt, majd a keresés későbbi szakaszában a konzervatív keresés irányába eltolva azt.

6. Diverzifikációs keresési fázis a feromon-szintek újrainicializálásával: amennyiben érzékeljük, hogy a keresés stagnáló fázisba jutott, azaz hosszú ideje ugyanazt a megoldást állítják elő a hangyák, akkor megpróbálhatunk elmozdulni a holtpontból azáltal, hogy a feromon-szinteket egységesen visszaállítjuk a kezdeti érékekre, ezzel kényszerítve a hangyákat a felfedezést eredményező viselkedésre.

A hangyakolónia koncepció fejlődése egyre sikeresebb implementációkat eredményezett. Ennek hatására a kombinatorikus optimalizálás számos területén próbálkoztak meg a szakemberek az új módszer bevetésével. A módszer általánosságát és rugalmasságát igazolja, hogy a legváltozatosabb statikus és dinamikus problémákra születtek sikeres implementációk (pl. hozzárendelés - quadratic assignment, gyártási sorok ütemezése - sequential ordering, gráfszínezés - graph coloring, többszörös hátizsák probléma - multiple knapsack problem, ütemezési problémák, hálózati forgalomirányítás - network routing, járatütemezési probléma, stb.).

A hangya algoritmus kombinálható a hegymászó és a genetikus algoritmussal is. A hangya algoritmus előnye a genetikus algoritmusokkal szemben, hogy a későbbiekben képes emlékezni a legjobb utakra.

6.6. Tabulistás keresés - taboo search

A tabulistás keresés teljes egészében az eddig ismertetett módszereken alapszik, azok közvetlen továbbfejlesztése. Az aktuális megoldásunk közvetlen környezetében vizsgálódunk új, jobb megoldások felé. A keresést ennél a módszernél állandóan igyekszünk mozgásban tartani, így mindig a környezetben található legjobb megoldás irányában kutatunk tovább, még akkor is, ha az a vizsgált megoldásnál rosszabb.

Mivel ezzel a módszerrel rengeteg lehetőséget kell vizsgálni, így számon kell tartanunk, meg kell jelölnünk, hogy melyik irányban járt már a keresés, és melyik irányban még nem. Ezt úgynevezett *tabu listákkal* oldjuk meg, melyek *tabu táblákból* állnak. A tabu táblák korlátozásai segítik azt, hogy a már bejárt utakat még egyszer ne járjuk be, ezzel növelve a keresés hatékonyságát. Egy keresés folyamán egyre több tabu táblát helyezünk el, amellyel kialakul a tabu lista, amit állandóan figyelembe véve ki tudjuk küszöbölni az algoritmus ciklikusságát. A legoptimálisabb helyet is mindig tárolnunk kell.

6.7. Egyéb példák

A metaheursztikus módszerek közös tulajdonsága egy alárendelt klasszikus heurisztika valamely magasabb szintű koncepció által irányított alkalmazása. A metaheursiztikák a hagyományos probléma-specifikus technikákra alapoznak, de ezeket a legváltozatosabb forrásokból származó (pl. biológia, genetika, fizika, stb.), általánosan alkalmazható elgondolásokkal ötvözik.

Néhány példa arra, hogy a különböző források hogyan kapcsolhatók algoritmusokhoz:

- „Kvantum-számítógépek": Az atommagok körül elektronok keringenek, melyeknek úgynevezett *spinjük* van. A spin nem más, mint az elektronok forgási iránya az atommag körül; ezt általában 0-val és 1-el szokás jelölni, attól függően, hogy jobbról balra vagy fordítva keringenek az elektronok. Ha ütköztetünk két atommagot, akkor a spinek megváltoznak attól függően, hogy milyen atomok ütköztek, illetve hogy milyen körülmények között, stb. Amennyiben minden körülmény ismert lenne, úgy a kvantummechanika bonyolult differenciálegyenletek segítségével ki tudja számolni, hogy pontosan milyen változások mennek majd végbe. A természet azonban egy pillanat töredéke alatt „dönt", és végrehajtódnak a folyamatok. Ilyen értelemben az atomok is „számítógépek", melyek „felmérve" a körülményeket, spineket, stb. azonnal reagálnak egy ütközésre, differenciálegyenletek sokaságait megoldva egy pillanat alatt. (Jelen esetben a számítógépet nem mint perifériákkal rendelkező eszközt tekintjük, hanem mint egy feladatokat megoldó szerkezetet, függetlenül a fizikai megvalósításától.) Ezek a „kvantum-számítógépek" a legkisebb, legelemibb „számítógépek".

- A vízfelület, mint számítógép: A vízfelület a környezeti változások (pl. időjárás) hatására hullámokat képez. Ha meg szeretnénk határozni, hogy a következő pillanatban a vízfelület milyen hullámokat fog előállítani, akkor a vízfelület mozgását leíró differenciálegyenleteket kell megoldanunk. Ez egy ember, sőt még egy számítógép számára is nehéz feladat hiszen rengeteg differenciálegyenletet kell megoldani, ha minden tényezőt figyelembe szeretnénk venni(pl. szél erőssége, folyó sodrása, ágak vízbeesése, örvények keletkezése, stb.). A vízfelület egy másodperc töredéke alatt meghatározza a hullámokat (≈ „megoldja a differenciálegyenleteket"), ebből a szempontból a vízfelület is egy „számítógép". Ha a vízfelületet számítógépként tekintjük, akkor ezen számítógép működéséhez az szükséges, hogy a bemenetet, vagyis a környezeti tényezőket (pl. időjárás) pontosan megadjuk, hiszem maga a számítás már rendkívül gyorsan elvégződik. A bemenet megadása itt igen összetett feladat, sokkal időigényesebb, mint maga a számítás. A vízfelszín párhuzamos (parallel) működésű, minden molekula az összes többivel együtt, összhangban végzi a tevékenységét („számítását"). Egy szekvenciális számítógépnél pont fordítva működik, hiszen általában az input megadása könnyű feladat, de a számítás lassú. Általában elmondható, hogy a parallel folyamatok gyorsabbak, viszont nehezebben kezelhetők. Az emberi gondolkodást is csoportosíthatjuk ezek szerint:

  - „Szekvenciális" gondolkodás: a gondolkodás folyamata (egy probléma megoldása) a kezdeti feltételrendszerből egyfajta sorrend betartásával történik (pl. tétel, bizonyítás, levezetés, következtetés, stb.)
  - Meditatív gondolkodás: meditáció révén egyfajta nyugalmi állapotba kerülve egy kezdeti feltételrendszerből egyszer csak előáll a végeredményt. Ez a párhuzamos gondolkodás szimulációja az agyban.

  Párhuzamos számításra példa, ha egy gráfot elkészítünk spárgából úgy, hogy a gráf csúcsainál legyenek a spárgán csomók. Ha két csomó mentén széthúzzuk a spárgát, megkapjuk a két csúcs közti

legrövidebb utat. Ennek a megoldásnak a számításigénye elhanyagolható, viszont a feladat elkészítése hosszadalmas folyamat.

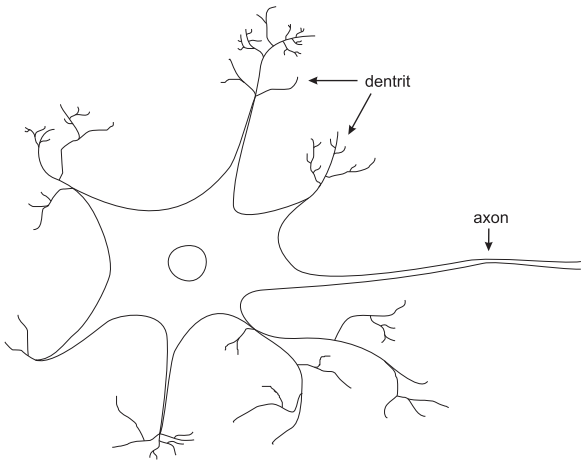
- A genetika és veszélyei: Legyen adott egy elzárt kis sziget, melyen különféle légytörzsek élnek. Az egyes légytörzs egyedeinek átlagos élettartama 3 hónap. Ezen törzs esetében minden 48-dik generáció tagjainak nincsenek szárnyai egy genetikai hiba révén. A tudósok azt tapasztalták, hogy mintegy 100 évente van a szigeten egy olyan időszak, amikor nagyon erős szél fúj a szigeten a szárazföld irányából. Ezen időszak alatt minden szárnnyal rendelkező légy elpusztul, és csak azok a legyek maradnak életben, melyek szárny nélküliek („nyomorékok"). Ezek alapján tehát az evolúció hosszú távon optimalizálást hajt végre az egyedeken. Az ember életében ez azt jelenti, hogy 400 generáció után használjuk ki a genetika hibáját. Az „algoritmus" futási ideje nagy (100 év és 3 hónap), viszont igen hatékony.

7. fejezet

Neurális hálózatok

7.1. Bevezetés

A mesterséges neuronhálók alapötletét szintén a természetből (biológiából) merítették. Az emberi agy nagyon jó feladatmegoldó képességgel bír, hiszen rengeteg olyan problémát képes megoldani, ami matematikailag nehezen megfogható (pl. arcfelismerés, egyensúlyozás, látás vezérlése, stb.), ezért megpróbálták az agy működésének bizonyos aspektusait felhasználni a számítástudományban. Ismert, hogy az emberi agy egymáshoz kapcsolt neuronok (speciális, nyúlványokkal rendelkező idegsejtek) sokasága, azok hálózata. A neuronok együttműködése révén vagyunk képesek problémák megoldására. Egy neuron felépítése a 7.1. ábrán látható.



7.1. ábra. Neuron felépítése.

A neuron nyúlványai közül a rövidebbek, melyek fogadják a más neuronoktól érkező információkat az úgynevezett *dendritek*, a hosszú pedig az úgynevezett *axon*, mely hozzákapcsolódva más neuronok dendritjeihez továbbít elektromos impulzusokat. Ezek a nyúlványok biztosítják a kapcsolatot a neuronok között, vagyis egy ingerület terjedését. A neuronok közti kapcsolat a *szinapszis*.

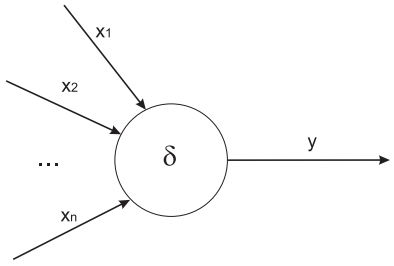
A nyúlványok működése a következő: a szinapszisokon keresztül kémiai *transzmitter* (továbbító) anyagok kerülnek a dendritbe, melyek hatására a sejttest elektromos potenciálja módosul, és amennyiben elér

a potenciál egy küszöbértéket, egy elektromos impulzus halad végig az axonon más neuronok felé. A neuronok kommunikációja tehát a nyúlványaik közvetítésével történik.

Az emberi agy megközelítőleg 10^{11} darab neuronból épül fel, és egy neuronnak csaknem 4000 kapcsolata lehet. Az agyat felfoghatjuk egy hatalmas automataként is, melynek olyan sok állapota van, hogy életünk során nem is tudjuk az összeset kipróbálni. Az agyban több neuron van, mint ahány bit a számítógépes munkaállomásokban, illetve az agyat nagyfokú párhuzamosság jellemzi.

7.2. A perceptron-modell

Az első neuronhálós modellek a *perceptron-modellek* voltak. A perceptron egy számítási modell, mely adott számú bemenetre generál 1 kimenetet, tulajdonképpen egy egyszerűsített természetes neuronmodell. Rosenblatt 1958-ban alkotta meg perceptron-modelljét (lásd 7.2. ábra).



7.2. ábra. Perceptron modellje.

Egy perceptron az alábbi elemekből áll:

- w_i az i. bemeneti (dendrit) csatorna súlya,
- x_i az i. bemeneti (dendrit) csatorna aktuális értéke,
- δ, a perceptronhoz tartozó küszöbérték,
- y, a perceptron kimenete,
- f, a perceptron integrációs függvénye, mely rendszerint a súlyozott összeg,
- g, a perceptron aktivációs függvénye.

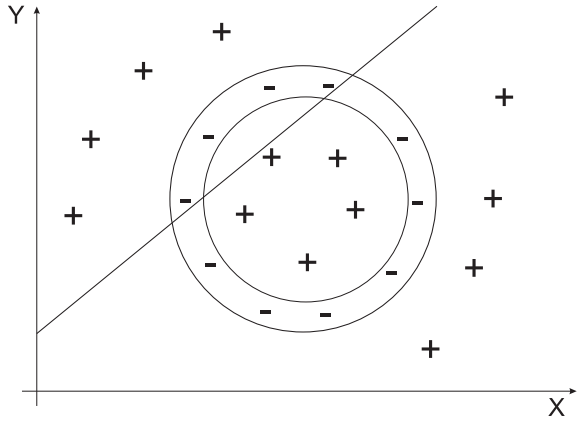
A perceptron-modellben az x_i és y értékek binárisak. Egy perceptronnal csak lineárisan szeparálható mintákat tudunk reprezentálni.

A perceptron tanulási modellben, a tanulás a w_i és δ értékek beállítását jelenti. Tulajdonképpen a perceptron tanulása során egy síkot szeretnénk találni, amely szétválasztja a pozitív és negatív példákat. (A döntési fák esetében csak tengelyekre merőleges síkokat tudunk találni a minták szétválasztására, viszont itt már képesek vagyunk meredekséggel rendelkező szeparáló síkot konstruálni.)

A perceptron-modell tanulási képességének vizsgálatánál a $w_1, w_2, ..., w_n$ súlyokat kell megpróbálni meghatározni. Az adatbázisban (adattáblában) lévő x_i elemekhez (minták) egy meghatározott y értéket

(osztályokat) rendelünk. A tanulóképesség az adatbázisunkkal konzisztens rendszer konstruálását jelenti perceptron segítségével.

A perceptron elmélet megjelenésekor voltak olyan tudósok, akik ellenezték a modellt, bírálták az algoritmus általánosságát, korlátosságát. Például ilyen volt Marvin Minski, aki még egy tanulmányt is írt a perceptron-modell hasztalanságáról. Híres ellenpéldája az algoritmus általánosságával szemben.

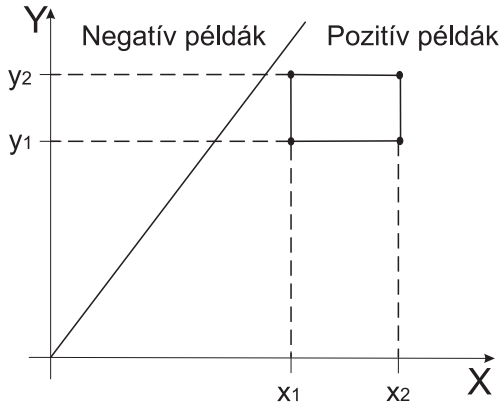


7.3. ábra. Marvin Minski ellenpéldája az algoritmus általánosságával szemben.

Belső körben legyenek a pozitív példák, a külső gyűrűben pedig a negatív példák (7.3. ábra). Ebben az esetben nem tudunk egyetlen olyan hipersíkot sem venni, ami szétválasztaná a példákat. Ez a megállapítás valóban igaz, a perceptron-modell képtelen egy ilyen mintahalmazt kezelni, viszont nem létezik olyan algoritmus, mely minden problémát képes lenne megoldani. Azonban attól, hogy létezik olyan probléma, amit az adott algoritmus nem tud kezelni, még nem biztos, hogy az algoritmus rossz.

A perceptron-modell hatékonyságát pedig az alábbiakkal támadta Minski: a példákat nem szabályokkal írja le, hanem szeparáló síkkal, ami nem elég egzakt.

Tekintsünk egy szeparáló síkot, mely pozitív és negatív példákra bontja a mintahalmazt! Ha a példákat szabályokkal szeretnénk körülírni, akkor ezt „négyszögek” definiálásával tudjuk megtenni, mely négyszögek a példák egy halmazát fedik le (7.4. ábra).



7.4. ábra. Pozitív és negatív példákat elválasztó szeparálósík.

Így rengeteg négyszöget kell definiálnunk annak érdekében, hogy a példákat szét tudjuk választani, tehát rengeteg szabályra lenne szükség. A szeparáló sík megadása jóval tömörebb formában tudja tárolni

az információt, mint a szabályrendszerek (jóval kevesebb paraméter), tehát igazából ez az érv nem szól a perceptron-modell ellen.

7.3. A perceptron konvergencia tétele

A perceptron tanulása nem más, mint a w_i súlyok, és a δ aktivációs szint meghatározása. Ebben az esetben δ egy konstanst jelent, ilyen értelemben más, mint a w_i súlyok. Vegyünk fel még egy bemeneti élt a rendszerben -1 súllyal! Ekkor a $\sum w_i x_i \geq \delta$ feltétel az alábbira módosul: $\sum w_i x_i + (-1)\delta > 0$. Így a tanulás kezelhetőbbé válik:

$$\begin{aligned} \textstyle\sum_{i=0}^{n} w_i x_i \geq 0 \Rightarrow \quad & \underline{x}(n) = [-1, x_1(n), x_2(n), \cdots, x_p(n)] \\ & \underline{w}(n) = [\Theta(n), w_1(n), w_2(n), \cdots, w_p(n)] \end{aligned}$$

Jelöljék az osztályokat a $\mathcal{C}_1, \mathcal{C}_2$ betűk! Ekkor:

$$\begin{aligned} ha \quad \underline{w}^T \underline{x} \geq 0 \quad &\rightarrow \quad \mathcal{C}_1 \\ ha \quad \underline{w}^T \underline{x} < 0 \quad &\rightarrow \quad \mathcal{C}_2 \end{aligned}$$

Továbbra is csak olyan adatokkal dolgozunk, melyek lineárisan szeparálhatók. A w_i súlyok csak akkor kerülnek módosításra, ha az osztályozás nem megfelelő. Vezessük be az $\eta(n)$ tanulási konstanst! Ekkor:

$$\underline{w}(n+1) = \underline{w}(n) - \eta(n) \cdot \underline{x}(n)$$

Tétel (a perceptron konvergencia tétele): Ha η értéke elég kicsi, és a példahalmaz lineárisan szeparálható, akkor az algoritmus minden esetben véges lépésben konvergál egy olyan súlyvektorhoz, mely lineárisan szeparálja a példahalmazt.

Bizonyítás:

Legyen a bejövő mintahalmaz $\underline{x}(n) = [-1, x_1(n), \ldots, x_p(n)]^T$, és a hozzájuk tartozó súlyok $\underline{w}(n) = [\Theta(n), w_1(n), \ldots, w_p(n)]^T$! A szeparáló sík egyenlete: $\underline{v}(n) = \underline{w}^T \underline{x}$.

Ha $\underline{w}^T \underline{x} > 0$, akkor $\underline{x} \in \mathcal{C}_1$, egyébként $\underline{x} \in \mathcal{C}_2$. Tegyük fel, hogy adott súlyok mellett egy olyan $\underline{x}$ vektort kapunk, amely nem teljesíti ezeket a feltételeket, ezért a súlyokat módosítani kell (η - tanuló konstans)!

$$\underline{w}(n+1) = \underline{w}(n) - \eta \underline{x}(n), \text{ ha } \underline{w}^T(n)\underline{x}(n) > 0 \text{ és } \underline{x} \in \mathcal{C}_2 \tag{7.1}$$

$$\underline{w}(n+1) = \underline{w}(n) + \eta \underline{x}(n), \text{ ha } \underline{w}^T(n)\underline{x}(n) \leq 0 \text{ és } \underline{x} \in \mathcal{C}_1 \tag{7.2}$$

Tekintsük a második (7.2) esetet! Ekkor

$$\begin{aligned} \underline{w}(n+1) &= \underline{w}(n) + \underline{x}(n) \\ \underline{w}(n+1) &= \underline{x}(1) + \cdots + \underline{x}(n) \end{aligned}$$

Feltettük, hogy a vektorrendszer lineárisan szeparálható, azaz $\underline{w}(n)^T\underline{x}(n) > 0$, ha $\underline{x} \in \mathcal{C}_1$. Legyen

$$\alpha = \min_{\underline{x}(n)\in\mathcal{C}_1} \underline{w_0}^T\underline{x}(n). \tag{7.3}$$

Az előbb kapott egyenlőség mindkét oldalát megszorozva w_0^T-tal, a következőt kapjuk:

$$\begin{array}{rcl} \underline{w_0}^T\underline{w}(n+1) & = & \underline{w_0}^T\underline{x}(1) + \cdots + \underline{w_0}^T\underline{x}(n) \\ ||\underline{w_0}||^T||\underline{w}(n+1)|| & \geq & n\cdot\alpha \end{array}$$

$$||a||^2||b||^2 \geq ||a\cdot b||^2 \tag{7.4}$$

Erre a Cauchy-Schwartz-Bunyakovszki egyenlőtlenséget (7.4) alkalmazva kapjuk:

$$\begin{array}{rcl} ||\underline{w_0}||^2||\underline{w}(n+1)||^2 & \geq & ||\underline{w_0}^T\underline{w}(n+1)||^2 \\ ||\underline{w_0}||^2||\underline{w}(n+1)||^2 & \geq & n^2\cdot\alpha^2 \\ ||\underline{w}(n+1)||^2 & \geq & \frac{n^2\cdot\alpha^2}{||\underline{w_0}||^2} \end{array}$$

Térjünk vissza a $\underline{w}(k+1) = \underline{w}(k) + \underline{x}(k)$, $k = 1, \ldots, n$ egyenletekhez! Négyzetre emelve egy ilyen egyenlet mindkét oldalát, kapjuk:

$$||\underline{w}(k+1)||^2 = ||\underline{w}(k)||^2 + ||\underline{x}(k)||^2 + 2||\underline{w}^T(k)\underline{x}(k)||$$

$||\underline{w}^T(k)\underline{x}(k)|| \leq 0$ a feltétel szerint (7.2), amiből következik:

$$||\underline{w}(k+1)||^2 \leq ||\underline{w}(k)||^2 + ||\underline{x}(k)||^2 \tag{7.5}$$

Összegezve ezeket az egyenlőtlenségeket $k = 1, \ldots, n$-re, kapjuk:

$$||\underline{w}(n+1)||^2 \leq \sum_{k=1}^{n} ||\underline{x}(k)||^2$$

Legyen $\beta = \max_{\underline{x}(k)\in\mathcal{C}_1} ||\underline{x}(k)||^2$! Ekkor:

$$||\underline{w}(n+1)||^2 \leq n\beta$$

Az eddigieket felhasználva:

$$\begin{array}{rcl} \frac{n^2\alpha^2}{||w_0||^2} & = & n\beta \\ n & = & \frac{\beta||w_0||^2}{\alpha^2} \end{array}$$

Ez lesz az az n, aminél meg kell álljon az algoritmus.

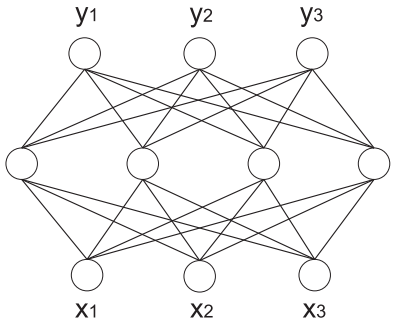
Megjegyzés:

- A tétel elméleti jellegű, gyakorlati tanácsot nem ad n meghatározására.
- η-ra 1-nél kisebb számot célszerű adni, és folyamatosan lehet változtatni.
- A perceptron az őt ért aktivitástól függően vagy ad kimenetet, vagy nem. Úgynevezett ugró függvényt használtunk ennek eldöntésére.

7.4. A neuronhálózatok és a Backpropagation algoritmus

7.4.1. Neuronhálózatok

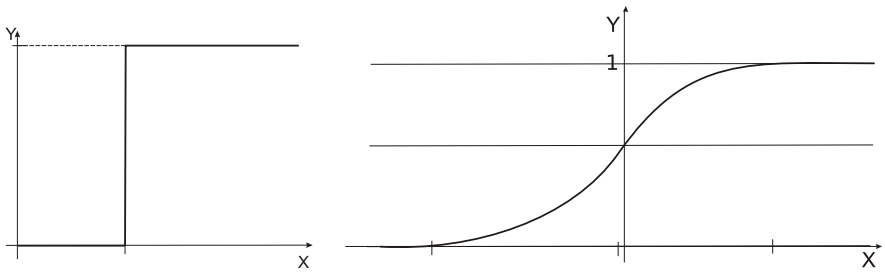
Egy perceptron önmagában számos problémát képes ugyan megoldani, viszont nem elég univerzális modell. Ha azonban több perceptront kötünk össze hálózattá, egy sokkal univerzálisabb struktúrát kapunk. A neuronhálózat perceptronok rendszere, több egymás után kapcsolt perceptron (7.5. ábra). A neurális hálózatok legáltalánosabban használt változata a *feed-forward*, vagy előrecsatolt hálózatok, ahol a perceptronok rétegekbe szerveződnek, és az ingerület mindig a következő réteg perceptronjaira tevődik át, nincs visszacsatolás, vagy rétegen belüli ingerület átadás.



7.5. ábra. Neuronháló.

Egyetlen közti réteg elegendő, hogy az előbbiekben említett (Minski-féle) koncentrikus kör példát az algoritmus meg tudja tanulni.

A perceptron kimenetére egy szigorú feltétel van: $\sum w_i x_i > \delta$, csak ekkor generálódik kimenet. Ez felfogható, mint egy *ugró függvény* vagy *„hard limiter”*.



a) b)

7.6. ábra. a) Ugró függvény; b) Szigmoid függvény

A valós életben azonban többnyire a döntéseknél „soft limiter” rendszereket használunk, vagyis nincsenek ilyen éles határok ($\approx$ fuzzy). Tehát az ugró függvény helyett érdemesebb szigmoid függvényt alkalmaznunk. Általában a neuronok szigmoid aktivációs függvénnyel dolgoznak, amelynek meg van az az előnye, hogy minden pontban differenciálható, az ugró függvénnyel ellentétben. A tanulási eljárás szigmoid

aktivációs függvény esetén gradiens módszer alapján történik.

$$\begin{aligned} g(x) &= \frac{1}{1+e^{-\lambda(x-\delta)}} \\ g'(x) &= \frac{e^{-x}}{(1+e^{-x})^2} = \\ &= \frac{e^{-x}}{1+e^{-x}}\ \frac{1}{1+e^{-x}} = \\ &= \frac{1}{e^{x}+1}\ \frac{1}{1+e^{-x}} = \\ &= \left(1-\frac{1}{1+e^{-x}}\right)\ \frac{1}{1+e^{-x}} = \\ &= (1-g(x))\,g(x) \quad (\lambda=1 \text{ és } \delta=0 \text{ esetén}) \end{aligned}$$

A perceptron tanulási algoritmus viselkedése rossz, ha a példahalmaz nem lineárisan szeparálható. A gradiens alapú keresés azonban a legjobban elválasztó hipersíkhoz fog konvergálni. Az alapötlet a következő: javítás során gradiens alapú keresést valósítunk meg a háló lehetséges súlyainak terében.
Megtanulandó: $d:\ R^n \to R^m$ függvény, $\underline{d} = (d_1, d_2, \ldots, d_m)$.
Ismert: $(\underline{x}_i, \underline{d}_i)$, $i = 1, \ldots, k$ input-output párok.
A hálózat által kiszámított függvény:

$$F(\underline{w}, \underline{x}) = (F_1(\underline{w}, \underline{x}), F_2(\underline{w}, \underline{x}), \cdots, F_m(\underline{w}, \underline{x}))$$

Minden súlyvektorhoz megadunk egy hibaértéket:

$$\sum_{i=1}^{k}\left(\sum_{l=1}^{m}(d_l(\underline{x}_i) - F_l(\underline{w}, \underline{x}_i))^2\right)$$

Vezessünk be egy energia függvényt! Az energia függvény képlete legyen:

$$E(\underline{w}, \underline{x}) := \frac{1}{2}\sum_{l=1}^{m}(d_l(\underline{x}) - F_l(\underline{w}, \underline{x}))^2$$

Ez egy parabolikus felület egy globális minimummal. A felületen a gradiens vektorral ellentétes irányba fogunk lépni. Legyen $E'(\underline{w}, \underline{x})$ a w változók szerinti parciális deriváltakból álló vektor.

$$E'(\underline{w}) = \left[\frac{\partial E}{\partial w_0}, \ldots, \frac{\partial E}{\partial w_n}\right]$$

Legyen $a = \sum_{i=1}^{n} w_i x_i$ illetve $y = g(a)$

$$\frac{\partial E}{\partial w_i} = -(d - g(a)) \cdot g'(a) x_i = -(d-y) \cdot g'(a) x_i$$

Módosítsuk a súlyokat!

$$\underline{w}_{uj} = \underline{w}_{regi} - \eta \nabla E(\underline{w})$$

$$\underline{w}_{uj} = \underline{w}_{regi} - \eta(d-y) \cdot g'(a) x_i$$

$$\underline{w}_{uj} = \underline{w}_{regi} - \eta(d-y) \cdot y(1-y) x_i$$

Megjegyzés:

- A gradiens alapú keresésnél egyszerre dolgoztuk fel az összes példát.
- Ha tökéletes szeparáció nem lehetséges, akkor az algoritmus minimális hibájú súlyokhoz konvergál, azonban nem garantált, hogy véges lépés után véget ér.

7.4.2. Perceptron tanítása gradiens eljárással

Közelítő eljárás: egy adott pontból indulva a pont gradiensének irányába lép, ami az új közelítő érték lesz. Gradiens eljárással keresi a hipersíkot.

Perceptron függvény:

$$J_p(\underline{w}) = \sum_{y \in Y} \underline{w}^T \cdot \underline{y}, \tag{7.6}$$

ahol Y a rosszul osztályozott pontok halmaza. Akkor kapunk szélsőértéket, ha nincs rosszul osztályozott pontunk. Gradiens jele: ∇

Vennünk kell a parciális deriváltakat w szerint, a perceptron gradiensét:

$$\nabla J_p(\underline{w}) = -\sum_{\underline{y} \in Y} \underline{y} \tag{7.7}$$

Általános gradiens eljárás

$$\underline{w}_{k+1} = \underline{a}_k + \eta(k) \cdot \nabla J_p(\underline{w}), \tag{7.8}$$

ahol $\underline{w}_{k+1}$ a keresett vektor következő közelítése, $\eta(k)$ skálázó konstans. A gradiens irányába lépünk. A perceptron gradiens eljárás:

$$\underline{w}_{k+1} = \underline{a}_k - \eta(k) \cdot \sum_{\underline{y} \in Y} \underline{y} \tag{7.9}$$

Példa:

Legyen $\eta(k) = 1$, minden k-ra értékre.

$$\underline{w}^T \cdot \underline{y} > 0,$$

ahol $\underline{w}_T$ vektor, $\underline{y}$ pedig a pontok. Ebbe a képletbe kell behelyettesíteni, ahol nem áll fenn a reláció (azaz nem nagyobb, mint 0), azok a pontok rosszak. Tehát a tükrözött pontok:

$$\begin{matrix} \text{A(-2,1)} \\ \text{B(2,4)} \\ \text{C(4,2)} \\ \text{D(5,1)} \end{matrix} \qquad \underline{w}_0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

1. lépés: $\underline{w}_0^T \cdot \underline{y} > 0$

A pontra $(1,1) \cdot \begin{pmatrix} -2 \\ 1 \end{pmatrix} = \text{-2+1} = \text{-1} < 0$ A rossz pont

B pontra $(1,1) \cdot \begin{pmatrix} 2 \\ 4 \end{pmatrix} = 2+4 = 6 > 0$ B pont megfelel

C pontra $(1,1) \cdot \begin{pmatrix} 4 \\ 2 \end{pmatrix} = 4+2 = 6 > 0$ C pont megfelel

D pontra $(1,1) \cdot \begin{pmatrix} 5 \\ 1 \end{pmatrix} = 5+1 = 6 > 0$ D pont megfelel

A gradiens: $\underline{w}_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} + 1 \cdot \underbrace{\begin{pmatrix} -2 \\ 1 \end{pmatrix}}_{Arosszpont} = \begin{pmatrix} -1 \\ 2 \end{pmatrix}$, $\underline{w}_1^T = (-1, 2)$

2. lépés: $\underline{w}_1^T \cdot \underline{y} > 0$

A pontra $(-1,2) \cdot \begin{pmatrix} -2 \\ 1 \end{pmatrix} = 2+2 = 4 > 0$ A pont megfelel

B pontra $(-1,2) \cdot \begin{pmatrix} 2 \\ 4 \end{pmatrix} = \text{-2+8} = 6 > 0$ B pont megfelel

C pontra $(-1,2) \cdot \begin{pmatrix} 4 \\ 2 \end{pmatrix} = \text{-4+4} = 0 < 0$ C rossz pont

D pontra $(-1,2) \cdot \begin{pmatrix} 5 \\ 1 \end{pmatrix} = \text{-5+2} = \text{-3} < 0$ D rossz pont

A gradiens: $\underbrace{\underline{w}_2}_{\underline{w}_{k+1}} = \underbrace{\begin{pmatrix} -1 \\ 2 \end{pmatrix}}_{\underline{w}_k} + 1 \cdot \underbrace{\begin{pmatrix} 9 \\ 3 \end{pmatrix}}_{\sum rossz\ pontok(C,D)} = \begin{pmatrix} 8 \\ 5 \end{pmatrix}$, $\underline{w}_2^T = (8, 5)$

rossz pont:A $\quad \underline{w}_3 = \begin{pmatrix} 6 \\ 6 \end{pmatrix} \quad \underline{w}_3^T = (6,6)$

rossz pont:A $\quad \underline{w}_4 = \begin{pmatrix} 4 \\ 7 \end{pmatrix} \quad \underline{w}_4^T = (4,7)$

rossz pont:A $\quad \underline{w}_5 = \begin{pmatrix} 2 \\ 8 \end{pmatrix} \quad \underline{w}_5^T = (2,8)$

nincs rossz pont!

Tehát a $(2,8)$ pontba mutató vektor normálisát (rá merőleges egyenesét) vesszük, akkor kapunk egy megoldás egyenest.

Példa:

Az előző példa, csak itt $\eta(k) = 0,1$, azaz sokkal finomabb a lépésköz, amivel toljuk majd a megfelelő irányba az egyenesünket, folyamatosan közelítve a megoldáshoz.

$$\underline{w}_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} + 0,1 \cdot \begin{pmatrix} -2 \\ 1 \end{pmatrix} = \begin{pmatrix} 0,8 \\ 1,1 \end{pmatrix}$$

$$\underline{w}_2 = \begin{pmatrix} 0,6 \\ 1,2 \end{pmatrix}$$

$$\underline{w}_3 = \begin{pmatrix} 0,4 \\ 1,3 \end{pmatrix}$$

Apró lépésenként mozgatja lefelé a rossz pontok irányába az egyenest. Szeparálható pontok esetén jó ez az eljárás, ellenkező esetben viszont nincs minimum.

Perceptron egy lépéses gradiens eljárás

Adottak a pontok (A,B,C,D), ciklikusan ismételve menjünk sorban végig a pontokon, ha a vizsgált pont jó, akkor menjünk tovább. Ha rossz, akkor azt az egy pontot módosítsuk, majd lépjünk tovább. Akkor áll meg az eljárás, ha már minden pont jó.

A	B	C	D	A	B	C	D	A	...
rossz	jó	rossz	jó	rossz	jó	jó	jó	jó	
$\underline{w}_1 = \begin{pmatrix} -1 \\ 2 \end{pmatrix}$		$\underline{w}_2 = \begin{pmatrix} 3 \\ 4 \end{pmatrix}$		$\underline{w}_3 = \begin{pmatrix} 1 \\ 5 \end{pmatrix}$					

Tehát $\underline{w}_3$-ra nézve jó lesz mind a négy pont, így 3 lépésben megkaptuk a jó megoldást ezzel a módszerrel.

$$\underline{w}_{k+1} = \underline{w}_k + \underbrace{\eta(k)}_{1} \cdot \sum_{\underline{y} \in Y} \underline{y} = \underline{w}_k + \underline{y}_k,$$

ahol $\underline{y}_k$ adott pillanatban egy rossz pont koordinátái. Az eljárás konvergens, ha szeparálható ponthalmazunk van.

Perceptron gradiens eljárás hiba becslése

Tegyük fel, hogy szeparálható ponthalmazunk van, $\underline{\hat{w}}$ a feladat egy lehetséges megoldása. Azt becsüljük, hogy az adott megoldás mennyivel tér el az optimálistól.

$$||\underline{w}_{k+1} - \alpha \cdot \underline{\hat{w}}||^2 = ||\underline{w_k} - \alpha \cdot \underline{\hat{w}} + \underline{y}_k||^2 = ||\underline{w}_k - \alpha \cdot \underline{\hat{w}}||^2 + 2 \cdot (\underline{w}_k - \alpha \cdot \underline{\hat{w}})^T \cdot \underline{y}_k + ||y_k||^2$$

$\underline{y}$ rosszul osztályozott pont, $\underline{y}_k \cdot \underline{w}_k$ negatív lesz, ezért ha ezt elhagyjuk, akkor

$$\leq ||\underline{w}_k - \alpha \cdot \underline{\hat{w}}||^2 - 2 \cdot \alpha \cdot \underline{\hat{w}} \cdot \underline{y}_k + ||\underline{y}_k||^2$$

Igaz az állítás, ha α-t elég nagyra választjuk, ekkor$2 \cdot \alpha \cdot \underline{\hat{w}} \cdot \underline{y}_k > 2 \cdot ||\underline{y}_k||^2$
ezért teljesül:

$$\leq ||\underline{w}_k - \alpha \cdot \underline{\hat{w}}||^2 - ||\underline{y}_k||^2$$

7.4.3. Backpropagation algoritmus

A „Backpropagation" angol kifejezés, szó szerint visszacsatolást jelent, amely jól tükrözi az algoritmus alapötletét: visszafelé haladva a hálózatban, előrejelzések segítségével módosítjuk a súlyokat.

Algoritmus:

1. Inicializálás: a súlyoknak kis véletlenértékeket adunk
2. Iteráció: amíg a megállási feltétel nem teljesül, hajtsuk végre minden tanulópéldára a következőket:

   a.) Engedjük át a tanulópéldát a neuronhálón!

   b.) Legyen a k a kimenőegység hibája: $\delta_k = y_k(1 - y_k)(d_k - y_k)$

   c.) Legyen a h a rejtett egység hibája (rétegenként visszafelé):

   $$\delta_h = y_h(1 - y_h) \sum_{k \in Uses(h)} w_{kh}\delta_k$$

   d.) $w_{ji} = w_{ji} + \eta\delta_j x_{ji}$, minden i-ből j-be menő párra

Az új súlyokat az alábbi levezetéssel kaphatjuk meg:

- Egy neuron tanítása szigmoid függvénnyel:

$$E(\underline{w}, \underline{x}) = \frac{1}{2}\left(d - g\left(\sum_{i=1}^{n} w_i \cdot x_i\right)\right)^2$$

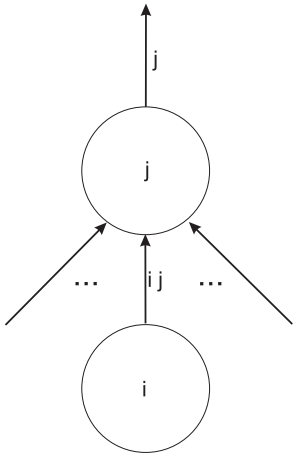
$$a := \sum_{i=1}^{n} w_i \cdot x_i$$

$$\frac{\partial E}{\partial w_i} = -(d - \underbrace{g(a)}_{y}) \cdot g'(a) \cdot x_i = -(d-y) \cdot g'(a) \cdot x_i$$

$$\underline{w}_{uj} = \underline{w}_{regi} + \eta \underbrace{(d-y)g'(a)}_{\delta = \frac{\partial E}{\partial a}} \underline{x} = \underline{w}_{regi} + \eta \cdot (d-y) \cdot y \cdot (1-y) \cdot \underline{x}$$

- Neuronhálózat tanítása szigmoid függvénnyel:

  Output neuron esetén:

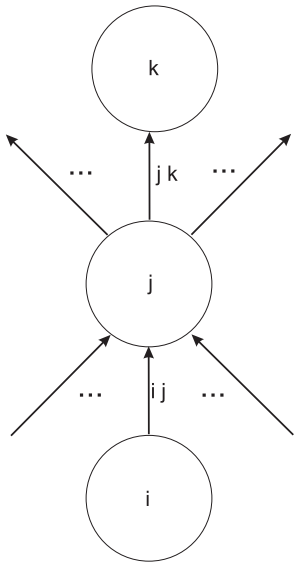


7.7. ábra. Output neuron esetén.

$$w_{ij} = w_{ij} + \eta \underbrace{(d_j - y_j) g_j'(a_j)}_{\delta_j = \frac{\partial E}{\partial a_{ij}}} y_i,$$

ahol tetszőleges k-val indexelt neuronra o_k a neuron kimenete, g_k ezen neuronra alkalmazott aktivációs függvény, w_{ij} pedig az i-vel indexelt neuronból a j-vel indexelt neuronba vezető kapcsolat súlya, illetve $a_j = \sum w_{ij} y_i$.

Közbülső neuron esetén:



7.8. ábra. Közbülső neuron esetén.

$$E(\underline{w},\underline{x}) = \cdots g_k\left(\underbrace{\sum w_{jk}\, g_j\left(\underbrace{\sum w_{ij}y_i}_{a_j}\right)}_{y_j}\right)\cdots$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}} = \underbrace{\left(\sum_k \underbrace{\frac{\partial E}{\partial a_k}}_{\delta_k} w_{jk}\right) g_j'(a_j)\, y_i}_{\delta_j}$$

Mivel a felhasznált g függvényre $g'(a) = g(a)(1-g(a)) = y\cdot(1-y)$, így a súlyok:

1. Output neuron esetén:

$$w_{ij} = w_{ij} + \eta\delta_j y_i, \text{ ahol } \delta_j = (d_j - y_j)y_j(1-y_j)$$

2. Közbülső neuron esetén:

$$w_{ij} = w_{ij} + \eta\delta_j y_i, \quad \text{ahol } \delta_j = \left(\sum_k \delta_k w_{jk}\right) y_j\,(1-y_j)$$

Megjegyzés:

- A rejtett (= hidden) rétegek számáról az algoritmus nem ad leírást, az szabad paraméter marad (problémától függően változtathatjuk a számát).

- Az algoritmus során először nagy lépesekkel érdemes közelíteni az optimumot, majd egyre kisebb lépésekkel. Ezt szabályozza a tanuló konstans.

- A neuronháló minden része össze van kötve. A súlyok beállítása hosszú idő, nagy tapasztalatot igényel.

A Backpropagation algoritmusnak különböző változatai vannak, attól függően, hogy

- milyen minimalizációs módszert használunk,
- hogyan adjuk meg a kezdeti súlyokat,
- hogyan adjuk meg a hálózat architektúráját.

A Backpropagation algoritmust felhasználják például tőzsdei árfolyamok előrejelzésére, hadiiparban, de akár felhasználható egy totószelvény kitöltésére is.

A neuron modell hátránya:

A neuron nem csak akkor ad kimenetet, ha az inputok súlyozott értéke meghaladja a küszöböt. A természetben azonban ez a küszöb valóban létezik, vagyis a matematikai modell távol áll a biológia analógiától. Míg a biológiai modellben van visszacsatolás, addig a neuronban nincs. Másrészt az evolúciós folyamat nem optimalizációs folyamat.

Visszacsatolást megvalósító egyéb modellek:

1. A Hebb féle tanulási eljárás visszacsatolt rendszert valósít meg.

2. Kohonen pedig az asszociatív jelleget vette figyelembe, lényegében a párhuzamos feldolgozás során tartományokat vett sorra. Ezt asszociatív klaszterezésnek nevezzük.

3. Radial basic function:
   Az e^{-x^2} a standard normális eloszlás képlete, melynek speciális tulajdonsága az, hogy nincsen primitív függvénye. A statisztikában ma is táblázatot használnak a standard normális eloszlás értékének meghatározásához. A standard normális eloszlás függvényapproximációja $\sum w, e^{-(\frac{x-a}{b})^2}$, melynek segítségével közelíteni lehet az értéket. Ez is egy tanulási eljárás.

4. Reinforcement learning (=megerősítéses tanulás):
   Például egy labirintusban az egér keresi a sajtot. Az egér minden egyes ajtóhoz „hozzárendel” egy valószínűséget, annak fényében, hogy mennyire érdemes belépni oda. A tanulás során ezeket a valószínűségeket sajátítja el. Az egér algoritmusa azonban diszkrét feladatot képes csak megtanulni.

8. fejezet

Interaktív bizonyítások

Berkeley gondolata:

„A világ dolgai nem léteznek, gyakorlatilag az egész világ csak virtualitás."

Berkeley tagadta a világ létezését azzal, hogy minden csak az érzékeinken (tapintás, látás, szaglás, stb.) alapszik. Ebből a példából is látható, hogy vannak olyan tézisek, melyek helyességét nem tudjuk bizonyítani a matematikából megismert klasszikus módszerekkel.

Tegyük fel, hogy egy olyan algoritmusunk van, amely meghatározza, hogy egy fán hány falevél található. Nyilvánvaló, hogy nem tudjuk ellenőrizni az algoritmust, hiszen ehhez meg kellene számlálni a faleveleket, ami szinte teljesen lehetetlen, mert rengeteg levél található a fán, és túlzottan hosszú ideig tartana az ellenőrzés ezen formája. Ha azonban elveszünk 20 levelet, majd újra lefuttatjuk az algoritmust és eredményül 20-szal kevesebb levelet tippel az algoritmusunk, mint az előbb, akkor már biztosabbak lehetünk annak helyességében. Ha elvégezzük ezt a kísérletet 100-szor és mindannyiszor helyes választ kapunk, akkor az algoritmus jóságának valószínűsége minden próbálkozással nő.

A fenti bizonyítás során nincs információnk a helyes megoldás értékéről. Azonban tudjuk azt, hogy a bemeneti értékek bizonyos módosítása mekkora változást kell eredményezzen a kimenetben. Tehát a fenti bizonyítás interakción alapszik. Ezt az újfajta bizonyítási eljárást 2000 környékén vezették és az *interaktív bizonyítás* nevet kapta.

A látási és tapintási ingereink egybeesnek attól függetlenül, hogy feltételezzük, hogy a minket körülvevő világ csupán vizualitás. Ha a kezünkbe veszünk egy tárgyat, attól még, hogy elhisszük Berkeley tézisét, látjuk, hogy a tárgy hol van, tapintásunk, és látásunk is alátámasztja a tényt, hogy a tárgy a kezünkben van. Ez az agy számára egy interaktív bizonyítás lehet arra vonatkozólag, hogy a valóság létezik. Kötődünk a minket körülvevő világhoz szaglásunkkal, érzékszerveinkkel és mentális képességeinkkel állandóan „ellenőrizzük" ezt a minket körülvevő világot. A valóságot elképzelhetjük virtuálisnak, de annak a valószínűsége, hogy minden érzékszervünk úgy csal, hogy konzisztens módon el tudja fedni a valóságot, nagyon kicsi.

Véges számú jó eset nem bizonyíthatja egy algoritmus helyességét. Azonban megfelelő bizonyítást adhat a valószínűségszámítás és az interaktív bizonyítás együttes használata. A klasszikus matematikában

nem létezik indukció olyan értelemben, hogy véges esetből induktív módon következtethetünk végtelen sok esemény kimenetére, de az interaktív bizonyítás esetén néhány (100) véges esetből *nagy valószínűséggel* tudunk következtetni a végtelenre.

8.1. Véletlen számok alkalmazásai az interaktív bizonyításban

Tekintsük az $(x-y)(z-u)-(x-z)(y-u)+(x-u)(y-z)=0$ egyenletet! A kérdés az, hogy ez vajon minden esetben teljesül-e. A zárójeles szorzatok kibontásával

$$(x-y)(z-u)-(x-z)(y-u)+(x-u)(y-z) = xz-xu-yz+yu-xy+xu+yz-uz+xy-xz-yu-uz = 0, \quad (8.1)$$

tehát azonossághoz jutunk.

Elvileg ezzel a módszerrel véges idő alatt bármilyen polinomokra vonatkozó azonosságot ellenőrizni tudunk, ám ha csak kéttagú összegeink vannak, de 50 tényező, akkor a szorzat kibontásához 2^{50} tagot kellene felírnunk, ez pedig a legfejlettebb számítógépek segítségével is rengeteg időbe tellene. Így más megoldást kell keresnünk a problémára.

Egy konkrét helyettesítésre könnyű az azonosságot ellenőrizni: ehhez ugyanis nem kell zárójeleket felbontani, hanem elvégezhetjük a kijelölt műveleteket úgy, ahogyan a zárójelezés mutatja. Ha kellően sok különböző, véletlenszerűen választott helyettesítést tekintünk, akkor valószínűleg a polinom értéke csak akkor lesz minden esetben 0, ha azonosságról van szó.

Pontosítsuk ezt a gondolatot. A következőkben tekintsük az $f(x_1, x_2, \ldots, x_n)$ polinom minden olyan helyettesítését, ahol $x_i \in \{0, 1, \ldots, N-1\}$, és f foka legyen legfeljebb k. Be fogjuk látni, hogy

ha f nem azonosan 0, akkor a tekintett N^n helyből legfeljebb kN^{n-1} esetben veszi fel a 0 értéket.

Bizonyításunkhoz teljes indukciót alkalmazunk n-re: $n=1$ esetén egy k-ad fokú egyváltozós polinomot vizsgálunk, amelynek valóban legfeljebb k-db valós gyöke van, így állításunk erre az esetre igaz. Tegyük fel, hogy $n>1$, és hogy $n-1$ változóra igaz az állítás! Az f polinomot felírhatjuk a következő alakban:

$$f(x_1, \ldots, x_n) = f_0(x_1, \ldots, x_{n-1}) + x_n f_1(x_1, \ldots, x_{n-1}) + \cdots + x_n^r f_r(x_1, \ldots, x_{n-1}), \quad (8.2)$$

és nyilván $r \le k$, illetve $f_r(x_1, \ldots, x_{n-1})$ foka legfeljebb $k-r$. Rögzítsük először $x_1, \ldots, x_{n-1}$ értékét! Ekkor f az x_n változó legfeljebb r-ed fokú polinomja, amelyben az együtthatók az f_i polinomok helyettesítési értékei. Ha $f_r(x_1, \ldots, x_{n-1}) \neq 0$, akkor ez az összeg x_n r-ed fokú polinomja, szükségképp legfeljebb r db egész gyöke van, így a lehetséges N db x_n értékből legfeljebb r fogja teljesíteni az egyenletet. Mivel $x_1, \ldots, x_{n-1}$ lehetséges különböző értékeinek száma N^{n-1}, a polinom legfeljebb rN^{n-1} ilyen tulajdonságú helyen tűnhet el. Ha $f_r(x_1, \ldots, x_{n-1}) = 0$, akkor akár az összes N db tekintett x_n helyen eltűnhet a polinom (ha minden együttható 0); az ilyen esetek száma viszont legfeljebb $(k-r)N^{n-2}$ lehet az indukciós feltevésünk miatt, így legfeljebb $(k-r)N^{n-2}N = (k-r)N^{n-1}$ ilyen típusú helyen tűnhet el a polinom. A két esetet összegezve kapjuk, hogy az $f(x_1, \ldots, x_n)$ polinom legfeljebb $rN^{n-1} + (k-r)N^{n-1} = kN^{n-1}$ esetben veszi fel a 0 értéket, ezzel állításunkat beláttuk.

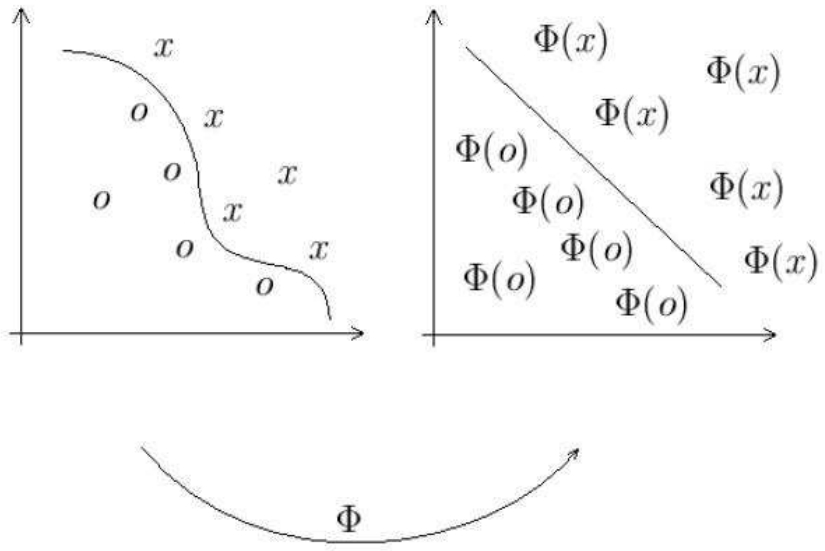
Tekintve, hogy összesen N^n esetünk van, annak a valószínűsége, hogy egy véletlen helyettesítésre a helyettesítési érték 0 lesz legfeljebb k/N. Így $N = 2k$ esetén $1/2$ valószínűséggel kiderül, ha az adott egyenlet nem azonosság. Ez persze túl nagy esélyt ad a hibának; de megismételhetjük a kísérletet egymás után mondjuk 20-szor, akkor a hiba valószínűsége kisebb lesz, mint $1/2^{20}$, ami már általában teljesen elfogadható.

Így véletlen számokat használva, egy polinomról viszonylag hatékonyan ki tudjuk deríteni, hogy azonosan 0-e vagy sem. Érdekes megjegyezni, hogy véletlen számok használata nélkül nem ismeretes olyan módszer, amely elkerülné a bevezetőben említett „exponenciális robbanást”.

9. fejezet

A kernel-alapú módszerek áttekintése

A kernel-alapú módszerek az alakfelismerési (pattern analysis) problémák megoldásának egy megközelítése. Ennek lényege, hogy a bemeneti adatok – általában az $\mathbb{R}^n$ tér pontjai – leképezésre kerülnek egy vektortérbe, a jellemző térbe (feature space), abból a célból, hogy az eredeti térben az adatpontok közötti nemlineáris relációk lineáris relációkként jelenjenek meg a jellemző térben.



9.1. ábra. Jellemző leképezés célja

Az említett leképezést jellemző leképezésnek (feature map) nevezzük. Ezután lineáris relációk felfedezése történik a bemeneti adatok jellemző térbeli képei között, oly módon, hogy a jellemző térbeli pontok nem kerülnek felhasználásra, csak azok páronkénti belső szorzatuk. Ezek a belső szorzatok közvetlenül számolhatók a bemeneti tér pontjaiból a kernel függvények használatával. Tehát a kernel függvény, a következő $\kappa : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ leképezéssel definiálható, ahol tetszőleges $\bar{x}, \bar{y} \in \mathbb{R}^n$ esetén $\kappa(\bar{x}, \bar{y}) = \langle \Phi(\bar{x}), \Phi(\bar{y}) \rangle$ leképezés, itt a bemeneti teret $\mathbb{R}^n$, a jellemző teret $F \subseteq \mathbb{R}^N$, a közöttük lévő leképezés pedig $\Phi : \mathbb{R}^n \to F$ jelöli. A kernel függvények lehetővé teszik olyan jellemző terek használatát is, melyek dimenziója expo-

nenciálisan nagy vagy néha végtelen, megtartava a számítási hatékonysággal támasztott követelményeket. Adott $\Phi : \mathbb{R}^n \to F$ jellemző leképezés, κ kernel függvény, $S = \{\bar{x}_1, \ldots, \bar{x}_l\} \subseteq \mathbb{R}^n$ vektorhalmaz esetén a kernel mátrix az a mátrix, melynek elemei $\mathrm{G}_{ij} = \kappa(\bar{x}_i, \bar{x}_j)$, ahol $1 \leq i, j \leq l$. A belső szorzat tulajdonságai miatt a kernel mátrix szimmetrikus, pozitív szemi-definit mátrix. A [2, 3] bővebb leírást tartalmaz a témáról. A kernel-alapú módszerek működését szemlélteti a következő ábra.

Az alábbi táblázat néhány példát mutat kernel függvényekre, [1] jegyzet alapján.

név	definíció	leírás
Gauss-RBF kernel	$\kappa(\bar{x}, \bar{y}) = \exp\left(-\frac{\|\bar{x}-\bar{y}\|^2}{\sigma}\right)$, $\sigma \in \mathbb{R}^+$	forgatás, eltolás invariáns
Polinomiális kernel	$\kappa(\bar{x}, \bar{y}) = \left(\bar{x}^T\bar{y} + \sigma\right)^q$, $\sigma \in \mathbb{R}^+,\ q \in \mathbb{N}$	forgatás invariáns
Cosinus polinomiális kernel	$\kappa(\bar{x}, \bar{y}) = \left(\cos\left(\angle(\bar{x}, \bar{y})\right) + \sigma\right)^q$, $\sigma \in \mathbb{R}^+,\ q \in \mathbb{N},\ \angle(\bar{x}, \bar{y})$ az $\bar{x}$, és $\bar{y}$ által bezárt szög	forgatás, nyújtás invariáns
Inverz multikvadratikus kernel	$\kappa(\bar{x}, \bar{y}) = \frac{1}{\sqrt{\|\bar{x}-\bar{y}\|^2+\sigma}}$, $\sigma \in \mathbb{R}^+$	forgatás, eltolás invariáns
Racionális kvadratikus kernel	$\kappa(\bar{x}, \bar{y}) = 1 - \frac{\|\bar{x}-\bar{y}\|^2}{\|\bar{x}-\bar{y}\|^2+\sigma}$, $\sigma \in \mathbb{R}^+$	

Ajánlott irodalom

- **Steven Pinker:**
  - **Hogyan működik az elme?**
    Jól reprezentálja, hol is tart napjainkban a mesterséges intelligencia. Nem tankönyv, sok vitát kiváltó mű.
  - **Nyelvi ösztön**
    Kissé ironikus hangnemű, természetes nyelvekkel, nyelvészettel foglalkozó mű.
- **Daniel C. Denett:**
  - **Micsoda elmék**
  - **Darwin veszélyes ideája**
    A darwinizmust kritizálja.
  - **Az intencionalitás filozófiája**
    Az intencionalizmus egy filozófiai irányzat, mely a tárgyak megszemélyesítésére épül - „a dolgok arra valók, hogy csináljanak valamit" = intenció.
- **William H. Calvin:**
  - **A gondolkodó agy**
- **Mérő László:**
  - **Észjárások**
- **Richard Dawkins:**
  - **Az önző gén**
    Az evolúciónak ellentmond az önzetlen viselkedés, például az, hogy a méhek életük árán védik a kaptárt; ezzel a kérdéskörrel foglalkozik a mű.
  - **A hódító gén**
  - **A vak órásmester**
- **Hraskó András (Szerk.)**

- **Új matematikai mozaik**

Irodalomjegyzék

[1] András Kocsor. *Alakfelismerés statisztikus módszerei.*

[2] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis.* Cambridge University Press, 2004.

[3] *http://www.kernel-machines.org/.*