Hogyan tanul a mesterséges intelligencia?

Csallner András Erik

Juhász Gyula Felsőoktatási Kiadó
Szeged, 2024

ISBN 978-963-648-027-1 (nyomtatott)
ISBN 978-963-648-028-8 (pdf)
Kiadja:
Juhász Gyula Felsőoktatási Kiadó, Szeged, 2024

Tartalomjegyzék

Előszó

Ezen könyvecske a teljesség igénye nélkül íródott, és célja azon olvasók kíváncsiságának kielégítése, akik szeretnének kicsit betekinteni a manapság annyira felkapott mesterséges intelligencia alapvető céljaiba, történetébe, de főleg működésébe, azaz szeretnének benézni a motorháztető alá. Az írás ehhez mérten néhol nagyon könnyen érthető, másutt gondolkodásra ösztönöz, míg máshol elkerülhetetlenül felsőbb matematikai ismereteket igényel.

A mesterséges intelligencia jelenleg a legdinamikusabban fejlődő tudományág, ezért lehetetlen ilyen korlátozott terjedelemben mindenről szólni, minden felhasználását, minden módszerét akár érintőlegesen is megemlíteni. Ezért itt csak néhány, talán fontosabb, népszerűbb, illetve a megértéshez legalkalmasabb megközelítés rövid ismertetésére van lehetőség.

Akár az általános áttekintés, akár későbbi mélyebb ismeretek megszerzésére irányuló motiváció az olvasó célja, ez a könyv talán segítséget nyújthat a számára.

A mesterséges intelligencia

Az ember évezredek óta próbálja megérteni a saját gondolkodását, annak módját, módszereit. Hogyan vagyunk képesek az agyunk segítségével az agynál sokkalta nagyobb és bonyolultabb dolgok felfogására, megértésére, manipulálására? A mesterséges intelligencia kutatási területe ennél még magasabbra tör: nem csupán megérteni próbálja az értelmet, hanem kísérletet tesz annak előállítására.

A mesterséges intelligencia a tudomány egyik legújabb területe. Az ezzel kapcsolatos kutatások nem sokkal a második világháború után kezdődtek, jelenlegi elnevezése 1956-ra datálható. A sejtbiológia mellett jelenleg ez a leginkább felkapott tudományág. Más, régi, hagyományos természettudományokhoz képest sokkal több nyitott kérdést tartalmaz, ami egyrészt számtalan további lehetőséget, de kihívásokat is tartogat.

A mesterséges intelligencia számos részterületet foglal magába, az általános elméletektől – mint a tanulás és az érzékelés – kezdve egészen a konkrét feladatmegoldásokig, mint a sakk, a matematikai tételek bizonyítása, a versírás, a járművezetés vagy éppen a betegségek diagnosztizálása. Ily módon a mesterséges intelligencia a legátfogóbb intellektuális tudományág.

A mesterséges intelligencia alapjai

A mesterséges intelligencia fogalma feltételezi, hogy létezik természetes intelligencia, amit mintának tekintve annak egy mesterségesen előállított változatát hozzuk létre. Ennek legjobb példája, hogy a mesterséges intelligencia egy mérőjének tartják az Alan Turing által 1950-ben javasolt úgynevezett Turing-tesztet, amely annak eldöntésére szolgál, hogy egy mesterséges intelligencia eléri-e az emberi intelligencia szintjét. Ennek azóta több változata is kialakult, de az alapötlet azonos. Egy terminálon keresztül emberek szabadon kommunikálnak egy értelmes terminállal, és el kell dönteniük, hogy a vonal másik végén ember vagy mesterséges intelligencia található-e. A kutatók ösztönzésére 1990-ben Hugh Gene Loebner létrehozta a róla elnevezett Loebner-díjat, ami minden évben egy verseny során próbál olyan

mesterséges intelligenciát találni, amely sikerrel teljesíti a Turing-tesztet. A verseny fődíja 100 000 USD, amelyet azonban azóta sem sikerült egyetlen mesterséges intelligenciának sem elnyernie. Ehhez elvileg egy mesterséges intelligenciának négy képességgel kellene rendelkeznie, amelyek a következők: az angol nyelv ismerete és használata, tudás tárolása, automatizált indoklás előállítása a tárolt tudás alapján, valamint gépi tanulás az új körülmények feldolgozásához. További három képességet szokás még elvárni, amely azonban már túlmutat a hagyományos Turing-teszten, ezek a gépi látás tárgyak felismeréséhez, a gépi beszédértés a hallott információ megértéséhez, valamint a robotika a mozgáshoz és tárgyak mozgatásához.

Mivel alapvetően a mesterséges intelligenciától azt várjuk, hogy úgy gondolkozzon, mint egy ember, először meg kell állapítanunk, hogyan gondolkodunk mi. Ezt háromféleképpen tehetjük meg: megfigyelhetjük a gondolkodásunk szabályszerűségeit, pszichológiai tesztek segítségével rögzíthetjük az adott helyzetek által kiváltott reakcióinkat, valamint feltérképezhetjük agyunk fizikai működését. Gondolkodásunk szabályszerűségeivel és a pszichológiai tesztekre adott válaszaink kialakulásával a kognitív tudományok foglalkoznak, amelyek a megismerés módszertanát kutatják. Ezek arra is választ adnak egy adott mesterséges intelligencia esetében, hogy miben azonos és miben különbözik az emberi gondolkodástól.

A mesterséges intelligenciától azt várjuk, hogy racionális döntéseket hozzon. Ennek két eleme van: a racionális döntés és a racionális cselekvés. Racionális döntéseket elvben a formális logika törvényeit alkalmazva, könnyedén hozhatunk – amennyiben a probléma megfogalmazható formálisan. A valóságban a rendelkezésre álló információk nagy része informális, ráadásul sokszor nem is teljesen bizonyos. A racionális cselekvés elvárását úgynevezett ágensekre ruházzuk, amelyek a racionális döntések alapján a környezetükkel összhangban, önállóan működnek, és erre hosszabb időn keresztül képesek változásokon és változó célok elérésén keresztül. Vannak azonban olyan esetek, amelyekben nincs jó döntés, de cselekedni kell. Mások során nincs idő minden információt figyelembe venni – ilyenek például az embereknél a feltétlen reflexek.

A mesterséges intelligencia létrejöttét és jelenlegi fejlődését is számos tudományág ösztönzi és szolgálja. Az első ilyen a filozófia, mely aspektusból olyan kérdések merülnek fel, mint hogy honnan ered a tudás, vagy hogy azt hogyan tudjuk helyes döntések meghozatalára fordítani. Egyértelműen fontos tudományág a matematika is, amely a felmerülő kérdések formális megválaszolására törekszik. Ebben több bizonyított buktató is van. Ilyen például Kurt Gödel 1931-es nemteljességi tétele, amely arra mutat rá, hogy minden végtelen formális elméletben – mint például a természetes számok elmélete – léteznek olyan állítások, amelyeket az elméleten belül sem bizonyítani, sem cáfolni nem lehet. Egy másik buktató a problémák kezelhetősége, ami az NP-teljesség elméletéhez vezet. Számos egyszerűnek tűnő logikai kérdés NP-teljes, ami hozzávetőlegesen azt jelenti, hogy nem kezelhető, vagyis nagy valószínűséggel a megoldásához szükséges idő a probléma kiindulási adatainak számával legalább exponenciálisan nő. Egy további fontos matematikai ág a valószínűség-számítás, amely jól modellezi mind a bemenő adatok zajos, hibás természetét, mind a valóságot csak részben tükrözni képes modellek által szolgáltatott válaszok racionális tartalmának megbízhatatlanságát, illetve annak mértékét. Meg kell említenünk a gazdaságtudományokat is, amelyek a játékelmélet és azon keresztül a döntéselmélet létrejöttét sürgették. Ezek folyományaként alakult ki a döntések sorozatának eredményeként előálló megoldások kiszámítására szolgáló operációkutatás önálló tudományága. Az információ fizikai feldolgozásában nagy szerep jut az idegtudományoknak (ebben az esetben különös tekintettel a neurobiológiára), amely az emberek agyi működésén keresztül keresi a választ a gondolkodás mikéntjére. De ide kapcsolható a pszichológia is, amely a viselkedés és megismerés (kognitív pszichológia) problémakörének megértésében nyújt segítséget. A mesterséges intelligencia megvalósításában, a kézzelfogható részek kialakításában azonban a legfontosabb a számítástudomány területe, azon belül kitüntetett szerep jut az irányításelméletnek és a kibernetikának. És végül, de nem utolsósorban a mesterséges intelligencia kiteljesítésekor az emberekkel folytatandó kommunikáció és az emberek által létrehozott információhalmaz megértése érdekében fontos lehet a nyelvészet, amelyen belül a számítógépes nyelvészet ma már a legjelentősebb részterület, és amely a mesterséges intelligencia kutatások fontos részét képezi.

A mesterséges intelligencia története

Manapság Warren McCullogh és Walter Pitts 1943-as cikkét szokás a mesterséges intelligenciával foglalkozó első tudományos közleménynek tekinteni. Ebben a szerzők három elméletre alapoztak: az agy idegsejtjeinek működésére, a formális logikára, valamint a Turing-gép elvére. Egy mesterséges neurális hálózat modelljét javasolták, amelyben az egyes neuronok kétféle állapotban lehetnek, amik között a velük közvetlen kapcsolatban álló neuronok állapotától függően váltanak. Bebizonyították, hogy tetszőleges kiszámítható aritmetikai függvényhez létezik olyan mesterséges neurális hálózat, amely azt kiszámolja, továbbá minden Boole-függvény egyszerű neuronhálóval megadható. McCullogh és Pitts azt is felvetette, hogy megfelelően megadott hálózatok tanulni is képesek lehetnek. Ezt 1949-ben Donald Hebb egy egyszerű frissítési szabály bevezetésével, amely a neuronok közti kapcsolat erősségét módosítja, demonstrálta is. A róla elnevezett Hebb-féle tanulás a mai napig alapját képezi számos tanuló algoritmusnak. Az első számítógépet, amely ezt az elvet alkalmazta, Marvin Minsky és Dean Edmonds építette 1950-ben a Harvardon. Ez 3000 elektroncsövet tartalmazott, és 40 neuronból álló hálózatot szimulált.

Számos korai munka sorolható valamilyen szempontból a mesterséges intelligencia területéhez, a leginkább meghatározó azonban Alan Turing ezzel kapcsolatos víziója. Már 1947-ben előadásokat tartott ezzel kapcsolatban a London Mathematical Society-ben, elképzeléseiről egy 1950-es cikkében adott meggyőző összesítést, amelyben a Turing-tesztet, a gépi tanulást, a genetikus algoritmusokat és a felügyelt tanulás alapjait is lefektette. Ő vetette fel, hogy ahelyett, hogy egy felnőtt gondolkodását szimuláló programot kísérlünk meg alkotni, miért nem készítünk gyermeket szimuláló programot, amely aztán mindent megtanul?

A mesterséges intelligencia másik teremtő személyisége John McCarthy. 1951-ben a Princetonon doktorált, majd oktatott a Stanfordon, ezt követően a Dartmouth College-hoz igazolt, amely kezdeményezését követően a mesterséges intelligencia hivatalos születési helye lett. McCarthy meggyőzte az akkorra már elismert Minskyt, Shannont és Rochestert, hogy hozzanak létre egy amerikai kutatókból álló csapatot, akik az automaták, neurális hálózatok

és intelligenciakutatások terén dolgoznak. Így aztán 1956 nyarán összehoztak egy két hónapos workshopot Dartmouth-ban. A következő két évtized mesterséges intelligencia kutatásait ennek résztvevői és az ő tanítványaik határozták meg.

Az 1980-as évekre a szakértői rendszerek tették ki a mesterséges intelligenciával kapcsolatos kutatások túlnyomó részét. Ennek az volt az oka, hogy ezek kézzelfogható haszonnal jártak. Az első sikeres szakértői rendszert, az R1-et a DEC számítógépipari vállalat alkalmazta, amely becslések szerint 1986-ra évi 40 millió dolláros megtakarítást eredményezett a cégnek. 1988-ban több milliárd dolláros hasznot hoztak a szakértői rendszerek, a gépi látás, a robotok, illetve az erre specializálódott szoftver és hardver eszközök.

Ez a fejlődés azonban az 1990-as évekre megtört, s mivel a beígért és elvárt fejlődési ütem nem volt tartható, az ágazat hanyatlani kezdett. Ekkor azonban ismét előtérbe került a mesterséges neurális hálózatok vizsgálata. A kutatások két fő csapásirányban folytatódtak: az egyik inkább a hatékony hálózati architektúrák, algoritmusok és matematikai elvek kidolgozását, a másik a valós neurális hálózatok minél pontosabb modellezését tartja szem előtt. Egy másik bifurkáció a 2000-es évek közepétől kezdett kialakulni: míg a gyakorlati alkalmazások mindinkább a speciális feladatok egyre hatékonyabb ellátására alkalmas önvezető autók, gépi sakkjátékosok vagy beszédfelismerők felé orientálódnak, addig a mesterséges intelligencia befolyásos megalkotói, McCarthy, Minsky, Nilsson és Winston inkább az eredeti feladatot tartanák szem előtt: olyan gépet készíteni, ami úgy gondolkodik, tanul és alkot, mint egy ember.

Manapság a mesterséges intelligencia számos megnyilvánulásával találkozhatunk konkrét alkalmazásokban és eszközökben a mindennapokban, akár a háztartásokban is.

A gépi tanulás elméleti alapjai

Egy matematikai modell megalkotása során bonyolult összefüggések feltárásakor, elemzésekor hajlamosak vagyunk hibákat ejteni, ez pedig magának a problémának a megoldását is többnyire lehetetlenné teszi. A gépi tanulás segítségünkre lehet az összefüggések hatékonyabb felismerésében, ezáltal hatékonyabban működő rendszerek felépítésében, hatékonyabb gépek megépítésében. A gépi tanulás megértéséhez azonban először a tanulás fogalmát kell tisztázni. Tanulás alatt azt a folyamatot értjük, amely során a világról tett megfigyeléseink alapján javítjuk a jövőbeli döntéseink, cselekvésünk hatékonyságát. Ez az egyszerű memorizálástól a bonyolult elméletek kigondolásáig terjedhet.

A gépi tanulás esetén különböző problémák megoldását, döntések meghozatalát várjuk az adott algoritmustól. A tanuló algoritmusok elnevezés egy gyűjtőfogalom. A tanulás lehet

- felügyelt vagy
- felügyelet nélküli.

A tanulás legfontosabb, leggyakrabban alkalmazható változatát a tanulási problémák azon osztálya alkotja, amelyek felügyelt tanulással oldhatók meg. Ezekben előre megadott bemeneti-kimeneti adatpárok megtanulása révén képessé válhatunk új, ismeretlen kimenettel rendelkező bemenetekre is megjósolni a helyes kimenetet. A felügyelt gépi tanulás segítségével a gyakorlatban megoldható feladatok két alapvető osztályba sorolhatók. Ezek az osztályozás és a regresszió. Osztályozásnak akkor nevezzük a problémát, ha előre meghatározott, diszkrét kimeneti értékekhez rendelendők a bemeneti adatok. Az egyes diszkrét értékek által meghatározott bemeneti adatok csoportjai alkotják ilyenkor az osztályokat. Amennyiben a kimeneti értékek folytonos skálán mozognak, akkor vagy intervallumokra osztjuk a skálát, és osztályozást hajtunk végre a bemeneti adatokon, vagy szabályszerűséget állítunk fel a hozzárendelés megadásához. Ekkor regressziónak nevezzük a problémát.

Felügyelet nélküli tanulás során csupán egy szabályrendszer adott, az algoritmus bemeneteken gyakorolva alakít ki egy olyan eljárást, amellyel meg

tudja oldani a feladatot. Jellemzően klaszterezési eljárások tartoznak ebbe az osztályba. A klaszterezés egy olyan osztályozás, ahol az egyes osztályokat az eljárás maga alakítja ki valamilyen észszerű módon; azok előre nem adottak.

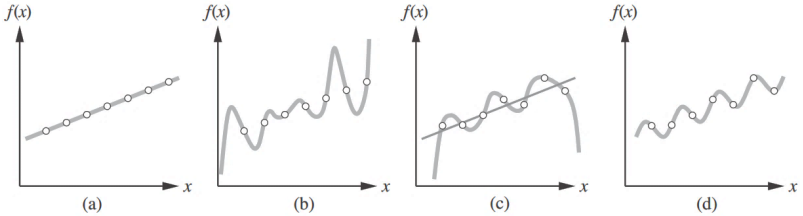
A következőkben inkább a felügyelt tanulásra szorítkozunk, körbejárjuk annak alapvető módszertanát, néhány konkrét megvalósítási lehetőségét, valamint megbízhatóságának mérési lehetőségeit.

Felügyelt tanulás

A felügyelt tanulás alapfeladata a következőképpen definiálható: Adott egy N példabemenetből és -kimenetből álló tanulóhalmaz, (x_1,y_1), (x_2,y_2),… (x_N,y_N), ahol minden y_i értéket egy ismeretlen $y=f(x)$ függvény szolgáltatja. Keressünk olyan h függvényt, amely ezt az f függvényt jól közelíti.

A fenti definícióban x és y tetszőleges értékek, nem feltétlenül csak számok lehetnek. A h függvény a hipotézis. A tanulás folyamata egy olyan hipotézis keresése a lehetséges hipotézisek terében, amely jó eredményeket szolgáltat, még olyan példákon is, amelyeket a tanuló halmaz nem tartalmaz. A hipotézis pontosságának méréséhez megadjuk példák egy teszthalmazát, aminek elemei nincsenek benne a tanuló halmazban. Azt mondjuk, hogy egy hipotézis jól általánosít, ha helyesen jósolja meg az y értékét az új példákon. Az f függvény bizonyos esetekben sztochasztikus, azaz az x-nek nem szigorú függvénye. Ilyenkor a $P(Y|x)$ feltételes valószínűségi eloszlást kell megtanuljuk.

Amennyiben az y kimenet egy véges halmaz eleme (például szövegfelismerésnél az ábécé betűi), akkor a tanulási problémát osztályozásnak hívjuk. Ha y csupán két értéket vehet fel, akkor Boole-félének vagy binárisnak nevezzük. Ha y egy valós szám (mint – mondjuk – egy tárgy távolsága), akkor a tanulási problémát regressziónak hívjuk. A regressziós probléma valójában abban áll, hogy megtaláljuk y feltételes várható értékét vagy átlagát. Annak valószínűsége ugyanis, hogy y pontos valós értékét kiszámítsuk, nulla.



1. ábra. Görbeillesztési problémák

A 1. ábrán a közismert görbeillesztési probléma megoldásai láthatók, ahol egy egyváltozós függvényt kell adatpontokra illeszteni. Az adatpontok az (x,y) síkban találhatók, ahol $y = f(x)$. Az f függvényt nem ismerjük, de megpróbáljuk közelíteni egy H hipotézistérből választott h függvénnyel, amely esetünkben legyen az egyváltozós polinomok halmaza. Az 1. ábra (a) részén az adatpontok egy egyenessel megvalósított pontos illesztése látható (a $0{,}4x+3$ elsőfokú polinommal, ami egy egyenes). Ezt az egyenest konzisztens hipotézisnek nevezzük, mivel minden adatpontra pontosan illeszkedik. Az 1. ábra (b) részén egy magasabb fokú polinom látható, amely szintén konzisztens ugyanazon adatpontokkal. Ez az induktív tanulás alapvető problémáját szemlélteti: hogyan válasszunk több konzisztens hipotézis közül? Az egyik kézenfekvő válasz, hogy válasszuk a legegyszerűbbet. Ezt az elvet Ockham borotvájának nevezik a XIV. századi angol filozófus, William of Ockham után, aki ezt az elvet alkalmazta a dolgok túlzott elbonyolításával szemben. Az egyszerűség fogalmának meghatározása persze általában nem könnyű feladat, de ebben az esetben elég nyilvánvaló, hogy egy elsőfokú polinom egyszerűbb egy hetedfokú polinomnál, így a (b) változat helyett az (a) változat választandó.

Az 1. ábra (c) rajza egy másik adathalmazt tartalmaz. Ehhez nem létezik konzisztens egyenes, sőt valójában legalább hatodfokú polinom szükséges pontos illeszkedés eléréséhez. Az adatpontok halmaza hét pontból áll; az így kapott, legalább hét paraméterrel rendelkező polinom viszont nem tűnik túl jó megoldásnak a problémában található mintázat felfedésére, ezért nem is várjuk tőle, hogy jó általánosítást adjon. A szintén az ábra (c) rajzán látható egyenes ugyan nem konzisztens az adatpontokkal, de vélhetően elég jól általánosítja az összefüggést ismeretlen, az ábrán nem látható pontokra is. Általában valami jó kompromisszumra van szükség a gyakorló adatokon konzisztens összetett hipotézisek és a valószínűleg jobban általánosító egyszerűbb hipotézisek között. A 1. ábra (d) rajzán kiterjesztettük a H hipotézisteret az x-en kívül a $\sin x$ feletti polinomokra, és az derült ki, hogy a (c) rajzon is jelölt adatpontok pontosan illeszthetők egy egyszerű, $ax+b+c\sin x$ alakú függvény segítségével. Ez a hipotézistér kiválasztásának jelentőségét jelzi. Azt mondjuk, hogy egy tanulási probléma megvalósítható, ha a hipotézistér tartalmazza a valódi f függvényt. Sajnos nem mindig lehet

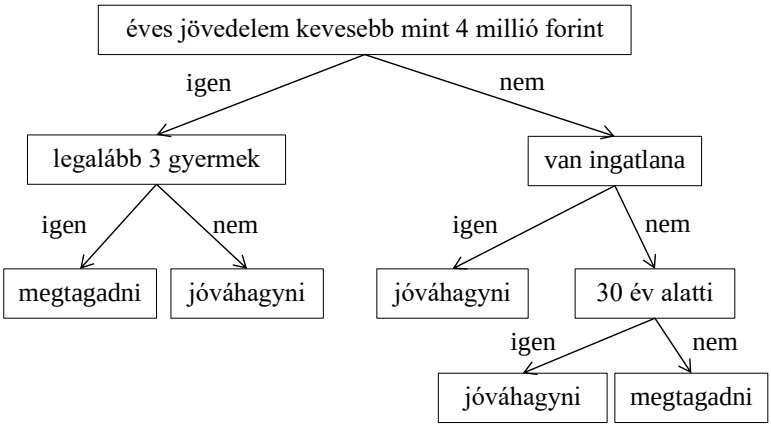
megmondani, hogy egy adott tanulási probléma megvalósítható-e, mivel a valódi f függvényt nem ismerjük.

De akkor miért ne legyen H az összes Python nyelvű program, vagy az összes Turing-gép osztálya? Végül is minden kiszámítható függvény reprezentálható Turing-géppel, ennél átfogóbb hipotézisteret nem is adhatnánk meg a tanulási probléma megvalósíthatóságának elérése érdekében. Az egyik probléma ezzel a megközelítéssel az, hogy nem veszi figyelembe a tanulás számítási bonyolultságát. Egyensúlyt kell teremteni a hipotézistér kialakításának bonyolultsága és az adott térben egy jó hipotézis kikeresésének bonyolultsága között. Például egy egyenes adatpontokra való illesztése egyszerű számítás, magasabb fokú polinom illesztése valamivel nehezebb, míg Turing-gép megadása általában a gyakorlatban kivitelezhetetlen. Egy további, az egyszerű hipotézisek mellett szóló érv, hogy vélhetően a h hipotézist, miután megtanultuk, alkalmazni is szeretnénk. Ha h lineáris függvény, akkor $h(x)$ kiszámítása bizonyosan gyors, ha azonban egy tetszőleges Turing-gépet szeretnénk futtatni, akkor még az sem biztos, hogy a programunk valaha megáll. Emiatt a gépi tanulás elmélete nagy hangsúlyt fektet az egyszerűségre. További versengés adódhat a hipotézisek kifejezőképessége és bonyolultsága között. Amennyiben az alkalmazott kifejezőeszközeink fejlettek, egyszerű hipotézisek felállítására adódhat lehetőség, de maga az eszközrendszer bonyolultabb. Ha egyszerűbb eszközrendszert alkalmazunk, akkor a hipotézisek megfogalmazása válhat bonyolulttá. Például a sakk szabályrendszere predikátumlogikai eszközökkel egyetlen oldalban megfogalmazható, míg egyszerű elsőrendű logikai nyelven megfogalmazva több ezer oldalt tehet ki.

Döntési fák

A döntési fák módszerei a gépi tanulás legegyszerűbb és leghatékonyabb eszközei. A döntési fa olyan függvényt reprezentál, amelynek bemenete egy objektum attribútumainak, értékeinek egy vektora, kimenete, azaz döntése pedig egyetlen érték, címke. A be- és kimenetek lehetnek diszkrét vagy folytonos értékek is. A döntési fa az eredményt döntések sorozatán keresztül alakítja ki. A fa minden belső pontja egy ilyen elemi döntésnek, tesztnek felel meg, amelynek bemenete egyetlen kiindulási attribútum, kimenetei pedig az adott teszt lehetséges kimenetelei. A levélcsúcsok a függvény értékét reprezentálják.

A döntési fák alapötlete, hogy bonyolult összefüggéseket egyszerű döntések sorozatára vezet vissza. Egy ismeretlen minta klasszifikálásakor a fa gyökeréből kiindulva a csomópontokban feltett kérdésekre adott válaszoknak megfelelően addig lépkedünk lefelé a fában, amíg egy levélbe nem érünk. A döntést a levél címkéje határozza meg.



2. ábra. Hitelbírálat döntési fája

Egy hipotetikus, leegyszerűsített, hitelbírálatra alkalmazható döntési fát mutat be a 2. ábra. A döntési fák nagy előnye, hogy jól felépítve őket automatikusan felismerik a lényegtelen változókat. Ha egy változóból nem nyerhető

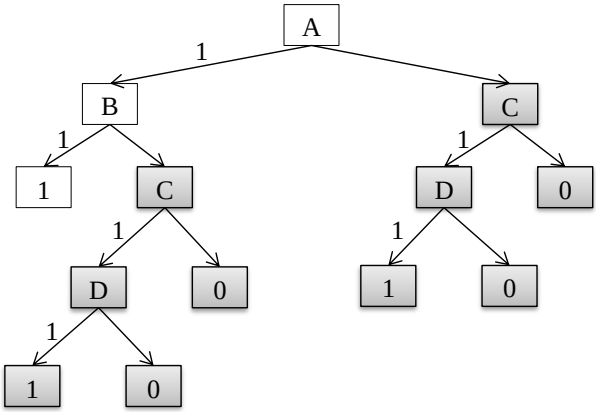
információ a magyarázott változóról, akkor azt nem is tesztelik. Ez a tulajdonság azért előnyös, mert így a fák teljesítménye zaj jelenlétében sem romlik sokat, valamint a problémamegértésünket is nagyban segíti, ha megtudjuk, hogy mely változók fontosak, és melyek nem. Általában elmondható, hogy a legfontosabb változókat a fa a gyökér közelében teszteli. További előny, hogy a döntési fák nagyméretű adathalmazokra is hatékonyan felépíthetők. A döntési fák egyik fontos tulajdonsága, hogy egy csomópontnak mennyi gyermeke lehet. Nyilvánvaló, hogy egy olyan fa, amely pontjainak kettőnél több gyermeke is lehet, mindig átrajzolható bináris fává. A legtöbb algoritmus ezért csak bináris fát tud előállítani.

A döntési fák előnyös tulajdonsága, hogy a gyökérből egy levélbe vezető út mentén a feltételeket összeolvasva könnyen értelmezhető szabályokat kapunk a döntés meghozatalára, illetve hasonlóan egy laikus számára is érthető módon azt is meg tudjuk magyarázni, hogy a fa miért pont az adott döntést hozta.

A döntési fákból nyert döntési szabályhalmazok egyértelműek. Ez nyilvánvaló, hiszen tetszőleges objektumot a fa egyértelműen besorol valamelyik levelébe. E levélhez tartozó szabályra az objektum illeszkedik, a többire nem. Vannak olyan döntési feladatok, amikor a döntési fák túl bonyolult szabályokat állítanak elő. Ezt egy példával illusztráljuk. Jelöljük a négy bináris magyarázó attribútumot *A, B, C, D*-vel! Legyen az osztályattribútum, azaz a kimenet is bináris, és jelöljük *Y*-nal! Álljon a döntési szabálysorozat három szabályból:

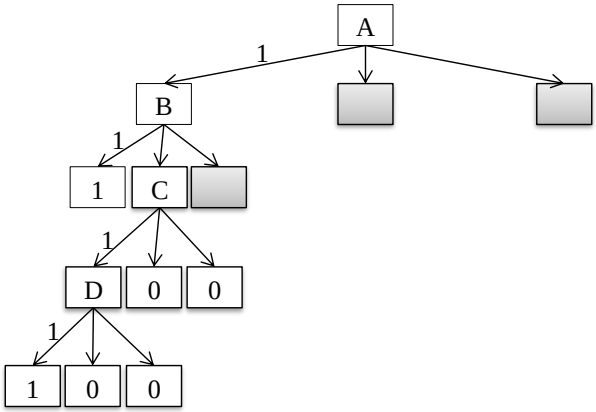
1. ha $A=1$ és $B=1$, akkor $Y=1$;
2. ha $C=1$ és $D=1$, akkor $Y=1$;
3. különben $Y=0$

A szabálysorozat teljes, hiszen az utolsó, feltétel nélküli szabályra minden további objektum illeszkedik. A fenti osztályozást a 3. ábra döntési fája adja.



3. ábra. Ismétlődő részfát tartalmazó döntési fa

A fenti példában a döntési fa az osztályozás bonyolultabb leírását adja, mint a szabálysorozat. A szürkített részfák izomorfak. A részfa által adott osztályozást egyszerűen tudjuk kezelni a döntési szabálysorozatokkal. A döntési fa esetében ugyanakkor kétszer is megjelent ugyanazon részfa. Ezt a problémát a szakirodalom ismétlődő részfa problémaként emlegeti, és a döntési fák egy alapproblémájának tekinti. A döntési fák a megoldást nagymértékben elbonyolíthatják. Az előző példában, ha az attribútumok nem binárisak, hanem három értéket vehetnek fel, akkor a megadott döntési sorozattal ekvivalens döntési fa a 4. ábrán látható. A szürkével jelölt részfa háromszor megismétlődik (az ismétlődő részfát egyetlen téglalappal helyettesítettük az áttekinthetőség érdekében). Természetesen a fa jóval egyszerűbb lenne, ha az attribútumot nem csak egy értékkel hasonlíthatnánk össze, hanem olyan tesztet is készíthetnénk, hogy az adott attribútum benne van-e egy adott értékhalmazban. Például a gyökérben csak kétfelé célszerű ágazni, attól függően, hogy $A = 1$ vagy $A \neq 1$ (másképp fogalmazva $A \in \{2, 3\}$). Ha ilyen feltételeket megengednénk, akkor – a címkéktől eltekintve – a 3. ábrán látható fával izomorf fát kapnánk.



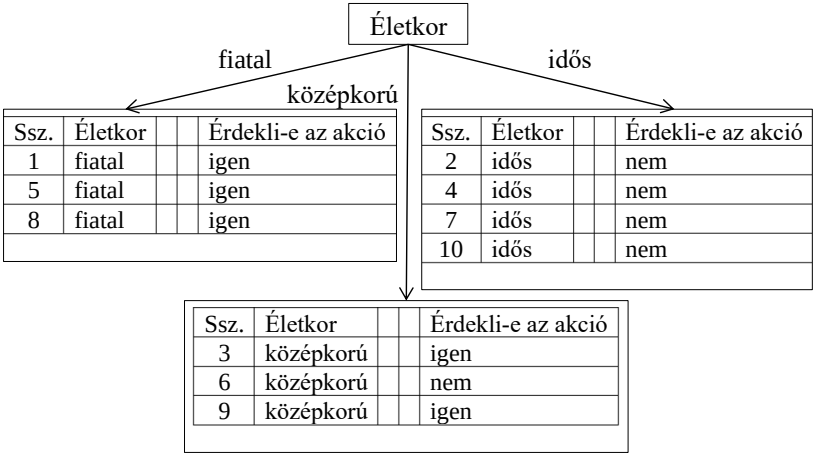
4. ábra. Trináris döntési fa

Egy döntési fát a tanító-adatbázisból rekurzívan állítunk elő. Kiindulunk a teljes tanító-adatbázisból, és egy olyan kérdést keresünk, aminek segítségével a teljes tanulóhalmaz jól szétvágható. Egy szétvágást akkor tekintünk jónak, ha a magyarázandó változó eloszlása a keletkezett részekben kevésbé szórt, kevésbé bizonytalan, mint a szétvágás előtt. Egyes algoritmusok arra is törekednek, hogy a keletkező részek körülbelül egyforma nagyok legyenek, azt remélve, hogy ezáltal a fa mélysége csökkenthető. A részekre rekurzívan alkalmazzuk a fenti eljárást. Lássunk erre egy példát! Először megadunk egy tanuló adatbázist (5. ábra).

Ssz.	Életkor	Testsúly	Sportol	Érdekli-e az akció
1	fiatal	alacsony	igen	igen
2	idős	közepes	nem	nem
3	középkorú	magas	nem	igen
4	idős	közepes	igen	nem
5	fiatal	magas	nem	igen
6	középkorú	alacsony	nem	nem
7	idős	alacsony	nem	nem
8	fiatal	közepes	nem	igen
9	középkorú	magas	igen	igen
10	idős	közepes	igen	nem

5. ábra. Tanuló adatbázis

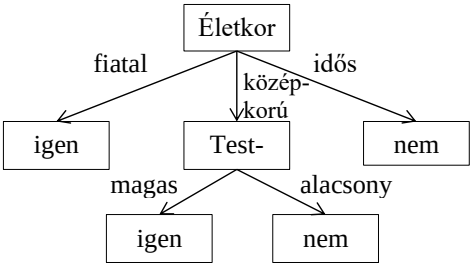
Arra vagyunk kíváncsiak, hogy ügyfelek közül kiket érdekel egy akció, tehát az utolsó oszlop az osztályattribútum, azaz a kimenet. A teljes adatbázist tekintve, az életkor alapján lehet leginkább szétválasztani az ügyfeleket aszerint, hogy érdekli-e őket az akció. Ezért a döntési fa gyökerében az életkorra vonatkozó kérdést tesszük fel. A három ágon a tanítóadatok különböző részhalmazai láthatóak az életkor különböző értékeinek megfelelően.



6. ábra. Döntési fa gyökere

Mindhárom ágon rekurzívan alkalmazzuk az eljárást az adott ághoz tartozó tanító-adatbázisra. A két szélső ágon minden példány egyazon osztályba tartozik, ezért egy-egy levél csomópontot hozunk létre. A középső ágon a testsúly szerint vágunk (7. ábra).

A példa illusztratív. A valóságban gyakran előírjuk, hogy olyan leveleket hozzunk létre, hogy minden levélhez legalább egy előre meghatározott mennyiségű adatbázisbeli objektum (példány) tartozzon, és előfordulhat, hogy akkor is létrehozunk egy levelet, ha az adott ághoz tartozó nem minden objektum tartozik ugyanazon osztályba, csak a nagy többségük.



7. ábra. A kész döntési fa

A következőkben részletezzük, hogy miként lehet kiszámolni azt, hogy melyik kérdést tegye fel a fa a különböző csomópontokban. Néhány döntési fát előállító algoritmus egy csomópont leszármazottjaiban nem vizsgálja többé azt az attribútumot, ami alapján szétosztjuk a mintát. Más algoritmusok megengedik, hogy például egy A numerikus attribútum esetében először $A > x$ kérdést tegye fel a döntési fa, majd ezen csomópont valamelyik leszármazottjában az $A > y$ kérdés szerepeljen. A rekurziót akkor szakítjuk meg valamelyik ágban, ha a következő feltételek közül teljesül valamelyik.

- Nincs több attribútum, ami alapján az elemeket tovább oszthatnánk. A csomóponthoz tartozó osztály ekkor az lesz, amelyikhez a legtöbb tanítópont tartozik.
- A fa mélysége elért egy előre megadott korlátot.
- Nincs olyan vágás, amely javítani tudna az aktuális osztályzáson.

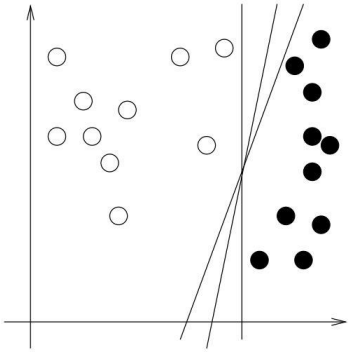
Minden levélhez hozzá kell rendelnünk a magyarázandó változó egy értékét, a döntést. Ez általában az úgynevezett többségi szavazás elve alapján történik: az lesz a döntés, amely kategóriába a legtöbb tanítóminta tartozik. Hasonló módon belső csomópontokhoz is rendelhetünk döntést. Ez akkor hasznos, ha olyan döntési fát kívánunk készíteni, amely nagyon gyorsan képes egy (nem feltétlenül optimális) döntést hozni. Ha nagyon kevés idő áll rendelkezésre egy-egy döntéshez, akkor csak az első néhány kérdést tesszük fel. Ha viszont bőségesen áll rendelkezésünkre idő, akkor feltesszük a további kérdéseket is. Az angol irodalomban ezt a technikát *anytime decision tree*-nek nevezik, az olyan osztályozókat pedig, amelyek képesek nagyon rövid idő alatt egy – az optimálistól akár jelentősen eltérő – döntést hozni, majd ezt a döntést a rendelkezésre álló idő függvényében folyamatosan pontosítani, *anytime classificator*oknak hívják.

Lineárisan szeparálható osztályok

Két osztály lineárisan szeparálható, ha egy hipersík segítségével el tudjuk különíteni a két osztály pontjait. Amennyiben a pontok *n* dimenziósak, akkor $n-1$ dimenziós hipersíkot kell meghatároznunk. Egy $x = (x_1, x_2, \dots, x_n)$ pont osztályozásához az

$$f_0(x) = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$$

függvényt hívjuk segítségül. Ha $f_0(x) > 0$, akkor az *x*-et az első osztályba soroljuk, egyébként a másodikba.



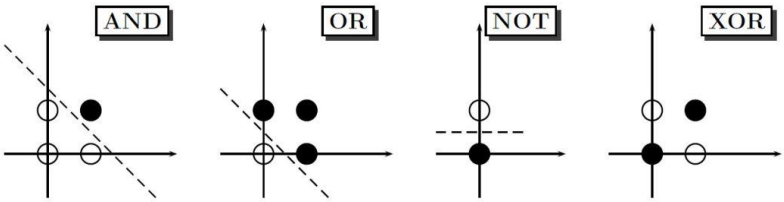
8. ábra. Példa lineárisan szeparálható osztályokra

Az osztályozó algoritmus tanulási fázisa a w_i súlyok meghatározásából áll. Új elemek osztályozásához meghatározzuk az új elem attribútumainak w_i értékekkel történő súlyozott összegét. Ha az összeg pozitív, akkor az első osztályba tartozik, ellenkező esetben a másodikba. Lineárisan szeparálható osztályokra láthatunk példát a 8. ábrán. Látható, hogy adott tanítóhalmazhoz több szeparáló hipersík is létezhet. A példában a pontok kétdimenziósak, így a hipersíkok egyenesek, azaz egydimenziós síkok.

Mennyire erős az a megkötés, hogy az osztályok lineárisan szeparálhatók legyenek? Ha az objektumok nem szeparálhatók lineárisan, akkor új attribútumok bevezetésével – amelyek az eredeti attribútumok nemlineáris transzformáltjai – olyan térbe kerülhetünk, amelyben már lehet lineáris szeparálást

végezni. Kérdés azonban, hogy ehhez az eredeti attribútumainkat konkrétan hogyan transzformáljuk egy adott feladat esetében.

Tekintsük azt az esetet, hogy minden attribútum bináris, azaz 0 vagy 1 értékeket vehet fel. Miként a 9. ábra mutatja, az AND, OR, NOT függvények lineárisan szeparálható osztályokat hoznak létre.



9. ábra. AND, OR, NOT logikai függvények tanulása, XOR függvény

Sajnos ugyanez nem mondható el az XOR függvényre. Tehát már egy ilyen egyszerű logikai függvényt, mint az XOR, sem tud megtanulni egy lineáris osztályozó. A neurális hálózatoknál vissza fogunk térni az XOR kérdéséhez. Látni fogjuk, hogy a neurális hálózatok tetszőleges logikai függvényt képesek megtanulni. Előbb azonban vizsgáljuk meg a perceptron tanulási szabályt.

A Perceptron tanulási szabály

A következőkben feltételezzük, hogy az összes attribútum valós. Az attribútumok számát az eddigiekhez hasonlóan *n*-nel jelölve, a hipersík dimenziója is *n* lesz, ugyanis felveszünk egy extra attribútumot, melynek értéke minden pontnál konstans 1 lesz. Ezt az extra attribútumot az angol irodalomban biasnak hívják. Így az osztályozáshoz használt egyenletet

$$f_0(x) = w_0x_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n = x^{\mathrm{T}}w$$

alakban írhatjuk, ahol $x_0 = 1$, így w_0 a korábbi formalizmusbeli b konstans tagot helyettesíti, $w = (w_0, w_1, \dots, w_n)$ és $x = (x_0, x_1, \dots, x_n)$ pedig $n + 1$ dimenziós vektorok.

Jelöljük a tanulópontok halmazát T-vel. Ekkor a Perceptron algoritmus a következő:

PERCEPTRON (T)
$w \leftarrow (0,0,\dots,0)$
while létezik rosszul osztályozott $t \in T$
    do for all $t \in T$
        do if t rosszul van osztályozva
            then if t az első osztályba tartozik
                then $w \leftarrow w + t$
                else $w \leftarrow w - t$
return

Kezdetben az összes $w_0, w_1, \dots, w_n$ súly értéke 0. Az algoritmus egyenként végighalad a tanítópontokon, akár többször is, ameddig van rosszul osztályozott tanítópont. Amennyiben eközben rosszul osztályozott ponttal találkozunk, úgy módosítjuk a hipersíkot, hogy a rosszul osztályozott tanítópont közelebb kerüljön hozzá, sőt akár át is kerülhet a sík másik oldalára. Ha egy t tanítópont a valóságban az első osztályba tartozik, de az aktuális súlyok alapján az algoritmus rosszul osztályozza, azaz $f_0(t) < 0$, akkor a w súlyvektort $(w + t)$-re változtatjuk. Ennek hatására $f_0(t)$, azaz a t az attribútumértékeinek súlyozott összege, $\sum w_i t_i$-ről $\sum (w_i + t_i) t_i$-re változik. Az $f_0(t)$ új értéke biztosan nem kisebb, mint a régi, hiszen

$$f_0^{\text{új}}(t) = \sum w_i t_i + \sum t_i^2 = f_0^{\text{régi}}(t) + \sum t_i^2 \geq f_0^{\text{régi}}(t).$$

Hasonlóképpen, ha egy rosszul osztályozott tanítópont a második osztályba tartozik, szintén úgy módosítjuk a súlyokat, hogy a helytelenül osztályozott pont vagy átkerüljön a hipersík túloldalára, vagy legalább közelebb kerüljön a hipersíkhoz.

A hipersík módosításai egymással ellentétesek is lehetnek, de szerencsére biztosak lehetünk benne, hogy a sok módosításnak előbb-utóbb vége lesz, ugyanis bizonyítható, hogy a Perceptron tanulási algoritmus véges lépésen belül leáll, feltéve, hogy az osztályok lineárisan szeparálhatók. Hátrány, hogy ha a tanítópontok nem szeparálhatóak lineárisan, akkor az algoritmus

ezt nem veszi észre, és nem áll le. A gyakorlatban ezért egy maximális iterációszámot szokás megadni, amelynek elérésekor sikertelen üzenettel leáll az algoritmus.

Gyakran az eddig ismertetett Perceptron algoritmus egy módosított változatát használják, ahol $w \leftarrow w + t$ és $w \leftarrow w - t$ helyett $w \leftarrow w + \varepsilon$, illetve $w \leftarrow w - \varepsilon$ lépésekben módosítják a w súlyvektort [28], ahol ε általában egy kicsi pozitív szám (pl. $\varepsilon = 0{,}01$). Az ε-t tanulási rátának hívják. Megjegyezzük továbbá, hogy a Perceptron algoritmus módosított változata használható a lineáris regresszióhoz, ahogyan azt a későbbiekben látni fogjuk.

Lineáris regresszió

A lineáris regresszió számos módszer alapja. A lineáris regresszió abban az esetben használható, ha minden attribútum valós típusú, beleértve a magyarázandó attribútumot (osztályváltozót) is.

Feltételezzük, hogy az $X_1, \dots, X_n$ magyarázó változók és a magyarázandó Y változó között lineáris kapcsolat áll fenn. Úgy tekintjük tehát, hogy egy $x = (x_1, \dots, x_n)$ objektumhoz tartozó magyarázott változó *y*-nal jelölt értéke az

$$\hat{y} = w_0 + \sum_{j=1}^{n} x_j w_j$$

összefüggéssel becsülhető, ahol $\hat{y}$ az *y* becslését jelöli. Ha a korábbiakhoz hasonlóan felveszünk egy extra változót, X_0-t, melynek értéke minden pontra $x_0 = 1$, akkor a vektoros felírást használva tovább egyszerűsíthetjük a képletet:

$$\hat{y} = x^{\mathrm{T}} w.$$

A *w* vektort kell meghatároznunk úgy, hogy adott tanítópontok mellett a négyzetes hibaösszeg minimális legyen. Az adatbázis *i*-dik tanítópontját (pontosabban annak magyarázó attribútumainak értékeiből álló vektort) x^i-vel, az *i*-dik tanítóponthoz tartozó magyarázandó változó értékét y^i-vel jelöljük. A minimalizálandó négyzetes hibaösszeget tehát így írhatjuk fel:

$$E(w|T) = \sum_{i=1}^{|T|} \left(y^i - \left(x^i\right)^{\mathrm{T}} w \right)^2.$$

Adott tanítóhalmaz esetén E a *w* függvénye. Az $E(w|T)$ függvény *w*-ben négyzetes, így minimuma mindig létezik és egyértelmű. Amennyiben a tanítópontokat egy $|T| \times n$-es *X* mátrixszal ábrázoljuk (egy tanítópontnak a mátrix egy sora felel meg), a magyarázandó változó tanítópontokhoz tartozó értékeit pedig az *y* vektorral jelöljük, akkor a fenti függvényt átírhatjuk más formába:

$$E(w|T) = (y - Xw)^{\mathrm{T}}(y - Xw).$$

Ennek *w* szerinti deriváltja

$$-2X^{\mathrm{T}}y + 2X^{\mathrm{T}}Xw.$$

Ha a deriváltat egyenlővé tesszük nullával, akkor egyszerűsítés után a következőhöz jutunk:

$$X^{\mathrm{T}}(y - Xw) = 0,$$

amelyből nemszingularitást feltételezve kapjuk, hogy

$$\hat{w} = \left(X^{\mathrm{T}}X\right)^{-1}X^{\mathrm{T}}y.$$

A gyakorlatban a mátrix invertálása nagy mátrixok esetén nem mindig célszerű, mert túl sok ideig tarthat, ezért a *w* súlyok meghatározásához a Perceptron algoritmus egy változatát szokták használni. Ekkor az iteratív algoritmusban a *w*-t a nem eredeti $w \leftarrow w + t$, illetve $w \leftarrow w - t$ lépésekben módosítják, hanem a

$$w \leftarrow w + \varepsilon\left(y^t - t^{\mathrm{T}}w\right)w$$

lépésben, ahol *t* egy tanítópont, azaz a *T* tanító-adatbázis egy objektuma, y^t a (numerikus) magyarázandó változó értéke a *t* objektum esetén, $t^{\mathrm{T}}w$ a magyarázó változó aktuális $w = (w_0, w_1, \ldots, w_n)$ súlyok által becsült értéke, ε pedig a tanulási együttható. A regresszió esetében a magyarázott változó folytonos, így az algoritmusban nem ágazunk el aszerint, hogy első osztálybeli pontot soroltunk-e a második osztályba, vagy fordítva, hanem egyszerűen a fenti képletet használjuk, amikor a magyarázott változó aktuálisan becsült értéke különbözik annak valós értékétől, y^t-től. Az *w*-vel való szorzásnak köszönhetően, mivel *w* komponensei pozitívak és negatívak is lehetnek, épp a jó irányba mozdítjuk el a hipersíkot.

Vegyük észre, hogy az előbbi algoritmus valójában az $E(w|T)$ egy lokális optimumát keresi adott *T* tanító-adatbázis mellett. Mivel az $E(w|T)$ hibafüggvény konvex, és optimumhelye egyértelmű, mivel egyetlen lokális optimummal rendelkezik, amely egyben a globális optimum is, ezért az algoritmus megfelelően megválasztott maximális iterációszám és ε mellett a

hibafüggvény optimumához tartozó $w_0, w_1, \dots, w_n$ súlyokat találja meg jó közelítéssel.

Lineáris regresszióra visszavezethető számos nemlineáris kapcsolat is. Például az $y = ax_1^b$ alakú összefüggés esetén y lineárisan függ x_1^b-től. Előfeldolgozási lépésként tehát x_1-et a b-dik hatványra emeljük, azaz az adatbázis összes objektuma esetében lecseréljük x_1 értékét x_1^b-re, és ezt követően használhatjuk a lineáris regressziót.

Logisztikus regresszió

Ha a lineáris regressziót osztályozásra akarjuk használni (de a magyarázó változók továbbra is valós számok), akkor az egyes osztályoknak egy-egy valós számot kell megfeleltetnünk. Bináris osztályozásnál a 0-t és az 1-t szokás használni. Ezzel azonban nem oldottuk meg a problémát. A lineáris regresszió egy tanítópont osztályozásánál egy számot fog előállítani, és a hibát a tanítópontnak ettől a számtól vett különbségével definiálja. Tehát 1-es típusú tanítópont esetén ugyanakkora lesz a hiba 0 és 2 kimenetek esetén, ami nem túl jó.

Egy x vektorral leírt pont osztályának felismerésénél meg kell határoznunk az $x^{\mathrm{T}}w$ értéket. Amennyiben ez nagyobb, mint 0,5, akkor az 1-eshez tartozó osztály a felismerés eredménye, ellenkező esetben pedig a 0-hoz tartozó osztály. Az egyszerűség kedvéért jelöljük az $x^{\mathrm{T}}w - 0{,}5$ értéket $\hat{y}$-pal. A jelölés további egyszerűsítése érdekében a korábbiakhoz hasonlóan most is vezessünk be egy X_0 extra attribútumot, amelynek értéke az összes adatbázisbeli objektumra $x_0 = 1$, így $w_0 = -0{,}5$ választása mellett $\hat{y}$ az $x^{\mathrm{T}}w$ szorzatot jelöli.

Az a függvény, amely nullánál kisebb értékekre 0-át ad, nagyobbakra pedig 1-et, eléggé hasonlít az előjel (szignum) függvényre. Ha megengedjük, hogy értelmetlen eredményt kapjunk $\hat{y} = 0$ esetében – amelyet értelmezhetünk úgy, hogy az osztályozó nem képes dönteni –, akkor az osztályozó által előre jelzett osztályt megkaphatjuk az

$$\frac{1 + \operatorname{sgn}(\hat{y})}{2}$$

kiszámításával.

Ha így definiáljuk a kimenetet, akkor a hiba definíciója is megváltozott, és az előző fejezetben látott lineáris regresszió nem használható a w vektor meghatározásához. Kérdés, hogy tudunk-e árnyaltabb kimenetet adni, mint pusztán egy osztályindikátor (0 vagy 1)?

Minél közelebb vagyunk az osztályok határához, annál bizonytalanabbak vagyunk a döntést illetően. Tehát természetes, hogy az osztályok előrejelzése helyett az osztály előfordulásának esélyét becsüljük. Ehhez csak annyit kell tennünk, hogy az előbbi, az előjel függvény segítségével definiált függvényt lesimítjuk, azaz egy olyan $f(\hat{y})$ függvénnyel helyettesítjük, amely

1. értéke 1-hez közelít, ha $\hat{y}$ tart végtelenhez;
2. értéke 0-hoz közelít, ha $\hat{y}$ tart mínusz végtelenhez;
3. $f(0) = 0{,}5$;
4. szimmetrikus nullára nézve, tehát $f(\hat{y}) + f(-\hat{y}) = 1 = 2f(0)$;
5. differenciálható minden pontban;
6. monoton növekvő.

Az ilyen függvényeket nevezzük szigmoid függvényeknek.

Sok függvény teljesíti a fenti elvárásokat. Könnyű belátni, hogy az

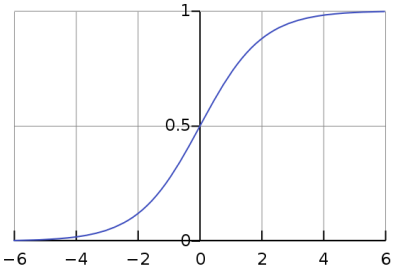
$$f_a(\hat{y}) = \frac{1}{1 + a^{-\hat{y}}}$$

alakú függvények minden $a > 1$ esetben megfelelnek az elvárásoknak. Amennyiben $a = e$, akkor az ún. logisztikus függvényt kapjuk (lásd 10. ábra).

$$f_e(\hat{y}) = \mathrm{P}(Y = 1|X) = \frac{1}{1 + e^{-\hat{y}}}.$$

A logisztikus függvény inverzét, az $\ln\frac{x}{1-x}$ függvényt logit függvénynek hívjuk. A logisztikus függvény szépsége, hogy a deriváltja $f(\hat{y})(1 - f(\hat{y}))$, amely a mi esetünkben $\mathrm{P}(Y = 1|X)\mathrm{P}(Y = 0|X)$-el egyezik meg.

A fenti feltételeket további függvények is teljesítik. Valószínűségi változók eloszlásfüggvénye is mínusz végtelenben nullához, plusz végtelenben pedig egyhez tart. A harmadik és negyedik feltétel ($f(0) = 0{,}5$ és $f(\hat{y}) + f(-\hat{y}) = 2f(0)$) megkívánja, hogy a sűrűségfüggvény szimmetrikus legyen, azaz az $f'(x) = f'(-x)$ teljesüljön minden x valós számra. A nulla várható értékű normális eloszlás eloszlásfüggvénye megfelel a feltételeknek.



10. ábra. A logisztikus függvény

Ezzel el is jutottunk a logisztikus regresszió feladatához. Szemben a lineáris regresszióval, lineáris kapcsolat nem az *X* és a magyarázandó *Y* változók között van, hanem $\ln \frac{\mathrm{P}(Y=1|X)}{1-\mathrm{P}(Y=1|X)}$ és $x^{\mathrm{T}}w$ között, tehát

$$\mathrm{P}(Y=1|X=x) = \frac{1}{1+e^{-x^{\mathrm{T}}w}},$$

$$\mathrm{P}(Y=0|X=x) = 1-\mathrm{P}(Y=1|X=x) = \frac{e^{-x^{\mathrm{T}}w}}{1+e^{-x^{\mathrm{T}}w}}.$$

A következőkben azzal foglalkozunk, hogyan határozzuk meg a *w* azon értékét, amelyik a legkisebb hibát adja. A *w* ilyen értékét jelöljük w^*-gal!

Sajnos a w^* meghatározására nincs olyan szép zárt képlet, mint ahogy a lineáris regresszió esetében volt. Iteratív, közelítő módszert használhatunk, amely a gradiens módszeren alapul. A hiba minimalizálása helyett a feltételes valószínűségeket maximalizáljuk:

$$w^* \leftarrow \mathrm{argmax}_w \sum_{i=1}^{|T|} \ln \mathrm{P}\left(Y=y^i|X=x^i,w\right).$$

A fenti képletben a regressziós függvény a szokásos $\mathrm{P}\left(Y=y^i|X=x^i\right)$ helyett $\mathrm{P}\left(Y=y^i|X=x^i,w\right)$, hiszen *w* most nem mint konstans szerepel, hanem mint változó: azon *w*-t keressük, amelyre $\sum_{i=1}^{|T|} \ln \mathrm{P}\left(Y=y^i|X=x^i\right)$ értéke maximális.

Felhasználva, hogy az *y* csak 0 vagy 1 értéket vehet fel, a maximálandó függvényt átírhatjuk:

$$l(w) = \sum_{i=1}^{|T|} \big(y^i \ln \mathrm{P}(Y = 1 | X = x^i, w) + (1 - y^i) \ln \mathrm{P}(Y = 0 | X = x^i, w) \big).$$

Az iteratív algoritmus során kiindulunk valamilyen szabadon megválasztott $w^{(0)}$ vektorból, majd a *k*-adik lépésben a $w^{(k-1)}$ vektorhoz hozzáadjuk a $\frac{\partial l(w)}{\partial w}$ vektor λ-szorosát, így megkapjuk a $w^{(k)}$ vektort. A λ egy előre megadott konstans, amelynek értékét tipikusan kicsire, például 0,01-ra szokták állítani. A $\frac{\partial l(w)}{\partial w}$ vektor a $\frac{\partial l(w)}{\partial w_j}$ parciális deriváltakból áll:

$$\frac{\partial l(w)}{\partial w_j} = \sum_{i=1}^{|T|} x_j^i \big(y^i - \mathrm{P}(Y = 1 | X = x^i, w) \big),$$

ahol $\mathrm{P}(Y = 1 | X = x^i, w)$ a logisztikus regresszió által adott becslés. Az $y^i - \mathrm{P}(Y = 1 | X = x^i, w)$ tag a hibát jelenti, amely meg van szorozva egy nagyság jellegű tényezővel: az x_j^i érték adja meg a becslésben a w_j szerepének nagyságát.

Az $l(w)$ konkáv, ezért a gradiens módszer a globális maximumhoz fog konvergálni. A gyakorlat azt mutatja, hogy a konvergencia igen gyors, a w_j értékek néhány iteráció után már alig-alig változnak.

A lineáris regresszióról szóló korábbi fejezetben említettük, hogy a lineáris regresszióhoz tartozó *w* súlyvektort a Perceptron algoritmus módosított változatával is kiszámolhatjuk. Vegyük észre, hogy ha a most említett gradiens módszert alkalmazzuk az optimális *w* súlyvektor megtalálására, minimális különbségtől eltekintve ugyanarra az algoritmusra jutunk, mint lineáris regresszió esetén. Ez a minimális különbség abban áll, hogy míg a Perceptron algoritmus módosított változata egyesével tekinti a tanító-adatbázis pontjait (illetve a magyarázott változó aktuális *w* vektor melletti becslésekor

elkövetett hibát), addig a fenti eljárásban a gradiens számításakor az összes tanítópontot tekintjük (összesített hibával dolgozunk).

Logisztikus regresszió általános osztályozásnál, one-versus-all technika

Az eddigiekben bináris osztályozással foglalkoztunk. Mi a teendő, ha a magyarázandó attribútum diszkrét, és *k*-féle különböző értéket vehet fel, ahol $k > 2$?

A többválaszú logisztikus regresszió (*multiresponse/multinomial logistic regression*) *k* darab osztály esetén *k*-szor alkalmazza a logisztikus regressziót. Veszi az első osztályt, és a többit egy kalap alá vonva végrehajt egy logisztikus regressziót, mely ad egy valószínűséget. Ezután a második osztályt emeli ki, és az összes többi osztályt vonja egy kalap alá. Így az összes osztályhoz meg tud határozni egy valószínűséget (bizonyosságot), mintha csak egy tagsági függvényt próbálna meghatározni.

Új objektum osztályozásánál az osztályozó arra az osztályra teszi le a voksát, amelyik a legnagyobb valószínűséget kapta. Ha csak a felismert (előre jelzett) osztály érdekel minket, és a kapott valószínűségre nem vagyunk kíváncsiak, akkor az új objektum osztályozása során nem szükséges a logisztikus függvény alkalmazása, mivel a logisztikus függvény monoton növekvő. Ezt az eljárást *one-versus-all* technikának hívják az angol irodalomban [23]. A one-versus-all eljárás nem csak logisztikus regresszió esetén alkalmazható. Segítségével többosztályos osztályozási feladatokat oldhatunk meg bármely olyan bináris osztályozó algoritmussal, amely képes arra, hogy az egyik, illetve másik osztályba tartozás bizonyosságát jellemző folytonos kimenetet adjon.

A one-versus-all technika logisztikus regresszióra történő alkalmazásakor a kapott P értékek összege nem feltétlenül 1, mivel az osztályozók függetlenek. Ezen a következőképpen segíthetünk: a fenti módszer helyett csak $k - 1$ darab w^l vektort állítsunk elő úgy, hogy minden $l = 1, \dots, k - 1$ értékre

$$\widehat{\mathrm{P}}(Y = l|X = x) = \frac{e^{-x^{\mathrm{T}}w^l}}{1 + \sum_{i=1}^{k-1} e^{-x^{\mathrm{T}}w^i}},$$

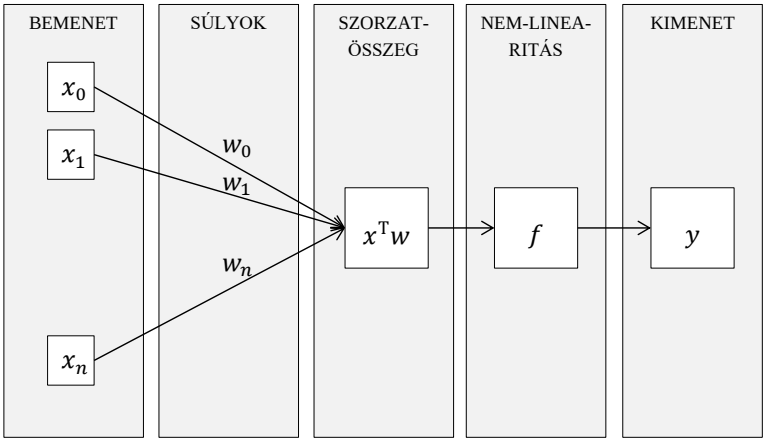
valamint

$$\widehat{\mathrm{P}}(Y = k|X = x) = \frac{1}{1 + \sum_{i=1}^{k-1} e^{-x^{\mathrm{T}}w^i}}.$$

A gradiens módszernél alkalmazott vektor – amely λ-szorosát hozzá kell adni az aktuális $w^{(l)}$ $(1 \leq l \leq k-1)$ vektorhoz – a következő komponensekből áll:

$$\frac{\partial l(w^1, \ldots, w^{k-1})}{\partial w_j^l} = \sum_{i=1}^{|T|} x_j^i \left(\delta\left(y^i = l\right) - \widehat{\mathrm{P}}\left(Y = y^l \middle| X = x^i, w^l\right)\right),$$

ahol $\delta\left(y^i = l\right) = 1$, ha az *i*-edik tanítópont osztálya *l*, különben 0.

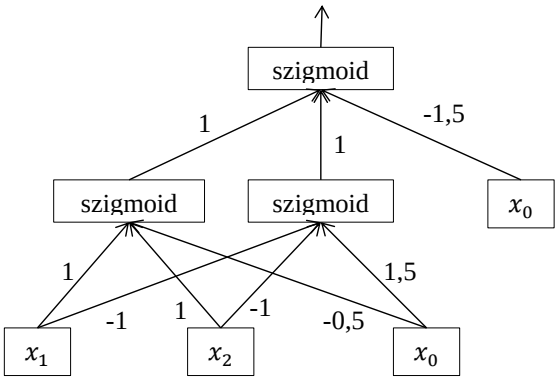


11. ábra. Logisztikus regresszió sémája

Mesterséges neurális hálózatok

A logisztikus regresszió 11. ábrán látható modelljét egyrétegű mesterséges neurális hálózatnak is nevezik. Ez az alapja a komolyabb mesterséges neurális hálózatoknak, melyek – némileg az agyműködést utánzó biológiai analógiára is támaszkodva – logisztikus regressziók kapcsolatai.

A neurális hálók kifejezőereje nagyobb a lineáris modellekénél, melyet az alábbi példán szemléltetünk. Amint azt a 24. oldalon a 9. ábra kapcsán láttuk, a lineáris szeparációt végző modellek nem képesek megtanulni az XOR függvényt. Három logisztikus regresszió felhasználásával azonban az XOR-t is ki tudjuk fejezni. Idézzük fel, hogy a példában az AND függvényt az $x_1 + x_2 - 1{,}5 = 0$, az OR függvényt az $x_1 + x_2 - 0{,}5 = 0$ egyenes, a NOT-ot pedig az $x_2 - 0{,}5 = 0$ egyenesek szeparálják. Az XOR függvény pedig felírható, mint $(x_1 \text{ OR } x_2) \text{ AND } \big(\text{NOT}(x_1 \text{ AND } x_2)\big)$. A 12. ábra ezt a konstrukciót mutatja. Az ábrabeli felső szignum függvényhez tartozó logisztikus regresszió az AND-et, a bal alsó az OR-t, a jobb alsó pedig a NAND-et adja vissza.

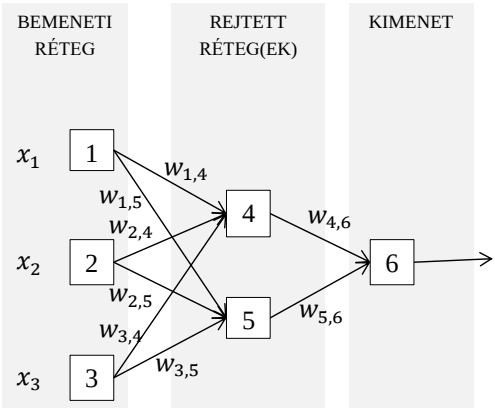


12. ábra. Az XOR függvény logisztikus regressziók összekapcsolásával

Az építőelemeket ismerve tetszőleges logikai formulát kifejezhetünk logisztikus regressziók összekapcsolásával, ezért a logisztikus regressziók kapcsolata univerzális függvényapproximátor.

A legnépszerűbb modell a többrétegű előrecsatolt neuronhálózat (lásd 13. ábra). Az első réteg csomópontjai a bemeneti neuronok, melyek a magyarázó változóknak felelnek meg (1, 2, 3-mal címkézett neuronok az ábrán). A kimenetet (magyarázott változókat) a legutolsó réteg kimenete (6. neuron) adja. A közbülső rétegeket rejtett (*hidden*) rétegeknek (4-5. neuronok) nevezzük. Minden réteg minden neuronjának kimenete a következő réteg összes neuronjának bemenetével kapcsolatban áll. A kapcsolat szorosságát a $w_{i,j}$ súlyok jellemzik.

A bemeneti réteg neuronjai (1-3. neuronok az ábrán) nem végeznek műveleteket, csupán a magyarázó változókat reprezentálják. A 13. ábrán látható 4-6. neuronok egyenként a 11. ábrán szereplő logisztikus regressziót végzik.



13. ábra. Többrétegű előrecsatolt neurális hálózat

Mind a logisztikus regresszió, mind a neurális hálózatok nemlineáris függvényapproximátornak tekinthetők. A tapasztalatok és az elméleti eredmények (lásd [12]) szerint is ugyanannyi paramétert (súlyt) használva nemlineárisan paraméterezett függvényekkel gyakran jobb közelítést érhetünk el, mint lineárisan paraméterezett társaikkal.

Az alkalmas súlyokat nemlineáris optimalizációs technikával, gradiens módszerrel kereshetjük meg gyakorlatilag ugyanúgy, mint a logisztikus regressziónál tettük.

A többrétegű előrecsatolt neuronhálózatok esetében az előrecsatolt topológiának köszönhetően az egész neuronháló hibafüggvényének *w* súlyok szerinti gradiensét könnyen kiszámíthatjuk. A súlyok megtalálása a tanítópéldák alapján az ún. *backpropagation* (hiba visszaterjesztés) eljárás szerint zajlik [28]. A Perceptron algoritmushoz hasonlóan egyesével végigmegyünk (akár többször is) a *T* tanító-adatbázis *t* objektumain. Ennek során

1. az inputokból előrehaladva kiszámítjuk az egyes neuronok outputjait;
2. az utolsó output rétegből kiindulva, rétegről rétegre visszafelé haladva a megfelelő backpropagation szabály szerint módosítjuk a $w_{i,j}$ értékeket úgy, hogy a módosítás hatására a *t*-re vonatkozó hiba csökkenjen.

Vegyük észre, hogy a Perceptron algoritmus a backpropagation algoritmus azon egyszerű esete, amikor a neurális háló egyetlen rejtett réteggel sem rendelkezik.

Mivel a neuronháló által reprezentált függvénynek lehetnek lokális maximumai, ezért a módszer nem biztos, hogy a globális optimumot adja. A backpropagation eljárást ezért többször szokás futtatni különböző kezdeti paraméterekkel.

A neuronhálók hátránya, hogy a súlyok rendszere közvetlenül nem értelmezhető emberek számára: legtöbbször nem tudjuk szemléletesen indokolni, hogy mi alapján hozta meg a neuronháló a döntést. A neuronháló tehát egy fekete doboz a felhasználó szemszögéből. A magyarázó értelmezés hiánya sok területen elfogadható (gondoljunk példaként arra, amikor azt szeretnénk meghatározni, hogy milyen termékeket reklámozzunk egy felhasználónak, amikor következő alkalommal belép egy webes áruházba). Más esetekben azonban a magyarázat hiánya korlátozza a neuronhálók alkalmazását. Léteznek ugyanakkor olyan eljárások, amelyek a neuronhálók súlyaiból emberek számára érthető, a döntéseket indokoló szabályokat nyernek ki [13].

Állítólag a 80-as években az amerikai hadsereg szolgálatba akarta állítani a mesterséges intelligenciát. Céljuk volt minden tankra egy kamerát tenni, a kamera képét egy számítógépnek továbbítani, amely automatikusan felismeri az ellenséges tankot. A kutatók neurális hálózat alapú megközelítés mellett döntöttek. A tanításhoz előállítottak 100 darab olyan képet, amelyen a fák mögött tank bújt meg, és 100 olyat, amelyen tank nem volt látható. Néhány iteráció után a hálózat tökéletesen osztályozta a képeket. A kutatók nagyon meg voltak elégedve, ugyanakkor még maguk sem voltak biztosak abban, hogy a neurális hálózat valóban a tank fogalmát tanulta-e meg. Független szakértőktől kért verifikáció során azonban a háló rosszul szerepelt: nem osztályozott pontosabban, mint egy teljesen véletlenszerűen tippelő osztályozó. Későbbi vizsgálatok során kiderült, hogy a tanításhoz használt összes tankos képen borult volt az idő, a tank nélküli képeken pedig sütött a nap.

Nem tudni, hogy a történet mennyiben igaz, az azonban tény, hogy a neurális háló nem ad magyarázatot az osztályozás okára. Ez komoly hátrány például a pénzügyi vagy orvosi világban.

Osztályozók kiértékelése

Az osztályozó algoritmusok két fázisban dolgoznak: az első, tanítási fázisban egy T tanító-adatbázis segítségével létrehoznak egy modellt, amely képes arra, hogy új objektumokat osztályozzon, olyan objektumokat, amelyekről nem ismert, hogy milyen osztályba tartoznak. Az osztályozó (és regressziós) algoritmusok tárgyalásánál mindeddig az első fázisra összpontosítottunk. Felmerül a kérdés, hogy ha túl vagyunk a tanításon, vajon hogyan tudjuk mérni, hogy a kapott osztályozó modell mennyire jól működik.

A legelső ötletünk az lehet, hogy az osztályozó modell tanításakor használt T tanító-adatbázis egyes objektumainak (példányainak) osztályát próbáljuk meg a modell segítségével felismerni, és a felismerés eredményét az adott példányok tényleges osztálycímkéivel összehasonlítani. A tanítóhalmazon ily módon számított hibát *resubstitution error*nak, azaz visszahelyettesítéses hibának nevezzük.

A visszahelyettesítéses hiba legtöbbször túlságosan is optimista. Például egy $k = 1$ legközelebbi szomszédot figyelembe vevő legközelebbi szomszéd osztályozó teljesítményét a tanítóhalmazon mérve azt kapjuk, hogy az osztályozó tökéletesen osztályoz. Amikor egy $x \in T$ objektumot (példányt) szeretnénk osztályozni, az x példány a tanítóhalmaz összes példánya közül nyilván saját magához van a legközelebb, tehát osztályozáskor tényleg a saját osztálycímkéjét kapja. Nem biztos ugyanakkor, hogy új, még nem látott objektumokra is ilyen jól működik az osztályozó.

A valóságban egy osztályozó – vagy regressziós – modellt új objektumok (példányok) osztályozására, illetve új objektumok (példányok) valamely nem mérhető tulajdonságának előrejelzésére, becslésére használunk. Egy hitelbírálati rendszerben például nem az az igazi kérdés, hogy a múltbeli ügyfelekről vissza tudjuk-e keresni, hogy azok késedelmesen fizették-e vissza a hitelüket, hanem az, hogy a potenciális új ügyfelekről mennyire nagy biztonsággal tudjuk előre jelezni, hogy melyikük fogja késedelmesen visszafizetni a hitelét. Az osztályozók kiértékelése során mindig egy ilyen gyakorlati szituációt szeretnénk modellezni.

A kiértékelés alapgondolata tehát az, hogy egy osztályozó algoritmus kiértékeléséhez a tanítóadatoktól független, új tesztadatokat kell használnunk, amelyeket korábban soha nem látott az osztályozó algoritmus. Ezért a gyakorlatban legtöbbször azt az elvet követik, hogy a rendelkezésre álló címkézett adatokat (amelyek esetében ismert, hogy melyik példány melyik osztályba tartozik) két diszjunkt részre osztják, egy T tanító és egy T_1 tesztadatbázisra, és a rendszer minőségét a tesztadatbázison mérik.

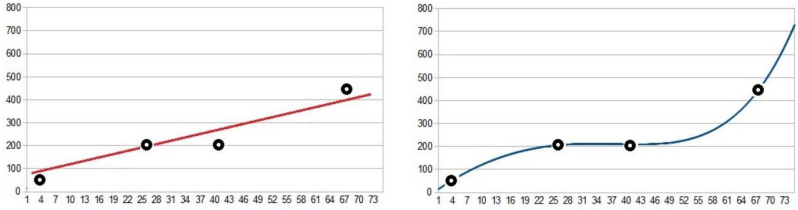
A következőkben a túltanulás (túlilleszkedés, overfitting) jelenségével foglalkozunk részletesebben, majd pedig az osztályozó és regressziós algoritmusok kiértékelésére használt protokollokkal. Bár a helyesen (vagy hibásan) osztályozott objektumok (példányok) aránya intuitív mérőszám lehet egy osztályozó minőségének mérésére, számos esetben (például, ha az osztályok eloszlása kiegyensúlyozatlan) félrevezető lehet. Az osztályozók és regressziós modellek minőségének mérésére használt legfontosabb mérőszámokat egy külön alfejezetben foglaljuk össze. Végezetül azzal a kérdéssel foglalkozunk, hogy mikor mondhatjuk nagy bizonyossággal, hogy az egyik osztályozó modell jobban teljesít, mint a másik.

Túltanulás

A túltanulás (túlilleszkedés, overfitting) jelensége arra vonatkozik, amikor egy osztályozó – vagy regressziós – algoritmus a tanulási fázisban a T tanítóadatok egyedi, speciális tulajdonságait tanulja meg, ezek alapján osztályoz, ahelyett, hogy az adott területre jellemző általános szabályszerűségeket tárná fel és használná a későbbiekben az új, ismeretlen osztályba tartozó objektumok (példányok) osztályozására.

A következőkben két példát mutatunk a túltanulásra. Tegyük fel, hogy egy döntési fa építésekor korlátozzuk a döntési fa csúcsainak számát n-ben. $n = 1$ esetében csak egyetlen csúcsot engedünk meg, ez nyilván egy levél lesz, amely minden objektumot (példányt) a többségi osztályba sorol. Az ilyen, szélsőségesen egyszerű döntési fa sem a T tanító-adatbázison mért visszahelyettesítéses hiba szerint, sem az attól független T_1 tesztadatbázison mért hiba szerint nem fog kimondottan jól teljesíteni. Ha kicsit több csúcsot engedünk meg, akkor ezáltal esélyt adunk a döntési fának, hogy valamilyen

nemtriviális összefüggést találjon a magyarázó változók és magyarázandó változó (osztályattribútum) között, és ezáltal az osztályozás minősége javulni fog, a hiba csökken mind a tanító-adatbázison, mind a tesztadatbázison mérve. Ha túl nagyra növeljük a döntési fában engedélyezett csúcsok számát, előbb-utóbb azt tapasztaljuk, hogy az osztályozás minősége csak a tanító-adatbázison mérve javul, a tesztadatbázison akár romolhat is. Ekkor a döntési fa a tanító-adatbázisra jellemző, egyedi, speciális sajátosságokat is megtanulja, ilyenkor beszélünk túltanulásról. Ahogy említettük, döntési fák esetében a metszés segíthet a túltanulás kiküszöbölésében. Ahhoz, hogy a modell kiértékelése igazságos legyen, figyelnünk kell az adatok megfelelő felosztására, ha a döntési fa építése után metszést is végzünk. Tegyük fel, hogy a T tanító-adatbázis segítségével felépítünk egy döntési fát, majd ennek minőségét a T-től független T_1 adatbázis segítségével mérjük, és azt tapasztaljuk, hogy a döntési fa túltanult. A korábbiakban látott módon ezt a problémát metszéssel orvosoljuk, egyes részfákat levelekkel, illetve utakkal helyettesítünk, ha a helyettesítés után a T_1 adatbázison javul az osztályozás minősége. Vajon mérhetjük-e ezek után a döntési fa minőségét a T_1 adatbázis segítségével? Nem, hiszen a T_1-t felhasználtuk a végső modell kialakításához, a T_1 segítségével döntöttünk arról, hogy a fa mely részeit akarjuk kimetszeni. Emlékezzünk: ahhoz, hogy igazságos módon értékeljünk ki egy osztályozó algoritmust, olyan adatokra van szükségünk, amelyeket soha nem látott korábban az algoritmus, most tehát szükségünk lesz egy T_2 adathalmazra, amely T-től és T_1-től egyaránt független (diszjunkt).



14. ábra. Példa a túltanulásra: a jobb oldali ábrán látható görbe ötödfokú polinom, minden pontra tökéletesen illeszkedik ugyan, mégsem mondhatjuk, hogy jobban megragadja az általános trendet, mint a bal oldali ábrán látható egyenes

A túltanulást szemléltető második példaként tekintsük a polinomiális regressziót. Tegyük fel, hogy a magyarázandó Y attribútumon kívül mindössze egyetlen magyarázó X attribútum van az adatbázisban. Vegyük észre, hogy a korábbiakban látott lineáris regressziós algoritmus alkalmazható magasabb fokú polinomiális regresszióra is, ehhez mindössze annyit kell tennünk, hogy előfeldolgozási lépésként további attribútumként felvesszük X^2-t, X^3-t stb. attól függően, hogy milyen fokú polinommal kívánjuk közelíteni Y-t. A 14. ábrán egy példát láthatunk a túltanulásra: a jobb oldali ábrán látható ötödfokú polinom minden pontra tökéletesen illeszkedik ugyan, mégsem mondhatjuk, hogy jobban megragadja az általános trendet, mint a baloldali ábrán látható egyenes.

A túltanulás mindkét látott példában, csakúgy, mint általánosságban is, a modell bonyolultságával van összefüggésben. A tanulás során túlságosan bonyolult modellt (túl nagy döntési fát, túl magas fokú polinomot) használtunk.

A túltanulást fel tudjuk ismerni. Például abból, hogy a modell lényegesen különböző sikerrel osztályoz a független teszthalmazon, mint a tanítóhalmazon. Az egyes modellek esetében gyakran tudunk is tenni valamit a túltanulás ellen. Döntési fáknál a metszés; neurális hálóknál az egyszerűbb struktúra, illetőleg kevesebb belső neuron használata; vagy az olyan esetekben, amikor az osztályozó, illetve regressziós algoritmus tanulása egy célfüggvény optimumának közvetlen keresését jelenti, beépíthetünk egy büntetőtagot a célfüggvénybe, amely a túl bonyolult modelleket bünteti, és ezáltal nem engedi, hogy a célfüggvény egy túl bonyolult modell esetében vegye fel az optimumát. A túltanulás a modellek kiértékelése szempontjából azért lényeges, mert aláhúzza azt az alapvető megállapításunkat, hogy a modell igazságos módon történő kiértékeléséhez új, korábban a modell által sohasem látott adatokat kell használnunk.

A korábbiakhoz képest kevésbé intuitív, hogy a túltanulás nem csak az egyes modellek szintjén léphet fel, hanem magasabb szinten is. Ezt is egy példával szemléltetjük: egy ügyfél azzal bízott meg bennünket, hogy készítsünk számára egy osztályozó modellt. Sok különböző eljárást kipróbáltunk: döntési fákat, neurális hálókat, Bayes-osztályozókat, szupport vektor gépeket stb. Az egyes osztályozó algoritmusok tanítása során a T tanító-adatbázist

használtuk, az algoritmusok kiértékeléséhez pedig a T-től független T_1 adatbázist. A döntési fák metszését a tanítás részének tekintjük, ehhez a T-t osztottuk két részre: T_A-ra és T_B-re. T_A segítségével építettük az eredeti döntési fát, T_B segítségével pedig metszettük a fát. A T_1 tesztadathalmaz, amelyet arra használtunk, hogy a legjobb modellt kiválasszuk, nyilván T_A-tól és T_B-től is független. Az osztályozási feladat megoldásként nyilván az általunk vizsgált sok, különféle modell közül a legjobbat szállítjuk az ügyfélnek. Vajon mit mondhatunk ezen legjobb modell teljesítményéről? Mondhatjuk-e, hogy a modell hibája (közelítőleg) akkora, amekkorának a T_1 halmazon mértünk? Elsőre azt gondolnánk, hogy igen, hiszen a modell készítése során a T_1-től független T-t használtuk tanítóadatként. A végső modell megalkotása szempontjából azonban a legjobb modell kiválasztása is a tanulási folyamat részének tekinthető. Előfordulhat, hogy az általunk legjobbnak választott modell pusztán a T_1 halmaz valamilyen specialitása miatt bizonyult a legjobbnak, és egy másik adathalmaz mellett már nem lenne jobb a többi vizsgált modellnél. Ahhoz tehát, hogy a kiválasztott legjobb modell hibáját igazságos módon becsüljük, egy újabb, T-től és T_1-től egyaránt független T_2 adathalmazra van szükségünk.

Kiértékelési protokollok

Általában feltételezzük, hogy az osztályozó algoritmus tanítása során használt tanítóadatok reprezentatívak, a későbbiekben osztályozandó adatok ugyanolyan eloszlásból kerülnek ki. Ez ugyan a gyakorlatban nem feltétlenül teljesül, egy ilyen implicit feltételezés hiányában azonban aligha lehetne bármilyen osztályozó algoritmust is tanítani, és legtöbbször a feltételezés legalább nagyjából teljesül. Így tehát az osztályozó algoritmusok kiértékelésekor használt protokollok során is megengedjük, hogy a tanítóadatok reprezentatívak legyenek.

Honnan tudjuk eldönteni, hogy az adathalmazunk egy része reprezentatív-e? Általánosan sehogy. Van azonban egy egyszerű vizsgálat, amelyet érdemes elvégezni. A tanító és a teszt adathalmazban az egyes osztályok eloszlása nagyjából meg kell, hogy egyezzen. Nem várhatunk jó osztályozást, ha a tanítóhalmazba nem került valamely osztályból egyetlen elem sem. Az eredeti

adathalmaz olyan felosztását (tanító és teszthalmazra), amelyre teljesül, hogy az osztályok relatív előfordulásai a tanítóhalmazban és a teszthalmazban nagyjából megegyeznek, rétegzett (stratified) particionálásnak/mintavételezésnek hívjuk.

A következőkben a leggyakrabban használt kiértékelési protokollokat tekintjük át.

Ismételt mintavételezés

Az eredeti adathalmaz nagyobb részét (általában kétharmadát) válasszuk tanítóhalmaznak, a maradékon határozzuk meg a modell hibáját! Ismételjük többször az eljárást különböző, véletlenszerűen választott tanítóhalmazokon! Az osztályozás végső hibáját az egyes felosztásokon mért hibák átlagaként adjuk meg.

Keresztvalidáció és a leave-one-out

Osszuk fel a tanítóhalmazt *N* részre! Az adott osztályozó módszerrel *N* különböző tanítást fogunk végezni. Minden tanításnál egy rész lesz a teszthalmaz, a többi uniója pedig a tanítóhalmaz. Minden tanításnál más teszthalmazt választunk. A végső hibát az egyes hibák átlaga adja. Igen elterjedt, hogy *N* értékének 10-et adnak meg.

A keresztvalidáció egy speciális esete, amikor *N* értéke megegyezik a tanítópontok számával, azaz csak egyetlen objektumból (példányból) áll a tesztadatbázis. Ezt a módszert *leave-one-out*-nak (egy kimarad) hívják. A módszer előnye, hogy teljesen determinisztikus, továbbá a tanításhoz a lehető legtöbb információt használja. Hátrány ugyanakkor, hogy a tanítást sokszor kell elvégezni, ami nagyon költséges lehet, továbbá a teszteléshez használt adathalmaz biztos, hogy nem rétegzett.

Egyes kutatók úgy vélik, hogy a keresztvalidáció jelentősége túl van értékelve, hiszen elméletileg nem lehet bizonyítani, hogy megbízhatóbb eredménnyel szolgál, mint az egyszerű oszd ketté (taníts, majd tesztelj!) módszer.

Mérőszámok

A legfontosabb mutatószám az osztályozó pontossága (*accuracy*), amely a jól osztályozott objektumok (példányok) számának arányát adja meg az összes objektum (példány) számához viszonyítva. A pontossághoz nagyon hasonló mérőszám a hibaarány (*misclassification ratio*, *error rate*), amely a helytelenül osztályozott objektumok (példányok) aránya. Nyilván

$$\text{hibaarány} = 1 - \text{pontosság}.$$

A pontosság és hibaarány megtévesztő lehet. A magas pontosság (illetve alacsony hibaarány) nem biztos, hogy a szofisztikált módszerünk eredménye. Ha például bináris osztályzás esetében az egyik osztály előfordulásának valószínűsége 90%, akkor egy 88% pontosságú osztályozó rossz osztályozó, hiszen pontossága rosszabb, mint azon naiv osztályozóé, amely mindig a gyakoribb osztályra tippel. Egy még naivabb osztályozó a véletlen osztályozó, amely a *C* osztályt p_C valószínűséggel választja, ahol p_C a *C* osztály előfordulásának valószínűsége. A valószínűséget relatív gyakorisággal közelítik. A véletlen osztályozó várható pontossága az előbbi példában $0{,}9 \cdot 0{,}9 + 0{,}1 \cdot 0{,}1 = 82\%$.

Egy osztályozó kappa statisztikája az osztályozó pontosságát a véletlen osztályozóhoz hasonlítja. Tegyük fel, hogy a tanítóhalmazon az egyes osztályok relatív gyakoriságai $p_1, p_2, \ldots, p_k$, és a tanítóhalmazon az osztályok előfordulása $n_1, n_2, \ldots, n_k$. Legyen $N = \sum_{i=1}^{k} n_i$ és $M = \sum_{i=1}^{k} n_i p_i$. A kappa statisztikát ekkor a

$$\frac{T - M}{N - M}$$

hányados adja, ahol *T*-vel a helyesen osztályozott pontokat jelöljük. A kappa statisztika 0 és 1 közé esik. A véletlen osztályozó kappa statisztikája 0, a tökéletes osztályozóé pedig 1.

A pontosság (és hibaarány) nem csak azért lehet félrevezető, mert a naiv, illetve véletlen modellek pontossága nagy (hibaaránya kicsi) lehet, és ezekhez kell viszonyítanunk. Kiegyensúlyozatlan osztályeloszlás esetén sem

célszerű a pontosság (és hibaarány) használata. Képzeljük el, hogy egy ritka betegség diagnosztizálására valamilyen osztályozó algoritmust használunk. A ritka betegség a népesség mindössze 0,1%-át érinti. Vajon melyik osztályozó a jobb?

a) Amelyik mindenkit egészségesnek osztályoz, vagy
b) amelyik az esetek 5%-ában téved ugyan, de a betegek nagy részét felismeri?

Az a) modell nyilván teljesen használhatatlan, míg a b) sokat segíthet a betegség diagnosztikájában, még akkor is, ha nem tökéletes. Ezzel szemben az a) modell pontossága mégis magasabb, mint a másodiké.

Az előbbi mérőszámoknál részletesebben írja le egy osztályozó teljesítményét az ún. keveredési mátrix (*confusion matrix*), amely annyi sorból és oszlopból áll, amennyi az osztályok száma. Az *i*-edik sor *j*-edik eleme adja meg azoknak a pontoknak a számát, amelyeket az osztályozó a *j*-edik osztályba sorol, holott azok az *i*-edik osztályba tartoznak. A diagonálisban található elemek adják meg a helyesen osztályozott pontok számát. Alább egy keveredési mátrixot láthatunk.

		Felismert osztály			
		a	**b**	**c**	**Σ**
Tényleges osztály	**a**	88	10	2	100
	b	14	40	6	60
	c	18	10	12	40
	Σ	120	60	20	

Bináris osztályozás esetére, amikor az osztályozó kimenete 0 vagy 1 (igaz/hamis vagy pozitív/negatív), további fogalmakat definiálunk. Többosztályos osztályozási feladat esetén kijelölhetünk egy kitüntetett (pozitív) osztályt, minden egyéb osztályt összevonhatunk egy negatív osztályba, és ekkor a bináris esethez hasonlóan használhatjuk az alábbi megnevezéseket. A jól osztályozott objektumok (példányok) számát *TP*-vel (*True Positive*) és *TN*-nel (*True Negative*) jelöljük attól függően, hogy melyik osztályba

tartoznak. A rosszul osztályozott objektumok (példányok) jelölése *FP*, illetve *FN* (*False Positive*, *False Negative*). A következő keveredési mátrix összefoglalja a jelöléseket.

		Felismert osztály	
		+	–
Tényleges osztály	+	*TP*	*FN*
	–	*FP*	*TN*

A felidézést vagy megbízhatóságot (angolul *recall* vagy *true positive rate*), amelyet bináris osztályozásnál érzékenységnek (*sensitivity*) is hívnak az

$$R = \frac{TP}{TP + FN}$$

hányados adja. A precision fogalmát a következőképpen számolhatjuk:

$$P = \frac{TP}{TP + FP}.$$

E két érték parametrikus harmonikus közepét *F*-mértéknek (angolul *F*-measure) nevezzük:

$$F = \frac{1}{\alpha \frac{1}{P} + (1 - \alpha) \frac{1}{R}}.$$

A leggyakrabban, amikor ennek ellenkezőjét nem jelezzük, $\alpha = 0{,}5$ érték mellett számítjuk az *F*-measure-t. Az $\frac{FP}{FP+TN}$ hányadost selejtnek (*fallout*, *false positive rate*) is nevezik. A korábban már tárgyalt pontosság (*accuracy*) is definiálható a *TP*-k és *TN*-k segítségével: $\frac{TP+TN}{N}$.

Tekintsünk egy olyan osztályozó modellt, amely nem csak egy diszkrét döntést ad eredményül, hogy egy adott teszthalmazbeli objektum (példány) a pozitív vagy negatív osztályba tartozik, hanem egy folytonos kimenetet eredményez, amely annál nagyobb, minél több eséllyel tartozik (a modell szerint) egy adott objektum (példány) a pozitív osztályba. A *TP*-k, *TN*-k, *FP*-k és

FN-k száma ekkor annak függvénye, hogy milyen Θ küszöbérték felett tekintjük a modell kimenetét pozitívnak. A true positive rate-t (recall, felidézés, megbízhatóság) jelöljük *TPR*-rel: $TPR = \frac{TP}{TP+FN}$. Ehhez hasonlóan jelöljük *FPR*-rel (false positive rate) a *FP*-k arányát az összes negatív osztályba tartozó objektumhoz képest: $FPR = \frac{FP}{FP+TN}$. Nyilván *TPR* és *FPR* is a Θ küszöbérték függvénye. A *TPR*-t ábrázolhatjuk *FPR* függvényében. Az így kapott görbét nevezik *Receiver-Operator Curve*-nek vagy röviden ROC görbének. Az *AUC* (*Area Under the Curve*) az ROC görbe alatti területre vonatkozik. Tökéletes osztályozó modell esetében, amikor található olyan küszöbszám, amely mellett a modell kimenete tökéletesen megegyezik a tényleges osztályokkal, az *AUC* értéke 1, véletlenszerű kimenetet adó modell esetében pedig 0,5.

Hiba mérése regresszió esetében

Amikor a magyarázandó attribútum szám típusú, akkor a leggyakrabban használt hiba a négyzetes hibaátlag (vagy annak nemnegatív négyzetgyöke). Az elterjedt használat oka, hogy a négyzetes hibaösszeg könnyen kezelhető matematikailag – gondoljunk csak a lineáris regresszióra, amely sok regressziós módszer kiindulópontjaként szolgál. Ha csökkenteni szeretnénk a kiugró értékekkel rendelkező pontok által okozott hiba mértékét, akkor használhatunk átlagos hibakülönbséget is. A leggyakrabban használt ilyen mennyiségeket alább felsoroljuk. Valamely teszthalmazban (amely származhat például keresztvalidációból) az *i*-edik objektum (példány) magyarázandó változójának tényleges értékét y_i-vel, ugyanezen példány magyarázandó változójának a modell által becsült értékét $\hat{y}_i$-pal jelöljük.

Átlagos négyzetes hiba:

$$\frac{(y_1 - \hat{y}_1)^2 + \cdots + (y_n - \hat{y}_n)^2}{n}.$$

Átlagos négyzetes hiba négyzetgyöke:

$$\sqrt{\frac{(y_1 - \hat{y}_1)^2 + \cdots + (y_n - \hat{y}_n)^2}{n}}.$$

Abszolút hibaátlag:

$$\frac{|y_1 - \hat{y}_1| + \cdots + |y_n - \hat{y}_n|}{n}.$$

Többször láttuk, hogy nem az abszolút, hanem inkább a relatív hiba érdekel minket. Azt gondoljuk, hogy ugyanakkora súlyú hibát vétünk, ha 200 helyett 220-at jósolunk, mint amikor 1 helyet 1,1-et. A fenti hibamértékek relatív változatainak képletei a következők:

Relatív négyzetes hiba:

$$\frac{(y_1 - \hat{y}_1)^2 + \cdots + (y_n - \hat{y}_n)^2}{(y_1 - \bar{y})^2 + \cdots + (y_n - \bar{y})^2}.$$

Relatív négyzetes hiba négyzetgyöke:

$$\sqrt{\frac{(y_1 - \hat{y}_1)^2 + \cdots + (y_n - \hat{y}_n)^2}{(y_1 - \bar{y})^2 + \cdots + (y_n - \bar{y})^2}}.$$

Relatív hibaátlag:

$$\frac{|y_1 - \hat{y}_1| + \cdots + |y_n - \hat{y}_n|}{|y_1 - \bar{y}| + \cdots + |y_n - \bar{y}|}.$$

Korrelációs együttható:

$$\frac{(y_1 - \bar{y})(y_1 - \hat{y}_1) + \cdots + (y_n - \bar{y})(y_n - \hat{y}_n)}{\sqrt{((y_1 - \bar{y})^2 + \cdots + (y_n - \bar{y})^2)((y_1 - \hat{y}_1)^2 + \cdots + (y_n - \hat{y}_n)^2)}}.$$

A korrelációs együttható (amely -1 és $+1$ közé esik) kilóg a sorból két dolog miatt. Egyrészt ez a mérték skálainvariáns, azaz, ha minden jósolt értéket megszorzunk egy adott konstanssal, akkor a korrelációs együttható nem

változik. Másrészt minél jobb az osztályozó módszer, annál közelebb lesz az együttható egyhez. A többi mérték értéke 0 lesz a tökéletes osztályozó esetében.

Irodalomjegyzék

[1] Aurenhammer, Franz (1991): *Voronoi diagrams — a survey of a fundamental geometric data structure*, ACM Computing Surveys, Volume 23, Number 3, pp. 345–405, ACM, New York, NY, USA.

[2] Bishop, Christopher M. (2006): *Pattern Recognition and Machine Learning*, Springer.

[3] Blum, A., R. L. Rivest (1988): *Training a 3-node neural network is NP-Complete*, Extended abstract, Proceedings of the Workshop on Computational Learning Theory, pp. 9–18, San Francisco, CA.

[4] Breiman, Leo, Jerome Friedman, Charles J. Stone, R. A. Olshen (1984): *Classification and Regression Trees*, Chapman & Hall/CRC.

[5] Gelfand, S.B., C.S. Ravishankar, E.J. Delp (1991): *An Iterative Growing and Pruning Algorithm for Classification Tree Design*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 13, Number 2, pp. 163–174, IEEE Computer Society, Los Alamitos, CA, USA.

[6] Cover, Thomas M. and Joy A. Thomas (1991): *Elements of Information Theory*, Wiley Series in Telecommunications.

[7] De Mántaras, R. López (1991): *A Distance-Based Attribute Selection Measure for Decision Tree Induction*, Mach. Learn., Volume 6, Number 1, pp. 81–92, Kluwer Academic Publishers, Hingham, MA, USA.

[8] Devroye, L., L. Györfi, G. Lugosi (1996): *A Probabilistic Theory of Pattern Recognition*, Springer-Verlag, New York.

[9] Dietterich, T. G., M. Kearns, Y. Mansour (1996): *Applying the Weak Learning Framework to Understand and Improve C4.5*, Proceedings of the 13th International Conference on Machine Learning (ICML'96), Morgan Kaufmann, pp. 96–104, San Francisco, CA, USA.

[10] Dietterich, Thomas G. (2000): *Ensemble Methods in Machine Learning*, Multiple Classifier Systems, Lecture Notes in Computer Science, Volume 1857/2000, Springer, pp. 1–15.

[11] Freund, Y., R. Schapire (1994): *A decision-theoretic generalization of on-line learning and an application to boosting*, Proceedings EuroCOLT94: European Conference on Computational Learning Theory. LNCS, Springer.

[12] Futó Iván (1999): *Mesterséges Intelligencia*, Aula Kiadó, Budapest [Fredkin, 1960] Edward Fredkin (1960): *Trie memory*, Communications of the ACM, Volume 3, Number 9, pp. 490–499, ACM Press.

[13] Han, Jiawei, Micheline Kamber (2006): *Data mining: concepts and techniques*, second edition, Morgan Kaufmann Publisher.

[14] Hornik, K., M. Stinchcombe, H. White (1990): *Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks*, Neural Networks, Volume 3, pp. 551–560.

[15] Hull, D. A. (1994): *Improving text retrieval for the routing problem using latent semantic indexing*, Proc. of SIGIR-94, 17th ACM Int. Conf. on Research and Development in Information Retrieval, pp. 282–289.

[16] Hyafil, L., R. L. Rivest (1976): *Constructing optimal binary decision trees is NP-Complete*, Information Processing Letters, Volume 5, Number 1, pp. 15–17.

[17] Kearns, M., Y. Mansour (1996): *On the boosting ability of top-down decision tree learning algorithms*, STOC ’96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, pp. 459–468, ACM Press, New York, NY, USA.

[18] Quinlan, J. R. (1986): *Induction of Decision Trees*, Machine Learning, Volume 1, Number 1, pp. 81–106, Kluwer Academic Publishers, Hingham, MA, USA.

[19] Quinlan, J. R. (1987): *Simplifying decision trees*, Int. J. Man-Mach. Stud., Volume 27, Number 3, pp. 221–234, Academic Press Ltd., London, UK.

[20] Quinlan, J. Ross (1993): *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[21] Rätsch, G., T. Onoda, K.-R. Müller (2001): *Soft Margins for AdaBoost* Machine Learning, Volume 42, Number 3, pp. 287–320.

[22] Read, T. R. C., N. A. C. Cressie (1988): *Goodness-of-Fit Statistics for Discrete Multivariate Data*, Springer Series in Statistics, Springer-Verlag, New York, NY, USA.

[23] Rifkin, Ryan, Aldebaro Klautau (2004): *In Defense of One-Vs-All Classification*, The Journal of Machine Learning Research, Volume 5, pp. 101–141.

[24] Russel, S., P. Norvig (2010): *Artificial Intelligence, A Modern Approach*, Prentice Hall, Upper Saddle River, NJ, USA.

[25] Sebastiani, F. (2002): *Machine learning in automated text categorization*, ACM Computing Surveys, Volume 34, Number 1, March 2002, pp. 1–47.

[26] Schapire, R. E., Y. Singer, A. Singhal (1998): *Boosting and Rocchio applied to text filtering*, Proc. of SIGIR-98, 21st ACM International Conference on Research and Development in Information Retrieval, pp. 215–223.

[27] Shih, Y.-S. (1999): *Families of splitting criteria for classification trees*, Statistics and Computing, Volume 9, Number 4, pp. 309–315, Kluwer Academic Publishers, Hingham, MA, USA.

[28] Tan, Pang-Ning, Michael Steinbach, Vipin Kumar (2006): *Introduction to Data Mining*, Addison-Wesley.

[29] Wiener, E. D., J. O. Pedersen, A. S. Weigend (1995): *A neural network approach to topic spotting*, Proc. of the SDAIR-95, 4th Annual Symposium on Document Analysis and Information Retrieval, pp. 317–332.

Kiadó:

Juhász Gyula Felsőoktatási Kiadó
Felelős kiadó: *Forró Lajos igazgató*

Tördelés
Innovariant Nyomdaipari Kft.
Felelős vezető: *Drágán György*
www.innovariant.hu
www.facebook.com/Innovariant