Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Sárándi István

EGÉSZSÉGÜGYI KÓDOLÁSTÁMOGATÓ RENDSZER FEJLESZTÉSE

KÜLSŐ KONZULENS

Héja Gergely

TANSZÉKI KONZULENS

Dr. Strausz György

BUDAPEST, 2011

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Sárándi István**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2011. 12. 09.

.....

Sárándi István

Összefoglaló

Az utóbbi években a számítógépek számítási kapacitásának robbanásszerű fejlődésével egyre több szellemi munkát igénylő feladat válik automatizálhatóvá. A gyorsabb hardvereknek köszönhetően a gyakorlatban is futtathatóvá váltak olyan számításigényes algoritmusok, melyek a mesterséges intelligencia, a gépi tanulás és a szövegbányászat területén jellemzőek. Szakdolgozatomban ezeket a módszereket az egészségügyi kódolásra alkalmazom.

Az egészségügyben finanszírozási és népegészségügyi (pl. járványkövetési) célokból statisztikai adatokat kell gyűjteni az elvégzett kezelésekről, a kezelt betegségekről. Ilyenkor a betegségeket formális kódrendszerekben ábrázolják, melyek sajnos meglehetősen bonyolultak, a zárójelentések és diagnózisok manuális kódolása rendkívül idő- és így pénzigényes tevékenység. A szakdolgozatban e folyamat részleges automatizálásának lehetőségét vizsgálom meg. Létrehozok egy olyan rendszert, amely a kódoló személyzetet segíti a kód gyorsabb megtalálásában azáltal, hogy egy diagnózisra válaszként visszaadja a legrelevánsabb kódok listáját. A későbbiekben előnyös lenne bevezetni Magyarországon egy ilyen központi, webes felülettel is rendelkező szolgáltatást, ezért a rendszer kialakítását úgy terveztem meg, hogy egy ilyen szolgáltatás alapja lehessen, illetve elkészítettem egy prototípus webes felületet is hozzá.

Szakdolgozatomban egy általános bevezető rész után bemutatom az eddigi nemzetközi megoldási javaslatokat. A feladat alapvető sémájának felvázolását követően részletesen ismertetem a felhasznált osztályozási módszereket (vektortér, naiv Bayes, perceptron, szupport vektor gép). Ezután megvizsgálom, milyen lehetőségek rejlenek több osztályozó eredményeinek keverésében, hibrid modellek létrehozásában. Ezt követően a konkrét implementációt mutatom be, felvázolom a létrehozott osztályozási keretrendszer felépítését és a webes prototípus működését. A rendszert magyar és német nyelvű BNO-kódrendszer szerint kódolt mintahalmazokkal részletes kiértékelésnek vetem alá. Megvizsgálom a felhasznált minták előfeldolgozottságának hatását, végül értékelem az elkészült megoldást és továbbfejlesztési lehetőségeket sorolok fel.

Abstract

As a result of the rapid growth in computing power in recent years, more and more intellectually demanding tasks can be automatized. It has become possible to run numerous computationally expensive algorithms that are characteristic for the fields of artificial intelligence, machine learning and text mining. I apply such methods in my thesis for the problem of medical coding.

The health care system collects statistical data about applied treatments and treated diseases in order to reimburse health care providers and to monitor public health (e.g. epidemics). In such statistics, diseases are represented using formal code systems. These systems assign codes in a non-trivial way, so that the manual coding of diagnoses and medical reports becomes a tedious, time and money consuming task. In my thesis, I examine the possibility of partially automatizing this process by creating a system for helping the coding personal in finding the correct code faster. The system returns a list of the most relevant codes for a given diagnosis. In the future, it would be advantageous to introduce such a central service with a web interface in Hungary. I developed my system so that it may become the basis of that and created a proof-of-concept web interface for the program.

After a general introduction, I explore the previous international works and solutions on the topic. Following a schematic explanation of the task, I present the concrete classification algorithms that I used (vector space, naïve Bayes, perceptron, support vector machine). After that, I investigate the possibility of mixing the results of multiple classifiers thus creating hybrid models. Then I explain my concrete implementation, present the implemented classification framework system and the proof-of-concept web interface. I test my system in detail using Hungarian and German sample sets coded according to the ICD system and examine the effects of sample preprocessing on the results. At the end, I evaluate the solution and look into further development possibilities.

1 Bevezetés

A gépi tanulás már nem csak egyetemi és speciális felhasználói körrel rendelkezik. Cégek, intézmények is egyre inkább érdeklődnek az ebben rejlő lehetőség iránt, saját speciális problémáik megoldása céljából. A potenciális felhasználói kör bővülésével párhuzamosan – egymást elősegítve – természetesen az algoritmusok fejlesztése, új ötletek megjelenése is zajlik.

Jelen szakdolgozatban a gépi tanulás és a szövegbányászat legújabb eredményeit az egészségügyi kódolásra alkalmazom. Az egészségügyben az automatikus feldolgozhatóság érdekében gyakran szigorúan formalizált kódrendszerekben ábrázolják a betegségeket, ilyen kódrendszer például az itthon és sok más országban is alkalmazott *Betegségek és az egészséggel kapcsolatos problémák nemzetközi statisztikai osztályozása* (BNO)[1]. A diagnózisok manuális kódolása idő- és pénzigényes, emiatt külföldön és itthon is egyre több helyen végeznek olyan kutatásokat, amelyek a feladat (részleges) automatizálási lehetőségét vizsgálják. Ennek ellenére még mindig leginkább emberi munkával, kézzel történik a kódolás.

1.1 Előzmények, aktualitás

A szakdolgozatom elkészítése során konzulensemtől, Héja Gergelytől, a *Gyógyszerészeti és Egészségügyi Minőség- és Szervezetfejlesztési Intézet* (GYEMSZI) munkatársától megtudtam, hogy körülbelül 10-15 éve merült fel egy magyar automatikus egészségügyi kódoló rendszer fejlesztésének ötlete, és bizonyos algoritmusok implementálásra is kerültek. Az ezekkel kapcsolatos eredményekről és az egészségügyi kódolás Magyarországi helyzetéről Dr. Surján György disszertációjában találhatók részletek [1]. Jelen szakdolgozat aktualitását az adja, hogy a technológia fejlődésével bonyolultabb osztályozó algoritmusok a 10 évvel ezelőtti helyzethez képest sokkal könnyebben kezelhetők, kivárható a futásuk, így többféle módszer is kipróbálható lett, az automatikus kódolási szolgáltatás bevezetése pedig még aktuálisabbá vált. Ennek szem előtt tartásával készítettem el a dolgozatban bemutatandó programot.

[1] angolul *International Statistical Classification of Diseases and Related Health Problems* (ICD)

2 A feladat részletezése

A szakdolgozat célja egy olyan rendszer létrehozása, mely a felhasználó által megadott néhány szavas diagnózisokhoz tartozó legrelevánsabb BNO-kódok listáját képes meghatározni és webes felületen megjeleníteni.

A feladat megoldásához a következőket végzem el:

- A szakirodalomban publikált statisztikus és tanuló algoritmusokat áttekintem, majd a legígéretesebbeket kiválasztom.
- Az algoritmusokat adott problémára szabva, platformfüggetlen módon implementálom, hogy az egy későbbi központi szolgáltatás alapja lehessen. Elkészítem az ehhez szükséges keretrendszert.
- Hibrid, kevert módszereket vizsgálok.
- Összevetem az eredményeket különböző előfeldolgozottsági szintű (szótövezett, morfológiailag elemzett), mintahalmazokon mérve.
- Megtervezem a későbbi kliensprogramokkal való kommunikációt, illetve kialakítok egy prototípus webes felületet.

2.1 Az elkészítendő felület

A felhasználói felületnek a következő séma szerint kell működnie:

1. A felhasználó (pl. orvos, egészségügyi kódoló) webes felületen begépeli a diagnózist
2. A böngészőben visszakap egy találati listát és a relevanciákat.
3. A felhasználó megnézi a talált kódok leírását a BNO adatbázisban (könyvben) és dönt a helyes kódról.

2.2 Az elkészítendő keretrendszer

A keretrendszernek képesnek kell lennie a következőkre:

- osztályozó és előfeldolgozó algoritmusok, mintahalmazok és eredménylisták egységes, moduláris rendszerben történő kezelése
- osztályozók több szempont szerinti kiértékelése
- rugalmasság, egyszerű bővíthetőség új algoritmusokkal, kiértékelési módokkal stb.

3 Előzmények, megközelítési lehetőségek

A megoldandó feladatban diagnózisokhoz szeretnénk BNO-kódokat rendelni. Ez a folyamat hagyományosan manuálisan, kódoló szakértők segítségével (illetve gyakran az orvosok bevonásával) folyik. Elolvassák a diagnózist, majd vagy fejből, vagy a BNO-kódkönyvben utánanézve kódot rendelnek hozzá. E folyamat automatizálásának kutatása az 1990-es évektől kezdve indult meg előbb lassanként, majd igazán a 2000-es évtized közepétől kezdve lett népszerű kutatási téma. A kódolás automatizálásának történetéről sok releváns cikk bevezetőjéből tájékozódhatunk, például [2]-ből és [3]-ből.

Attól függően, hogy az automatizálás során mennyi apriori tárgyterületi tudást építenek a rendszerbe, különböző csoportokba sorolhatók a megoldások.

3.1 Szabályalapú megközelítés

A kezdeti megoldások tipikusan sok tárgyterületi tudást használtak fel, szabályalapú módon működtek. Ez azt jelenti, hogy szakértők saját tudásukat szabályok formájában fogalmazzák meg, ami alapján számítógépes következtetés végezhető. Ilyen szabályrendszerek összeállítása hosszadalmas, szakértelmet igényel, de várhatóan jó eredményekre vezet.

3.2 Ontológiák

A fogalmak és a köztük lévő kapcsolatok tárgyterületi ontológiák segítségével modellezhetők, ebben pl. betegségek, szervek stb. ábrázolhatók. Léteznek „open-source” egészségügyi ontológiák is, ilyen például a 2003-ban létrehozott *Disease Ontology* [4]. Ontológiákat használnak más típusú, statisztikai alapú rendszerekben is, kiegészítésként. Ez például szinonimák kezelésénél és szavak egyértelműsítésénél hasznos. Ilyen hibrid modellt mutat be a [5] cikk.

3.3 Természetesnyelv-feldolgozás

Mivel a zárójelentések és egyéb esetleírások természetes nyelven készülnek, felhasználnak nyelvfeldolgozó rendszereket is a kódolási folyamatban (pl. mondat- és kifejezésstruktúrák elemzése, szófaj megállapítása, vagy például Bulgáriában latin és

cirill ábécé közi konverzióra [6]). Jelen szakdolgozatban az előfeldolgozás kapcsán fogom megvizsgálni a természetesnyelv-feldolgozási módszerek hatását.

3.4 Gépi tanulás, statisztikai módszerek

Az utóbbi időben egyre inkább teret nyer a statisztikai alapú és a gépi tanulás felőli megközelítés, melyek korábban a számítási kapacitás szűke miatt nem kerültek felhasználásra. E módszerek használatához nincs szükség szakértelemre, ill. tárgyterületi tudásra, ezek általános célú módszerek, melyek már nagyon különböző területeken bizonyítottak (pl. képfelismerés, dinamikus rendszerek szimulációja, nyelvfeldolgozás stb.). Emiatt a szakdolgozatban ezekre a megoldási lehetőségekre koncentrálok.

Szinte minden létező tanulási algoritmust kipróbáltak már a BNO-kódolás automatizálásához, ez is jelzi a téma aktualitását. A teljesség igénye nélkül ilyen módszerek:

- k legközelebbi szomszéd (vektortér) [6]
- Döntési fák [7]
- Maximális entrópia alapú osztályozás [7]
- Naiv Bayes-hálók [8]
- Neurális hálózatok [9]
- Szupport vektor gépek [6] [10]
- Rejtett Markov-modellek [11]

Az irodalomban publikált feladat azonban sok esetben más, mint amiről ez a szakdolgozat szól. Sokszor egy hosszú esetleírásból indul ki a rendszer [12], maga választja ki ebből a fontos szavakat, kontextust elemez stb. Ebben a szakdolgozatban már kész, néhány szavasra összefoglalt diagnózisokkal dolgozom, nem például teljes zárójelentésekkel. Szintén előfordul a szakirodalomi publikációkban, hogy az egyes dokumentumokhoz több kód is tartozhat. Ilyen feladatot adtak ki a 2007-ben megrendezett Medical NLP[2] Challenge nemzetközi versenyen [13] is, melyet végül a szegedi egyetem kutatói nyertek meg. A szakdolgozatban tárgyalt esetben azonban a hozzárendelés egyértelmű, minden diagnózishoz (elvben) egyetlen kód tartozik. Ezek a különbségek az eredmények összehasonlítását meglehetősen nehézzé teszik.

[2] Az NLP a természetesnyelv-feldolgozás angol rövidítése (Natural Language Processing)

4 Az osztályozásról

Az osztályozás a felügyelt gépi tanulás egyik klasszikus problématípusa. Adott egy bemeneti halmaz, melynek elemei valamilyen ismeretlen hozzárendelés szerint osztályokba sorolandók. Esetünkben ezek a diagnózisok, melyek BNO-kódokhoz mint osztályokhoz rendelendők, azonban a hozzárendelés módját nem pontosan ismerjük, nem tudjuk egyszerűen definiálni. Felügyelt tanulásnál azonban abból indulunk ki, hogy rendelkezésünkre áll egy már osztályokkal „felcímkézett" bemenethalmaz. Ennek segítségével szeretnénk „rájönni" az osztályokba sorolás helyes módjára, hogy megbecsülhessük a még nem látott bemenetekhez tartozó osztályt is. Erre a problémára módszerek sokaságát dolgozták már ki.

Esetünkben az osztályozási feladatokon belül a dokumentumosztályozásnak nevezett problématípussal van dolgunk. Itt szövegek (sztringek) osztályokba sorolását kell megvalósítani. Tipikus példa erre a hír- és szakcikk-kategorizáló rendszerek által megoldott feladat. A diagnóziskódolási feladat annyiban különleges, hogy egy-egy diagnózis (dokumentum) sokkal rövidebb, mint az előbb említett példában. Ennek ellenére sok, hosszabb dokumentumok osztályozására tervezett módszer megállja a helyét (kisebb módosításokkal) ebben az esetben is.

Egy felügyelt tanuló algoritmus használata során először modellt építünk a felcímkézett bemenetek alapján (hipotézis kialakítása), majd a felépített modellt újabb mintákon alkalmazzuk. A modell jóságának mérésére több módszer is lehetséges.

4.1 Kiértékelési módok

Az elkészült osztályozókat valamilyen módon értékelni kell, hogy eldönthessük melyiket érdemes alkalmazni. Mivel minket a rendszer „általánosító képessége" érdekel, azaz, hogyan fog működni számára még ismeretlen bemenetek esetén, a tanítóhalmaztól független teszthalmazt kell alkalmaznunk. A tanító- és teszthalmazra bontást többféleképp is elvégezhetjük, majd az így a kapott értékeket átlagolva stabilabb eredményt érünk el. A leggyakoribb ilyen módszer a keresztkiértékelés (*cross-validation*) [14]: A tanítóhalmazt k részre osztjuk, majd közülük mindig egy részt használunk tesztelésre, a többit pedig tanításra.

Meg kell határozni a jóság mértékét is. Adja magát, hogy a találati arányt mérjük, azaz egy tesztminta osztályozása vagy sikeres, vagy sikertelen, és a sikerek arányát keressük az összes eset között.

Ebben a feladatban a probléma nehézsége miatt csak félautomatikus osztályozásra érdemes vállalkozni, azaz a rendszer megadott számú kódot adhat vissza, és már sikernek tekintjük, ha a kód szerepel a visszaadott listán. Legyen a visszaadott lista megengedett maximális hossza h, a tesztdiagnózisok száma t. Az arányt jelölje:

$$M(h) = \frac{s}{t} \qquad \textbf{4.1.1.}$$

Az $M(1)$ azt adja meg, hogy teljesen automatizált környezetben milyen sikere lenne a rendszernek.

Az tesztelést azonban célszerű szabványos módon (is) végezni. Ehhez az osztályozásnál általánosan használt mértékeket vizsgálom meg. Ezek a pontosság (*precision*), és a teljesség (*recall*), amiket a 2007-es *Medical NLP Challenge* verseny kiírásában [15] definiáltak szerint értelmezek.

Ezek első megközelítésben a bináris esetre vonatkoznak, azaz amikor a bementeteket két osztály valamelyikébe kell sorolni, pl. az egyszerűség kedvéért „igen” vagy „nem”. Ilyenkor egy tesztbemenetnél négy eset léphet fel, aszerint, hogy mi a rendszer kimenete és mi a kívánt válasz. Jelölje ezen esetek előfordulási darabszámát A, B, C, D a következő táblázat szerint:

	Kívánt: Igen	Kívánt: Nem
Rendszer: Igen	A	B
Rendszer: Nem	C	D

Ezek az esetek betűrendben az igaz pozitív, hamis pozitív, hamis negatív, igaz negatív nevet viselik. A pontosság megadja az igaz pozitív esetek részarányát a „rendszer: igen” esetek között.

$$P = \frac{A}{A + B} \qquad \textbf{4.1.2.}$$

A teljesség pedig az igaz pozitív esetek és a „kívánt: igen” esetek közti arányt adja meg.

$$R = \frac{A}{A + C} \qquad \textbf{4.1.3.}$$

Ezek tehát bináris osztályozásra vonatkoznak, a diagnóziskódolás esetén azonban sok kódból választunk. Ehhez a feladatot – a kiértékelés erejéig – értelmezhetjük több bináris osztályozási feladat összességeként: minden egyes kód meghatározására egy-egy külön feladatként tekintve kitölthetjük a táblázatot, ahol az „igen-nem” azt jelenti, hogy az adott kód az adott tesztdiagnózishoz tartozik-e. (Esetünkben „Rendszer: igen” = a kód a listán van)

Ezeket a kódonként külön-külön kitöltött táblázatokat össze kell vonni valamilyen módon, hogy a teljes tesztet értékelhessük. Erre két megoldás szokványos, a mikroátlagolás (*micro-averaging*) és a makroátlagolás (*macro-averaging*). Mikroátlagolás esetén a táblázatokat mezőnként összeadjuk, majd az így összegzett értékeket írjuk a fenti képletekbe:

	Kívánt: Igen	Kívánt: Nem
Rendszer: Igen	$\sum_{kód} A$	$\sum_{kód} B$
Rendszer: Nem	$\sum_{kód} C$	$\sum_{kód} D$

A végső pontosság és teljesség tehát:

$$\bar{P} = \frac{\sum_{kód} A}{\sum_{kód} A + \sum_{kód} B} \qquad \textbf{4.1.4.}$$

$$\bar{R} = \frac{\sum_{kód} A}{\sum_{kód} A + \sum_{kód} C} \qquad \textbf{4.1.5.}$$

Makroátlagolás esetén a már kiszámolt P és R értékeket átlagolnánk, így minden kód egyenlő súlyt kapna. Ez nem szerencsés, hiszen a gyakrabban előforduló kódok fontosabbak, így mikroátlagolást fogok használni, ahogy a versenyben is tették.

A szakdolgozatban tárgyalt probléma a fenti általános leíráshoz képest speciális megszorításokkal rendelkezik. A következőkben az alfejezet elején bevezetett jelöléseket használom.

Minden diagnózishoz csak egy kód tartozik, így $\sum_{kód} A + \sum_{kód} C = t$, illetve $\sum_{kód} A = s$. Ebből következik, hogy a mikroátlagolt teljességérték megegyezik az „intuitív” találati aránnyal.

$$\bar{R}(h) = M(h) \qquad \textbf{4.1.6.}$$

A visszaadott találati lista jelen esetben mást jelent, mint egy diagnózisonként több kódot használó modell esetén (ahogyan például a versenyben volt), emiatt a pontosságértékek itt nem érdekesek. Ugyanis a pontosság nevezőjében található kifejezés

$$\sum_{kód} A + \sum_{kód} B = \bar{h} \cdot t \qquad \textbf{4.1.7.}$$

Ahol $\bar{h}$ a visszaadott lista átlagos hossza[3]. A pontosság így:

$$\bar{P}(h) = \frac{M(h)}{\bar{h}} \qquad \textbf{4.1.8.}$$

Így a pontosság javítása érdekében az osztályozót arra kéne biztatni (a sikerarány növelésén túl), hogy kevesebb kódot adjon vissza átlagosan. Ez azonban nem célunk, a beállított h értékkel már szabályoztuk, hogy hány találattal vagyunk hajlandók foglalkozni. Ebben az esetben tehát a pontosság nem ad többletinformációt.

Összefoglalva tehát a találati arányt (teljesség, recall) fogom mérni h hosszúságú listát figyelembe véve, és ezt $R(h)$-val fogom jelölni.

[3] Az osztályozó nem köteles kitölteni a teljes h hosszúságú listát, kevesebb találatot is visszaadhat..

5 Osztályozó algoritmusok

A következőkben sorra veszem a dokumentumosztályozás legnépszerűbb megoldásait, modelljeit, algoritmusait.

5.1 Előfeldolgozás

A gépi tanulási módszerek legtöbbje vektorbemenetet igényel. Meg kell oldani tehát valamilyen módon a diagnózisok vektorrá alakítását, az úgy nevezett jellemzőkinyerést (*feature extraction*).

Dokumentumosztályozásban leggyakrabban az úgynevezett szóhalmaz (*bag of words*) reprezentációt használják, melynek lényege, hogy a szavak sorrendjétől eltekintünk, és csupán azzal jellemezzük a dokumentumot, hogy mely szavak szerepelnek benne. Ez esetünkben jó absztrakció, hiszen a rövid diagnózisok különösebb struktúra nélkül, névszói csoportokból (*noun phrase*) állnak, melyeknél a szavak sorrendje viszonylag kevés információt hordoz. Példák:

- „Metastasis glandulae suprarenalis”
- „Táplálkozási hiányállapot, k.m.n. Inanitio”
- „Senilitas”

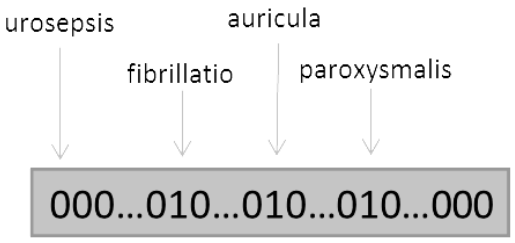
Hosszabb dokumentumoknál értelme lehet a szavak előfordulási számát is figyelembe venni, azonban a diagnóziskódolási feladatban nem jellemző, hogy egy szó többször is előforduljon egy adott diagnózisban, így az egyszerűség kedvéért csak a szavak jelenlétével vagy hiányával fogok dolgozni.

A vektorizálás folyamata tehát a következőképp néz ki:

1. A diagnózist szavakra bontjuk
2. A szavakat megtisztítjuk
3. Elkészítjük a vektort, melyben minden elem (koordináta) egy-egy szó jelenlétét vagy hiányát jelzi.

A szavakra bontásnál meg kell határozni az elválasztó jeleket, melyek új szó kezdetét jelzik. Ez a tanítóminta átnézésével viszonylag egyszerűen megállapítható. A szavak tisztításakor eltüntetünk bizonyos lényegtelen jellemzőket, például kisbetűsítjük a szót, a végéről levesszük a pontot stb. Itt van lehetőség például szótövezésre is,

jellemzően a szóvégi toldalékok eltávolítására[4]. A vektor elkészítésekor a tanítóhalmazban szereplő összes szó kap egy indexet, ami azt adja meg, hogy az adott szónak a vektorban hányadik pozíció felel meg. Ennek alapján a vektorizálandó diagnózisunkban szereplő szavakhoz tartozó vektorpozíciókba 1-est írunk, a többi pozícióban 0 áll (lásd 5.1.1. ábra).



5.1.1. ábra: A "fibrillatio aricula paroxysmalis" diagnózis vektoros reprezentációja bináris szóhalmaz-modellel

Az így kapott vektorok úgy nevezett „ritka vektorok” (*sparse vector*) (kb. 1-8 nem nulla komponensük van a több ezerből). Hogy spóroljunk a memóriával illetve a számítási idővel, ritka vektoroknál célszerű csak a nem nulla komponenseket eltárolni, a többit implicite 0-nak tekinthetjük.

E transzformáció után tehát a következő formát ölti a tanítóhalmaz:

$$\mathcal{D} \subseteq \{0;1\}^n \times C$$

Ahol n a vektorok dimenziószáma (n szó szerepel a tanítóhalmaz diagnózisaiban), C pedig a BNO-kódok halmaza.

A továbbiakban négy osztályozótípust fogok bemutatni, melyeket (az SVM-et kivéve) saját kezűleg implementáltam, és alkalmaztam a diagnóziskódolási feladatra.

5.2 Vektortér-modell

Az szóhalmaz modellel kapott vektorokat közvetlenül az *n*-dimenziós térben kezelve jutunk a vektortér-modellhez. Az lesz az alapfeltevésünk, hogy az azonos osztályba tartozó vektorok egymáshoz közel helyezkednek el, míg a különbözők egymástól távol. Ezt az irodalomban „folytonossági hipotézis”-nek nevezik [16].

[4] A német mintahalmazhoz szótővezett és morfológiailag elemzett verzió is rendelkezésemre áll, így ennek hatása az *Eredmények* fejezetben megtekinthető

Ha egy új vektort kell osztályozni, akkor sorra vesszük a tanításkor eltárolt, osztályozott vektorainkat, majd a leghasonlóbbakhoz tartozó kódokat adjuk vissza, a hasonlóság sorrendjében.

Először is választani kell egy vektorok közötti hasonlósági mértéket. Természetes módon adná magát, hogy az euklideszi távolságmértékből induljunk ki, azonban dokumentumosztályozásra léteznek jobban működő mértékek is. Legelterjedtebb a koszinuszos hasonlóságmérték [16], én is ezt fogom alkalmazni. Két vektor koszinuszos hasonlósága a bezárt szögük koszinusza. A *v* és *w* vektor esetén képlettel:

$$cosSim(\boldsymbol{v}, \boldsymbol{w}) = \frac{\boldsymbol{v} \cdot \boldsymbol{w}}{\|\boldsymbol{v}\| \cdot \|\boldsymbol{w}\|} \qquad \textbf{5.2.1.}$$

Bináris szóhalmazmodell esetében két diagnózis között a koszinuszos hasonlóság jelentése szemléletes:

$$cosSim = \frac{közös\ szavak\ száma}{a\ két\ diagnózis\ szószámának\ mértani\ közepe} \qquad \textbf{5.2.2.}$$

Ez tehát egy nagyon egyszerű összehasonlításon alapuló módszer, ami több szempontból sem megfelelő ebben a formában.

Egyrészt minden dimenziót (szót) azonos módon kezel. Célszerű lenne különbséget tenni közöttük, és az összehasonlítás során jobban figyelembe venni a „fontosabb", jobb megkülönböztető erejűeket.

Másrészt hatékonyság szempontjából: a tanítóhalmaz több tízezer mintából áll, melyeket az eddigiek szerint mind végig kellene járnunk, hogy meghatározzuk a kódolandó diagnózishoz képest vett hasonlóságukat, hogy végül felállítsuk köztük a sorrendet. Így a módszer kiértékeléskor nagyon lassú lenne.

5.2.1 IDF-súlyozás

Az egyes szavak fontosságának megállapítására a szövegbányászatban gyakran használt módszer az IDF (*inverse document frequency*) tényező számítása. Ez a tényező azt jellemzi, hogy a dokumentumok hányadrészében szerepel egy adott szó (a vektorok hányadrészében szerepel 1-es egy adott pozícióban). Ha egy szó gyakran szerepel, az viszonylag jó indikátor arra, hogy az adott szónak nem túl jó a „megkülönböztető ereje", nem olyan fontos.

Az IDF súlyozás képlete (a *v* bemeneti vektor *j*-edik komponensének új értéke, *D* az összes bemenetek halmaza):

$$\boldsymbol{v'}_j = \boldsymbol{v}_j \cdot \log\left(\frac{|D|}{\left|\{\boldsymbol{w} \in D \mid \boldsymbol{w}_j = 1\}\right|}\right) \qquad \textbf{5.2.3.}$$

Ha például egy szó minden diagnózisban szerepel, annak a súlya 0 lesz.

A képletben szereplő logaritmus csökkenti a súlyozás erőteljességét. Például, ha egy szó csak egy-két diagnózisban szerepel, attól még nem lesz óriási súlya, lassabban – csak logaritmikusan – növekszik a súly a gyakoriság csökkenésével. A logaritmus alapja nem számít, hiszen változtatása minden komponensre azonos szorzót jelent, és így nem befolyásolja a kapott vektorok között bezárt szöget.

Ez a módszer első ránézésre puszta heurisztikának tűnik, azonban számos elméleti érvelés is született, ami a sikerét alátámasztja, legtöbbjük valószínűségi és információelméleti alapokon. Ezeket foglalja össze [17].

5.2.2 Invertált index

A hatékonysági probléma megoldására invertált indexet [18] fogok alkalmazni, melynek segítségével nem járom végig azokat a vektorokat, melyekre a hasonlóság mértéke 0.

Az invertált index logikailag nem más, mint egy lista, melynek *i*-edik eleme egy halmaz: Ez a halmaz azon mintákat (azaz bemeneti vektor, osztály párokat) tartalmazza (illetve mutatókat rájuk), melyeknél az *i*-edik bemeneti vektorpozícióban nem 0 szerepel. Ebből az indexből lekérdezhető tehát, hogy mely tanítómintákban szerepel egy adott (*i*-edik) szó.

Egy új, ismeretlen osztályba tartozó kódolandó $\boldsymbol{v}$ vektor esetén az invertált indexből unió művelettel előállítható az a mintahalmaz, amelyben minden minta szerepel, amelyben legalább egy közös szó van $\boldsymbol{v}$-vel. Ehhez azokat az indexbeli halmazokat kell egyesítenünk, melyek a $\boldsymbol{v}$ vektorban szereplő nemzérus pozíciókhoz tartoznak. Ezután már csak ezekre a mintákra számoljuk ki a hasonlóságmértéket, hiszen ha nincs közös szó, azaz minden vektorpozícióban legalább az egyik vektor 0-t tartalmaz, akkor az 5.2.1. egyenletben a számlálóban szereplő skalárszorzat 0 lesz, így a hasonlóság mértéke is 0.

A diagnózisos adathalmaz jellegzetességeiből adódóan ezzel a módszerrel átlagosan a megvizsgálandó vektorok 96%-át ki tudtam szűrni (német minta), ezáltal

jelentősen javítva a futásidőt. (Az összehasonlítások átlagos száma 75091-ről 3139-re csökkent).

5.3 Naiv Bayes-osztályozás

A Bayes-osztályozás valószínűségi alapon történik. A módszer Thomas Bayes (1701–1761) angol matematikusról kapta a nevét, akitől a Bayes-tételként ismert összefüggés származik. A tétel feltételes valószínűségek számítására vonatkozik, így osztályozási feladatoknál különösen szemléletes az alkalmazhatósága. Az egyes osztályok relevanciájának feltételes valószínűségkénti értelmezése intuitív módon adja magát: egy kód relevanciája legyen az adott kódot jelképező osztály feltételes valószínűsége a kódolandó bemenetet esetén. Ez a feltételes valószínűség a következőképpen számolható (C_i a kód, v a bemeneti vektor):

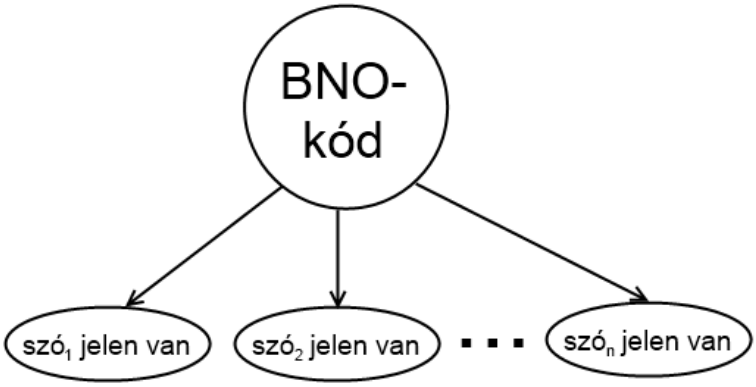
$$\forall i: p(C_i|\boldsymbol{v}) = p(\boldsymbol{v}|C_i) \cdot \frac{p(C_i)}{p(\boldsymbol{v})} \qquad \textbf{5.3.1.}$$

Osztályozáskor ezen feltételes valószínűség alapján rendezzük tehát sorba a kódokat. A jobb oldali nevezőt nem szükséges kiszámítani, hiszen az minden kód esetén azonos, így a relevancia-sorrend szempontjából érdektelen. A képlet többi összetevőjét az osztályozó tanításakor a tanítóhalmaz alapján számítom ki. A $p(C_i)$ kifejezés számítása egyszerű, az adott kóddal ellátott minták arányát kell kiszámolni az összes minta között.

A $p(\boldsymbol{v}|C_i)$ kifejezés azonban értelmezésre szorul. Ha ugyanis ezt a valószínűséget közvetlenül a tanítóhalmazbeli gyakoriság alapján próbáljuk becsülni, azaz az adott kóddal ellátott minták között meg akarnánk számolni, hogy a most osztályozandó bemenet hányszor szerepel, nyilván legtöbbször 0-t kapnánk. Az egyes attribútumok (vektorpozíciók) együttes eloszlását ilyen nagy dimenziójú bemeneti tér esetén tehát nem tudjuk kiszámítani. Egyrészt több milliárd paramétert kellene megtanulni, másrészt hatalmas, a gyakorlatban elképzelhetetlen méretű tanítóhalmazzal kellene rendelkeznünk, hogy meghatározhassuk ezeket a paramétereinket [19]. Együttes eloszlások tömör reprezentációjára kiváló módszer Bayes-hálók alkalmazása [20]. Ennek segítségével feltételes függetlenségek fennállása esetén a paraméterek száma a kiszámítható tartományba vihető.

A vektorok osztályozásánál sajnos nem lehetünk biztosak semmiféle függetlenségben, ezért a paraméterek számának csökkentése érdekében önkényesen kell feltételeznünk bizonyos függetlenségek fennállását. A lehető legtöbb feltételes

függetlenséget feltéve kapjuk a naiv Bayes-hálót. Ebben az osztályváltozót feltéve a bemeneti vektorok komponensei páronként függetlenek egymástól. Esetünkben másképp fogalmazva: egy konkrét kód esetén nincs összefüggés az egyes szavak megléte vagy nem léte között (egyik szó jelenléte sem valószínűsíti vagy valószínűtleníti egy másik szó jelenlétét).



5.3.1. ábra: Naiv Bayes-háló a diagnóziskódolási problémához

Ez a feltevés már lehetővé teszi az együttes eloszlás meghatározását: az egyes vektorkomponensekre vonatkozó feltételes valószínűségek szorzatát kell képeznünk:

$$p(\boldsymbol{v}|C_i) = \prod_j p(v_j|C_i) \qquad \textbf{5.3.2.}$$

Az így működő osztályozót naiv Bayes-osztályozónak nevezik és egyszerűsége illetve viszonylag jó eredményei miatt széles körben alkalmazzák, a dokumentumosztályozásban az egyik standard eszköznek számít. Gyakran előfordul, hogy egy-egy probléma megoldására bemutatott módszerek értékelésénél a naiv Bayes szolgáltatja a kiindulási alapot, ehhez, mint jól ismert módszerhez képest mérik a többi eredményt. A naiv Bayes-osztályozás meglepő módon a függetlenségi feltevés nyilvánvaló hamissága[5] ellenére jól használható. Ezen empirikus eredmény magyarázatára több elméleti indoklás is született, lásd pl. [21]. A magyarázatot általában abban találják, hogy nem szükséges a teljes együttes eloszlást jól megbecsülni ahhoz, hogy jó találati arányt érjünk el. Elegendő, ha a legelső (illetve esetünkben a legelső néhány közül valamelyik) találat megegyezik a valódi együttes eloszlással kapható eredménnyel, még ha a relevancia-számadatok (visszaadott valószínűségek) nem is pontosak. A naiv Bayes-osztályozó többféleképpen is alkalmazható. Ha hosszú

[5] Gondoljunk bele, hogy egy diagnózis esetén egyes kifejezések helyettesíthetők szinonimákkal vagy körülírással, így a szavak nem lehetnek függetlenek, még adott betegség esetén sem.

dokumentumok közül kell egy rövid kereső lekérdezés alapján kikeresni a leginkább relevánsakat, akkor a szavak előfordulásának számát is figyelembe szokták venni (*multinomial naive Bayes*). A diagnóziskódolás esetén azonban a dokumentumok rövidek, így bináris változókat elegendő használni (*multivariate Bernoulli naive Bayes*).

5.3.1 Laplace-simítás

Érdekes kérdés, hogy az 5.3.2. egyenletben a szorzásban melyik szavakra (attribútumokra) hogyan számoljuk ki a szorzótényezőt. Aszerint, hogy egy szó (pl. a *j*-edik) szerepel-e a kódolandó $\boldsymbol{v}$ diagnózisvektorban ($v_j = 1$) illetve a C_i osztályhoz tartozó tanítóhalmazbeli diagnózisvektorok bármelyikében, négyféle eset különíthető el.

1. $v_j = 1$ és a tanítóhalmazban előfordult C_i kódú diagnózisban
2. $v_j = 1$ de a tanítóhalmazban nem fordult elő C_i kódú diagnózisban
3. $v_j = 0$ de a tanítóhalmazban előfordult C_i kódú diagnózisban
4. $v_j = 0$ és a tanítóhalmazban sem fordult elő C_i kódú diagnózisban

Az 1. esetben egy egyszerű gyakorisághányadost kell kiszámolnunk.

$$p(v_j|C_i) = \frac{|\{\boldsymbol{w} \mid w_j = v_j \wedge \mathcal{D}(\boldsymbol{w}) = C_i\}|}{|\{\boldsymbol{w} \mid \mathcal{D}(\boldsymbol{w}) = C_i\}|} \qquad \textbf{5.3.3.}$$

Ahol $\mathcal{D}(\boldsymbol{w}) = C_i$ azt jelenti, hogy $\boldsymbol{w}$ vektor kódja a tanítóhalmazban C_i. Ezt egyszerűbben úgy jelölöm, hogy

$$p(v_j|C_i) = \frac{count(v_j \wedge C_i)}{count(C_i)} \qquad \textbf{5.3.4.}$$

A 2. esetben ugyanezt a képletet alkalmazva a tényező 0 lenne. A kevés mintából adódóan a 2. eset viszonylag gyakran fellép, ugyanis túl sok lehetséges szó vonatkozhat egy adott betegségre, amelyek közül mindet sose fogja tartalmazni a tanítóhalmazunk. A gond az, hogy már egy ilyen 0 tényező elfedi, hogy az adott tényezőtől eltekintve mekkora lenne a szorzat, az így létrejövő 0 relevanciák között pedig nem tudunk sorrendet felállítani. Ezen a gyakorlatban különféle „simításos” módszerekkel lehet segíteni. Legegyszerűbb, ha minden 0 helyett egy konstans kis számmal (simító tényezővel) számolunk. Ennél matematikailag elegánsabb – ha nem is feltétlenül szolgáltat jobb eredményt –, ha nem teszünk ilyen esetszétválasztást, hanem ugyanaszerint a módosított képlet szerint számítjuk minden esetben a feltételes valószínűséget.

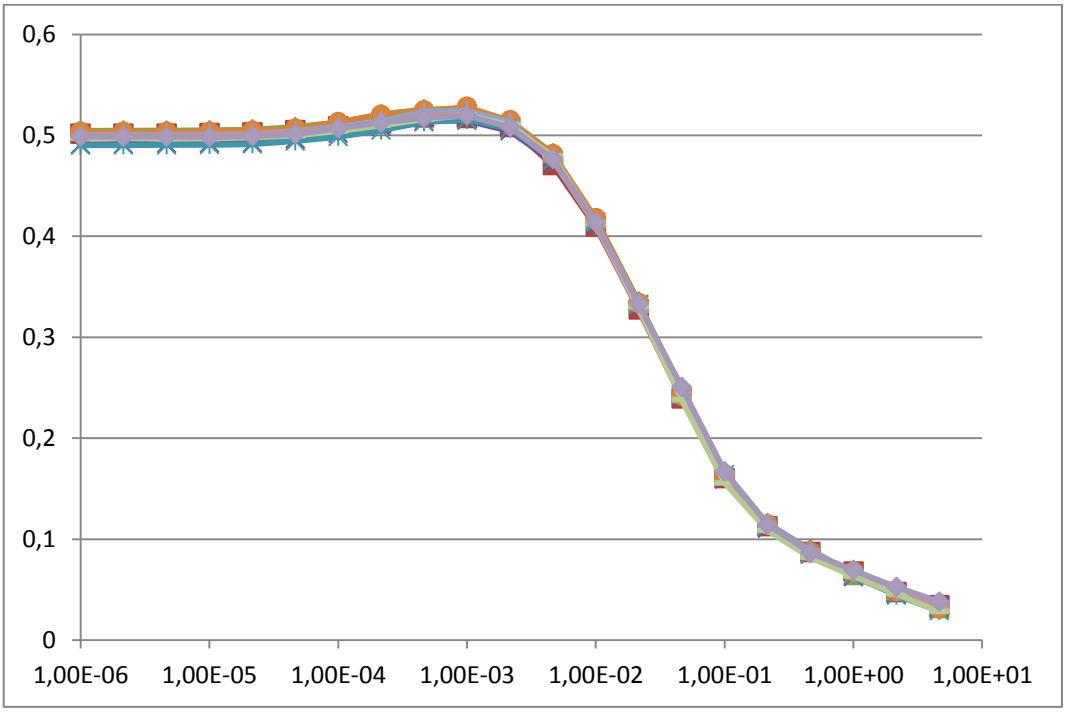
Ilyen módszer a Laplace-simítás. Itt első megközelítésben minden kód relevanciájának megállapításakor úgy teszünk minden szó esetén, mintha a tanítóhalmazban lett volna még két diagnózis az adott kóddal, melyeknél az egyikben benne volt az adott szó, a másikban pedig nem. Ezzel virtuálisan szimmetrikusan bővítjük a tanítóhalmazt (azért két új mintát tételezünk fel, mert az attribútumok binárisak). Ezeket a virtuális mintákat hallucinációknak is nevezik [19]. Az új, simított képlet:

$$p(v_j|C_i) = \frac{1 + count(v_j \wedge C_i)}{2 + count(C_i)} \qquad \textbf{5.3.5.}$$

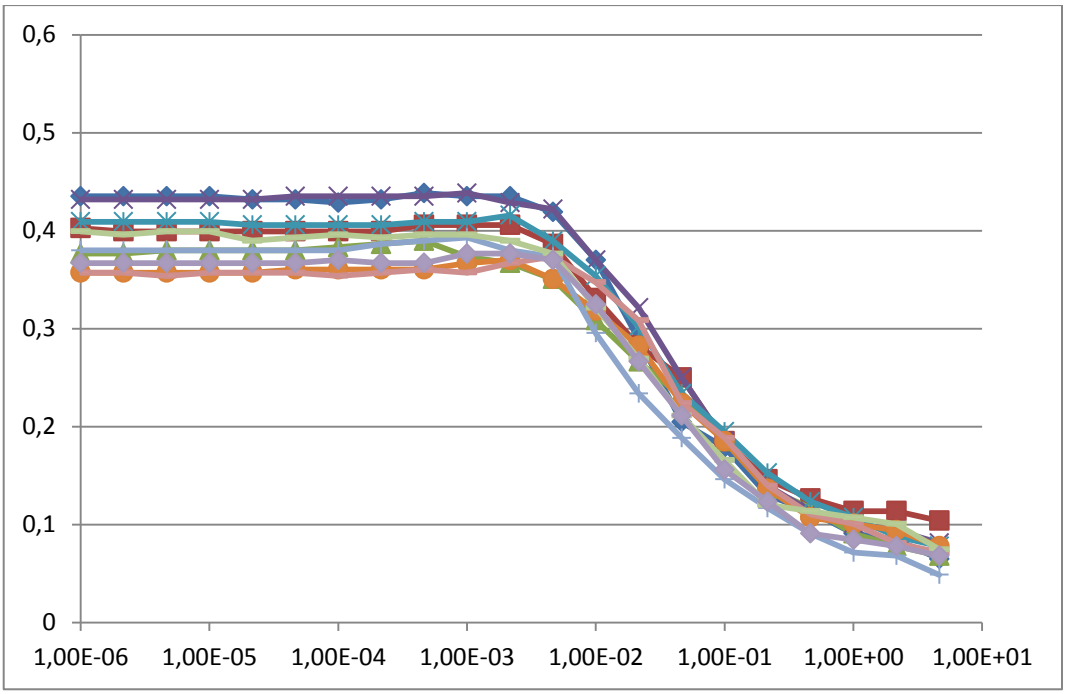
Ez a módszer azonban túl drasztikus ebben a formában. A simítást finomíthatjuk a következőképpen:

$$p(v_j|C_i) = \frac{\theta + count(v_j \wedge C_i)}{2 \cdot \theta + count(C_i)} \qquad \textbf{5.3.6.}$$

Ahol θ egy kis konstans. Így már csak mintegy „tört” számú virtuális, hallucinált diagnózist számítunk hozzá a tanítóhalmazhoz. A θ paramétert empirikusan célszerű beállítani. Ehhez az osztályozó jóságát az elsőként visszaadott kód helyességeként fogom értelmezni. A θ paramétertől függően a következőképp alakul az osztályozó jósága (10-szeres keresztkiértékelési adatsorok).



5.3.2. ábra: A Laplace-simítás paraméterének hatása a naiv Bayes-osztályozó találati arányára, 10-szeres keresztkiértékeléssel (német mintahalmaz). Összehasonlításképp simítás nélkül 0,44 a találati arány.



5.3.3. ábra: A Laplace-simítás paraméterének hatása a naiv Bayes-osztályozó találati arányára 10-szeres keresztkiértékeléssel (magyar mintahalmaz). Simítás nélkül az átlageredmény 0,283.

A nagyobb, német mintahalmazon az egyes keresztkiértékelési eredménysorok szinte szórás nélkül a 0,001 paraméterértéknél veszik fel maximumukat. A magyar minta kis mérete zajt visz a kiértékelésbe, de a görbe lefutása ott is hasonló. Fontos megemlíteni, hogy a paraméter rendkívül kis értékei esetén sem csökken már tovább a találati arány, a simítás nagyon kis paraméter esetén is tartja a grafikonok bal oldalán látható szintet. Ennek oka, hogy ha már egy kicsit is pozitív értékek a feltételes valószínűségeink a szorzatban, akkor különbséget lehet tenni az egyes osztályok között. Ha azonban ténylegesen 0-val számolunk, ez a lehetőség megszűnik, és a találati arány visszaesik.

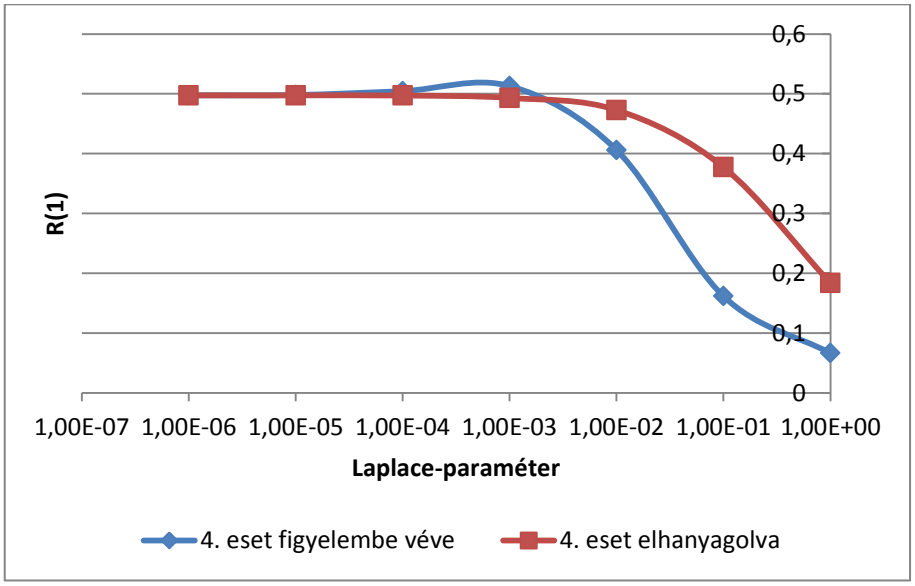
A Bayes-háló modellje alapján az olyan szavakat is figyelembe kell venni, amelyek nem szerepelnek a diagnózisban (3. és 4. eset), hiszen bizonyos szavak hiánya is utalhat a helyes kódra. Ezekben az esetekben is felhasználhatók a fenti képletek (mintha a szó szerepelne a diagnózisban), azonban az eredményt 1-ből ki kell vonni, hogy a szó hiányára vonatkozó feltételes valószínűséget kapjuk. Emiatt elegendő tanításkor az első két eset kiszámításához szükséges valószínűségeket letárolnunk, ebből a második két esethez szükséges számítások is elvégezhetők.

Különösen érdekes eset a 4., amikor olyan szavak által hordozott információt próbálunk megragadni, ami sem a tanítóhalmazban (az adott kóddal), sem a kódolandó

diagnózisban nem szerepel. Ez már elméleti problémákat is felvet, hiszen hány olyan szó létezik, amiről csak annyit tudunk, hogy valahol *nem* szerepel? Ha hűek szeretnénk maradni a vektoros modellhez, akkor a teljes tanítóhalmazban szereplő összes szót kell számba venni, így sokezres nagyságrendű eredményt fogunk kapni. Sok ezer szóhoz kell tehát kiszámolni a Laplace-simított szorzótényezőt, amikkel összesen a következő szorzó adódik:

$$\left(\frac{\theta + count(C_i)}{2 \cdot \theta + count(C_i)}\right)^m \quad m \gg 1000 \qquad \textbf{5.3.7.}$$

Ez egy kicsit nagyobb θ paraméter mellett jelentős torzítást okoz, annak ellenére, hogy egyenként ezek a szorzók alig kisebbek, mint 1. A 4. eset figyelembevételének előnye, hogy bizonyos paraméter esetén jobb eredményt szolgáltat (a már korábban is látott maximum 0,001-nél).



5.3.4. ábra: A 4. eset figyelembevételének hatása az első helyen vett találati arányra, német mintán 5-szörös keresztkiértékeléssel

5.3.2 Logaritmikus megfogalmazás

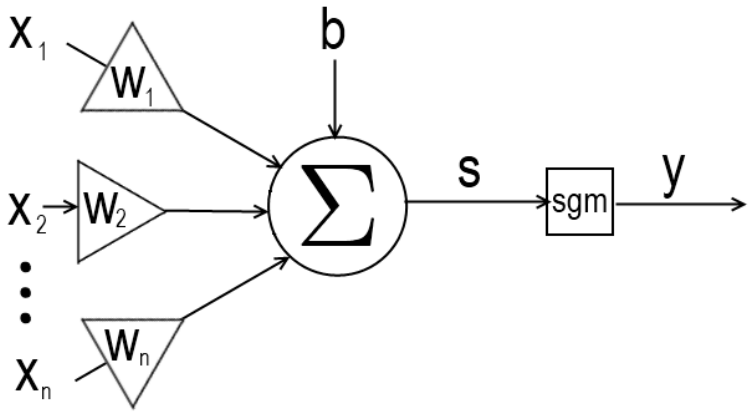
Az 5.3.2. feltételes valószínűség kiszámításához nagyon sok szorzást kell elvégeznünk, főképp a 3. és 4. kategóriás szavakból adódóan. Praktikus okokból ilyenkor célszerű logaritmusok összegével számolni, így elkerülhetjük a lebegőpontos számábrázolásból adódó alulcsordulási jelenséget, bár ilyet nekem szorzásokkal és float típusú változók használatával sem sikerült előidézni. A probléma logaritmikus megfogalmazása azonban elméleti szempontból is érdekes: így láthatóvá válik, hogy a

naiv Bayes-osztályozó tulajdonképpen logaritmikus térben lineáris szeparálást valósít meg [21]. A lineáris (azaz egy hipersík két oldalát megkülönböztető) osztályozás dokumentumosztályozásnál a modell egyszerűsége ellenére is jól működik a nagy dimenziószám miatt.

5.4 Neurális hálózatok: MLP

A (mesterséges) neurális hálózatok az ideghálózat felépítéséből ellesett mintázatokon alapuló tanuló rendszerek. Az alapötlet az 1950-es évek körül fogalmazódott meg, de igazán az 1990-es évektől kezdve terjedt el az alkalmazásuk. A biológiai ihlet nem azt jelenti, hogy konkrétan egy darab idegszövet vagy az agy szimulációja történne, csupán néhány mechanizmust másolnak le, és számos tisztán mérnöki-matematikai megfontolásnak is szerepe van egy-egy háló megtervezésében.

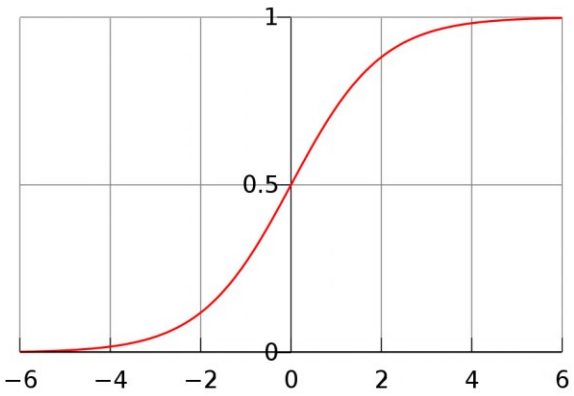
A neurális hálózat különálló, be- és kimenetekkel rendelkező számító egységekből (elemi neuronokból) és ezek összeköttetéseiből épül fel. A különböző hálófajták magyar nyelvű ismertetése [14]-ben található. A dolgozatban szigmoid kimeneti nemlinearitással rendelkező elemi neuronokból felépülő többrétegű perceptron (MLP) modellel fogok foglalkozni. Az elemi neuron a következőképp épül fel:



5.4.1. ábra: elemi neuron, kimenetén szigmoid nemlinearitással

Először az $\boldsymbol{x}$ bemeneti vektor elemeinek egy $\boldsymbol{w}$ súlyvektor szerinti lineáris kombinációjának és egy eltolásértéknek (b) az összegét képezzük, ez az úgy nevezett „inger” (*excitation*). Erre alkalmazunk egy szigmoid függvényt (aktivációs függvény), melynek eredménye a neuron „válasza” (*activation*). A szigmoid nemlinearitás leggyakoribb esetben a logisztikus függvény, mely 0 és 1 közé képzi le a valós számokat. (A –1 és +1 közé leképező szigmoid függvény a tangens hiperbolikusz.)

$$sgm(s) = \frac{1}{1 + \exp(-s)}$$



5.4.2. ábra: Logisztikus függvény

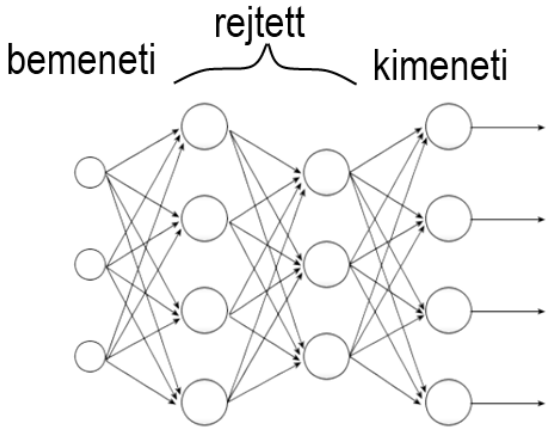
Az elemi neuron kimenete tehát:

$$y(\boldsymbol{x}, \boldsymbol{w}, b) = \frac{1}{1 + \exp(-(\boldsymbol{w} \cdot \boldsymbol{x} + b))} \qquad \textbf{5.4.1.}$$

Ahol · a skalárszorzatot jelöli. A $\boldsymbol{w}$ súlyvektor és a b eltolás megválasztását tanítással végezzük.

5.4.1 Az MLP felépítése

A fent bemutatott elemi neuronokból rétegesen felépülő háló a többrétegű perceptron (*multi-layer perceptron*, MLP), melyben minden réteg neuronjainak kimenete rá van kötve a következő réteg összes neuronjának egyik bemenetére:



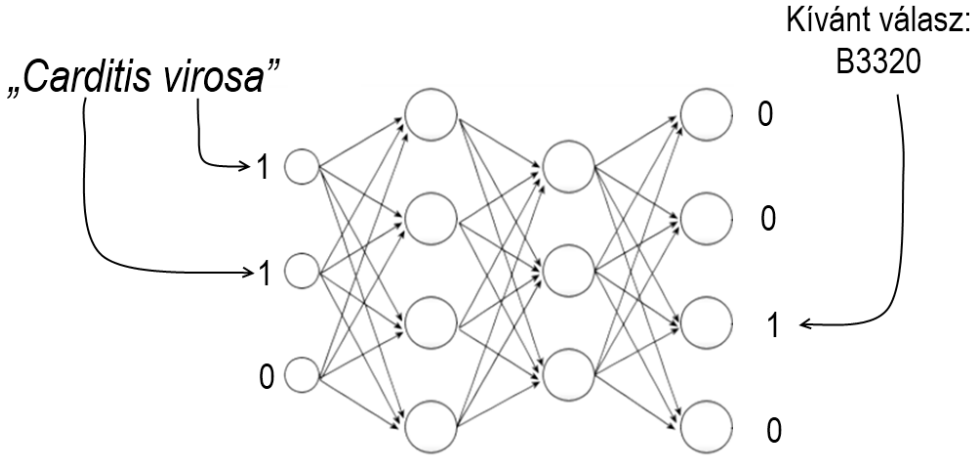
5.4.3. ábra: Többrétegű perceptron (MLP)

Az első rétegben (bemeneti réteg) lévő neuronok processzálást nem végeznek, ezek feladata csak a bemenetek eljuttatása a következő réteg minden neuronjához. Az

utolsó réteg a kimeneti réteg, a köztes rétegeket rejtett rétegeknek hívják. A háló kimenete tehát úgy keletkezik, hogy lépésekben a rétegek között kimenetről bemenetre előre terjednek az értékek, majd a kimeneti réteg kimenete adja a háló végső kimenetét. Ezt a felépítést előrecsatolt hálózatnak hívják.

MLP-vel mind regressziós (függvénybecslő), mind osztályozási feladatok elvégezhetők. Előbbire a különböző módszerek keverésekor fogom alkalmazni.

A diagnóziskódolási feladatban a bementi vektorokat az eddigieknek megfelelően a szóhalmaz modell szerint hozom létre, így annyi neuron lesz a bemeneti rétegben[6], ahány szó előfordul a tanítóhalmazban. A kimeneti réteg annyi neuronból épül fel, ahány BNO-kód van, mindegyik az adott kód relevanciáját próbálja meghatározni. A háló kívánt válasza tehát úgy épül fel, hogy a helyes BNO-kódnak megfelelő helyen 1, az összes többi helyen 0 szerepel:



5.4.4. ábra: MLP alkalmazása a diagnóziskódolási feladatra

A rejtett rétegek és neuronok számára később visszatérek.

5.4.2 Az MLP tanítása: hiba-visszaterjesztés

Az MLP tanítására (azaz a súlyok beállítására) többféle algoritmus létezik. A legnépszerűbbet, a hiba-visszaterjesztéses algoritmust fogom használni. A tanítás ciklusokban (*epoch*) történik. Minden ciklusban sorban felhasználjuk az összes tanítómintát, és minden tanítóminta esetén úgy módosítjuk a súlyokat, hogy a háló válasza javuljon.

[6] A bemeneti réteg általában nem jelenik meg explicit módon az implementációban.

Ehhez definiálni kell a háló válaszának hibamértékét. Legegyszerűbb az átlagos (várható) négyzetes hibával számolni, de később egy másik hibafüggvényt is kipróbálok. A várható négyzetes hiba:

$$J(W) = \mathbb{E}_{\boldsymbol{x}} \left\{ \sum_{j=1}^{n} \left[g_j(\boldsymbol{x}) - y_j(\boldsymbol{x}, W) \right]^2 \right\} \qquad \textbf{5.4.2.}$$

$$g_j(\boldsymbol{x}) = \begin{cases} 1, \text{ha } d(\boldsymbol{x}) = C_j \\ 0, \text{különben} \end{cases}$$

Ahol n a bemenő vektorok dimenziója, $d(\boldsymbol{x})$ a kívánt válasz, C_j a j-edik kód, $y_j(\boldsymbol{x}, W)$ a háló kimeneti rétegének j-edik neuronjának kimenete $\boldsymbol{x}$ bemenet és W súlyok és eltolások esetén. A várható érték a bemenetek eloszlása szerint értendő.

Ha ezt a várható hibát a súlyok terében elképzeljük, akkor egy hibafelületet kapunk, és ennek a minimumát keressük. Szélsőérték-kereső eljárások egész tárháza áll rendelkezésre, ezek közül a legmeredekebb lejtő módszerét (*gradient descent*) szokás alkalmazni az MLP-nél, ami a következő súlymódosítást eredményezi:

$$\Delta w = -\alpha \frac{\partial J}{\partial w} \qquad \textbf{5.4.3.}$$

Azaz minden súlyt úgy módosítunk, hogy a hiba hozzá képest vett parciális deriváltjával ellenkező irányba változtatunk. Ezáltal a súlytérben a hiba gradiensével ellentétes irányba, a legmeredekebb lejtő mentén haladunk. Az α tanulási tényező vagy bátorsági faktor a lépések méretét szabja meg és célszerű empirikusan beállítani.

Az igazi várható értéket természetesen nem tudjuk kiszámolni, mert nem ismerjük az eloszlást és a kívánt válasz is csak adott bemenetekre van meg. A súlymódosítást így a pillanatnyi hibaérték alapján végezzük, azaz az aktuálisan a hálóra tett mintára adott válasz hibáját próbáljuk csökkenteni. Mivel a háló pillanatnyi hibafelülete az egyes bemenetekre nézve különböző, ez nem azonos a várható hibafelület szerinti minimalizálással, de közelíti azt.

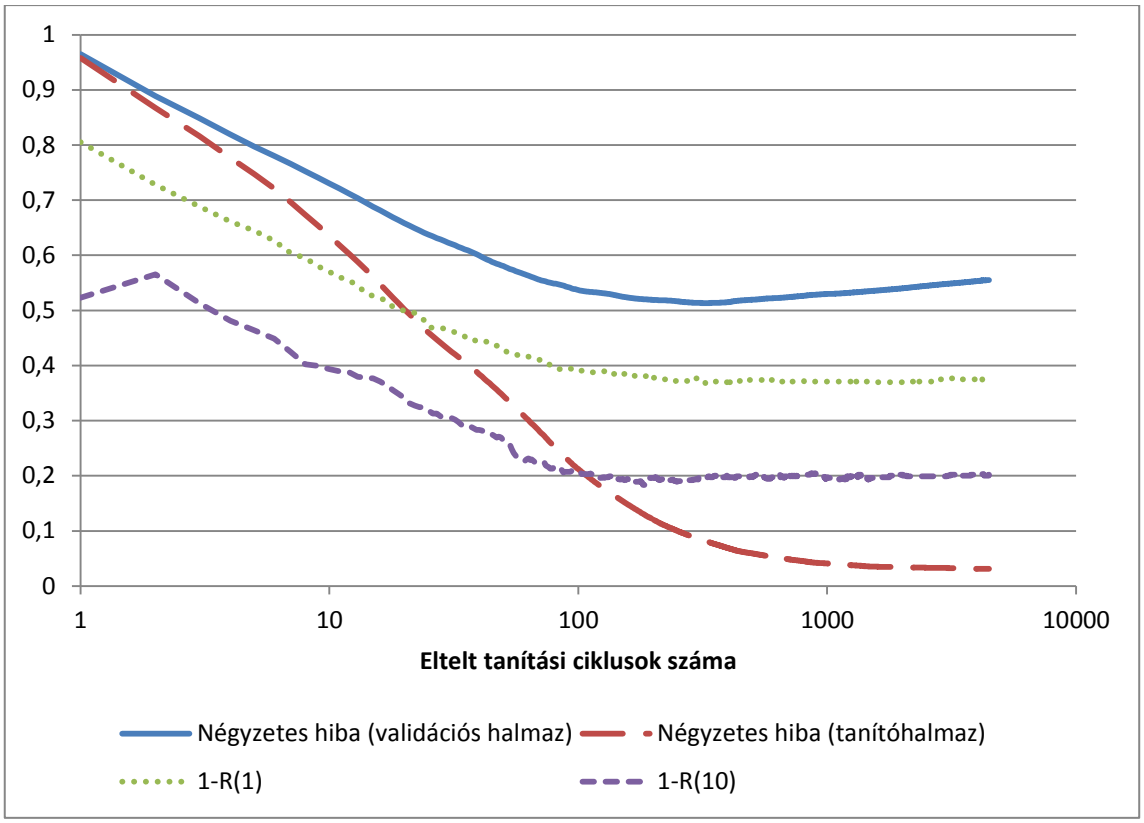
Az MLP esetében a súlyok szerinti parciális deriváltak számításához a deriválás láncszabályának megfelelően minden, az adott súlyra alkalmazott függvény deriváltját sorban össze kell szorozni (ezek a szigmoidok, a súllyal szorzások, illetve a végén a hibafüggvény). Ezt célszerű a kimenettől visszafelé elvégezni, azaz először meghatározzuk a kimenetek szerint vett parciális deriváltakat a négyzetes hiba képlete alapján. Ha tudjuk egy réteg kimenetei szerint vett parciális deriváltakat, akkor ez és az adott réteg bemenete alapján meghatározható a súlyok szerinti parciális derivált és ez

alapján elvégezhető a módosítás. Szintén kiszámolható a pillanatnyi hibának az előző réteg kimenetei szerint vett parciális deriváltja, ami alapján a folyamat ugyanígy folytatható visszafelé. Ezt nevezik hiba-visszaterjesztéses algoritmusnak, melynek részletei [14]-ben elolvashatók. (Az eltolásértékek egy virtuális „$x_0 \equiv 1$” bemenethez tartozó súlyként is felfoghatók, így a fentiek az eltolásokra is vonatkoznak).

5.4.3 A tanítás leállítása

Fontos kérdés, hogy hány ciklust hajtsunk végre tanításkor. Ha túl sokáig tanítjuk a hálót, akkor túltanulás léphet fel, ugyanis az 5.4.2. egyenletben szereplő várható érték helyett csak a tanítópontjaink szerint optimalizálunk. Ennek a szélsőértékét pedig nem érdemes túl pontosan meghatározni, mert annak pontos helye már a tanítómintában „véletlenül” jelen lévő tulajdonságoktól is függ. Intuitíve nem szabad túlságosan bonyolult összefüggést levonni próbálni viszonylag kevés adatból. A torzítás-variancia dilemma [14] szerint az általánosítási hiba két összetevőből áll, melyek jellemzően egymás rovására csökkenthetők. Ha a torzítás kicsi (azaz a modell átlagosan képes jól ráilleszkedni a problémára), akkor a variancia nagy (azaz különbözőképpen mintavételezett tanítóhalmazokra tanítva nagy a szórása a háló becslésének), és fordítva: ha azt szeretnénk, hogy a válasz ne függjön jelentősen attól, hogy épp mely mintákból tanulunk, akkor „rugalmatlan” modellt kell választani, ami eleve képtelen jól illeszkedni a problémára (akár végtelen minta esetén is). Regularizáció nélkül kevés adat és komplex modell esetén túltanulás lép fel.

A tanulás leállítása a modell túl komplexszé válását hivatott megakadályozni, ez egyfajta regularizáció. A túltanulást elkerülendő a tanítóhalmazunkat további két részre kell bontani: egy altanítóhalmazra és egy alteszthalmazra (validációs halmazra). A rendszer az altanítóhalmazt tanulja, és minden ciklus után kiértékelem az alteszthalmazon. Az alábbi grafikon a tanulás lefolyását mutatja.



5.4.5. ábra: MLP tanítása átlagos négyzetes hibakritériummal[7]

Látható, hogy az altanítóhalmazon mért négyzetes hiba mindvégig csökken, azonban a validációs halmazon egy idő után emelkedni kezd – igaz rendkívül lassan (pedig log skála!) –, ekkor indul meg a túltanulás. Szintén leolvasható az ábráról a kapott osztályozó teljesség (recall) értéke az első helyet és az első 10 helyet tekintve (pontosabban ezek invertált, 1-ből kivont értéke). Ezek szerencsére a validációs négyzetes hiba növekedésekor sem növekednek jelentősen, így a tanulás leállításának szerepe ez esetben inkább az időspórolás. Megállapítható, hogy a teljességértékek stagnálása körülbelül a validációs hiba növekedésének kezdetekor indul meg, így ekkor érdemes már leállítani a tanulást. Ennek a pontnak a megtalálására a gyakorlatban egy saját, viszonylag egyszerű módszert használok: Csúszóablakkal figyelem a hiba alakulását, és ha az utóbbi k váltás során a hiba nem csökkent legalább c-szer, akkor nem érdemes tovább folytatni. Ez jobb, mintha az első hibanövekedésnél leállítanám a tanítást, mert így néhány véletlenszerű hibanövekedést még tolerál a rendszer.

[7] Szótövezett német mintahalmaz 6-os főcsoportján (320-359-ig tartó kódok). 80% tanító, 20% validációs halmaz, rejtett réteg nélkül, bátorsági faktor 0,2.

Az alteszthalmazban (validációs halmazban) szereplő minták jobb kihasználása érdekében az optimális ciklusszám meghatározása után ennyi ciklussal újratanítom a hálót, ám ekkor már a teljes tanítóhalmazt felhasználva. Elméletileg lehetséges lenne, hogy az altanítóhalmazon meghatározott optimális ciklusszám a teljes tanítóhalmazon nem megfelelő, azonban látható, hogy az osztályozási hiba a leállás pillanata környékén stagnál, így a leállás pontos helye nem befolyásolja a végső eredményeket.

5.4.4 Maximum likelihood változat

Bár a leggyakrabban alkalmazott hibafüggvény a várható négyzetes hiba, használnak más kritériumokat is, például a tapasztalati maximum likelihood kritériumot, ami [22] szerint jobb lehet, így ezt is megvizsgálom. A módszer lényege, hogy azt a likelihood értéket maximalizáljuk, hogy a megtanított modellünket alapul véve mennyire valószínű a tanítóbemenetek olyan kódolása, ahogyan az a tanítóhalmazban szerepel. Ehhez a háló kimenetén megjelenő válaszokat valószínűségként értelmezzük, azaz:

$$P(G_{\boldsymbol{x}} = c_j)_W = y_j(\boldsymbol{x}, W) \qquad \textbf{5.4.4.}$$

Ahol $G_{\boldsymbol{x}}$ valószínűségi változó $\boldsymbol{x}$ bemenet helyes kódját jelképezi. A kifejezés W szerint nem igazi feltételes valószínűség (így nem is klasszikus likelihood), hiszen a bemenet valamely osztályhoz tartozása nem „függ” a súlyvektoroktól, hanem annak alapján becsüljük (értelmezzük) ezt a valószínűséget, ezért írom alsó indexbe a W-t.

Ezután a valószínűségi feltételezésből kiindulva kiszámoljuk $P(\mathcal{D})_W$-t ahol $\mathcal{D}$ a tanítóhalmazt (azaz a kódok ottani hozzárendelési módját) jelöli. Azt a súlykombinációt keressük, ahol ez a likelihood érték a legnagyobb:

$$\arg\max_W P(\mathcal{D})_W \qquad \textbf{5.4.5.}$$

Élünk azzal a feltevéssel, hogy az MLP függetlennek tételezi fel az egyes bemeneti vektorok kódolását (holott igazából hasonló szavakat tartalmazó diagnózisok között erős függés van), így a teljes tanítóhalmaz valószínűsége az egyes bemenetek ilyetén kódolásának valószínűségének szorzata:

$$P(\mathcal{D})_W = \prod_{\boldsymbol{x}\in\mathcal{D}} P(G_{\boldsymbol{x}} = \mathcal{D}(\boldsymbol{x}))_W \qquad \textbf{5.4.6.}$$

Ahol $\mathcal{D}(\boldsymbol{x})$ az $\boldsymbol{x}$ $\mathcal{D}$-beli kódját jelöli (így $\mathcal{D}$ egyszerre a halmaz és a leképezést is jelöli, de ez mindig egyértelmű). Egy tanítóminta olyan kódolásának valószínűsége,

mint az a halmazban látható, tovább bontható, ha még egy függetlenségi feltevéssel élünk: feltesszük, hogy az MLP azt sem „tudja”, hogy minden diagnózishoz pontosan 1 kód jó, ehelyett az egyes bemenet–kód párok összetartozását egymástól függetlenül tippeli meg:

$$P(G_{\boldsymbol{x}} = \mathcal{D}(\boldsymbol{x}))_W = \prod_j \begin{cases} P(G_{\boldsymbol{x}} = c_j)_W, & \text{ha } \mathcal{D}(\boldsymbol{x}) = c_j \\ P(G_{\boldsymbol{x}} \neq c_j)_W, & \text{ha } \mathcal{D}(\boldsymbol{x}) \neq c_j \end{cases} \qquad \textbf{5.4.7.}$$

A tényezők azt adják meg, hogy milyen valószínű, hogy $\boldsymbol{x}$ bemenethez a j-edik kód tartozik, illetve nem tartozik, aszerint hogy a tanítóhalmaz mit tartalmaz. Ez sokkal egyszerűbben felírható, ha az MLP kimenetén kívánt válaszvektort használjuk, $\boldsymbol{d}(\boldsymbol{x})$-et, amely vektor annyiadik komponense 1, ahányadik kódot rendeli a tanítóhalmaz $\boldsymbol{x}$-hez, a többi 0.

$$\begin{aligned} &P\big(G_{\boldsymbol{x}} = \mathcal{D}(\boldsymbol{x})\big)_W \\ &\qquad = \prod_j \Big(d_j(\boldsymbol{x}) \cdot P\big(G_{\boldsymbol{x}} = c_j\big)_W + \big(1 - d_j(\boldsymbol{x})\big) \cdot \big(1 - P\big(G_{\boldsymbol{x}} = c_j\big)_W\big)\Big) \\ &\qquad = \prod_j \Big(d_j(\boldsymbol{x}) \cdot y_j(\boldsymbol{x}, W) + \big(1 - d_j(\boldsymbol{x})\big) \cdot \big(1 - y_j(\boldsymbol{x}, W)\big)\Big) \end{aligned}$$

5.4.8

Érdemesebb inkább a log-likelihood-dal számolni, mert az összegzés jobban kezelhető, a maximumhelye pedig ugyanott lesz, így a teljes maximalizálandó kifejezés:

$$\log P(\mathcal{D})_W = \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{D}}} \sum_j \log\Big[d_j(\boldsymbol{x}) \cdot y_j(\boldsymbol{x}, W) + \big(1 - d_j(\boldsymbol{x})\big) \cdot \big(1 - y_j(\boldsymbol{x}, W)\big)\Big]$$

A hibafüggvény ennek a mínusz egyszerese lesz tehát, azt fogjuk minimalizálni. Felhasználva, hogy $d_j(\boldsymbol{x})$ csak 0 vagy 1 lehet, a logaritmusok is átrendezhetők, és $d = 1$ illetve $d = 0$ behelyettesítéssel ellenőrizhető a következő átalakítás:

$$J(W) = -\sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{D}}} \sum_j \Big\{d_j(\boldsymbol{x}) \cdot \log\big[y_j(\boldsymbol{x}, W)\big] + \big(1 - d_j(\boldsymbol{x})\big) \cdot \log\big[1 - y_j(\boldsymbol{x}, W)\big]\Big\}$$

5.4.9

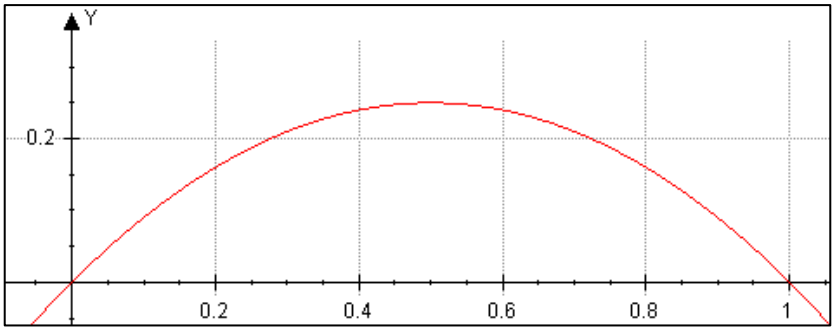
Ezt a kritériumot használva a hibavisszaterjesztéses algoritmusnál a kimeneti rétegre más hibát kell beterjeszteni, azonban utána a visszaterjesztés azonos módon

folyik, mint a várható négyzetes hiba kritériumfüggvénynél. Ha logisztikus nemlinearitást használunk a kimeneti rétegen, akkor belátható, hogy a kimeneti rétegen:

$$\delta_j = \frac{\partial J}{\partial s_j} = y_j - d_j \qquad \textbf{5.4.10.}$$

Ahol s_j a kimeneti réteg j-edik neuronjának ingere.

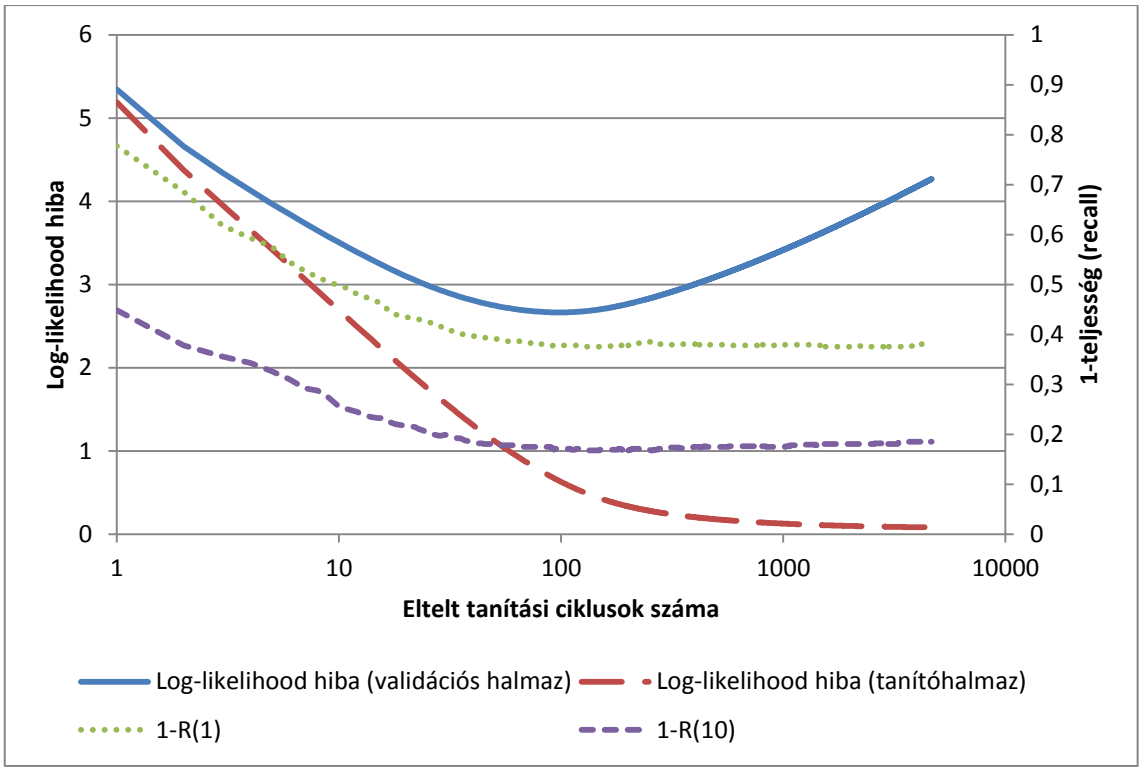
Ezt összevetve a négyzetes hibát minimalizáló változattal (pl. [14]-ben), látható, hogy a különbség egy $y_j(1 - y_j)$ szorzó hiánya. Ez tényező a visszaterjesztés során a súlymódosításokat aszerint befolyásolja (négyzetes kritérium esetén), hogy a megfelelő kimeneti neuron mennyire állt be 1 vagy 0 szintre, azaz osztályozási feladatnál mennyire egyértelműen tud dönteni.



5.4.6. ábra: Az $f(x) = x(1 - x)$ függvény

Ez a szorzó akkor is kicsi, ha a neuron hibásan adja a 0 vagy 1 közeli értéket, így a súlytérbeli hibafelületen fennsíkok alakulhatnak ki, amik a tanulást megnehezítik [22]. Ha elhagyjuk ezt a szorzót, akkor jobban kezelhető hibafelületet kapunk, ez tehát a módszer alkalmasságának másik elméleti alapja (az első a likelihood maximalizálási gondolatmenet volt).

Így tanítva a hálót a következőképp néz ki a tanítási folyamat:



5.4.7. ábra: MLP tanítása maximum-likelihood szabállyal[8]

Látható, hogy nagy vonalakban hasonló a kép a négyzetes kritériummal végzett kísérlethez. Két fő különbség van:

- A konvergencia gyorsabb, az osztályozási hiba kevesebb ciklus alatt eléri a végleges minimumát.
- A validációs halmazon mért hiba erősebben nő, nagyobb mértékű a túltanulás. Ennek intuitív oka valószínűleg az, hogy a négyzetes esetben a fent említett $y_j(1-y_j)$ szorzó a már 0 vagy 1 közelében lévő kimenetű neuronok súlymódosítását fékezi, míg a log-likelihood esetben ezek is könnyen módosulnak, így a túltanulás is könnyebb. Szerencsére ez az osztályozási hibát már nem növeli jelentősen, csak stagnálás indul meg. Ennek oka az lehet, hogy a túltanulás során már a sorrendek alig változnak, inkább csak a relevanciaértékek túltanulása folyik, értékeléskor pedig csak a visszaadott kódok sorrendjét veszem figyelembe.

Mindezek alapján a maximum likelihood hibakritériumot fogom alkalmazni.

[8] Szótövezett német mintahalmaz 6-os főcsoportján (320-359-ig tartó kódok). 80% tanító, 20% validációs halmaz, rejtett réteg nélkül, bátorsági faktor 0,05.

5.4.5 Rejtett rétegek

A rejtett rétegek legfontosabb hátránya szövegosztályozás esetén a lassúság. Mivel szóhalmaz modellel képzett, ritkán ábrázolt vektorokat teszek a bemenetre, az első processzáló réteg kimenetének kiszámolása gyors, hiszen a bemenő vektor 0 komponenseit a lineáris kombináció képzésekor nem kell figyelembe venni. Ez azonban a további rétegeknél már nem igaz, mert ott már sűrű vektoraink vannak, ráadásul a hibavisszaterjesztés miatt a tanítás is hosszadalmasabb lesz. Sajnos jelenleg nincs olyan matematikai eredmény, ami megmondaná, hogy hány rejtett rétegre van szükség illetve ezek hány neuront tartalmazzanak, így csak próbálgatni lehet.

Tapasztalataim szerint a rejtett rétegben a gyorsabb konvergencia érdekében tangens hiperbolikusz aktivációs függvényt érdemes alkalmazni, azonban még akkor is nagyon lassú a tanítás, ami abból is adódik, hogy a bátorsági faktort csökkenteni kell, különben a háló nem konvergál. Emellett rejtett réteget is alkalmazva semmilyen paraméterezéssel nem sikerült jobb eredményt elérnem, mint rejtett réteg nélkül. Ezeket mérlegelve úgy döntöttem, hogy az egyszerű, lineárisan szeparáló egy aktív rétegű perceptron modellt fogom alkalmazni a továbbiakban.

Keveréses módszereknél regresszióra fogom használni az MLP-t, ott rejtett réteget is felhasználok.

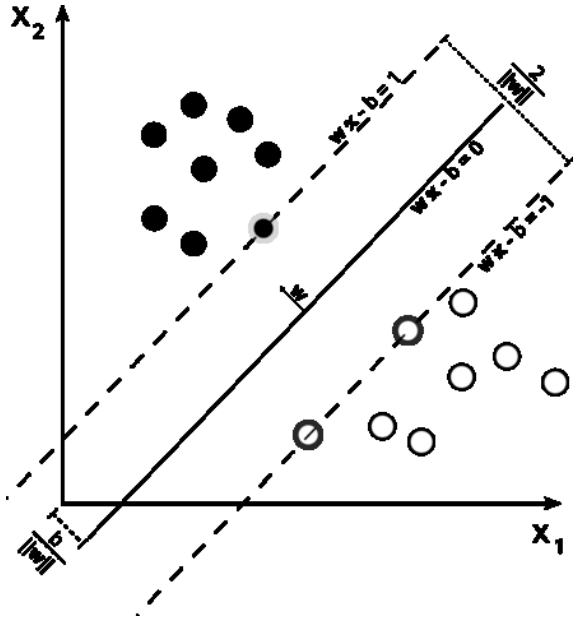
5.5 Szupport vektor gépek

5.5.1 Alapvető működés

Az utóbbi években a gépi tanulás nagy felfedezettje a szupport vektor gép (*Support Vector Machine*, SVM). A módszer első, lineáris változata a [23] cikkben jelent meg. Az azóta eltelt körülbelül két évtized során különböző módosítások és kiterjesztett változatok is megjelentek. Az SVM magyar nyelvű bemutatása megtalálható a [14] könyvben, itt csak röviden foglalom össze a legegyszerűbb SVM működését.

Az SVM osztályozó legegyszerűbb, kétosztályos (bináris), lineáris változata egy jól megválasztott hipersíkkal két részre osztja a bemeneti teret, elválasztva a két osztályba tartozó tanítópontokat. Ez természetesen csak lineárisan szeparálható problémáknál lehetséges. Az SVM konkrétan azt a hipersíkot keresi meg, amely a lehető legnagyobb tanítópontok nélküli üres részt – biztonsági sávot – biztosítja a

hipersík két oldalán (lásd 5.5.1. ábra). Ez nem puszta hasraütéses heurisztika, a statisztikus tanuláselmélet alátámasztja a módszer jó működését.



5.5.1. ábra: Lineárisan szeparálható kétdimenziós bináris probléma megoldása lineáris SVM-mel. Forrás: [24]

Ehhez egy feltételes optimalizálási feladatot kell megoldani. Legyen a következő a tanítóhalmazunk:

$$\mathcal{D} = \{(\boldsymbol{x_i}, d_i) \mid \boldsymbol{x_i} \in \mathbb{R}^n, d_i \in \{-1; 1\}\} \quad i = 1..P \qquad \textbf{5.5.1.}$$

Az elválasztó hipersík egyenletét jelöljük a következőképpen:

$$\boldsymbol{w} \cdot \boldsymbol{x} - b = 0 \qquad \textbf{5.5.2.}$$

A pont a skalárszorzást jelöli, $\boldsymbol{w}$ a hipersík normálvektora, b az eltolása. Ezeket a paramétereket kell úgy beállítani, hogy a margó a lehető legnagyobb legyen. Mivel a $\boldsymbol{w}$ és b paraméterek szabadon szorozhatók az 5.5.2. egyenlet értelmének megváltoztatása nélkül, valamint mivel a hipersík a két margó között félúton helyezkedik el, az általánosság korlátozása nélkül kijelölhetjük margókként a következő egyenletű hipersíkokat:

$$\begin{aligned} \boldsymbol{w} \cdot \boldsymbol{x} - b &= 1 \\ \boldsymbol{w} \cdot \boldsymbol{x} - b &= -1 \end{aligned} \qquad \textbf{5.5.3.}$$

E két hipersík távolsága $\frac{2}{\|\boldsymbol{w}\|}$, így a normálvektor abszolút értékét kell minimalizálnunk, úgy hogy közben minden tanítópont a biztonsági sávon kívül, a megfelelő oldalon legyen, azaz minden i–re teljesüljön:

$$d_i(\boldsymbol{w} \cdot \boldsymbol{x}_i - b) \geq 1 \qquad \textbf{5.5.4.}$$

További átalakítások után, Lagrange-multiplikátoros módszerrel (leírása [14] függelékében) egy kvadratikus optimalizálási problémához jutunk:

$$\min_{\boldsymbol{w},b} \max_{\boldsymbol{\alpha}} \left\{ \frac{1}{2} \|\boldsymbol{w}\|^2 - \sum_{i=1}^{P} \alpha_i [d_i(\boldsymbol{w} \cdot \boldsymbol{x}_i - b) - 1] \right\} \qquad \textbf{5.5.5.}$$

Ez további átalakításokkal egy úgy nevezett duális problémává alakítható, ami kvadratikus programozási (QP) eszközökkel megoldható.

Osztályozáskor az optimalizált paraméterekkel felírva a következő lesz a kimenet:

$$y(\boldsymbol{x}) = \text{sign}\left(\sum_{i=1}^{P} \alpha_i d_i \boldsymbol{x}_i{}^T \boldsymbol{x} + b \right) \qquad \textbf{5.5.6.}$$

A nem 0 α-khoz tartozó mintapontokat szupport vektoroknak nevezik, innen ered a módszer neve. Ezek a pontok vannak a legközelebb az elválasztó hipersíkhoz, így ezek „tartják” a megoldást.

A módszer kiterjeszthető lineárisan nem szeparálható feladatokra „lágy margó” használatával, amikor megengedjük, hogy egyes mintapontok rossz helyen legyenek (a hipersík jó oldalán, de a biztonsági sávban illetve a hipersík másik oldalán). Ezeket azonban büntetjük, így már nem csak a normálvektor abszolút értékét, hanem a belógások összegét is beszámítjuk a minimalizálandó kifejezésbe egy C súlytényezővel:

$$J(\boldsymbol{w}) = \frac{1}{2} \boldsymbol{w}^T \boldsymbol{w} + C \sum_{i=1}^{P} \xi_i \qquad \textbf{5.5.7.}$$

Ahol a ξ_i-k szemléletesen a tanítópontok belógásának mértékét adják meg.

Nemlineáris szeparálást úgy valósíthatunk meg SVM-mel, hogy a bemeneti pontokat először egy nagyobb dimenziójú jellemzőtérbe transzformáljuk, majd ezekre a pontokra végzünk lineáris szeparálást, mert ott már nagyobb eséllyel oldható ez meg. A leképezés explicit megadása elkerülhető („kernel trükk”).

Dokumentumosztályozásnál azonban általában a nagy dimenziószámból adódóan elegendő a lágy margójú lineáris SVM használata.

5.5.2 Többosztályos SVM

Az SVM-et eredendően bináris osztályozási feladatokra találták ki, azaz olyanokra, ahol két osztályt kell megkülönböztetni. Mivel a diagnóziskódolási probléma több (több ezer) osztályt használ, az SVM szokásos változata közvetlenül nem alkalmazható. A többosztályú (*multi-class*) problémákra kitalált megoldásokról és történetükről [25] és [26] cikkekben olvashatunk részleteket.

Többosztályú osztályozók előállíthatók több bináris osztályozó együtteséből. Például ha n osztályunk van:

- Megtaníthatunk n db bináris osztályozót, melyből az i-edik azt dönti el, hogy az adott minta az i-edik osztályba tartozik, vagy egy másikba. Ezután abba az osztályba soroljuk a mintát, amelyikhez tartozó bináris osztályozó hipersíkjától a legmesszebb található a bemeneti mintapont.
- Létrehozhatunk $\binom{n}{2}$ db bináris osztályozót, melyek mindegyike két konkrét osztály közül dönti el, hogy melyikbe tartozik *inkább* a bemeneti minta. Tanításkor mindegyik bináris osztályozót csak azokkal a mintákkal tanítjuk, amik az osztályozó által megkülönböztetett két osztály valamelyikéhez tartoznak. Osztályozáskor mindegyik kódpárt „versenyeztetjük” (futtatjuk a megfelelő bináris osztályozót a bemeneti mintára), majd a legtöbbször nyertes osztályt adjuk vissza. Lehetséges a „győzelmek” mértékének számításba vétele is, például valószínűségi alapokon [27].

Ezek a módszerek az SVM-től függetlenek, bármilyen típusú bináris osztályozó kiterjeszthető ilyen módon többosztályos problémát megoldó osztályozóvá. Lehetőség van azonban közvetlenül az SVM algoritmus átfogalmazására is [25], ezáltal az osztályok közti korrelációk is figyelembe vehetők. Ebben a változatban az 5.5.4. egyenlőtlenség helyébe tehát több egyenlőtlenség lép, minden osztályhoz fog tartozni egy. Ez így azonban egy nehezen megoldható és nehezen tárolható kvadratikus problémához vezet, amihez képest hatékonyabb lehet a bináris osztályozókból történő építkezés. Különböző trükkökkel azonban a kutatóknak mégis sikerült dekomponálni a nagy kvadratikus problémát kisebbekre. [25]

5.5.3 Megvalósítások

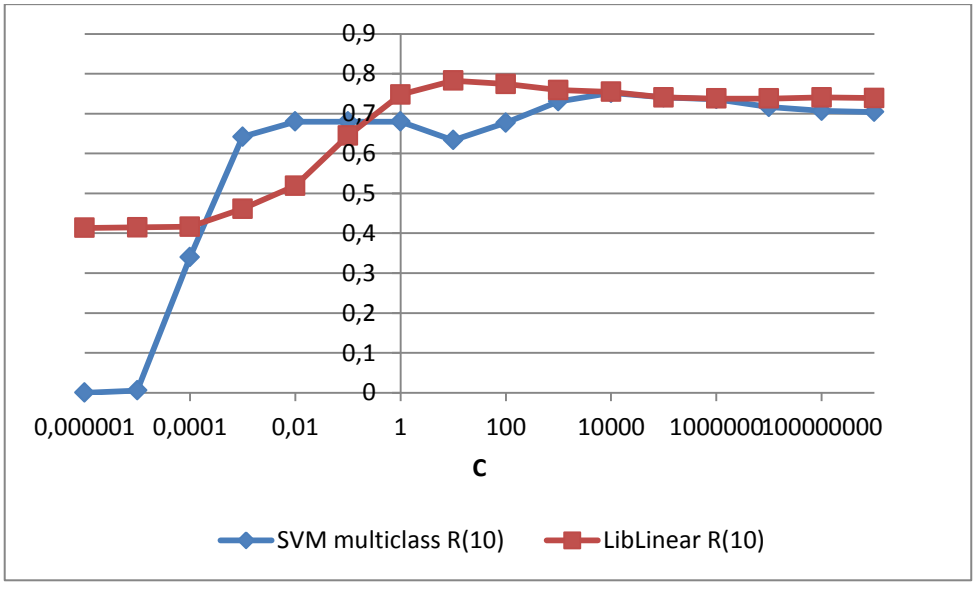
Az SVM esetén nem vállalkozok saját implementáció készítésére, ezt már nagyon sokan elvégezték, akik szakértői a témának.

Kifejezetten sokdimenziós problémákra, például dokumentumosztályozásra optimalizálták a LibLinear [28] nevű lineáris SVM programot, amely ilyen esetekben töredék idő alatt végez a tanítással, mint például a LibSVM általános SVM program. Java könyvtárként is letölthető [29], így egy kis csomagolóosztályon keresztül egyszerűen felhasználható a keretrendszeremben.

Az $SVM^{multiclass}$ nevű implementáció [30] nem bontja bináris osztályozók problémájára a feladatot, hanem egyszerre oldja meg. E különleges tulajdonsága miatt ezt is kipróbáltam (sajnos Java könyvtárként nem adták ki, így fájlokon keresztül történik a kommunikáció egy windowsos parancssori alkalmazással).

Mindkét implementáció képes az egyes osztályok relevanciájának meghatározására, nem pusztán a legvalószínűbb osztályt adják vissza, így felhasználhatók a BNO-kódlista előállítására.

A lineáris SVM-nek egyetlen paramétere van, a biztonsági sávba belógó, vagy az elválasztó hipersík rossz oldalán lévő tanítópontokat büntető C paraméter, melyet empirikusan szokás megválasztani (logaritmikus skálán próbálkozva).



5.5.2. ábra: A C paraméter hatása az osztályozás eredményére a két SVM megvalósítás esetén, az időigényességből adódóan csak a német mintahalmaz 6. főcsoportján mérve (320-359 kódok)

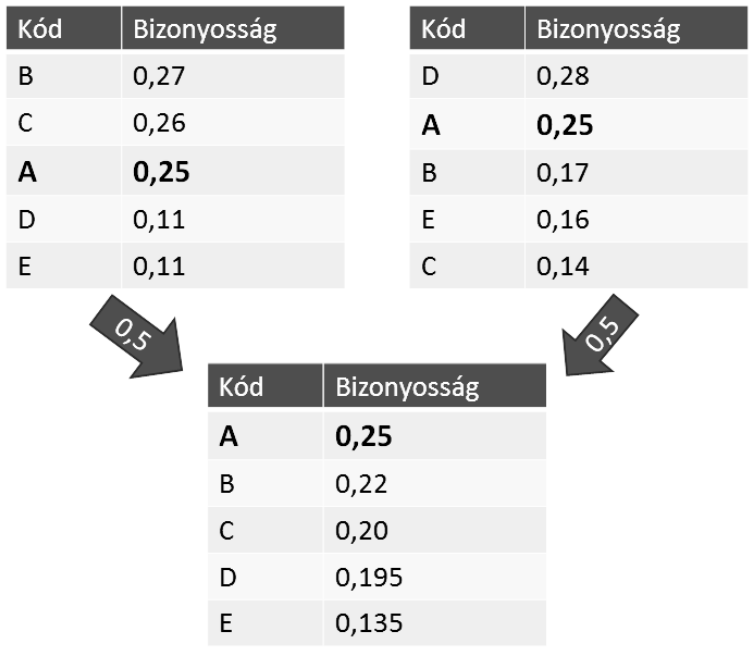
A LibLinear csomag jobb eredményt képes elérni, illetve ezt a megfelelő optimalizációja miatt sokkal gyorsabban is teszi. Emiatt, és mert Java könyvtárként elérhető, a továbbiakban SVM gyanánt a LibLinear csomagot alkalmazom.

6 Keverési módszerek, hibrid modellek

Mint sok más műszaki probléma esetén, a gépi tanulásnál is javíthat a teljesítményen hibrid, kevert modellek használata. A diagnóziskódolási feladatnál több osztályozó eredménylistáját egyesítve jobb eredménylistához juthatunk.

Az egyesítés során valamilyen súlyozás szerint egybevonjuk az eredménylistákat. A kódok végső relevanciája az egyes osztályozók által szolgáltatott relevanciák súlyozott összege lesz. Ezzel két okból is javíthatunk az eredményeken:

1. Ha a hibás kódok különböző osztályozók eredménylistáin különbözőek (vagy a sorrendjük véletlenszerű), a helyes kód viszont a nagyrészükön szerepel, akkor a helyes kód a keverés hatására feljebb kerülhet.
2. Ha az osztályozók közül „eltérő típusú” bemenetek esetén inkább az egyik vagy inkább a másik ad jó eredményt és ezt meg tudjuk előre jósolni, akkor átlagos eredményben a legjobbnál is jobb eredményt érhetünk el (hiszen annak hibázási helyein az inkább megfelelő osztályozó nagyobb súlyt kap).



5.5.1. ábra: Fiktív példa két eredménylista keverésére 0,5-0,5 súlyokkal. A helyes kód az „A” jelű. (A bizonyosságok összege 1-re normalizálva)

Két kérdést kell megválaszolni.

- Miként hozzuk létre az osztályozókat, ha a cél a fenti két pont feltételeinek megteremtése? Az osztályozók különbözősége például a következőkből származhat:
  - eltérő osztályozómodell
  - eltérő tanítókészlet
  - eltérő tanítási paraméterek, kezdeti feltételek
- Mi alapján történjen az eredmények súlyozása?
  - A súlyozást végezheti egy erre létrehozott ún. „kapuzó” rendszer a bemenet vagy az egyes osztályozóktól kapott eredménylisták alapján (osztályozáskor csak ennyi információ áll rendelkezésre)
  - Rögzíthetjük fixen is az egyes osztályozók súlyait.

Hasonló jellegű feladatra találták ki a boosting eljárásokat [14] (több, eltérő mintán tanított gyenge osztályozókból erős osztályozó építése), ám mielőtt ágyúval lőnénk a verébre, érdemes megvizsgálni, hogy körülbelül mit várhatunk a keverésektől.

6.1 Konstans súlyozás

Tekintsük az egyszerűbb esetet először, amikor fix súlyokat határozunk meg. Ezt a következőképp végzem:

1. A tanítóhalmazt két részre bontom (altanítóhalmaz és alteszthalmaz).
2. Az altanítóhalmazzal megtanítom az osztályozóinkat
3. Az alteszthalmazra legenerálom az így megtanított osztályozók találati listáit.
4. Különböző súlyozásokat állítok be, és mindig kiértékelem az így kapott összevont listák átlagos jóságát. A jóság mértéke lehet például az R(1) vagy R(10).
5. Ha a különböző súllyal történt keveréseket kiértékeltem, a legjobb eredményt elérő súlyozást tartom meg.

Ez két osztályozó esetén egyszerű kimerítő kereséssel, azaz lépegetéssel megoldható: legyen a két osztályozó súlya rendre w és $1 - w$. Ezután w értékét 0-tól 1-ig léptetjük valamilyen kis lépéssel, például 0,02-vel, mindig kiértékelve az adott súlyozást. Így lesz 51 eredményünk, amik közül az altesztmintán legjobb eredményt hozót fogjuk megtartani.

Az optimális súly megkeresése kettőnél több osztályozó esetén nehézkes, mert az eredménylisták összevonásával kapott új lista jósága nehezen kezelhető optimalizálási célfüggvény. Ez ugyanis a helyes kódnak az összevont listán elfoglalt pozíciójától függ, ami szakadásos függvényt eredményez. Kifinomultabb optimalizálási eljárások használata felmerülhetne, de amint az *Eredmények* fejezetben látható, két osztályozó esetén ideális esetben is meglehetősen csekély a javulás, így nem valószínű, hogy több osztályozó bevonásával akkora javulás jelentkezne, hogy megérje ezzel foglalkozni.

Vegyük észre, hogy egyre kevesebb tanítómintával dolgozunk, mivel mindig csak független mintákon értékelhetjük ki a rendszereket[9], ha el akarjuk kerülni a túltanulást. Egyrészt fenntartunk általában kb. 20%-nyi mintát az egész módszer végső értékelésére. A maradékból továbbvágunk szintén néhányszor 10%-ot a keverési súlyok kiértékeléséhez. Egy esetleges MLP tag-osztályozónál a maradékot még tovább bontjuk, mert a tanulás leállítását a validációs hiba szerint végezzük. Így sok esetben nem használjuk ki megfelelő mértékben a mintákat. Az eredmények javítása érdekében a következőt tehetjük minden ilyen altanítómintára és altesztmintára bontási szinten:

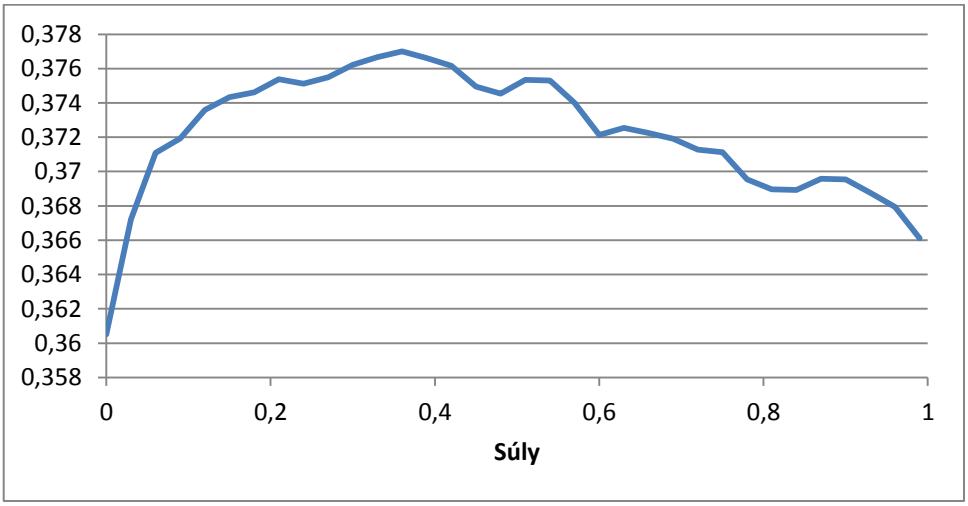
1. Az altanítómintával elvégezzük a tanítást.
2. Az altesztmintán optimalizáljuk a paramétereket (pl. keverési súlyok, MLP leállási ciklusszáma stb.)
3. Az optimális paramétereket megtartjuk és végzünk egy újabb tanítást, immár a teljes tanítóhalmazzal.

A módszer hátránya, hogy nem lehetünk biztosak benne, hogy a 2-es lépésben talált paraméterek megfelelők lesznek a teljes tanítóhalmazzal tanított osztályozó esetén is. Azonban a futtatások során úgy tapasztaltam, hogy így is megéri újratanítani, mert a plusz adatok többet javítanak a rendszeren, mint amennyit az optimális paraméterek körüli bizonytalanság ront.

Fontos kérdés továbbá, hogy milyen arányban vágjuk szét a tanítómintát. Ha túl kevés mintával tanítjuk az osztályozóinkat, akkor nem lesz elég reprezentatív az eredményük a későbbiekre nézve, ha pedig túl keveset hagyunk a kiértékelésre, akkor

[9] Ez persze a legfelső szinten is igaz, így a szakdolgozat-készítés során végzett próbálkozások (mit érdemes, mit nem) is felfoghatók tanulásnak, és itt is felléphet túltanulás (a konkrét adathalmazban lehet olyan specialitás, ami miatt bizonyos módszerek véletlenül működnek jól), ám én is végzek bizonyos „regularizációt”, nem engedem a modellt értelmetlenül komplexszé válni.

nem lesz pontos a kapott súlybecslésünk. Empirikusan a 40% altanítóminta és 60% altesztminta vált be. Ilyen beállítás melletti kiértékelést mutat a következő grafikon, melyen az optimálisnak becsült konstans súly 0,36 lett.



6.1.1. ábra: Konstans súly és az altesztmintán vett eredmény összefüggése (magyar minta) IDF és perceprton MOE (lásd később) keverésekor. A függőleges tengely egy szűk tartományt ábrázol.

6.2 Súlyozásbecslő

Az optimális súlyozás bemenetenként eltérő lehet, ha bizonyos típusú bemenetek esetén inkább az egyik, más típusúaknál inkább a másik osztályozó eredményesebb (azaz a bemeneti tér különböző részein jók). Azt, hogy egy adott bemenet esetén mi lesz a megfelelő súlyozás, nem tudhatjuk előre, azonban egy tanuló rendszernek megtaníthatjuk a tanítóminta alapján. Ha a tanuló rendszer képes összefüggést felderíteni a bemenet és a súly között, lehetséges, hogy jobb eredményt érhetünk el, mint ha konstans súlyokat alkalmaznánk. A következő, egyszerű folyamatot találtam ki:

1. A tanítómintát két részre bontom, egy altanítómintára és egy altesztmintára.
2. Az altanítómintával megtanítom az osztályozókat.
3. Az altesztminta minden elemére megkeresem az optimális súlyozást (két osztályozó esetén kimerítő keresés elegendő).
4. Az altesztminta bemeneteivel és rendre a hozzájuk tartozó optimális súlyozással – amit az előző lépésben határoztam meg – megtanítok egy regressziós (függvényapproximáló) rendszert, konkrétan egy MLP-t.
5. Új bemenet osztályozásakor először az MLP megbecsüli, hogy mi lesz az optimális súlyozás, és eszerint súlyozom az egyes osztályozókat.

6.3 Jóságbecslő

A kimerítő keresés két osztályozó esetén sem gyors, azonban több dimenzióban ez hatványozottan igaz („dimenzió átka”). Jó lenne, ha megkerülhetnénk az optimális súly kiszámítását. A súlyokat például az alapján is becsülhetjük, hogy milyen jó eredményt érnek el az egyes osztályozók az egyes bemeneteken, ehhez nincs szükség keresésre. Részletesen:

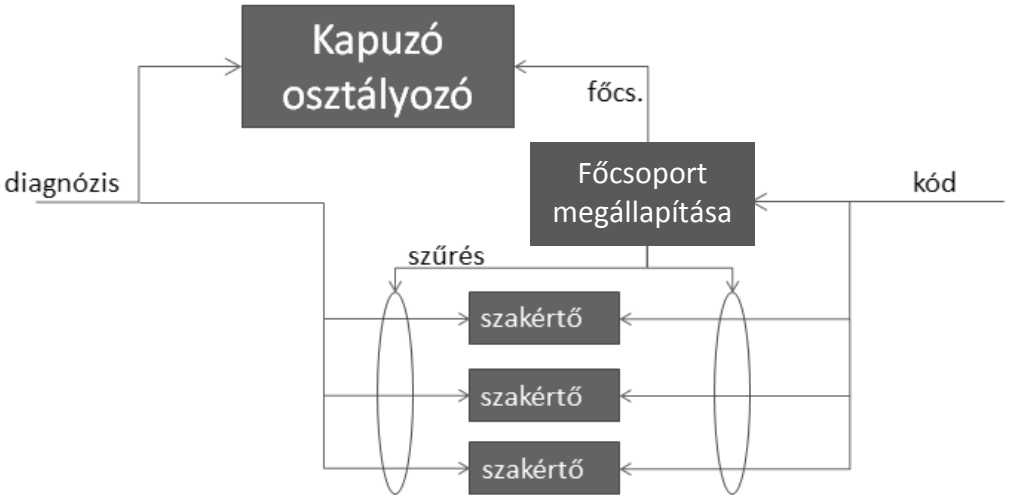
1. A tanítómintát két részre bontom, egy altanítómintára és egy altesztmintára.
2. Az altanítómintával megtanítom az osztályozókat.
3. Az altesztminta minden elemére megállapítom, hogy milyen jó eredményt érnek el az egyes osztályozók (kiértékelem a visszaadott listát).
4. Az altesztminta bemeneteivel és rendre a hozzájuk tartozó jóságeredményekkel – amit az előző lépésben határoztam meg – megtanítok egy regressziós rendszert, konkrétan egy MLP-t.
5. Osztályozáskor az MLP által szolgáltatott jóságbecsléseket súlytényezőkként használom.

A fenti keveréses módszerek valóban javítanak valamennyit az eredményen, azonban sajnos keveset. Részletek az *Eredmények* fejezetben.

6.4 Szakértőegyüttes

Ha rendelkezésünkre áll valamilyen apriori tudás a tárgyterületről, azt mindig célszerű felhasználni. Ilyen információ, hogy a BNO-kódrendszer hierarchikus szervezésű Ennek kihasználásával készíthetünk egy olyan moduláris osztályozót, melyben az egyes részosztályozókat csak egy-egy BNO-főcsoportba tartozó mintákkal tanítunk, ezek lesznek a szakértőink. Az osztályozók különbözősége tehát az eltérő tanítómintákból fog eredni.

Itt is szükség lesz kapuzórendszerre, hiszen egy osztályozandó, ismeretlen bemenet esetén nem tudjuk, hogy melyik szakértő „kompetens” (melyik főcsoportba tartozik majd a kód). A kapuzó osztályozó a főcsoportot tippeli meg adott bemenet esetén. Az így kapott rendszer a szakértőegyüttesnek (*mixture of experts*, MOE) [14] nevezett modell egy változata. A tanítása a következőképp zajlik:



6.4.1. ábra: Szakértőegyüttes tanítása

1. A mintákat szétválogatom főcsoportok szerint. (A főcsoport a kódból állapítható meg. Egy főcsoport egy intervallum, pl. D50…- D89…)
2. A kapuzó osztályozót egy olyan mintahalmazzal tanítom, amely a bemenetekhez a megfelelő főcsoportot társítja.
3. Létrehozom a szakértőket és a megfelelő főcsoportba tartozó mintákkal tanítom őket.

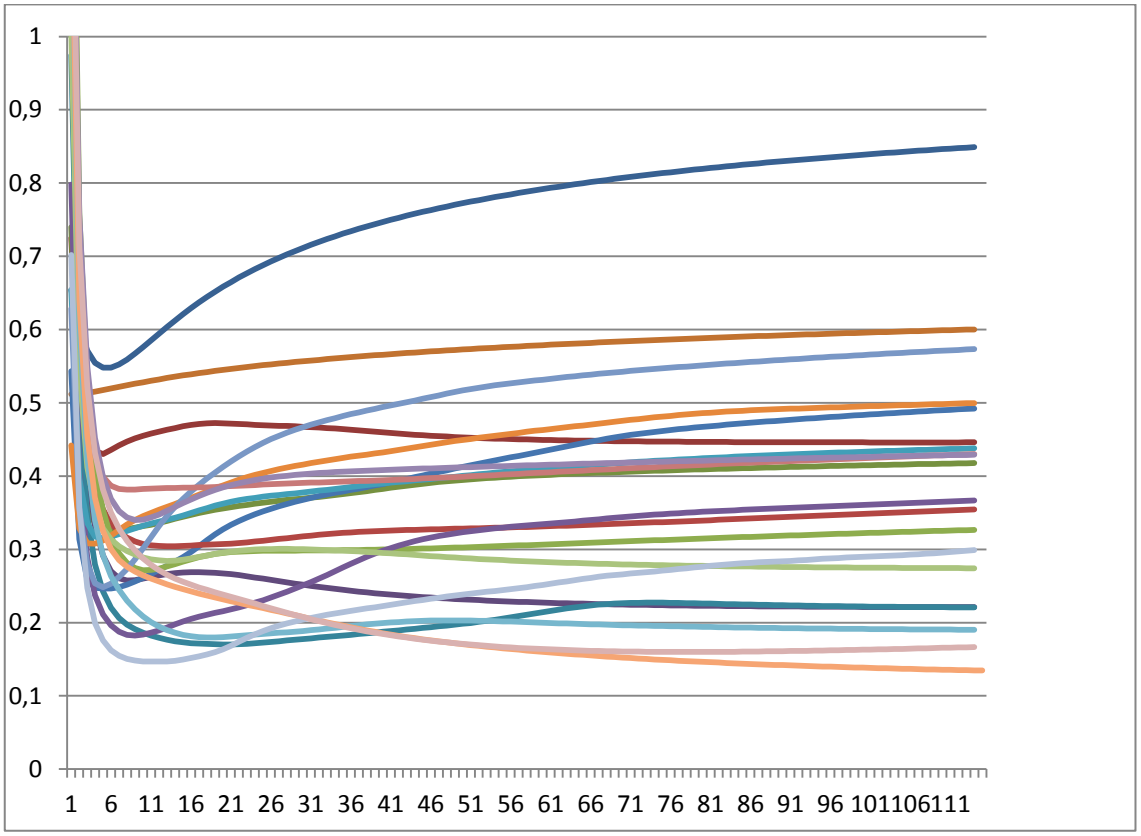
A szakértőegyüttes új, ismeretlen bemenet osztályozásakor:

6.4.2. ábra: Szakértőegyüttes új bemenet osztályozásakor

1. A kapuzó osztályozóval megállapítom az egyes főcsoportok relevanciáját.
2. Minden szakértővel osztályozom a bemenetet, ezáltal relevanciákkal ellátott kódlistákat kapok.

3. Az egyes szakértők eredménylistáit a nekik megfelelő főcsoport relevanciájával (ld. 1. lépés) súlyozva egyesítem.

A szakértőegyüttes alkalmazásának igazi előnye neurális hálózatoknál és SVM-nél, azaz a számításintenzív módszereknél mutatkozik meg. Mivel minden mintát csak a neki megfelelő kis szakértő tanul meg, a tanulás sokkal gyorsabb, mint ha például egy „lapos", egyszerű neurális hálózatot tanítanánk. A nagyságrendek érzékeltetése végett: a német mintán a lapos neurális háló tanítása 109 percig tart, míg a szakértőegyüttes elkészül 12 perc alatt, ráadásul jobb eredményt is ér el. A magyar mintán 54 másodpercről 15 másodpercre csökken a tanítási idő szakértőegyüttes esetén. További időadatok az *Eredmények* fejezetben találhatók.

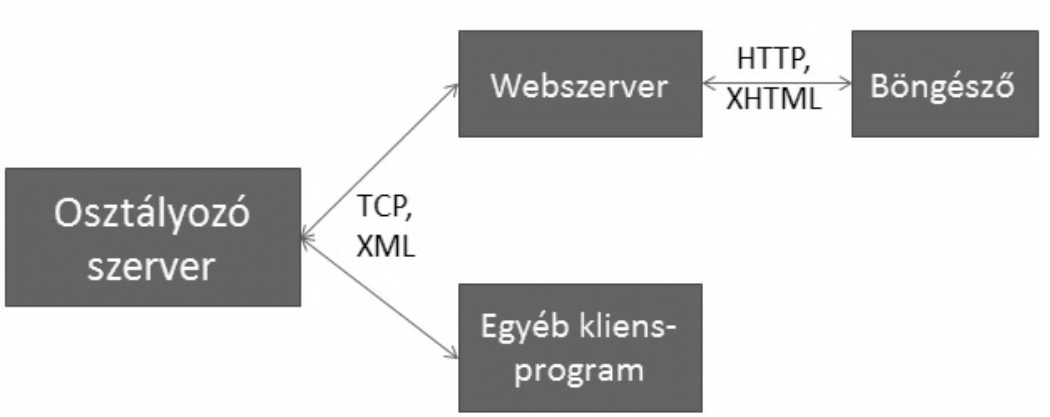


6.4.3. ábra A 21 főcsoporthoz tartozó szakértő perceptronok tanulása (négyzetes hiba a validációs halmazon eltelt ciklusok függvényében). Mindegyik szakértőnél más az optimális ciklusszám, így elengedhetetlen a neurális háló bemutatásakor említett dinamikus leállás

7 Implementáció

A fent bemutatott módszereket (az SVM-et leszámítva) egy saját általános osztályozási keretrendszerben implementáltam és teszteltem. Most röviden bemutatom az elkészült program felépítését, működését. Az implementációt Java nyelven, NetBeans fejlesztőkörnyezetben végeztem. Azért esett a Java-ra a választásom, mert platformfüggetlen, jól dokumentált, gyorsan futó alkalmazás készíthető vele, valamint az objektumorientált szemlélet a keretrendszer moduláris fejlesztését megkönnyíti.

7.1 Alapvető felépítés



7.1.1. ábra: A komponensek közti kommunikáció

A kész rendszer két programból áll. Az első a keretrendszert és az implementált osztályozó módszereket tartalmazza és többféle paraméterezéssel indítható[10]. A főbb paraméterezési lehetőségek

1. Tanítás: meg kell adnunk az osztályozó felépítését definiáló XML fájlt (erről későbbi alfejezetben lesz szó), a tanítóhalmaz fájlját, és hogy hová kerüljön a megtanított osztályozó objektum szerializált változata. Példa az ilyen paraméterezésre:

```
train svm_moe.xml germanSamples.txt svm_moe.cls
```

2. Keresztkiértékelés: meg kell adni az osztályozó felépítését megadó XML fájlt, a mintahalmazt, hány részre vágódjon a minta, és hogy ezek közül hány

[10] További részletek és futtatási példák (batch fájlok) a szakdolgozat elektronikus mellékletében

menetet futtasson és átlagoljon ténylegesen a rendszer. Utolsó argumentumként opcionálisan egy fájlnév adható, ahová CSV formátumban kiírja az eredményeket a program. Példa:

```
crossval svm_moe.xml germanSamples.txt results.csv
```

3. Szerverkénti működés: az első argumentumként megadott szerializált fájlból (amit az 1. pontban említett módon hoztunk létre) betölti az osztályozó objektumot, majd a második argumentumként megadott TCP porton figyel, és az érkező XML nyelven megfogalmazott kérésekre (osztályozandó diagnózisokra) válaszol (megküldi a releváns kódok listáját). Az XML formátumról később írok. Ilyen paraméterezésre példa:

```
serve svm_moe.cls 5555
```

A másik elkészített program egy webszerveren futó JSP alapú weboldal, mely ebben a szcenárióban, az osztályozó szerverhez viszonyítva kliensként jelenik meg. Fejleszthető lenne egyéb kliensprogram is, például kórházi információs rendszerekbe plugin modul, ami a webes felülethez hasonló módon az osztályozó szerverrel kommunikálna.

Az első program logikailag két részből áll. Egy általános keretrendszerből és a konkrét algoritmusok ebben a keretrendszerben történő implementációjából.

7.2 Keretrendszer

A keretrendszer egy általános, osztályozó algoritmusok implementációját és tesztelését lehetővé tevő osztálygyűjtemény. Nem célom a rendszer teljes körű leírása a méretéből adódóan, illetve nem is túl érdekesek bizonyos implementációs részletek[11]. Inkább a működési elvre koncentrálok, és csak a legfontosabb osztályok legfontosabb aspektusait ismertetem. Későbbi alfejezetben UML diagramok is megtekinthetők.

A gépi tanulás „osztály” szavát nem szeretném keverni az objektumorientált „osztály” kifejezéssel, ezért kimenetnek nevezem az osztályozás eredményét (pl. BNO-kód).

[11] A forráskód megtalálható a szakdolgozat elektronikus mellékletében.

7.2.1 Központi osztályok

- `Sample<In, Out>, SampleSet<In, Out>:` Egy mintát (bemenet, kimenet párt) és az ilyenekből összeálló mintahalmazt reprezentáló osztályok. A `SampleSet` metódusai többek között a mintahalmaz véletlenszerű összekeverését, tanító- és teszthalmazra bontását támogatják. A `SampleSet` iterálható, így for-each ciklusokban könnyen kezelhető. Elkérhető külön-külön a bemenetek és a kimenetek listája. A keretrendszer erősen kihasználja a Java generikus típusaiban rejlő lehetőségeket. A `SampleSet` osztálynak két típusparamétere van, `In` a bemenet típusa, `Out` a kimenet típusa. Ezáltal egységesen kezelhetők az osztályozási feladatok, típusra való megkötés nélkül.
- `Result<Out>`, `ResultSet<Out>`: egy (kimenet, relevancia) párt illetve ilyenekből összeálló eredményhalmazt reprezentáló osztályok. A `ResultSet` adott számú eredményt (például amekkora listát a felhasználó lekér) tárol magában, mindig a legrelevánsabbakat, így feleslegesen nem tárol a memóriában sokezer kimenetet és relevanciát. Az osztályozók egy `ResultSet`-et építenek fel, így a maximumkeresés terhét a `ResultSet` leveszi a vállukról, egyszerűen sorban meghívják a `ResultSet push` metódusát az adott kimenettel és relevanciával, az pedig egy megfelelő adatstruktúrában nyilvántartja a legjobbakat. Ezen kívül lekérdezhető tőle például, hogy egy adott kimenetet tartalmaz-e, ill. hányadik helyen, milyen relevanciaértékkel. A `ResultSet` szintén iterálható for-each ciklusokban.
- `Classifier<In, Out>`: az osztályozó algoritmusok ősinterfésze. A `train(SampleSet)` metódussal tanítható, a `classify(In)` metódussal új bemenetet osztályoz és visszaad egy `ResultSet<Out>` eredménylistát. Ez az osztályozó lehető legáltalánosabb definíciója.
- `Vector`: vektorok őse, mely képes adott pozíció lekérésére, változtatására, skalárszorzat, összeg, stb. számítására. Végig lehet iterálni a nem nulla pozícióin egy egyszerű for-each ciklusban. Leszármazottai:
    - `DenseVector`: egy Java tömb körüli csomagoló, például a neuronok súlyvektorai illetve az MLP kimenete számára.
    - `SparseVector`: csak a nem nulla vektorpozíciókban álló értékeket tárolja, például az IDF súlyozott vektortérben.

- `IndicatorVector`: csak 0-s és 1-es értékeket tartalmazhat, így elég az 1-esek pozícióit tárolni. Tipikusan minden szóhalmaz modellel készített vektort ezzel ábrázolok, ahol nincs IDF súlyozás.

7.2.2 Transzformációk

Különböző típusokat kezelő osztályozók között szükség van konverzióra, például sztringek vektorosítására. Ezenkívül más előfeldolgozó lépéseket is lehet végezni, például ilyen az IDF súlyozás vektorok esetén. Erre szolgálnak a keretrendszerben a különféle transzformációk, adapterek. Az UML diagramok között több osztály is megtekinthető.

- `InputTransform`: bemeneti transzformációt végző osztályok ősinterfésze. Képes egy tanítóhalmaz bemenetlistájából egy új reprezentációt kialakítani. Osztályozáskor az új bemeneteket szintén át tudja transzformálni. Tipikus implementálója a `BagOfWordsTransform`, amely tanításkor átkonvertálja a bemeneti listát `IndicatorVector`-okra, eltárolja a felhasznált (szó, index) leképezést, majd osztályozáskor a leképezésnek megfelelően előállítja a megfelelő `IndicatorVectort`. Ilyen ezen kívül az `IDFTransform` osztály is, ami egy bemeneti vektorlistához kiszámolja az IDF súlyokat, majd a megjegyzett súlyokat alkalmazza osztályozáskor is.
- `OutputTransform`: A kimenetek transzformációját végzi. Tanításkor a külső formátumból állítja elő az osztályozó által előállított típusú kimenetet. Osztályozáskor az osztályozó által előállított kimenetet konvertálja a külső formátumra. Például egy neurális osztályozó `Vector-Integer` leképezést hajt végre. Ahhoz, hogy diagnózisokból (`String`) BNO-kódokat (`String`) tudjunk előállítani, körbe kell bástyáznunk a neurális osztályozót bemeneti oldalról egy `BagOfWordsTransform`-mal, kimeneti oldalról pedig egy ún. `CounterOutputTransform`-mal, ami egyszerűen sorszámozza a külső kimeneteket, így állítva elő a neurális háló által tanulható belső, `Integer` kimenetet.
- `ClassifierAdapter`: egy `Classifier`-t és `Input`- illetve `OutputTransform`-okat tartalmaz. Implementálja a `Classifier` interfészt, így egy transzformációkkal körberakott osztályozó bárhol használható, ahol egy egyszerű osztályozó.

7.2.3 Kiértékelés

- `EvaluationMetric`: a kiértékelő mércék ősinterfésze. Képes kiértékelni (egy `double` értékkel) egy eredménylistát (`ResultSet`) a helyes kimenet ismeretében. Ilyen például a `PositionAtMostMetric`, mely az alapján ad 0 vagy 1 pontot, hogy a helyes kód az eredménylistában egy beállított értéknél kevesebbedik helyen van-e. Ezzel számítottam az R(1), R(10) stb. értékeket.
- `Evaluator`: statikus metódusokat tartalmazó osztály, képes kiértékelni, illetve keresztkiértékelni egy osztályozót adott mintahalmazzal illetve kiértékelési mércékkel.

7.2.4 Párhuzamosítás

A keretrendszerben sok tevékenység illetve több osztályozó algoritmus is párhuzamosítható. A kiértékelés tipikusan ilyen, hiszen a tesztminták osztályozása egymástól független. Az algoritmusok közül nagyon jól párhuzamosítható a vektortérbeli keresés és a neurális háló neuronjainak feldolgozása, a hibrid modellek részosztályozóinak tanítása stb. Ez több esetben sokszoros gyorsítást jelent, amit az időspórolás, és a kivárható eredmények miatt megéri implementálni. Sajnos Java-ban a párhuzamosítás nem triviális (nincs hozzá például olyan egyszerű megoldás, mint C++-hoz az OpenMP), így létrehoztam egy `ParallelFor` nevű osztályt, amivel ez megoldható. Az osztály `ForBody` metódusa felüldefiniálandó, majd meg kell hívni rajta az `execute` metódust. A `ParallelFor` elvégzi a feladat feldarabolását, majd elindít annyi szálat ahány magos a számítógép, végül megvárja, míg mind elkészül. Például a kiértékelés a következőképp néz ki:

```
new ParallelFor<Sample<In, Out>>()
{
    @Override
    protected void ForBody(Sample<In, Out> testSample)
    {
        //... testSample bemenetének osztályozása

        //... kiértékelési mérce számítása az eredmény és
        // testSample kimenetének összevetésével
    }
}.execute(testSet);
```

7.2.5 Osztályozó felépítésének megadása XML nyelven

Sokkal könnyebbé teszi a munkát, ha nem kell programkóddal létrehozni a bonyolultabb felépítésű (sokféle ki-és bemeneti transzformációt, keverő módszereket használó) osztályozókat, hanem egy átlátható XML fájlban is megadható a felépítés. Ezáltal a programkód szerkesztése nélkül, parancssorból is végezhetünk keresztkiértékelést különféle osztályozó-felépítésekkel. Szintén egyszerűen fájlba menthetjük a megtanított osztályozót, amivel azután szervert működtethetünk.

Az XML fájlra példaként egy perceptronokból álló szakértőegyüttes és az IDF súlyozott vektorteres módszer konstans súlyú keverését mutatom be:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE architecture SYSTEM "architecture.dtd">

<architecture>

    <classifier id="idf">
        <type>VectorSpace</type>
        <inputtransform>idf</inputtransform>
    </classifier>

    <classifier id="mlp">
        <type>MLP</type>
        <param name="learnrate">0.1</param>
        <param name="maxepochs">3000</param>
        <param name="splitratio">0.9</param>
        <param name="learningmonitor">
            <learningmonitor>
                <type>DecreaseRatio</type>
                <param name="ratio">0.2</param>
                <param name="windowsize">5</param>
            </learningmonitor>
        </param>
        <param name="errorcalculator">LogLikelihood</param>
    </classifier>

    <classifier id="moe">
        <type>Hierarchic</type>
        <param name="root">
            <classifier ref="mlp" />
        </param>
        <param name="child">
            <classifier ref="mlp">
                <inputtransform>compress</inputtransform>
                <outputtransform>counter</outputtransform>
            </classifier>
        </param>
        <param name="childselector">
            <converter>
                <type>Interval</type>
                <param name="file">code_chapters.txt</param>
            </converter>
        </param>
    </classifier>

    <classifier id="team">
        <type>ConstantWeightTeam</type>
        <param name="c1">
```

```
            <classifier ref="moe" />
        </param>
        <param name="c2">
            <classifier ref="idf" />
        </param>
        <param name="ratio">0.4</param>
    </classifier>

    <classifier id="main" ref="team">
        <inputtransform>bagofwords</inputtransform>
    </classifier>
</architecture>
```

Megadható az egyes részosztályozók típusa és paraméterei, valamint be- és kimeneti transzformációk adhatók hozzájuk (akár több is). Maga a „fő” osztályozó az `id` attribútumban „main”-ként elnevezett lesz. Korábban már definiált osztályozókra a `ref` attribútum segítségével lehet hivatkozni.

7.2.6 Az MLP implementációjáról

A legtöbb osztályozó implementációjának bemutatása érdemében nem tenne hozzá az algoritmusokat bemutató fejezetben leírtakhoz. Az MLP osztályozó azonban bonyolultabb módon, több osztályból áll össze. A fejlesztés során moduláris megoldás elkészítése volt a célom, melyben a különböző feladatok elkülönülnek és külön-külön cserélhetők.

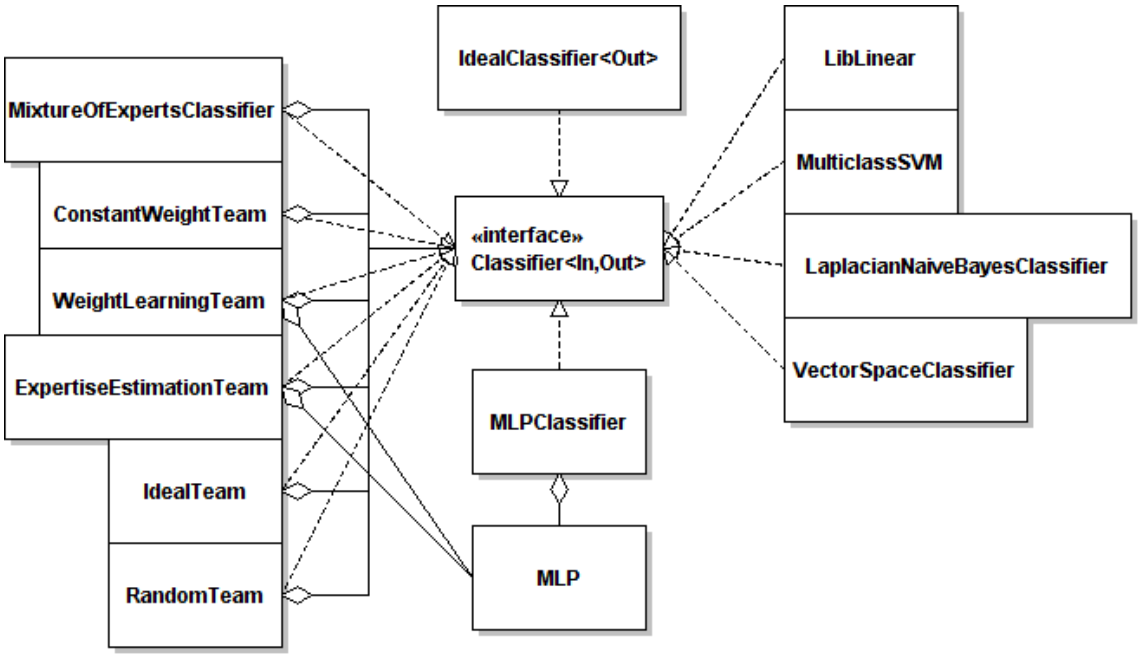
- `MLP`: egy vektorból vektort készítő, `SampleSet<Vector,Vector>` mintakészlettel tanítható rendszer (kívülről nézve). Így nem csak osztályozásra, hanem regresszióra is használható.
- `Layer`: egy réteg súlyait tárolja, lekérdezhető tőle az aktuális kimenete, amit az előző rétegtől kapott értékek és a súlyvektor alapján számít ki.
- `ActivationFunction`: aktivációs függvényt és annak deriváltját számolni képes osztályok interfésze. Ilyen például a `LogSig` és a `TanhSig` aktivációs függvény.
- `LayerTrainer`: egy réteg tanulásáért felel, az előrébb lévő rétegből (vagy a hibakritérium alapján) visszaterjesztett hiba alapján korrigálja a súlyokat, illetve a hátrébb lévő rétegnek továbbterjeszti a hibát.
- `MLPTraining`: a teljes `MLP` tanulásának lefolytatásáért felel. Tanítási ciklusokat végez, méri a validációs halmazon keletkező hibát.
- `LearningMonitor`: interfész, amelynek megadható az aktuális validációs hibaérték. Ezek sorozatát figyeli, és lekérdezhető tőle, hogy kell-e tovább folytatni a tanulást. Ilyen például a `DecreaseRatioLearningMonitor`, ami az utolsó

megadott számú hibaértéket megjegyzi és ellenőrzi, hogy elég sokszor sikerült-e csökkenteni a hibát. Ha túl sokszor nőtt, leállítást javasol.

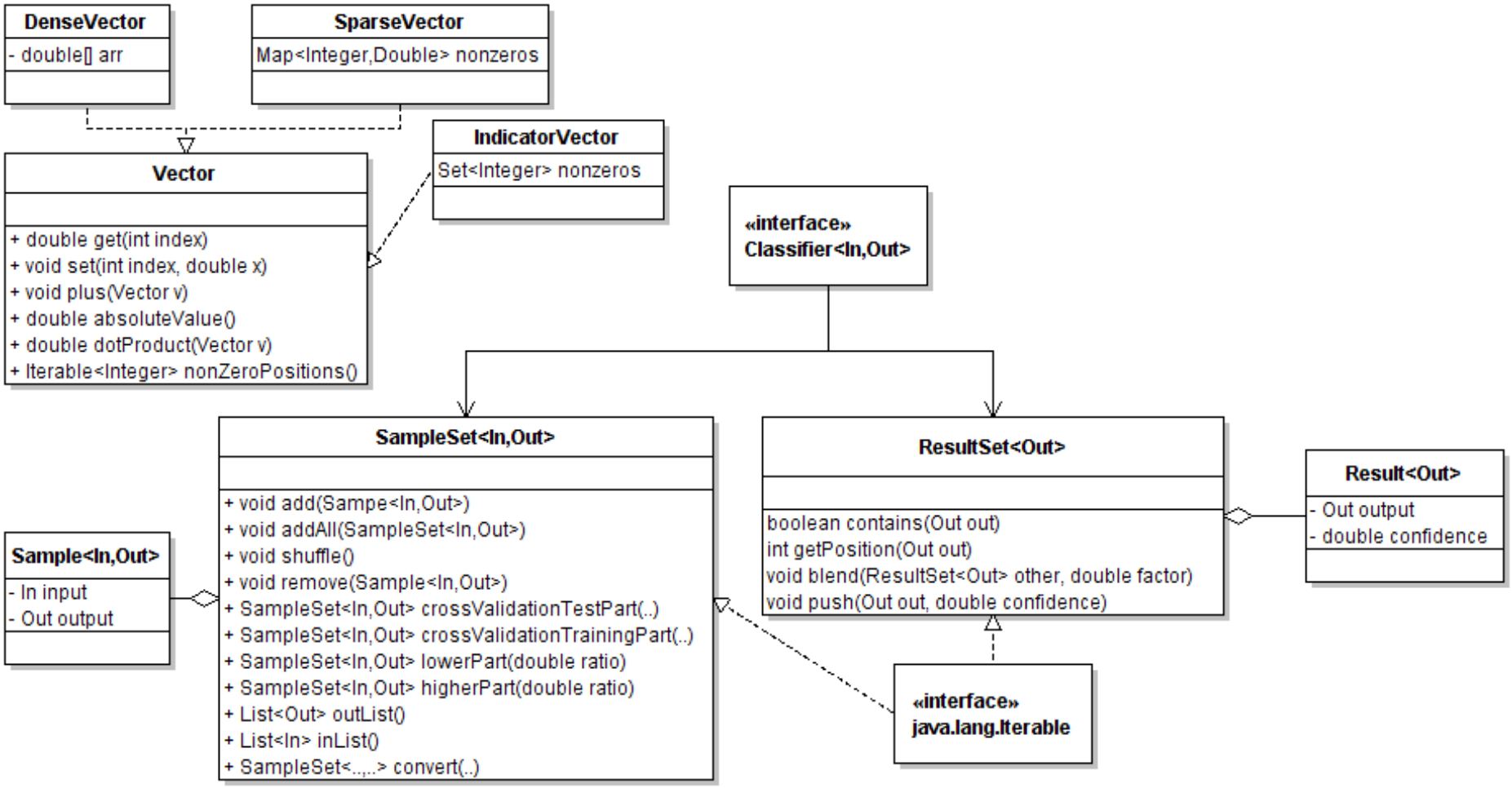
- `ErrorCalculator`: interfész, mely az `MLP` kimeneti rétegén elvárt vektort és a ténylegesen kapott vektort hasonlítja össze és megmondja a hiba mértékét. Szintén képes megmondani a hiba deriváltját a kimenethez képest, amire a hibavisszaterjesztés kezdetekor van szükség. Implementálja a `SquareErrorCalculator` és a `LogLikelihoodErrorCalculator`.
- `MLPClassifier`: belül egy `MLP` objektumot tartalmaz. Tanításkor az osztályozási feladat elvárt kimeneti osztálycímkéje és az `MLP` elvárt kimeneti vektora közti konverziót végzi el. Osztályozáskor az `MLP` által szolgáltatott kimeneti vektorból épít `ResultSet`-et.

7.2.7 UML-diagramok

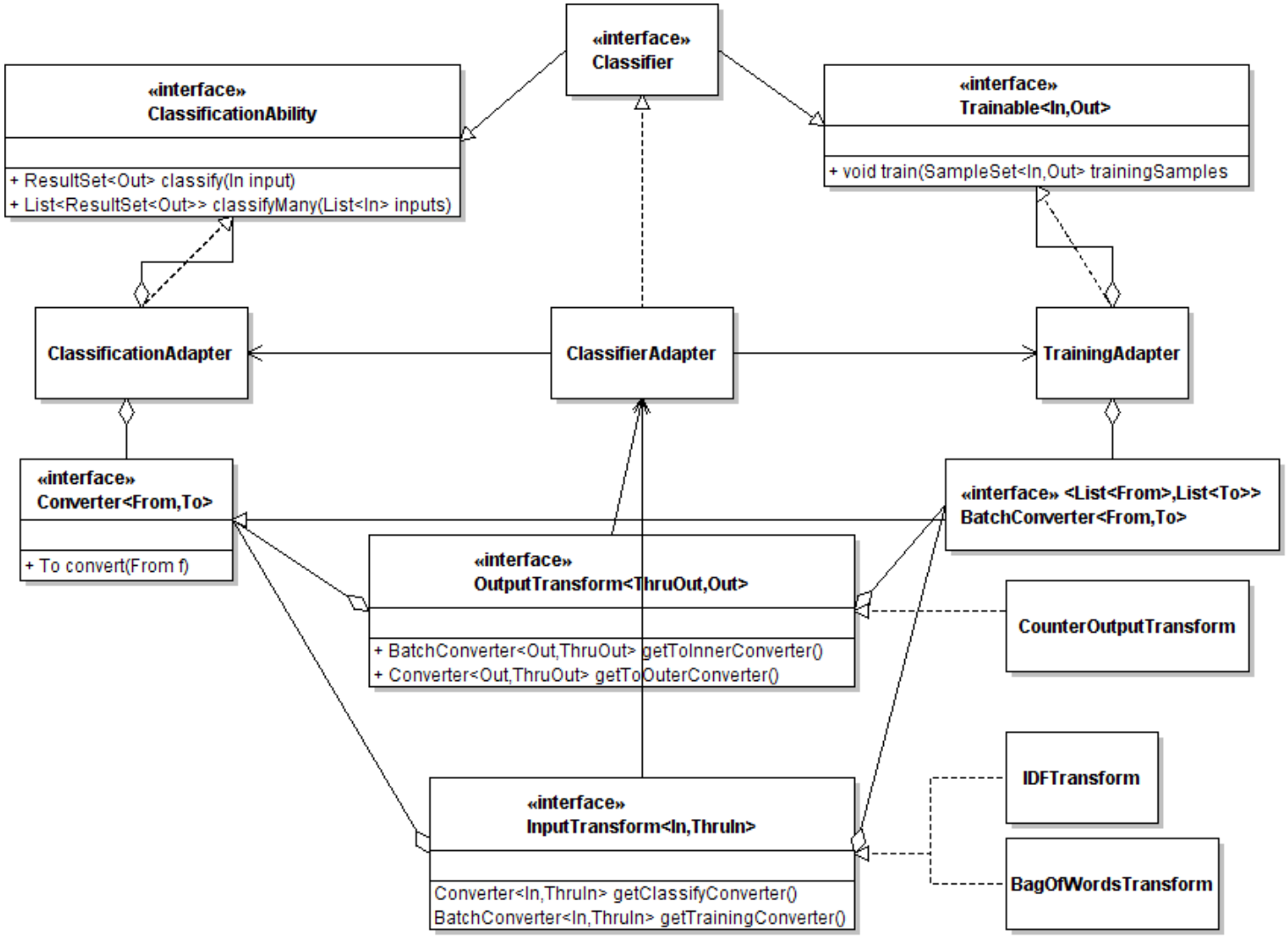
A következő néhány oldalon UML osztálydiagramokon mutatom be az fent leírt osztálystruktúrát.



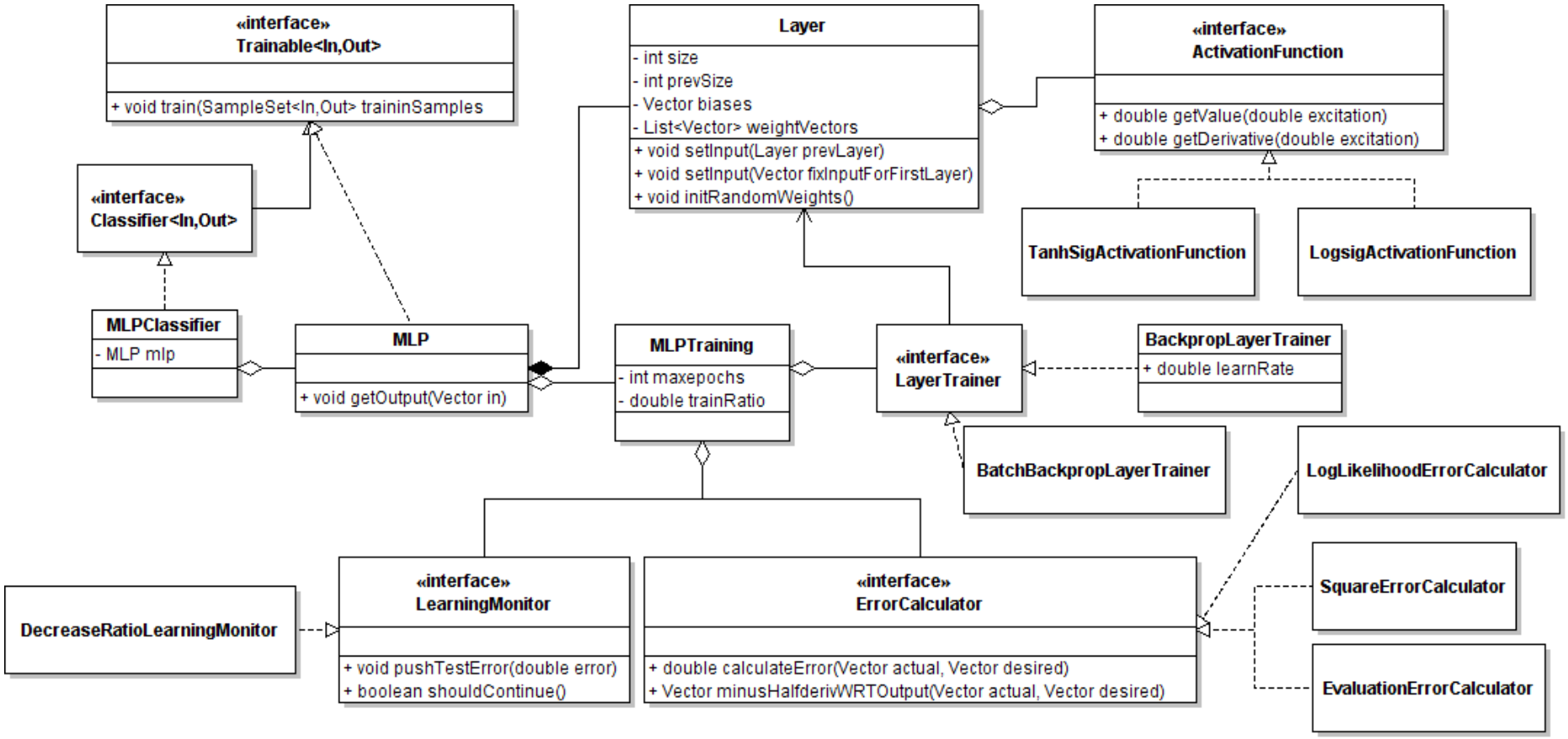
7.2.1. ábra: Konkrét osztályozók a keretrendszerben



7.2.2. ábra: Alapvető osztályok a keretrendszerben (részlet)



7.2.3. ábra: Transzformációk felépítése a keretrendszerben (részlet)



7.2.4. ábra: Az MLP-vel kapcsolatos osztályok struktúrája (részlet)

7.3 Kliens-szerver kommunikáció

A kliens (például a webes felület, vagy egy feltételezett kórházi információs rendszer modulja) és az osztályozószerver TCP socketen küldött XML üzenetekkel kommunikál egymással. Készítettem néhány függvényt, amelyek a programkódban könnyen kezelhető Document Object Model (DOM) formátumban ábrázolt dokumentumokat (org.w3c.dom Java-csomag) TCP socketen keresztül küldeni és fogadni képesek. A teljes folyamat:

1. A felhasználó begépeli a kliensprogramba a kódolandó diagnózist.
2. A kliensprogram elkészíti a kérést XML formátumban, csatlakozik az osztályozószerverhez és átküldi socketen az XML dokumentumot.
3. Az osztályozószerver elemzi a kapott XML-kérést, az osztályozó objektummal elkészítteti az eredménylistát, az eredménylistát XML formátumra alakítja és visszaküldi a kliensnek.
4. A kliens a kapott XML eredménylistát formázza és megjeleníti a felhasználó számára.

A kliens által külött XML-re példa:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<request>
  <inputs>
    <input>ruptura corpus lutea haemorrhagia</input>
  </inputs>
  <resultcount>5</resultcount>
</request>
```

Erre az osztályozó szerver a következő XML üzenettel válaszol:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<resultsets>
  <resultset>
    <result>
      <class>N8310</class>
      <confidence>0.38768336305190293</confidence>
    </result>
    <result>
      <class>D6990</class>
      <confidence>0.17026160892877706</confidence>
    </result>
    <result>
      <class>H4310</class>
      <confidence>0.15751956314998392</confidence>
    </result>
    <result>
      <class>R58H0</class>
      <confidence>0.15043689961141468</confidence>
    </result>
```

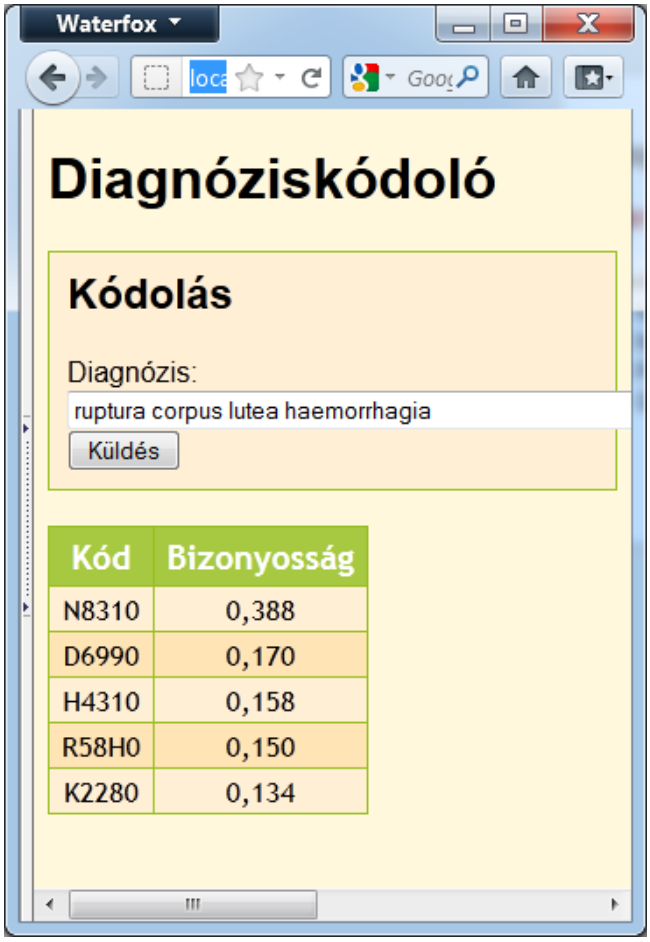
```
    <result>
      <class>K2280</class>
      <confidence>0.13409856525792138</confidence>
    </result>
  </resultset>
</resultsets>
```

Ha a folyamat közben hiba lép fel, például hibás XML-kérést kap az osztályozószerver, akkor egy error típusú XML üzenettel válaszol (feltéve, hogy a hiba nem érinti a socket kapcsolatot). Például:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<error>
  The request XML could not be parsed.
</error>
```

7.3.1 Webes felület

A felvázolt megoldás működését demonstrálandó elkészítettem egy egyszerű JSP (Java Server Pages) alapú weboldalt, amely kliensként képes funkcionálni. A felhasználó beírja a diagnózist egy szövegdobozba, a Küldés gombra kattint, vagy lenyomja az Enter gombot, és megkapja a találati listát.



7.3.1. ábra: Prototípus webes felület működés közben

8 Eredmények, értékelés

A következőkben a különböző algoritmusok különböző mintahalmazokon elért eredményeit ismertetem. Az algoritmusok, amelyeket futtatni fogok:

- Naiv Bayes-háló Laplace-simítással
- IDF-súlyozott vektortér, invertált indexes gyorsítással
- Perceptron: rejtett réteg nélkül, maximum likelihood kritérium szerint tanítva
- LibLinear SVM
- Szakértőegyüttesek (MOE)
    - Perceptronokból építve
    - SVM-ekből építve

A többi keveréses módszert külön alfejezetben tárgyalom.

8.1 Összehasonlítási alap

Az eredményeket önmagukban közölni értelmetlen volna a kontextus elemzése nélkül. Az osztályozás végső százalékos eredménye függ a használt algoritmustól, a feladat jellegétől és az adott mintahalmaztól is.

A BNO-kódolási feladat különösen nehéz, sokszor szakértők számára sem egyértelműen megállapítható a helyes kód. A felhasznált minta tehát gyakran inkonzisztens és ez az eredményeken is meglátszik. A manuális BNO-kódolás átlagos hibájára a szakirodalomban több, jelentősen eltérő értéket is megadnak, ezek nagyságrendileg néhányszor 10% körüli értékek. A kódolás során előforduló hibák elemzése megtalálható [31]-ben. Ezen felül a diagnózisok sokféleképp megfogalmazhatók, amit egy automata rendszernek nehéz felismerni.

A tanítóhalmazból megragadható információ mennyiségét olyan módon fogom jellemezni, hogy definiálok kétféle „ideális” osztályozót, melyek futáskor belenézhetnek a teszthalmazba, azaz csalhatnak. Mindezt annak érdekében, hogy külön elemezhessem az algoritmusokból és a mintahalmaz tulajdonságaiból eredő hibát. Az ideális osztályozó mindig a helyes kódot adja, kivéve:

1. Ha az elvárt kód nem szerepelt a tanítóhalmazban, üres listát ad vissza.
2. Ha egyetlen tanítóhalmazbeli minta sincs, ami legalább egy szót tartalmazna a kódolandó diagnózisból, akkor a tanítóhalmazban szereplő kódokat a gyakoriságuk sorrendjében adja vissza.

Definiálok egy „gyengített ideális osztályozót” is, itt a 2. szabály a következő lesz:

2. Ha egyetlen, *az elvárt kóddal kódolt* tanítóhalmazbeli minta sincs, ami legalább egy szót tartalmazna a kódolandó diagnózisból, akkor a tanítóhalmazban szereplő kódokat a gyakoriságuk sorrendjében adjuk vissza. Tehát ha a diagnózis „rossz közérzet”, a helyes kód pedig *C* lenne, de a tanítóhalmazban nem szerepel olyan *C*-vel kódolt diagnózis, amiben benne lenne akár a „rossz” akár a „közérzet” szó, akkor e szabály szerint csak tippelni tudunk.

Az ideális osztályozó biztosan elméleti felső korlátja minden általam használt algoritmusnak, hiszen új kódok előállítására nem képesek és szavak közötti hasonlóságot sem ismernek. Elméletileg lehetne tervezni a BNO-kódrendszer behatóbb ismeretével olyan rendszert is, ami nem látott kódokat is kitalál, illetve például ontológiák segítségével teljesen különböző szavak között is felderíthető a kapcsolat, így az ideális osztályozó csupán az itt használt algoritmusoknak szab felső határt.

A gyengített ideális osztályozó csak azoknak az algoritmusoknak elméleti felső korlátja, melyek külön-külön tanulják meg az egyes kódok osztályozását, így két kód közti hasonlóság, asszociáció felismerésére képtelenek. Ez igaz az egy aktív rétegű perceptronra, hiszen a kimeneti neuronjai közvetlenül a bemeneti neuronokra vannak rákötve, így a kimenetek között asszociáció nem jöhet létre. A LibLinear SVM csomag bináris SVM-ek segítségével oldja meg a többosztályos problémát, így ez sem képes a kódok közi összefüggések felderítésére. A naiv Bayes-háló is minden osztály esetén csak az adott osztályúnak címkézett tanítóhalmazbeli mintákból indul ki. A vektorteres módszerben a bemeneti vektor ilyenkor minden olyan vektorra merőleges, ami a helyes kóddal van címkézve, így biztosan képtelen lesz helyes választ adni. A szakértőegyüttesekre ez már nem igaz, hiszen ott a megfelelő szakértő nagy súlya esetén kicsit nagyobb eséllyel kerül a lista elejére a helyes kód, mint az apriori valószínűség.

Az ideális osztályozókat is a keretrendszerben implementáltam, így a tesztelés során ugyanolyan módon használhatók, mint bármely más algoritmus.

8.2 Kiértékelési módok

Ahogy a 4.1-es fejezetben megindokoltam, az $R(h)$ teljesség (recall) értékeket fogom kiszámolni különböző h hosszúságú listák esetére. Négyféle eredményt fogok megadni, az 1, 5, 10 és 20 hosszúságú listákon vett találati aránynak megfelelően.

8.3 Magyar mintahalmazok

A konzulensem által rendelkezésemre bocsátott magyar mintahalmazok:

	Minták száma	Szavak száma	Kódok száma	BNO-verzió
Magyar	3081	2936	1373	BNO-10
Magyar, tisztított		1894		

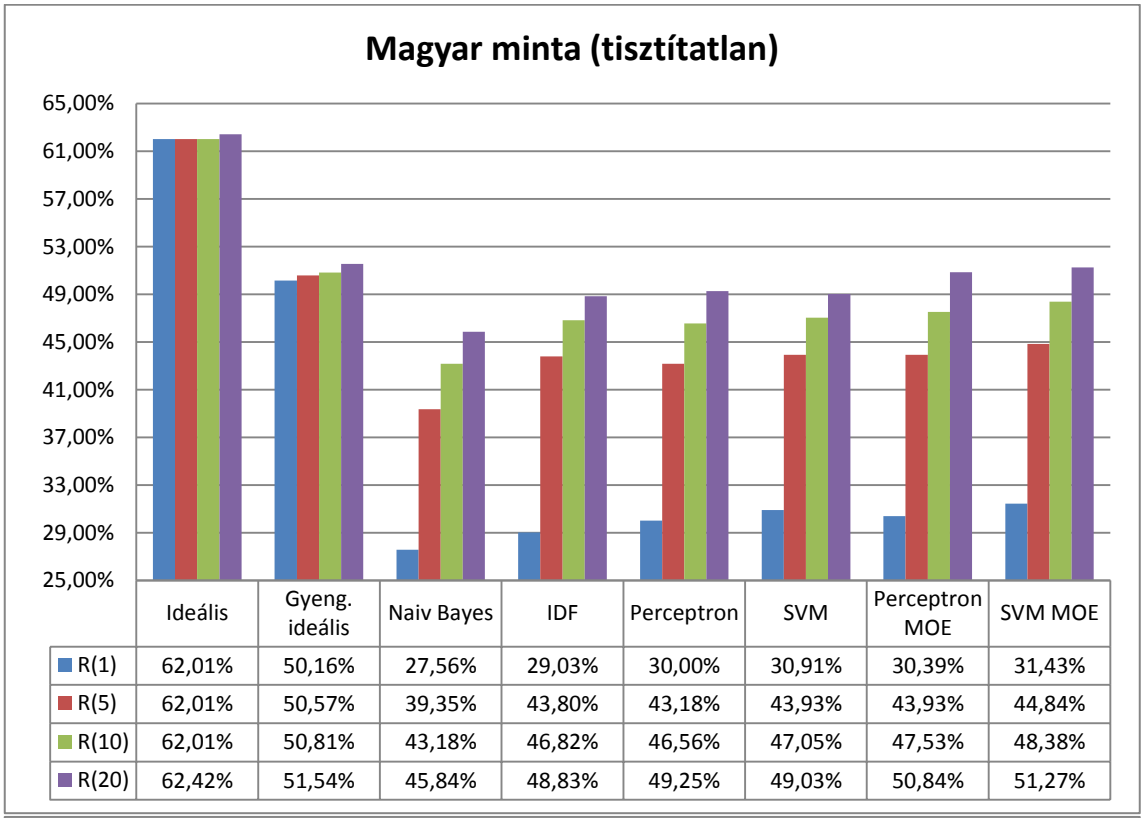
Az első az egykori Haynal Imre Egészségtudományi Egyetemen használt kódolt diagnózislista, mely gyakran előforduló diagnózisokból áll. Ebből kézi kódolás során a kódolók gyorsabban megtalálták a kódot, mint a vaskos BNO-könyvben.

A tiszítatlan mintán elért eredmények (következő oldalon) különböző osztályozó algoritmusok esetén nem mutatnak nagy szórást a százalékos eredmények szempontjából, azonban az egyértelműen látszik, hogy a szakértőegyüttesek jobban teljesítenek. A grafikonokon a százalékskála az érdekes tartományt mutatja. A neurális hálózat 20 hely után vett 49,25%-os eredménye már nagyon jól megközelíti az elméleti 51,54%-os felső korlátot. A szakértőegyüttesek kódasszociációs képessége nem elegendő ahhoz, hogy átlépjék a gyengített ideális osztályozó eredményét, azonban nagyon szorosan megközelítik. Meglepő ugyanakkor az ideális osztályozó gyenge szerepése. Eszerint külön szofisztikált nyelvi előfeldolgozás, esetleg ontológiák bevetése nélkül kb. 62% a maximálisan elérhető eredmény. Ennek oka a kevés mintában (3081) és az ehhez képest sok kódban keresendő (1373).

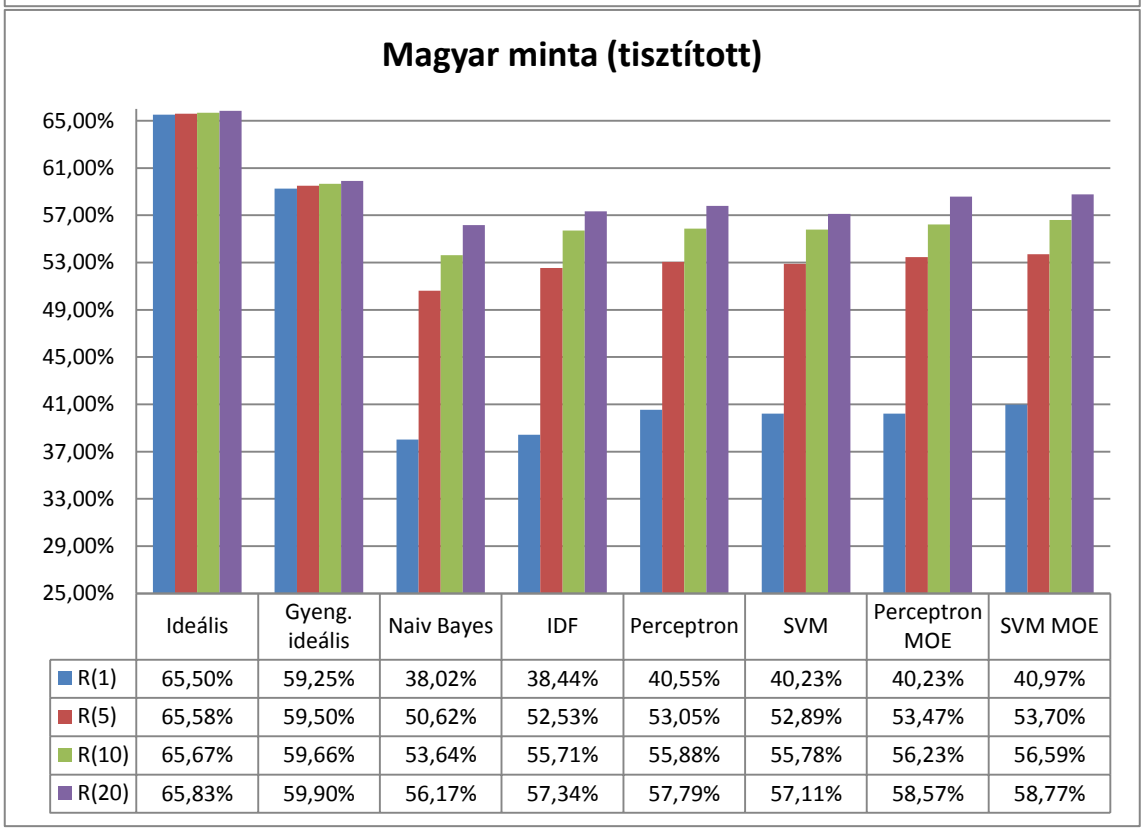
A tisztított mintahalmazt szakértők manuálisan hozták létre a tisztítatlan mintából kiindulva. A kódokat ellenőrizték, a konzisztenciát javították A diagnózisok terminológiáját egységesítették, a rövidítéseket feloldották, így csökkent a szavak száma. Például a *„Tu.hypopharyngis cum met.colli (l.s.)”* diagnózisból a *„neoplasma hypopharynx cum metastasis cervix”* jött létre az átalakítás során.

A tisztított mintán elért eredmények nagyon hasonló képet mutatnak, azonban minden érték 8-10 százalékponttal megnőtt, ami jóval nagyobb, mint bármely két algoritmus közötti különbség. Ez köszönhető a kódok konzisztensebb kódolásának és az

egységes szóhasználatnak. Az ideális osztályozó azonban még mindig 65% körül maradt, ami megerősíti azt a sejtést, hogy sok a kód és kevés a minta.



	Ideális	Gyeng. ideális	Naiv Bayes	IDF	Perceptron	SVM	Perceptron MOE	SVM MOE
R(1)	62,01%	50,16%	27,56%	29,03%	30,00%	30,91%	30,39%	31,43%
R(5)	62,01%	50,57%	39,35%	43,80%	43,18%	43,93%	43,93%	44,84%
R(10)	62,01%	50,81%	43,18%	46,82%	46,56%	47,05%	47,53%	48,38%
R(20)	62,42%	51,54%	45,84%	48,83%	49,25%	49,03%	50,84%	51,27%



	Ideális	Gyeng. ideális	Naiv Bayes	IDF	Perceptron	SVM	Perceptron MOE	SVM MOE
R(1)	65,50%	59,25%	38,02%	38,44%	40,55%	40,23%	40,23%	40,97%
R(5)	65,58%	59,50%	50,62%	52,53%	53,05%	52,89%	53,47%	53,70%
R(10)	65,67%	59,66%	53,64%	55,71%	55,88%	55,78%	56,23%	56,59%
R(20)	65,83%	59,90%	56,17%	57,34%	57,79%	57,11%	58,57%	58,77%

8.3.1. ábra: A tiszított és tisztítatlan magyar mintán elért eredmények

8.4 Német mintahalmazok

Konzulensem egy jóval nagyobb méretű német nyelvű mintahalmazt is rendelkezésemre bocsátott, itt már sokkal jobb a minták száma/kódok száma arány, így jobb eredmények várhatók.

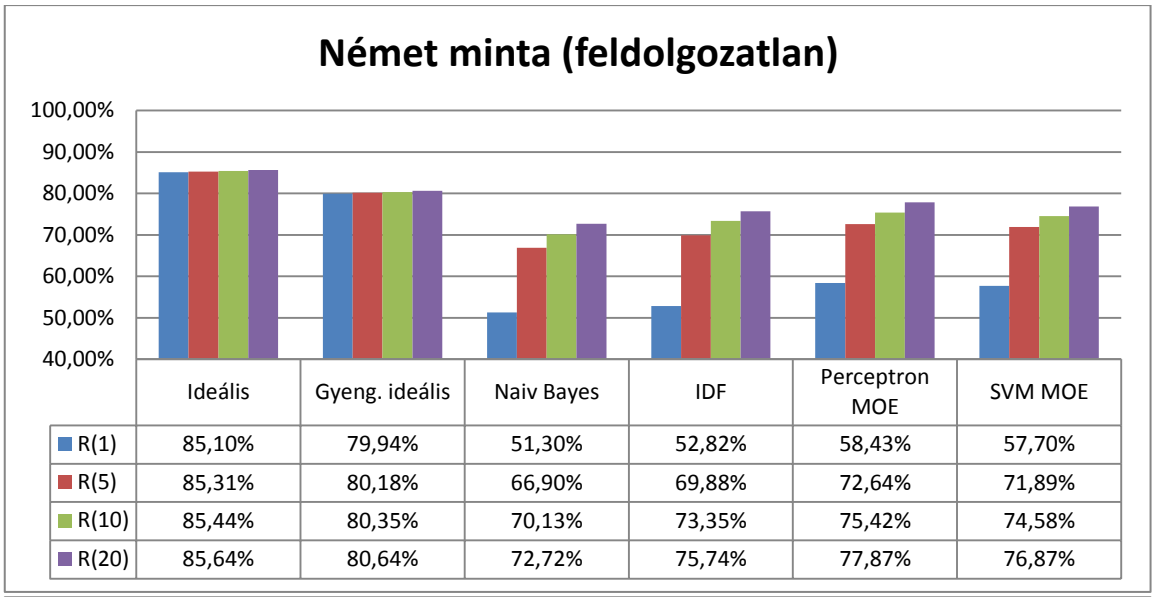
	Minták száma	Szavak száma	Kódok száma	BNO-verzió
Német	93863	40564	3593	BNO-9
Német szótövezett		35134		
Német morfológiailag elemzett		3641		

Ilyen nagy méretű halmazon az egyszerű neurális hálózatot és az SVM-et nem célszerű közvetlenül futtatni (később az időeredményeknél látható a futásidő). A megfelelő szakértőegyüttesek töredék idő alatt megtaníthatók, és amint a magyar mintánál is láttuk, az eredményük jobb, mint az egyszintű megfelelőjüknek.

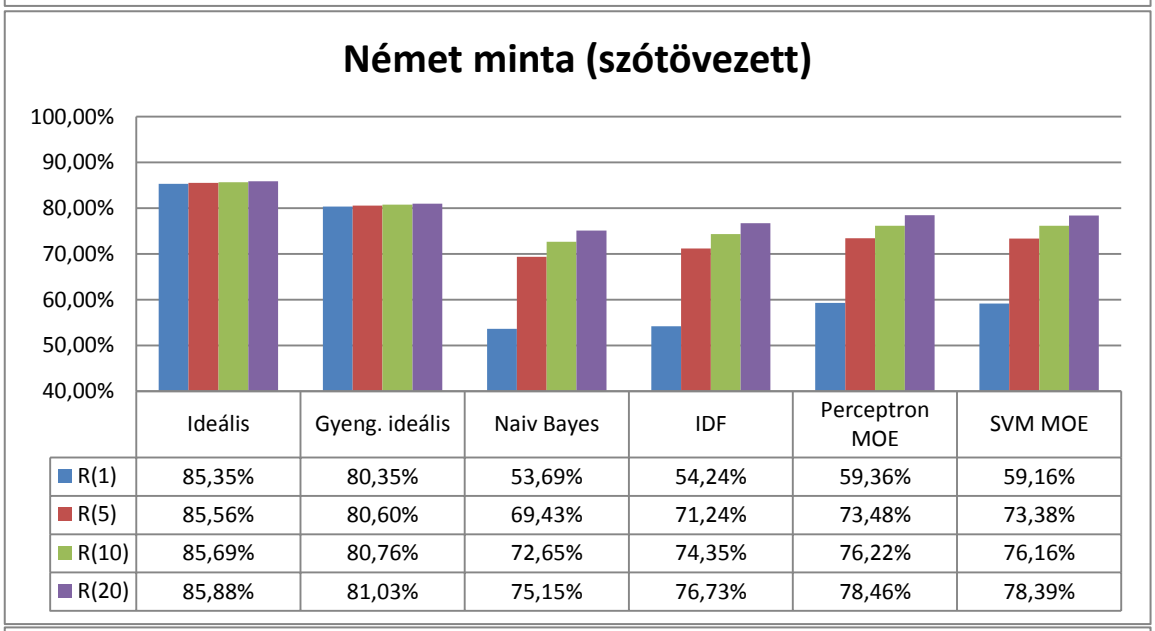
A feldolgozatlan német mintán elért eredmények hasonlóak a magyarhoz, azzal a különbséggel, hogy az eredmények kb. 25 százalékponttal jobbak, körülbelül ugyanannyival, mint amennyivel az ideális osztályozó által kikötött felső korlát feljebb van.

A szótövezett mintahalmaz egy egyszerű szótövező algoritmussal gépi úton készült. Egy átalakítási példa: *„Ergüsse, subdurale bds.”* → *„erguss subdural bds.”*. Ez a kódok hozzárendelését nem érinti, így az ideális osztályozók eredménye alig változik a szótövezés hatására (a kevesebb szó miatt a 2. szabály valamivel ritkábban kerül alkalmazásra, így az eredmények kicsivel jobbak). Az algoritmusok eredménye kb. 2 százalékpontot javult. Ez még összemérhető nagyságrendű az egyes algoritmusok közti különbségekkel.

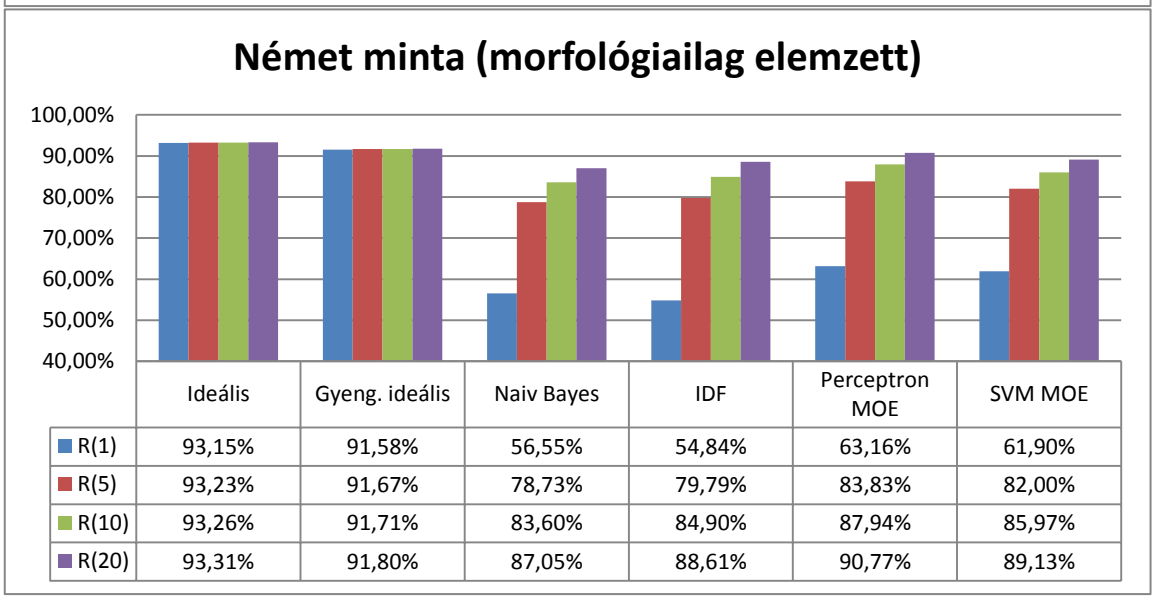
A harmadik, morfológiailag elemzett minta pontos készítési algoritmusát nem ismerem, azonban az előforduló szavak számát drasztikusan csökkentő módszerről van szó, a morfológiai elemzésen túl szinonima-összevonás is történt. Az előbb említett diagnózis itt *„guessijwkrj hypoiiiqqia duraiiiwwpa”*-ként jelenik meg (felismerhető, hogy például a *sub*- előtagból egy *hypo*- kezdetű szót generált a rendszer). Ez az előfeldolgozás további kb. 10 százalékpontnyi javulást okoz (kivéve az első helyen vett találatot, ami a valós felhasználásnál nem annyira lényeges). Ennek alapján érdemes lehet kifejleszteni egy hasonló morfológiai elemző rendszert kevert magyar/latin nyelvű diagnózisokhoz is.



	Ideális	Gyeng. ideális	Naiv Bayes	IDF	Perceptron MOE	SVM MOE
R(1)	85,10%	79,94%	51,30%	52,82%	58,43%	57,70%
R(5)	85,31%	80,18%	66,90%	69,88%	72,64%	71,89%
R(10)	85,44%	80,35%	70,13%	73,35%	75,42%	74,58%
R(20)	85,64%	80,64%	72,72%	75,74%	77,87%	76,87%



	Ideális	Gyeng. ideális	Naiv Bayes	IDF	Perceptron MOE	SVM MOE
R(1)	85,35%	80,35%	53,69%	54,24%	59,36%	59,16%
R(5)	85,56%	80,60%	69,43%	71,24%	73,48%	73,38%
R(10)	85,69%	80,76%	72,65%	74,35%	76,22%	76,16%
R(20)	85,88%	81,03%	75,15%	76,73%	78,46%	78,39%



	Ideális	Gyeng. ideális	Naiv Bayes	IDF	Perceptron MOE	SVM MOE
R(1)	93,15%	91,58%	56,55%	54,84%	63,16%	61,90%
R(5)	93,23%	91,67%	78,73%	79,79%	83,83%	82,00%
R(10)	93,26%	91,71%	83,60%	84,90%	87,94%	85,97%
R(20)	93,31%	91,80%	87,05%	88,61%	90,77%	89,13%

8.4.1. ábra: A három német mintán elért eredmények

8.5 Néhány keveréses eredmény

A sokféle keverési módszer (ideális, konstans súlyú, súlybecslő és jóságbecslő), a mintahalmazok nagy száma, valamint az algoritmusok rengeteg kombinációs lehetősége miatt csak néhány példán mutatom be a keveréses eredményeket.

A keverésekhez az altanítómintára és altesztmintára vágást 40%-60% arányban végeztem. A jóságbecslő módszernél a jóság mértékének a helyes kód listán való pozíciójának reciprokát választottam.

A regressziót végző MLP-k beállításai:

- bátorsági faktor: 0,004
- kimeneti réteg: logisztikus aktivációs függvény
- rejtett réteg: 150 neuronból, tangens hiperbolikusz aktivációs függvénnyel
- tanulási kritérium: maximum likelihood
- altanítóminta-altesztminta arány: 90%-10%

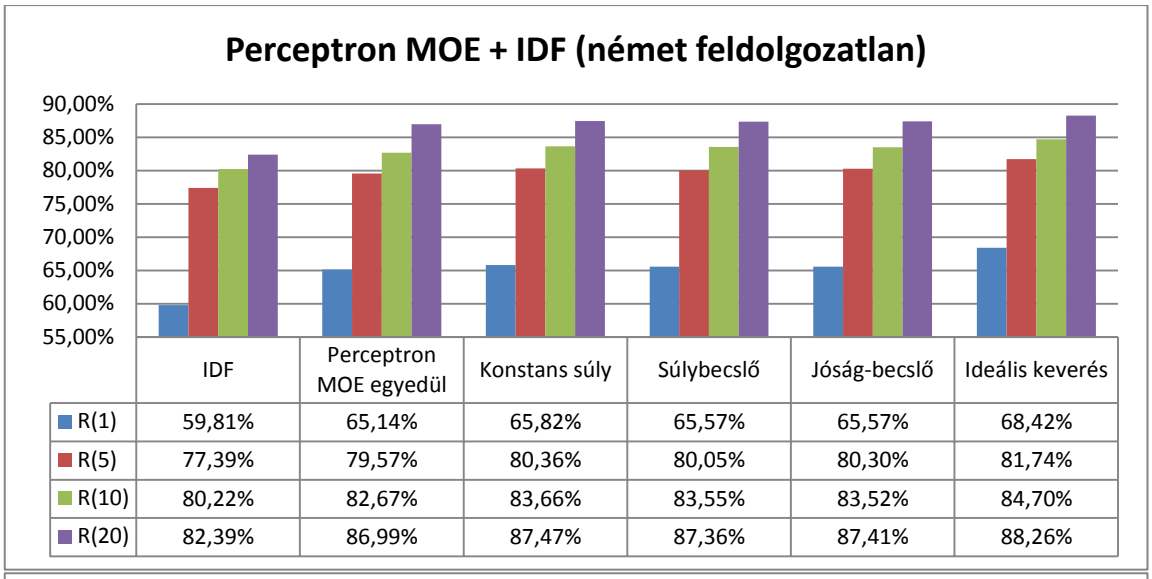
Itt is számításba veszem a lehető legjobb, „ideális keverés” eredményét. Ez a módszer minden egyes tesztmintára a lehető legjobb súlyozást választja ki (csak két osztályozó esetére implementáltam, kimerítő kereséssel). Ez szintén a tesztmintába történő betekintéssel valósul meg, csakúgy, mint a korábbi ideális osztályozók esetén.

Az eredményekből kitűnik, hogy *még ideális esetben is* csak 1-2% javulás lenne elérhető a legtöbb esetben, a ténylegesen elért javulás pedig – bár mérhető – szabad szemmel alig látható. Az azonban beigazolódott, hogy egy gyengébb modell (pl. IDF vektortér) összekeverése egy jobb modellel (pl. perceptron MOE) valóban képes egy mindkettőjüknél jobb modellt létrehozni. A gyenge eredmények lehetséges magyarázata:

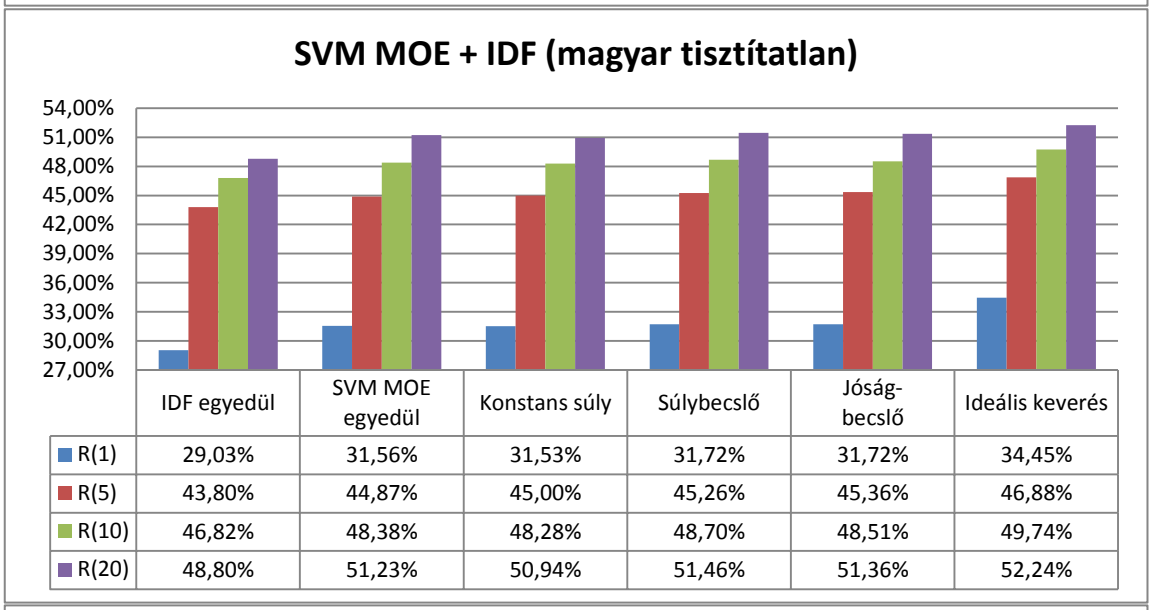
1. Az ideális esetben is gyenge eredményekből következik, hogy a 6. fejezet elején említett első potenciális javulási ok csekély mértékben teljesül. Ott azt feltételeztem, hogy a hibás kódok véletlenszerűek, a helyes felé viszont közös „tendencia” áll fenn. Ez megbukhat azon, hogy
   a. a hibás kódok nem véletlenszerűek, hanem abban is közös tendencia mutatkozik a különböző osztályozók között (pl. mert inkonzisztens a tanító- és a teszthalmaz),

b. illetve azon is, hogy hibázás esetén minden eredménylista véletlenszerű, helyes kód felé mutató közös tendencia nem jelentkezik.

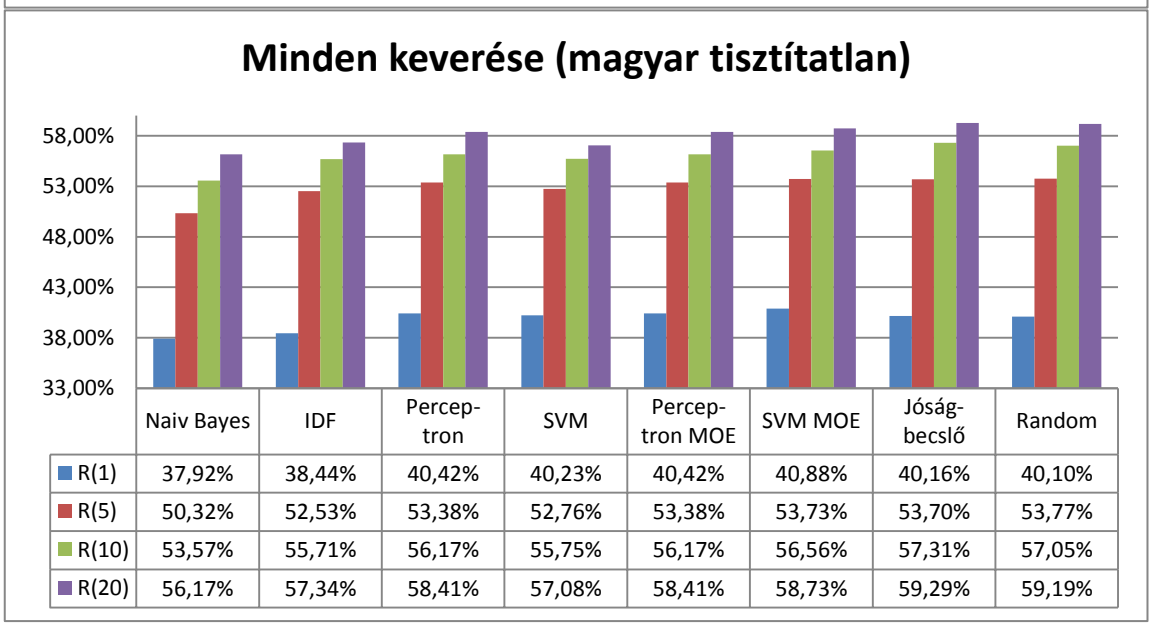
2. A bemenetfüggő MLP alapú regresszióval súlyokat becslő rendszerek nem teljesítenek határozottan jobban, mint a konstans, vagy akár a véletlenszerű keverés. Ez azt jelenti, hogy egy adott bemenet és az osztályozók rajta elért jóságmértéke közti összefüggés az MLP-vel nem tanulható meg (több, nagyobb rejtett réteggel sem sikerült). Ez valószínűleg abból ered, hogy ilyen összefüggés nincs is, mert az osztályozó típusa nem korlátozza a jóságot a bemeneti tér egy adott szegmensére.



	IDF	Perceptron MOE egyedül	Konstans súly	Súlybecslő	Jóság-becslő	Ideális keverés
R(1)	59,81%	65,14%	65,82%	65,57%	65,57%	68,42%
R(5)	77,39%	79,57%	80,36%	80,05%	80,30%	81,74%
R(10)	80,22%	82,67%	83,66%	83,55%	83,52%	84,70%
R(20)	82,39%	86,99%	87,47%	87,36%	87,41%	88,26%



	IDF egyedül	SVM MOE egyedül	Konstans súly	Súlybecslő	Jóság-becslő	Ideális keverés
R(1)	29,03%	31,56%	31,53%	31,72%	31,72%	34,45%
R(5)	43,80%	44,87%	45,00%	45,26%	45,36%	46,88%
R(10)	46,82%	48,38%	48,28%	48,70%	48,51%	49,74%
R(20)	48,80%	51,23%	50,94%	51,46%	51,36%	52,24%



	Naiv Bayes	IDF	Percep-tron	SVM	Percep-tron MOE	SVM MOE	Jóság-becslő	Random
R(1)	37,92%	38,44%	40,42%	40,23%	40,42%	40,88%	40,16%	40,10%
R(5)	50,32%	52,53%	53,38%	52,76%	53,38%	53,73%	53,70%	53,77%
R(10)	53,57%	55,71%	56,17%	55,75%	56,17%	56,56%	57,31%	57,05%
R(20)	56,17%	57,34%	58,41%	57,08%	58,41%	58,73%	59,29%	59,19%

8.5.1. ábra: Keveréses módszerekkel elért eredmények

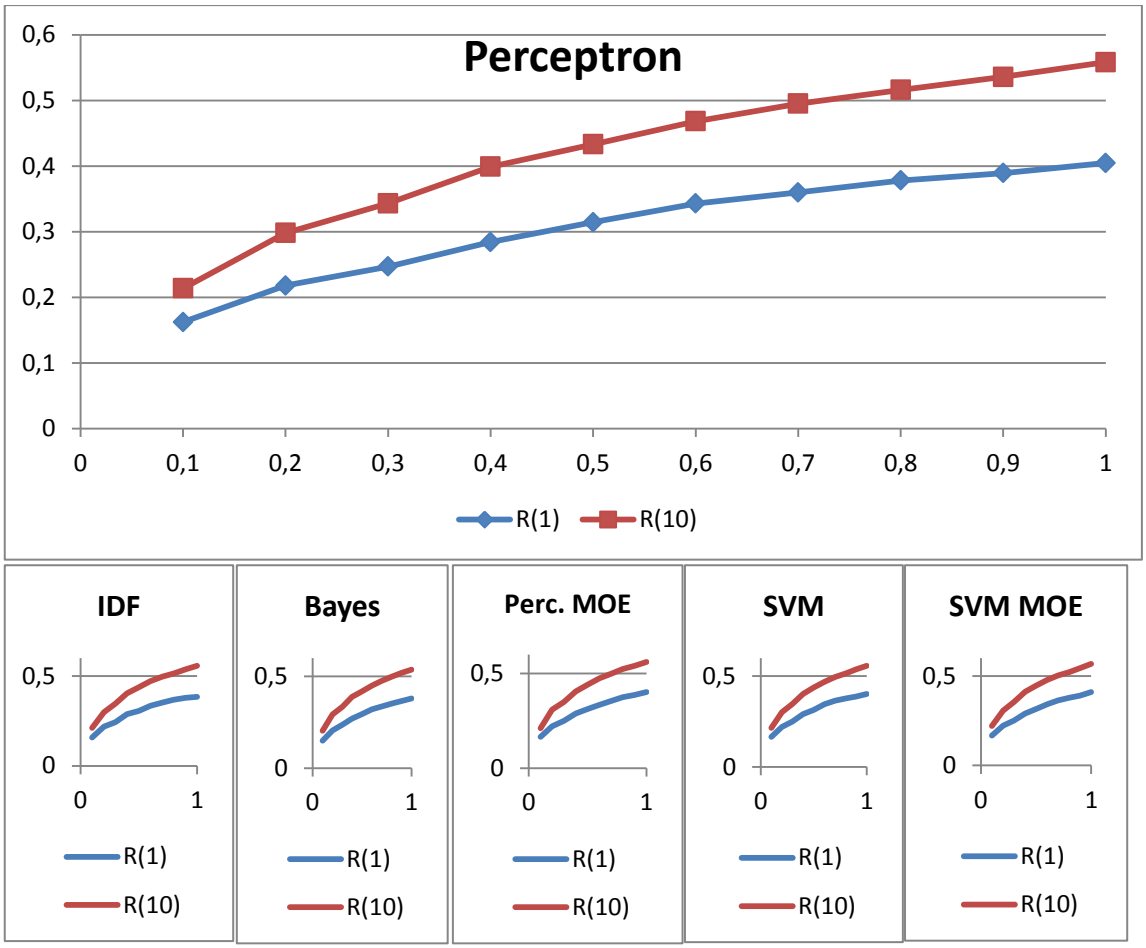
8.6 Tanulási görbék

Tanulási görbének azt a diagramot nevezik, aminek vízszintes tengelyén a tanítóhalmaz mérete, függőleges tengelyén az osztályozó jóságmértéke van ábrázolva. Egy ilyen diagramra ránézve megállapítható, hogy mennyire tudna további mintákból profitálni a rendszer (lásd pl. ezt a publikációt [32]). Ha a rendszer hibája a torzításból ered (lásd torzítás-variancia dilemma), akkor a görbe ellaposodik. Ha a varianciából, akkor még emelkedés látható a görbe végén.

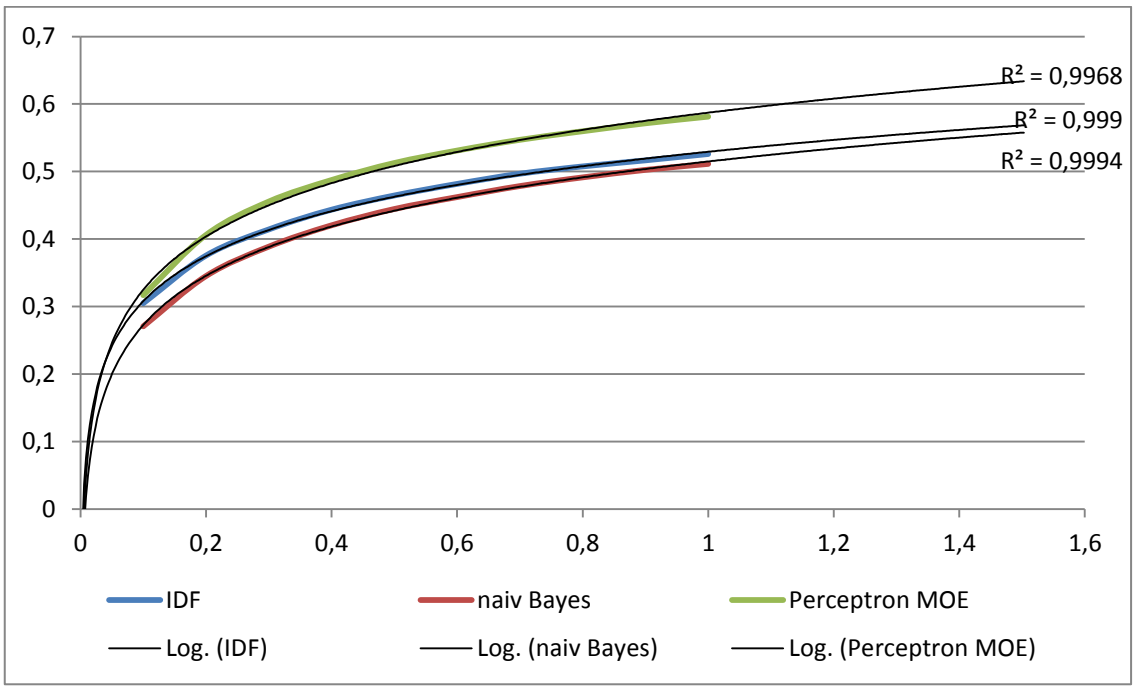
A tanulási görbéket 10%-os független teszthalmazt használva készítettem el. A vízszintes tengelyen a tanításkor felhasznált tanítóminták arányát (a 90%-on belül), a függőleges tengelyen pedig az elkülönített teszthalmazon mutatott eredményt ábrázolom.

Mindkét mintahalmaz esetén emelkedő végű görbét látunk, eszerint a rendszer tovább tudna javulni, azaz a hiba inkább a varianciából ered, mint a torzításból. Sok gépi tanulási feladatban ez nem igaz, elképzelhető például, hogy egy tanítóhalmaz minden lényeges információt tartalmaz az adott tárgyterületről, és további minták beszerzése csak a redundanciát növelné.

A német görbékhez nagyon jól illeszthető logaritmikus trendvonal. Ennek segítségével óvatosan akár számszerűleg is megbecsülhetjük további minták beszerzésének hatását.



8.6.1. ábra: A tisztítatlan magyar mintán készített tanulási görbék



8.6.2. ábra: A feldolgozatlan német mintán készített tanulási görbék R(1), logaritmikus trendvonal-illesztéssel

8.7 Futási idők

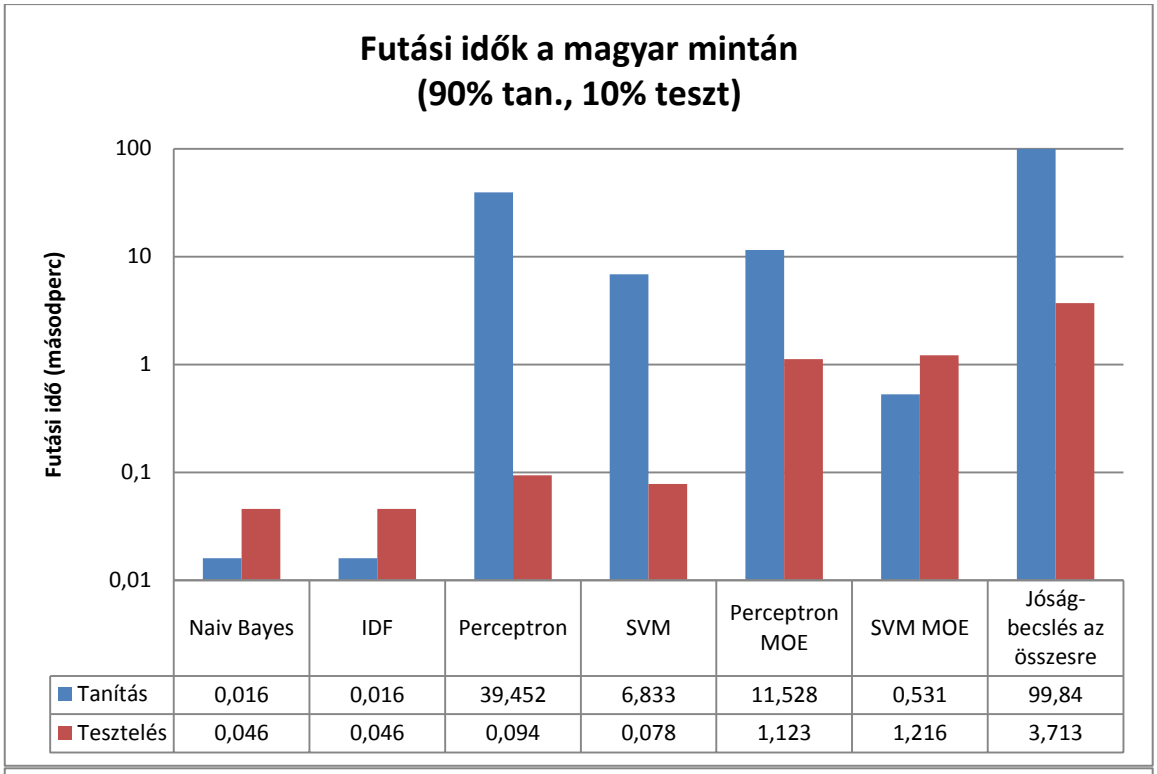
Az algoritmusok futtatását a következő számítógépen végeztem:

- CPU: Intel Core i7-920 @ 2.67 GHz
- RAM: 6 GB
- Operációs rendszer: Windows 7

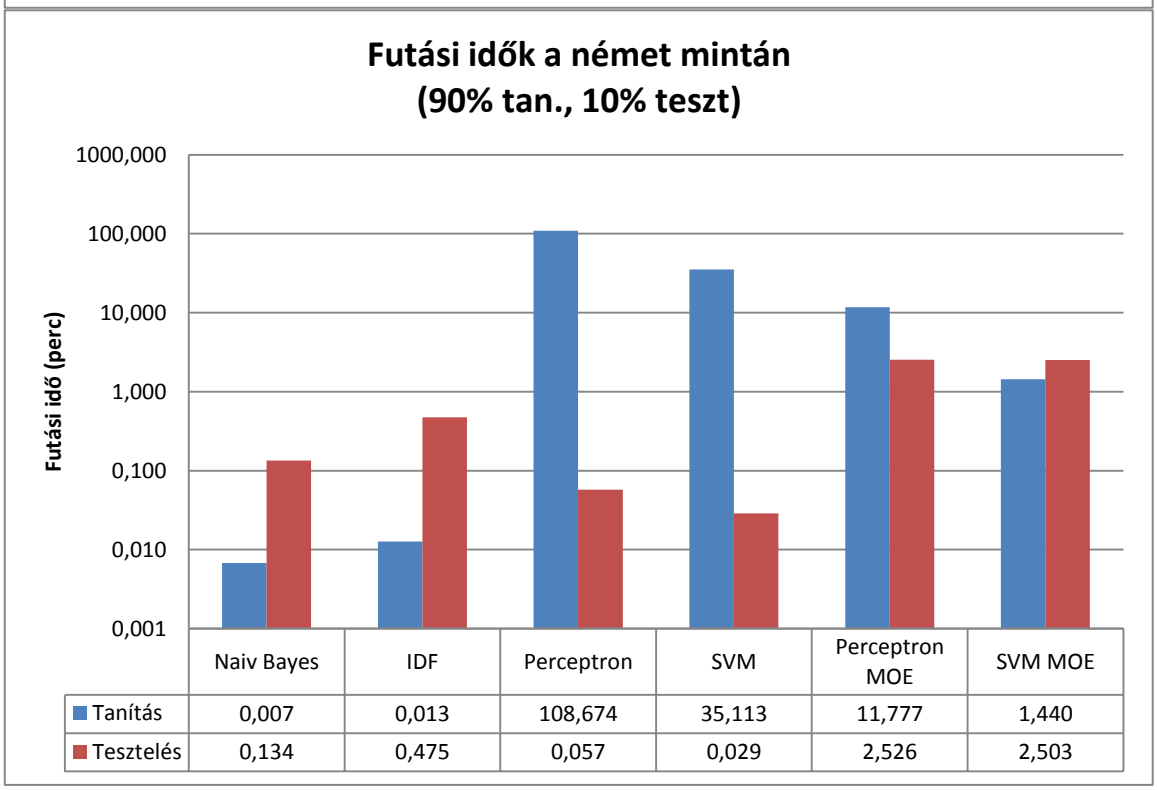
Megfigyelhető, hogy a szakértőegyüttesek sokszoros gyorsítást hoznak tanítási időben, emellett láttuk, hogy az eredményük is jobb. Kiértékeléskor a „lapos" megfelelőik gyorsabbak, ekkor ugyanis a szakértőegyüttesben minden szakértő elvégzi az osztályozást valamint a kapuzóhálózat is osztályoz, míg a lapos változatban egyetlen osztályozás történik.

Az is látható, hogy a LibLinear SVM valóban nagyon gyorsan tanítható (kifejezetten dokumentumosztályozásra optimalizálták), a perceptronhoz képest többször gyorsabb, és mégis hasonló eredményt képes elérni.

A kiértékelési idő a legrosszabb esetben is 12 ms/minta. Ha a párhuzamosítás gyorsításától megtisztítjuk az adatot, akkor is kb. 50 ms alatt osztályozható a diagnózis. Ez a hálózati kommunikáció időigényével azonos nagyságrend, így tökéletesen megfelelő a webes szolgáltatásnál.



	Naiv Bayes	IDF	Perceptron	SVM	Perceptron MOE	SVM MOE	Jóság-becslés az összesre
Tanítás	0,016	0,016	39,452	6,833	11,528	0,531	99,84
Tesztelés	0,046	0,046	0,094	0,078	1,123	1,216	3,713



	Naiv Bayes	IDF	Perceptron	SVM	Perceptron MOE	SVM MOE
Tanítás	0,007	0,013	108,674	35,113	11,777	1,440
Tesztelés	0,134	0,475	0,057	0,029	2,526	2,503

8.7.1. ábra: Futási idők a magyar (tisztított) és a német (feldolgozatlan) mintán. Figyelem! Eltérő idő-mértékegységek és logaritmikus skála!

9 Összegzés, továbbfejlesztési lehetőségek

Szakdolgozatomban megvizsgáltam orvosi diagnózisok BNO-kódolásának gépi tanulási és statisztikai alapú automatizálási lehetőségeit. Kifejlesztettem egy keretrendszert, amely osztályozási problémák megoldását támogatja. A szakirodalom áttekintése után három algoritmust saját kezűleg implementáltam a keretrendszerben, egy létező implementációt pedig a rendszerben használható formára hoztam. Megvizsgáltam a módszerek keverésében, hibrid modellek előállításában rejlő lehetőségeket. Az elkészült osztályozókat különféle mintákon teszteltem, és megvizsgáltam az előfeldolgozás hatását az eredményre. Kialakítottam egy kliens-szerver kommunikációs módot, melynek alapján diagnóziskódolási szolgáltatás hozható létre. Elkészítettem egy weboldal klienst, amellyel a működést demonstráltam.

Az eredmények alapján maguk az implementált algoritmusok jól teljesítenek, 10 hosszúságú eredménylistát engedélyezve már jól megközelítik az elméleti maximumot. Így az elméleti maximum növelése lehet új cél. Ez megoldható további minták beszerzésével (a tanulási görbék meredeksége azt mutatta, hogy a rendszer további mintákból még sokat tudna profitálni), illetve előfeldolgozó rendszer kifejlesztésével. A német minták eredményeit összevetve inkább egy morfológiai elemző segítene, mint az egyszerű szótövezés. Ontológiák és szinonimakezelés hatása is megvizsgálandó.

Valós felhasználás során érdemes lenne felhasználói visszajelzéseket is gyűjteni a rendszer teljesítéséről. Például a felületen megadhatná a felhasználó, hogy végül mely kódot fogadta el, amit elkülönítve lehetne gyűjteni. Mivel a használt algoritmusok elvi illetve gyakorlati okokból inkrementálisan nem taníthatók, meg kellene várni, míg összegyűlik kellő mennyiségű felhasználói adat, majd ezeket hozzávéve újratanítani az osztályozót. További gondot jelent, hogy az így létrejövő halmaz nem biztos, hogy konzisztens (a felhasználó téved, vagy az eredeti tanítóhalmaz volt hibás), ezért az újratanítás előtt konzisztenciavizsgálatot érdemes végezni.

Ezenkívül célszerű volna a felhasználói felület bővítése, összekötése BNO-adatbázissal, így az egyes kódok rövid összefoglalóját is meg lehetne jeleníteni a találati listán.

Irodalomjegyzék

1. **Surján, György.** *Barriers and challenges of using medical coding systems.* Veenendaal : Universal Press, 2010.

2. *Large Scale Diagnostic Code Classification for Medical Patient Records.* **Lita, Lucian Vlad, és mtsai.** 2008. Proceedings of the Third International Joint Conference on Natural Language Processing.

3. *MIDAS: An Information-Extraction Approach to Medical Text Classification.* **Sotelsek-Margalef, Anastasia and Villena-Román, Julio.** 2008, Procesamiento del lenguaje Natural No 41, pp. 97-104.

4. Disease Ontology. [Online] [Hivatkozva: 2011. 12 9.] http://do-wiki.nubic.northwestern.edu/index.php/Main_Page.

5. *Ontology-supported processing of clinical text using medical knowledge integration for multi-label lassification of diagnosis coding.* **Waraporn, Phanu, Meesad, Phayung és Clayton, Gareth.** 3, 2003., International Journal of Computer Science and Information Security, 7. kötet. ISSN 1947-5500.

6. *Automatic Matching of ICD-10 codes to Diagnoses in Discharge Letters.* **Boytcheva, Svetla.** Hissar, Bulgaria : ismeretlen szerző, 2011. Proceedings of the Workshop on Biomedical Natural Language Processing. old.: 11-18. ISBN 978-954-452-020-5.

7. *From Indexing the Biomedical Literature to Coding Clinical Text: Experience with MTI and Machine Learning Approaches.* **Aronson, Alan R., et al.** Prága : Association for Computational Linguistics. BioNLP 2007: Biological, translational, and clinical language processing. pp. 105-112.

8. *Automatic construction of rule-based ICD-9-CM coding systems.* **Farkas, Richárd and Szarvas, György.** Singapore : s.n., 2008. The Second International Symposium on Languages in Biology and Medicine (LBM) .

9. *Automating the Assignment of Diagnosis Codes to Patient Encounters Using Example-based and Machine Learning Techniques.* **Serguei V.S. Pakhomov, James D. Buntrock, Christopher G. Chute.** Issue 5, 2006, Journal of the American Medical Informatics Association, Vol. Volume 13, pp. 516-525. ISSN 1067-5027, 10.1197/jamia.M2077.

10. *Medical coding classification by leveraging inter-code relationships.* **Yan, Yan, et al.** New York : s.n., 2010. Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining. ISBN: 978-1-4503-0055-1.

11. *Lexically-Triggered Hidden Markov Models.* **Kiritchenko, Svetlana and Cherry, Colin.** Portland, Oregon, USA : Association for Computational Linguistics, 2011. Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics. pp. 742-751.

12. *Machine learning and features selection for semi-automatic ICD-9-CM encoding.* **Medori, Julia and Fairon, Cédrick.** Los Angeles : Association for Computational Linguistics, 2010. Proceedings of the NAACL HLT 2010 Second Louhi Workshop on Text and Data Mining of Health Documents. pp. 84-89.

13. 2007 International Challenge: Classifying Clinical Free Text Using Natural Language Processing. *Computational Medicine Center.* [Online] [Hivatkozva: 2011. 10 20.] http://computationalmedicine.org/challenge/previous.

14. **Horváth, Gábor.** *Neurális hálózatok.* Budapest : Panem Könyvkiadó Kft., 2006.

15. 2007 Medical Natural Language Processing Challenge Details. *Computational Medicine Center.* [Online] [Hivatkozva: 2011. 10 20.] http://computationalmedicine.org/webfm_send/5.

16. **Manning, Christopher D., Raghavan, Prabhakar és Schütze, Hinrich.** Introduction to Information Retrieval. Cambridge : Cambridge University Press, 2008.

17. *Understanding Inverse Document Frequency: On theoretical arguments for IDF.* **Robertson, Stephen.** 2004, Journal of Documentation Vol. 60, pp. 503-520.

18. *Inverted Files for Text Search Engines.* **Zobel, Justin and Moffat, Alistair.** 2006, ACM Computing Surveys Vol. 38 No. 2, Article 6.

19. **Mitchell, Tom M.** Generative and discriminative classifiers: naive Bayes and logistic regression. *Machine Learning.* 2010.

20. **Russel, Stuart J. és Norvig, Peter.** *Mesterséges intelligencia modern megközelítésben.* Budapest : Panem, 2005.

21. *An empirical study of the naive Bayes classifier.* **Rish, I.** 2001. Proceedings of IJCAI-01 workshop on Empirical Methods in AI. pp. 41-46.

22. *Improving the Performance of Multilayer Perceptron through Empirical Maximum Likelihood (EML) Learning Rule.* **Semnani, S. and Holt, M. J. J.**

Loughborough, England : s.n., 1990. IAPR Workshop on Machine Vision Applications. pp. 109-112.

23. *A training algorithm for optimal margin classifiers.* **Boser, B., Guyon, I. and Vapnik, V.** Pittsburgh : ACM, 1992, Fifth Annual Workshop on Computational Learning Theory, pp. 144-152.

24. **Wikipedia.** [Online] 2011. [Hivatkozva: 2011. 10 14.] http://en.wikipedia.org/wiki/Support_vector_machine.

25. *On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines.* **Crammer, Koby and Singer, Yoram.** 2001, Journal of Machine Learning Research, pp. 265-292.

26. *A comparison on methods for multi-class support vector machines.* **Hsu, C.-W. and Lin, C.-J.** 2002. IEEE Transactions on Neural Networks. pp. 415-425.

27. *Probability Estimates for Multi-class Classification by Pairwise Coupling.* **Wu, Ting-Fan, Lin, Chih-Jen and Weng, Ruby C .** 2004, Journal of Machine Learning Research, pp. 975-1005.

28. **Lin, Chih-Jen.** LIBLINEAR -- A Library for Large Linear Classification. *Machine Learning Group at National Taiwan University.* [Online] 2011. [Hivatkozva: 2011. 11 5.] http://www.csie.ntu.edu.tw/~cjlin/liblinear/.

29. **Waldvogel, Benedikt.** Java version of LIBLINEAR. [Online] [Hivatkozva: 2011. 12 6.] http://www.bwaldvogel.de/liblinear-java/.

30. **Joachims, Thorsten.** Multi-Class Support Vector Machine. [Online] 2.20, 2008. 8 14. [Hivatkozva: 2011. 10 14.] Cornell University Department of Computer Science. http://svmlight.joachims.org/svm_multiclass.html.

31. *Measuring Diagnoses: ICD Code Accuracy.* **O'Malley, Kimberly J., et al.** s.l. : Health Research and Educational Trust, October 2005, HSR: Health Services Research, pp. 1620-1639.

32. *Classification with Hybrid Generative/Discriminative Models.* **Raina, Rajat, és mtsai.** hely nélk. : MIT Press, 2003. In Advances in Neural Information Processing Systems 16.