

TDK Dolgozat

Véges Márton

2012

Vélemények érzelemosztályozása RapidMinerrel

Véges Márton, prog. inf. BSc III. évf.
TDK dolgozat, Budapest 2012
ELTE IK



Témavezető: Rácz Gábor
Doktorandusz
ELTE IK, Információs Rendszerek Tanszék

A dolgozat elkészítését az ELTE IK, Információs Rendszerek Tanszék KMS projektlaborja támogatta.



Tartalomjegyzék

1.	Az érzelmosztályozás.....	4
1.1.	Felhasználási területek.....	5
1.2.	Az érzelmosztályozás részfeladatai	6
2.	Két korábbi eljárás.....	8
3.	RapidMiner.....	12
3.1.	A repositoryk	13
3.2.	Processzek és felépítésük.....	15
3.3.	Adatok és adattípusok.....	16
4.	SVM	17
4.1.	Formális leírás.....	18
4.2.	Paraméteroptimalizálás.....	21
5.	Az algoritmus leírása	24
5.1.	A használt paraméterezések	26
5.2.	Betanítás.....	28
5.2.1.	LoadComments.....	28
5.2.2.	CollectAdjectives	29
5.2.3.	CalculateAdjectiveWeights.....	30
5.2.4.	WeightedAdjectiveModeler	32
5.2.5.	Az optimális paraméterek	34
5.3.	Használat	34
5.3.1.	WeightedAdjectiveEval.....	34
5.4.	Futási idő	35
5.5.	Problémák a processzek írásakor	36
6.	Eredmények.....	37
7.	Összefoglalás	39
8.	Irodalomjegyzék	41

1. Az érzelemosztályozás

Mások véleményére mindig is sokat adtak az emberek. Komolyabb döntések meghozatala előtt gyakran kikérjük ismerőseink tanácsát és segítségét. Barátaink tapasztalatai illetve élményei gyakran befolyásolják a mi döntéseinket is, például egy új film megtekintésekor. Ma már nem csak barátainkra hagyatkozhatunk, hanem internetes fórumokon keresztül ismeretlenek véleményét is megtudhatjuk egy-egy filmről vagy könyvről. Ezért fontos a gyártóknak, hogy ezeket a bejegyzéseket ők is figyelemmel tudják kísérni. Ez hatalmas munka lehet és gyakran nem is pontosan definiált mikor pozitív az emberek megítélése és mikor nem. Esetleg árnyaltabb a termék megítélése: míg egyes tulajdonságait dicsőítik, a többit lehúzzák. Az érzelem-analízis feladata ezt megkönnyíteni: a kapott forrásszövegeket dolgozza fel és osztályozza egy vagy több szempont szerint.

Az osztályozás több tengely mentén történhet, vegyük például a következő szöveget¹:

“I bought an iPhone a few days ago. It is such a nice phone. The touch screen is really cool. The voice quality is clear too. It is much better than my old Blackberry, which was a terrible phone and so difficult to type with its tiny keys. However, my mother was mad with me as I did not tell her before I bought the phone. She also thought the phone was too expensive, ...”

Itt dicsérik az iPhone képernyőjét, hangminőségét és billentyűzetét. Azonban magát a készüléket túl drágának tartják. Tehát az érzelemosztályozás során meg kell állapítani, hogy melyik tulajdonságról milyen véleménye van az írónak. Továbbá lehetséges, hogy nem ismerjük előre a szempontokat, amik mentén a véleményezés történik. Ekkor ezeket is nekünk kell kinyerni.

Az osztályozás történhet előre meghatározott kategóriákba sorolással (például pozitív, semleges, negatív). Legtöbbször egy tanítóhalmazon betanított osztályozó algoritmus segítségével kapjuk meg a modellt, amit az üzenetek besorolására fogunk használni. Lehet, hogy nem elég a fenti három csoport és finomabb eredményt akarunk kapni. Ekkor egy valós számot rendelhetünk az elemzett szöveghez, ami egy megfelelően választott skálán a teljesen negatívtól teljesen pozitív besorolásig terjed.

Az osztályozáshoz kapcsolódó feladat az érzelemkifejező mondatok felismerése. Meg kell tudnunk különböztetni a semleges kifejezéseket az érzelmekkel telítettektől. Ez egy újabb számítógépes nyelvészeti probléma, amivel meg kell tudni birkózni. Tovább nehezíti a dolgot, hogy az érzelem függ a szöveggörnyezettől. Végso soron egy szótár összeállítása a cél, amiben fel vannak sorolva az érdekes/fontos kifejezések.

Egy további ide kapcsolódó terület a spamvélemények kiszűrése. A bevezetőben említettük, hogy az internetes fórumoknak nagy hatásuk van az emberek döntéseire. Ezt természetesen a gyártók is

felismerték, így vannak alkalmazottaik, akik pozitív véleményeket írnak a saját termékekhez és negatívat a konkurenciáéhoz. Itt a szokásos spam-szűrési technikák nem működnek, hiszen a bejegyzéseket emberek írják, és egy fórumon jelennek meg teljesen valódi hozzászólásként. A szerző többi hozzászólásának analizálásával dönthetjük el mennyire kiegyensúlyozott a véleménye. Azonban óvatosan kell eljárni, hogy ne keverjük össze valódi emberek valódi pozitív tapasztalatával, és ehhez kell a véleményanalízis.

1.1. Felhasználási területek

Az érzelemanalízis alkalmazási területei elég szélesek. Egy cég feldolgozhatja a róla szóló bejegyzéseket és egyszerűen megtudhatja a termékei hiányosságait és előnyeit². Erre eddig külön felmérést kellett indítani a felhasználók körében, ami rengeteg extra költséggel és teendővel jár (például gondoskodni a válaszadók megfelelő statisztikai eloszlásáról figyelembe véve a végzettséget, jövedelmet, stb.), vagy fókuszcsoportokat használni. Az interneten elérhető információk automatikusan történő feldolgozásával azonban ezek a problémák megkerülhetők, hiszen a felhasználók eloszlása egyezni fog a kommentelőkével. Külön pénzt se kell költeni minden egyes felmérésre termékenként. Ezen kívül nem csak a saját áruinkról tudhatnak meg új információkat, hanem a konkurencia termékeiről is ugyanazzal a programmal. Ez is plusz extra tudást jelent további költségek nélkül.

A dolgot a másik oldalról, a fogyasztó szemszögéből is nézhetjük. Neki szintén csak egy módja van a tájékozódásra, ha végigolvas több száz hozzászólást és elemzést a témában. Ez elrettentő és időigényes feladat, nem véletlen, hogy sok ember nem vállalkozik rá. Egy keretrendszer, ami megteszi ezt helyette és kigyűjti a főbb pontokat pro és kontra nagy segítség lenne. Ez hatékonyan növelné a fogyasztói tudatosságot, ami végső soron másoknak is jót tesz a verseny és minőség javításával³.

A termékek véleményezéséhez hasonlóan vizsgálható az egyes politikai pártok szereplése és népszerűsége. Jelenleg ez csak közvélemény-kutatási adatokkal lehetséges. Itt is hasznos lenne mindkét félnek (politikus és választó) a többi ember véleményének összefoglaló megismerése. Az újságíróknak pedig egy további eszközt adna az elemzéseik megírásához. Például a New York Times oldalán jelent meg egy Twitter üzeneteket osztályozó alkalmazás a 2010-es amerikai képviselői választások időszakában⁴. Ez valós időben mutatta az egyes pártokra vonatkozó véleményeket, összegezve ki hogy áll éppen. Folyamatosan dolgozta fel a beérkező tweeteket, és osztályozta az egyes pártokról alkotott véleményeket, az eredményeket egy grafikonon megjelenítve.

Egy további felhasználási lehetőség a reklám-elhelyezések optimalizálása. A reklámszolgáltatók elsődleges feladata növelni a hirdetések hatékonyságát és erre a legjobb módszer az, ha a megfelelő embernek mutatjuk meg. Általában a felhasználó által meglátogatott weboldalak és földrajzi elhelyezkedés alapján keresnek releváns hirdetéseket. Egy újabb lehetőség a vásárlók véleményének figyelembe vétele: egy olyan helyre nyilván megéri reklámot elhelyezni, ahol dicsérik az adott terméket. Ez jóval megnöveli az átkattintások arányát, ami mind a reklámszolgáltatónak, mind a reklámot megrendelőnek jó. Ha pedig egy helyen szidnak egy terméket, oda a konkurencia reklámját kell elhelyezni.

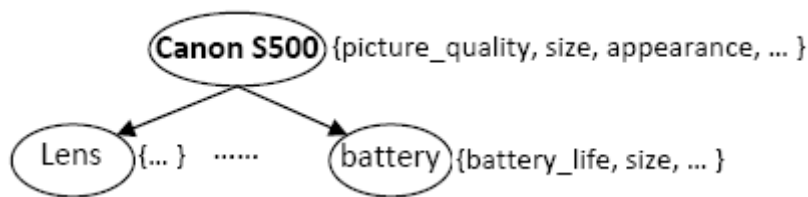
1.2. Az érzelemosztályozás részfeladatai

Ahhoz, hogy sikeresen választ adjunk a fenti problémákra több kisebb feladatot kell megoldani. Például gyakori algoritmus a melléknevek eloszlásának elemzése. Ehhez szükség van egy úgynevezett POS-taggerre (part-of-speech tagger), ami nyelvtani elemzés után besorolja a szavakat a megfelelő szófajokba. Szerencsére ezek az eszközök már rendelkezésre állnak, és megfelelő pontossággal működnek. A Stanfordi Egyetem NLP csomagja 97.3% pontossággal működik a standard Wall Street Journal korpuszon⁵. A dolgozatban is ezt az osztályozót használok.

El kell még dönteni, hogy egy szövegben valóban van-e valamilyen pozitív vagy negatív kritika megfogalmazva, vagy teljesen semleges. Tipikus példa lehet egy telefonokkal foglalkozó fórumon a Wifi beállítására vonatkozó kérdés. Ez gyakran maga is egy osztályozási feladatként merül fel: a szöveget az objektív és szubjektív kategóriák valamelyikébe kell sorolni⁶. Az objektív és szubjektív mondatok keveredhetnek egy dokumentumon belül. Egy, amúgy objektív cikkben, megjelenhetnek vásárlói vélemények. Vagy egy szubjektív hozzászólás végén előfordulhat a fent említett Wifi-re vonatkozó kérdés. Az objektív mondatok szűrése javíthat az érzelemosztályozás pontosságán is. Néhány használt módszer⁷: azt találták, hogy az érzelemmel rendelkező mondatok hasonló szerkezetűek, így egy megfelelően címkézett tanítóhalmaz mondataival hasonlították össze az osztályozandó szövegben levőket. Ahol nagy volt a hasonlóság, ott szubjektívként osztályozták a kifejezést. Egy másik módszer a naiv Bayes algoritmust használja. A használt paraméterek közt szerepeltek a szavak eloszlása, illetve a szófajok is.

További probléma szétválasztani egy termék komponenseire vonatkozó véleményeket. Például egy fényképezőgépnek ilyen részegységei az akkumulátor és a lencse. Még ezen alkotóelemeknek is lehetnek további "attribútumai", például akkuélettartam, fókusztávolság, stb.

Ezeket a komponenseket és attribútumokat egy fán ábrázolhatjuk⁸:



1. ábra Egy fényképezőgép aspektusai⁸

A fa csúcsaiban vannak a termék komponensei, az attribútumok pedig mellettük szerepelnek. Egy vélemény ezek bármelyikére vonatkozhat. Ezért összefoglaló néven aspektusnak (vagy tulajdonságnak) hívjuk ezen részek és attribútumok összességét. Ez a fa néha a rendelkezésünkre áll, ha tudjuk milyen termékről keresünk éppen véleményt. Ezt kézzel is össze lehet állítani, meghatározva az adatbányászat szempontjait.

Azonban az is előfordulhat, hogy nem áll rendelkezésünkre ilyen fa. Vagy azért, mert nem tudjuk mit analízálunk éppen, vagy mert nem vagyunk biztosak benne, hogy mik az osztályozás számára a legjobb szempontok. Utóbbi eset lényeges a vélemények feldolgozásakor: meg kell határozni azokat a kulcspontokat, ahol a termék jól vagy rosszul teljesít. Tehát az osztályozás előtt az algoritmusnak kell kikeresnie a megfelelő aspektusokat. Egy gyakran használt eljárás, hogy azok a főnevek lesznek az aspektusok, amik közelében sok melléknév van. Problémát okozhat, ha ugyanarra az aspektusra különböző szinonimákat használnak (képernyő és kijelző), ekkor ezeket is össze kell párosítani.

Rögtön a fentiekből következik egy újabb probléma: a megfelelő érzelemkifejező szavakat a megfelelő termék megfelelő aspektusához kell párosítani. Egy összehasonlító kritikában több tárgyról is szó eshet és könnyen lehet, hogy a termékek neveire csak névmásokkal hivatkoznak, ezzel is nehezítve a párosítást. Ha egy termék különböző aspektusairól van szó, akkor az aspektusok kiválasztásakor használt módszer jól jöhet: egy érzelemkifejező szó ahhoz a tulajdonsághoz kapcsolódik, amihez a mondatban a legközelebb van. Egy másik módszer a tagmondatokra bontás, hiszen egy tagmondatban már nagyobb eséllyel csak egy dologról szól a vélemény. Természetesen még mindig lehetnek kivételek ("Apple is doing very well in this lousy economy").

Egy újabb feladat meghatározni a megfelelő érzelemkifejező szavakat és kifejezéseket. Itt az adja a nehézséget, hogy különböző szöveggörnyezetben más lehet a jelentése ugyanannak a szónak. Például a hosszú akkuélettartam egy jó dolog, de a hosszú betöltési idő már nem. Több megközelítés létezik: a legegyszerűbb, hogy kézzel összeállítunk egy listát. Ez hatékony és elég egyszer elvégezni, továbbá az adott témához tudjuk igazítani a kiválasztott szavak polaritását. A második a korpusz alapú megközelítés. Itt kötőszavak segítségével kapcsoljuk össze a szavak jelentését: az "és", "vagy"

szavak hasonló jelentésre utalnak, a "de", "ellenben" pedig ellentétesre. Ha elkészítünk egy kiindulóhalmazt, amiben néhány kiválasztott szót már besoroltunk, akkor a fenti kapcsolatokon keresztül bejárhatjuk a szavak egy gráfját, és a fenti szabály szerint rakjuk az új szavakat a pozitív és negatív kategóriákba. A harmadik megközelítés is egy kiindulóhalmazt vesz első lépésként. Ezután egy szinonimaszótár segítségével köti össze az azonos jelentésű szavakat, és így gyűjti ki őket rekurzívan az előző módszerhez hasonlóan.

2. Két korábbi eljárás

A gépi tanuló algoritmusoknak két fő fajtája van: felügyelt és felügyelet nélküli betanítás. Az érzelmosztályozásban mindkét fajta módszert használják, bár az előbbire jobb eredmények születtek. Köszönhető ez annak a korábban már jelzett problémának, hogy ugyanaz a szó különböző szöveggörnyezetekben hol pozitív, hol negatív jelentéssel bír. Az alábbiakban vázlatosan bemutatjuk mindkét megközelítést egy-egy példán keresztül.

Az alábbi felügyelet nélküli osztályozás Turney munkája⁹. Az eljárás három fő lépésből áll. Az elsőben elvégezzük a szavak POS-taggelését (szófaji besorolását). Ezután vesszük az összes 2-gramot. Az n-gram az n egymás után jövő szavak halmazát jelöli. Esetünkben tehát az összes szomszédos szópárt tekintjük. Ezek közül kiválasztjuk azokat, amiknek a szófaji besorolása megfelel az alábbi táblázat valamelyik sorának.

	Első szó	Második szó	Harmadik szó (nincs megjegyezve)
1.	JJ	NN vagy NNS	akármí
2.	RB, RBR vagy RBS	JJ	sem NN, sem NNS
3.	JJ	JJ	sem NN, sem NNS
4.	NN vagy NNS	JJ	sem NN, sem NNS
5.	RB, RBR vagy RBS	VB, VBD, VBN, VBS	akármí

A táblázatban a Penn Treebank Project által használt címkék vannak. Például a JJ jelöli a melléknéveket, az NN a főneveket¹⁰. A táblázatból kiderül, hogy főleg a jelzős szerkezetek próbálták kiemelni, illetve a segédigével ellátott ígétet. A harmadik szavak ellenőrzése azért kellett, hogy a második szóban levő melléknév valóban az első szóhoz tartozzon és ne a harmadikkal együtt alkosson egy szó szerkezetet. Azonban ez a harmadik szó csak a 2-gramok kiválasztásakor szerepelt, magában szópárban már nincsen benne.

A következő lépésben a kiválasztott kifejezések érzelmi töltetét határozzuk meg. Először a következő képlettel meghatározzuk, mennyire közös a jelentéstartalma két szónak:

$$PMI(word_1, word_2) = \log_2 \left(\frac{P(word_1 \wedge word_2)}{P(word_1)P(word_2)} \right)$$

Itt $P(word_1 \wedge word_2)$ a két szó közös előfordulási esélyét jelenti. Ha a két szó független egymástól, akkor ez az érték egyenlő a nevezőben levő $P(word_1)P(word_2)$ számmal. Tehát a hányados a két szó összefüggőségét írja le. A logaritmus segítségével nyerjük ki a közös információmennyiséget. A P függvény értékét az AltaVista keresőn adott találatok száma határozta meg.

Ezután egy kifejezés pozitív vagy negatív töltetét a következő képlettel számoljuk:

$$SO(phrase) = PMI(phrase, "excellent") - PMI(phrase, "poor")$$

Tehát megvizsgáljuk, hogy a kifejezés jelentéstartalma mennyire azonos az "excellent" és "poor" szavakkal. Ezután vesszük a különbséget. Ha egy kifejezés pozitív jelentéstartalmú, akkor nyilván nagy lesz az "excellent" szóval vett PMI értéke, hiszen nagy a közös jelentés. Ugyanakkor a "poor"-ral vett érték alacsony lesz, tehát a különbség egy pozitív szám lesz. Hasonlóan, egy negatív jelentésű szónál a végső SO érték is egy negatív szám lesz.

A harmadik lépésben az összes kiválasztott kifejezésre kiszámoljuk az SO értéket és ezeket összeadjuk. Ha az összeg pozitív, akkor túlsúlyban vannak a pozitív jelentéstartalmú szó párok, így a teljes dokumentum végül maga is a pozitív kategóriába kerül. Ellenkező esetben a dokumentumot a negatív osztályba tesszük.

Az algoritmus hatékonysága különböző témakörök esetén más és más lett. Az alábbi listában négy különböző tárgykörben mért pontosságot (helyesen osztályozott dokumentumok/összes dokumentum) soroltuk fel:

- autók – 84%
- bankok – 80%
- filmek – 65,8%
- üdülési úti célok – 70,5%

Látható, hogy jelentős eltérések tapasztalhatóak az egyes témák között. A filmértékelések alacsony eredményességét a következőnek tudja be a szerző: egy film gyakran tartalmaz erőszakos és véres jeleneteket, ezen kifejezések PMI értéke pedig nyilván egy nagy abszolút értékű negatív szám, ez

pedig lefelé tolja a film értékelését. Ugyanez igaz a másik irányban is: rossz film is tartalmaz kellemes jeleneteket. Itt a problémát a film és filmjelenetek szétválasztásának hiánya adja. Hasonlóak az indokok az úticélok esetében. Egy rossz nyaralásról szóló bejegyzés hatással van magára az üdülőhely értékelésére, hiába független az magától a helyszíntől. Az előzőeken túl azt is észrevehetjük, hogy ugyanaz a pozitív hangulatú kifejezés előfordul mind a pozitív, mind a negatív osztályokban is, például a "direct deposit" a bankok értékelésekor.

A fentiek ki lehet kerülni felügyelt tanuló algoritmusokkal. Ezeknek az algoritmusoknak szüksége van egy tanítóhalmazra. Ezt elkészíthetjük saját magunk is, egy megfelelő szövegkészletet osztályozva kézzel. Lehetőség van már meglévő adathalmazokkal is dolgozni. Ez azért szerencsés, mert az interneten sokfelé elérhetők adatbázisok, amik felhasználók véleményét tartalmazzák. Ezeken az oldalakon egyaránt van szöveges véleményezés és számszerű értékelés (általában csillagozás formájában). Ez ideálissá teszi őket a betanításhoz, hiszen a szövegek már előre be vannak sorolva a megfelelő kategóriákba. Egy gyakran használt adathalmaz például az IMDb filmes adatbázisból letöltött 1000 pozitív és negatív hozzászólást tartalmazó korpusz, amit Lee és Pong állított össze¹¹. Ezt több mint 100 cikkben és publikációban alkalmazták tanítóhalmazként.

Egy összefoglaló cikkében Pang három különböző osztályozó algoritmust próbált ki¹². Ezek a naiv Bayes, maximum entrópia és az SVM voltak. Mindegyiket több különböző paraméterezéssel is tesztelte. A paraméterezések a következők voltak:

- unigram - az előforduló szavak halmaza
- bigram - az egymás után jövő szó párok
- unigram és bigram egyszerre
- csak melléknevek
- POS tag-gel ellátott szavak

Az unigram feature a fent említett 1-gramnak felel meg. Itt a szövegben előforduló szavakat vizsgálták. A bigram a 2-gramot jelenti. Kipróbálták az algoritmusokat a két halmaz uniójával is, ez a harmadik a paraméterezés. A negyedikben csak a mellékneveket vizsgálták. Szemléletesen gondolhatjuk, hogy a melléknevek jelentősebbek a többi szónál, hiszen valamilyen jellemzőt írnak le. Végül az egyes szavakat szófajlag is megkülönböztették. Ennek célja, hogy olyan kifejezések, mint a következő kettő ne mosódjanak egybe: "I love this film" és "This is a love story". A két mondat közül az elsőben található valódi érzelem, a második egy objektív állítás. Azonban ennek hatása végig elhanyagolható volt mindhárom algoritmusra.

	Naiv Bayes	Maximum Entrópia	SVM
unigram	81,0%	80,4%	82,9%
bigram	77,3%	77,4%	77,1%
unigram+bigram	80,6%	80,8%	82,7%
csak melléknevek	77%	77,7%	75,1%
POS taggelve	81,5%	80,4%	81,9%

A tesztelés során a bigram és a csak mellékneveket vizsgáló paraméterezések kivételével mindig az SVM nyújtotta a legjobb eredményeket. Ebben a két esetben a maximum entrópia volt a leghatásosabb módszer. Összességében a leghatékonyabb az SVM+unigram párosítás volt. Ekkor az elért pontosság 82,9% lett. Az unigram és bigram modellek párosítása az előbbinél kissé rosszabb eredményt adott 82,7%-kal. A csökkenés oka a bigram modell gyengesége, hiszen az önmagába csak 77,4%-os hatékonyságot ért el. Minden más esetben 82% alatt maradt a pontosság.

Látható, hogy a megfelelő feature-ök kiválasztása kritikus az algoritmus szempontjából. A későbbiekben több megközelítést is kipróbáltak. Ezek között szerepelt a szavak szövegbeli gyakoriságának figyelembe vétele. Ekkor egy egyszerű bináris 0-1 paraméter helyett az adott szónak az összes többi szóhoz viszonyított aránya szerepelt. Egyéb súlyozásokat is kipróbáltak, mint például a TF-IDF érték. Egy másik megközelítésben csak az érzelemkifejező szavakat és szókapcsolatokat vették figyelembe. Ezek olyan kifejezések, amikben egyértelműen meghatározható szubjektív töltet van. Ez a fenti melléknév-kiválasztásos technikát fejleszti tovább, hiszen itt már főnevek is szerepelhetnek, illetve az objektív melléknevek (például "piros") kiesnek.

A tagadásokat is figyelembe lehet venni. Ekkor azt várjuk, hogy a "good" és "not good" kifejezések jelentéstartalma ellentétes legyen, és ezt figyelembe is akarjuk venni a kiértékeléskor. A tagadás megállapítására több stratégia is van. Az első esetén egy tagadószó és a következő mondatkötő közt minden szó tagadottnak veszünk. Egy ettől eltérő megközelítésben azok a szavakat tekintjük tagadottnak, amik közel vannak egy tagadószóhoz. A közelséget, mint paramétert is nekünk kell meghatározni.

Speciálisan a filmkritikák osztályozásánál merült fel, hogy ne csak a pozitív-negatív polaritást próbáljuk megjósolni, hanem a kritikához az író által adott értékelést (csillagozás 1-től 10-ig) is. Ekkor jóval több osztályunk van, ami nagyobb bizonytalanságot jelent. A probléma nehézségét jelzi, hogy amikor embereknek kellett megtippelni a csillagok számát, csak az esetek 70%-ban találták el egy csillagnyi hibával az eredeti értéket¹³. A feladatot nem csak osztályozási problémaként, hanem regresszióként is lehet tekinteni. Ekkor a csillagok számát, mint egy folytonos függvény értékének

kerekítéseként lehet felfogni. Koppel és Schler úgy találták, hogy ez a megközelítés jobb eredményeket ad az osztályba sorolások megoldásánál.

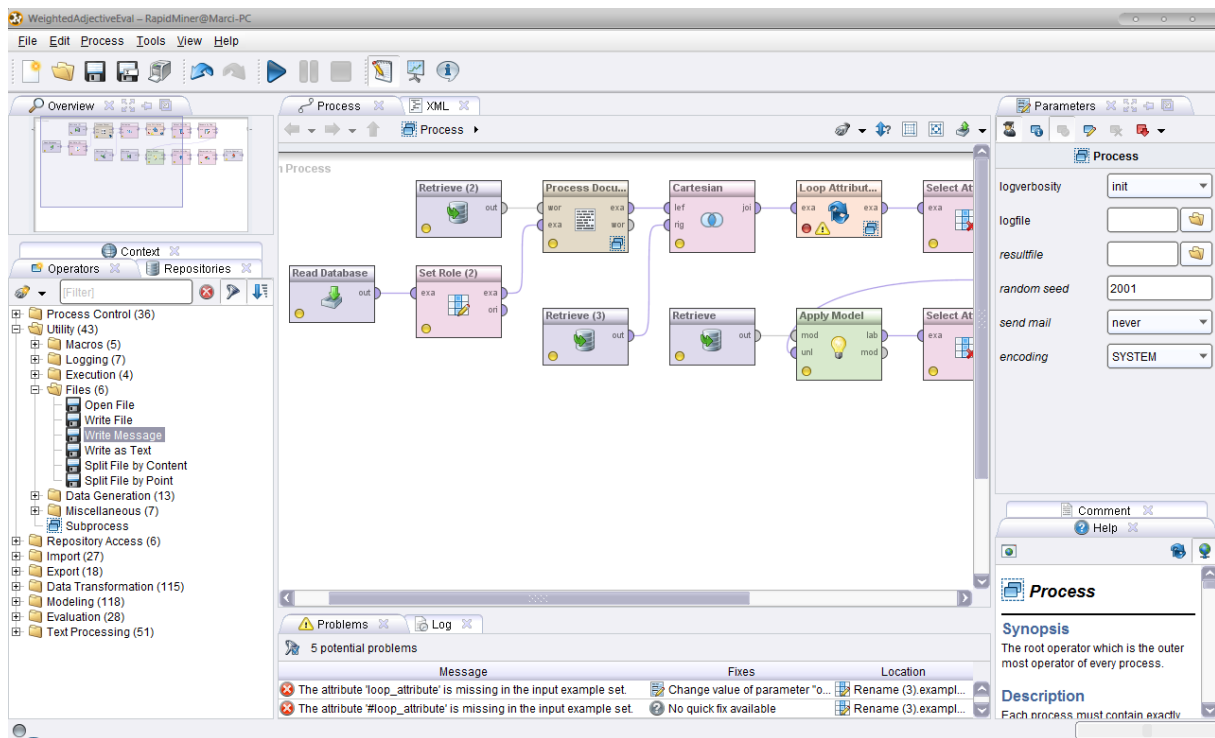
A felügyelet melletti tanításnál nagy szerepe van a tanítóhalmaznak. Az osztályozás hatékonyságát nagyban rontja, ha a tanítóhalmaz témája eltérő az osztályozott szövegtől. Ezt az általam írt osztályozó tesztelésekor is tapasztaltam: a modellt filmkritikákon tanítottam be, itt a hatékonysága 83%-os volt. Azonban gazdasági fórumokra alkalmazva az eredmény jóval kisebb, 75% körüli volt. Ennek illusztrálására tekinthetjük a következő mondatot: "this vacuum cleaner really sucks". Ez nyilván dicséri a porszívót, azonban a "really sucks" kifejezés minden más helyen egy nagyon erős negatív érzelmű. Ennek kiküszöbölésére egy módszer, hogy a betanító szöveget és az osztályozandó szöveget mondatonként hasonlítták össze, hogy kinyerjék az azonos szerkezeteket és érzelmű kifejező szavakat¹⁴.

Az eltérő témájú tanítóhalmazok mellett az eltérő nyelv is gondot okoz. Néha nem érhető el egy adott nyelvhez megfelelő mennyiségű felcímkézett adathalmaz. Angol nyelven többfajta osztályozott adathalmaz is van, azonban magyarul még nem létezik ilyen. Ebben az esetben gépi fordítással segíthetünk a helyzeten. Ez nem mindig ad megfelelő eredményeket. Példaként említhetjük a kínai "pianyi" szót, amit gyakran használnak dicsérő kifejezésként. Ennek ellenére a Google Translate "cheap"-nek fordítja, amit inkább negatív értelemben használnak¹⁵. Egy másik eljárás az LDA algoritmus egy módosított változatát használja¹⁶. Itt azt találták, hogy jobb eredményeket kaptak, amikor egyszerre használtak német és angol tanítóhalmazokat német szövegek osztályozására, mint ha csak németet használták volna.

3. RapidMiner

A RapidMiner egy széles körben használt adatbányász szoftver. Ismeri a legtöbb gépi tanuló algoritmust, szövegbányászati és üzleti intelligenciai eljárásai is vannak. Egyaránt használják oktatásban, kutatásban és ipari alkalmazásban. A KDnuggets adatbányászati magazin felmérése szerint a 2010-ben a valódi projekthez legtöbbet használt adatbányászati program volt¹⁷. A program nyílt forráskódú, az Affero General Public License alatt terjeszthető¹⁸.

Eredetileg a Dortmundi Műszaki Egyetem Mesterséges Intelligencia laborjában kezdték fejleszteni még 2001-ben. Eredeti neve YALE volt, ez később változtatták jogi okokból RapidMinerre. 2006-ban céget alapítottak a további fejlesztéshez és terméktámogatáshoz Rapid-I néven. A vállalat gyors fejlődésnek indult, hála a könnyen megtanulható rendszernek és a sokfajta implementált algoritmusnak.



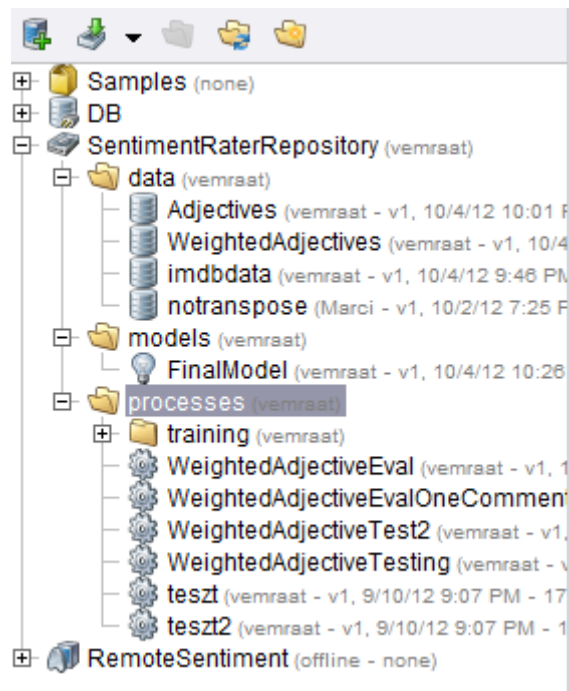
2. ábra A RapidMiner kezelőfelülete

A céljuk egy gyors prototípus-fejlesztésre használható platform volt, amit kifejezetten adatbányászathoz terveztek¹⁹. A prototipizálás segít felmérni a projekt várható erőforrásigényét és hatékonyságát. Ez lehetőséget nyújt arra, hogy még a fejlesztés korai szakaszában eldöntsenek néhány kritikus tervezési kérdést. Ehhez számos kritériumnak kellett megfelelni: először is rugalmasnak kell lennie az elérhető adatfeldolgozó és analízáló eszközök terén. Ez azt jelenti, hogy a különböző algoritmusok széles választékát kell elérhetővé tennie. Ezeknek az algoritmusoknak könnyen kicserélhetőnek kell lenniük, tehát azonos felülettel kell rendelkezniük. Továbbá sokfajta adattípust ismerni kell, idősoroktól kezdve az adattábláig. Végül pedig ajánlott a felhasználó számára könnyen megtanulhatónak lennie, ne igényelje egy különleges nyelv megtanulását. Ezért választották, hogy vizuálisan lehessen programozni, kódolás nélkül.

A RapidMinerben írt programok (processzek) megértéséhez ismerni kell néhány alapvető fogalmat. Ezeket a következőekben részletesen is kifejtjük.

3.1. A repositoryk

Az adatfeldolgozás során számos új file keletkezik: processzek, bemeneti adatok, tesztadatok, a feldolgozás során keletkezett átmeneti táblák, stb. Ezeket nyilván jó lenne valamilyen mappaszerkezetben rendezetten tárolni. Az operációs rendszer fájlrendszere jó választásnak tűnik elsőre, de ha nagy mennyiségű adattal dolgozunk, akkor gyorsan kényelmetlenné válik a feldolgozó algoritmusoknak. Ezért hozták létre a RapidMiner-ben az adatok tárolására szolgáló repositorykat.



3. ábra Az érzelemosztályozó által használt repository

Ezek tartalmazhatnak mappákat, a mappák pedig processzeket illetve egyéb adatokat. Egy elem elérése ebben a rendszerben hasonló módon történik, mint más fájlrendszerek esetén:

//RepositoryNev/Mappa/Almappa/File

Itt a // jelenti, hogy a gyökérkönyvtárból indulunk, és a következő elem egy repository lesz. Ezután a szokásos formában adjuk meg egy fájl elérési útvonallát. Lehetőség van még az aktuális repository gyökérkönyvtárából indulni, ekkor a / jelet kell használni. A fenti elérési úttal ekvivalens a következő, ha az aktuális repository a 'RepositoryNev':

/Mappa/Almappa/File

Relatív címzést is lehet használni, ha teljesen elhagyjuk a kezdő / jeleket.

A valódi célja a repositorynak, hogy adatokat tudjunk benne tárolni, amik a RapidMiner által gyorsan olvasható és betölthető formátumban vannak. Mindenfajta adat tárolható a repositorykban, amit egy processz outputjaként kaphatunk, például: adathalmazok, szólisták, gépi tanuláshoz modellek. A 3. ábra látható a megírt érzelemosztályozó repositoryja. A FinalModel elem például a betanítás során kapott modell. Amikor az egyes filmkritikákat akarjuk besorolni, akkor innen fogjuk betölteni a modellt az osztályozó processzbe.

A repositoryban tárolt adatokról további adatokat is tárol a rendszer, ezek az úgynevezett meta-adatok. Néhány példa, mik lehetnek ezek:

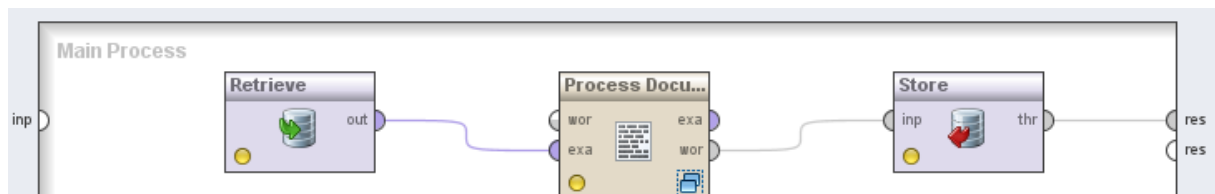
- sorok száma
- oszlopok száma, típusa
- attribútumonként a hiányzó értékek száma
- adatok eloszlása oszloponként

Ezek segítségével gyors áttekintést kaphatunk az adathalmazunkról és egyszerű hibakeresésre is használható.

Végül, a RapidMiner lehetővé teszi, hogy a repositorykat külön gépen tároljuk. Ekkor a kliens szempontjából az adatok szerkesztése, elérése, létrehozása, stb. ugyanúgy működik, mint a helyi gépen tárolt halmazok esetén. A távoli gépen levő processzeket futtathatjuk akár a saját, akár magán a távoli gépen. Ez kényelmes, ha nagy adatokkal szeretnénk dolgozni, amik meghaladják egy személyi számítógép kapacitását. Az eredményeket pedig ugyanúgy megtekinthetjük a helyi gépen.

3.2. Processzek és felépítésük

A RapidMinerben írt programokat processzeknek hívjuk. A processzek operátorokból állnak, ezek a kezelőfelületen dobozokként jelennek meg. Az alábbi ábrán látható egy processz, ami a repositoryban tárolt adatokat feldolgozza, és az eredményt szintén a repositoryba menti.



4. ábra Egy RapidMiner processz

A Retrieve operátor adatokat tölt be a repositoryból, a Process Documents feldolgozza ezeket, majd a Store eltárolja az eredményt. Az egyes operátorok ki- és bemenetét a szélükön található félkörök jelképezik. Ezeket portoknak hívjuk. Az adatáramlás irányát az összekötő kábelekkel határozhatjuk meg. Az egyes portok adattípusát a port mellett olvashatjuk rövidítve.

Például a Process Documents operátor második bemenete az `exa` címkét kapta. Ez az ExampleSet rövidítése. A RapidMiner ExampleSet-nek hív minden táblázatos formában tárolt adatot. Az elnevezés alapja, hogy a táblázat egy sorát a táblázat által leírt típus egy példányának veszik. Például, ha a táblázatunk emberekről tartalmaz adatokat, akkor egy sor egy embernek felel meg. Egy konkrét

ember adatai egy *példát* adnak. A dokumentációban illetve a programban néha használják még a DataSet elnevezést is.

A Store operátor kimenete `thr`, azaz through típusú. Ez azt jelenti, hogy az operátor változás nélkül továbbengedi a bemenő adatot. Ez a viselkedés logikus, hiszen az adatmentés után még további feldolgozásra szükség lehet a folyamat végeztéig. Esetünkben azonban ilyen nem történik, hanem kivezetjük az outputot a szerkesztési terület `res` portjára. Ez a result rövidítése. Az ide kerülő adatok a processz végeztével kiírásra kerülnek.

Az `inp` port segítségével fogadhat a processz bemenő adatot. Ebben a példában, mivel repositoryból olvasunk, erre nincsen szükség, tehát a port összekötetlenül maradt.

Bug-kereséshez és az adatok jobb megértéséhez hasznos, hogy futás után az egyes operátorok által adott közbenső eredmények is megtekinthetők a legtöbb esetben. Ehhez elég az adott operátorra kattintva kiválasztani a *Show Results* menüpontot. Néhány feltételnek azért teljesülnie kell. Először is az átmeneti adathalmaz mérete nem lehet túlságosan nagy, különben a RapidMiner memória-kímélési okokból kitörli azokat futás közben. Másodszor, az optimalizálás miatt a RapidMiner nem minden esetben menti el a teljes új adathalmazt, csak a változásokat. Néha azonban az eredeti adathalmaz felülíródik egy későbbi operátor hatására, így már nem lehet visszaállítani a közbenső állapotokat.

3.3. Adatok és adattípusok

Korábban említettük, hogy a táblázatos adathalmazokat a RapidMiner ExampleSet-nek hívja, a sorait pedig example-nek. A táblázat oszlopait attribútumnak nevezzük az adatbázis-kezelő rendszerekhez hasonlóan. Az egyes attribútum típusai a következők lehetnek:

- text – szöveget tartalmazó oszlop
- numeric, real, integer – számok leírására szolgál
- nominal, binomial, polynomial – olyan oszlopok, amik értékkészlete véges sok elemből áll. Binomial típus esetén csak két elem, polynomial esetén pedig több elem is lehet.
- date, time – dátumok és időpontok tárolására van

Különösen fontos megérteni a nominal típust. Erre nagy szükség van osztályozó algoritmusok esetén, amikor az osztályozás célváltozója véges értékkészletű kell legyen. Az összes operátor, ami egy ilyen modellt épít, olyan adathalmazt vár bemenetül, amiben van egy nominal típusú oszlop. A modell által létrehozott osztályok az oszlop értékkészletének fognak megfelelni. A nominal speciális esete a binomial, amikor csak két különböző értéket vehet fel az oszlop. Tipikus példa rá az igaz - hamis

logikai változók. A RapidMinerben nincs is külön logikai típus, ezeket binomial-ként kell megvalósítani.

Az egyes oszlopoknak nem csak típusa van, hanem szerepe is. Ez az attribútum feladatának leírására szolgál. Néhány fontosabb szerep:

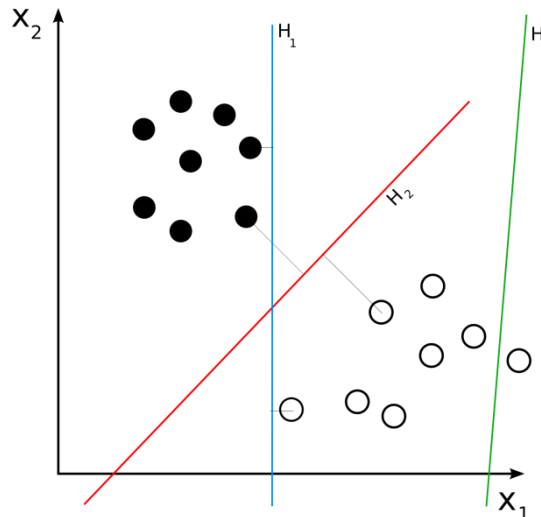
- regular
- label
- ID
- weight

Az ID szerepű oszlopból csak egy lehet a táblázatban. Ez az egyes sorok azonosítója, az adatbázis-kezelőknél megszokott működik: minden értéke különböző, és nem hiányozhat egy sorból sem. A label szerepű oszlopok a tanuló algoritmusok célváltozói. A tanuló algoritmus által adott modell ezt az oszlopot próbálja kifejezni a többi oszlop függvényében. Ha az oszlop típusa nominal, akkor osztályozó algoritmust lehet használni, ha szám, akkor regressziót. A weight-tel jelölt oszlopok súlyozásra szolgálnak. Az adott sor súlyát fejezik ki a célváltozóra vonatkozóan. Végül a regular szerepű oszlopok azok, amik nem rendelkeznek semmilyen különleges tulajdonsággal, hanem csak szimpla adatot tartalmaznak. Betanításkor alapesetben csak az ilyen oszlopok értékeit veszi figyelembe a RapidMiner.

Megjegyezzük, hogy további szerepek is létrehozhatók a felhasználó által. Ezeket az attribútumok további megkülönböztetésére használhatjuk, illetve a saját magunk által írt kiegészítőkhöz is alkalmazhatjuk.

4. SVM

Az SVM (support vector machines) egy, a gépi tanulásban gyakran használt algoritmus. Vapnik és Cortes javasolta 1995-ben²⁰. Egyaránt használható osztályozásra és regresszióra. A módszer lényege a következő: a tanítóhalmaz minden elemét beleképezi egy elég nagy dimenziós euklideszi térbe. Ezután a két kategória közt keres egy elválasztó hipersíkot úgy, hogy minél nagyobb legyen a rés a két kategória közt. Egy új adat osztályozása a következő módon zajlik: ugyanezt a leképezést használva belevisszük az elemünket az euklideszi térbe. Itt attól függően rakjuk be a megfelelő osztályba, hogy a sík mely oldalára esik. A módszer egy változatában az elválasztó felületnek nem szükséges síknak lennie, bonyolultabb formát is felvehetnek.



5. ábra Példák elválasztó síkokra²¹

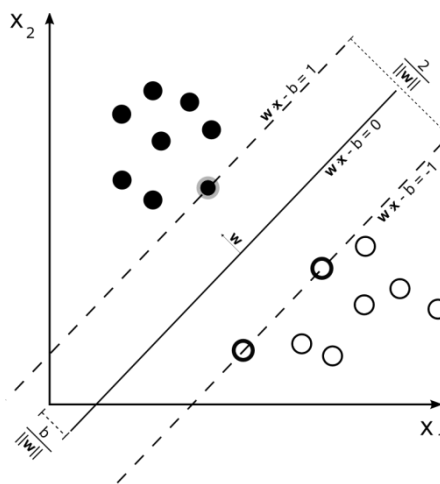
A képen látható egy példa adathalmaz. A két kategóriát fehér illetve fekete körök jelzik. A H_1 és H_2 hipersíkok egyaránt elválasztják a két osztályt, a H_3 pedig nem. A többfajta lehetséges elválasztó hipersík közül azt kell választani, ahol a minimális távolság a síktól maximális. A képen H_1 jóval közelebb van az adatokhoz, mint az optimális H_2 .

4.1. Formális leírás

Az alábbi leírás a Wikipedia megfelelő szócikkén alapul²², az ábrák is onnan származnak. Jelölje a tanítóhalmazunkat D , elemszámát n . A halmazban levő pontokat a következő formában írhatjuk:

$$\{(x_i, y_i) : x_i \in \mathbb{R}^p, y_i \in \{-1, 1\}\}$$

Itt x_i egy p -dimenziós pont, ez jelzi a leképezett tanítóhalmazbeli elemet. Az y_i érték jelöli, melyik osztályba esik a pont. Ismert, hogy a w normálvektorú síkokat a $wx - b = 0$ egyenlettel lehet leírni.



6. ábra A két elválasztó sík

Először tegyük fel, hogy a két kategória elválasztható egy síkkal. Ekkor két párhuzamos síkkal is elválaszthatók úgy, hogy ezen síkok közé nem esik egy pont se. A két sík távolságát akarjuk maximalizálni, jelöljük ezt d -vel. A d változót nevezik még margin-nak is. Most a két sík egyenlete a következő módon írható:

$$wx - b = 1$$

és

$$wx - b = -1$$

Ekkor a két sík távolságát egyszerűen meghatározhatjuk, az eredmény $d = \frac{2}{\|w\|}$. d -t akarjuk maximalizálni, tehát $\|w\|$ -t kell minimalizálni. A következő egyenlőtlenségek biztosítják, hogy egy pont se essen a két sík közé:

$$wx_i - b \geq 1$$

az első osztály minden x_i pontjára, továbbá

$$wx_i - b \leq -1$$

A második osztály pontjaira. Ez röviden a következő módon is írható:

$$y_i(wx_i - b) \geq 1$$

Nyilván, ha $y_i = 1$, akkor az első, ha $y_i = -1$, akkor a második eset teljesül. Tehát a fenti egyenlőtlenségrendszer teljesülése mellett kell minimalizálni $\|w\|$ -t. Mivel a norma nemnegatív, ezért minimalizálhatjuk $\|w\|^2$ -t is. Ez jóval kényelmesebb, mert ekkor kezelhetjük a problémát egy kvadratikus programozási feladatként.

A Lagrange-multiplikátorok módszerével a feladat a következő formába írható:

$$\min_{w,b} \max_{\alpha \geq 0} \left\{ \frac{1}{2} \|w\|^2 - \sum_{i=1}^n \alpha_i [y_i(wx_i - b) - 1] \right\}$$

Ezt már meg tudja oldani a legtöbb kvadratikus programozási módszer.

Most térjünk át arra az esetre, amikor nem lehet a két halmazt elválasztani egy síkkal. Ekkor egy olyan elválasztást szeretnénk, ami minél kevesebb elemet enged át a másik oldalra, és a jó helyen levő elemek távolsága a síktól továbbra is maximális. Ehhez vezessük be a ξ_i változókat. Ezek mérik, mennyire csúsznak át az egyes pontok a másik oldalra.

Pontosabban, a következő egyenlőtlenségnek kell teljesülnie:

$$y_i(wx_i - b) \geq 1 - \xi_i$$

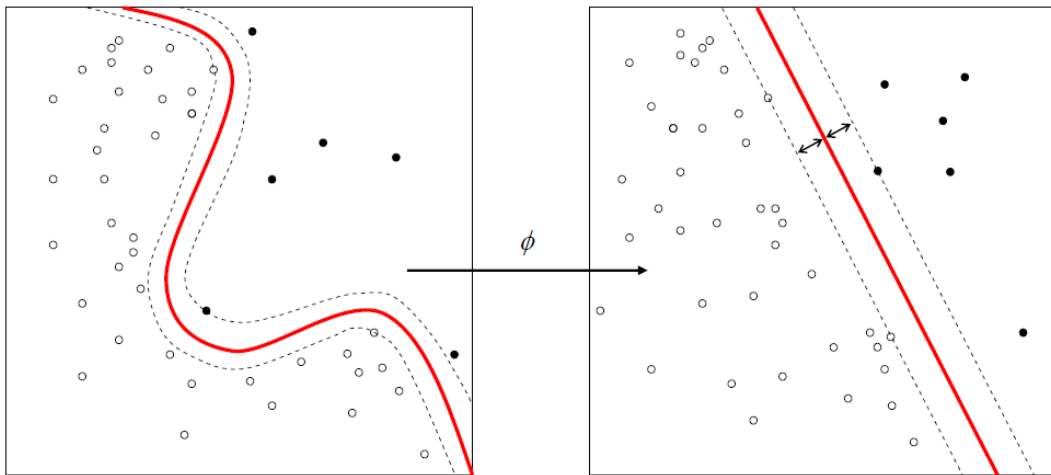
Kell egy költségfüggvény, ami ξ_i számoktól függ. A legegyszerűbb esetben választhatjuk a lineáris költséget, ekkor rögzítünk egy C számot, és a következő kifejezést minimalizáljuk:

$$\min_{w,b,\xi} \left\{ \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i \right\}$$

a fenti feltételek mellett. Azaz a fenti $\|w\|^2$ értékhez még hozzáadjuk az összes hibák C -szeresét. Az előbbiekhez hasonlóan a Lagrange-multiplikátorok módszerét alkalmazva a következő feladatra vezethetjük vissza a problémát:

$$\min_{w,b,\xi} \max_{\alpha \geq 0, \beta \geq 0} \left\{ \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i - \sum_{i=1}^n \alpha_i [y_i(wx_i - b) - 1 + \xi_i] - \sum_{i=1}^n \beta_i \xi_i \right\}$$

A fent vázolt SVM algoritmus csak lineáris elválasztó felületet (egy hipersíkot) keres. Lehetséges azonban, hogy az adatok úgy helyezkednek, hogy azokra nem illeszkedik természetesen egy sík, de egy másik felület már jól elválasztja őket.



7. ábra Kernelfüggvény használata

A 7. ábra mutat egy példát. A bal oldali ábrán a fekete és fehér pontok nem választhatók el szépen egy síkkal, így az eredmény nem lenne kielégítő. Ehelyett transzformáljuk át az eredeti teret egy másikba, ahol már el lehet a pontokat egy síkkal választani.

A trükk a következő: a fenti eljárásban a skaláris szorzást mindenhol egy másik függvényre, az úgynevezett kernelre cseréljük. A kernelfüggvénynek nem kell lineárisnak lennie, így a transzformált

térben levő hipersíkok az eredeti térben nem feltétlen lesznek síkok, hanem bonyolultabb felületek, mint az ábrán látható. Néhány gyakrabban használt kernel:

- polinomiális: $k(x, y) = (xy)^d$
- exponenciális (RBF): $k(x, y) = \exp(-\gamma \|x - y\|^2)$
- szigmoid: $k(x, y) = \tanh(\kappa xy + c)$

A kernelből a φ transzformációt a $k(x, y) = \varphi(x)\varphi(y)$ egyenlet megoldásával kapjuk. A bemutatott filmkritikákat osztályozó algoritmus lineáris kernelt használ, hiszen rengeteg paraméter (az egyes melléknevek) alapján dolgozik az osztályzás. Így a tér, amibe beágyazzuk a tanítóhalmazt sokdimenziós, tehát könnyű találni a pontokat elválasztó hipersíkot.

Az SVM algoritmusnak hátrányai közé sorolhatjuk a következőket: nem ad egy valószínűséget, ami az osztályozás helyességét méri, azaz nincsen számszerű érték arra, hogy "mennyire" tartozik bele egy pont a kiválasztott kategóriába. Az elválasztó síktól való távolságot tekinthetjük egy ilyen értéknek, azonban ez bőven hagy maga után kívánnivalót.

A következő probléma, hogy a betanítás eredményét nehéz értelmezni. A kiválasztott sík, illetve a síkra eső pontok (az úgynevezett support vectorok) nem mutatnak semmi szemléletes jelentést. Továbbá az adatok és az eredmény vizualizálása is nehézkes. A fentebb használt ábrák mind 2-dimenziósak, azonban már 4 dimenzióban is nehézkessé válik a megjelenítés (egy háromdimenziós ábrán a pontok színezeése adhatja a negyedik dimenziót), később meg még átláthatatlanabb lesz. Szövegbányászati algoritmusokban gyakori a többszáz paraméter, ekkor nincs mód egy értelmezhető ábrát adni.

A harmadik, kisebb probléma, hogy az SVM csak két osztályra bont. Ha több osztályunk van, akkor több lépésben tudunk eljutni a végső modellhez. Néhány módszer ennek végrehajtására²³: az első módszerben minden c osztályhoz készítünk egy modellt, ami az adatokat a c és az összes többi osztály között osztja ketté. Egy új adat abba az osztályba kerül, amelyiknél a legnagyobb lesz a margin, azaz az elválasztósíktól való távolság. Egy másik módszernél bármely két osztály-párra elkészítünk egy modellt. Ezután az új pont abba az osztályba kerül, amibe ezek a modellek a legtöbbször rakják. Ez a probléma az általam írt osztályozó algoritmusnál nem merült fel, hiszen csak pozitív és negatív csoportok voltak.

4.2. Paraméteroptimalizálás

Az SVM-ben van néhány paraméter, amik értékeit empirikusan kell meghatároznunk. Ilyen a pontok átlógásának C költsége, illetve az exponenciális kernel γ paramétere. Esetünkben csak az előbbi

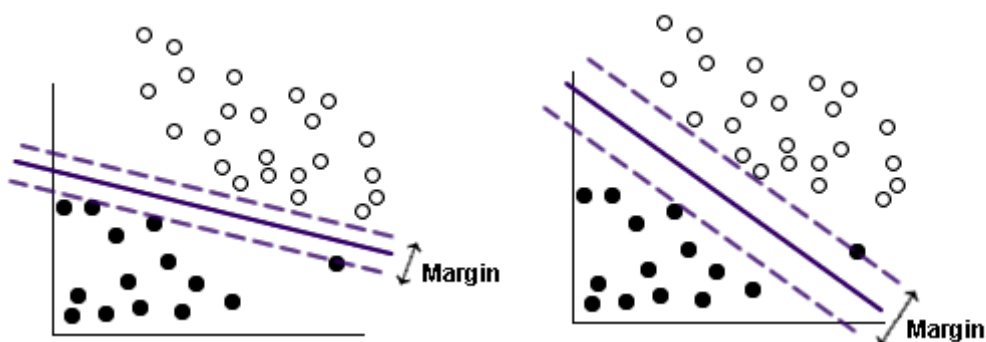
értéket kellett meghatározni, hiszen lineáris kernelt használtam. Az optimalizálási eljárás Hsu, Chang és Lin leírása²⁴ alapján történt, akik a LibSVM nevű SVM osztályozót készítették. Ők általánosságban az RBF kernelt javasolják, azonban külön kiemelik, ha a feature-ök száma nagy, mint szövegbányászat esetében, akkor a lineáris kernel gyorsabb és azonos eredményeket ad. A lépések sorrendje a következő:

1. Adatok normalizálása
2. Cross-validation segítségével megkeresni a legjobb C paramétert
3. Eredmények vizsgálata

Az adatok normalizálásakor minden paramétert azonos nagyságrendűvé kell transzformálni. Ellenkező esetben a nagyobb értékek elnyomhatják a kisebbeket. Továbbá számolási pontatlanságokhoz is vezethet, ha túl nagy a különbség a legkisebb és legnagyobb nagyságrendű adat közt. Ajánlott az adatokat a $[-1;1]$ vagy $[0;1]$ intervallumba képezni, lineárisan. A teszteléskor figyelni kell, hogy ugyanazt a transzformációt alkalmazzuk, mint a modell elkészítésekor.

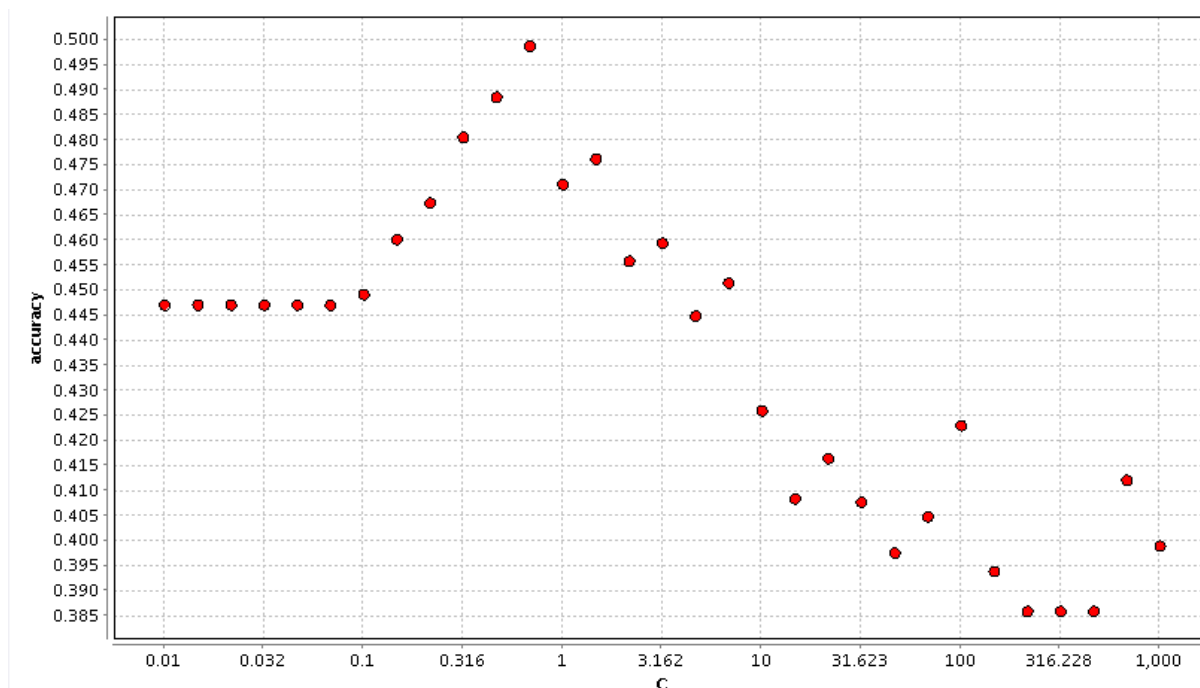
A második lépésben jön a paraméteroptyimalizálás lényegi része. Ahhoz, hogy tesztelni tudjuk az osztályozás hatékonyságát, címkézett adatra van szükségünk. Ez azonban gyakran része a tanítóhalmaznak, annak meg nincsen sok értelme, hogy a tanítóhalmazon teszteljük az eredményt. Ilyenkor két részre osztjuk az adatainkat: az egyiket betanítunk, a másikon tesztelünk. Ennek egy fejlesztése a cross-validation. Ekkor az adatokat n egyenlő részre osztjuk véletlenszerűen. Ezután sorban minden így kapott részhalmazon teszteljük az osztályozást úgy, hogy a tanítóhalmazt a többi $n - 1$ részhalmaz alkotja. A tesztelés során minden adatot pontosan egyszer osztályoztunk, így a tesztelés eredménye a helyesen osztályozott példányok aránya az összeshez képest. Ez az accuracy érték.

A cross-validation megóv minket a túltanítástól. Túltanításról akkor beszélünk, ha az elválasztó hipersík a pontokat jobban választja el (azaz csak kevés pont van a rossz oldalon), viszont az így kapott távolsága a halmazoknak kicsi. Tekintsük az alábbi két felosztást²⁵:



A fenti ábrán például a bal oldali elválasztás tökéletes, minden pont egy oldalon van. A jobb oldali ábrán ez nem teljesül, viszont a síkok távolsága sokkal nagyobb, tehát ebben az értelemben ez egy jobb választás.

Az optimális paraméterek kiválasztásához cross-validationt végzünk, különböző C értékekre. A hivatkozott leírásban exponenciálisan növekvő C értékeket ajánlanak, például $10^{-5}, 10^{-4}, \dots, 10^{10}$. Ezt jó kezdeti közelítésnek találták. Ha megkaptuk ezek közül a legjobbat, akkor ennek egy környezetére szűkíthetjük a keresést. Például ha 10^{-2} -t kaptunk, akkor az $10^{-3}, 10^{-2.5}, \dots, 10^0$ értékeket próbáljuk a következő menetben.



8. ábra Paraméter optimalizálása (logaritmikus skálán ábrázolva)

Ez a megoldás hatékony, hiszen csak egy paramétről van szó. Fejlettebb algoritmusok (például genetikusan algoritmusok) nem fognak gyorsabb eredményt adni, hiszen nagyjából ugyanannyi cross-validationt kell nekik is végezniük. Továbbá az eredményeket grafikonon is ábrázolhatjuk a jobb megértés érdekében. Itt is előny, hogy csak 1 dimenziós függvényt kell ábrázolni. 2 paraméter esetén érdemes a szintvonalakat ábrázolni.

Végül, ha megtaláltuk az optimális C értéket, az összes adaton betanítjuk az algoritmust, és kész a modellünk. Ezt lehet felhasználni új példák osztályozására.

5. Az algoritmus leírása

A használt algoritmus többféle paraméterezést használt. Ezeket és a hozzá tartozó fogalmakat az alábbiakban foglaljuk össze. Előbb azonban bemutatjuk a program működését vázlatosan, a részletekre később térve ki.

Az eljárás először előfeldolgozza az osztályozandó szövegeket. A lépések sorban:

1. POS-taggelés
2. stopwordök kiszűrése
3. szótövezés

A POS-taggelés a Part of speech tagging kifejezés rövidítése. A feladata, hogy a mondatot nyelvtanilag elemezzük, és minden szónak meghatározzuk a mondatbeli szerepét: megkeresi a szófajt, megkülönbözteti az egyes és többes számot, illetve a nemeket. Különböző szófajt meghatározó algoritmusokat már a 60-as évek óta fejlesztenek, és egyre jobb eredményeket értek el. A legtöbb eljárás valamilyen tanuló algoritmust használ, mint például SVM vagy maximum entrópia. A RapidMiner által használt POS-taggelő a Stanfordi egyetemen kifejlesztett csomag. Az általuk használt tanítóhalmaz a Pennsylvanai egyetemen 1993-ban összeállított Penn Treebank nevű adathalmaz. A szófaji meghatározás hatékonysága 97,3%⁵, ez bőven tekinthető elégségesnek az érzelmosztályozás szempontjából.

Az algoritmus a melléneveket gyűjti ki, így számunkra az összes elérhető címke közül a következő három címke a fontos:

- JJ – melléknév
- JJR – középfokú melléknév
- JJS – felsőfokú melléknév

Az összes elérhető címke listája megtalálható a végjegyzetben hivatkozott weboldalon¹⁰.

A következő lépés az előfeldolgozásban a stopword-ök kiszűrése. A stopwordok olyan gyakran előforduló szavak, amik nem hordoznak számunkra fontos jelentést, például kötőszavak vagy névelők. Ezeket a feldolgozás gyorsítása érdekében mindenképpen ki kell törölni. Egy dokumentumon tesztelve például 825 szóból a szűrés után csak 220 maradt. Ennek a lépésnek mindenképpen a POS-taggelés után kell jönnie, hiszen a taggelő eljárások a címkézendő szó szövegkörnyezetét is figyelembe veszik. Ha előbb szűrnénk, akkor címkézés nagyon rossz eredményeket adna.

A harmadik lépés a szavak szótővekre redukálása. Erre azért van szükség, hogy a fokozott mellékneveket ne különböztessük meg az eredeti alaktól. Ha különbözőknek vennénk őket, akkor a ritkán előforduló fokozott alakok elvesznének a többi szóhoz viszonyítva. A szótővezéshez (angolul stemming) használt eljárás a WordNet szótárt veszi alapul. Ez egy olyan angol szótár, ami tartalmazza az egyes szavak szinonimáit, definícióit, illetve egyfajta rendszertanát (például a kutya egy állat, az állat pedig egy élőlény). Az előbbieken kívül a szavak gyökei is benne vannak, ezt használja ki az alkalmazott eljárás.

Most, hogy megtörtént az előfeldolgozás, minden osztályozandó szövegből egy szóhalmazt kaptunk (szigorúan véve nem halmazt, mert egy szó többször is előfordulhat). A következő lépésben egy adattáblát generálunk a halmazunkból. Ennek sorai a szövegeknek, oszlopai a mellékneveknek felelnek meg. Az egyes cellák tartalma a paraméterezéstől függ. Például a legegyszerűbb esetben minden cellába a szavak előfordulási számát írjuk.

Row No.	value	text	13th	19th	20th	able	absolute ▼	abst
191	0	various films	0	0	0	0	2	0
33	1	african detec	0	0	0	0	1	0
61	1	watching ten	0	0	0	1	1	0
1	1	films adapte	0	0	0	0	0	0
2	1	movie come	0	0	0	0	0	0
3	1	've got mail v	0	0	0	0	0	0
4	1	"love" story	0	0	0	0	0	0

9. ábra Az előfordulások számát tartalmazó táblázat

A 9. ábra látható, hogy a 191-es azonosítójú szövegben az "absolute" szó kétszer fordul elő, az able meg egyszer sem.

A tanuló algoritmusunk e táblázat alapján fogja elkészíteni a modellt. Az osztályozás célváltozója a value oszlop. Ennek 1 a tartalma, ha a filmkritika pozitív hangvételi, 0 ha negatív. Többfajta eljárást próbáltam ki: az SVM-et és a naiv Bayes-t. Utóbbival nagyon rossz eredményeket kaptam, általában 60% körüli volt a hatékonyság. Ez messze elmarad az SVM 80% körüli eredményeitől. Részletesebb eredmények az 6. szakaszban olvashatók. Neurális hálókkal is próbáltam az algoritmust futtatni, azonban túl sok feature (azaz az osztályozás alapjául szolgáló paraméter, esetünkben a szavak) van, így az algoritmus nagyon lassú, belátható időn belül nem fejeződött be.

A fenti betanító lépés után eredményül egy modellt kapunk. Ezt a processz FinalModel néven menti el a repositoryba. Amikor új szövegeket akarunk osztályozni, akkor a fenti lépéseket végrehajtjuk és eredményül a fenti táblázathoz hasonló eredményt kapunk. Ezután a kapott modellt futtatva a táblázatra, minden sort (azaz szöveget) behelyez az algoritmus a megfelelő osztályba, és az

eredményt a prediction(value) oszlopba helyezi. Továbbá még egy confidence nevű oszlopot is generál, ami az osztályozás helyességét írja le. Ha ez az érték közel van 1-hez, akkor az osztályozás nagy valószínűséggel helyes volt, ha 0.5-höz, akkor ennek kisebb a valószínűsége. (A confidence lényegében a helyes osztályozás valószínűsége, így értéke mindig legalább 0,5, ellenkező esetben a másik osztályba kerülne az elem).

5.1. A használt paraméterezések

A legegyszerűbb paraméterezés az egyes szavak előfordulási száma (lásd a fenti példát). A RapidMinerben ezt Term Occurrences-nek hívják, így a későbbiekben TO-ként jelöljük. A TO érték egyszerűen egy adott szónak a szövegbeli előfordulási száma. Ezt nem túl gyakran használják, hiszen a különböző hosszúságú dokumentumok esetén a hosszabb szövegekben a melléknevek TO értéke nagyobb lesz, torzítva ezzel az adatokat. A teszteredményekből látszik, hogy a mi esetünkben sem volt hatékony ez a megközelítés.

Ha TO értéket lenormáljuk, akkor kapjuk a term frequency értéket. Ezt formálisan a következő módon írhatjuk le:

$$TF(t, d) = \frac{\text{a } t \text{ előfordulási száma } d \text{ – ben}}{\text{a } d \text{ – ben levő szavak száma}}$$

ahol t egy szó, d pedig egy dokumentum. A TF értéket már több helyen használják, mint a sima TO-t. A saját eredményeim is jelentős javulást mutatnak hozzá képest. Ennek nyilvánvaló oka, hogy az érték normálva van, így a szöveg hossza nincs hatással az eredményekre.

A harmadik paraméter a TF-IDF érték. Az előbbiekben csak a szavak szövegbeli gyakoriságát vettük figyelembe, azt nem, hogy a többi dokumentumban hányszor fordul elő. Ha egy kifejezés majdnem minden szövegben előfordul, akkor kevésbé fontos, mintha csak néhány kiválasztott bejegyzésben. Előbbi esetben tehát kisebb értéket kell neki adni, még utóbbi esetén növelni kell azt. A TF-IDF érték pontosan ezt csinálja: megszorozza a fent már leírt TF számot az IDF értékkel. Az IDF az Inverse Document Frequency rövidítése, és azt írja le, milyen ritkán fordul elő egy szó a szövegben. A formális képlet:

$$IDF(t, d, D) = \log \frac{|D|}{|\{d \in D: t \in d\}|}$$

ahol t egy szó, d egy szöveg, és D a szövegeket tartalmazó korpusz. A számlálóban a dokumentumok száma van. A nevező azon szövegek száma, amik tartalmazzák a t kifejezést. A hányados pont a t szó dokumentumok belső gyakoriságának reciproka. Például, ha egy szó minden dokumentumban szerepel, akkor a nevező értéke is $|D|$, tehát a hányados 1 lesz. Ekkor az IDF érték 0 lesz, vagyis ez a

szó elhanyagolható jelentőségű. Ha a t melléknév csak egyetlen dokumentumban van jelen, akkor a hányados értéke $|D|$. Az IDF érték ekkor $\log|D|$, tehát a szónak jóval nagyobb a jelentősége. Vegyük észre, hogy 0-val osztástól nem kell tartanunk, mert csak olyan mellékneveket vizsgálunk az algoritmus során, amelyek biztos szerepelnek valamelyik szövegben (hiszen onnan nyertük ki őket). Ha ettől tartani kéne, akkor a nevezőhöz hozzáadhatunk 1-et.

Összefoglalva, a TF-IDF érték pontos képlete:

$$TFIDF(t, d, D) = TF \cdot \log \frac{|D|}{|\{d \in D: t \in d\}|}$$

A TF-IDF érték nagyon népszerű, rengeteg helyen használják őt vagy egy változatát a szövegfeldolgozásban. Például a BM25 nevű rangsoroló algoritmus a TF-IDF egy súlyozott módosítását használja. Keresőprogramok és szöveg-összefoglaló programok is hasonló függvényeken alapulnak.

A fenti három paraméterezést egy súlyozás segítségével is módosítottam. Míg a fenti értékek általában bármely szövegbányászati feladathoz használhatók, a bevezetett súlyozás kifejezetten az érzelmosztályozásra gondolva készült. Az eddigi eljárások alapja az volt, hogy összehasonlítjuk a melléknevek eloszlását a pozitív és negatív kritikákban. Azt várjuk például, hogy a good szó inkább a pozitív, a bad pedig a negatív kritikákban fordul elő. Azonban a fejlesztés korai szakaszában azt tapasztaltam, hogy a good megjelenik mind a két típusú hozzászólásban. Ennek számos oka van, például a film helyett az egyik színészre vonatkozott a megjegyzés, vagy csak egyszerűen tagadva volt (not good). Ezt próbáltam meg korrigálni a következő módszerrel.

Először is minden melléknév kap egy w_i súlyt. Az i . melléknévhez tartozó w_i súly képlete a következő:

$$w_i = \frac{p_i - n_i}{p_i + n_i}$$

ahol p_i azoknak a pozitív kommenteknek a száma, amikben előfordul az i . melléknév. Hasonlóan, n_i azon a negatív kommenteknek a száma, amikben előfordul az i . melléknév. Vegyük észre, hogy a nevezőben azoknak a szövegeknek a száma van, amelyekben szerepel a melléknév.

Ha egy melléknév szinte csak pozitív hozzászólásban szerepel, akkor a számláló egy nagy pozitív szám lesz, így a hányados is pozitív értékű. Ha csak negatívban, akkor negatív a számláló és a w_i szám is. Végül, ha nagyjából azonos mennyiségű pozitív és negatív kommentben fordul elő, akkor az érték 0-hoz közelít. Ez tehát jól leírja egy szó jelentéstartalmát. A fenti három számot megszorozzuk a megfelelő súllyal, így újabb három értéket kapunk. A tesztelés során, ezek jobban szerepeltek, mint a

súlyozatlan változatok, tehát elmondhatjuk, van értelme figyelembe venni a melléknevek hozzászólások szerinti eloszlását.

5.2. Betanítás

A betanítás több részlépésből áll. Ennek oka a sebesség optimalizálása. A feldolgozás lépései közül messze a leglassabb a POS-taggelés. 200 komment címkézése körülbelül 20 percet vesz igénybe. Ha minden teszteléskor végre akarnánk ezt hajtani, az rengeteg holtidőt jelentene. Ezért a mellékneveket előre kiválasztjuk és kiszámoljuk a súlyokat. Ezután a különböző paraméterek teszteléseinél már nem kell elvégezni ezt a műveletet, jelentősen gyorsítva a folyamatot.

A processzek hívási sorrendje a következő:

1. LoadComments
2. CollectAdjectives
3. CalculateAdjectiveWeights
4. WeightedAdjectiveModeler

Az egyes processzek feladatait és felépítését az alábbiakban részletezem.

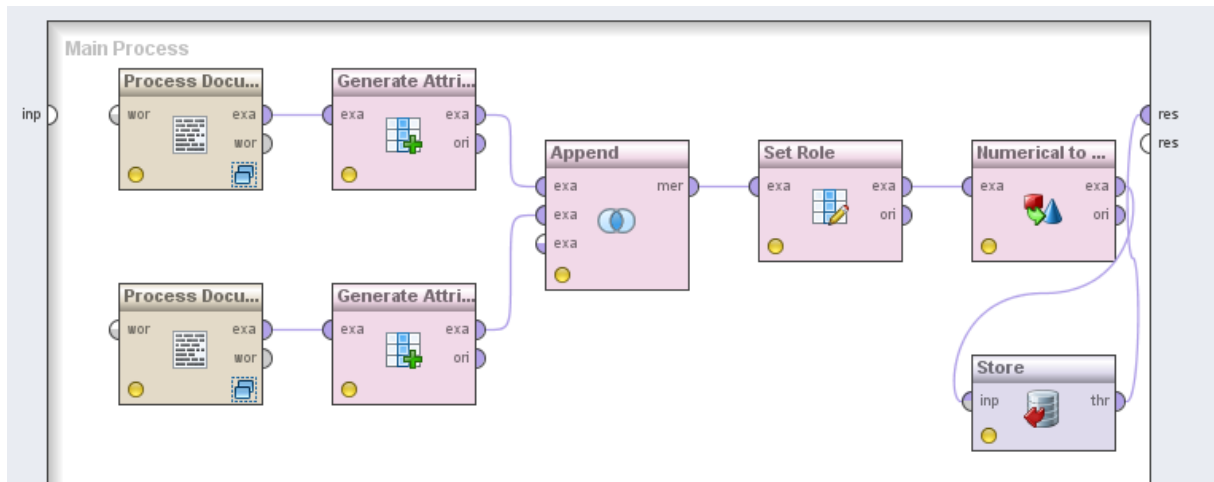
5.2.1. LoadComments

Betölti egy repositoryba (*imdbdata* a neve) a tanítóhalmazt és hozzáad egy *value* oszlopot, aminek 0 az értéke, ha negatív; 1 ha pozitív a komment. A *value* oszlop lesz az osztályozó célváltozója, ennek az értékét próbálja kitalálni az algoritmus.

A tanítóhalmazt két mappára kell osztani: egy pozitív és egy negatív kommenteket tartalmazóra. Az egyes kommenteknek külön txt fájlban kell lenniük. Ennek a követelménynek az az egyszerű oka, hogy a betanításhoz használt adathalmaz ilyen formátumban van.

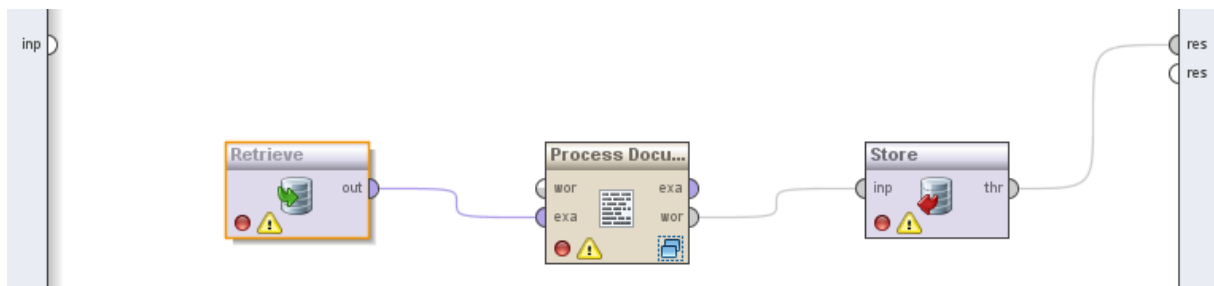
1. **Process Documents** - betölti a szövegeket. A felső ágon a pozitív, az alsón a negatív kommenteket kell betölteni. Ehhez állítsuk be a *text directories* paraméter értékét a megfelelő mappára.
2. **Generate Attributes** – pozitív kommenteknél 1, negatív kommenteknél 0 értékű oszlopot hoz létre (az új oszlop neve *value*)
3. **Append** – összefűzi a két adathalmazt
4. **Set Role** – a *value* oszlop típusát label-re változtatja, ez lesz az osztályozás célváltozója
5. **Numerical to Polynomial** - nominallá konvertálja a *value* oszlop értékét. Erre azért van szükség, mert osztályozni csak nominal típusú oszlopokra lehet.

A kommenteket tartalmazó mappákat a *Process Documents* operátorokban kell beállítani.



10. ábra A LoadComments process

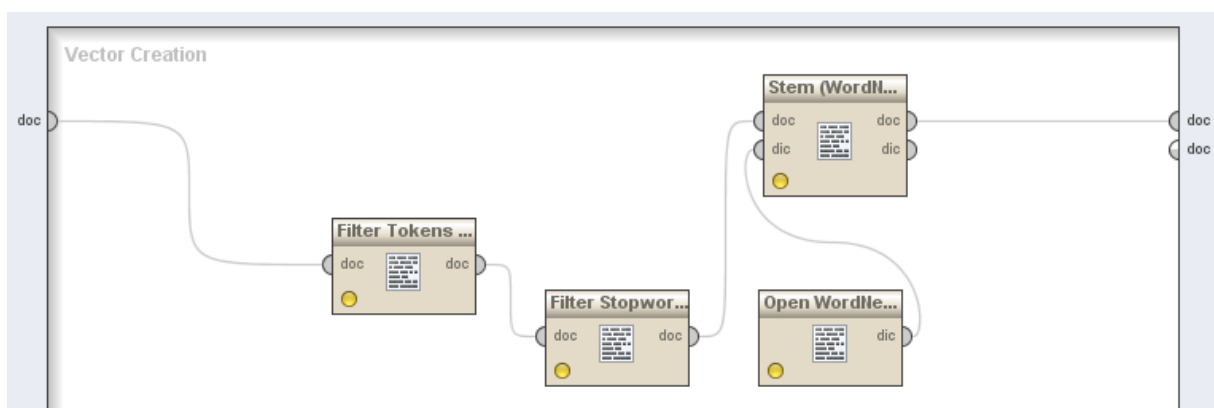
5.2.2. CollectAdjectives



11. ábra A CollectAdjectives processz

A processz kigyűjti az *imdbdata* repositoryban levő szövegekből a mellékneveket, és eltárolja az *Adjectives* nevű repositoryba.

Először betölti a **Retrieve** operátor segítségével a már elmentett adatokat. Ezután a **Process Documents** operátor végzi el a feldolgozás lényegi részét, a részlépések:



12. ábra A Process Documents operátor belseje

1. **Filter Tokens (by POS tags)** – POS taggelés után csak a mellékneveket tartja meg. A keresett minta a következő: $\bar{J} . *$
2. **Filter Stopwords** – a kötőszavakat és egyéb nem fontos szavakat kitörli
3. **Open WordNet** – a WordNet szótár megnyitása, a directory paraméterben meg kell adni a szótár elérési útját
4. **Stem (WordNet)** – WordNet alapú szótővezés

Ezen felül a *Process Documents* operátor csak azokat a szavakat tartja meg, amik legalább 2 dokumentumban előfordultak. Ennek az a célja, hogy az egyszer előforduló szavakat töröljük, mert az osztályozás hatékonyságát nem javítják, viszont a gyorsaságát jelentősen rontják.

Végül az eredményt az *Adjectives* nevű repositoryba teszi az utolsó **Store** operátor.

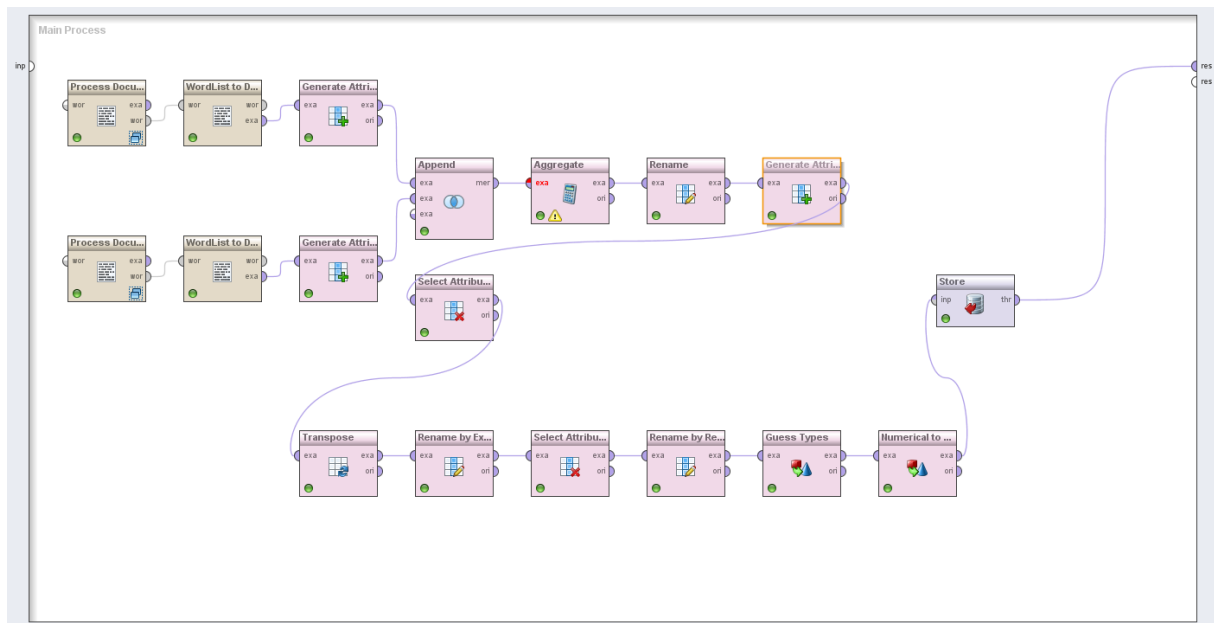
5.2.3. CalculateAdjectiveWeights

Kiszámítja a melléknevek w_i súlyát és eltárolja egy repositoryban. A súly képletét lásd fentebb.

1. **Process Documents from files** – Kigyűjti a pozitív kommentekben levő mellékneveket. Az operátor tartalma ugyanaz, mint amit fent a CollectAdjectives operátornál leírtunk.
2. **WordList to Data** – Átkonvertálja az előző operátor kimenetét. Erre azért van szükség, mert a *WordList* típusú objektumokon nem lehet semmilyen műveletet végezni.
3. **Generate Attributes** – Hozzáadja az előzőekhez a signedcnt attribútumot. Ez a pozitív kommentek esetén egyenlő a total attribútummal, negatív kommentek esetén annak (-1)-szerese.
4. **Append** – Összefűzi a negatív és pozitív adathalmazokat.
5. **Aggregate** – Mivel az összefűzés után minden szó kétszer szerepel (egyszer a pozitív kommentek közül, és egyszer a negatívakból), ezért összeadjuk a két sor értékeit. Ekkor a signedcnt attribútumban lesz a (poz. előfordulások-neg. előfordulások) száma.
6. **Rename** – Ez csak kozmetikai operátor, ugyanis a RapidMiner átnevezi az aggregált attribútumokat, itt visszanevezzük őket.
7. **Filter Examples** – Kitöröljük azokat a sorokat, ahol a signedcnt érték 0. Erre a sebesség növelése miatt van szükség, hiszen az ilyen melléknevek $TF \cdot w$, $TO \cdot w$ és $TF-IDF \cdot w$ értékei mindig nulla lesz, tehát elhagyhatóak.
8. **Generate Attributes** – létrehozza a *weight* oszlopot, ami a signedcnt és a total oszlopok hányadosa. Ez az érték a kiszámítandó súly. A processz további része átalakítja az így kapott eredményt olyan formátumba, ami kényelmesebb lesz az adatok használatakor.
9. **Select Attributes** – kiválasztja a *word* és *weight* attribútumokat. A későbbiekben csak ezekre az attribútumokra lesz szükség.

10. **Transpose** – Transzponáljuk az oszlopvektorunkat sorvektorrá. Most annyi attribútumunk van ahány szavunk volt, továbbá két sorunk van, amik az előző vektor két oszlopának felelnek meg.
11. **Rename by Example Values** – Átnevezi az attribútumokat a szavakra, most már csak egyetlen sorból áll az eredmény, és a fejlécben a szavak szerepelnek.
12. **Select Attributes** – Kitörli a word oszlopot (ez az eredeti oszlopvektor fejlécéből marad itt).
13. **Rename by Replacing** – Kitörli a zárójeleket az attribútumnevekből és minden szó elé egy # karaktert tesz. Erre az összeszorzás miatt van szükség, részleteket lásd a modell kiszámításánál.
14. **Guess Types** – Numerikussá konvertálja az összes oszlopot. Ez azért szükséges, mert transzponálás után az attribútumok értéke *nominal* lesz, viszont nekünk valósakra van szükség
15. **Numerical to Real** – Valóssá konvertálja az oszlopokat.
16. **Store** - Eltárolja az eredményt a *WeightedAdjectives* nevű repositoryban.

A kommenteket tartalmazó mappákat a *Process Documents* operátorokban kell beállítani.



13. ábra A CalculateAdjectiveWeights processz

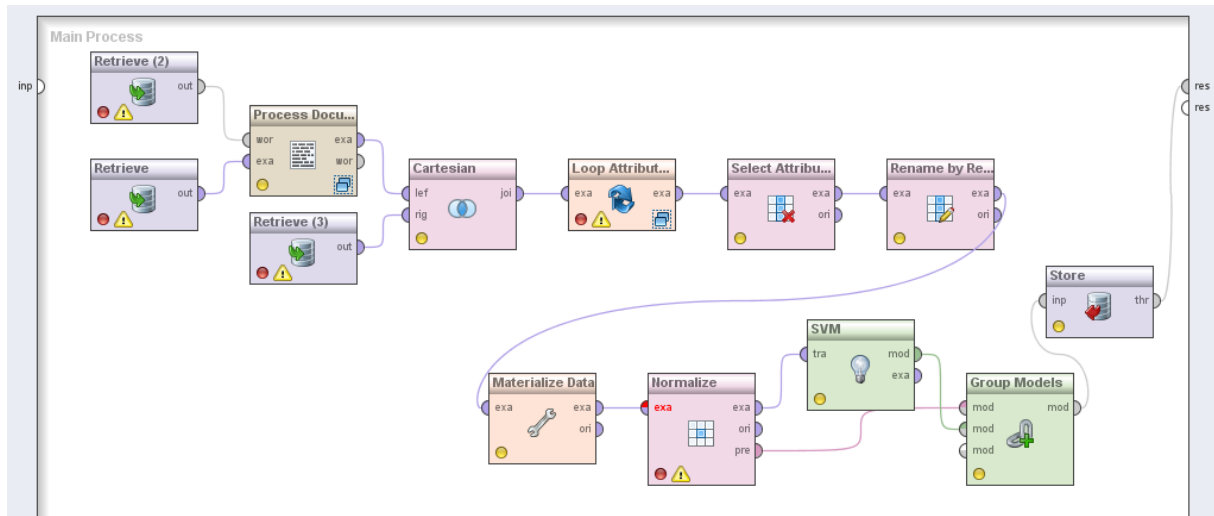
5.2.4. WeightedAdjectiveModeler

Elkészíti a végleges modellt, amit később lehet használni az osztályozáshoz.

1. **Process Documents from Data** – Kiszámolja a melléknevek Term Frequency-jét. Két bemenő adata van: a már előre kigyűjtött melléknevek és az *imdbdata* repositoryból betöltött szövegek. Mind a kettőt egy Retrieve operátor segítségével kapja meg. A processzen belül csak sima tokenizálás és szótövezés van. Mellékneveket már nem kell keresni, hiszen azokat megkapja bemenő paraméterként. Ez a teszteléskor kényelmes, mert megspóroljuk a POS tagginget, amivel rengeteg időt lehet nyerni.
2. **Cartesian** - Direktszorozzuk a Term Frequencyket tartalmazó táblázatot és az egyes melléknevek súlyát tartalmazó sorvektort. Itt szükség van arra, hogy ne legyen két azonos attribútumnév, ezért kellett a # előtag a súlyok kiszámolásakor.
3. **Loop Attributes** – itt számoljuk ki a $tf_i \cdot w_i$ értékeket az egyes melléknevekhez. A ciklus végigmegy az összes olyan attribútumon, ami nem #-tel kezdődik. Ha egy ilyen attribútumnak van #-tel kezdődő párja, akkor azokat összeszorozza. Az összeszorozás elég macerás, a problémák a következők: A *Generate Attributes* elfogad makrókat, de ha egy szóban speciális karakter van (ködőjel, szám) akkor hibát jelez. A *Generate Products* nem fogad el makrókat, de elfogad speciális karaktert tartalmazó attribútumokat. A megoldás, hogy átnevezzük az összeszorozandó oszlopokat, összeszorozzuk őket, majd visszanevezzük őket. A szorzások eredménye egy @ prefixet kap a könnyű kezelhetőség miatt.
4. **Select Attributes** – Csak a szorzások eredményét tartja meg, a többi attribútumot eldobja. Az eredményoszlopokat a fent hozzáadott @ karakter segítségével azonosítja.
5. **Rename by Replace** – Kitörli a @ karaktert az attribútumnevek elejéről.
6. **Materialize Data** – A létrejött táblát újra létrehozza, ezzel megszabadul a tábla feldolgozása közben létrejött egyéb adatoktól, így memóriát szabadít fel. Továbbá a modell betanítása is gyorsabb lesz, ha az adattábla ki van tisztítva.
7. **Normalize** – A [0; 1] intervallumba transzformálja az összes attribútum értékét. Ez jelentősen javítja az SVM osztályozás hatékonyságát.
8. **SVM** – Kiszámolja a modellt. A paraméterek megtalálásáról lásd az SVM osztályozásról szóló fejezetet.
9. **Group Models** – Összefűzi a normalizálást és az SVM modellezés eredményét egy közös modellbe. Ez egyrészt kényelmi célokat szolgál, hogy ne kelljen egy újabb előfeldolgozó lépést beiktatni a kiértékelésbe. Másrészt biztosítja, hogy ne felejtjük el futtatni a

normalizálást. Ha mégis kihagynánk, akkor az osztályozás természetesen nagyon rossz eredményeket adna.

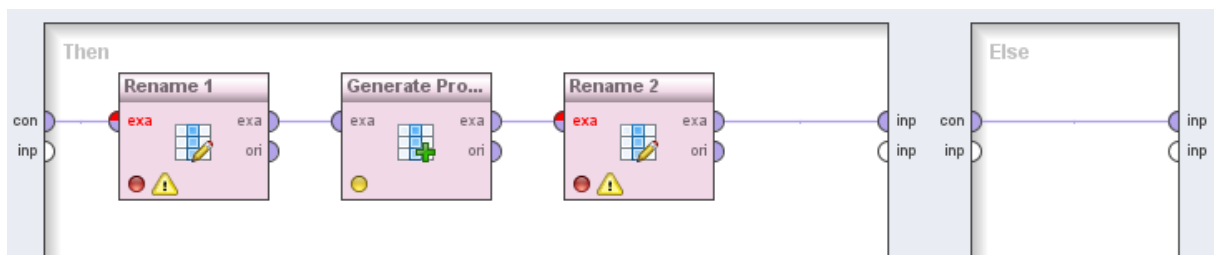
10. **Store** – Ementi a modellt *FinalModel* néven.



14. ábra A Weighted Adjective Modeler processz

A Loop Attributes paraméter belsejében van egy Branch operátor. A Branch operátor az elágazásnak felel meg. Esetünkben eldönti, hogy az éppen vizsgált oszlopnak van-e olyan párja, ami azonos nevű, de #-tel kezdődik. Emlékezzünk vissza, hogy #-tel kezdődtek a súlyozott mellékneveket tartalmazó táblázatban az attribútumok, és ezt Descartes-szoroztuk össze a szövegeket tartalmazó táblázattal. Például az *absurd* oszlop tartalmát az *#absurd* oszlopéval kell összeszorozni. Ha nincs #-tel kezdődő pár, az azt jelenti, hogy annak a melléknévnek a súlya 0 lett volna, ezért nem szerepel a kiválogatott melléknévek közt.

Ezután kell összeszorozni az egyes oszlopokat.



15. ábra A Branch operátor belseje

Az egyes operátorok feladata a következő:

1. **Rename 1** – átnevezi az aktuális oszlopot, amit a loop_attribute makró jelöl xxxloop_attributexx nevűre, a #-tel kezdődő változatát pedig yyyloop_attributeyyy-ra.
2. **Generate Product** – összeszorozza a fenti két oszlopot

3. **Rename 2** – visszanevezi az xxxloop_attributexxx és yyyloop_attributeyyy nevű változókat.

A fenti átnevezésre azért van szükség, mert a Generate Products operátor nem támogatja a makrókat.

5.2.5. Az optimális paraméterek

A RapidMinerben lehetőség van a paraméterek dinamikus változtatására az *Optimize Parameters* operátor segítségével. Ez többször lefuttatja a neki megadott gyerekfolyamatot különböző paraméterekkel, amelyet mi állíthatunk be. Lehetőség van még evolúciós algoritmus használatára az *Optimize Parameters (Evolutionary)* operátor segítségével, azonban ez nem ad jobb eredményeket az előző operátornál, mivel csak egy paramétert kell optimalizálni.

Az *Optimize Parameters* visszaadja a legnagyobb pontosságú beállítást, illetve grafikonon is tudjuk ábrázolni az eredményeket. Itt többfajta grafikont lehet rajzolni, így több paraméter esetén is könnyen értelmezhető eredményeket tud adni, például szintvonalak használatával.

5.3. Használat

Az osztályozóhoz készült egy Java interfész is, ami egy adatbázisból tölti be a hozzászólásokat, majd az eredményeket visszaírja oda. Ez a Java program a WeightedAdjectiveEval processzt futtatja. Ennek a processznek paraméterül kell adni egy SQL lekérdezést, ami az osztályozandó szövegeket adja vissza. Az eredményeket SQL update utasítások formájában egy szövegfájlba írja. Ezeket az utasításokat a Java program futtatja.

Sajnos a RapidMiner nem biztosít megfelelő operátorokat tetszőleges SQL update utasítások végrehajtására. Lehetőség van egy táblázatot egészben feltölteni új táblaként, vagy insert utasítások formájában hozzáfűzni egy meglevő táblához. Azonban adatokat frissíteni nem lehet. Ezért született a fenti öszvér megoldás, hogy Java-n keresztül történik a frissítés.

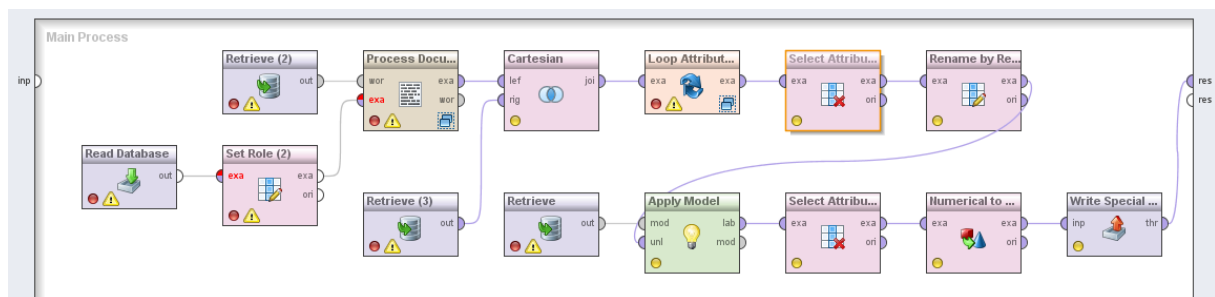
5.3.1. WeightedAdjectiveEval

Letölti és osztályozza a megadott szövegeket. A processz a Java-s programba való integráláshoz van tervezve. A Processz működése:

1. **Read Database** – Letölti az adatokat az adatbázisból. A kapcsolatot az adatbázishoz előre be kell állítani a RapidMinerben. A lekérdezést az *sql_query* makróban lehet átadni. A lekérdezés eredményében szerepelnie kell az ID, FORUMID, TOPICID és COMMENTENG mezőknek. Az adatbázisban ez a három oszlop adja az elsődleges kulcsot.
2. **Set Role** – Az ID, FORUMID és TOPICID attribútumok szerepét állítja át. Csak egy *id* típusú mező lehet egy táblában, ezért saját típusokat kell használni.

3. **Feldolgozás** – Kiszámolja minden kommenthez és melléknévhez a $tf \cdot w$ értéket. A működése megegyezik a *WeightedAdjectiveModeler* processznél leírtakkal.
4. **ApplyModel** – Osztályozza az adatokat a kiszámolt modellel.
5. **Select Attributes** – Kiválasztja a fenti azonosítókat tartalmazó oszlopokat, illetve az osztályozás eredményét, a többi oszlopot törli.
6. **Numerical to Polynomial** – Nominallá alakít mindent. Erre azért van szükség, hogy kiíráskor ne használjon tizedespontot, ami gondot okozhat az SQL utasításokban.
7. **Write Special Format** – Minden komment osztályozását egy SQL update utasításként írja ki. A kimeneti fájl elérési útját az *output_file* makróban kell megadni. Ezek után csak végre kell hajtani az eredmény SQL fájlát. Az alábbi példához hasonló update utasításokat generál:

```
update comments set rate=-1 where id=45 and forumid=1 and topicid=1
```



16. ábra A *WeightedAdjectiveEval* processz

A processz inputja az *sql_query* makró. Ennek kell értékül adni a megfelelő lekérdezést. A lekérdezésnek tartalmaznia kell az ID, FORUMID, TOPICID és COMMENTENG mezőket. A lekérdezés által visszaadott kommenteket fogja osztályozni a program és visszatölteni az azonosítóknak megfelelő rekordba.

Az eredmények az *output_file* makróban megadott fájlba kerülnek. Ebben soronként egy SQL update utasítás van, amiket végrehajtva bekerül az adatbázisba a megfelelő osztályozás.

5.4. Futási idő

A program futási ideje különbözik RapidMinerben futtatva illetve Java-ból hívva. Előbbi esetén 500 (rövid) kommentet 2 perc 30 másodperc alatt dolgoz fel, Java-ban 2 perc alatt. Ha csak 20 kommentet osztályozunk, akkor előbbinek 16, utóbbinak 36 másodpercre van szüksége. Ennek oka, hogy Java-ból futtatva mindig szükség van egy 30 másodperces inicializáló szakaszra. A programot egy Intel 2 magos, 2.1 GHz-es személyi számítógépen futtattam, 2 GB memóriával.

A legtöbb időt a POS-taggelés igényelte, azonban ezt elég egyszer futtatni, így ezt figyelmen kívül lehet hagyni. A RapidMinerben a w súllyal való szorzás sebessége elsősorban az oszlopok számától függ, és kevésbé az adatoktól. Ennek az ideje 800 attribútum esetén akár 1 perc is lehet. Végül az SVM osztályozás a leggyorsabb, egy osztályozás nem igényel többet 10 másodpercnél. A paraméteroptimalizáláskor 5-szörös cross-validation mellett 40 helyen számolja ki az osztályozás pontosságát az Optimize Parameter operátor. Ez tehát összesen 200-szor hívja meg az SVM algoritmust, így az itt eltöltött idő lesz összességében a legnagyobb.

5.5. Problémák a processzek írásakor

A RapidMinert elsősorban gyors prototípusírára tervezték. Ennek megfelelően sok előre implementált algoritmussal rendelkezik. Ezeknek az algoritmusoknak könnyen meghívhatónak kell lenniük az egyszerű cserélhetőség érdekében. A RapidMiner ezért választotta a korábban bemutatott adatfolyamokra épített modellt.

Ezzel kényelmes meghívni a már implementált programrészleteket. Azonban, ha olyan funkciót szeretnénk elérni, ami nem áll rendelkezésünkre, akkor csak igen körülményesen haladhatunk előre. Esetünkben ilyen volt a melléknevek súlyozása. A TO, TF és TF-IDF értékek kiszámítása már megtalálható a programban, így ezzel nem volt gond. A súlyok kiszámításához is rendelkezésre álltak megfelelő operátorok.

A nehézséget a súlyok és TF-IDF értékek összeszorozása jelentette. A RapidMinerben kétféle szorzás operátor kapott helyet: az egyik két előre adott oszlopot szoroz össze. A másiknak meg kell adni az oszlopok két részhalmazát. Az operátor ezek után a két halmazból bármely két elem szorzatát hozzáadja a táblázathoz. Látható, hogy egyik se felel meg a céljainknak: nekünk adott oszlopokat adott értékekkel kell megszorozni. Szerencsére van ciklus operátor, ami a tábla oszlopain megy végig. Ez a helyes irány, azonban újabb problémák adódnak.

A ciklusban az aktuális feldolgozott oszlop címére a `loop_attribute` makróval lehet hivatkozni. Egy makró értékét a `%` jellel lehet lekérdezni, a következő módon: `%{loop_attribute}`. A fenti két szorzás operátor azonban nem támogatja teljesen a makrókat. Az egyikük támogatja a makrókat, de nem lehet speciális karakter az oszlopnévben. Például a `'` kötőjelt kivonásjelnek érzékeli, egy számot tartalmazó oszlopnévet meg magának a számnak. Mivel az oszlopok nevei maguk a szavak, és egyes szavak tartalmazhatnak nem betű karaktert, ez az út járhatatlan.

A másik operátor pedig nem támogatja a makrókat, viszont bármilyen oszlopnévet elfogad. A megoldás a következő: az éppen aktuális oszlopot át kell nevezni egy fix névre, esetünkben ez

xxxloop_attributexxx lett. Ezután ezt az oszlopot kell megszorozni a megfelelő értékkel, majd visszanevezni az oszlopot az eredeti nevére. Sajnos ezt a hack-et nem lehet megkerülni.

Bonyolultabb műveletekhez lehetőség van saját operátorok írására. Ezt Java nyelven lehet megtenni. Az általunk írt operátort integrálhatjuk a RapidMiner rendszerébe. Ezt a lehetőséget még nem próbáltam ki, elsősorban azért, mert a külső operátor írásának dokumentációja fizetős.

Gondot okozott még az adattípusok közti konvertálás. A nominallá konvertálás jól működött, azonban a fordított irányú, nominalról számmá konvertálás nem. Ez azért volt, mert a Parse Number nevű operátort nem erre a célra tervezték. A konvertálást a Guess Types operátor biztosítja, azonban ez hiányosan szerepel a dokumentációban.

6. Eredmények

A RapidMiner segítségével gyorsan lehet különböző algoritmusokat kipróbálni és felparaméterezni. A gépi tanuló algoritmusok ki- és bemenete mind nagyon hasonló, így csak kis munka őket kicserélni. Ezért több ilyen algoritmust is kipróbáltam: a naiv Bayest, a neurális hálókat és az SVM-t. Ezek közül a naiv Bayes nagyon rossz eredményeket adott, a tesztelt pontosság 56%-os volt. A későbbiekben ezért nem is foglalkoztam vele. A neurális hálók rosszul működnek nagy paraméter tér esetén. Ezt én is tapasztaltam, a neurális hálót használó algoritmus nem tudta feldolgozni a teljes tanítóhalmazt belátható időn belül. Az SVM-t azonban kifejezetten ilyen problémákra tervezték. Lineáris kernel esetén a paramétertér akár több tízezer dimenziós is lehet, az algoritmus így is lefut pár percen belül egy asztali számítógépen²⁴. Esetünkben 836 kiválasztott melléknév volt, tehát ennyi dimenziós volt a paramétertér. A továbbiakban csak az SVM alapú osztályzás eredményeiről lesz szó.

A tanítóhalmaz a már említett IMDb-s adathalmaz volt^{11,12}. Ez 1000 pozitív és 1000 negatív filmkritikát tartalmaz, amit az imdb.com nevű filmekkel foglalkozó oldalról töltöttek le. Az egyes kritikák általában 4-500 szót tartalmaznak, ezzel közepesen hosszúnak számítanak. Egy szerzőnek legfeljebb 20 cikke szerepel, hogy elkerüljék azt, hogy néhány kiválasztott ember dominálja a teljes halmazt. Annak ellenére, hogy magán az imdb.com oldalon csillagokkal is ellátják az egyes kritikákat, a tanítóhalmazban csak annyi szerepelt, hogy az adott kritika pozitív vagy negatív-e.

Az érzelemosztályozót többfajta feature-rel is betanítottam, ezek jelentését lásd fentebb. A különböző paraméterezések hatékonyságát 5-szörös cross-validationnel teszteltem. A kapott accuracy értéket a következő formula adja:

$$\text{Accuracy} = \frac{\text{helyesen osztályzott kommentek száma}}{\text{összes komment száma}}$$

Az SVM osztályozó C paraméterének beállítását a fent leírt eljárással végeztem. Mindenhol csak az optimális C -hez tartozó pontosság szerepel.

Az eredményeket az alábbi táblázat foglalja össze. A w jelöli a saját súlyozással kapott eredményeket.

Feature	Accuracy
tf-idf	75,6%
tf	76,7%
to	72,1%
tf-idf * w	81,5%
tf * w	83,6%
to * w	75,6%

A TO paraméterezés bizonyult mind közül a legrosszabbnak. 76,6%-os és 72,1%-os eredményeket adott súlyozott és anélküli felállásban. Ez jóval rosszabb az összes többi értéknél. Ez köszönhető annak például, hogy nem veszi figyelembe a szöveg hosszát, így az osztályozás szükségszerűen torzul.

A TF-IDF volt a második legjobb. Annak ellenére, hogy az irodalomban a TF értéknél gyakrabban használják, itt mégis utóbbi volt az eredményesebb. Ez valószínűleg annak is köszönhető, hogy a stopword-öket kiszűrtem, ezért minden szó gyakorisága nagyjából ugyanaz volt, így az IDF értékek csak kis szerepe volt.

A legjobban a Term Frequency paraméterezés szerepelt. A 76,7% és 83,6%-os eredményei jobbak mind az egyszerű TO és a fejlettebb TF-IDF paraméterezésnél tapasztaltaknál. A 83,6% már elfogadhatónak számít, 80% feletti értékével.

Az adatokból az is látszik, hogy a w értékkel való szorzás minden esetben javított az osztályozás hatékonyságán. Van tehát értelme annak, hogy az egyes mellékneveket súlyozzuk jelentéstartalmuk szerint. További hasonló súlyozási módszereket is ki lehet próbálni, amik figyelembe veszik a melléknév megjelenését a pozitív és negatív kommentekben.

Az osztályozót kipróbáltam más témakörből vett adatokon is. Egy bankokról szóló fórum hozzászólásain futtattam a filmkritikákon betanított modellt. Az eredmény lényegesen rosszabb lett a fent leírtaknál. A kapott pontosság 61% volt, ami messze elmarad a korábban tapasztaltaknál. Ennek több oka is van. Egyrészt a másik témakör miatt több olyan szó is volt, amit a filmekben betanított modell nem ismert. Ezeket semlegesként kezelte az osztályozó, kihagyva számos lehetőséget, ami a helyes osztályozáshoz segítette volna. Másodszor, a fórumokon a kommentek szerkezete alapvetően eltért a kritikákétól. Előbbiek nem túl hosszúak, körülbelül 3-4 sorosak voltak legfeljebb. Legtöbbször a bankkal kapcsolatos problémák vannak bennük, a pozitív hozzászólások száma jóval kevesebb, mint

a negatívoké. A kritikák azonban hosszabbak, általában 30-40 sorosak, egy rövid esszének felelnek meg. A nyelvezetük fejlettebb, jóval több melléknév található bennük, amiken az osztályozás is alapult. Előfordult, hogy az egyik fórumbejegyzés nem is tartalmazott a modell által ismert melléknevet.

7. Összefoglalás

A dolgozatban bemutatam az érzelemosztályozás feladatát: eldönteni egy forrásszövegről, mennyire pozitív vagy negatív a benne megjelenő vélemény. Vázoltam az alkalmazási területeit és az egyéb felvetődő problémákat, amik az általános esetben megjelennek. Ilyen volt például a semleges szövegek kiválasztása, vagy az értékelt tárgy különböző tulajdonságainak részletes vizsgálata. Ezek az általam megoldott feladat esetében nem merültek fel.

Leírtam kétfajta megközelítést a problémának, az egyik felügyelt, a másik felügyelet nélküli tanítást használ. Az utóbbinál olyan problémák jelennek meg, amelyek nehezen áthidalhatóak: egy szónak különböző jelentései vannak különböző szöveggörnyezetben vagy egy negatív hangulatú melléknév előfordulhat egy dicsérő filmkritikában. Ezeket kikerülhetjük a tanítóhalmazt használó eljárásokkal. Sokfajta paraméterezéssel és ismert osztályozó módszerrel végeztek tesztek. A legjobb eredményt a unigram modell és az SVM algoritmus adta, 82,9%-kal.

A további részben bemutatam a RapidMiner adatbányász szoftvert. Ismertettem az adattároló rendszerét, illetve az általa használt alapfogalmakat, mint például role vagy label. Leírtam, hogy épülnek fel az egyes processzek: az alapegységek az operátorok, őket az adatáramlást mutató kábelek kötik össze. Az inputot és outputot az operátorok dobozán levő portok jelképezik.

Részleteztem az SVM algoritmus működését: a formális definíciót és azt, hogyan lehet visszavezetni a kezdeti problémát egy kvadratikus programozási feladatra. Ennek megoldására léteznek gyors és hatékony eljárások, és a terület matematikája is ismert. Az SVM szabad paramétereinek beállítására bemutatam egy rövid módszert, amivel hatékonyan lehet az eredményeket optimalizálni.

Az általam írt algoritmus többfajta paraméterezést próbált ki, például TF és TF-IDF. Egy, a mellékneveket az érzelmi töltetük alapján súlyozó paraméterezést is teszteltem. A RapidMinernek köszönhetően a váltás a különböző modellek közt gyors volt. Az algoritmus több lépésből állt: tokenizálás, melléknevek kiválasztása, stopwordok szűrése és szótövezés. A melléknevek megkeresése a leglassabb művelet, ezért a betanító eljárást több részletre bontottam, hogy ezt csak egyszer kelljen elvégezni. Készítettem egy további processzt, ami kényelmesen meghívható Java-ból, az adatokat egy adatbázisból tölti le és oda írja vissza.

A hatékonyságot a helyesen osztályozott szövegek arányán mértem. A tesztelés eredményei azt mutatták, hogy a melléknevek súlyozása mind a három (TO, TF, TF-IDF) esetben jobb pontosságot adott. Ebből arra következtethetünk, hogy az egyes szavak eloszlásán túl azoknak egy becsült érzelmi töltetét is érdemes figyelembe venni. A legjobb eredményt a súlyozott Term frequency értékek adták, 83,6%-kal.

További fejlesztési lehetőség más súly-képletek kipróbálása, például a súly négyzetre emelésével vagy a melléknevek tagadásának figyelembe vételével. Érdemes lehet egy, a témához kapcsolódó szótár összeállítása és ennek alkalmazása. Egy másik irány, hogy a pozitív-negatív osztályozáson túl a komment csillagozását becsüljük meg. Ez egy másik tanítóhalmazt igényel, hiszen a jelenleg használt nem tartalmazza a kommenthez tartozó csillagok számát.

8. Irodalomjegyzék

- ¹ B. Liu. Sentiment Analysis and Opinion Mining. AAAI, 2011.
- ² A Wright. Mining the Web for Feelings, Not Facts. NY Times, <https://www.nytimes.com/2009/08/24/technology/internet/24emotion.html>, 2012.
- ³ A felmérések szerint aggasztóan gyenge a fogyasztói tudatosság és kompetencia. EU sajtóközlemény. http://ec.europa.eu/magyarország/press_room/press_releases/20110412_a_fogyasztoi_tudatossag_es_kompetencia_hu.htm, 2012.
- ⁴ The Election Will Be Tweeted (and Retweeted). <https://www.nytimes.com/interactive/us/politics/2010-twitter-candidates.html>, 2012.
- ⁵ Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. HLT-NAACL 2003.
- ⁶ B. Pang, L. Lee. Subjectivity Detection and Opinion Identification. Opinion Mining and Sentiment Analysis, Now Publishers Inc, 2008.
- ⁷ H. Yu, V. Hatzivassiloglou. Towards answering opinion questions: Separating facts from opinions and identifying the polarity of opinion sentences. In Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP-2003), 2003.
- ⁸ M. Hu, B. Liu. Mining and summarizing customer reviews. In Proceedings of ACM SIGKDD, International Conference on Knowledge Discovery and Data Mining (KDD-2004), 2004.
- ⁹ P. Turney. Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews. In Proceedings of Annual Meeting of the Association for Computational Linguistics (ACL-2002), 2002.
- ¹⁰ Alphabetical list of part-of-speech tags used in the Penn Treebank Project. http://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html, 2012.
- ¹¹ Movie Review Data. <https://www.cs.cornell.edu/people/pabo/movie-review-data/>, 2012.
- ¹² B. Pang, L. Lee, S. Vaithyanathan. Thumbs up? Sentiment classification using machine learning techniques. In Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP-2002), 2002.
- ¹³ B. Pang, L. Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In Proceedings of Meeting of the Association for Computational Linguistics (ACL-2005), 2005.
- ¹⁴ D. Bollegala, D. Weir, J. Carroll. Using multiple sources to construct a sentiment sensitive thesaurus for cross-domain sentiment classification. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL-2011), 2011.
- ¹⁵ B. Wei, C. Pal. Cross lingual adaptation: an experiment on sentiment classifications. In Proceedings of the ACL 2010 Conference Short Papers (ACL-2010), 2010.
- ¹⁶ J. Boyd-Graber, P. Resnik. Holistic sentiment analysis across languages: multilingual supervised latent Dirichlet allocation. In Proceedings of EMNLP-2010, 2010.
- ¹⁷ Data Mining / Analytic Tools Used Poll. <http://www.kdnuggets.com/polls/2010/data-mining-analytics-tools.html>, 2012.
- ¹⁸ GNU Affero General Public License. <https://www.gnu.org/licenses/agpl-3.0.html>, 2012.
- ¹⁹ M. Ingo, M. Wurst, R. Klinkenberg, M. Scholz, T. Euler. YALE: Rapid Prototyping for Complex Data Mining Tasks. In Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-06), 2006.
- ²⁰ C. Cortes, V. Vapnik, N. Vladimir. Support-Vector Networks. Machine Learning, 20, 1995.
- ²¹ Svm separating hyperplanes. https://en.wikipedia.org/wiki/File:Svm_separating_hyperplanes.png, 2012.
- ²² Support vector machine. https://en.wikipedia.org/wiki/Support_vector_machine, 2012.
- ²³ K. Duan, and S. S. Keerthi. Which Is the Best Multiclass SVM Method? An Empirical Study. Proceedings of the Sixth International Workshop on Multiple Classifier Systems, 2005.
- ²⁴ C. W. Hsu, C. C. Chang, C. J. Lin. A Practical Guide to Support Vector Classification. <https://www.cs.sfu.ca/people/Faculty/teaching/726/spring11/svmguide.pdf>, 2012.
- ²⁵ Ábrák forrása: IBM SPSS dokumentáció. http://pic.dhe.ibm.com/infocenter/spssmodl/v15r0m0/index.jsp?topic=%2Fcom.ibm.spss.modeler.help%2Fsvf_perfimp.htm, 2012.