TANSZÉKVEZETŐ

SZAKDOLGOZAT FELADAT

Kacsó Zoltán
Mérnökinformatikus hallgató részére

Természetes nyelvű szövegek kategorizálása adatbányászati eszközökkel

A természetes nyelvek kategorizálása napjainkban széles körben felhasznált technológia. Leggyakoribb felhasználása a szövegek témakörének megismerése, viszont ezen adatbányászati eszközök felhasználhatóak az adatok más szempontbeli csoportosítására is. Esetünkben a szöveg írójának kiderítésére szeretnénk alkalmazni. Komoly feladatot jelent a felhasznált algoritmusoknak a szövegeket olyan formában ábrázolása, ami alapján az író szóhasználata hangsúlyozódik ki.

A hallgató feladata különböző adatbányászati algoritmusok keresése és megvizsgálása, valamint kiválasztani, a megvizsgált algoritmusok közül melyek alkalmazhatóak jelen feladatához. A kiválasztott algoritmusokhoz az adatok elő feldolgozása, az algoritmus belső paramétereinek (ha létezik) konfigurálása minél jobb pontosság elérésének érdekében szintén a hallgató feladata. Végül a kapott eredmények statisztikai elemzése, az algoritmusok összehasonlítása és konklúziók levonása a cél.

A hallgató feladatának a következőkre kell kiterjednie:

- Ismertesse a kategorizálás alapeszközeit.
- Mutassa be a természetes nyelvű szövegek kategorizálásának nehézségeit.
- Mutassa be a felhasznált algoritmusok működését.
- Készítsen implementációt a választott algoritmusokhoz Java nyelven.
- Elemezze a kategorizálás eredményeit. Hasonlítsa össze a különböző algoritmusok pontosságát.

Tanszéki konzulens: Dr. Dudás Ákos

Budapest, 2016. szeptember 19.

Dr. Charaf Hassan
egyetemi tanár
tanszékvezető

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Kacsó Zoltán

TERMÉSZETES NYELVŰ SZÖVEGEK KATEGORIZÁLÁSA

KONZULENS

Dr. Dudás Ákos

BUDAPEST, 2016

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Kacsó Zoltán**, 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 hitelesített 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, 2016. 12. 08.

...…….

Kacsó Zoltán

Összefoglaló

Ezen dokumentum során a dokumentumok írójuk szerinti osztályozásával fogok foglalkozni.

Először az alap adatbányászati eszközöket ismertetem a csoportosításból kiindulva. Bemutatom a valószínűség alapú csoportosítás leggyakoribb modelljét a topic-model-t, majd ennek egy megvalósítását ismertetem a Latent Dirichlet Allocation algoritmust.

Ezek után ismertetem az osztályozást, mint adatbányászati eszközt. Bemutatom az egyik gyakran felhasznált osztályozó algoritmust, a Naiv Bayes algoritmust. Ismertetem ezen algoritmus modelljének felépítését és működését. Majd a Naiv Bayes algoritmus továbbfejlesztéséhez felhasználom és bemutatom az Expectation-Maximization algoritmust, mellyel pontosabb eredményeket lehet elérni.

Emellett elméleti betekintést adok a szövegosztályozásban felhasznált specifikus megoldásokba, például dimenziócsökkentés, szemantikus mezők használata, szótövezés. Ezen technológiák jelentőségét online újság cikkeken alkalmazott osztályozás segítségével bemutatom, és hatásukat részletesen elemzem.

Abstract

In this document I explore the possibility of classification of documents by their author.

First, I introduce the basics of datamining and specifically the text mining, starting with clustering. First, demonstrating the probability based clustering, then present an often used data model, the topic-model and an algorithm that uses this model, known as Latent Dirichlet Allocation.

After that I describe what classification is, what the difference between clustering and classification is. Introduce one of the most used text classifier algorithm, the Naive Bayes algorithm. Show its model and functionality. Present an iterative technique for improving the model and the precision of the Naive Bayes algorithm, called Expectation-Maximization, which uses not just the labelled data for creating its probability model, but makes use of the unlabelled data as well.

Finally, I introduce the basic problems during text classification and demonstrate the impact of the different solutions on those problems. Mostly focusing on the frequency-based selection, the size of the training set and the usage of semantic fields. In this example the data is made up by online news articles.

1 Bevezetés

A mai világban minden az adatok körül forog, mindenkinek a kezében található egy telefon, ami egyre gyakrabban mindig fel van csatlakozva az internetre, másodpercenként több ezer facebook és twitter üzenetek keletkeznek. Internetes blogokon és hírforrásokon másodperceken belül értesülhetünk arról, hogy mi történt a bolygónk másik felén. Az adatbányászat feladatául tűzte ki, hogy ennek a nagy mennyiségű adatból olyan információkat nyerjenek ki, melyeket fel tudunk használni ahhoz, hogy számunkra már átláthatatlan rendszereknek jelentést adjunk, felismerjünk mintákat és ismétlődéseket, trendeket.

Bár legtöbb esetben az adatbányász algoritmusok a numerikus adatok feldolgozására lettek kitalálva, vannak modellek és algoritmusok, melyek kifejezetten az emberi szöveg összefüggéseit próbálja figyelembe venni. Jelen dokumentum is ilyen algoritmusokról, alkalmazásukról és működésükről fog szólni.

Saját célom az volt, hogy elegendő adat felhasználásával egy szöveg tartalmából annak írójára tudjak rájönni. Ezen belül pedig, hogy megvizsgáljam, hogy maga az algoritmus milyen információkra hagyatkozva jutott a jó vagy rossz döntésre. Ezen ötletet maga az internet és annak bizonyos anonimitása adta, hogy megtudjam mégis mennyire nehéz ezt az anonimitást feloldani. Jelen dokumentumban online újság cikkeken fogjuk végig nézni, hogy az adatbányászati algoritmusok hogyan működnek és milyen pontosságot tudnak elérni a cikkek írójának eldöntésében.

2 Feladat értelmezés és kifejtés

2.1 Irodalomkutatás

Feladatom megoldásához különböző csoportosító vagy osztályozó algoritmust kellett keresnem, melyek alkalmasak lehetnek arra, hogy a dokumentumok olyan tulajdonságaira fókuszáljanak, melyek segítségével a dokumentumok írójának sajátosságait ki tudják nyerni.

A csoportosító algoritmusok használatát hamar elvetettem, mivel nem használnak a modelljük építéséhez tanítóhalmazt, így nem lehetett a tanulásuk irányát befolyásolni. Csak a Latent Dirichlet Allocation[7] alkalmazását próbáltam ki az előbbi feltételezésnek a demonstráláshoz, ennek az algoritmusnak a modellje nagyban hasonlít egy osztályozó algoritmusban felhasznált generatív modellhez, ezért került rá a választás.

Az leggyakrabban alkalmazott osztályozó algoritmusok kutatásom alapján: Tartóvektor-gép[4] (Support Vector Machine) algoritmus, k-Nearest Neightbors algoritmus[14], Naiv Bayes[1] algoritmus és a Neurális Hálók.

A Tartóvektor-gép algoritmus alapvetően csak két osztály közül képes döntést hozni, így nem lehetne alkalmazni alkalmazásunkban, ahol több író is jelen van az adathalmazunkban. Megjegyzendő, hogy vannak módosításai melyekkel lehet alkalmazni több osztályra is[6].

A neurális hálók alkalmazására leggyakrabban a képfeldolgozás során lehet hallani, viszont felhasználásuk nem korlátozódik e területekre csak. Ismert algoritmusok a szövegek osztályozására a Convolutional Neural Network (CNN) és a Recurrent CNN[10].

A k-NN algoritmus esetén a dokumentumokat a tér pontjaiként reprezentáljuk és az osztályozást a legközelebbi szomszédos dokumentumok alapján döntünk arról, hogy mely osztályba soroljuk be az adott dokumentumot. Az algoritmus jól tanítható, és egy jó megközelítését adja a szövegek osztályozásának.

A Naiv Bayes algoritmus egy a LDA algoritmushoz hasonló generatív modellből indul ki, melyet feltételezve a Bayes tételt felhasználva visszatudja következtetni a dokumentum osztályát. Modelljének egyszerűsége ellenére gyakran és nagy pontossággal felhasznált algoritmus.

Az algoritmusok mellett a gyűjteményre alkalmazott technikák is léteznek, melyek segítségével jobb reprezentációt tudunk elérni a feladatunkhoz.

Ilyen eszköz a szemantikus mezők, felhasználásával a dokumentumaink nagydimenziójú méretér le tudjuk csökkenteni egy sokkal könnyebben megfogható kategóriákra, melyekkel ki tudjuk emelni az osztályaink közötti különbséget[3].

Más lehetőségünk az osztályozás hierarchiába rendezése, először egy magasabb szinten, vagy más szempontból osztályozzuk a dokumentumainkat, hogy ezeken az osztályokon belül már lecsökkentsük a keresett osztályok jelenlétét. Például először nagyobb témakörökbe rendezzük a dokumentumokat, mint sport, politika, stb. majd ezeken belül alosztályokba osztályozunk, mint tenisz, labdarúgás, kosárlabda, stb.

Végül egy technológia, mellyel tovább javíthatunk a modellünk helyességén azzal, hogy a tanítóhalmazon kívül az osztályozandó adatot is felhasználjuk a finomításához. Az Expectation-Maximization algoritmussal a modellünkben felhasznált rejtett változók és a keresendő paraméterek iterált újraszámolásával tudunk javítani modellünkön[12].

2.2 Választott Algoritmusok

Jelen dokumentumunk célja, hogy az algoritmusok által a dokumentumokban megragadt sajátosságokat tudjuk megfigyelni, és hogy minél pontosabb eredményt hozzunk ki a felhasznált algoritmusokból.

Ezt az elvet követve a neurális hálót használó algoritmusok modelljének komplexitása miatt a kapott eredmények részletes elemzése nehéz, sokszor értelmezhetetlen. Ezen algoritmusokat, így nem alkalmaztam a feladathoz.

A k-NN algoritmus működése mögötti megfontolások alkalmasak lehetnek a dokumentumok írójának jellegzetességeinek megragadására. A nem felhasználására az Expectation-Maximization algoritmus adott okot. Az EM algoritmus ugyanis nem alkalmazható a k-NN algoritmusra.

Így végül a talált technológiák közül a Naiv Bayes algoritmust használom fel, az Expectation-Maximization és a szemantikus mezők felhasználásával, hogy jobb modell reprezentációt érjek el.

3 Szövegbányászat

3.1 Elő feldolgozás, modellkészítés

A legtöbb adatbányászati algoritmus nem arra készült, hogy szövegeket dolgozzunk fel vele, így különböző megoldásokat kell keresni, hogy a szöveget számokkal másképp ábrázoljuk. A leggyakoribb megoldás a szózsákmodell, mely figyelmen kívül hagyja a nyelvtani összefüggéseket és a szövegre úgy tekint, mint szavaknak véletlenszerű halmaza, végig megy a szövegen és megnézi, melyik szó hányszor szerepel benne és így egy vektorként reprezentálja.

Sajnos ez a reprezentáció így túl naiv, ahhoz hogy alkalmazható is legyen. A szövegekre jellemző, hogy rengeteg különböző szóból állnak, így a szózsákmodell dimenziója hamar túl nagy lesz. Például jelen feladatban ~500 dokumentumot használtam fel és 40 ezer különböző szót tartalmaztak bármiféle elő feldolgozás nélkül. Ennek a problémának a kiküszöbölésére több módszert is alkalmazunk, melyek azon megfontolásokon alapulnak, hogy egy szó bevétele a szózsákmodellbe mekkora jelentéssel bír a gyűjteményben.

Elsőnek, vannak szavak melyek jelenléte szükséges a nyelvtani szabályok betartása miatt, viszont a szöveg tartalmához kevés jelentőséggel bírnak. Ezalatt leggyakrabban a névmásokat, névelőket és kötőszavakat értjük. Ezeket hívjuk stopszavaknak (angolul stopword), leggyakoribb eljárás pedig, hogy elhagyjuk őket.

Második megfontolás, abból származik, hogy a legtöbb nyelv ragozza a szavakat, így gyakori eset, hogy ugyanabban a szöveg megtalálható egy szó és annak több ragozott alakja is. Ha naivan implementáljuk a szózsákmodellt, akkor ezek a ragozott formák mind egy újabb dimenzióként jelennek meg a modellünkben. Ennek megoldásaként alkalmazzuk a szótövezést (angolul stemming), mely az összes szó végéről (vagy egyes nyelvekben az elejéről) eltávolítja a különböző ragokat, így csökkentve a szavak halmazát.

Az előző két lépést bármilyen szövegbányász alkalmazás során érdemes elvégezni, modelltől és alkalmazástól független a felhasználásuk. Az utolsó lépésben pedig már figyelembe véve a szövegek típusát és az alkalmazásunk célját tovább csökkentjük a szavak halmazát. Leggyakoribb módszer a frekvenciaalapú dimenzió

csökkentés. Megnézzük, hogy a szavak hányszor fordulnak elő a gyűjteményünkben és gyakoriság alapján sorba rendezve elhagyjuk a leggyakoribb és a legritkább szavak egy részét. A gondolat ez a lépés mögött, hogy azok a szavak, amik túl gyakran fordulnak elő, azok az összes dokumentumban gyakran fordulnak elő, így nem segítenek abban, hogy megkülönböztessük őket egymástól. Hasonlóképp a ritkán előforduló szavak valószínűleg csak néhány dokumentumban vannak jelen, így ezek pedig nem segítenek abban, hogy hasonlóságot találjunk köztük. Szoktak még alkalmazni más módszert is a frekvencia mellett, például entrópia alapú, mely a szavak jelentőségét próbálja megadni a gyűjteményben, viszont jelen alkalmazásunkban a frekvencia alapú dimenzió csökkentés megfelelőnek bizonyult. Ha ezek a lépések után is úgy találjuk, hogy túl nagy maradt a gyűjteményünk szótára, akkor az előző lépésben használt mérőszámok szerint kiválaszthatunk egy adott mennyiségű szót belőlük, például az 5 ezer leggyakoribb szót. Itt már nem lehet teljesen általános megfontolást tenni, hiszen olyan szavakat hagyunk el, melyek akár meghatározóak is lehetnek egyes alkalmazásokban, viszont másikban pedig zajt visznek a rendszerbe.

3.2 Csoportosítás

Az adatbányászaton belül egy gyakori eljárás, hogy az adatot valamilyen hasonlósági faktor szerint csoportokba szeretnénk rendezni. Ezt a folyamatot hívjuk csoportosításnak (angolul clustering). Kifejezetten a szövegbányászat világában gyakori és széleskörű a felhasználása, gondoljunk például egy ajánló listára bármilyen tartalommegosztó oldalon, azokat az elemeket kell kitenni egy ajánló listára, melyek valamilyen logika szerint hasonlítanak vagy a most megnyitott tartalomhoz, vagy ami leginkább illeszkedik a felhasználó szokásaihoz.

Bár vannak általános felhasználású csoportosító algoritmusok, mint például k-átlag vagy hierarchikus algoritmusok, ezek legtöbbször nem működnek jó pontossággal a szövegek speciális tulajdonságai miatt, melyeket az előző pontban tárgyaltunk.

A csoportosítás egy változata a valószínűség alapú csoportosítás. A valószínűség alapú csoportosítás alap feltevése, hogy az adatok hasonlóságot mutatnak az összes csoporttal, de különböző mértékben, így tekinthetünk erre a hasonlóságra úgy, mintha egy valószínűség lenne. Ez a reprezentáció lehetőséget ad olyan alkalmazásokra is, amikor az adatot diszkrét halmazokba csoportosítjuk, hanem a rá leginkább illeszkedő címkéket keressük.

3.2.1 Valószínűség alapú szöveg csoportosítás és a topic-model

Egy elterjedt változata a valószínűség alapú szöveg csoportosításnak a topic-model-ezés. A topic-model algoritmusok generatív modellt követnek. Az alap feltevések a topic-model készítés során:

- Az *n* darab dokumentumról a gyűjteményben feltesszük, hogy valamilyen valószínűséggel tartoznak a k témakörbe (angolul topic). Egy adott dokumentum D_i és a témakörök megadásával $T_1 \dots T_k$, annak a valószínűsége, hogy az i. dokumentum a j. témakörbe tartozik a $P(T_j|D_i)$ kifejezés adja meg. Ezen valószínűségek kiszámolása a csoportosító algoritmus egyik fő kimeneteli eredménye.
- A témakörökre úgy tekintünk, mint valószínűségi eloszlásokra a gyűjtemény szavai felett. Legyenen $t_1 \dots t_d$ a gyűjteményben előforduló szavak. Annak a valószínűsége, hogy az t_l szó előfordul a T_j témakörben a $P(t_l \mid T_j)$ kifejezés adja meg. Ezen valószínűség érték szintén egy fontos paraméter, melyet az algoritmusnak meg kell becsülnie.

A legtöbb topic-model algoritmus ezeket a paramétereket maximum likelihood metódusokkal próbálják megtanulni, hogy a legjobb illeszkedést kapják a gyűjtemény dokumentumaira. Ilyenek például a Probabilistic Latent Semantic Indexing[5] (PLSI) és a Latent Dirichlet Allocation (LDA). Ebben a dokumentumban csak az LDA algoritmussal fogunk foglalkozni.

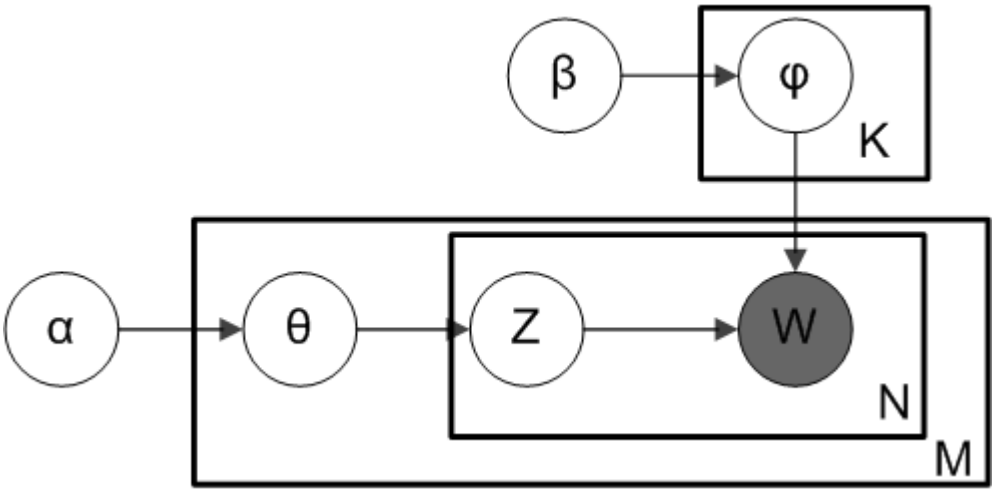
3.2.2 Latent Dirichlet Allocation algoritmus

A LDA egy valószínűség alapú szöveg csoportosító algoritmus, mely generatív modellen alapszik. Feltételezi, hogy a dokumentumok, amiket megadunk véletlenszerűen generálódtak az alábbi módon:

- Véletlenszerűen választunk egy dokumentum hosszt (valamilyen valószínűsége eloszlás szerint, például: poisson-eloszlás).
- Választunk egy témakör eloszlást a dokumentumhoz (dirichlet eloszlást követve a k témakör felett).
- Generáljuk le az összes szót a következőképpen:

- Először válasszunk véletlenszerűen egy témakört az előbb választott eloszlás szerint. Majd a kiválasztott témakör szóeloszlása szerint válasszunk egy szót.

Ezt a generatív modellt feltételezve az algoritmus megpróbálja visszaszámolni a valószínűségeket a dokumentumokból, hogy megtalálja a témaköröket, amik generálhatták a gyűjteményünket. A modellt az alábbi Bayes hálóval írhatjuk le.



3.1. ábra LDA algoritmus Bayes modellje

α és β paraméterei a dirichlet eloszlásnak. θ adja meg a dokumentumok témakör eloszlását. φ adja meg a szavak eloszlását a témakörök felett. z a témaköre a dokumentumok szavainak. w pedig a dokumentumok szavai. Ebből a modellből w az egyetlen megfigyelhető változó, α és β értékeit pedig megadjuk.

3.3 Osztályozás

A csoportosítás és osztályozás között a fő különbség, hogy míg csoportosítás során csak általánosan szét akarjuk osztani az adatunkat valamely tulajdonságuk során, osztályozás esetén előre definiált osztályaink vannak, és arra vagyunk kíváncsiak, hogy az adataink melyik osztályba tartoznak. Az előre definiált osztályokból következően osztályozáskor tudunk felhasználni tanítóhalmazt, hogy a kezdeti modellünk jobban reprezentálja az elvárt osztályokat.

3.3.1 Naiv Bayes osztályozás

A LDA algoritmus hátránya, hogy nem tartalmaz megerősítéses tanulást, ennek orvoslására felhasználhatjuk a Naiv Bayes algoritmust, mely az LDA algoritmushoz hasonló generatív modellből indul ki. A [9] alapján: legyen M az osztályok száma, $|X|$ a dokumentumok száma, x_i az i-edik dokumentum $|x_i|$ pedig a dokumentum szavainak száma, a valószínűségi modellt jelölje θ. Egy dokumentumot, x_i-t, úgy hozzuk létre, hogy először generálunk egy osztályt neki, jelöljük ezt c_j-nek (c_j eleme {1...M} halmaznak), a valósínűségét $P(c_j \mid \theta)$-val jelöljük. A dokumentumok generáljuk a kiválasztott osztály eloszlásával, ennek valószínűsége $P(x_i \mid c_j, \theta)$. Tehát annak a valószínűsége, hogy a dokumentumot generáljuk a jelenlegi modellünkkel:

$$P(x_i|\theta) = \sum_j P\left(c_j|\theta\right) P\left(x_i|c_j, \theta\right). \tag{3.1}$$

Minden dokumentum egy osztályhoz tartozik, ezt jelöljük y_i-vel, tehát ha az x_i dokumentum a c_j osztályból lett generálva azt mondjuk, hogy $y_i = c_j$.

A szózsákmodellből következően a dokumentumok szóelőfordulások vektora, legyen x_{it} a w_t szó előfordulásának száma az x_i dokumentumban. Ezekből ki tudjuk bontani az előző képlet második kifejezését:

$$P\left(x_i|\, c_j, \theta\right) \propto P(|x_i|) \prod_{w_t \in X} P\left(w_t|\, c_j, \theta\right)^{x_{it}}. \tag{3.2}$$

Ez az összefüggés tartalmazza az alap Naiv Bayes osztályozás feltevését, miszerint a dokumentum szavainak generálása függetlenek egymástól. A jelenlegi feladat megoldásához a dokumentum hosszának generálásától eltekintünk, úgy vesszük, hogy egyenletes eloszlás szerint lettek sorsolva.

A Naiv Bayes osztályozás megtanulása egy felcímkézett dokumentumhalmazból abból áll, hogy a modell paramétereit megbecsüljük. A θ paraméter megbecsültjét θ'-vel jelöljük. A naiv Bayes algoritmus maximum a posterior (MAP) becslést alkalmaz, vagyis keresi a $arg\, max_\theta P(\theta|X, Y)$. Ez a legvalószínűbb modell megadva a tanítóhalmazt és egy priort.

A mi esetünkben a prior egy dirichlet eloszlás[2], egy polinom (angolul multinomial) eloszlás minden osztályhoz és még egy az osztályokhoz összességében. A dirichlet eloszlás formulája:

$$P\left(\theta_{w_t|c_j}|\alpha\right) \propto \prod_{w_t \in X} P\left(w_t|c_j\right)^{\alpha_t - 1} \tag{3.3}$$

Ahol $\theta_{w_t|c_j} \equiv P\left(w_t|c_j, \theta\right)$ és az α_t értékek pedig pozitív konstansok. Az alkalmazásunkban az összes α értéket 2-nek választottuk, ami egy egyenletes eloszlású priorhoz vezet. A szó előfordulási valószínűségek becsült értéke, Lagrange simítást alkalmazva:

$$\theta'_{w_t|c_j} = \frac{1 + \sum_{x_i \in X} \delta_{ij} x_{it}}{|X| + \sum_{s=1}^{|X|} \sum_{x_i \in X} \delta_{ij} x_{is}}, \tag{3.4}$$

ahol δ_{ij} értékét a dokumentum osztályából számoljuk, 1, ha $y_i = c_j$ és 0 egyébként. Az osztály előfordulási valószínűségek pedig:

$$\theta'_{c_j} = \frac{1 + \sum_{i=1}^{|X|} \delta_{ij}}{M + |X|}. \tag{3.5}$$

Megadva a változók becsült értékét egy felcímkézett tanítóhalmazból, megoldható, hogy megfordítsuk a generatív modellt és kiszámítsuk a valószínűségét annak, hogy egy adott osztályból generáltuk a dokumentumot, így osztályozva a felcímkézetlen dokumentumokat. Annak a valószínűsége, hogy egy dokumentum egy adott osztályba tartozik a becsült modell alapján:

$$P\left(y_i = c_j | x_i, \theta'\right) = \frac{P(c_j|\theta\prime) P(x_i|c_j, \theta\prime)}{P(x_i|\theta\prime)} \tag{3.6}$$

Ha alkalmazásunkban a feladat, hogy egy osztályba soroljuk be a dokumentumokat, akkor a legnagyobb valószínűséggel rendelkező osztályt választjuk ki.

3.4 Expectation-Maximization algoritmus

Az Expectation-Maximization algoritmust, továbbiakban EM, maximum likelihood paraméterek keresésére alkalmazzuk, olyan modellekben, amikben az egyenleteket nem tudjuk megoldani közvetlenül. Általában ezek a problémák tartalmaznak rejtett vagy ismeretlen paramétereket és ismert megfigyelt adatot. A maximum likelihood megoldás megkereséséhez általában szükséges a likelihood függvény deriváltjának kiszámolása az ismeretlen értékeket figyelembe véve, és a kapott

egyenletek megoldása. Ez a feladat sokszor nem lehetséges statisztikai feladatok során, ehelyett, az eredmény általában olyan egyenletek, ahol az ismeretlenek megoldásához szükségesek a rejtett paraméterek értékei és fordítva. Az EM algoritmus erre a problémára nyújt egy megoldást felhasználva a megfigyelt adatot. Induljunk el véletlenszerűen generált rejtett paraméterekkel, majd ezekkel az értékekkel számoljuk ki az ismeretleneket a rendszerünkben, majd az így kapott értékekkel számoljuk újra a rejtett paramétereket, ezen két lépés között alternálva számolunk addig, amíg a megoldásunk nem konvergál valamilyen megoldáshoz.

3.4.1 EM alkalmazása a Naiv Bayes algoritmus kibővítésére

A felügyelt tanulás során két halmazunk van, egy felcímkézett tanítóhalmaz és egy felcímkézetlen halmaz. Mivel nincsenek címkék a felcímkézetlen halmazon, ezért nem tudjuk rájuk alkalmazni az előző szekció képleteit. Viszont, az EM algoritmust alkalmazva tudunk találni lokális MAP becsléseket a generatív modell változóira.

Először lefuttatjuk a Naiv Bayes algoritmust az előbb ismertetett módszerrel a tanítóhalmazon, így megkapva a modellünket. Majd osztályozzuk a felcímkézetlen halmazt e modell alapján. Az így kapott címkéket, mint helyes eredményt felhasználva újraépítjük a modellünket a teljes adathalmazt felhasználva. Iteráljuk az algoritmust a felcímkézetlen adat osztályozása és a naiv Bayes modell újraépítése között, míg a modellünk el nem kezd konvergálni.

Másképp megközelítve, az osztályozáshoz megpróbáljuk kiszámolni a maximum a posteriori becslését θ-nak, esetünkben $argmax_\theta P(\theta)P(X,Y|\theta)$, ami ekvivalens azzal, mintha ennek a kifejezésnek a logaritmusát maximalizálnánk. Modellünk szerint ennek képlete:

$$l(\theta|X,Y) = \log\big(P(\theta)\big) + \sum_{x_i \in X_u} log \sum_j P\big(c_j\big|\theta\big)P\big(x_i\big|c_j,\theta\big) +$$

$$\sum_{x_i \in X_l} \log(P\big(y_i = c_j\big|\theta\big)P\big(x_i\big|y_i = c_j,\theta\big)). \quad (3.7)$$

Érdemes megfigyelni a képlet második felét, a valószínűsége egy felcímkézetlen dokumentumnak a valószínűségek összege az összes osztályhoz. Ezzel szemben a felcímkézett dokumentumok esetén már tudjuk az osztályát, így nincs szükség a többi osztályhoz tartozó valószínűséget figyelembe venni.

Vegyük észre, hogy a kifejezésben egy összeg logaritmusa szerepel, ami nagyban megnehezíti a derivált kiszámítását. Ebben a problémában segít az EM algoritmus, ami iteratív módon keresi meg egy lokális maximumot a paraméter térben. Az M lépés maximalizálja a modell valószínűségét az E lépésben kiszámolt hiányzó értékek felhasználva, mintha valódi értékek lennének.

3.5 Szemantikus mezők

A szöveg csoportosítás és osztályozás során egy, már említett, probléma a szövegek dimenziójának mérete. A szemantikus mezők felhasználásával jelentősen lecsökkenthetjük a dokumentumok vektor terét. Így felgyorsítva az algoritmusok futását és remélhetőleg pontosságát is. A mi alkalmazásunkhoz hasonló feladatban láthatjuk felhasználását a [3] dokumentumban.

Az alkalmazása meglehetősen egyszerű. A dokumentumok szavai tematikájuk szerint csoportokba rendezzük, például: ige.mozgás, főnév.forma, stb. Majd leképezzük őket ezekre a mezőkre, így drasztikusan lecsökkentve a szavak terét.

Felhasználás során azt használjuk ki, hogy egyes csoportjaink nagy valószínűséggel a szemantikus mezők közül kevésben nagyon kiugróak lesznek, a többségében pedig elenyésző lesz a jelenlétük, ezzel kinagyítva a dokumentumok közötti különbséget.

Ezen alkalmazásban a szemantikus mezőket a WordNet[8] lexikonját használtam fel, mely az angol szavaknak egy nagy halmazát lefedve a szavakat 45 kategóriába rendezi. A dokumentumainkat még stopszó szűrés és szótövezés előtt leképezzük e mezőkre, majd az így kapott dokumentumokon futtatjuk algoritmusunkat. A frekvencia alapú dimenziócsökkentés alkalmazása ebben a csökkentett térben is javíthat az algoritmusunk eredményén, ha egyes szemantikus mezőkön túl nagy hasonlóságot mutatnak az osztályaink/csoportjaink.

4 Feladat implementálása

4.1 Adatgyűjtés

Az alkalmazáshoz szükséges volt olyan szövegekre, melyeknek tudjuk az íróját. Ehhez online újságcikkeket választottam, amiket a New York Times online hírforrás cikkeit használtam fel. Kiajánlott API-juk segítségével kerestem olyan cikkeket, amikhez van író fűzve és ezeknek a cikkeknek a tartalmát pedig a cikk webcímét lekérdezve gyűjtöttem be az alábbi szkripttel:

```
api_request = requests.get(base_url + '?begin_date=' +
                        begindate.strftime('%Y%m%d') + '&end_date=' +
                        enddate.strftime('%Y%m%d') +
                        '&fl=web_url%2Cbyline&page=' + str(page))
        docs = api_request.json()['response']['docs']
        for j in range(0,len(docs)):
            url = docs[j]['web_url']
            byline = docs[j]['byline']
            if (byline is not None and byline) and (not
url.startswith('http://www.nytimes.com/video')):
                people = byline['person']
                if people:
                    person = people[0]
                    if 'firstname' in person and 'lastname' in person:
                        try:
                            r = requests.get(url, headers=headers)
                            htmlpage =
html.fromstring(r.text.strip().replace('\n','').replace('\r','').replace('
"',''))
                            articleContent =
htmlpage.xpath('//p[@class="story-body-text"]/text()')
                            articletext = ''.join(articleContent)
                            if(articletext != ''):
                                with
open('nytimesarticles.csv','a',encoding='utf8') as f:
                                    f.write('"'  +  person['firstname']  +
person['lastname'] + '";"' + articletext + '"\n')
                        except Exception as ex:
                            print('error occured in a request')
```

Célom az volt, hogy 15-20 íróhoz legalább 15-15 cikket gyűjtsek össze, hogy az írókról viszonylag meghatározó mennyiségű cikkem legyen és tudjak az algoritmusok esetén a tanítóhalmaz méretével is konfigurálni.

Az algoritmusok első futtatása során észrevettem néhány kiugró értéket, ezért ránéztem az összegyűjtött adathalmaz tartalmára is. Kiderült, hogy volt rengeteg spanyol nyelvű cikk az adathalmazomban, illetve sok cikkhez hibás volt a tartalom lekérdező logika, így csak néhány szóból állt, amiket gyakran a stopszó szűrés és a frekvencia alapú

dimenziócsökkentés teljesen kiszűrt. Ezért úgy döntöttem, hogy kiszűröm az összes 500 karakternél rövidebb cikkeket.

Végül a gyűjteményem 14 írót tartalmazott, összességében 547 újságcikkel. Átlagosan 30-55 cikk tartozott egy íróhoz, de tartalmaz néhány kiugró esetet is szándékosan, 2 írónak csak 16 cikke van, illetve egy másiknak 150. Bár a kiugró értékek egyes esetekben torzították az eredményeket, ezek az eloszlások jobban reprezentálják a valóságot, hiszen nem minden íróhoz áll rendelkezésünkre ugyanannyi adat.

4.2 Java implementáció

Valószínűségekkel működő algoritmusok esetén gyakori problémát jelent, hogy az alkalmazásunk számábrázolása nem elég pontos. Jelen alkalmazásban szerencsére csak ~500 dokumentumom volt és a szavak számának lekorlátozása miatt a modellhez tartozó valószínűségek még a Java double számábrázolásának határain belül maradt. Viszont az EM algoritmussal kibővített naiv Bayes algoritmus log valószínűségének kiszámolásakor nagyon hamar túl lépjük ezt a korlátot. Erre a problémára nem találtam olyan, megoldást mellyel meg tudnánk tartani a konvergencia figyelését, mint funkció, ilyen megoldás lehetne egy 128 bites lebegő pontos számábrázolás. Ehelyett fix iteráció számmal futtattam az algoritmust, de néhány teszt futtatás alatt azt vettem észre, hogy az EM algoritmus elég gyorsan konvergál megoldáshoz, így ez is megfelelő megoldás.

Másik komoly probléma volt, hogy a képletek kiszámolása egyszerű implementációval meglehetősen sokáig tartott. Ennek megoldására több kiszámítandó értéket kivezettem belső változóba, hogy az adott képlethez ne kelljen többször ugyanazt az értéket kiszámolni, illetve a naiv Bayes algoritmus δ_{ij} értékeinek kiszámolása helyett a dokumentumok adott osztályra szűrésével, rengeteg 1-gyel való szorzást és 0 értékek összeadását kiváltottam.

4.2.1 Program bemutatása

A program futása három fázisból áll ezek: az adatok beolvasása és egyszerűsítése, az adott algoritmus futtatása, az algoritmus eredményeinek kiolvasása/kimentése.

Az adatokat csv formátumú fájlokban tároltam, az író nevét és a cikkek szövegét tartalmazza, még eredeti formájukba írásjelekkel. A transzformáció többszöri elvégzésének elkerülése érdekében a szemantikus mezőkre leképzést előre elvégeztem az alábbi szkripttel:

```
semanticFieldsMap = {}
notCharactersRegex = re.compile('[^A-Za-z ]')
for semanticField in os.listdir(semanticFieldDirectory):
    with
open(os.path.join(semanticFieldDirectory,semanticField),'r',encoding='utf8
') as file:
        for word in file:
            semanticFieldsMap[word[:-1]] = semanticField

with open(articlesPath,'r',encoding='utf8') as articlesFile:
    with open(outputFilePath,'w',encoding='utf8') as outputFile:
        for article in articlesFile:
            author = article.split('";"')[0][1:]
            newArticleText = []
            articleText = article.split('";"')[1][:-1].replace('\t',' ')
            for i in range(1,100):
                articleText = articleText.replace('  ',' ')
            articleText = notCharactersRegex.sub('',articleText)
            for word in articleText.split(' '):
                if word in semanticFieldsMap:
                    newArticleText.append(semanticFieldsMap[word])

            outputFile.write('"' + author + '";"' + ' '.join(newArticleText)
+ '"\n')
```

A fájlból beolvasás után felépítjük a gyűjteményt reprezentáló osztályt, a Corpus-t. Ennek az osztálynak feladata a fájlból beolvasása az adatoknak, a szótövezést és a dimenziócsökkentést az inicializálás során végezzük el két segédosztály felhasználásával. A szótövező egy külső könyvtárat használ fel a Snowball Stemmer implementációját[11]. A szótövezést és a stopszavak szűrését az alábbi kóddal végezzük el rögtön a fájlból beolvasás után:

```
String[] words = articleContent.split(" ");
words = dimensionReducer.filterStopWords(words);
StringBuilder sb = new StringBuilder();
for(String word : words)
{
    sb.append(stemmer.stem(word)).append(' ');
}
```

A stemmer az előbb említett Snowball Stemmer egy példánya. A dimenziócsökkentéshez egy saját implementációt készítettem, két algoritmust tartalmaz a dimenziócsökkentés, a stopszavak szűrését és a szavak dimenziójának frekvenciaalapú csökkentését. A dimenziócsökkentés elvégzéséhez a teljes gyűjteményre szükség van, ezekből megszámoljuk, hogy a szavak hányszor fordulnak elő, majd előfordulásuk alapján rendezzük ezen listát.

```
HashMap<String, Integer> wordCount = calculateWordCount(articles);

List<String> wordList = new ArrayList<>();
wordList.addAll(wordCount.keySet());
```

```
Collections.sort(wordList, (s1,s2) ->
Integer.compare(wordCount.get(s1),wordCount.get(s2)));
```

A kapott lista felhasználásával annyi feladatunk maradt, hogy a cikkek szavait kiszűrjük a szavak előfordulása alapján.

```
List<Article> reducedArticles = new ArrayList<>();
for(Article a: articles)
{
    StringBuilder sb = new StringBuilder();
    for(String word: a.TextAsWords)
    {
        int wordIdx = wordList.indexOf(word);
        if(wordIdx < wordList.size()*leastFrequentCutPercentage ||
           wordIdx > wordList.size()*(1- mostFrequentCutPercentage) ||
           wordIdx  <  wordList.size()*(1-  mostFrequentCutPercentage)  -
maxNumberOfWords)
            continue;
        sb.append(word);
        sb.append(" ");
    }
    if(sb.toString().length() == 0)
        throw new RuntimeException("Empty articles");
    reducedArticles.add(new Article(a.Author,sb.toString()));
}
return reducedArticles;
```

Különböző konfigurációk kipróbálása során gyakran futottam abba a problémába, hogy a dimenziócsökkentés miatt egyes dokumentumok összes felhasznált szavát kiszűrtem. Ennek kezelésére két lehetőségünk van, ezeket a dokumentumokat eldobjuk, vagy ilyen konfigurációkat nem alkalmazunk. Az utóbbi mellett döntöttem, hiszen egy dokumentum összes szavának kiszűrése inkább a rossz konfiguráció jele, mint a rossz dokumentumé. Ezek után megkapjuk a kiválasztott cikkek végleges listáját, egy ilyen dokumentumot reprezentál az Article osztály.

Az algoritmusok implementációi során gyakran hivatkozunk a dokumentumok szavaira index szerint, vagy egy adott szó előfordulásának mennyiségére. Ezen kérések kiszolgálásának gyorsítására az Article osztály tömbként is elérhetővé teszi a dokumentumok szövegét, így egyszerűvé téve a lekérdezést.

```
public final String[] TextAsWords;
this.TextAsWords = text.split(" ");
```

A Corpus osztály pedig a dimenziócsökkentés után megkapott cikkekből felépít egy szó-dokumentum mátrixot, mely az adott szó adott dokumentumban előfordulását tárolja.

```
private  void initializeTermDocumentMatrix(List<Article> articles,
Map<String, Integer> wordMap)
{
    termDocumentMatrix = new Matrix(wordMap.size(), articles.size());
    for(int articleIdx = 0; articleIdx < articles.size(); articleIdx++)
    {
        Article article = articles.get(articleIdx);
        for(String word : article.TextAsWords)
        {
            int termIndex = wordMap.get(word);
            termDocumentMatrix.Increment(termIndex, articleIdx);
        }
    }
}
```

Ezen kiegészítések után az algoritmusokat implementáló osztályok ugyanazon Corpus példányon dolgoznak. Az algoritmusok implementálásakor szintén a teljesítményt figyelembe véve jártam el. Azokat a változókat, melyeket sokszor használunk fel számításokhoz, el tárolom egy belső változóba, hogy miután egyszer kiszámolta az algoritmus az értékét ne kelljen többször kiszámolni.

A LDA algoritmus implementálásához a Gibbl sampling nevű eljárást alkalmaztam. Először véletlenszerűen sorsoljuk ki az összes dokumentum összes szavához a témaköröket. Majd egyesével végig megyünk a dokumentumok szavai, levesszük a jelenlegi címkéjét az adott szónak, és a modellünk alapján újragenerálunk egy címkét. A tanító algoritmus a következőképp néz ki:

```
while(currIteration < numOfIterations)
{
    if(currDocumentIndex == numDocs)
        currDocumentIndex = 0;

    int numberOfWordsInDocument =
        documentWordsToTopicAssignment.get(currDocumentIndex).length;
    for(int wordIndex = 0; wordIndex < numberOfWordsInDocument; wordIndex++)
    {
        unassignWordOfDocument(currDocumentIndex, wordIndex);
        int newTopicIndex = generateNewTopic(currDocumentIndex, wordIndex);
        assignWordOfDocumentToTopic(currDocumentIndex,wordIndex,
newTopicIndex);
    }
    currDocumentIndex++;
    currIteration += numberOfWordsInDocument;
}
```

A címke levétele és feltétele csupán az osztály belső számlálóinak csökkentéséből és növeléséből áll, amik a szavak és dokumentumok témakörökhöz rendelésének számát tartják nyílván, hogy ezeknek felhasználásához ne kelljen minden egyes alkalommal végigmenni a teljes gyűjteményen. A fontosabb logika a generateNewTopic függvényben található. A következő címkéje a szónak véletlenszerűen generálódik, de a modell felépítése alapján súlyozva az egyes témaköröket. Annak a valószínűsége, hogy egy szót egy adott témakörhöz rendeljünk két részből áll össze. Először annak a valószínűsége, hogy a dokumentum, amiben vagyunk, milyen valószínűséggel tartozik az adott témakörbe, és hogy a szó az egész gyűjteményben milyen valószínűséggel tartozik a témakörbe. Ezek alapján a következő képletek értékeit kell kiszámolnunk:

$$P(y_d = c_k) = \frac{n_{d,k} + \alpha_k}{\sum_i^{|X|} n_{d,i} + \alpha_i} \tag{4.1}$$

$$P(w_{d,n} \mid c_k) = \frac{v_{k,w_{d,n}} + \beta_{w_{d,n}}}{\sum_i v_{k,i} + \beta_i} \tag{4.2}$$

A (4.1)-es egyenlet azt írja le, hogy egy dokumentum milyen valószínűséggel tartozik egy adott témakörhöz, ahol $n_{d,k}$ a d-edik dokumentum k-adik témakörbe tartozó szavainak száma. A (4.2)-es egyenlet pedig azt adja meg, hogy egy szó milyen valószínűséggel tartozik egy témakörbe, ahol $w_{d,n}$ a d-edik dokumentum n-edik szava, $v_{k,i}$ pedig azt adja meg, hogy az egész gyűjteményben az i szó hányszor van a k-adik témakörhöz rendelve. Az alfa és béta értékek pedig a dirichlet eloszlás paraméterei. Ezen két valószínűség összeszorzásából számoljuk ki annak a valószínűségét, hogy az adott dokumentum adott szava mely témakörhöz milyen valószínűséggel tartozik. Ezeket a valószínűségi értéket számolja ki a calculateWordToTopicProportions függvény, majd a következőképp választunk új témakört a szavunkhoz:

```
private int generateNewTopic(int currDocumentIndex, int wordIndex)
{
    double[] wordTopicProportions =
calculateWordToTopicProportions(currDocumentIndex,wordIndex);
    double sumOfWordTopicProportions = 0;
    for(double p: wordTopicProportions)
        sumOfWordTopicProportions += p;
    Random r = new Random();
    double random = r.nextDouble() * sumOfWordTopicProportions;
    for(int i = 0; i < numTopics; i++)
    {
        if(wordTopicProportions[i] > random)
            return i;
        else
            random -= wordTopicProportions[i];
```

```
        }
        return numTopics - 1;
    }
```

A Naiv Bayes algoritmus és az ismertetett EM algoritmus implementációja szinte teljesen megegyezik, hiszen a különbség annyi, hogy az EM algoritmus többször futtatja le a Naiv Bayes algoritmus modell felépítését és a dokumentumok felcímkézését. A modell felépítése a (3.4)-es és a (3.5)-ös egyenletek kiszámítását jelentik a tanítóhalmaz alapján, EM esetén későbbi iterációk során a teljes adathalmaz alapján végezzük el.

```
    private void buildModel(List<Article> articles)
    {
        for(int j = 0; j < numTopics; j++)
        {
            //A képletek kiszámolása
            …
            //A kiszámolt értékek eltárolása:
            for(int t = 0; t < numWords; t++)
            {
                termTopicMatrix.SetValue(t,j,(termTopicReckoner[t] + 1) /
                                             termTopicDenominator);
            }
            topicModel[j] = (1.0 + articlesForTopic.length) /
                            (numTopics + articles.size());
        }
    }
```

A dokumentumok felcímkézését pedig a (3.6)-os valószínűségek kiszámolásával tudjuk eldönteni. A tanítóhalmazban szereplő dokumentumok esetén nem számítjuk ki ezt a valószínűséget, hiszen tudjuk, hogy melyik íróhoz tartoznak. A felcímkézetlen dokumentumokhoz gyorsítás szempontjából érdemes észrevenni, hogy a képlet nevezője a témakörtől független érték, azt adja meg, hogy a dokumentum milyen valószínűséggel állhat elő általánosan, emiatt ennek az értéknek a kiszámítását meg tudjuk spórolni. Ezzel azt az információt viszont elveszítjük, hogy a kiszámolt értékek összege 1, így elesünk attól a gyorsítástól, hogy ha egy íróhoz tartozás valószínűsége több, mint 50% akkor biztos hozzá lesz rendelve.

```
    private void labelDocuments(List<Article> trainingSet)
    {
        //If labeled document use its label
        double[] posteriorProbability = new double[numTopics];
        for(int j = 0; j < numTopics; j++)
        {
            posteriorProbability[j] = topicModel[j];
            for(String word : a.TextAsWords)
            {
                int n = corpus.GetWordIndex(word);
                posteriorProbability[j] *= termTopicMatrix.GetValue(n, j);
            }
```

```
        }
    //Label unlabeled documents with the topic with the highest probability
        Y[i] = 0;
        for(int j = 1; j < numTopics; j++)
        {
            if(posteriorProbability[Y[i]] < posteriorProbability[j])
                Y[i] = j;
        }
    }
```

Az algoritmusok felépítették a modelljüket és felcímkézték a dokumentumokat, utolsó lépésünk, hogy az eredményeket kimentsük, ha szeretnénk, hogy későbbiekben a jelen állapota az algoritmusok modelljének visszatölthető legyen, akkor az ahhoz szükséges adatokat is kimentsük.

Az LDA algoritmus esetén a dokumentumok írókhoz rendelése nem egyértelmű, hiszen az LDA algoritmus tanítóhalmaz hiányában nem íróhoz rendelt csoportokból indul ki, hanem véletlenszerűen kialakítottakból. Ennek a problémának a feloldásáról az eredmények kiértékelésekor részletesebben beszélek. Ahhoz, hogy az LDA algoritmus jelenlegi modelljét vissza tudjuk állítani, ahhoz ismernünk kell a dokumentumok szavainak témakörhöz rendelését, hiszen ezen értékekből és a képletekből ki tudjuk számolni a modellben ismertetett témakörök feletti és a szavak feletti valószínűségi eloszlásokat. Ebből kifolyólag szükségünk van még a konkrét dokumentumokra és azoknak a kódban használt indexeikre.

A Naiv Bayes algoritmus és az EM algoritmus esetén a dokumentumok írókhoz rendelésén kívül, az írókhoz rendelt szóeloszlás értékeket és az írók feletti eloszlás értékeit szükséges lementeni, hogy visszaállíthatóak legyenek. Ahhoz, hogy az érékekhez fűzött jelentést is visszatudjuk nyerni, a gyűjteményünk szavaihoz rendelt indexeket és az írókhoz rendelt indexeket is szükséges kimenteni.

Ezen értékeket szintén csv formátumba mentettem ki, hogy rögtön be lehessen őket olvasni excel táblázatokba, ahol könnyen kiszámolható az eredményekből levonható statisztikai adat, mint precizitás és felidézés.

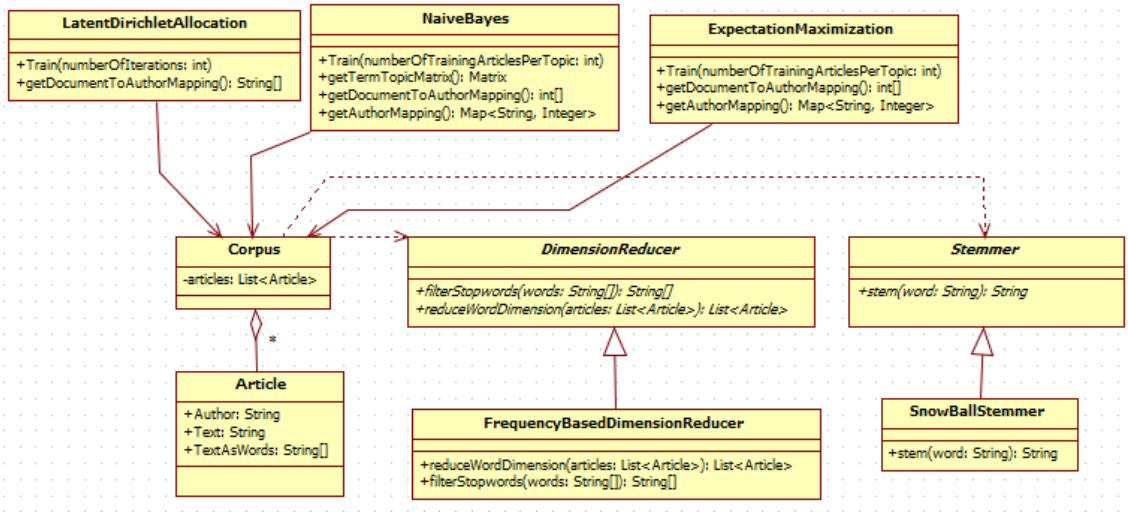
A főprogram célja az volt, hogy az algoritmusok és felhasznált technológiák konfigurációs beállításait kívülről, parancssorból, módosítani lehessen. Ehhez ezen paramétereket kivezettem parancssori argumentumokba, plusz egy index értéket, mely a kimeneti fájlok egyediségéhez volt szükséges.

A program többszöri, felparaméterezett futtatásához python szkriptet alkalmaztam. A kapott eredményeket algoritmusonként egy excel makró segítségével egy fájlba fűztem össze fájlba, eredményenként külön munkalapra.

Futtató szkript:

```
highCuts = ['0.0','0.01','0.05','0.1']
maxWords = ['2000','5000']
lowCut = '0.0'
trainingSetSizes = ['1','5','10']
for ix in range(1,20):
    for highCut in highCuts:
        for maxWord in maxWords:
            for trainingSetSize in trainingSetSizes:
                subprocess.call(…)
                subprocess.call(…) //With semantic fields
```

Összességében a program írásakor a programkód olvashatóságát és lazán kötöttségét próbáltam szem előtt tartani, hogy későbbi felhasználás során könnyen felhasználható legyen.



4.1. ábra A program egyszerűsített osztálydiagrammja

5 Algoritmus konfigurálása és eredmények

5.1 Konfigurációs lehetőségek

Az algoritmusok több dimenzióban is nyújtanak konfigurációs lehetőségeket. Az első és talán legfontosabb ilyen érték a frekvencia alapú dimenziócsökkentés adja. Ezen belül két értékünk van az alsó és felső vágási határ, jelen alkalmazásban csak azt vettem figyelembe, hogy hányadik helyet foglalják el a szavak a frekvencia szerinti sorrendben, viszont ennél komplikáltabb szabályt is meg lehet fogalmazni, például figyelembe vesszük az előfordulás tényleges értékét, vagy azt is megnézzük, hogy az előfordulás hogyan oszlik el a dokumentumokon. Az algoritmus futtatása során csak a felső vágási határt konfiguráltam. A dimenziócsökkentés tényleges jelentőségét megmutatván megnéztem 0 és 1%-os vágással az eredményeket és további vizsgálatokért 5 és 10%-os vágásokkal is futtattam az algoritmusokat. Sajnos 10%-os vágási határnál elértem, hogy egyes dokumentumok összes felhasznált szavát kiszűrtem, így ezzel a konfigurációval csak szemantikus mezőket is alkalmazva futtattam az algoritmust. Az alsó vágási határral a következő konfiguráció miatt nem foglalkoztam, mivel az esetek nagy részében nem lett volna hatása, értéke mindig 0% volt.

Második konfigurációs lehetőségünk a szavak számának mennyisége. Azt adjuk meg, hogy a felső vágás után a leggyakoribb szavakból hány darabot tartsunk meg a lexikonunkban. Megvizsgálási szempontból 2 különböző értéket próbáltam ki, egy alacsonyabbnak tűnő értéket 2000-t és egy közepes értéket 5000-t. Természetesen ki lehetne próbálni egy nagy értékkel is, hogy megmutassuk a ritkán előforduló szavak hatását az algoritmusokon, viszont a szavak mennyisége nagyban befolyásolja a futási időt.

A harmadik és nagy jelentőségű konfiguráció a tanítóhalmaz mérete. Ennek az értékét úgy adjuk meg, hogy írónként hány cikket tartalmazzon a tanítóhalmaz. Figyelembe véve, hogy a legkevesebb cikkel rendelkező író a gyűjteményünkben csupán 16 cikkel rendelkezik, 1, 5 és 10 cikk/író méretű tanítóhalmazokkal vizsgáltam meg az eredményeket. A cikkeket, amik belekerülnek a tanítóhalmazba, véletlenszerűen választjuk ki, emiatt érdemes az algoritmusokat többször lefuttatni és az eredményeket átlagolni, hogy megszabaduljunk a véletlenszerűség miatt bevitt inkonzisztenciáktól.

Az utolsó konfiguráció csupán abból áll, hogy alkalmazzuk-e a szemantikus mezőket az adatgyűjteményünkre vagy sem.

5.2 Eredmények értelmezése, mértékek

5.2.1 LDA algoritmus

A naiv Bayes és az ismertetett EM algoritmussal szemben az LDA algoritmus nem osztályozó algoritmus, így nem lehet számára tanítóhalmazt megadni. Emiatt az általa létrehozott csoportokról is nehezen mondható meg, hogy mely íróhoz tartozik ténylegesen.

Ahhoz, hogy ki tudjuk értékelni az eredményét az általa kiválasztott csoportokhoz úgy rendeltem hozzá az írót, hogy végül a lehető legtöbb összerendelése legyen helyes az algoritmusnak. Ez az eljárás megszegi azt a szabályt, hogy egy algoritmus tud-e a felcímkézetlen halmaz valódi címkéiről, így csak szemléltetés céljából vettem bele az eredményekbe. A legjobb lefedés megkereséséhez a „Magyar algoritmust”[13], másnevén Munkres összerendelő algoritmust, használtam fel.

5.2.2 Mértékek

Az algoritmusok eredménye egy cikk-író összerendelés. Ismervén a cikkek eredeti íróját ezekből meg tudjuk adni, hogy az összerendelések közül melyek helyesek. A legegyszerűbb mértékünk egy általános pontosság, hogy az algoritmus hány írót talált el a gyűjteményünkből. Ezt a mértéket egyszerű pontosságnak fogom nevezni.

A mintafelismerő és bináris osztályozó algoritmusoknál gyakran felhasznált mérték még a precizitás (angolul precision) és a felidézés (angolul recall). A precizitás annak mértéke, hogy mennyire „hasznosak” az eredmények, a felidézés pedig azt mondja meg, hogy mennyire teljes az eredményünk. Képletük:

$$Precizitás = \frac{Valós\ pozitív}{Valós\ pozitív + Hamis\ pozitív} \tag{5.1}$$

$$Felidézés = \frac{Valós\ pozitív}{Valós\ pozitív + Hamis\ negatív}. \tag{5.2}$$

A képletekben használt hamis negatív és hamis pozitív fogalmak az eredményünk jelen értelmezésére nem értelmezhetőek, hiszen csupán annyit tudunk, hogy egy összerendelés vagy helyes lett vagy helytelen. Viszont, ha úgy tekintünk az adatra, hogy a célja csupán

egy író cikkeinek megtalálása, akkor tudjuk értelmezni ezeket a mértékeket és ki tudjuk számolni őket. Az algoritmusunknak a célja az volt, hogy a cikkekről az általa ismert írók közül megadja, hogy ki írta, emiatt a precizitás és felidézés értékeket kiszámoljuk az összes íróra értelmezve és átlagoljuk az eredményeket.

Annak érdekében, hogy könnyebben összehasonlíthatóak legyenek az algoritmusok a precizitás és felidézés szempontjából be tudunk vezetni egy származtatott mértéket, az f-mérték (angolul f-measure) képlete:

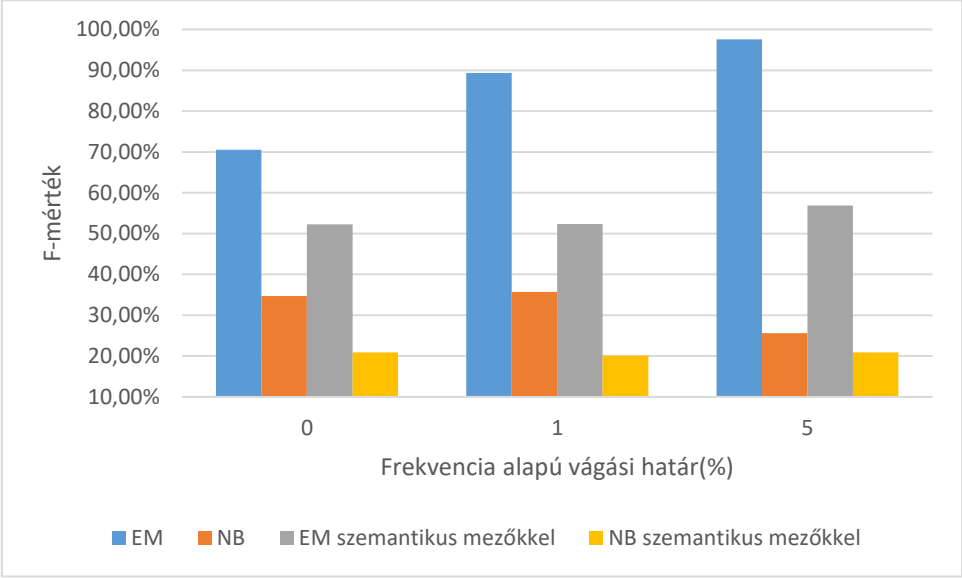
$$F = 2 * \frac{Precizitás * Felidézés}{Precizitás + Felidézés} \tag{5.3}$$

5.3 Futtatási eredmények

Először általánosan megvizsgáljuk, hogy a különböző konfigurációk milyen hatással vannak az algoritmusok eredményeire, majd külön megnézünk néhány konfigurációs beállítást, amik érdekes eredményeket adtak.

5.3.1 Felső vágási határ

Az 5.1. ábra által mutatott eredményekhez tartozó egyéb konfigurációs értékek: 2000 szó felhasználása a lexikonból, 1 cikk/író méretű tanítóhalmaz.



5.1. ábra F-mérték frekvenciavágás függvényében

Láthatjuk, hogy a szemantikus mezőket felhasználó eredményeket kis mértékben befolyásolta a dimenziócsökkentés, ezzel a megfigyeléssel később fogunk foglalkozni. Két fontosabb észrevétel az ábráról még: Az EM algoritmusra nagyon pozitív hatással

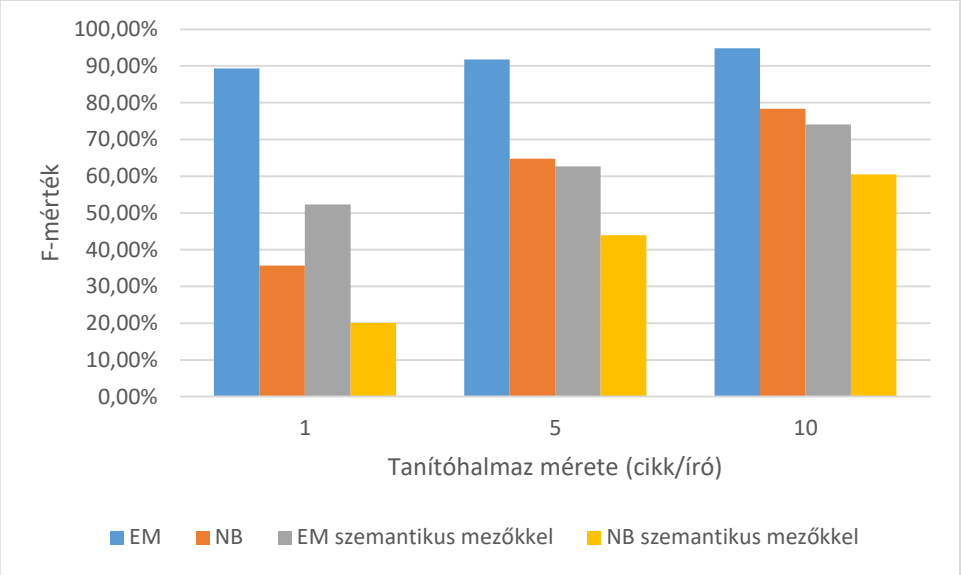
van a gyakori szavak kiszűrére, leginkább az első 1%. A másik pedig, hogy az NB algoritmusnak csökken a pontossága az 5%-os vágási határnál.

Először vizsgáljuk meg részletesebben az utóbbi jelenséget. A Naiv Bayes algoritmus csak a tanítóhalmazt használja fel arra, hogy megismerje az írókra jellemző szóhasználatot. Emiatt szavak, melyek nem jelennek meg egy íróhoz a tanítóhalmazban, nagyon alacsony valószínűséggel fognak szerepelni a modellünkben. Következetesen minél több gyakori szót hagyunk el a lexikonunkból, annál ritkábbak lesznek a modellünkben az előforduló ismert szavak, így az algoritmusunknak kevesebb információ áll rendelkezésére, amiből a hasonlóságról tudna dönteni. Konkrét értékeket megtekintve az 1%-os vágással átlagosan 67 szavat tartalmazott a modell írónként (ha a tanítóhalmaz 1 cikket tartalmaz írónkén), ezzel szemben, ha a gyakori szavak 5%-át szűrjük ki, akkor átlagosan 15 szavat tartalmaz a modellünk.

Az EM algoritmus esetén az előző probléma nem áll fenn ennyire szélsőségesen, hiszen a modellünket folyamatosan újraszámoljuk a felcímkézetlen adatokat felhasználva. Így annyira nem is meglepő, hogy javuló tendenciát mutat az algoritmusunk. Amit viszont meg kell figyelnünk az, hogy hogy áll össze az EM pontosságának javulása. A 70%-os f-mérték úgy áll össze a 0%-os vágásnál, hogy 96%-os precizitás és 55%-os felidézés. Az összerendeléseket jobban megvizsgálva láthatjuk, hogy a magas precizitást úgy érte el az algoritmus, hogy egy író kivételével mindenkinek közel 100%-os precizitása van, ez az egy író pedig pontosan az, akinek a gyűjteményünkben rengeteg cikke van. Így el tudott érni nagy precizitást úgy, hogy a többi íróhoz több cikket rendelt, mint amennyi tényleg hozzájuk tartozik, ez viszont ahhoz vezet, hogy a felidézés értékek nagyon alacsonyak voltak (legrosszabb ebben az esetben 6%-os volt).

5.3.2 Tanítóhalmaz méretének konfigurálása

Az alábbi ábra által mutatott értékek konfigurációja: maximum 2000 szót használunk fel a lexikonból és 1%-os vágást alkalmazunk.

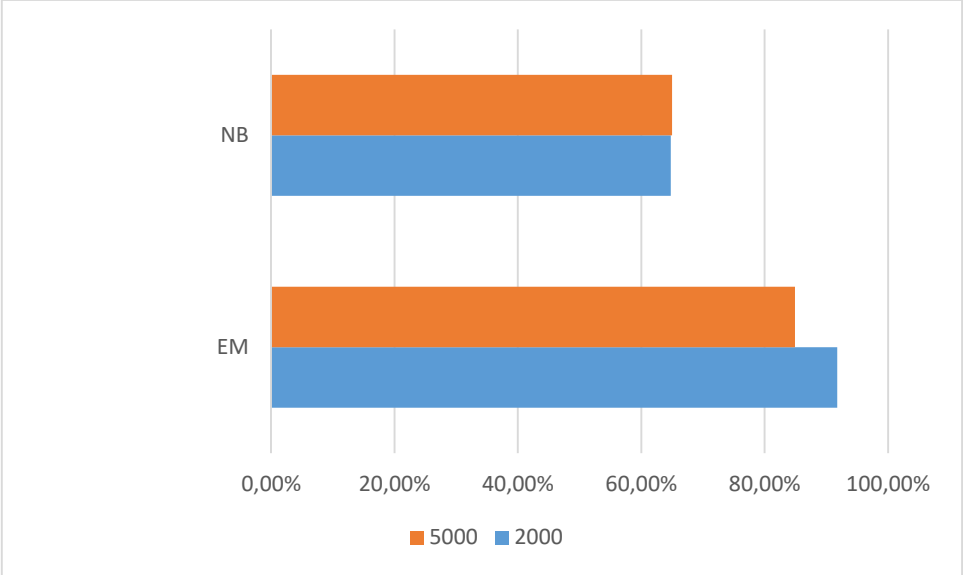


5.2. ábra F-mérték értékek a tanítóhalmaz méretének függvényében

A tanítóhalmaz méretének növelésével nem meglepően az összes algoritmus pontossága növekszik. Érdemes megfigyelni, hogy a Naiv Bayes algoritmus pontossága milyen mértékben növekszik a tanítóhalmaz méretével. Ezt a jelenséget az előző pontban részletezett megfigyelésnek tudhatjuk be, miszerint a Naiv Bayes algoritmus csak a tanítóhalmazból nyer ki információt a modellje felépítéséhez, így szükséges neki a nagyobb tanítóhalmaz.

5.3.3 Felhasznált szavak mennyisége

Az alábbi ábrában használt egyéb konfigurációk: 1%-os vágás és 5 cikk/író méretű tanítóhalmaz.



5.3. ábra F-mérték értékei a felhasznált szavak mennyiségének függvényében

A szavak számának növelésének nincs értelme a szemantikus mezők használatakor, hiszen a technika lényege, hogy lecsökkentjük a szavak terét egy már kezelhető mennyiségre.

A Naiv Bayes algoritmusra nem számottevően, de konzisztensen pozitív hatással van a felhasznált szavak növelése. Hiszen a szavak számának növelésével növeljük a tanítóhalmazból kinyerhető információ mennyiségét, de figyelembe kell venni, hogy az újonnan bevett szavak mind ritkábban fordulnak elő, mint az eddig is felhasznált szavak, így nem fogják nagymértékben befolyásolni az eredményeket.

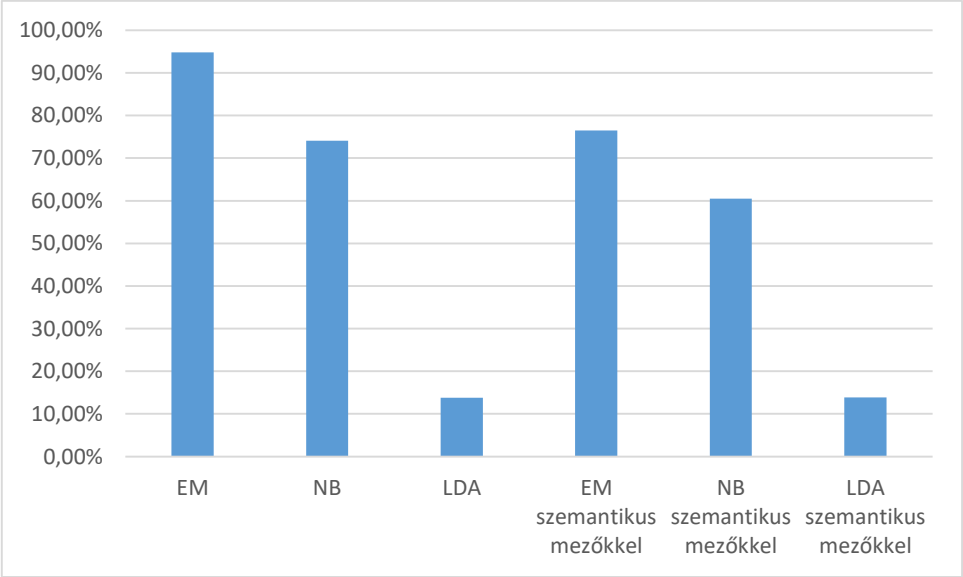
Az EM algoritmus esetén, érdekes módon, ront a pontosságunkon a felhasznált szavak számának növelése. Az eredmények részletesebb vizsgálatából kiszűrhető, hogy míg a precizitás értékek a két konfigurációra nagyjából megegyeznek, a felidézés értékeket komolyan befolyásolja a felhasznált szavak száma. Ennek lehetséges magyarázata, hogy az író, melynek sok cikkét tartalmazza, a gyűjteményünk természetesen nagyobb súllyal van jelen a modellünkben, mint a többi író. Emiatt a leggyakoribb szavak nagy része szintén ehhez az íróhoz fognak tartozni, ezzel növelne annak esélyét, hogy egy cikket hibásan ehhez az íróhoz rendeljünk.

5.3.4 Szemantikus mezők alkalmazása

Az 5.1. ábra és az 5.2. ábra alapján láthatjuk, hogy a szemantikus mezők alkalmazása nagymértékben csökkentette az algoritmusaink pontosságát. A probléma a szemantikus mezők esetében egy nagyon egyszerű eset. Amikor leképezzük a dokumentumainkat a szemantikus mezőkre, rengeteg információt veszítünk az adatunkból, viszont abban bízunk, hogy az az információ vész el, mely csak zajnak számított és kihangsúlyozódik a dokumentumok közti különbség. Ezen feltételezések sok esetben helyesek lehetnek, viszont online újságcikkek esetén, melyek nagyjából azonos időszakból származnak (fél év hosszú időtartamból gyűjtöttem adatot), akkor tematikájuk sokszor hasonló lesz, így a szemantikus mezők használata pont a feltételezésünk ellenkezőjét fogja elérni és a megkülönböztetéshez szükséges információt veszítjük el.

5.3.5 Általános összehasonlítás

Ebben a szekcióban azokra a tulajdonságokra térek ki, melyek segítségével dönteni tudunk, hogy saját alkalmazásunkban melyik algoritmust lenne érdemes használni. Az alábbi ábra azt kívánatos reprezentálni, hogy a tanítóhalmaz jelenléte mennyire fontos ahhoz, hogy a dokumentumokat író alapján osztályozzuk.



5.4. ábra Algoritmusok pontossága egymáshoz viszonyítva

Az EM algoritmus láthatóan sokkal nagyobb pontosságot tud biztosítani az osztályozásunkban, viszont nagy hátránya, hogy ezen eredményhez csak több iteráción keresztül tud eljutni, és ha a konvergenciát nem tudjuk meghatározni a számábrázolás miatt, akkor megbecsülni se tudjuk a szükséges iterációk számát, csak próbálkozással.

Az NB algoritmus azt az előnyt nyújtja, hogy tanulóhalmazból hamar felépít egy modellt, viszont nagyon rosszul működik akkor, ha kisméretű a tanítóhalmazunk.

Az LDA algoritmust, mivel nem osztályozó algoritmus, a gyűjteményünk témáinak meghatározására tudjuk alkalmazni, írók meghatározására csak kevéssel bizonyult jobbnak a véletlenszerű tippelésnél.

A szemantikus mezőkkel kibővített algoritmusok használatára az előző pontban mutattunk egy hátrányos alkalmazást. Alkalmazásuk előnyhöz vezethet, ha az íróink szóhasználatuk témájában is eltérnek, ilyen felhasználás lehet: könyvek, versek osztályozása.

6 Konklúzió

Az írók szóhasználatának megtanulása, bár nem triviális feladat, meglepő módon magas pontossággal elérhető. Ebben a dokumentumban megvizsgáltuk, hogy a precizitást és a felidézést, hogyan befolyásolja a frekvencia alapú dimenziócsökkentés és a tanítóhalmaz mérete. Példák alapján láthattuk, hogy a gyakori szavak kiszűrése kezdetben komoly hatással van az eredményünk pontosságára, viszont ahogy egyre ritkábban előforduló szavakat veszünk bele a lexikonunkba a dokumentumok közötti hasonlóság mértékéből veszítünk. Az Expectation-Maximization algoritmus segítségével képesek voltunk a kisméretű tanítóhalmazból fakadó információhiányt kiküszöbölni azzal, hogy a felcímkézetlen dokumentumokat is belevettük a modellünkbe, így javítani tudtuk a pontosságon a ritkább szavak felhasználásával is. Az eredmények részletesebb megvizsgálásával láthattuk, hogy mily módon befolyásolja a modellünk felépítését az osztályoknak a túlreprezentálása a többi osztályhoz képest. Láttuk, hogy a szemantikus mezők felhasználása azzal a hátulütővel járhat, hogy az osztályaink reprezentációit közelebb hozzuk egymáshoz, ahelyett hogy az elvárt megkülönböztetés kihangsúlyozását érnénk el a szövegek témájának hasonlósága miatt.

7 Továbbfejlesztési lehetőségek

Jelen dokumentumunkban felhasznált modellben és algoritmusokban feltételeztük, hogy az összes dokumentum az általunk ismert osztályok egyikébe tartozik. A valószínűség alapú modell alkalmazásának nagy előnye, hogy láthatjuk azon értékeket is melyek alapján az algoritmus dönt, hogy mely osztályba sorolja az adott dokumentumot. Ezen értékek megfigyelésével tudnánk olyan döntéseket hozni, hogy egy dokumentum nagy valószínűséggel egyik írónkhoz sem tartozik.

Másik irány melyben tovább tudjuk fejleszteni az algoritmusunkat, hogy a dokumentumok hosszát is vegye figyelembe, mint osztályozási szempont. Általában elmondható, hogy egy író milyen hosszú cikkeket/könyvet/verset ír, így ha a szavak mellett a dokumentumok hosszát is felhasználjuk a modell felépítéséhez, akkor várhatóan nagyobb pontosságot tudunk elérni.

Utolsó fejlesztési lehetőség, mely az algoritmusok implementálása során felkeltette érdeklődésemet a többrétegű osztályozás. A dokumentumokat nem rögtön az írók szerint osztályozzuk, hanem először például témakör szerint, így témakörökön belül már kevesebb osztályunk lesz és a témakörökön belül megváltozik a felhasznált szavak eloszlása, így jobban ki tudjuk nyerni az írók közötti különbségeket. Ezt akkor lehet jól kihasználni, ha már sok írónak az osztályozását kell elvégeznie az algoritmusnak, így sok egymáshoz hasonló osztály alakul ki. Hátránya, hogy kétszer kell lefutnia az osztályozó algoritmusunknak, mely az Expectation-Maximization algoritmus esetén meglehetősen hosszú időt is igénybe vehet.

Irodalomjegyzék

[1] Andrew McCallum, Kamal Nigam: *A comparison of event models for naive Bayes text classification*. Learning for Text Categorization: Papers from the AAAI Workshop, AAAI Press, Tech. rep. WS-98-05, 1998, pp. 41–48

[2] Bela A. Frigyik; Amol Kapila; Maya R. Gupta: *Introduction to the Dirichlet Distribution and Related Processes*, 2010, UWEETR-2010-0006

[3] Bohdan Pavlyshenko: *Clustering of Authors' Texts of English Fiction in the Vector Space of Semantic Fields*, Cybernetics and information technologies, Vol. 14, 2014, 25-36

[4] C. J.C. Burges: *A tutorial on support vector machines for pattern recognition*. Data Mining and Knowledge Discovery, 1998, 2:121–167

[5] Charu C. Aggarwal, ChengXiang Zhai: *A survey of text clustering algorithms*, Mining Text Data, 2012, 77-128

[6] Chih-Wei Hsu, Chih-Jen Lin: *A comparison of methods for multiclass support vector machines*, IEEE Transactions on Neural Networks, Vol. 13, Issue 2, 2002, pp. 415-425

[7] David M. Blei, Andrew Y. Ng, Michael I. Jordan: *Latent Dirichlet Allocation*, 2003

[8] Fellbaum, C. WordNet. An Electronic Lexical Database. Cambridge, MA, MIT Press, 1998

[9] Kamal Nigam, Andrew McCallum, Tom M. Mitchell: *Semi-Supervised Text Classification Using EM*, 2006

[10] Siwei Lai, Liheng Xu, Kang Liu, Jun Zhao: *Recurrent Convolutional Neural Networks for Text Classification*, Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015, pp. 2268-2273

[11] Snowball Stemmer: http://snowballstem.org/ (revision 14:00, 07 December 2016)

[12] Wikipedia: *Expectation–maximization algorithm*, http://en.wikipedia.org/wiki/Expectation%E2%80%93maximization_algorithm (revision 13:40, 07 December 2016)

[13] Wikipedia: Hungarian algorithm, http://en.wikipedia.org/wiki/Hungarian_algorithm (revision 14:00, 07 December 2016)

[14] Wikipedia: *k-Nearest Neighbors algorithm*, http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm (revision 13:15, 07 December 2016)