Általános információk

A diplomaterv szerkezete:

1. Diplomaterv feladatkiírás
2. Címoldal
3. Tartalomjegyzék
4. A diplomatervező nyilatkozata az önálló munkáról és az elektronikus adatok kezeléséről
5. Tartalmi összefoglaló magyarul és angolul
6. Bevezetés: a feladat értelmezése, a tervezés célja, a feladat indokoltsága, a diplomaterv felépítésének rövid összefoglalása
7. A feladatkiírás pontosítása és részletes elemzése
8. Előzmények (irodalomkutatás, hasonló alkotások), az ezekből levonható következtetések
9. A tervezés részletes leírása, a döntési lehetőségek értékelése és a választott megoldások indoklása
10. A megtervezett műszaki alkotás értékelése, kritikai elemzése, továbbfejlesztési lehetőségek
11. Esetleges köszönetnyilvánítások
12. Részletesés pontos irodalomjegyzék
13. Függelék(ek)

Felhasználható a következő oldaltól kezdődő Diplomaterv sablon dokumentum tartalma. Ügyeljen a konzulens nevét és a beadás évét jelölő szövegdobozokra, mert azokra külön ki kell adni a frissítést. A mezők tartalma a sablonban a dokumentum adatlapja alapján automatikusan kerül kitöltésre.

A diplomaterv szabványos méretű A4-es lapokra kerüljön. Az oldalak tükörmargóval készüljenek (mindenhol 2.5cm, baloldalon 1cm-es kötéssel). Az alapértelmezett betűkészlet a 12 pontos Times New Roman, másfeles sorközzel.

Minden oldalon - az első négy szerkezeti elem kivételével - szerepelnie kell az oldalszámnak.

A fejezeteket decimális beosztással kell ellátni. Az ábrákat a megfelelő helyre be kell illeszteni, fejezetenként decimális számmal és kifejező címmel kell ellátni. A fejezeteket decimális aláosztással számozzuk, maximálisan 3 aláosztás mélységben (pl. 2.3.4.1.). Az ábrákat, táblázatokat és képleteket célszerű fejezetenként külön számozni (pl. 2.4. ábra, 4.2 táblázat vagy képletnél (3.2)). A fejezetcímeket igazítsuk balra, a normál szövegnél viszont használjunk sorkiegyenlítést. Az ábrákat, táblázatokat és a hozzájuk tartozó címet igazítsuk középre. A cím a jelölt rész alatt helyezkedjen el.

A képeket lehetőleg rajzoló programmal készítsék el, az egyenleteket egyenlet-szerkesztő segítségével írják le.

Az irodalomjegyzék szövegközi hivatkozása történhet a Harvard-rendszerben (a szerző és az évszám megadásával) vagy sorszámozva. A teljes lista névsor szerinti sorrendben a szöveg végén szerepeljen (sorszámozott irodalmi hivatkozások esetén hivatkozási sorrendben). A szakirodalmi források címeit azonban mindig az eredeti nyelven kell megadni, esetleg zárójelben a fordítással. A listában szereplő valamennyi publikációra hivatkozni kell a szövegben. Minden publikáció a szerzők után a következő adatok szerepelnek: folyóirat cikkeknél a pontos cím, a folyóirat címe, évfolyam, szám, oldalszám tól-ig. A folyóirat címeket csak akkor rövidítsük, ha azok nagyon közismertek vagy nagyon hosszúak. Internet hivatkozások megadásakor fontos, hogy az elérési út előtt megadjuk az oldal tulajdonosát és tartalmát (mivel a link egy idő után akár elérhetetlenné is válhat), valamint az elérés időpontját.

Fontos:

* a szakdolgozat készítő/diplomatervező nyilatkozata (a jelen sablonban szereplő szövegtartalommal) kötelező előírás Karunkon, ennek hiányában a szakdolgozat/diplomaterv nem bírálható és nem védhető!
* mind a dolgozat, mind a melléklet maximálisan 15 MB méretű lehet!

Jó munkát, sikeres szakdolgozat készítést ill. diplomatervezést kívánunk!

FELADATKIÍRÁS

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.



Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Konzulens

BUDAPEST, 2016

Tartalomjegyzék

[Összefoglaló 6](#_Toc468494427)

[Abstract 7](#_Toc468494428)

[1 Bevezetés 8](#_Toc468494429)

[1.1 Formázási tudnivalók 8](#_Toc468494430)

[1.1.1 Címsorok 8](#_Toc468494431)

[1.1.2 Képek 8](#_Toc468494432)

[1.1.3 Kódrészletek 8](#_Toc468494433)

[1.1.4 Irodalomjegyzék 8](#_Toc468494434)

[2 Bevezetés 10](#_Toc468494435)

[3 Feladat értelmezés és kifejtés 11](#_Toc468494436)

[4 Szövegbányászat 12](#_Toc468494437)

[4.1 Előfeldolgozás, modellkészítés 12](#_Toc468494438)

[4.2 Csoportosítás 13](#_Toc468494439)

[4.2.1 Valószínűség alapú szöveg csoportosítás és a topic-model 14](#_Toc468494440)

[4.2.2 Latent Dirichlet Allocation algoritmus 14](#_Toc468494441)

[4.2.3 Expectation-Maximization algoritmus 15](#_Toc468494442)

[4.2.4 EM alkalmazása naiv Bayes osztályozás kibővítésére 16](#_Toc468494443)

[4.2.5 Felügyelt szöveg osztályozás 17](#_Toc468494444)

[4.2.6 EM alkalmazása a naiv Bayes algoritmusban 18](#_Toc468494445)

[4.3 Szemantikus mezők 18](#_Toc468494446)

[5 Feladat implementálása 20](#_Toc468494447)

[5.1 Adatgyűjtés 20](#_Toc468494448)

[5.2 Java implementáció 20](#_Toc468494449)

[6 Algoritmus konfigurálása és eredmények 22](#_Toc468494450)

[7 Továbbfejlesztési lehetőségek 23](#_Toc468494451)

[8 Konklúzió 24](#_Toc468494452)

[9 Utolsó simítások 25](#_Toc468494453)

[Irodalomjegyzék 26](#_Toc468494454)

[Függelék 27](#_Toc468494455)

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. 03.

...…………………………………………….

Összefoglaló

Ide jön a ½-1 oldalas magyar nyelvű összefoglaló, melynek szövege a Diplomaterv Portálra külön is feltöltésre kerül.

Abstract

Ide jön a ½-1 oldalas angol nyelvű összefoglaló, amelynek szövege a Diplomaterv Portálra külön is feltöltésre kerül.

# Bevezetés

A következő fejezet pár példán keresztül bemutatja a diplomatervekben és szakdolgozatokban szokásosan előkerülő formázások megvalósítását.

## Formázási tudnivalók

A dokumentum folyószövegéhez használjuk a **Normál** (angol Word esetén Normal) stílust.

### Címsorok

A fejezetcímek esetén a **Címsor 1-4** (Heading 1-4) stílusokat használjuk.

### Képek

A képhez használjuk a **Kép** stílust.

Képaláírást a képen jobb gombbal kattintva a Képaláírás beszúrása… opcióval adhatunk hozzá, így az automatikusan **Képaláírás** (Caption) stílusú lesz.



1.1. ábra: Példa képaláírásra

### Kódrészletek

Kódrészletek beillesztése esetén használjuk a **Kód** stílust.

using System;

namespace MyApp

{

class Program

{

static void Main( string[] args )

{

Console.WriteLine( "Szia Világ!" );

}

}

}

### Irodalomjegyzék

Az Irodalomjegyzékben szereplő hivatkozásokat **Irodalomjegyzék sor** stílussal formázzuk, a címüket pedig **Irodalomjegyzék forrás** stílussal emeljük ki.

A szövegbe a hivatkozásokat a Kereszthivatkozás beszúrása (Insert cross-reference) funkcióval helyezzük el (példa egy így beszúrt hivatkozásra: [1]), így azok automatikusan frissülnek a hivatkozások átrendezésekor.

# 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.

# Feladat értelmezés és kifejtés

# Szövegbányászat

## Előfeldolgozás, 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 előfeldolgozás 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.

## 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.

### Valószínűség alapú szöveg csoportosítás és a topic-model

Egy elterjedt változata a valószínűség alapú szöveg csoportosításnak a topic-model-ezés. A topic-model algoritmus 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 *Di* és a témakörök megadásával , annak a valószínűsége, hogy az i. dokumentum a j. témakörbe tartozik a 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 a gyűjteményben előforduló szavak. Annak a valószínűsége, hogy az *tl* szó előfordul a témakörben a 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 (PLSI) és a Latent Dirichlet Allocation (LDA). Ebben a dokumentumban csak az LDA algoritmussal fogunk foglalkozni.

### 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, pl poession).
* 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.



4.1. ábra LDA algoritmus Bayes modellje

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

### 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.

## Osztályozás

### EM alkalmazása naiv Bayes osztályozás kibővítésére

A LDA algoritmus hátránya, hogy nem tartalmaz megerősítéses tanulást, ennek orvoslására felhasználhatjuk az EM algoritmust az LDA modelljére. Feltételezzük, hogy a dokumentumok az LDA algoritmus generatív modellje szerint jöttek létre. Legyen M az osztályok száma, a dokumentumok száma, minden dokumentum pedig a dokumentum szavainak száma, a valószínűségi modellt jelölje . Egy dokumentumot, -t, úgy hozzuk létre, hogy először generálunk egy osztályt neki, jelöljük ezt -nek( eleme {1..M} halmaznak), a valósínűségét -val jelöljük. A dokumentumok generáljuk a kiválasztott osztály eloszlásával, ennek valószínűsége . Tehát annak a valószínűsége, hogy a dokumentumot generáljuk a jelenlegi modellünkkel:

. (4.1)

Minden dokumentum egy osztályhoz tartozik, ezt jelöljük -vel, tehát ha az dokumentum a cj osztályból lett generálva azt mondjuk, hogy .

A szózsákmodellből következően a dokumentumok szóelőfordulások vektora. Legyen a wt szó előfordulásának száma az dokumentumban. Ezekből ki tudjuk bontani az előző képlet második kifejezését:

(4.2)

Ez az összefüggés tartalmazza az alap naiv Bayes csoportosítás 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.

### Felügyelt szöveg osztályozás

A naiv Bayes osztályozás megtanulása egy felcímkézett dokumentumhalmazból abból áll, hogy a modell paramétereit megbecsüljük. A paraméter megbecsültjét -vel jelöljük. A naiv Bayes algoritmus maximum a posterior (MAP) becslést alkalmaz, vagyis keresi a . Ez a legvalószínűbb modell megadva a tanítóhalmazt és egy priort.

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

(4.3)

Ahol és az értékek pedig pozitív konstansok. Az alkalmazásunkban az összes értéket 2-nek választottuk, ami egy egyenletes eloszlású priorhoz vezet. A szó előfordulási valószínűségek becsült értéke, Lagrange simítást alkamazva:

(4.4)

ahol értékét a dokumentum osztályából számoljuk, 1, ha és 0 egyébként. Az osztály előfordulási valószínűségek pedig:

. (4.5)

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:

(4.6)

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

### EM alkalmazása a naiv Bayes algoritmusban

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 -nak, esetünkben , ami ekvivalens azzal, mintha ennek a kifejezésnek a logaritmusát maximalizálnánk. Modellünk szerint ennek képlete:

. (4.7)

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 mezőben. Az M lépés maximalizálja a modell valószínűségét az E lépésben kiszámolt hiányzó értékek felhasználva, mintha valódi értékek lennének.

## 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.

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 lexikonját használtam fel.

# Feladat implementálása

## 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.

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 stopszószűrés és a frekvencia alapú dimenziócsökkentés teljesen kiszűrt. Ezért úgy döntöttem, hogy kiszűröm az összes 500 karakternél rövidebb cikkeket.

## 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 értékeinek kiszámolása helyett a dokumentumok adott osztályra szűrésével, rengeteg 1-gyel való szorzást és 0 értékek összeadását kiváltottam.

<még valami, hogy ne legyen üres oldal>

# Algoritmus konfigurálása és eredmények

<paraméterek jelentősége, megfontolások a módosításaik mögött>

<a gyakori szavak elhagyásának jelentősége, felhasznált algoritmusonként értelmezve az eredményt>

<a felhasznált szavak számának hatása>

<training set méretének befolyása az eredményre>

# Továbbfejlesztési lehetőségek

<ami itt eddig eszembe jutott az a modell kibővítése azzal, hogy a dokumentumok hosszát is figyelembe vegye. Erre ha jól tudom az a megszokott eljárás, hogy valamilyen valószínűségi eloszlásúnak tekinti és ezt a leállási feltételbe és a hasonlóságba is beleveszi. Viszont ez az információ akár negatívan is hathat egyes alkalmazásokban, ha nagyon véletlenszerű a dokumentumok hossza, de a cikkeknél szerintem segítene.>

# Konklúzió

<a szerző kereséséhez mik a legfontosabb információk egy szövegben, melyek jelenlétét ki kell emelni ahhoz, hogy minél pontosabb osztályozást tudjunk elérni>

# Utolsó simítások

Miután elkészültünk a dokumentációval, ne felejtsük el a következő lépéseket:

* Kereszthivatkozások frissítése: miután kijelöltük a teljes szöveget (Ctrl+A), nyomjuk meg az F9 billentyűt, és a Word frissíti az összes kereszthivatkozást. Ilyenkor ellenőrizzük, hogy nem jelent-e meg valahol a "Hiba! A könyvjelző nem létezik." szöveg.
* Dokumentum tulajdonságok megadása: a dokumentumhoz tartozó meta adatok kitöltése (szerző, cím, kulcsszavak stb.). Erre való a Dokumentum tulajdonságai panel, mely a Fájl / Információ / Tulajdonságok / Dokumentumpanel megjelenítése úton érhető el.
* Kinézet ellenőrzése PDF-ben: a legjobb teszt a végén, ha PDF-et készítünk a dokumentumból, és azt leellenőrizzük.

Irodalomjegyzék

1. Levendovszky, J., Jereb, L., Elek, Zs., Vesztergombi, Gy.: Adaptive statistical algorithms in network reliability analysis, Performance Evaluation - Elsevier, Vol. 48, 2002, pp. 225-236
2. National Istruments: LabVIEW grafikus fejlesztői környezet leírása, <http://www.ni.com/> (2010. nov.)
3. Fowler, M.: UML Distilled, 3rd edition, ISBN 0-321-19368-7, Addison-Wesley, 2004
4. Wikipedia: Evaluation strategy, <http://en.wikipedia.org/wiki/Evaluation_strategy> (revision 18:11, 31 July 2012)