



M Ű E G Y E T E M 1 7 8 2

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Távközlési és Médiainformatikai Tanszék

Kiss Balázs

DOKUMENTUM OSZTÁLYOZÁS ADATBÁNYÁSZATI MÓDSZEREKKEL

KONZULENS

Nagy Gábor

BUDAPEST, 2015

Tartalomjegyzék

Összefoglaló	4
Abstract.....	5
1 Bevezetés	6
2 Adat- és szövegbányászat	8
2.1 Bevezetés	8
2.2 Adatbányászat	8
2.2.1 Adatbányászat lépései	9
2.2.2 Adatbányászati módszerek: osztályozás	11
2.3 Szövegbányászat	12
2.3.1 Előfeldolgozás	13
3 Felhasznált webes technológiák.....	16
3.1 Python	16
3.2 Webalkalmazások	17
3.3 Django.....	17
3.4 Beautiful Soup	19
3.5 Gensim	20
3.5.1 Gensim funkciók.....	21
4 Tervezési fázis	22
5 Szoftver implementálása	24
5.1 Webes megjelenítő rész	24
5.2 Alkalmazás architektúra	24
5.3 Parser modul	26
5.4 Szövegbányászati modul.....	27
6 Eredmények.....	30
7 További lehetőségek.....	31
Irodalomjegyzék.....	32

HALLGATÓI NYILATKOZAT

Alulírott **Kiss Balázs**, szigorló hallgató kijelentem, hogy ezt a diplomatervet 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, 2015. 12. 11.

.....
Kiss Balázs

Összefoglaló

Feladatom során egy webes alkalmazást alapjait készítettem el, mely képes internetes hírportálok tartalmát feldolgozni.

A webalkalmazás adat- és szövegbányászati eszközöket felhasználva, alkalmas az index.hu portál RSS hírfolyamának felolvasására, tárolására. A felhasználó által megadott szöveget képes a hírfolyam korábbi cikk kategóriáinak egyikébe besorolni, kizárólag a szöveg tartalma alapján.

A feladat megvalósításának érdekében megismerkedtem a webalkalmazások működési elvével. Elsajátítottam a Python nyelv ismeretét, megismertem a Django webes keretrendszer használatát. Korábbi tanulmányaim alatt elért eredményeimet felhasználva automatikus adatbányászati folyamatokat implementáltam, melyeket integráltam a webalkalmazásba.

A webes felületen keresztül lehetősége van a felhasználónak irányítani a hírportál felolvasását, illetve képes saját szöveg megadására is. A keretrendszerbe integrált adat- és szövegbányászati eljárások, web-crawler technológiák és megjelenítő eszközök segítségével, webes felületen, külső beavatkozás nélkül képes az adatok feldolgozására, eredmények közzétételére.

Abstract

In my work, I created a web application, which is able to analyse the text data of a web news portal.

The applications is able to parse and store the contents of the index.hu site, using data- and text mining tools. It is able to categorize the text, given by the user, according to the learnt news stream, relying only on the given texts.

To be able to implement this, I had to learn about the web application technology. I acquired the knowledge necessary about the Python programming language and studied the usage of the Django web framework. I used the results of my earlier works to implement an automated datamining process, which I later on integrated into the web application.

Through the web interface, the user can control the parsing of the news site and can input their own text. Through the help of the integrated data- and text mining procedures, web crawler technologies and editor interfaces in the web framework, the user can achieve the processing of the given data and get results, without outside help.

1 Bevezetés

A diplomatervezésem első félévében arra a kérdésre kerestem a választ: megvalósítható-e egy olyan alkalmazás, amely képes webes portálok cikkeinek automatikus kategorizálására?

Egy ilyen eszköz hasznos kiegészítője lehet egy hírportál íróinak, esetlegesen ellenőrzés gyanánt felhívhatja a figyelmet arra, ha egy cikk jobba beleillik egy másik kategóriába. A felhasználók ízléséhez pontosan illeszkedő kategóriákat gyakran nehéz körvonalazni: egy ilyen eljárás segítségével lehet az újságírónak az írásaik pontos elhelyezéséhez nem egyértelmű esetekben.

Önálló kutatásaim keretein belül már végeztem kimutatást a magyar nyelvű szövegbányászat ilyen jellegű alkalmazásainak eredményességéről[1] Ekkor első sorban a cikkeket összefoglaló kulcsszavak, azaz címkék meghatározásának lehetőségeit vizsgáltam.

Eredményeim arra mutattak, hogy a magyar nyelvű szövegből kinyerhetőek azon fontos kulcsszavak, amelyek a legjobban jellemeznék egy adott cikket. Első sorban a szövegbányászat webes környezetben történő alkalmazása, és a későbbiekben felhasznált keretrendszer bevezető megismerése jelentette a kihívást.

Önálló laboratórium tárgyam során egy olyan, a mostani feladatomat megalapozó alkalmazást készítettem el, amely segítségével cikkek közötti kapcsolatok alapján ajánlásokat tett egy új írás címkéire[2]. Azt kerestem, hogy az új szöveg mely szavai a legjellemzőbbek az adott bejegyzésre és hogy adatbányászati értelemben mely más dokumentumok tartalma hasonlít a legjobban az új íráshoz. Így, a két kutatás eredményének uniója adta a megoldás halmazát a cikkek címkéinek.

Diplomamunkámnak egy olyan komplex webes alkalmazás elkészítését vállaltam, amely ezen eredményekre épülve képes automatikusan felolvasni egy hírportál tartalmát, majd pedig az azon definiált kategóriákba automatikusan besorolja az újonnan beérkező híreket. A feladat kihívása abban rejlik, hogy megismerjem az ehhez szükséges web-crawler technológiákat[3], és hogy megalkossam azt a dokumentum feldolgozási- és adatbányászati technológiát, amely segítségével automatikusan meghatározható az új írások kategóriája.

A munkám során tovább mélyítettem a felhasznált technológiák ismeretét, bővítettem a korábbi szoftveremet. Az alapgondolat egy példa-implementációjával történő megalkotása során alapot teremtettem arra, hogy egy komplex eszközt fejlesszek le a munkám második felében.



1. ábra Korábban elkészített címkéző alkalmazásom

2 Adat- és szövegbányászat

2.1 Bevezetés

Az adatbányászat[4] modern korunk adatfeldolgozási módszereinek egyik legújabb, és legfontosabb területe. A témáról korábbi önálló laboratóriumi beszámolóimban és szakdolgozatomban[5] írtam részletesebben. Itt csak a jelen feladatom megoldásához szükséges alapfogalmakkal, és felhasznált témakörökre térek ki.

Ahogy Abonyi János fogalmaz[4]: „Az adatbányászat egy olyan döntéstámogatást szolgáló folyamat, mely érvényes, hasznos, és előzőleg nem ismert, tömör információt tár fel nagy adathalmazból”.

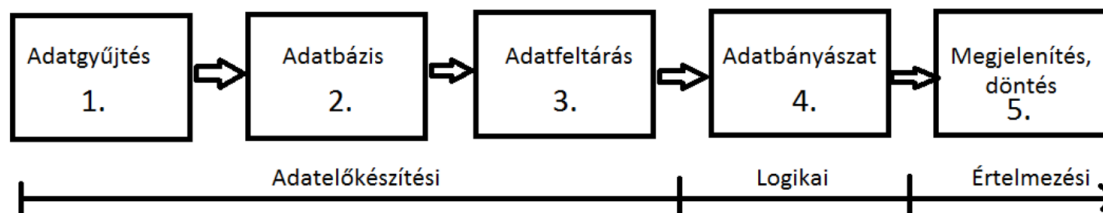
A szövegbányászat[6] szorosan kapcsolódik az adatbányászathoz. Itt azonban az adathalmazunk strukturálatlan: nevéből adódóan általános jellegű, értelmes szövegek gépi feldolgozására vonatkozik. A szövegbányászat feladata, hogy előkészítse a már említett rendezetlen adathalmazt későbbi adatbányászati elemzésekhez. A kihívás az emberi nyelvtan összefüggéseinket matematikai módszerekkel történő feldolgozása, ezen információk rendezése és kiemelése a későbbi kutatásokhoz.

2.2 Adatbányászat

Az adatbányászat célja tehát, hogy egy strukturált adathalmazon matematikai eszközökkel keressünk összefüggéseket, mintázatokat. A feltárt összefüggések értelmezése különösképp fontos a hasznos információ kinyeréséhez. Modern korunk számítási sebesség növekedésének köszönhetően rengeteg új eszköz és lehetőségünk nyílik erre.

2.2.1 Adatbányászat lépései

Az adatbányászat általános lépései 2. ábra láthatóak.



2. ábra Az adatbányászat általános lépései

Adatgyűjtés: kritikus lépés a megfelelő adatok kiválasztása a feladat megoldásához. Az adatgyűjtés során törekednünk kell arra, hogy releváns adatot gyűjtsünk össze a későbbi elemzéshez. Már ebben a lépésben érdemes figyelni arra, hogy lehetőleg minél tisztább, az elemzés tárgyára vonatkozó fókuszáló elemeket gyűjtsünk össze. Feladatom során az adatgyűjtés is automatikus megvalósítással történik. Egy hírportál tartalmát cikkenként elválasztva, nyers szövegfájlokként tároltam, a hozzá kapcsolódó témákkal együtt.

Adatbázis létehozása: meg kell határoznunk a későbbi módszerek számára emészthető olyan formátumot, mely segítségével matematikailag strukturált adattárolást hajtunk végre. Nem csak a kompatibilitás, de a feldolgozás megvalósíthatósága és sebessége múlik ezen. Konkrét adatbázist a feladat megoldása során nem használtam, hiszen a szövegbányászat jellemzői miatt egyszerűbb volt a kapott szövegekből képzett adatmodellt a szoftver belső adatstruktúrájában értelmezett mátrixokként tárolni.

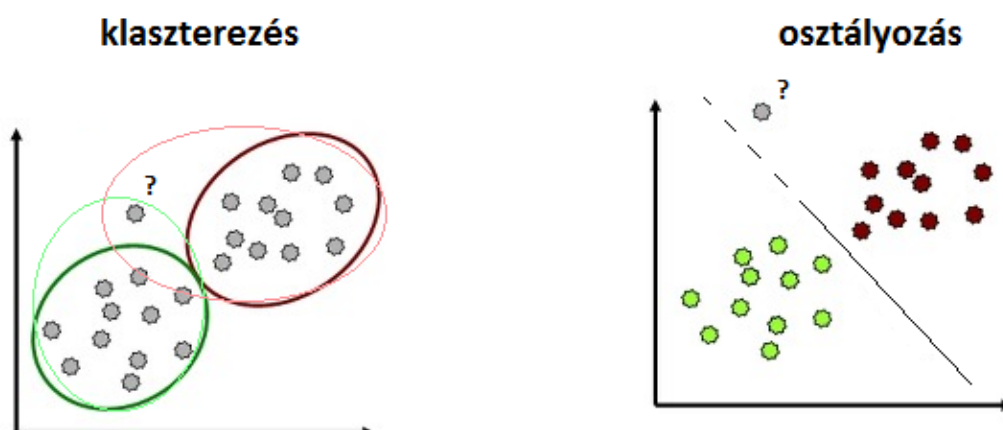
Adatfeltárás: ebben a lépésben történik meg az adatok tisztítása, mely során törekedünk a célterület minél erősebb szűkítésére, felesleges információ eltávolítására. Adattisztítás során olyan zajokra kellett figyelni, mint például a felesleges meta-adatok leválasztása, vagy a hírben szereplő reklámok.

Adatbányászat: ekkor már a pontosan előkészített anyag áll rendelkezésre az adatbányászati eszközök felhasználásához. A különböző módszerek segítségével típusproblémákra tudunk megoldásokat találni, a feladathoz használt konkrét eszköztet lentebb fejtem ki.

Megjelenítés, döntés: itt a legfontosabb, hogy az eredményeket értelmezzük: nem szabad elfelejtenünk, hogy elsősorban a valóságra vetített igazságtartalmára vagyunk kíváncsiak az eredményeknek, nem csak a számokra. Könnyű a metrikákat félreértelmezni: elengedhetetlen, hogy ellenőrizzük, és magasabb szintű (pl. üzleti) elemzéseknek megfelelően előkészítsük az eredményeket. Megjelenítési kérdések a saját feladatom során nem okoztak külön nehézséget, hiszen meg kellett mondanom, hogy melyik témába tartozik az inputként megadott szöveg, és ez elég egyértelmű információ már önmagában is.

Az adatbányászat egyik gyakran előforduló feladata, hogy a rendelkezésre álló adatokat valamilyen módon **csoportosítsa**. Ennek két típusa lehet.

1. **Csoportosítás / klaszterezés:** az adatok feldolgozása során folyamatosan kialakuló szabályrendszer alapján daraboljuk fel az adathalmazt csoportokra. Ekkor a klaszterek határai nem ismertek, erről információt csak fokozatosan tudunk mi magunk kialakítani, nekünk kell „megtanulni” lépésről lépésre.
2. **Osztályozás:** akkor beszélünk osztályozásról, ha a csoportjaink már a feldolgozás első pillanatában ismertek. Ilyenkor a csoporthatárok kialakítása nem a mi feladatunk, csak az elemek folyamatos bekezelésének. Ezt hívjuk felügyelt tanulási folyamatnak, feladatom során ez volt az egyik központi módszere a megoldásnak.



3. ábra Klaszterezés és Osztályozás [7]

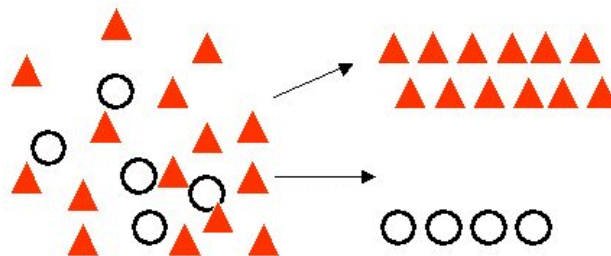
2.2.2 Adatbányászati módszerek: osztályozás

Az adatbányászat több különböző problémával is foglalkozik. Ezek közé tartoznak az osztályozási módszerek, besorolás, vagy például a prediktív analízis. Feladatom során jelenleg egy osztályozási problémával kerültem szembe.

Az osztályozás[4] lényege, hogy előre definiált csoportokba kell valamilyen módszer alapján elhelyezni az újonnan felvitt adatokat. Ez több módon is megtörténhet.

Az osztályozás tehát olyan összefüggések kutatása, az újonnan a rendszerbe kerülő adat, és a már kategorizált adathalmaz között, mely valamilyen hasonlóságot keres. A hasonlóság jellemzője általában valamilyen távolság érték, vektor.

Ahhoz, hogy eme távolságokat értelmezni tudjuk, az adatokat általában vektortérben helyezzük el, különböző jellemzőik segítségével: például ha szeretnénk meghatározni két autó távolságát, akkor a közös jellemzőik (teljesítmény) lesznek egy-egy dimenzió, míg az adott jellemző paraméterei (lóerő) pedig a dimenzióban értelmezett értékek, pontok. Így több jellemző mentén komplex, többdimenziós vektorokat tudunk meghatározni, melyek között már könnyedén tudunk távolságot számolni.



4. ábra Osztályozás [7]

Az így meghatározott vektorokról már rendelkezünk egy bizonyos tudással, osztállyal: előző példánkat tekintve például legyenek a sport- és családi autók. Tudjuk, példának okáért tegyük fel 500 autóról, hogy a kettő közül mely kategóriákba tarznak bele. A kategóriák csoportot alkotnak: az így létrejövő csoportokba próbál meg az osztályozás egy új autót beilleszteni. Paramétereit hozzávetve a már meghatározott többi autóhoz megmondhatjuk, hogy hol helyezkedik el a vektortérben, és nagy valószínűség szerint mi az osztálya az új belépőnek.

Az osztályozási módszereket két csoportra szokás bontani:

- **Modellalapú eljárások:** ha az adott osztályokat jól el tudjuk határolni egymástól. A fenti autós példa is ide tartozik.
- **Modell független eljárások:** ha az osztályok határai nem élesek az egyes egyedek között: itt nem feltétlenül a területeket vesszük figyelembe, hanem az új elem közvetlen környezetét vizsgáljuk, a környékén lévő már osztályozott egyedekhez hasonlítunk.

A feladatom során modell független eljárást alkalmaztam, méghozzá úgy, hogy az újonnan beérkező cikkhez megkerestem a legjobban hasonlító elemet a már feldolgozottak közül, és ennek a kategóriáját adtam vissza. Ezt az úgynevezett Cosinus hasonlóság[9] segítségével valósítottam meg, melynek központi gondolata, hogy euklideszi távolságot számol a vektorok között.

2.3 Szövegbányászat

Az adatbányászat és a szövegbányászat közötti fő különbség tehát az, hogy a szövegbányászat **előkészíti** a strukturálatlan szövegeket olyan adatszerkezetbe, amelyen már értelmezhetőek a korábban említett adatbányászati lépések.

A szövegbányászat feladata, hogy megtalálja, „kibányássza” a feldolgozáshoz szükséges és releváns szövegrészeket. Magas komplexitást igénylő, (és egyelőre még teljes körűen nem megvalósított) lépése a nyelvtani összefüggések, sajátosságok feltárása. A szöveg jelentésének automatikus feldolgozása jelenleg még az emberi agy számára is nehézséget jelent: elég csak a szavak egymásra gyakorolt hatására gondolnunk, de olyan nyelvi elemek is nehezítik a feldolgozást, mint például az ugyanazon szavak ragozott alakjai. A szövegbányászat során próbálunk ezen összefüggések közül minél többet megjelölni, felesleges információkat eltávolítani.

A szövegbányászat nagy fontossága abban rejlik, hogy az emberek által íródott tudásanyagot képes bizonyos fokig automatikusan, természetesen emberi sebességnél jóval gyorsabban feldolgozni. Ügyfélszolgálati, biztonsági, bűnüldözési, üzleti intelligenciai és tudományos kutatásokkal kapcsolatos szövegek kutatása, közöttük lévő kapcsolatok feltárása korábban nem látott lehetőségeket nyitott meg.

Külön területe az internetes keresőmotorok támogatása, ma már a Google keresőmotorja is nagyban támaszkodik a különféle szövegbányászati eszközökre a releváns találatok megjelenítéséhez.

A szövegbányászat általános lépései kissé eltérnek az adatbányászatétól:



5. ábra Szövegbányászat általános lépései

Dokumentum gyűjtemény készítése: elsőként összegyűjtjük azokat a dokumentumokat, melyek az elemzés tárgyát fogják képezni.

Előfeldolgozás során olyan adattisztítást végzünk, amely segítségével egy, az adatbányászati lépéseknek is megfelelő adatstrukturát hozunk létre. Általában valamiféle halmazt, vektormodellt, mátrixot stb. szeretnénk előállítani a nyers szövegből. Ehhez azonban nem szabad 1:1 leképezést használnunk: hiszen rengeteg nyelvi sajátosság bújkál a szövegben, amely kihívást jelent számunkra.

A további lépések megegyeznek az adatbányászatban használt lépésekkel.

2.3.1 Előfeldolgozás

Feladatom során webes portálok szövegeit dolgozzuk fel, Magyar nyelven.

Mivel az egyes cikkek szavai alapján szerettem volna összehasonlítást végezni, különösen fontos volt, hogy csak olyan szavakat tároljak le, amelyek valóban jellemzik az adott szöveget.

2.3.1.1 Fogalmak

A **Stop-szavak** kiszűrése során egy előre definiált, magyar nyelvű stopszótárt[10] használtam. Ebben a magyar írásban található leggyakoribb, nyelvtani és kötésszavakat gyűjtöttem össze. Amennyiben egy dokumentumban ráakadtam ilyen szavakra, azokat egyszerűen kitöröltem az adott dokumentumból. Ilyen szavak például az „az, és, vagy”.

Topic modelling a szövegbányászat egy gyakran használt eszköze. Ez egy statisztikai modellezési eljárás[11] mely során kiemeljük a dokumentum tartalmának központi elemeit, témáit. Például, ha egy szövegben gyakran előfordulnak a sportban használt kifejezések, akkor valószínűleg a kategóriája a sport rovat. Ilyen témabeli hasonlóságok alapján helyezem el a vektortérben az új, beérkező írást, így tippelve meg ennek a témáját, rovatát.

Szótár: az összes, adatbázisban szereplő cikkből készíték egy szótárat, melyben minden szó megtalálható, amelyet a cikkek felhasználnak. Ez maga az az értékkészlet, melyekből a későbbi vektorok felépülnek.

Bag-of-words[12]: Általában egy dokumentumot úgy készítünk elő az adatbányászati lépésekhez, hogy a benne található szövegeket nem a nyers, „rendezetlen” formájában, hanem valamilyen csoportosítás szerint reprezentáljuk. A feladatom során minden dokumentumban a szavakat előfordulásuk gyakoriságával megjelölve tárolok le. Például, ha két dokumentumom létezik: „Gólt lőtt Détári” és „lesérült Détári”, akkor a belőlük készített szótár: „Gól, lő, Détári, lesérül”. Így a két cikkem Bag-of-words modellje a következő: (1,1,1,0) és (0,0,1,1). Ebben az esetben ez a két vektor alkotja a corpus-t.

Corpus: a már elkészült szótár segítségével, bag-of-words által súlyozott, vektorizált dokumentumtér összessége. A kész corpus-on definiáljuk a matematikai- és adatbányászati elemzéseket.

Term Frequency-inverse Document Frequency (tf-idf): ez egy olyan statisztikai érték[13], mely meg tudja határozni, az egész corpus-t figyelembe véve, hogy egy-egy szó mennyire befolyásolja az adott cikk végső kategóriáját. Ha egyes szavak dokumentumai mind ugyanahhoz a kategóriához kapcsolódnak, akkor nagyobb ez az érték, míg ha több kategóriában is előfordulhatnak, ezért bizonytalan a hatásuk a végső kategória-ítéletre, ekkor kisebb. (pl. a sportolók nevei magasabb, míg pl. a stadion szó alacsonyabb értéket kapott, mivel ez utóbbi gyakran jelenik meg belföldi hírekben is.)

Latent semantic indexing (LSI)[14]: nagyméretű dokumentumok esetén láthatjuk, hogy a szótár több tízezer szóból is állhat. Ekkora dimenzió mellett elég nehéz számolnunk: az LSI egy olyan matematikai modell, amely feladata redukálni ezt a dimenziószámot, főként a nem túl releváns dimenziók kiszűrésével, dimenziók összevonásával, a vektortér transzformálásával. a tf-idf elemzés által kijelölt releváns, súlyozott szavak mentén egy pontosabb adatszerkezetet kaphattam.

3 Felhasznált webes technológiák

A feladatom során nagyban építkeztem a korábbi önálló laboratórium során megismert technológiákra. Segítségükkel bővítettem a korábbi megoldást, és implementáltam az új feladatot, kiegészítve a szükséges új eszközöket megismerésével.

A feladat megoldásához egy olyan könnyen konfigurálható, sokrétű webalkalmazás elkészítésére törekedtem, amely bővíthető, könnyedén beépíthető a későbbiekben más rendszerekbe, illetve természetesen az adat- és szövegbányászati modulok is integrálhatóak legyenek bele. A választott keretrendszer így a Python nyelvre íródott framework[15] a Django[16] lett.

3.1 Python

A Python[22] egy általános célú, magas szintű programozási nyelv. Elterjedésének okai többek között a jó felhasználhatóság, elegáns megoldások, és olvasható kódkép. Én a Python 3.3.3-as verzióját használtam a feladat megoldására.

A nyelv objektumorientált, ingyenes felhasználású, működése a legtöbb operációs rendszeren megvalósítható. Scriptek írásától a komplex, többretegű alkalmazásokig használható.

Kódképe tömör, és jól olvasható: szokásos programozási konvencióktól több helyen is eltér az egyszerűség és átláthatóság céljából. A program elválasztó eleme a bekezdések, a szokásos kapcsos zárójelek helyett.

```
>>> def fib(n):
>>>     a, b = 0, 1
>>>     while a < n:
>>>         print(a, end=' ')
>>>         a, b = b, a+b
>>>     print()
>>> fib(1000)
```

6. ábra Fibonacci számok kiszámolása[16]

Automatikus, és effektív erőforrás kezelést valósít meg, adattípusai magasan fejlettek. Pointerek használata egyáltalán nincs. Könnyedén többszálúsítható, és jól interpretált, compile működése is lehetséges. Széles fejlesztői réteg munkálkodik mögötte, melynek köszönhetően folyamatosan fejlődik.

3.2 Webalkalmazások

A feladathoz megalkotott szoftverem egyik jellemzője, hogy webes alkalmazásként elérhető a szabványos böngészők segítségével.

A szoftver felhasználó felülete tehát HTML weblapokból áll, melyeket egy webszerver állít elő. Ez a webszerver felelős az alkalmazás üzleti logikájának futtatásáért, a webes kérések kiszolgálásáért, megjelenítendő weboldalak előállításáért és az érkező http kérések kiszolgálásáért.

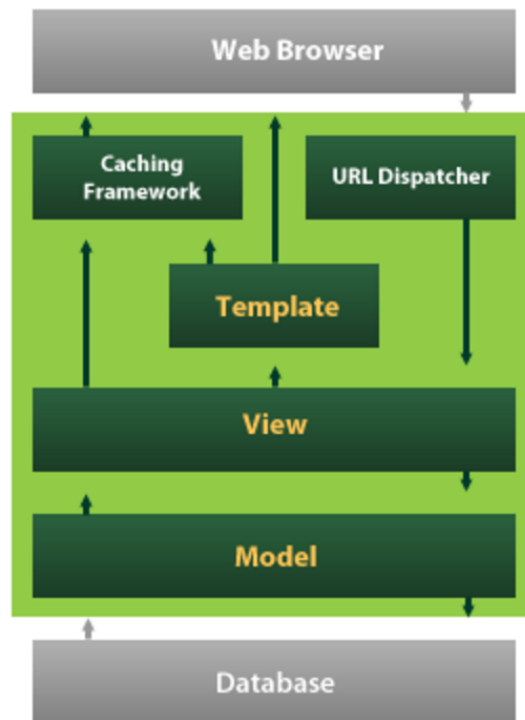
A webszerver http felett különböző csomagokat küld, és fogad. A kéréseknek két kategóriája van:

GET: a kliens oldaláról érkező http request csomag, mely valamilyen adat lekérdezésére irányul. Például az egyes linkekre való kattintás, vagy gombok lenyomása.

POST: a kliens oldaláról érkező, szintén http request típus, amely valamilyen adatot küld feldolgozás céljára. A felhasználó a feladatomban során, a webes felületen adhatja meg azt a szöveget, amelyhez szeretne kategória címkét kérni az alkalmazástól. A szöveget egy POST üzenetben küldi el a szerver számára, mely azt továbbítja az adatbányászati modul felé.

3.3 Django

A Django keretrendszer egy Python programcsomag. A többi keretrendszerhez hasonlóan, egy már megvalósított, gyakran újrafelhasznált osztályokat tartalmazó csomag, mely segítségével nem kell nulláról kezdenünk egy webalkalmazás lefejlesztését.



7. ábra Django architektúra[19]

Az alkalmazás Model-View-Controller[17] alapokra támaszkodik és fő célja az új webalkalmazások egyszerű létrehozása: az előre meghatározott építőelemek segítségével percek alatt képesek vagyunk egy működő webszervert indítani rajta a telepített alkalmazásunkkal, mely képes weboldalak kiszolgálására, websevice-ek üzemeltetésére.

Az 7. ábra látható a szoftver általános szerkezete. Az adatbázis és a model közötti szinkronizáció automatikus. A view réteg dolgozza fel a beérkező kéréseket, szólítja meg a belső üzleti logikát, (ami a mostani esetben az adatbányászati modul függvényei) éri el az adatbázist, és e hármusból állítja elő az előre definiált weboldal-template-ek kiegészítésével a megjelenítendő weboldalt.

Egy tipikus weboldal létrehozása a következő lépésekből állhat:

1. Architektúra, könyvtárszerkezet kialakítása. A `manage.py` paraméterezésével könnyedén kialakítható egy előzetes könyvtárszerkezet, a különböző rétegeknek megfelelően. Ennél a lépésnél már képesek vagyunk elindítani az alkalmazást, kiszolgálni a kéréseket, adminisztrátori felületen belépni, felhasználókat menedzselni: tehát a leggyakoribb feladatokhoz megkapjuk a keretet.

2. Model definiálás. A keretrendszer segítségével előre definiált módon tudunk model objektumokat létrehozni, melyek előre generált megjelenítő felületekkel rendelkeznek. A feladatom során olyan „cikk” objektumokat hoztam létre, melyek ilyen módon már előre rendelkeztek a hozzájuk tartozó megfelelő model form-mal, és adatbázis perzisztálással.
3. URL-ek megadása: a weboldalak bekötésére külön felület áll rendelkezésre, ahol az egyes kérések belépési pontjait, és a válasz oldalak elérhetőségét állíthatjuk be.
4. Template-ek létrehozása. A webes megjelenítő felületre egy saját template nyelv segítségével lehet definiálni a későbbiekben legenerálandó, statikus és dinamikus tartalommal rendelkező oldalakat.
5. Control réteg. Itt ér össze az útvonalak, kérések kiszolgálása. A beérkező GET és POST http kérésekre az üzleti logikát felhasználva válaszokat állítunk elő, amelyeket a template-ek, és a bennük található dinamikus változók behelyettesítésével hozunk létre.

Természetesen ez csak egy szelete a keretrendszerben rejlő lehetőségeknek: a feladatom megoldásához azonban az itt felsorolt lehetőségek elegendőnek is bizonyultak az adatbányászati modul felhasználásához, annak megjelenítéséhez.

3.4 Beautiful Soup

A Beautiful Soup[18] egy Python library, ami web parse feladatok könnyű megvalósításához tervezek.

Képes egyszerű navigációs lépésekre weboldalak lekéréséhez.

A lekérdezett weboldalak dokumentum alapú tartalmát egy belső fa adatszerkezetben tárolja, melyen különböző feldolgozási lépéseket lehet megvalósítani.

Támogatja a tartalmak felolvasását, könnyű és egyszerű szintaktikája segítségével gyorsan elérhetjük az XML fájlok számunkra érdekes részeit. Unicode kódolást támogatva ideális magyar nyelvű szövegek kinyerésére is.

Erőssége abban rejlik, hogy azon parser-eken felül, amelyre építkezik, (lxml, html5lib) előre definiált bejárési eszközökkel is rendelkezik, tehát ezt nem nekünk kell megvalósítani. Ez utóbbi tulajdonsága komplex webcrawl feladatoknál nagyon hasznos.

Feladatom során ezt az eszközt használom a weboldalak RSS feed-jének[20] lekérésére, ebből a linkek kiválogatására. A linkek mögött rejlő weboldalak szövegét, és az adott cikk kategóriáját is ennek az eszköznek a segítségével érem el.

Például egy cikkekre mutató url-ből a következő módon tudjuk kinyerni a szöveg címét:

```
def parse_title(urlText):  
    resp = requests.get(urlText)  
    soup = BeautifulSoup(resp.text)  
    return soup.find('h1').text
```

8. ábra URL parse-olása

Első lépésben a resp objektumban eltároljuk a lekérdezett html oldalt. a soup objektum tárolja a html oldal szöveg részét, a felesleges meta adatok nélkül, a már említett fa formátumban. A find függvény visszatér a html dokumentum <h1> class tag-el jelölt szövegével.

3.5 Gensim

A gensim[21] egy topic modellezést megvalósító Python programcsomag. Open source projektként a fő feladata, hogy Python nyelven egy könnyedén, de mégis sokrétűen alkalmazható topic modelling eszköz legyen.

```
>>> from gensim import corpora, models, similarities  
>>>  
>>> # Load corpus iterator from a Matrix Market file on disk.  
>>> corpus = corpora.MmCorpus('/path/to/corpus.mm')  
>>>  
>>> # Initialize Latent Semantic Indexing with 200 dimensions.  
>>> lsi = models.LsiModel(corpus, num_topics=200)  
>>>  
>>> # Convert another corpus to the latent space and index it.  
>>> index = similarities.MatrixSimilarity(lsi[another_corpus])  
>>>  
>>> # Compute similarity of a query vs. indexed documents  
>>> sims = index[query]
```

9. ábra Gensim képességek

A szövegbányászat előkészítési, és adatbányászati lépéseit hivatott megvalósítani a programcsomag. Erőssége abban rejlik, hogy nem kell kézzel végrehajtani a lépéseket, hanem a weboldalba beépülve képes a szerveren elvégezni az összes feladatot.

A gensim segítségével végzem el a dokumentum vektor transzformációját, a szótár és a corpus létrehozását, a tf-idf elemzést és az LSI transzformációt is.

3.5.1 Gensim funkciók

A gensim a procedurális feldolgozás jegyében a korábban ismertetett adat- és szövegbányászati lépéseket a következő funkciók segítségével valósítom meg.

Corpus és Vektortér kezelés: A dokumentumok feldolgozásakor String-listákat használ, hiszen ezeket könnyű végigiterálni például a stop-szavak kiszűréséhez. Automatikusan elvégzi a szavak tokenizációját (azaz az összefüggő szövegből a szóközök mentén történő feldarabolását) és létrehozza a szótárat belőlük. Létrehozza a szógyakoriság-előfordulás vektort. Képes eme vektorok összességét (a corpus-t) együtt kezelni.

Corpus, mint mátrix: a mátrix műveletek segítségével gyorsítja a feldolgozást. A corpus-t a gensim már, mint mátrixot tárolja.

Matematikai modulok: a NumPy és SciPy matematikai függvénykönyvtárak felhasználásával végezzük el a különböző átalakításokat, műveleteket.

Transzformációk: könnyedén tudunk bag-of words, tf-idf, vagy bármilyen, saját magunk által definiált vektorteret előállítani, egyikből a másikba transzformálni.

Hasonlósági elemzés: különböző vektorok közötti távolság számítását is elvégzi, a hasonlósági mutatók szerint könnyedén kiválaszthatjuk egy új cikk szomszédjait.

4 Tervezési fázis

Ahogy általánosságban is elmondható az adatbányászatról, a feladatom megoldása során is kiemelt figyelmet kellett fordítanom a tervezésre.

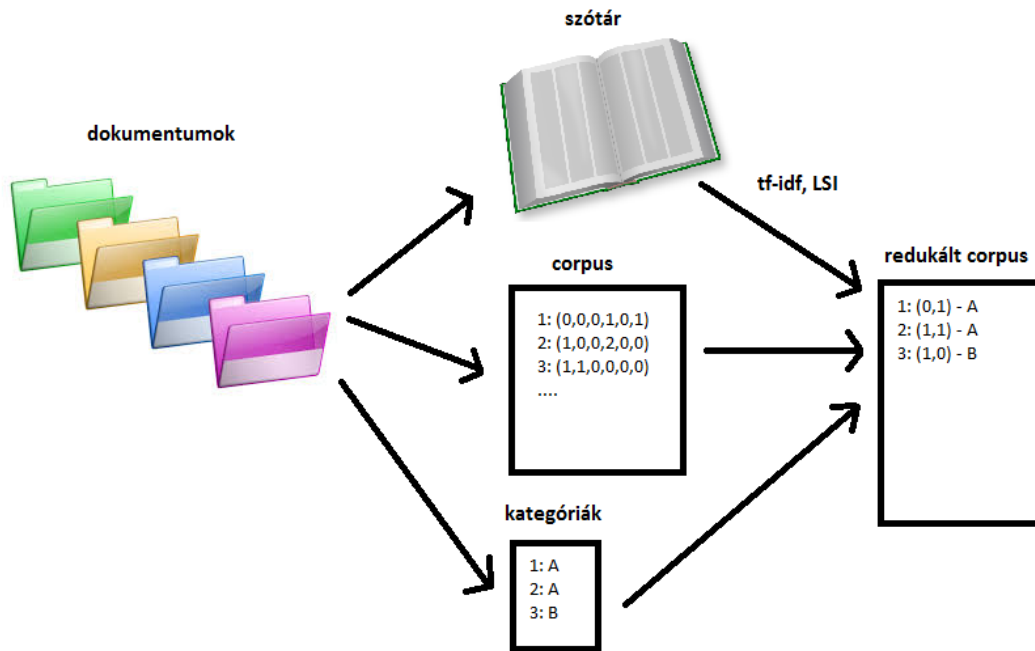
Figyelembe kellett vennem, hogy olyan feladatot jelöljek ki, amely megvalósítható, és releváns információkhoz juttatja a felhasználót. Fontos, hogy úgy lássunk hozzá a munkához, hogy van egy elképzelésünk arról, milyen elemzések eredményeit fogjuk látni, a felhasznált módszereknek milyen sajátosságaira kell ügyelni, közülük melyiket kell választani.

A tervezési fázis során az Önálló laboratóriumom 2 tárgy keretein belül szerzett tapasztalataimat vettem alapul. Ebben sikeresen alkalmaztam szövegbányászati elemzésekre a most is használt eszközöket, és sikerült integrálni a Django keretrendszerbe az akkori moduljaimat.

Fontos tapasztalat volt, hogy minél egyszerűbb elemzésekkel kezdjek, és csak ha az alapgondolat működik, akkor lássak neki a megoldás pontosságának növeléséhez. Ezek mellett figyelnem kellett arra, hogy például néhány eredmény hibásnak tűnik, viszont mégsem az: tesztelés alatt például felmerült, hogy különböző kategóriák között nagy volt az átfedés. Például a Tudomány, és Tech rovatok között nehezen határozható meg egy-egy hír pontos hovatartozása.

Másik tapasztalat volt, hogy az eredmény legyen minél egyszerűbb. A feladat során csak a kategóriát kell meghatároznom: ha ez sikeres, akkor joggal feltételezhetjük, hogy az elemzés további bonyolításának van még létjogosultsága egyéb adatokra is.

A tervezés során a 10. ábra látható lépéseket terveztem megvalósítani.



10. ábra Az alkalmazás terve

Elsőként az összegyűjtött dokumentumokból kialakítom a szótárt, a corpus-t, és a kategóriákat. A szavak súlyozása (tf-idf) és a dokumentumtér dimenziójának csökkentése (tf-idf) után redukált corpus-t hozok létre, ahol mindegyik dokumentumhoz eltárolom a megfelelő kategóriát.

Az új cikkeket eme lépések után ehhez a redukált corpus-hoz hasonlítom: terveim szerint így világos eredményekre juthatunk, hiszen egyrészt csak a cikket jellemző legfontabb, és a corpus-ban előforduló leggyakoribb szavakat használom fel.

5 Szoftver implementálása

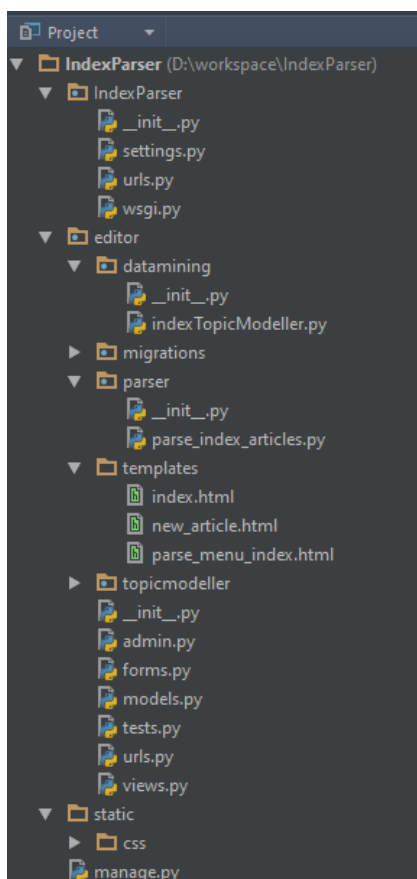
A következő részben részletesen bemutatom, hogy milyen lépéseken keresztül valósítottam meg a weboldalt.

5.1 Webes megjelenítő rész

A Django keretrendszernek hála könnyedén tudtam működőképes weboldalt létrehozni. A hivatalos tutorial-t követve[16] könnyedén működő weboldalt tudunk összeállítani, ennek felhasználásával haladtam én is.

A megvalósítás specifikus részei a következők voltak:

5.2 Alkalmazás architektúra



11. ábra Az alkalmazás szerkezete

Az alkalmazás a következő modulokból áll:

IndexParser: a csomagban található fájlok segítségével konfigurálható a webserver.

Editor/datamining: ebben a csomagban található az a fájl, amelyben a szöveg- és adatbányászati lépések megtörténnek a feldolgozás során.

Editor/parser: ebben a csomagban foglalnak helyet a web parser eszközök.

Editor/templates: itt találhatóak a válaszként generálódó weboldalak template-jei. Jelenleg egy kezdőoldal, egy parser felület, és egy új cikk megadására szolgáló szerkesztő ablak található itt.

```
<form action="/index/" method="post">
  {% csrf_token %}
  {{ form }}
  <input type="submit" value="Submit" />
</form>
```

12. ábra Egy példa template

Az 12. ábra egy egyszerű template kódot láthatunk: megfigyelhetjük, hogy a html tartalom mellett a kapcsos zárójelek között szereplő értékek mutatnak a dinamikus tartalomra. Az itt szereplő példa egy egyszerű form-ot valósít meg, egy darab küldés gombbal, amelyet POST kérésben ad vissza a weboldalnak. A form dinamikusan generálódik attól függően, hogy melyik objektumot rendeli hozzá a view.py a template-hez.

Editor/forms.py : az az a fájl, amelyben a form típusú modelleket definiáltam.

Editor/models.py : a modellek definiálása. Az általam használt központi form, és az ezt implementáló model a következőképpen nézett ki:

```
class webArticle(forms.ModelForm):
    class article(models.Model):
        textBody = models.TextField()
        category = models.CharField(max_length = 100)
```

13. ábra Felhasznált Django model

Editor/admin.py: adminisztrációs felület beállításai, jelenleg nem használt.

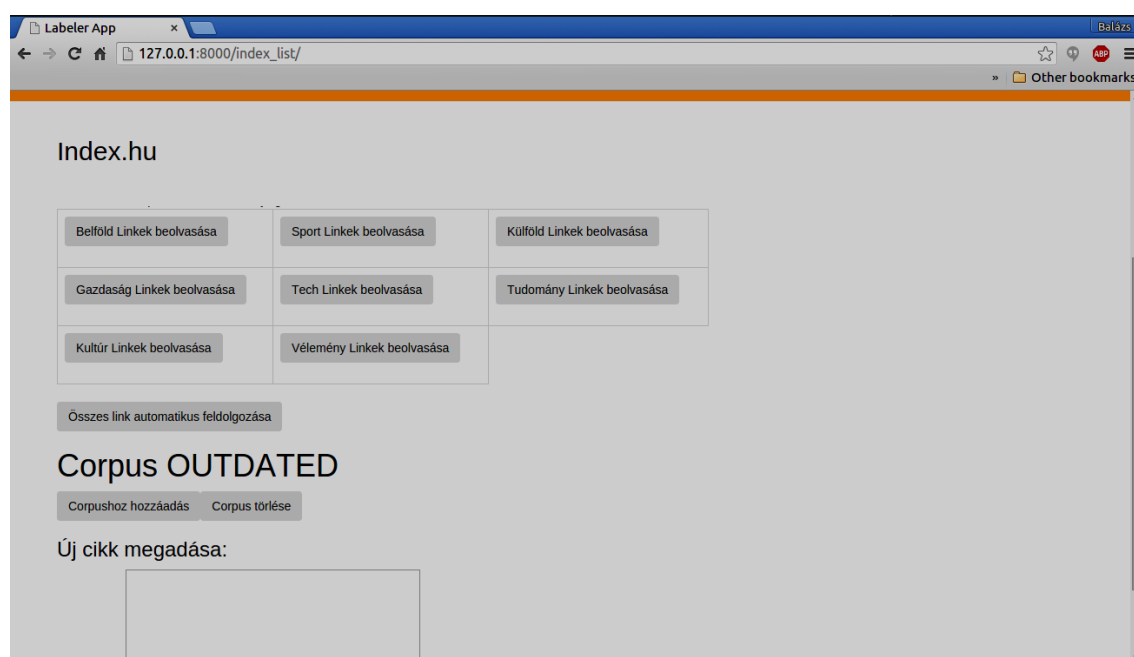
Editor/url.py: az útvonalak definiálása a különböző lekérésekhez.

Editor/videws.py: neve csalóka: valójában a control réteg megvalósítása, és a view generálására szolgáló függvények találhatóak itt. Ez gyakorlatilag az alkalmazás központi magja, ide futnak be végül a webes lekérések, innen szólítjuk meg az adatbázist és ebből az osztályból hívjuk meg a parser és adatbányász függvényeket, mielőtt legeneráljuk a kimenetet.

Static/css: stíluslapok.

A kezdőoldal az 14. ábra látható.

5.3 Parser modul



14. ábra Felhasználói felület

A parser modult a view.py akkor hívja meg, ha a felületen egy RSS feed gombra kattintunk: ekkor a gomb paramétereinek megfelelő RSS töltődik be, mint cél url lista.

A corpus-hoz hozzáadás gomb, mindig az aktuális RSS tartalmával hívja meg a parser modul erre a célra kijelölt függvényét. Ekkor a parser felolvassa az RSS feed tartalmát, kiszűri belőle a cikkek linkjeit, és ezeket egyesével lekérdezi a címtől.

A lekérdezett weboldalak szövegtörzsét kiválogatja, és fájlként lementi egy erre kijelölt mappába. Így jönnek létre a cikkek.

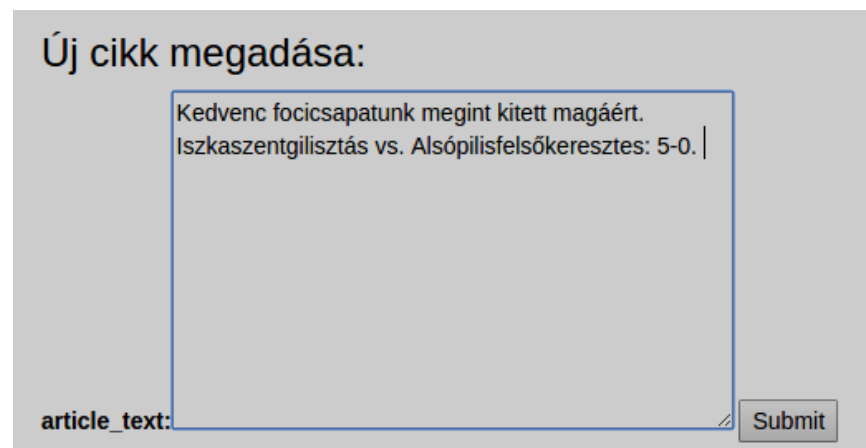
Minden weboldal mellé eltároljuk az adott cikk kategóriáját is a fájl első sorában.

Ezek után meghívódik a szövegbányászati modul egy függvénye, amely a már kész adatokkal „tanítja” a corpus-t, tehát itt prediktív analízist nem végzünk, csak hozzáadjuk a szótárhoz az új szavakat, lementjük a szöveget a corpus-ba, és eltároljuk a hozzá tartozó kategóriát

5.4 Szövegbányászati modul

A szövegbányászati modul lépései a következőképp történnek:

1. Egy új cikket írunk be az erre a célból létrehozott form-ba a weboldalon.
Rányomunk a mentés gombra.



15. ábra Új cikk megadása felület

2. A view.py feldolgozva a POST kérést meghívja az input szöveggel a szövegbányászati modult erre kijelölt függvényét.

```
def add_to_corpus(request):  
    if request.method == "POST":  
        szoveg = request.POST("textArea")  
        category = indexTopicModeller.addToCorpus(szoveg)  
        return render(request, 'parse_menu_index.html',  
            {'categoryResult':category})
```

16. ábra POST kérés feldolgozása

3. A függvény megnyitja a letárolt corpus-t, szótárat, és a kategóriák listáját.
4. A beérkezett szöveget tokenizáljuk.
5. A szövegből kiszűrjük a stop-szavakat

```
tokenizalt = [[word for word in document.lower().split() if word not in  
stopwordlist] for document in documents_list]
```

17. ábra Stopszó szűrés és tokenizálás

6. Az új szövegben előforduló esetleges új szavakat hozzáadjuk a szótárhoz.

```
dictionary = corpora.Dictionary(tokenizalt)
```

18. ábra Szótár bővítése

7. Transzformáljuk a szöveget vektortérbe.

```
corpus = [dictionary.doc2bow(text) for text in tokenizalt]
```

19. ábra Vektortér transzformáció

8. A gensim saját adatmodellje segítségével a szöveget tf-idf szerint súlyozzuk.

9. Az így kialakult szöveget LSI mentén transzformálom kisebb dimenziós térbe. (jelenleg ez megegyezik a kategóriák számával)

```
vec_bow = dictionary.doc2bow(article.lower().split())  
vec_lsi = lsi[vec_bow]
```

20. ábra LSI transzformáció

10. A tf-idf és LSI transzformációkat alkalmazom a corpus-on is.

```
lsi = models.LsiModel(corpus, id2word=dictionary, num_topics=8)
```

21. ábra Corpus transzformálása

11. A két eredményt összehasonlítom és megkeresem azt a dokumentumot, amelyik a legközelebb helyezkedik el a vektortérben a megadott szöveghez.

```
index = similarities.MatrixSimilarity(lsi[corpus])  
sims = index[vec_lsi]  
sims = sorted(enumerate(sims), key=lambda item: -item[1])  
print("--- Similarities:")  
print(sims)
```

22. ábra Hasonlósági elemzés

Az így kapott lista tetején lévő dokumentum lesz a legjobban hasonló elem.

12. A legközelebbi szöveg kategóriáját felajánlom, mint valószínűsíthető kategória.

13. Letárolom az adatokat a merevlemezre.



23. ábra Az eljárás kimenete

A corpus-hoz való hozzáadás után az alkalmazás kijelzi a meghatározott kategóriát a felhasználónak. Ez az alkalmazás központi működési mechanizmusa.

6 Eredmények

A feladatot elvégzése során az index.hu-n található 8 főbb kategória 50-50 cikkét olvastam be. A megfelelő hosszúságú, és jól paraméterezett szövegekre az alkalmazás jól tippelte be a kategóriákat.

Tesztelés során olyan, a tanulási folyamathoz nem használt cikkek szövegeit adtam meg, amelyek kategóriáját előre tudtam az index.hu-ról. Az eredmények igen biztatóak. Tesztelés során ~100 szöveget parsoltam fel ilyen-olyan tesztelési célokból. (eleinte csak 1-1 szó elhelyezkedését vizsgáltam, későbbiekben már új cikkek szövegeit)

Elsőként csak két kategóriát teszteltem „egymás ellen”: például a Tech, és a Belföld hírkategóriák között nagyon jól teljesített a program. A teljesítmény relatív, hiszen nagyban függ a megadott cikkek szövegétől, illetve a corpus nagyságától is.

Az olyan témák, mint a Sport, vagy Belföld jobban körvonalazhatóak voltak, de például a Vélemény rovat nehezen megközelíthető: hiszen egy politikai hír tudósítása és kommentálása között nagyon sok hasonlóság van, könnyedén téveszti őket össze a program.

Összességében úgy érzem, sikeres volt a feladat megvalósítása, bár még pontosításokra a jövőben mindenképp van lehetőség.

A feladat során ~600 sor kódot írtam. Elmélyítettem ismereteimet a Django keretrendszer működésében, saját modult és felületeket terveztem. Az időm jelentős hányadát tette ki az adatbányászati modul, és a szövegbányászati lépések megtervezése, tesztelése. Az elkészült függvényeket beillesztettem a webalkalmazásba, megismertem mélyrehatóan a feladathoz szükséges webes parse megoldásokat. Elkészítettem egy működő alkalmazást, amely megvalósítja a feladatkiírást egy konkrét hírportál esetében.

7 További lehetőségek

Az egyik lehetséges út a felhasznált adatbányászati eljárás finomítása. Szótövezési eljárások bevezetése, illetve további adatbányászati eljárások kipróbálása, tesztelése, sikeresség esetén implementálása. Ehhez tervezek kialakítani egy automatikus tesztelő egységet.

A másik útvonal a weboldalaktól való függetlenítés: olyan menüpont elkészítése, amellyel dinamikusan tudunk megadni új hírforrásokat az alkalmazás számára.

Továbbfejlesztési lehetőség még a cikk egyéb meta adatainak betippelése: például címkék hozzáadása, integrálva ezzel a korábbi önálló laboratórium tárgyam eredményeit, illetve statisztika mutatása, több közeli, témában hasonló cikk megkeresése, illetve a felhasználó számára a felület további eszköztárának bővítése.

Másik ötlet, hogy a fájlok memóriában, majd pedig a diszken történő szöveges fájllokként való tárolása helyett egy adatbázist építsek fel a rendszer mögé. Az adattárolási réteg a skálázhatóságot és a szoftver feldolgozó erejét növelhetné, illetve új technológiák megismerését vonná maga után.

Szintén érdekes eredményekre lehet jutni, ha növeljük a coprus méretét, ha több hírportál, nagyságrendekkel nagyobb adatbázisát hasonlítjuk össze. A legtöbb portálon sok kategória közös, ezeken végzett széles körű elemzés nagyban növelheti az eredmények pontosságát.

Irodalomjegyzék

- [1] Kiss Balázs, Nagy Gábor, BME TMIT: *Dokumentum osztályozás adatbányászati módszerekkel* (2014)
- [2] Kiss Balázs, Nagy Gábor, BME TMIT: *Webes dokumentum osztályozás* (2015)
- [3] How Stuff Works: *How internet search engines work*
<http://computer.howstuffworks.com/internet/basics/search-engine1.htmtemp>
(2015)
- [4] Dr. Abonyi J. 2006. *Adatbányászat, a hatékonyság eszköze* ISBN:963-618-432-2, ComputerBooks
- [5] Kiss Balázs, Gáspár Csaba, BME TMIT: *E-mail postafiók adatbányászati elemzése* (2014)
- [6] Tikk D. 2007. *Szövegbányászat* ISBN 978-963-9664-45-6, Typotex
- [7] Arisiri: *Clustering VS Classifictaion* <http://arisri.tistory.com/entry/Clustering-VS-Classification> (2015)
- [8] Oracle: *Classification*
<http://www.oracle.com/ocom/groups/public/@otn/documents/digitalasset/115094.gif> (2015)
- [9] Wikipedia: *Cosine similarity* https://en.wikipedia.org/wiki/Cosine_similarity (2015)
- [10] Tutorial.hu: *Hunagrian stop words* <http://www.tutorial.hu/hungarian-stop-words> (2015)
- [11] Wikipedia: *Topic model* http://en.wikipedia.org/wiki/Topic_model (2011)
- [12] Wikipedia: *Bags of words model* http://en.wikipedia.org/wiki/Bag-of-words_model (2014)
- [13] Tf-idf: *A single-page tutorial* <http://www.tfidf.com> (2015)
- [14] Amy Langville: *The Linear Algebra Behind Search Engines – And Andvanced Vector Space Model: LSI* Mathematical Association of America
<http://www.maa.org/publications/periodicals/loci/joma/the-linear-algebra-behindsearch-engines-an-advanced-vector-space-model-lsi> (2005)
- [15] Jeff Knupp: *What is a Web Framework?*
<https://www.jeffknupp.com/blog/2014/03/03/what-is-a-web-framework/> (2015)
- [16] Django Software Foundation: *Django* <https://www.djangoproject.com> (2015)

- [17] Wikipedia: *Modell-nézet-vezérlő* <https://hu.wikipedia.org/wiki/Modell-n%C3%A9zet-vez%C3%A9rl%C5%91> (2015)
- [18] Leonard R. *BeautifulSoup* <http://www.crummy.com/software/BeautifulSoup> (2015)
- [19] MyTARDIS *Django Architecture* https://mytardis.readthedocs.org/en/latest/_images/DjangoArchitectureJeffCroft.png (2015)
- [20] Network Working Group, *The application/rss+xml Media Type* <http://tools.ietf.org/id/draft-nottingham-rss-media-type-00.txt> (2007)
- [21] Radim Ř. *Gensim* <https://radimrehurek.com/gensim> (2009)
- [22] Gérard Swinnen: *Tanuljunk meg programozni Python nyelven* <http://mek.oszk.hu/08400/08435/08435.pdf> (2005)