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

Dokumentum osztályozás adatbányászati módszerekkel

Diplomaterv

Készítette
Kiss Balázs

Konzulens
Nagy Gábor

2017. május 12.

Tartalomjegyzék

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 autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2017. május 12.

Kiss Balázs
hallgató

Kivonat

Feladatom során egy webes alkalmazást készítettem el, mely képes weboldalak szöveges tartalmának feldolgozására, elemzésére adat- és szövegbányászati módszerek segítségével.

A feladat megvalósításának érdekében megismerkedtem a webalkalmazások működési elvével, elsajátítottam a Python nyelv alapjait, megismertem a Django webes keretrendszert. Implementáltam a korábbi tanulmányaim alatt szerzett szöveg- és adatbányászati ismereteket, és az elért eredményeimet felhasználva automatikus adatbányászati folyamatokat integráltam a webalkalmazásba.

A webalkalmazás adat- és szövegbányászati eszközöket felhasználva alkalmas általános információk kinyerésére és weboldal kategóriák meghatározására. Keretrendszert nyújt specifikus weboldalakra történő, célzott adatbányászati elemzések elkészítéséhez.

A webes felületen keresztül lehetősége van a felhasználónak az előre elkészített kutatás és általános célú elemzések futtatására is. A weboldal az alapvető webes funkcionalitásokkal rendelkezik (felhasználó azonosítás, internetes működés, reszponzív grafikus felületek), és a felhasználó által betáplált dokumentumokat és webcímeket analizálja.

A keretrendszerbe integrált adat- és szövegbányászati eljárások, web-crawler technológiák és megjelenítő eszközök segítségével, külső beavatkozás nélkül képes az adatok feldolgozására, eredmények közlésére.

Abstract

I have created a web software, which is able to process and analyze the content of websites, with the help of data- and textmining methods.

In order to solve my task, I learned about the basic concepts of web-based applications, the fundaments of the Python programming language, and the core functionality of the Djnago web framework. I implemented my knowledge about data- and text mining methods based on my earlier experiences, and introduced the results to an automated dataminer web application.

The web software, by using data- and text mining tools, is able to recover general informations and categorise websites. It is a framework for specialized datamining tasks, aimed at websites.

On the web based user interface, the user is able to run pre-defined researches as well as general-purpose analyses. The website offers basic functionalities (user identification, web-based operation, responsive graphical interfaces), and researches the webadresses, entered by the user.

The data- and text mining methods, web-crawler technologies and the presentation tools that are build into the framework are able to process data, present results without further user intervention.

Bevezető

A diplomám elkészítése során a következő kérdésre kerestem a választ: tisztán adat- és szövegbányászati eszközök segítségével milyen mélységig és milyen eszközök segítségével lehet internetes tartalmat automatikusan feltérképezni? A kérdés megválaszolása érdekében több különböző szövegbányászati és adatbányászati feladatot oldottam meg, körbejárva a témakört.

Munkámat a korábbi szövegbányászati munkáim eredményeiből kiindulva, azokat felhasználva készítettem el. Létrehoztam egy webes felületen keresztül irányítható alkalmazást, mely megvalósítja az adat- és szövegbányászati elemzéseket. Így a feladatom nem csupán az elemzések, algoritmusok megalkotásából állt, hanem ezek egy közös alkalmazásba rendezéséből, webes technológiák mélyreható felhasználásával.

A kutatásaimra épült végleges webalkalmazás két nagy feladatra alkalmas:

1. Segítségével nagyméretű internetes tartomány térképezhető fel, úgy, hogy a weboldalak szöveges tartalma alapján egy általános elemzést, kategorizálást végez el.

2. A szoftver képes (előre kiválasztott) specifikus weboldalak mélyreható vizsgálata, tartalmuk alapos elemzésére.

Mivel természetesen az egész internetet emberi értelmezés nélkül a jelenleg rendelkezésre álló eszközökkel feltérképezni nem lehet, így az általános feltérképező modul nem ad olyan részletes elemzést, mint az előre kijelölt weboldalak specifikus tartalmára felkészített második modul. Itt hírportálok részletes elemzését jelöltem ki, mint megvalósítandó feladat: elsősorban a hírportálra érkező cikkek kategorizálását, automatikus beolvasását, adatbányászati elemzését tűztem ki célul, erre hasznos webes funkcionalitást építve.

A nagyméretű webtartomány automatikus feltérképezése segítséget nyújthat a webes tartalmak kategorizálásához, válogatásához, veszélyes weblapok kiszűréséhez.

A részletes webes elemző segítségére lehet az újságíróknak a napi aktualitások gyors megállapítására, új cikkek írásakor a megfelelő kategóriába történő besoroláshoz.

Önálló kutatásaim keretein belül már végeztem kimutatást a magyar nyelvű szö-

vegbányászat ilyen jellegű alkalmazásainak eredményességéről[13] Ekkor elsősorban internetes hírportál cikkeit összefoglaló kulcsszavak, azaz címkék meghatározásának lehetőségeit vizsgáltam.

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

Második 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[14]. 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 az említett eredményekre épülve képes automatikusan felolvasni webes tartalmakat, és azokat megszűri, kategorizálja illetve elemzi. A feladat kihívása abban rejlik, hogy megismerjem az ehhez szükséges web-crawler technológiákat[33], megállapítsam, milyen elemzéseken és milyen komplexitásig lehet elvégezni a rendelkezésre álló eszközökkel, és hogy megalkossam a dokumentum feldolgozási- és adatbányászati modelleket, melyek elvégzik magát a kutatást.

A munkám során tovább mélyítettem a felhasznált technológiák ismeretét, jelentősen bővítettem a korábbi szoftveremet, mely végül alkalmassá vált komplex dokumentumelemzési feladatokra.

A feladat

A diplomamunkám célja, hogy egy komplex szöveg- és adatbányászati elemzőszoftvert valósítsak meg, mely beépül egy webes szoftverbe. Az alkalmazás komplex szöveg és dokumentumelemzési feladatait böngészőn keresztül irányíthatja a felhasználó.

A munkám során három fő feladatot kellett megvalósítanom:

1. A magyar nyelvű internetes tartalom témaelemzési megvalósíthatóságának vizsgálata egy konkrét példán keresztül (6. fejezet).

2. Ismeretlen webes tartalom feltérképezése, kategorizálása automatikus tartalomkinyerés segítségével (5. fejezet).

3. Előkészített webtartalom elemzése kijelölt weboldalak esetében (6. fejezet).

A feladatok egymásra épülnek. A megoldáshoz szükséges eszközök megismerése, az algoritmusok megtervezése és implementálása, a kapott eredmények fényében a módszerek kiválasztása és módosítása, saját algoritmusok és eszközök elkészítése mind a feladat részét képzik.

Kihívás a megfelelő módszerek kiválasztása a cél érdekében, ezek implementálása az adott webes keretrendszerben, és az eredmények értelmezése, elemzése.

A végső szoftverek alkalmasnak kell lennie komplex adatbányászati folyamatok megvalósítására.

1. fejezet

Adat- és szövegbányászat

1.1. Bevezetés

Az adatbányászat[12] 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 alapfogalmakra, és a feladatban érintett témakörökre térek ki.

Ahogy Abonyi János fogalmaz[12]: "Az adatbányászat egy olyan döntéstámogatást szolgáló folyamat, mely érvényes, hasznos, és előzőleg nem ismert, tömör információt tár fel nagy adathalmazból".

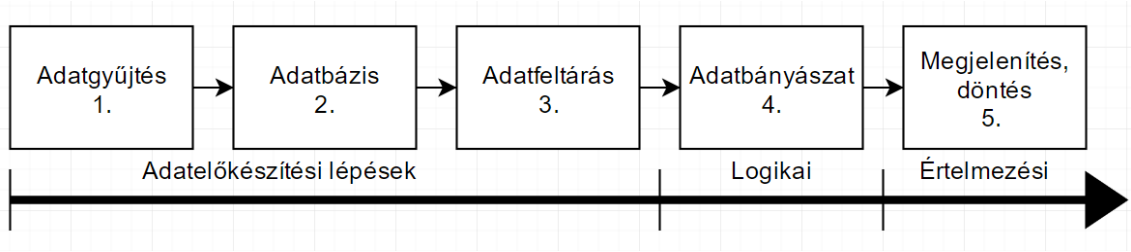
A szövegbányászat[5] szorosan kapcsolódik az adatbányászathoz. Csak ellenben az adatbányászattal, ahol általában valamilyen matematikai struktúrában tárolt adatokat elemzünk, a szövegbányászat során adathalmazunk jellemzően strukturálatlan. Nevéből adódóan általános jellegű, értelmes szövegek gépi feldolgozására vonatkozik. A szövegbányászat feladata, hogy előkészítse ezt a rendezetlen adathalmazt későbbi adatbányászati elemzésekhez. A kihívás az emberi nyelvtan összefüggéseinek matematikai módszerekkel történő feldolgozásában rejlik, illetve hogy eme információkat rendezzük és kiemeljük a későbbi kutatásokhoz.

1.2. 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 matematikai eszköz és lehetőségünk nyílik erre [12].

1.2.1. Adatbányászat lépései

Az adatbányászat általános lépései az 1.1 ábrán láthatóak.



1.1. 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 figyelnünk arra, hogy lehetőleg minél tisztább, az elemzés tárgyára fókuszáló dokumentumokat gyűjtsünk össze. Feladatom során az adatgyűjtés is automatikus megvalósítással történt.

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

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 figyelnem, mint például a felesleges meta-adatok leválasztása vagy a weboldalakon szereplő reklámok.

Adatbányászat: ekkor már a pontosan előkészített anyag áll rendelkezésre az adatbányászati eszközök futtatásához. A különböző módszerek segítségével típusproblémákra tudunk megoldásokat találni. A feladatomhoz használt konkrét eszközöket a vonatkozó fejezetekben fejtem ki.

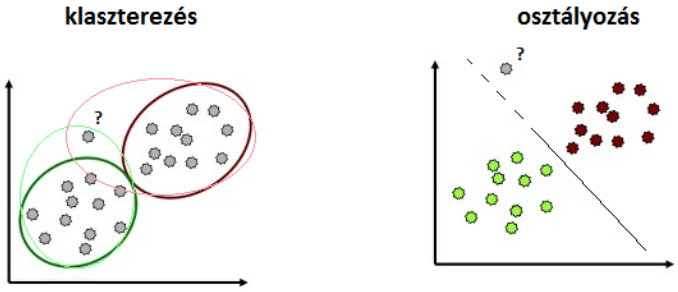
Megjelenítés, döntés: végül pedig kritikus az is, ahogy az eredményeket értelmezzük: nem szabad elfelejtenünk, hogy elsősorban a valóságra vetített igazságtartalmára vagyunk kíváncsiak az eredményeknek, nem csak a számokra. Könnyű a metrikákat félreértelmezni: elengedhetetlen, hogy ellenőrizzük és magasabb szintű (pl. üzleti) elemzéseknek megfelelően előkészítsük az eredményeket.

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

1. Csoportosítás / klaszterezés: az adatok feldolgozása során, egy fokozatosan

kialakuló szabályrendszer alapján daraboljuk fel az adathalmazt csoportokra. Ekkor a klaszterek határai nem ismertek, erről információt csak az algoritmus futásával párhuzamosan tudunk mi magunk kialakítani, nekünk kell "megtanulni" lépésről lépésre. A csoportosítási feladatokat szokás még felügyelet nélküli tanulási folyamatoknak hívni.

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 bekategorizálása. Ezt hívjuk felügyelt tanulási folyamatnak.



1.2. ábra. *Klaszterezés és osztályozás [1]*

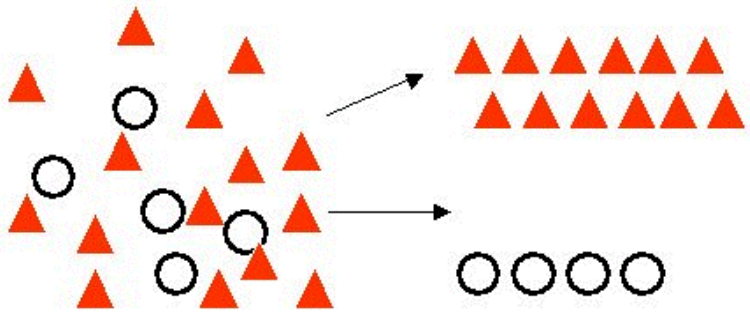
1.2.2. Adatbányászati módszerek: osztályozás

Az adatbányászat több különböző problémával is foglalkozik. Ezek közé tartoznak például az osztályozási módszerek, a besorolás, vagy a prediktív analízis. Feladatom során osztályozási feladatokkal is szembe kerültem. Az osztályozás [12] lényege, hogy előre definiált csoportokba kell valamilyen módszer alapján elhelyezni az újonnan felvitt adatokat. Erre több módszer is létezik.

Az osztályozás olyan összefüggések kutatása az újonnan a rendszerbe kerülő adat és a már kategorizált adathalmaz között, mely valamilyen hasonlóságot reprezentál. 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.

Az így meghatározott vektorokról már rendelkezünk egy bizonyos tudással, osztállyal: előző példánkat tekintve például legyen az osztályozás tárgya a sport- és



1.3. ábra. *Osztályozás [18]*

családi autók. Tételezzük fel 500 autóról, hogy a kettő közül mely kategóriákba tartozik bele mindegyik. Az így létrejövő csoportokba próbál meg az osztályozás egy új autót beilleszteni. Paramétereit hozzávetve a már meghatározott többi autóhoz megmondhatjuk az új elemről, hogy hol helyezkedik el a vektortérben, és nagy valószínűség szerint melyik osztályba tartozik.

Az osztályozási módszereket két csoportra szokás bontani:

- **Modell alapú 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, amikor egy nagyméretű, felcímkézett internetes cikkekből álló dokumentumtérhez érkező új elemekhez kellett címkéket rendelni. A 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.

1.2.3. Adatbányászati módszerek: klaszterezés

Dokumentumhalmaz rendszerezésének alternatívája az osztályozáson kívül a klaszterezés (csoportosítás).

Cél, hogy az egy csoportba kerülő elemek minél jobban hasonlítsanak egymásra, és a különböző csoportba kerülő elemek között minél nagyobb legyen az eltérés. Az osztályozáshoz képest a fő különbség, hogy nincsenek ismert osztályozású adatok és előre rögzített csoportszám sem.

Így a fő nehézséget az jelenti, hogy nincs olyan tanítóhalmaz, mint az osztályozás során a tanítókörnyezet. Emiatt a klaszterezés ún. felügyelet nélküli gépi tanuló

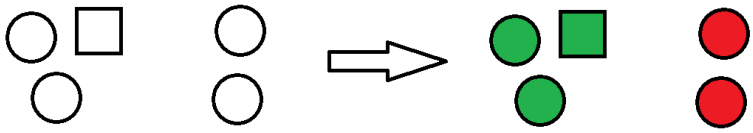
módszer [12].

A klaszterező eljárásoknak két alapvető típusa létezik:

1. **Particionáló módszerek**: segítségükkel a dokumentumok egy olyan felosztását adhatjuk meg, ahol az egyedek függetlenek egymástól. Tipikus particionálás a dokumentumokat nyelv, témakör vagy stílus alapján felosztani. Előnye, hogy jellemzően hatékony algoritmusok állnak rendelkezésünkre alacsony számításidővel, viszont hátrány a rugalmatlanság, mivel a csoportok számát bizonyos korlátozásokkal illetni kell, ahogy hamarosan látni fogjuk.

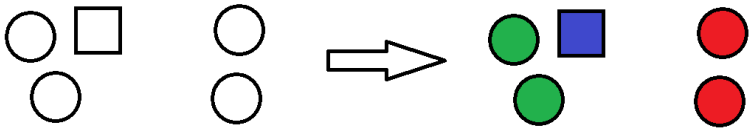
2. **Hierarchikus klaszterezés**: lehetséges csoportok egymása ágyazott sorozatát adják eredményül. Egy-egy elem itt tehát több csoportba is tartozhat. Így az algoritmus rugalmasabb, viszont cserébe jóval nagyobb számításigényű. Tipikus ilyen elemzés például a dokumentumok címkézése.

A klaszterezés alkalmazása során felmerülő alapvető probléma a csoportok várható méretének meghatározása. Például ha kétdimenziós klaszterezést várunk el az algoritmustól akkor hasonló megoldásra számítunk, mint ami az 1.4 ábrán látható.



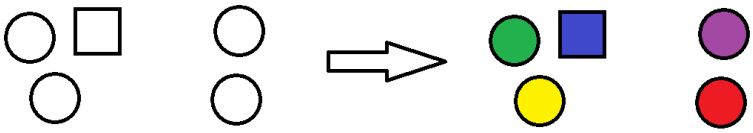
1.4. ábra. *Klaszterezés két dimenzió mentén*

Míg ha részletesebb elemzést, ezáltal magasabb csoportszámot is megengedünk, akkor új, pontosabb eredményre jutunk, ahogyan az 1.5 ábrán látszik.



1.5. ábra. *Klaszterezés több dimenzió mentén*

Viszont elméletben a legpontosabb klaszterezés a dokumentumtér elemszáma, melyben minden elem egyedi - ezt láthatjuk az 1.6 ábrán - ez viszont számunkra nem hordoz információt.



1.6. ábra. *Klaszterezés túl pontos kategória határokkal*

Így a klaszterező algoritmus paraméterezése az eredmények szubjektív hasznosságát megítélve a tervező feladata. A weboldalak kategorizálásánál egy saját klaszterező algoritmust implementáltam, melyben a vektortérbe a dokumentumokban szereplő "fontos" szavak alapján helyeztem el az elemeket. Fontosak azok a szavak, amelyek egy dokumentumra jellemzőek, de a többire nem (lásd: tf-idf elemzés később). Az elemzés során több maximum csoportszámmal is futtattam az eljárást, hogy ki tudjam választani az optimális legközelebbi csoportlétszámot.

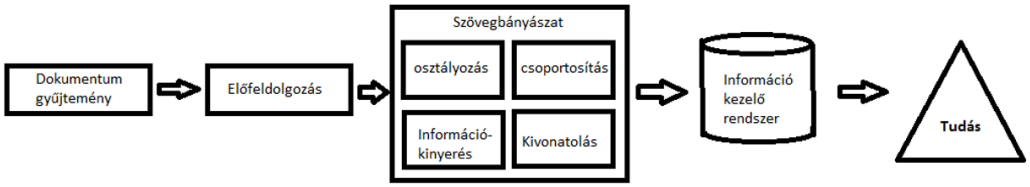
1.3. Szövegbányászat

Az adatbányászat és a szövegbányászat közötti fő különbség 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 ugyanazon szavak ragozott alakjai. A szövegbányászat során próbálunk ezen összefüggések közül minél többet megjelölni, felesleges információkat eltávolítani.

A szövegbányászat potenciálja abban rejlik, hogy az emberek által írt 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 legtöbb ilyen program nagyban támaszkodik a különféle szövegbányászati eszközökre a releváns találatok megjelenítéséhez [11].

A szövegbányászati elemzés általános lépései kissé eltérnek az adatbányászatétól, ahogyan ezt az 1.7 ábra mutatja.



1.7. ábra. *Szövegbányászat általános lépései*

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

Előfeldolgozás során olyan adattisztítást végzünk, amely segítségével egy, az adatbányászati lépéseknek is megfelelő adatstruktúrát 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 bujkál a szövegben, amely kihívást jelent számunkra.

A további lépések megegyeznek az adatbányászatban használt lépésekkel.

1.3.1. Előfeldolgozás

Feladatom során webes portálok szövegeit dolgoztam fel, több nyelven.

Mivel az egyes dokumentumok szavai alapján szerettem volna összehasonlítást végezni, különösen fontos volt, hogy csak olyan szavakat tároljak le, amelyek valóban jellemzik az adott szöveget. Ezt az igényt követve válogattam össze a feladat elvégzéséhez szükséges szövegbányászati eszköztárat.

Fogalmak

A Stop-szavak kiszűrése során egy előre definiált, többnyelvű stopszótárt [30] használtam. Ebben az írásban található leggyakoribb, nyelvtani és kötőszavakat gyűjtöttem össze. Amennyiben egy dokumentumban ráakadtam ilyen szavakra, azokat egyszerűen kitöröltem az adott dokumentumból. Ilyen szavak például az "az, és, vagy".

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

Vegyük a következő két mondatot:

- "Megint gólt lőtt Détári"
- "Sajnos újra lesérült Détári"

Ezekből a mondatokból a következő szótárt készítettem el a "megint" és az "újra" stop-szavak kiszűrése után:

"Gól, lő, Détári, lesérül".

Bag-of-words [25]: 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. Vegyük az előző példánkat: ebben az esetben a két mondatom Bag-of-words modellje a következő:

- (1,1,1,0) - ("Megint gólt lőtt Détári")
- (0,0,1,1) - ("Sajnos újra lesérült Détári")

Corpus: a már elkészült szótár segítségével, bag-of-words által súlyozott, vektorizált dokumentumtér összessége. A kész corpus-on definiáljuk a matematikai- és adatbányászati elemzéseket. A fenti példánk esetén az ott szereplő két, egyébként négydimenziós vektor alkotja a corpus-t.

Topic modelling a szövegbányászat egy gyakran használt eszköze. Ez egy statisztikai modellezési [16] eljárás, mely során kiemeljük a dokumentum tartalmának néhány szava által behatárolt központi elemeit, témáit. Például, ha egy szövegben gyakran előfordulnak a sportban használt kifejezések, akkor valószínűleg a központi témája a sport rovat. Az így kialakult központi témák segítségével hasonlítottam össze a dokumentumokat.

Term Frequency-inverse Document Frequency (tf-idf): ez egy olyan statisztikai érték [29], mely meg tudja határozni, az egész corpus-t figyelembe véve, hogy egy-egy szó mennyire befolyásolja egy adott dokumentum végső kategóriáját. Ha egyes szavak leginkább ugyanabban a kategóriában szereplő dokumentumokban fordulnak elő, akkor nagyobb ez az érték, míg ha több kategóriában is előfordulhatnak, akkor kisebb (hiszen bizonytalan a hatásuk a végső kategória-ítéletre). Például a sportolók nevei magasabb, míg pl. a stadion szó alacsonyabb értéket kap a sportrovat besorolás ítéletében egy mai hírportálon, mivel ez utóbbi szó gyakran jelenik meg belföldi, politikai hírekben is.

Latent semantic indexing (LSI) [16]: nagyméretű dokumentumok esetén 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.

Hasonlósági elemzés: A fenti vektor reprezentációk, súlyozások, transzformációk célja, hogy egy olyan matematikai modellt építhessünk fel, melyen értelmezhetőek az úgynevezett Hasonlósági eljárások [27]. Ezek célja, hogy az előállított dokumentumtéren számosítsa a különböző dokumentumok közötti tartalmi kapcsolatot, megállapítsa valamely formában, hogy a dokumentumtéren egy elem mely más dokumentumokhoz helyezkedik el "közel" és melyektől áll "messze". Munkám során a más említett Cosinus hasonlósági elemzést használtam fel a távolságok megállapítására.

2. fejezet

Felhasznált webes technológiák

A feladatom során nagyban építkeztem a korábbi önálló laboratórium során megismert technológiákra. Segítségükkel bővítettem a korábbi megoldást, és implementáltam új eljárásokat, melyhez szükségem volt új technológiák megismerésére.

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[6] lett.

2.1. Python

A Python[28] egy általános célú, magas szintű programozási nyelv. Elterjedésének okai többek között a jó felhasználhatóság, jó futásidő-optimalizáció, elegáns megoldások és olvasható kódkép. Én a Python 3.3.3-as verzióját használtam a feladat megvalósítására.

A nyelv objektumorientált, ingyenesen használható, működik a legtöbb modern operációs rendszer felett. Scriptek írásától a komplex, többrétegű alkalmazások implementálásáig használható.

Kódképe tömör és jól olvasható: szokásos programozási konvencióktól több helyen is eltér az egyszerűség és átláthatóság céljából. A kódban a sortörés kiváltja a pontosvesszők használatát az utasítások lezárásához, illetve a kódblokkok elválasztó elemei a bekezdések, a szokásos kapcsos zárójelek helyett.

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

2.1. lista. *A Fibonacci számok kiszámolása Python nyelven*

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

2.2. Webalkalmazások

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

A szoftver felhasználó felülete így generált 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 [26] felett különböző csomagokat küld és fogad. A kéréseknek két kategóriáját használtam fel:

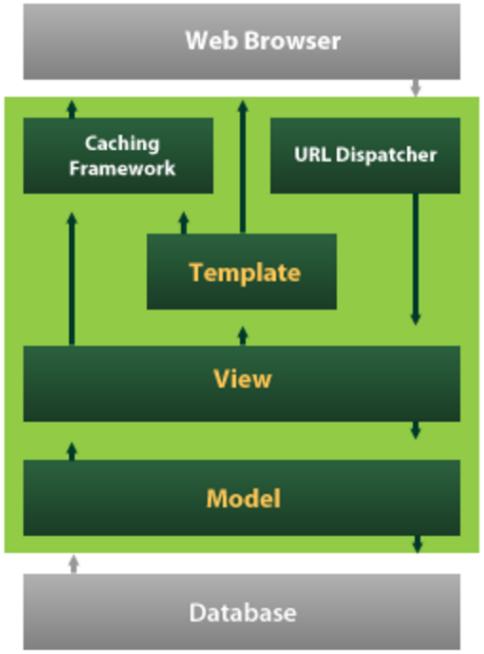
- **GET**: a kliens oldaláról érkező http request csomag, mely valamilyen adat lekérdezésére irányul legtöbbször (RESTful[23] web service esetében). Például az egyes linkekre való kattintás, vagy gombok lenyomása.
- **POST**: a kliens oldaláról érkező, szintén http request típus, amely valamilyen adatot küld feldolgozás céljára. A felhasználó a feladatom során, a webes felületen adhatja meg azt a szöveget, linkeket, melyekre szeretné futtatni az elemzési eljárásokat. Az információt egy POST üzenetben küldi el a szerver számára, mely azt továbbítja az adatbányászati modul felé.

2.3. Django

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

Az alkalmazás Model-View-Controller [21] 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 így képes webes kérések kiszolgálására, webservicek [31] üzemeltetésére.

A 2.1 ábrán látható a szoftver általános szerkezete. Az adatbázis és a model közötti szinkronizáció automatikus. A view réteg dolgozza fel a beérkező kéréseket,



2.1. ábra. *Django architektúra[17]*

szólítja meg a belső üzleti logikát (ami a mostani esetben az adatbányászati modul függvényei), kommunikál az adatbázissal. Ebből a hármasból 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. Az alkalmazás vázának létrehozásakor már képesek vagyunk elindítani az webszervert, kiszolgálni a kéréseket, adminisztrátori felületen belépni, felhasználókat menedzselni: tehát a leggyakoribb feladatokhoz megkapjuk a keretet.

2. Model definiálás. A keretrendszer segítségével előre meghatározott módon tudunk model objektumokat létrehozni, melyekek előre generált megjelenítő felületekkel rendelkeznek. A feladatom során olyan dokumentum objektumokat hoztam létre, melyek ilyen módon már előre rendelkeztek a hozzájuk tartozó megfelelő model form-mal, és igény szerint akár adatbázis perzisztálással is.

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 és 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 felületes áttekintése a keretrendszerben rejlő lehetőségeknek: a feladatom megoldásához az itt felsorolt lehetőségek elegendőnek is bizonyultak az adatbányászati modul felhasználásához, annak megjelenítéséhez.

2.4. Beautiful Soup

A Beautiful Soup [19] egy Python libary, amit web parse feladatok(azaz weboldal tartalom kinyerésének) könnyű megvalósításához terveztek.

Fő feladata, hogy 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 bejárási eszközökkel is rendelkezik, tehát ezt nem nekünk kell megvalósítani. Ez utóbbi tulajdonsága egyes dokumentumfa bejárási feladatoknál nagyon hasznos.

Az előzetes kutatás során ezt az eszközt használtam többek között a weboldalak, és azok RSS feed-jének [9] lekérésére, ebből a linkek kiválogatására. A linkek mögött rejlő weboldalak szövegét is ennek az eszköznek a segítségével értem el. Például egy weboldalra mutató url-ből a 2.2 kódrészleten látható módon tudjuk kinyerni a szöveg címét.

2.2. lista. *URL parse-olása*

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

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

2.5. Gensim

A gensim [22] egy topic modellezést megvalósító Python programcsomag. Open source projektként a fő feladata, hogy Python nyelven egy könnyen, 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]
```

2.2. ábra. *Gensim képességek*

A szövegbányászat előkészítési és adatbányászati lépéseit hivatott megvalósítani. 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.

2.5.1. Gensim funkciók

A gensim a procedurális feldolgozás jegyében a korábban ismertetett adat- és szövegbányászati lépéseket a következő funkciók segítségével valósítja 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 az említett vektorok összességét (a corpust) 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 transzformációkat, 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 dokumentum témaszomszédjait.

3. fejezet

Előzetes kutatás: szövegelemzés

Ahogy általánosságban is jellemző az adatbányászatra, 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 volt, hogy úgy lássak hozzá a munkához, hogy van egy elképzelésem arról, milyen elemzések eredményeit fogom látni, a felhasznált módszereknek milyen sajátosságaira kell ügyelni, közülük melyeket kell választanom.

A diplomamunkám témájához kapcsolódó önálló munkáim eredményeire építve készült el a projekt. A saját kutatásaim eredményének felhasználásával alakítottam ki az alkalmazás végleges felépítését, ezek mentén választottam ki a felhasznált eljárásokat. Az alkalmazás végleges céljai és feladata így folyamatosan alakult, ahogyan az azok eléréséhez szükséges technológiákat és eszközöket kipróbáltam, teszteltem.

A legelső ilyen kutatásom a szövegelemzés és a címkézhetőség vizsgálata volt.

3.1. Szövegelemzés és címkék

Az automatikus webelemzés implementálása előtt fontos volt megállapítani: magyar nyelvű webes tartalomra lehet-e egyáltalán szövegbányászati elemzéseket futtatni, és ha igen, milyen hatékonysággal?

A kérdés megválaszolása során előkészített, internetes hírportálokról kézzel letöltött szövegek között folytattam elemzéseket. A fő feladatom az volt, hogy néhány cikk beolvasása után, az újonnan érkező dokumentumhoz javasoljak olyan címkét, amit az alapján állítok elő, a dokumentum szövegtörzse melyik másik dokumentummal lehet közös témájú. Ezt magyar nyelven, és néhány jól elkülöníthető tanító szöveggel sikerült sikeresen implementálni.

Láthatunk a 3.1 ábrán egy beolvasott példadokumentumot, alatta a címkéket amiket megadott hozzá a szerző, és a "|" jel után azt a címkét, amit az alkalmazás javasol.

3.1. ábra. *Címkéző app*

A feladat során kézzel kellett (egy webes felületen keresztül) betáplálni a szövegeket, és a tanítóhalmazt is az osztályozási módszerekhez.

3.1.1. Eredmények

Az önálló laboratórium tárgyam során elvégzett feladatok alapján a felhasznált webes keretrendszer, az elemzési módszerek valamint szöveg- és adatbányászati függvénykönyvtárak működőképesnek bizonyultak.

3.2. Dokumentum osztályozás

Az előző pontban említett eredményekre alapozva a következő lépés az volt, már a diplomamunkám keretein belül, hogy egy bonyolultabb elemzést hozzak létre, mely, amennyiben megvalósítható, megnyitja az utat a végleges munkám elkészítéséhez.

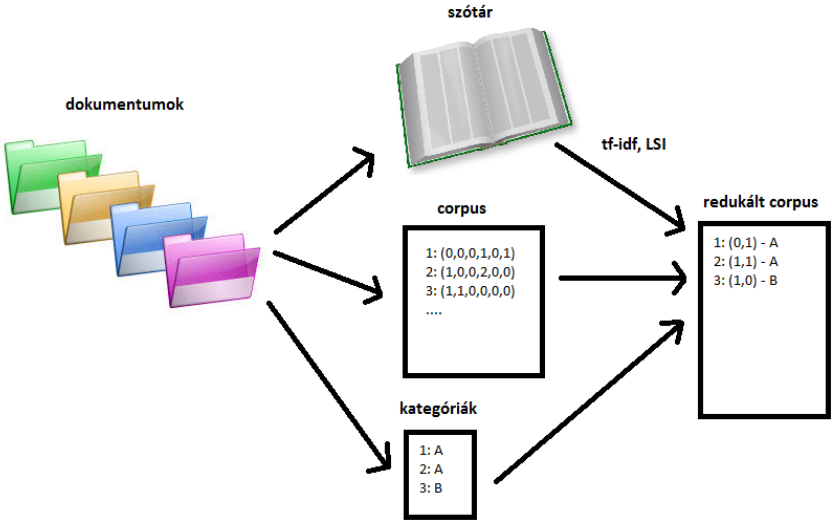
Fontos tapasztalat volt a korábbi kutatásom során, hogy lehetőleg érdemes minél egyszerűbb elemzésekkel kezdenem, és csak ha az alapgondolat működik, akkor lássak neki a feladat és az algoritmusok komplexitásának növeléséhez. Ezek mellett figyelnem kellett arra, hogy hibás eredmények esetén alaposan körüljárjam, mit is várok el az algoritmustól, miért nem az az eredmény, amit várok, és hol lehet azt esetleg javítani.

Az alkalmazás funkcionalitását kibővítettem azzal, hogy az egyszerű címkék ajánlásán túl, a beérkező dokumentumokat képes volt kategorizálni is adott előzetesen kialakított kategóriák mentén (osztályozás).

A feladat során az index.hu-n található, 8 főbb rovatából, kategóriánként 50-50 cikket olvastam fel.

3.2.1. Tervezés

Megfelelő mennyiségű betanító dokumentum után az újonnan érkező szövegtörzset - legyen az a webportál egy eddig még fel nem dolgozott cikke, vagy a szerző által a webes felületen begépelt szöveg - elemezte, és bekategorizálta nyolc témakör egyikébe.



3.2. ábra. *Dokumentum osztályozás vázlatos terve*

A 3.2 képen látható az a felületes terv, hogy hogyan valósítottam meg ezt a feladatot.

Elsőként az összegyűjtött dokumentumokból kialakítottam a szótárt, a corpus-t, és a kategóriákat. A szavak súlyozása (tf-idf) és a dokumentumtér dimenziójának csökkentése (tf-idf) után redukált corpus-t hoztam létre, ahol mindegyik dokumentumhoz eltároltam 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 használtam fel.

3.2.2. Implementálás

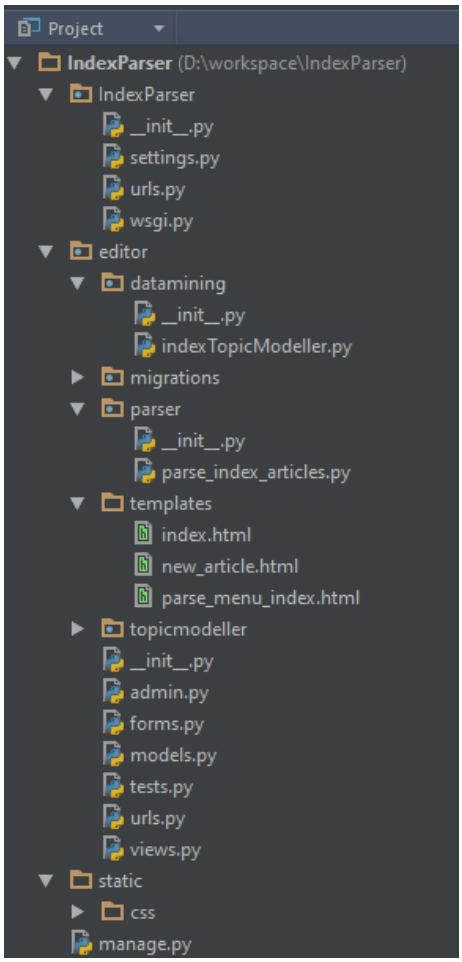
Ebben a fejezetben részletesen bemutatom, hogy milyen lépéseken keresztül valósítottam meg a dokumentum osztályozást.

Webes megjelenítő rész

A Django keretrendszernek hála könnyedén tudtam működőképes weboldalt létrehozni. A hivatalos tutorial-t követve [6] könnyedén működő weboldalt tudunk

összeállítani, ennek felhasználásával haladtam én is. Célom a keretrendszerrel a <link>tervezés- előzetes eredmények fázis alatt az volt, hogy a legalapvetőbb funkciókat megvalósítsa (webszerver kiszolgálja a kéréseket, megjeleníti az algoritmusok eredményét, nyers HTML tartalmat mutat, stb.), a kényelmi funkcionalitásokat még nem implementáltam ekkor.

Alkalmazás architektúra



3.3. ábra. *A Dokumentum osztályozó alkalmazás szerkezete*

Az alkalmazás a következő modulokból áll:

IndexParser: a csomagban található fájlok segítségével konfigurálható a webserver.

Editor/datamining: ebben a csomagban található az az osztály, 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 templatejei. Kezdetben egy kezdőoldal, egy parser felület, és egy új cikk megadására szolgáló szerkesztő ablak volt található itt.

3.1. lista. *Egy példa template*

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

A 3.1 kódrészleten egy egyszerű template kód látható: megfigyelhető, 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: ebben a form típusú objektumokat definiáltam.

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

3.2. lista. *Egy példa Django model*

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

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

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

Editor/views.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 adatbányászati függvényeket, mielőtt legeneráljuk a kimenetet.

Static/css: stíluslapok.

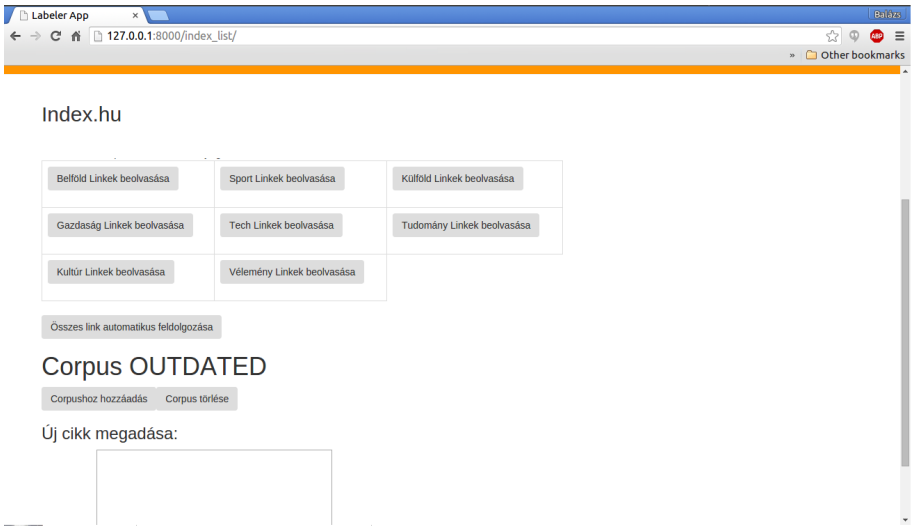
A kezdőoldal a 3.4 ábrán látható.

Parser modul

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



3.4. ábra. *A Dokumentum osztályozó alkalmazás felhasználói felülete*

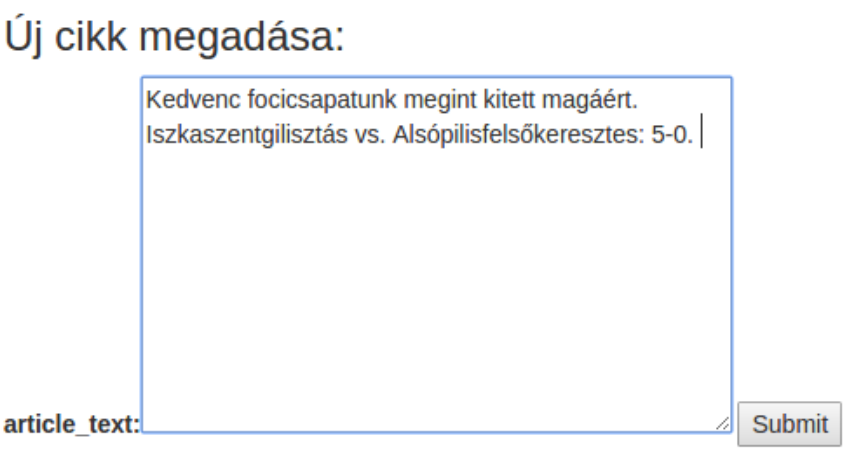
Minden weboldal mellé eltároljuk az adott cikk kategóriáját is a fájl első sorában.

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.

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 Submit gombra.



3.5. ábra. *Új cikk megadása felület*

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

3.3. lista. *POST kérés feldolgozása*

```
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})
```

3. A függvény megnyitja a letárolt corpus-t, szótárat és a kategóriák listáját.

4. A beérkezett szöveget tokenizálja.

5. A szövegből kiszűri a stop-szavakat

3.4. lista. *Stopszó szűrés és tokenizálás*

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

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

3.5. lista. *Szótár bővítése*

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

7. Transzformálja a szöveget vektortérbe.

3.6. lista. *Vektortér transzformáció*

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

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

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

3.7. lista. *LSI transzformáció*

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

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

3.8. lista. *Corpus transzformáció*

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

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

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

3.9. lista. *Hasonlósági elemzés*

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

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

13. Letárolja az adatokat a merevlemezre

3.6. ábra. *Az eljárás kimenete*

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

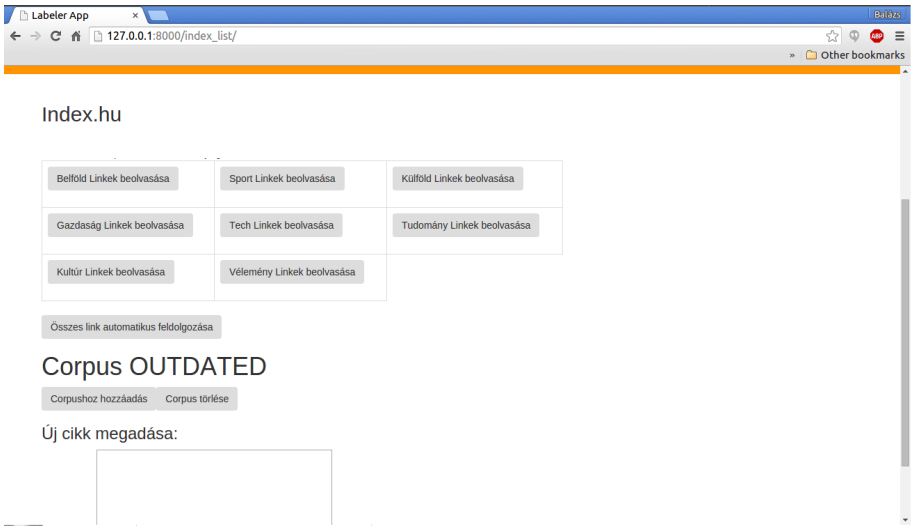
3.2.3. Eredmények

A feladat 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.

Előre kinyert, kézzel kimásolt cikkeket olvasott fel .txt formátumú fájlokból az alkalmazás.

Tesztelés során olyan, a tanulási folyamathoz nem használt cikkek szövegeit adtam meg, amelyek kategóriáját előre tudtam az index.hu-ról. Az eredmények igen bíztatóak. Elsőként csak két kategóriát teszteltem egymás ellen, így a jól elkülönülő kategóriák (pl. Tech és Belföld) között a szintén jól szövegezett dokumentumokat általában jól kategorizálta be az alkalmazás.

A kiszélesített, összesen nyolc témakört felölelő elemzésben már előfordultak pontatlanságok, ezek többsége azonban abból fakadt, hogy a cikk témája nem volt egyértelműen körülhatárolható, vagy a cikk szövegezése volt olyan, ahol félreérthető volt



3.7. ábra. *A kategóriák*

a tartalom (pl. nemzetközi focicsapat meccsének elemzése Külföld, politikai témájú cikk pedig Sport kategóriába került bele a témaátfedés és a szakzsargon miatt).

Összességében az eredmény, hogy igenis használhatóak a szövegbányászat eszközei az ilyen jellegű feladatok megvalósítására.

4. fejezet

Dokumentumbányászati tervezés

Az előzetes kutatás sikeres eredményeinek megállapítása után következhetett a komplex dokumentumelemzési eljárás implementálása.

Szövegbányászati elemzések típusfeladatit többek között az alábbi két fő csoportra is feloszthatjuk:

1. Ismeretlen dokumentumok szövegbányászata
2. Ismert dokumentumok szövegbányászata

Az ismeretlen dokumentumterek feltérképezése során általános megoldásokat kell használnunk, és célunk, hogy a rendelkezésre álló kevés információ segítségével a lehető legtöbb hasznos információt állapítsuk meg - nem várjuk el, hogy az emberi értelmezéshez hasonló adaptív algoritmust kapjunk.

Ismert dokumentumterek elemzése azzal az előnnyel jár, hogy a dokumentumok szerkezeti, tartalmi és egyéb sajátosságai mentén sokkal specifikusabb algoritmusokat írhatunk, melyek segítségével sokkal részletesebb kutatásokat végezhetünk.

A diplomám írása során mindkét feladattípusból választottam egyet-egyet, és ezekre kerestem a választ:

1. Ismeretlen dokumentumtér elemzés: nagyméretű, ismeretlen html weboldalak tartalmának összehasonlítása, kategorizálása.
2. Ismert dokumentumok elemzése: előre kiválasztott internetes portálok specifkus tartalmának rekurzív kinyerése, főbb témaköreinek megállapítása.

A Szövegelemzés és Dokumentum osztályozás fejezetekben kapott eredmények alapján következhetett a konkrét feladat megtervezése. Tapasztalataimmal felvértezve, a következő fejlesztéseket eszközöltem a webalkalmazásban:

4.1. Webalkalmazás kiegészítése és új eszközök

A Django webalkalmazás bevált, de működése kezdetleges. Ezért szükséges, hogy az alapvető, modern webalkalmazásoktól elvárt működést megvalósítsa a program. Ezek az adatbányászati modul működését érdemben nem befolyásolják, de a felhasználói élményt nagyban növelik.

4.1.1. Biztonság

A webalkalamazásom jelenleg nem valósít meg felhasználókezelést. Ezt orvosolandó, létrehoztam a felhasználókezelési modult, implementáltam a regisztráció és bejelentkezés funkciókat, és az oldalt így csak regisztrált felhasználók számára tettem elérhetővé. Ehhez az alapértelmezett Django belső adatbázist használtam fel, illetve a login_required dekorátort [7].

A dekorátorok a webalkalmzás oldalaihoz való hozzáférést korlátozzák. Ha egy dekorátor feltétele nem teljesül a http kérés beérkeztekor, akkor a dekorált weboldal helyett tetszőleges hibaüzenetet küldhetünk a válaszban.

4.1. lista. *Login dekorátor példa*

```
@login_required
def url_multi_parse(request):
        if request.method == "POST":
                links = getLinks(request.POST["linkTexts"])
```

A 4.1 kódrészletben láthatjuk, hogy a @Login_required dekorátor védi a lekérések végpontjait.

4.2. lista. *Template autentikáció render példa*

```
{% if usenipr.is_authenticated %}

<div class="col-md-4">
<!--weboldal tartalma itt -->
</div>

{% else %}
<a href="{% url 'login' %}"><span class="glyphicon glyphicon-lock"></span></a>
{% endif %}
```

Ebben a kódrészletben a HTML template van felkészítve a be nem lépett felhasználók kezelésére: ebben az esetben a weboldal tartalma megváltozik, a login oldalra mutatunk linket a felhasználónak.

A felhasználók adatait egy, a szerver memóriájában tárolt H2-es <link> adatbázisban tároltam.

4.2. Webcrawler

Az egyszerűbb webtartalom kinyerésére a korábban ismertetett BeautifulSoup megfelel. Azonban ahhoz, hogy akár több tízezer weboldalt végig tudjak iterálni, nem alkalmas: csak előre megadott címeket tud kinyerni, főként a DOM-ot [32] értelmezve, dinamikus feltételek nélkül, és ami a legnagyobb hátránya: szekvenciálisan.

Szükségem volt egy olyan komponensre, amely:

- képest több ezer weboldal párhuzamos bejárására
- képes dinamikusan tartalmat keresni a dokumentumokban
- képes feltételek, elágazások implementálása
- képes rekurzívan lehívni linkeket a meglátogatott weboldalakról, és azokat is bejárni, ezeket a gazda weboldal tartalmához hozzácsatolni

Írnom kellett tehát egy webcrawler-t, ami egy önálló ágensként képes webtartalom akár rekurzív lehívására.

Ha egy-egy ilyen ágens képes önállóan feldolgozni megadott webcímeket, akkor őket párhuzamosítva felgyorsíthatom a működését az alkalmazásomnak.

4.2.1. Scrapy

A feladat megvalósításához a Scrapy eszközt [3] választottam.

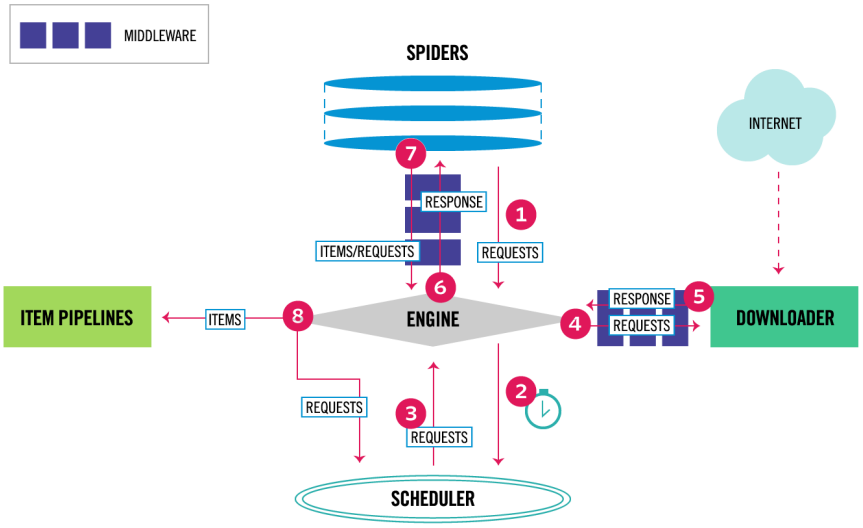
A Scrapy egy open source, nagyon gyors és jól skálázható python webcrawler keretrendszer. Arra találták ki, hogy gyorsan és egyszerűen lehessen olyan webcrawlert írni, amely tetszőleges adatokat le tud kérdezni weboldalakról, emellett pedig megőrzi a magas fokú testreszabhatóságot és fejlesztői szabadságot.

Az alkalmazás úgynevezett "Spider"-eket definiál: ezek azok az osztályok, amely a webtartalom kinyerését végzik. Őket futtatja az alkalmazás Engine, és ő rajta keresztül érik el az internetes tartalmat is.

Az Engine által futtatott Spider-ek "Crawl" kérést küldenek az Engine-nek egy megadott URL címmel. Ezt a kérést dolgozza fel az Engine, küldi tovább a Downloadernek, aki pedig az idővel letöltött weboldalt továbbítja az Engine-en keresztül a Spider-nek feldolgozásra.

Párhuzamos futtatás esetén szükséges a Scheduler (ütemező) használata, hogy egyik Spider-t se éheztessük ki, lévén hogy a Downloader véges erőforrással gazdálkodik. A kapott HTML tartalom feldolgozását az éppen futó Spiderek konkurens módon végezhetik.

Egy példa Spider a 4.7 kódrészetben látható.



4.1. ábra. *Scrapy belső működés folyamata*

4.3. lista. *Scrapy példa*

```
import scrapy

class QuotesSpider(scrapy.Spider):
name = "quotes"

def start_requests(self):
        urls = [
                'http://quotes.toscrape.com/page/1/',
                'http://quotes.toscrape.com/page/2/',
        ]

        for url in urls:
                yield scrapy.Request(url=url, callback=self.parse)

def parse(self, response):
        page = response.url.split("/")[-2]
        filename = 'quotes-%s.html' % page
        with open(filename, 'wb') as f:
                f.write(response.body)
        self.log('Saved file %s' % filename)
```

Egy Spider-nek az Engine-nel való effektív kooperáció érdekében előre meghatározott interfésznek kell megfelelnie.

- Tartalmaznia kell egy **name** paramétert: ez határozza meg egyértelműen a Spider-t. Egy projekten belül egyedinek kell lennie - indítás előtt a fejlesztő felelőssége ugyanabból az osztálypéldányból példányosított Spider-ek elnevezése.

- definiálnia kell egy **start_request** függvényt: ennek feladata, hogy egy iterálható Request objektumokból álló listát adjon vissza, amit az Engine a Downloader-nek adhat, és amire válaszként érkező HTML tartalmat a Spider bejárhat. Szintén a fejlesztő felelőssége "megetetni" a Spidert indítás előtt

ezzel az adattal.

- definiálnia kell egy **parse()** metódust: ez a Spider legfontosabb függvénye. A Spider számára érkező HTML(tehát szöveges) tartalmakkal az Engine ezt a függvényt hívja meg - itt kell lekezelnünk és kinyernünk az adatokat a weboldalból.

A példánk során a parse-olás abból áll, hogy a kiinduló két weboldalt lekérdezi, majd tartalmukat két fájlban, "quotes-1.html" és "quotes-2.html"-ben lementi.

Egy Spider futattására két lehetőségünk van:

1. létrehozunk egy Scrapy projektet, és ebbe elhelyezve a Spider-eket, a projekt gyökérkönyvtárában kiadhatjuk a következő parancssoros parancsot: scrapy crawl <ami nevet megadtunk>

2. Másik lehetőség, hogy importáljuk modulként, és az alkalmazáson belül alakítjuk ki a Scrapy projektünket. Ekkor több kompatibilitási problémát is meg kell oldanunk, cserébe az alkalmazással sokkal szorosabb módon működik együtt az eszköz.

Tartalom kinyerése

A Scrapy segítségével egy html dokumentumból két fő módon nyerhetünk ki információt:

1. CSS selector-ok [20] segítségével, mint például a 4.4 kódrészletben. itt a <title> html elemből nyerhetjük ki a tisztán szöveges tartalmat.

4.4. lista. *Példa CSS selector*

```
>>> response.css('title::text').extract_first()
```

2. XPath kifejezésekkel, mint a 4.5 kódrészetlben.

4.5. lista. *Ugyanaz a példa XPath selector segítségével*

```
>>> response.css('title::text').extract_first()
```

Rekurzív eljárás

A Scrapy igazi erőssége a rekurzív tartalomfeldolgozásban rejlik. A kezdetben megadott "start_request" által generált listán kívül, a parse() függvény futása során is van lehetőségünk újabb lekéréseket indítani, más függvényeket meghívni, tartalmat összefűzni. Legegyszerűbb ennek használatát egy példán keresztül megnézni:

4.6. lista. *Példa rekurzív Scrapy eljárásra*

```
class QuotesSpider(scrapy.Spider):
name = "quotes"
start_urls = [
        'http://quotes.toscrape.com/page/1/',
]

def parse(self, response):
        for quote in response.css('div.quote'):
                yield {
                        'text': quote.css('span.text::text').extract_first(),
                        'author': quote.css('small.author::text').extract_first(),
                        'tags': quote.css('div.tags a.tag::text').extract(),
                }

        next_page = response.css('li.next a::attr(href)').extract_first()
        if next_page is not None:
                next_page = response.urljoin(next_page)
                yield scrapy.Request(next_page, callback=self.parse)
```

A példaweboldal felépítése a következő: egy oldalon egymás alatt felsorolva találhatóak elemek. Ezek tartalmaznak egy-egy idézetet, melynek van szerzője, címkéje, szövege. A weboldal alján található egy link a következő oldalra:

4.7. lista. *"Példa link"*

```
        <li class="next">
                <a href="/page/2/">Next <span aria-hidden="true">&rarr;</span></a>
        </li>
```

Ezt az oldalt nyeri ki a "next_page" változóba a függvény. Ha ez a query eredményt hoz, akkor a Spider egy teljes URL-t épít az ott szereplő linkből, és egy új Request-et ad vissza, mely a következő oldalra mutat - ebbe beregisztrálja magát, mint callback, és a kért oldal feldolgozásának végeztével hozzáfűzi a saját tartalmához. Így amíg ez meg nem történik, tudja folytatni a feldolgozást, vagy várakozhat a hívásra.

4.3. JSON

A JSON egy speciális adatformátum [4]. Jelentése: JavaScript Object Notation. Szöveges fájlformátum, első sorban adattárolásra és továbbításra.

Számunkra azért fontos, mert egyrészt a Python natívan támogatja a JSON fájlok feldolgozását, ami így jóval gyorsabb, másrészt, formátuma jóval letisztultabb az XML-hez képest. XML-ben nehezebb adatot tárolni és olvasni, több benne a nem hasznos adat, és kevésbé illeszkedik a modern alkalmazások belső adatszerkezetéhez. például az következő XML:

```
<employees>
        <employee>
        <first  Name>John</firstName> <lastName>Doe</lastName>
        </employee>
</employees>
```

JSON formában ennyi:

```
{"employees":[
        { "firstName":"John", "lastName":"Doe" },
]}
```

Adatbányászati elemzéseknél a JSON-t ajánlják a legtöbb helyen [10]. Mindezek mellett szeretném hangsúlyozni, hogy a JSON nem "jobb" mint az XML, egyszerűen szituáció függő, melyik bizonyul a hatékonyabbnak, jelen esetben a natív Python támogatás miatt döntöttem mellette.

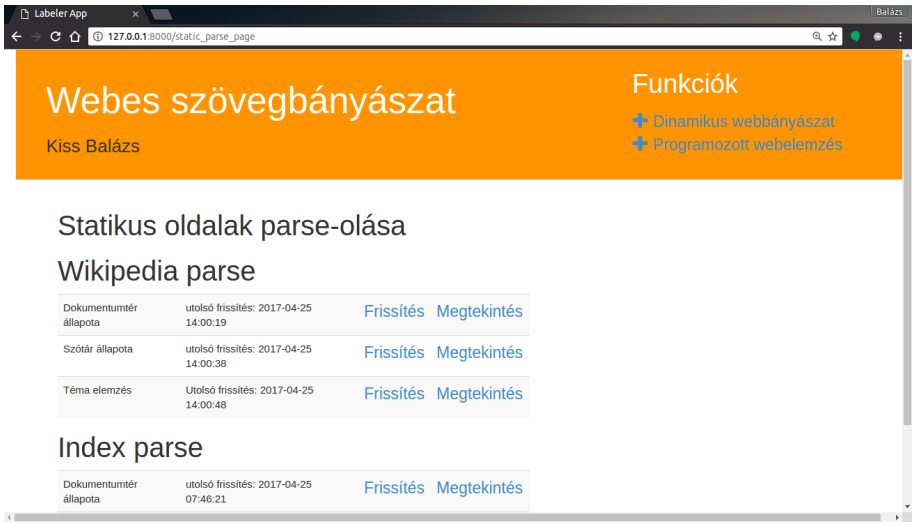
4.4. Langdetect

A Langdetect 1.0.7 [2] egy Python függvénykönyvtár, mely a Google nyelvfelismerő algoritmusát portolja át. Segítségével az osztálynak megadott szöveges függvény-paraméterről megpróbálja eldönteni, mely nyelvbe tartozhat. Összesen 55 nyelvet támogat.

Az algoritmus [24], így a túl kicsi, illetve túl "szemetes" (pl. scriptek, html tag-ek, vegyes nyelvű) szövegek esetén más-más eredményt adhat. Jelenleg azonban megfigyeléseim alapján nagyon pontos a feladatomban felmerülő dokumentumok nyelvének megállapítása, arról nem is beszélve, hogy ez a ma elérhető, és a jelenlegi technológiai korlátok ellenére létező legjobb eszköz.

4.5. Webes felület

Az alkalmazás felületét két menü részre osztottam fel, ahogyan az a 4.2 ábrán is látható.



4.2. ábra. *Az alkalmazás felülete*

Webalkalmazásom, megnyitva a Programozott webelemzés modult, ahogy böngészőben megjelenik.

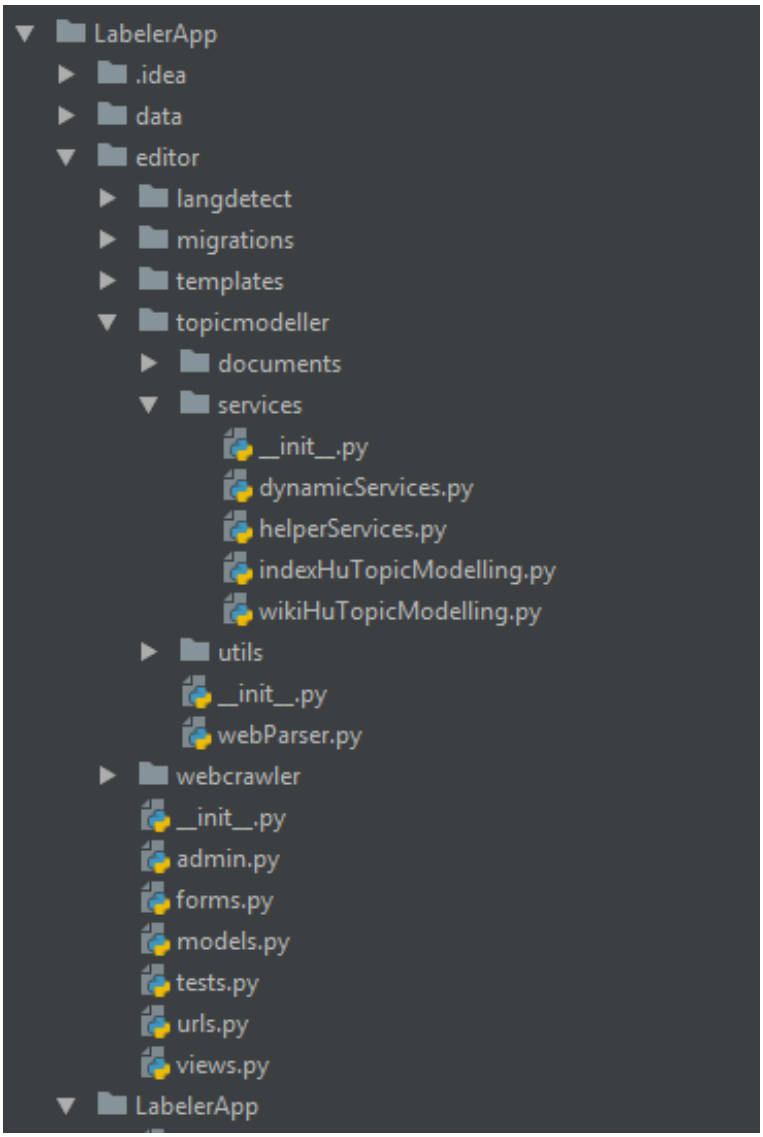
A Dinamikus webbányászat funkció valósítja meg az nagy mennyiségű weboldalak adatbányászatát.

A Programozott webelemzés funkció felel az előre felkészített webtartomány részletesebb elemzéséért.

Az új funkciók implementálásához szintúgy a Django keretrendszert használtam fel, mint a korábbi feladatok során. Az alkalmazást összesen több mint 20 új oldallal egészítettem ki.

Új modulok:

- **data**: itt tárolom el az elemzések során keletkező adathalmazokat JSON fájlformátumban.
- **editor/langdetect**: a nyelv detektálásához szükséges modulok
- **editor/topicmodeller/documents**: a Scrapy által előállított dokumentumokat tárolom itt.
- **editor/topicmodeller/services**: itt tároltam az adat- és szövegbányászati modulokat
- **editor/topicmodeller/utils**: segédosztályok és segédfüggvények, pl. az alkalmazás által használt elérési utak konstansai, string kezelő algoritmusok, leggyakoribb szó kiválasztása, stb.
- **editor/webcrawler**: a Scrapy által használt könyvtár, benne a Scrapy motor, és az általam írt Spider-ek. Ezen belül itt tároltam még a futtatások beállításához szükséges szkripteket is.



4.3. ábra. *Végleges alkalmazás felépítése*

5. fejezet

Ismeretlen dokumentumtér: Webtérképezés

A feladat során rendelkezésemre bocsátottak egy 700.000 webcímet tartalmazó .csv fájlt. Itt minden URL-hez meg volt adva egy vagy több kategória, ami alapján a weboldal tartalmát már egyéb módszerek segítségével jellemezték. Összesen több mint ötven kategóriára osztotta fel a címeket a fájl, pl.: oktatás, webáruház, közösségi oldal, híroldal, tárhely, stb. Ezt használtam fel az általam előállított kategóriák ellenőrzésére.

5.1. Paraméterek

Az elemzés során a következő információkat szerettem volna megtudni egy weboldalról:

- Mik a leggyakoribb, jól elkülöníthető témakörök?
- Hány témakörre lehet hatékonyan bekategorizálni a weboldalakat?
- Milyen nyelvűek az oldalak?
- Kinyerhetőek-e a weboldal tartalmára legjellemzőbb kifejezések?

5.2. Funkciók tervezése és megvalósítás

A tervezés fázisban a már ismertetett módszereket vettem alapul. A webelemzés során a következő lépéseket terveztem meg, majd implementáltam:

1. Fázis: **linkek formázása**: ebben a fázisban a felhasználó által megadott URL címeket rendezem össze, és tárolom el olyan formában, hogy a webcrawler-em képes legyen azokat bejárni. A linkek megadása során egy szöveges beviteli

mezőbe írhatóak be az linkek. Ezt az alkalmazás dinamikusan formázza, majd pedig JSON objektumként lementi.

2. Fázis: **Dokumentumtér kialakítása**: ebben a fázisban indítom el az URL lista bejárását a megírt Spider-rel. Segítségével elkülönített dokumentumokat hozok létre, melyek tartalmazzák az adott internetes címek mögött rejlő weboldalak szöveges tartalmát.

3. Fázis: **Webelemzés**: ebben a fázisban történik meg a Topic-modelling eljárás futtatása, mely során megpróbálom összehasonlítani a kiolvasott weboldalak tartalmát, és Klaszterezési eljárás segítségével kategóriákra bontom őket.

5.2.1. Dokumentumtér kialakítása

A dokumentumtér elkészítésére az általam írt Spider-t használtam fel. Ez az alkalmazás leginkább időigényes lépése. A futást megkönnyítendő, egyszerre több példány is futtatható, felosztva közöttük a bejárandó webtartományt. A Spider-t úgy konfiguráltam fel, hogy hibás vagy nem létező weboldal esetén ne próbálkozzon újra a DNS lekérdezéssel [8], illetve a nagyon lassan letöltődő weboldalak esetén állítsa meg a letöltést és ugorjon tovább. Ezek nélkül ezer link letöltése több órát vett igénybe, így csak néhány percet. A spider a következő algoritmus szerint működik:

1. A DOM-ból kiválasztja a <body> tag között lévő tartalmat.

2. Kitörli a dokumentumban található sortöréseket.

3. Kitörli a dokumentumból a <script> és <style> tag-ek között lévő tartalmat, ez számunkra nem hordoz információt.

4. Kitörli a dokumentumban található egyéb HTML tag-eket, a maradék szöveges tartalmat pedig összefűzi.

Minden dokumentumhoz letároltam egy egyedi azonosítót, az eredeti URL címet és a Google nyelvdetektáló algoritmusának futási eredményét.

5.2.2. Webelemzés

A dokumentumok klaszterezését végeztem el ebben a modulban. Alapjául szolgál a korábban ismertetett szövegbányászati modul, ezt fejlesztettem tovább. Az algoritmus futása a következő lépésekből áll:

1. Az előző lépésben letárolt összes dokumentumot szavakra bontom, szavaikat STOP-szótáramat felhasználva megszűröm, belőlük az írásjeleket kitisztítom.

2. Előállítom mindegyik szó előfordulási gyakoriságát. A leggyakrabban előforduló szó letárolásra kerül az egyes dokumentumhoz, megfigyeléseim alapján ez hasznos megfigyelés a témaköröket illetően.

3. Elkészítem a közös szótárat, letárolom külön elemzésekhez.

4. A dokumentumteret vektorizálom, és tf-idf alapján súlyozom.

5. A súlyozott vektorteret LSI model szerint 2, 4, 8 és 12 dimenziós vetkortérré redukálom.

6. A létrejött modell-ben topic-modelling elemzést futtatok, és a létrejövő témakörök legjellemzőbb szavait kimentem külön.

7. A dokumentumok mindegyikét besorolom a hozzá legközelebb eső témakör egyikébe.

5.2.3. Felhasználói felület

A felhasználó az 5.1 ábrán látható felületet használhatja a funkciók elérésére.

5.1. ábra. *Webtérképező modul kezelőfelülete*

A webelemzés megtekintése oldalon (5.2 ábra) elsőként a négy különböző dimenziójú elemzés által meghatározott kategóriák szerepelnek.

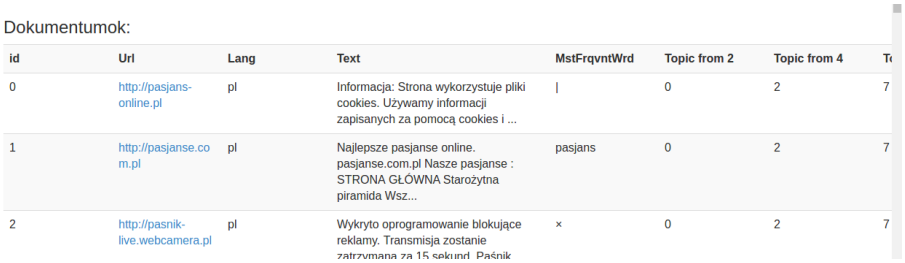
A dokumentumok, a megállapított nyelv, és a leggyakoribb szavak és a besorolt kategóriák pedig alattuk (5.3 ábra).

A feladat során e dokumentumokon végeztem el témakutatást.

5.3. Előzetes eredmények

Sajnos az adatbányászat egyik jellemző sajátossága, hogy bár az algoritmus amit terveztem, elméletben megfelelően működik, a kapott dokumentumtéren azonban

5.2. ábra. *Témakörök megállapítása*



5.3. ábra. *Dokumentumok és elemzésük eredményei*

sajnos az eredmények nem voltak kielégítőek. Több probléma is felmerült a futtatás során:

- **Túl nagy dokumentumtér.** A kapott linkgyűjtemény összesen 375.441 dokumentumot tartalmazott. Ezek mindegyikéből általában több oldal szöveg töltődött le. Mivel az alkalmazáson működése egy webszerveren valósul meg, tapasztalataim alapján már egy pár tízezres linklista is jelentős memóriahiányt okozott a 8 gigabite ram-mal felszerelt gépen.
  - Megoldásként egy 10.000 linkből álló címhalmazra redukáltam az elemzést.
- **Túl nagy kategóriahalmaz.** A kapott linkekhez összesen 52 különböző, legalább egy, maximum három címkét rendeltek. Ekkora halmazt szinte lehetetlen pontosan reprodukálni, pláne nem az ott meghatározott általános címkék alapján.
  - Megoldásként az elemzésemhez összesen négy modellt futtattam, 2, 4, 8 és 12 maximális kategóriát meghatározva. Az általam kapott kategóriákban szereplő dokumentumokon meg tudtam figyelni, hogy van-e szabályos minta a címkékben.

A fenti problémákra bár találtam kompromisszumos megoldást, azonban néhány olyan probléma is felmerült, amely miatt a feladat újratervezése mellett döntöttem.

- **Túl sok nyelv.** Az elemzés során a világ összes tájáról találtam nyelveket. A szövegbányászati modul nincsen felvértezve azzal a tudással, hogy hasonló témájú oldalakat nyelvtől függetlenül egy csoportba rendezze, hiszen az általam implementált módszert szinte teljes egészében a dokumentumokban található szavakra épít.

  – Általános nyelvi elemzőprogram írása lenne a megoldás, de ez túlmutat az egy ember által megírható projekt határain.

ɔn-	de	× Fitness Fitnessgeräte Hoverboard Kalorienbedarf berechnen Stoffwechsel anregen Sensewear Armban...	und	1	1	1	1
ŋgcu	en	Toggle navigation Passing Curiosity Home About Tags Archive Contact Passing Curiosity Welcome to ...	festival	0	2	4	2
assi n/	de	Die Pilze sind da! Und endlich schöne Pilzsammler-Porträts! Dazu der aktuelle Pilz-Ticker!Die Nac...	und	1	1	1	1
ɔass /	pt	Passei Direto Já possui cadastro? Login Facebook ou Esqueci a senha Entrar Conectando alunos e se...	ed	0	2	4	6
assi	fr	TENDANCE: Magnifique carré plongeant nuque courte Camille (3) Sublime carré long plongeantAccueil...	carré	0	2	4	10
ɔndri	de	Suchen: Der AutorImpressum HomeAlle Tests und FahrberichteNürburgring &	und	1	1	1	1

5.4. ábra. *Elemzés eredménye. Balról jobbra: weboldal nyelve, kivonat a tartalomból, leggyakoribb szó az oldalon, majd pedig elsőként a 2, 4, 8, végül a 12 kategóriás klaszterezés eredménye.*

Az 5.4 ábrán látható 1000 weboldal elemzéséből részlet. Erősen észrevehető, hogy a nyelvek mentén alakulnak ki a csoportok.

- Weboldal tartalma félrevezető. Rengeteg olyan esettel találkoztam, amikor a weboldalon elhelyezkedő reklámok mennyisége nagyobb volt, mint a weboldal hasznos tartalma. Gyakran fordult elő, hogy a generált tartalomban olyan kódrészleteket rejtettek el, melynek feladata a keresőmotorok félrevezetése. Sok esetben én magam sem tudtam a nyers HTML tartalom alapján kisilabizálni, miről is szólhat az adott oldal.

- Az oldal kódja nem utal a tartalomra. A legnagyobb probléma a feladattal az volt, hogy az általános webcrawler modul nem volt felkészítve a weboldalak sajátosságaira. A gyakran előforduló, személyre szabott reklámokon felül olyan oldalrészletek, mint pl. a cookie-k elfogadása, bejelentkezés, regisztráció, főoldal belépés és még számos más, általános weboldal elem, nem utal a weboldal tartalmára, mégis megjelennek a szövegben, ezáltal hamis kategóriákat alkotva. Ezen felül egy megjelenített weboldal felületének lehet, hogy nagy részét

egy szövegdoboz tesz ki, az a HTML tartalomban azonban csak az oldal tartalmának töredéke: elmondható, hogy egy weboldalt **nem lehet a vizuális megjelenítése nélkül hatékonyan bekategorizálni.**

12 témakörös bontás

id	Fő szavak
0	pdf, the, and, files, file,
1	und, die, der, von, für,
2	pdf, the, and, of, files,
3	в, и, на, с, для,
4	à, pour, une, des, du,
5	►, para, que, con, à,
6	►, para, que, con, por,
7	wiki, •, softonic, windows, wikia,
8	wiki, softonic, •, password, windows,
9	το, και, του, της, για,
10	een, |, op, w, apr,
11	pdf, een, pattaya, files, op,

5.5. ábra. *12 témakörös klaszterezés eredménye*

Az 5.5 ábrán látható 1000 weboldal elemzése során kialakult kategóriák legfőbb szavai. Látszik, hogy bár észrevehetőek témabeli kategóriák, többségében kezd eluralkodni a nyelvek mentén kialakuló kategorizálás.

Az elemzés kimenetét látva a következőt állapítottam meg:

- A program szinte kivétel nélkül nyelvek mentén határozza meg a klasztereket.
- A dokumentumok szövegezése között akkora óriási különbség van, hogy a vektorértékek közötti különbség elenyésző - azaz nincsenek jól elkülönülő kategóriák.

5.4. Újratervezés

A feladathoz eredeti célja bár nem megvalósítható, mégis érdemes megállapítani, mi az a pontosítás, ahol már használható, hasznos eredményeket kapunk. Ennek érdekében a következő módosításokat hajtottam végre:

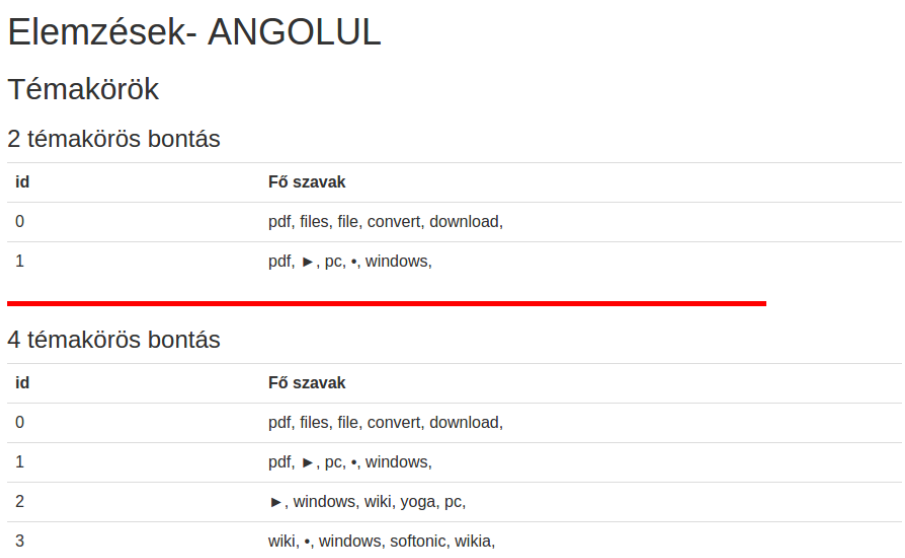
- Tovább pontosítottam a webcrawler modul működését. Kiegészítettem, hogy még több, gyakran előforduló tartalmat szűrjön ki, és még inkább a hasznos tartalmat nyerjem ki a weboldalakról.

- Leszűkítettem a működést angol nyelvre - tehát a dokumentumok közül csak azon weboldalak tartalmára futtattam le a programot, melyek nyelvét angolnak állapította meg a nyelvdetektáló modul.

5.5. Angol nyelvű eredmények

Az új elemzés futtatása után a következő eredményeket tapasztaltam:

- Az egynyelvű futtatás hatására kiküszöböltem a nyelvkezelési problémát.
- Kis dimenzió esetén általában két nagy klasztert állapít meg az algoritmus:
  1. Az egyik kategóriába tartozik általában egy specifikus, de gyakran előforduló weboldaltípus,
  2. a másik kategóriába pedig az összes többi oldal.
- Magasabb dimenziószámok esetében is csak néhány jól elkülöníthető weboldaltípus jön létre, nem jönnek létre újabb, jól elkülönülő csoportok.



Elemzések- ANGOLUL

Témakörök

2 témakörös bontás

id	Fő szavak
0	pdf, files, file, convert, download,
1	pdf, ►, pc, •, windows,

4 témakörös bontás

id	Fő szavak
0	pdf, files, file, convert, download,
1	pdf, ►, pc, •, windows,
2	►, windows, wiki, yoga, pc,
3	wiki, •, windows, softonic, wikia,

5.6. ábra. *Egy angol elemzés témakörei*

Az 5.6 ábrán látható, hogy a kétdimenziós bontás esetén, az első kategória valamilyen fájlletöltés/konvertálás témájú oldalakat fog össze.

A második kategória szavainak súlyozási értékeit vizsgálva az látszódik, hogy az összes szó egymáshoz képest nagyon közel van, értékük szinte azonos. Ebből arra következtettem, hogy ezek a szavak a "minden más" kategóriát képviselik.

Tovább emelve a dimenziószámot láthatjuk, hogy a korábbi első kategóriánk megmarad, az új kategóriákban viszont továbbra is egymástól nem túl jól elkülönülő csoportok jönnek létre. Ez látszik a dokumentumtéren is(5.7 ábra).

			All ROM...					
410	http://www.pd4pic.com/	en	Contacts Signup Login Cliparts goat, farm, animal, nature, white gopher tortoise, gopherus polyph...	animal	1	1	4	1
412	http://pdbooks.ca	en	PDBooks.ca Great Books in Canadian Public Domain Site Navigation[Skip] Home Languages English Fre...	en	1	1	0	1
414	http://pdcomedy.com	en	Home Menu Sections Cartoons TV - Part 1 TV - Part 2 Clips Movies Music Radio Cinema Serials Silen...	tv	1	1	6	1
416	https://www.pdf-archive.com/	en	PDF Archive Share your PDF documents on the Web, Facebook and Twitter - Protect and restore your ...	pdf	0	0	0	0

5.7. ábra. *Angol elemzés dokumentum besorolásai. Balról jobbra a következő látható: weboldal sorszáma, URL-je, megállapított nyelve, szöveg kivonat, leggyakoribb szó, majd pedig a 2-4-8-12 dimenziós klaszterezés eredményei.*

Az ábrán látszik, hogy a 416-os, online pdf tárhelyet a "0"-s kategóriába sorolja a program, míg az összes többi oldalt a másik kategóriába. Magasabb dimenziószám esetén érdemben ez a felosztás nem változik.

Megállapítottam, hogy az elemzést a korábban említett nehézségek, a webes tartalom zajossága, és a html tartalom sajátosságai miatt ilyen mértéknél jobban az általam felhasznált eszközökkel pontosabban nem lehet megoldani.

Az elemzéseket akkor lehetne pontosabb eredménnyel futtatni, ha sokkal szélesebb algoritmikus és hardveres eszköztár állna a rendelkezésre, illetve ha komplex feldolgozását végeznénk el egy weboldalnak a DOM szövegkinyerése mellett.

6. fejezet

Ismert dokumentumtér: Programozott webelemzés

A programozott webelemzés során a következő két weboldalra írtam egy-egy, a weboldal sajátosságainak megfelelően erősen specializált webcrawler-t:

- index.hu
- wikipedia.hu

Az által, hogy a weboldalak sajátosságaira előre fel tudtam készülni, jóval pontosabb kutatásra volt lehetőségem. A feladat során arra a kérdésre kerestem a választ: a webportálon egy adott napon mely témakör a legfelkapottabb, és mely cikkek tartoznak bele?

Az index.hu egy internetes hírportál, így az adott nap legnépszerűbb témakörét és a hozzá tartozó újságcikkeket kerestem.

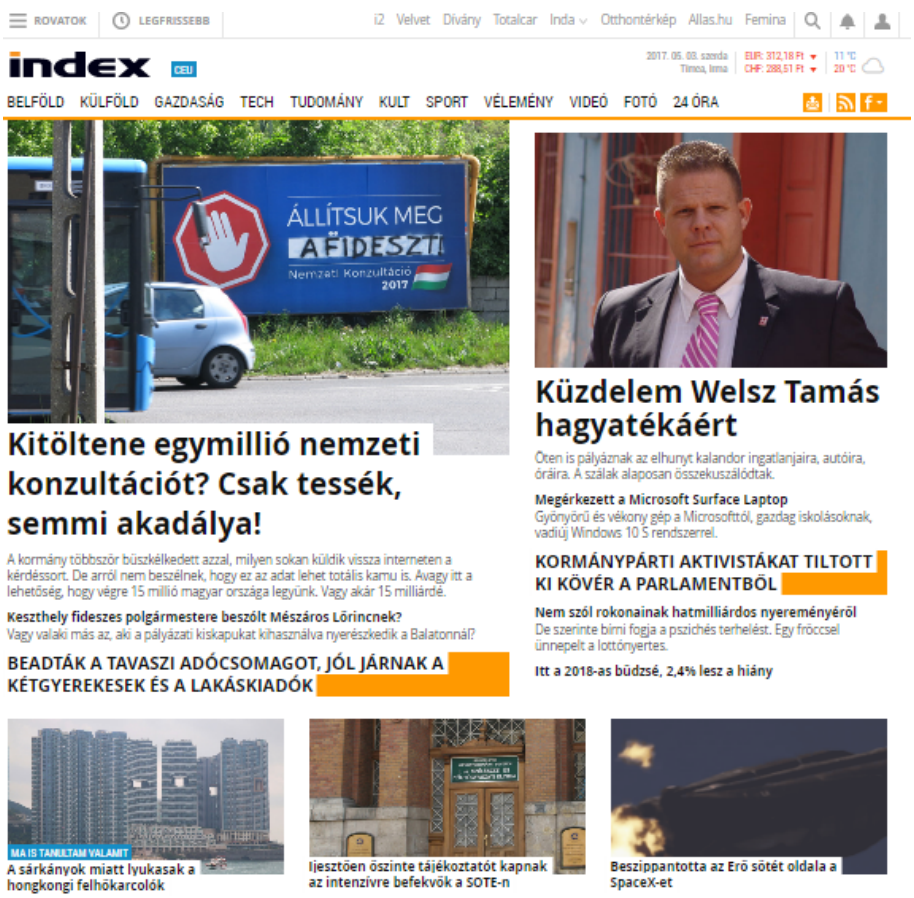
A wikipedia.hu internetes tudásbázis főoldalán minden nap vegyesen ajánlanak a weboldalon tárolt szócikkek közül, kíváncsi voltam felfedezhető-e bennük valamilyen közös téma.

6.1. Paraméterek

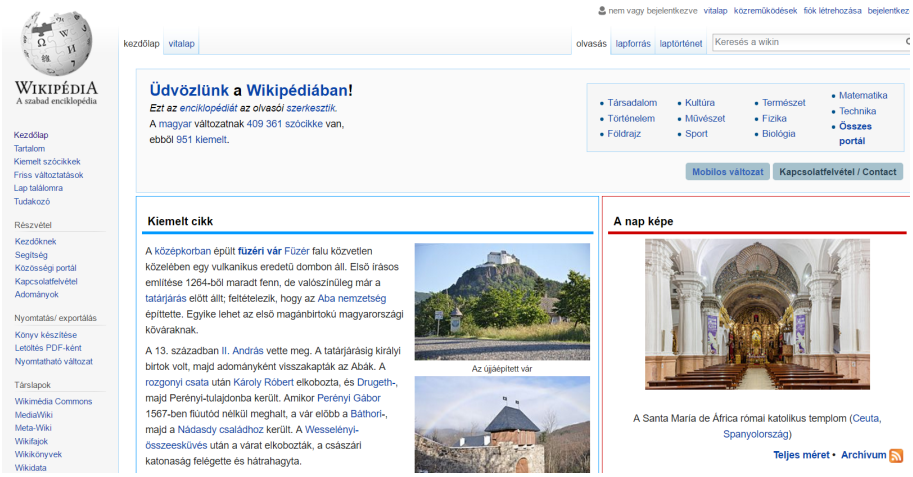
Mindkét weboldal feltérképezése során rekurzív módon nyerem ki a főoldalon található linkeket, melyek közül a weboldalhoz köthető hivatkozásokról kinyerem a szöveges tartalmat. Így az index.hu internetes hírportálon az éppen aktuális cikkeket, a wikipedia.hu-n pedig a főoldalról elérhető szócikkeket térképeztem fel.

6.2. Funkciók tervezése, megvalósítás

A feladat megoldásához a következő lépéseket terveztem meg:



6.1. ábra. *Az index.hu kezdőoldala*



6.2. ábra. *A wikipedia magyar kezdőoldala*

1. **Fázis: Dokumentumtér kialakítása.** Ebben a lépésben a weboldal aktuális tartalmát letöltöm, rajta szereplő linkeken keresztül felolvasom a weboldal hasznos tartalmát.

2. **Fázis: Témaelemzés:** elvégzem a már ismertetett szöveg- és adatbányászati lépéseket, megállapítom témaelemzés segítségével a legjellemzőbb témákat a

dokumentumtéren.

6.2.1. Dokumentumtér kialakítása

A dokumentumtér kialakításához egy általam írt Scrapy Spider-t használtam fel.

A Spider konfigurálása során törekedtem arra, hogy minden elérhető cikket letöltsön a weboldalról, még ha az első alkalommal bármilyen okból nem is sikerül, próbálkozzon újra néhányszor. Ezen felül egyedül az http://index.hu és https://hu.wikipedia.org címet adtam meg a rekurzió kezdetekor, minden más információt a weboldalról nyerek ki.

A Spider a következő lépéseket végzi el:

1. Letölti a kapott link tartalmát.
2. Kiszűri belőle azokat a <a href="..">> tag-eket, melyek az eredeti weboldalhoz tartoznak. (tehát pl. index.hu esetén a főoldalon linkelt blog cikkeket nem olvasom be.)
3. Meghívja a dokumentum feldolgozó eljárást mindegyik érvényes cikkre.

A dokumentum feldolgozó eljárás során történik a weboldal-specifikus tartalom kinyerése. A megoldás megtalálása során a Scrapy CLI (Command Line Interface) felületét használva lépésről-lépésre nyertem ki a tartalmat, és a végső kódsort implementáltam a programba (6.1 kódrészlet).

6.1. lista. *Példa részlet a CLI működéséből float*

```
 >>> response.css('meta').re(r'.*author.*')[0]
Out[9]: u'<meta name="author" content="Stubnya Bence">'
```

A specializált webcrawler kódnak köszönhetően minden dokumentumban csak a konkrét szövegtörzs szerepel (6.3 ábra).

167	http://index.hu/gazdasag/allas/2017/04/18/londonban_kell_a_legtobbet_dolgozni/	Londonban kell a legtöbbet dolgozni	A londoniak nagyjából száz órával többet dolgoznak egy évben, mint az ország többi része, írja a . A helyi mindenkit, tehát a részmunkaidősöket is belevéve 33 órás az átlagos munkahét arrafelé, ami a 2008-as vál legmagasabb. Nem mellesleg pedig az ország többi részén 331 óra az átlag.Hivatalos munkában töltött idő kevesebbet dolgoznak a britek, mint a magyarok, legalábbis az alapján. Igaz ez alapján a görögök dolgozn hát ők töltenek a legtöbb időt hivatalosan munkával az EU-ban. Az OECD-országok közt is talán csak Cost tudná őket megszorongatni.Visszatérve a britekhez, például az lehet a londoniak több munkája mögött, hog dolgoznak ott, mondta el Roanld McQuaid, a strilingi egyetem munkaerőpiaccal foglalkozó közgazdásza. A költségei is magasabbak, mait egyszerűen ki kell termelnie a dolgozóknak. Arányaiban egyébként több a te foglalkoztatott is a fővárosban.Emellett többet is fizetnek Londonban, átlagosan óránként 10 fonttal többet, részein, jobban meg is éri benn maradni. Emellett Londonban koncentrálódnak az olyan, hagyományosan munkaidővel járó iparágak, mint például a pénzügyi szolgáltatásokhoz kapcsolódóak.
168	http://index.hu/kulfold/2017/04/18/utasokkal_teli_busz_zuhant_a_szakadekba_a_fulop-szigeteken/	Utasokkal teli busz zuhant a szakadékba a Fülöp-szigeteken	Legalább 24 ember meghalt, amikor egy busz 30 méteres szakadékba zuhant a Fülöp-szigeteken – a szige Sunstar.com.ph.Az áldozatok száma még emelkedhet, a buszon ugyanis negyvenen utaztak a fővárostól é Ecija tartományban. A busz a szemtanúk szerint túl gyorsan hajtott, azt azonban egyelőre nem tudni, hogy műszaki hiba vezetett a keddi tragédiához.

6.3. ábra. *Az előkészített dokumentumok*

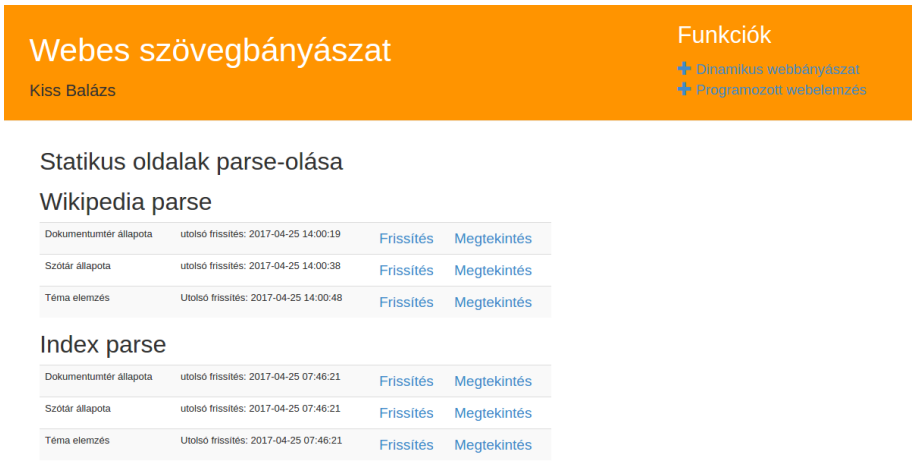
6.2.2. Témaelemzés

A témaelemzés során a már ismertetett Topic modelling eljárásokat alkalmaztam. A legfőbb különbségek a Dinamikus webelemzéshez képest a következők:

- Szótár létrehozáskor csak a magyar nyelvű STOP-szótáramat használtam fel, hiszen az oldalak nyelve is kizárólag Magyar.
- A legfőbb témakör meghatározásához az optimális módszer a kétdimenziós LSI transzformálás futtatása: így egyértelműen megállapítható a leginkább kiugró téma.
- Az elemzés során a cikkeket tovább súlyoztam aszerint, hogy a bennük szereplő szavak közül melyek fordulnak elő az algoritmus által meghatározott két fő téma szavai közül. Ezáltal még jobban megszűrtem a cikkek közül a valóban legfőbb témába vágó oldalakat.
- Az elemzés eredményéül a leginkább a témára jellemző, "nap cikke" dokumentumot külön lementettem.

6.2.3. Felhasználói felület

A felület a már megismert sémára épült, összesen három menüpont segítségével érhető el az egy-egy weboldal elemzése.



6.4. ábra. *A statikus webparse-olás kezdőoldala*

A dokumentumtérben a lementett cikkek szerepelnek a már látott módon Sorszám, URL, Cím, Szöveg sorrendben.

A szótár menüpontban a szavak gyakoriság szerinti csökkenő sorrendben tekinthetőek meg.

Az index.hu-n átlagosan 200-240 cikk szerepel. A wikipedia.hu-n átlagosan 250 szócikk szerepel a főoldalon.

6.5. ábra. *Szótár*

6.3. Eredmények

6.3.1. Hírportál elemzés

Az index.hu hírportálon az adott napok függvényében sikerült pontosan meghatározni a feladatban kijelölt, nap legfontosabb témakörét, és eme témakört legjobban jellemző cikket.

Például 2017.április 13-án a két fő témakör a 6.6. ábrán látható.

6.6. ábra. *Eredmények: fő témák*

1. Gulyás Gergely és Varga Gergely eljárása rongálásért. Ezt indikálják a kiemelt szavak: "Gulyás", "két", "kormány".

2. Az összes többi cikk. Itt minden szó szinte egyforma, alacsony fontossággal rendelkezik, azok a szavak dominánsak a kategóriában, melyek kizárólag csak egy-egy cikkben szerepelnek: pl. "műtét", "színház", "györgy"

A két témakörre legjobban vonatkozó cikkek is ezt mutatják:

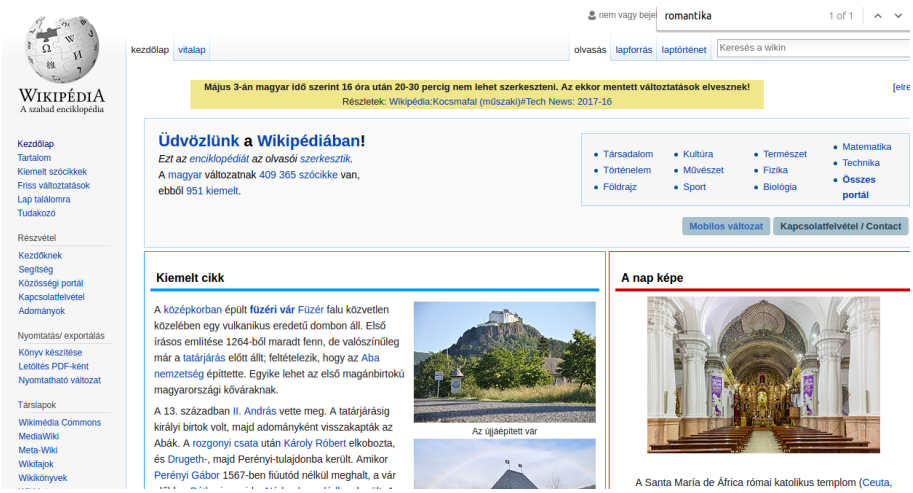
1. Az első kategóriát legjobban jellemző cikk a bírósági tárgyalásról szóló fő cikk. A többi, még kategóriába tartozó cikk a kapcsolódó tüntetésekről, politikai elemzésekről szólnak.

2. A második kategóriát legjobban jellemző cikk, mivel ez az összes cikk közös gyűjtése, az jellemzi a legjobban, amely az előző kategória témakörétől a legtávolabb áll. A legerősebben ez a Cserhalmi György színművész műtétjéről szóló cikk, de csak nagyon kis értékkel maradnak le a különböző magazinok egyéb specifikus cikkei, autókról, bulvárról, technológiai eseményekről tudósítva.

Az eredményeket értelmezve megállapítható, hogy az algoritmus alkalmas arra, hogy megállapítsa, mely a legfontosabb hír egy olyan napon, amikor egy "forró" téma dominál.

6.3.2. Wikipedia elemzés

A wikipedia elemzése ugyanazon logika mentén került implementálásra, mint a hírportál elemzése. Itt az eredmények a következőképpen alakultak:



6.7. ábra. *A wikipedia főoldala az elemzés pillanatában*

A wikipedia főoldala általában egy kiemelt cikkel és a nap képével kezd, majd pedig a mai napon és a mostanában történt aktuális évfordulókkal folytatja a cikkek felsorolását.

Az eredmény a témaelemzés során a következő:

Megfigyelhetjük a 6.8 ábrán, hogy a Romantika, mint szócikk közvetlen hivatkozásként elérhető a nap kiemelt cikkéből. A nap képe és a kiemelt szócikk egyéb linkjei is különböző történelmi korokból szemezget, ahogy az évfordulókról szóló rész is. Ez utóbbinál az évfordulók jelentős többsége az elmúlt 200 év eseményei közül válogat.

6.8. ábra. *Témaelemzés eredménye a wikipedia oldalon*

Így világossá válik számunkra a győztes kategóriák eredményei mögött a logika:

1. Első kategória a 19-20 és 21. század eseményei

2. A másik kategória az összes többi cikk.

Ezek alapján az eredményről azt állapítottam meg, hogy bár az algoritmus jól működik, és a dokumentumok ismeretében helyes eredményt ad, a wikipedia.hu-n nem figyelhető meg kifejezett "nap témaköre" besorolás.

Több különböző napon is lefuttatva az elemzést nagy többségben általában a már fent említett, elmúlt három évszázad jött ki győztes kategóriaként: így megállapítható, hogy a weboldal sajátossága, hogy az elmúlt három évszázadból található rajta a legtöbb információ a többihez képest, így ezek a cikkek szerepelnek túlsúlyban az oldalon, ezért kapjuk ezt az eredményt.

6.3.3. Konklúzió

Az előzetes kutatásaim során megállítottam, hogy magyar nyelvű hírportálok cikkeit sikeresen lehet szöveg-és adatbányászati eljárások segítségével betanítás után automatikusan címkézni.

Dinamikus szövegbányászati kutatásaim során megállapítottam, hogy egy átfogó, mindent jól bekategorizáló alkalmazás írása túlságosan nagy feladat. A probléma felbontása után sikeres részeredményeket értem el, sikerült angol nyelvű weboldalak esetén hasonló weboldalakt közös kategóriákba helyeznem.

Az előre felkészített szövegbányászati modul sikeresen állapítja meg az általa elemzett weboldalak legfőbb témaköreit. A specifikus tartalomra felkészített elemzőszoftver precízebb eredményekkel szolgál, mint a dinamikus elemzés során használt társa.

Feladat	Módszerek	Eredmény
1. Előzetes szövegelemzés	Szövegtörzsek hasonlósági témaelemzése	Sikeres magyar nyelven
2. Dokumentum osztályozás	Internetes hírportál cikkek kategorizálása, osztályozás segítségével	Sikeres magyar nyelven
3. Webtérképezés	Ismeretlen weboldalak nyelvi feltérképezése klaszterezéssel	Részben sikeres angol nyelven
4. Programozott webelemzés	Témaelemzés és főbb cikkek meghatározása klaszterezéssel	Sikeres magyar nyelven

7. fejezet

Összefoglalás, értékelés

A Diplomamunkám során egy több iterációból és al-feladatokból álló, komplex webes szöveg- és adatbányászati szoftvert készítettem el.

A működéshez webes segédkönyvtárakat, általam írt algoritmusokat, webcrawlereket és egy komplett webes keretrendszert használtam fel.

A feladat során több elemzést is elvégeztem, eredményeiket, megfigyeléseimet értelmeztem, beépítettem a szoftverbe. Több lépés során jutottam el a végső algoritmusokig, hol a sikeres elemzéseket továbbfejlesztve, hol pedig a hibás elgondolásokat újratervezve. Elkészítettem egy működő webalkalmazást, amely keretrendszerként többféle web, adat- és szövegbányászati feladatra is felhasználható.

Mindezen eredményeket egy komplex webalkalmazásba ültettem át, mely segítségével az elemzések könnyedén reprodukálhatóak. A szöveg- és adatbányászati elemzések több lépése és a kód futtatása teljesen automatikusan történik.

A tervezés lépései során kialakított algoritmusok kimenetei hasznos és helyes információval szolgálnak. Kutatásaim és munkám során a szövegbányászat eszközeiről igazoltam, hogy felhasználhatóak a magyar nyelvű weboldalak esetében is.

A feladat során összesen 3000 sor kódot írtam Python, HTML és egyéb scriptnyelveken. Elmélyítettem ismereteimet a Django keretrendszer működésében, saját modult és felületeket terveztem. Több segédkönyvtár és eszköz működését megismertem, elsajátítottam, azokat a feladat megoldásához felhasználtam.

Az időm jelentős hányadát tette ki az adatbányászati modul, és a szövegbányászati lépések megtervezése, tesztelése. Az elkészült programkódokat alapos tesztelés után beillesztettem a webalkalmazásba.

7.1. Továbbfejlesztési lehetőségek

A webalkalmazás továbbfejlesztéséhez az egyik lehetséges út a felhasznált adatbányászati eljárások finomítása, főleg a dinamikus modul esetében. Más adatbányá-

szati elemzések kipróbálása, neurális hálók alkalmazása, weboldalak közötti egyéb összefüggések (pl. egymásra mutató linkek, címkék) kutatása.

Jelenleg nem létezik ingyenesen elérhető és hatékony magyar szótövezési eljárás - ennek kidolgozása nagyban növelné az algoritmus hatékonyságát.

Az elemzések testreszabhatóságát növelné, ha a különböző webes elemzések precíziós paramétereit a felhasználó beállíthatná a webalkalmazás felületén.

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

Másik ötlet, hogy a fájlok memóriában, majd pedig a diszken történő szöveges fájlokként való tárolása helyett inkább egy adatbázist építsek fel a rendszer mögé. Az adattárolási réteg a skálázhatóságot és a szoftver feldolgozó erejét növelhetné, illetve a párhuzamos webcrawl esetén a hardveres szűk keresztmetszetet csökkentené esetleg egy elosztott rendszerrel történő web bejárás.

Nagyszabású projekt a statikus elemzések során, ha több hasonló oldalra is felkészítjük az alkalmazást, pl. több más magyar hírportált is együttesen elemeznénk, és ebből állapítanánk meg egy nap legfontosabb híreit - akár angol vagy más nyelveken is.

A szövegbányászati és topic modelling eszközök közül még jó pár létezik, melyeket nem használtam fel, nem próbáltam ki: ilyen az LSI helyett Random Pojections, RP [22], osztályozásnál a k-nn módszer, vagy a klaszterezésnél a k-medoid klaszterezés [12], [5].

7.2. Köszönetnyilvánítás

Köszönöm konzulensemnek, Nagy Gábornak az általános segítséget és a téma mesterképzésen átívelő vezetését. Kazi Sándornak a szövegbányászati, python és LaTeX segítségnyújtást. Köszönöm a lektorálást Ökrös Tamásnak, Módly Márknak, Mérész Dánielnek és Pálmai Nórának.

Irodalomjegyzék

[1] Arisiri. Clustering vs classifictaion. http://arisri.tistory.com/entry/Clustering-VS-Classification (, 2015.

[2] Python Community. python. https://pypi.python.org/pypi/langdetect, 2016.

[3] Scrapy community. Scrapy. https://scrapy.org, 2017.

[4] Douglas Crockford. Json. http://www.json.org/xml.html, 2001.

[5] Tikk Domonkos. *Szövegbányászat ISBN 978-963-9664-45-6*. Typotex, 2007.

[6] Django Software Foundation. Django. https://www.djangoproject.com, 2015.

[7] Django Software Foundation. View decorators. https://docs.djangoproject.com/en/1.11/topics/http/decorators/, 2015.

[8] Network Working Group. Domain names - concepts and facilities. https://tools.ietf.org/html/rfc1034, 1987.

[9] Network Working Group. The application/rss+xml media type. http://tools.ietf.org/id/draft-nottingham-rss-media-type-00.txt, 2007.

[10] Jeroen Janssens. 7 command-line tools for data science. http://jeroenjanssens.com/2013/09/19/seven-command-line-tools-for-data-science.html, 2013.

[11] Fred W. Prior David A. Rubin Joseph P. Erinjeri, Daniel Picus and Paul Koppel. Development of a google-based search engine for data mining radiology reports. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3043709/, 2009.

[12] Dr. Abonyi János. *Adatbányászat, a hatékonyság eszköze ISBN:963-618-432-2*. ComputerBooks, 2004.

[13] Nagy Gábor Kiss Balázs. Dokumentum osztályozás adatbányászati módszerekkel, 2014.

[14] Nagy Gábor Kiss Balázs. Dokumentum osztályozás adatbányászati módszerekkel, 2015.

[15] Jeff Knupp. What is a web framework? https://www.jeffknupp.com/blog/2014/03/03/what-is-a-web-framework, 2015.

[16] Tom Griffiths Mark Steyvers. *Probabilistic Topic Models*. 2009.

[17] MyTARDIS. Django architecture. https://pythonhosted.org/MyTARDIS/architecture.html, 2015.

[18] Oracle. Classification. http://www.oracle.com/ocom/groups/public/@otn/documents/digital 2015.

[19] Leonard R. Beautifulsoup. http://www.crummy.com/software/BeautifulSoup, 2015.

[20] W3C Recommendation. Selectors level 3. https://www.w3.org/TR/selectors/, 2011.

[21] Trygve M. H. Reenskaug. Mvc. http://heim.ifi.uio.no/ trygver/themes/mvc/mvc-index.html, 1978.

[22] Radim Rehurek and Petr Sojka. Gensim. https://radimrehurek.com/gensim, 2009.

[23] Richard N. Taylor Roy T. Fielding. Principled design of the modern web architecture. http://www.ics.uci.edu/ taylor/documents/2002-REST-TOIT.pdf, 1999.

[24] Nakatani Shuyo. Cybozu labs, inc. https://www.slideshare.net/shuyo/language-detection-library-for-java, 2010.

[25] C. V. Jawahar Siddhartha Chandra, Shailesh Kumar. Learning hierarchical bag of words using naive bayes clustering. pages 382–395.

[26] The Internet Society. Hypertext transfer protocol. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3043709/, 1999.

[27] Radim Rehurekand Petr Sojka. Software framework for topic modelling with large corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, page 45.

[28] Gerard Swinnen. Tanuljunk meg programozni python nyelven. http://mek.oszk.hu/08400/08435/08435.pdf, 2005.

[29] Tf-idf. A single-page tutorial. http://www.tfidf.com, 2015.

[30] Tutorial.hu. Hunagrian stop words. http://www.tutorial.hu/hungarian-stop-words, 2015.

[31] W3C. Web services glossary. https://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice, 2004.

[32] W3C. Document object model (dom). https://www.w3.org/DOM/, 2009.

[33] How Stuff Works. How internet search engines work. http://computer.howstuffworks.com/internet/basics/search-engine1.html, 2015.