

Önálló laboratórium beszámoló

Távközlési és Médiainformatikai Tanszék

|  |  |
| --- | --- |
| Készítette: | **Kiss Balázs** |
| Neptun-kód: | **DCZUKT** |
| Ágazat: | **Alkalmazott informatika** |
| E-mail cím: | **kiss.balazs.forest@gmail.com** |
| Konzulens(ek): | **Nagy Gábor** |
| E-mail címe(ik): | **nagyg.tmit.bme.hu** |

**Téma címe: Webes dokumentum osztályozás szövegbányászati módszerekkel**

Feladat

Az Önálló laboratórium 1. tárgy keretein belül megoldott feladat eredményeinek felhasználásával olyan Python alapú webalkalmazás elkészítése, amely adat- és szövegbányászati módszerek segítségével képes szövegek tömeges feldolgozására. A feldolgozás során webes cikkek szövegtörzsének összehasonlítását végezzük. Az összehasonlítás segítségével a szöveg címkézésére készítünk automatikus javaslat funkciót a korábban letárolt cikkek címkéi alapján.

**2014/2015. 2. félév**

# Bevezetés

Az önálló laboratórium 1. tárgyam keretein belül az alábbi kérdésre kerestem a választ: lehetséges-e adat- és szövegbányászati módszerek segítségével olyan algoritmust, programot írni, mely képes több, a magyar újságírásban megjelenő cikk elemzésére. Az elemzés tárgya elsősorban a cikk tartalmát röviden összefoglaló címkék. Ezen néhány szó, esetleg rövid kifejezés tipikusan jellemzője az adott szövegnek.

Az előző dolgozatom[2] eredményeként kijelenthető: a K-monitor[3] által rendelkezésemre bocsátott, összesen 15.000 webes cikk elemzése kapcsán azt a bíztató eredményt kaptam, hogy a címkék jellegükből adódóan előre megjósolhatóak egy-egy cikk kapcsán. Ennek egyik módja az adott szövegtörzsből való kikövetkeztetés: e helyett azonban az igazi kihívást az adatbázisban található többi cikkhez való hozzávetés jelenti. Kutatásom eredményeként akkor kiderült, hogy adatbányászati módszerek segítségével jól meghatározható azon cikkek halmaza, melyek egymáshoz hasonlítanak tartalmuk szerint annyira, hogy a rájuk megírt címkék közösek lehetnek.

Ennek a megállapításnak a konkrét, webes alapú implementációját készítettem el a feladatom során.

A munka a felhasználandó adat- és szövegbányászati eszközök megismeréséből, a python nyelv és Django keretrendszer elsajátításából, illetve implementációs logika összeállításából állt.

# Adat- és szövegbányászat

## Bevezetés

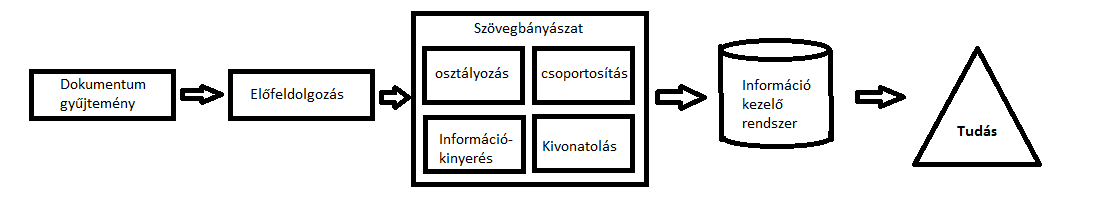
Az adatbányászat[4] a modern adatfeldolgozás egyik legfontosabb területe. A témáról bővebben a korábbi önálló laboratóriumi beszámolómban[2] és a szakdolgozatomban[1] is foglalkoztam, jelenleg a feladathoz szükséges elméleti alapfogalmakat mutatom be.

Az adatbányászat lényege, hogy valamilyen strukturált adathalmazon matematikai eszközökkel keresünk összefüggéseket. A számítástechnika feldolgozási sebessége révén eddig ismeretlen elemzési módszerek is megvalósíthatóak.

A szövegbányászat[5] szorosan kapcsolódik az adatbányászathoz. Feladata, hogy a strukturálatlan szövegekből, az adatbányászathoz szükséges strukturált adathalmazt hozza létre. Az adott szavakat a dokumentumok összességéből kigyűjtjük, és egy szótárban helyezzük el. A szótár számértéket határoz meg az egyes szavakhoz. Gyakran alkalmazunk szűrést, ahol a számunkra felesleges szavakat nem tároljuk el (Stopszó-szűrés) illetve a ritkán előforduló kifejezéseket is elvethetjük.

Végeredményként az adatbányászati elemzésekhez (általában) olyan vektortér jön létre, melyben minden sor egy-egy dokumentumnak felel meg: a vektorok értékei pedig az egyes, szótárban szereplő szavak, és előfordulási gyakoriságuk. Ebből aztán a későbbi elemzések fényében különféle transzformációkat alkalmazhatunk, amiket pedig matematikai módszereken alapuló modellekké formázhatunk. Az adott modellek segítségével az új, vagy már meglévő dokumentum-téren végezhetünk el különféle elemzéseket, értékeléseket.

## A szövegbányászat lépései



1. ábra. A szövegbányászat lépései

A szövegbányászat lépései az 1-es ábrán láthatóak szerint a következő képen alakulnak: elsőként össze kell gyűjtenünk az elemezni kívánt dokumentumokat. Előfeldolgozás során egységes formátumba hozzuk őket, kiszűrjük belőlük a nem kívánt szavakat, előformázzuk az adatbányászati lépésekhez. A szövegbányászat során már megoldott statisztikai, gépi tanulási, illetve adatbányászati problémákra próbáljuk meg visszavezetni a feladatot, különböző eszközök segítségével. A kész eredmények kézhezvételével még korántsem végeztünk: külön kihívás az eredmények értelmezése, validálása, és feldolgozása.

## Néhány fontos kifejezés:

* + **Topic modelling**: a szövegbányászat egyik kiemelten használt eszköze, a jelenlegi feladatom megoldásához is ezt használtam. A topic modelling egy olyan statisztikai modellezési eljárás[7], amely során kiemeljük a dokumentumok tartalmának központi elemeit, témáit. Például ha egy szövegben gyakran előfordulnak politikusok nevei és a parlament szó, akkor valószínűleg az országgyűlés a témája. Az ilyen témabeli hasonlóságokat és különbségeket a dokumentumok között értelmezzük, két- vagy többdimenziós térben elhelyezett pontok távolságaként definiáljuk.
  + **Stop-szó**: azon szavak gyűjteménye, amelyeket nem szeretnénk a szótárunkban letárolni: ilyenek például a kötőszavak, ragok, köszönés, stb.
  + **Bag of words[6]**: szavak és azok előfordulási gyakoriságainak tárolása. például ha két dokumentumom van („Andris szeret enni”, „Andris gyakran alszik”) akkor a belőlük elkészített szótár: (Andris, szeret, enni, gyakran, alszik) és a két dokumentum vektora pedig (1,1,1,0,0) és (1,0,0,1,1).
  + **Corpus**: a már strukturált, nagyméretű dokumentumhalmazokra használt kifejezés. A kész corpuson már definiálhatunk adatbányászati elemzéseket.
  + **Term Frequency-inverse Document Frequency (tf-idf**): egy olyan statisztikai érték[8] amely megmutatja, mennyire fontos (jellemző) egy szó egy dokumentumban az adott corpusra nézve. Segítségével megállapíthatjuk a központi kifejezéseket, valamint eltávolíthatjuk a kevésbé fontos (nem jellemző) szavakat. Működése arra alapszik, hogy minden szónál emeli a fontosságát (súlyát) az adott szónak, ha az csak az adott dokumentumban fordul elő; ennek megfelelően csökkenti, ha a témától távoli dokumentumokban is szerepel. A mai napig is az egyik leggyakrabban használt, és jól bevált eszköz például a keresőmotorok világában. A feladatom során ennek a modellnek a segítségével hasonlítottam össze a különböző cikkeket.
  + **Latent semantic indexing (LSI)**: egy olyan matematikai modell[9], amely leredukálja a dokumentumunkat olyan értékhalmazra, amely segítségével könnyebben és gyorsabban tudjunk azt elemezni. Például ha adott két (rövid) dokumentum 3-3 szóval, akkor őket 3 dimenziós térben tudjuk a legjobban elemezni. Hosszú dokumentumoknál és nagy szótárak esetében ez nem hatékony. A tf-idf elemzés csak a legfontosabb szavakat használja fel, segítségükkel kisebb dimenzióba redukálja a vektorteret. Ezen már könnyedén tudunk távolságot számítani, és hasznosabb, pontosabb eredményeket is kapunk.

# Webalkalmazás

A feladat során egy olyan webalkalmazás megvalósítását tűztük ki célul, mely könnyen kiépíthető, egyszerűen használható, mégis sokrétű. Természetesen feltétel volt, hogy könnyedén beépíthető legyen a webalkalmazás logikájába az adatbányászati modul. Tekintve, hogy a Gensim Python 2.7.3-as verzióra íródott, a legjobb megoldásnak ez a választás bizonyult.

## Python

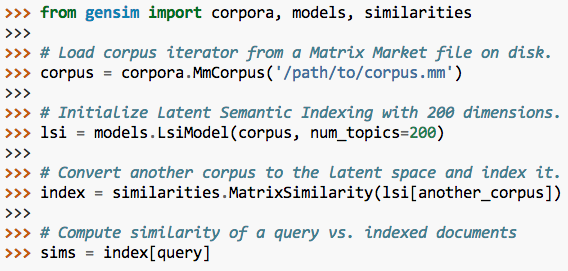
A Python egy általános célú, magas szintű programozási nyelv[10]. 1989 óta fejlesztik, jelenleg a 3.3.3-as verzió van elterjedőben. (a 2 és 3-as verziók közötti váltás jelentős volt, a teljes körű kompatibilitást is feláldozták az újítások kedvéért.)

A nyelv főbb jellemzői:

* portabilitás: Linux, Windows, Mac OS, stb. Ingyenesen
* objektum orientáltság
* scriptek írásától a komplex alkalmazásokig, széles skálán mozog felhasználása
* szintaxisa egyszerű, fejlett adattípusokat használ, tömör és jól olvasható a kód, felesleges karakterek nélkül. Például a bekezdések, sorközök, új sor mind a struktúrát is jelölik
* automatikus és effektív erőforrás kezelés. Nincsenek pointerek sem
* igény szerint többszálúsítható
* beépített hibakezelő rendszer
* interpretált és compile működés is lehetséges (előbbit egy virtuális gép hajtja végre, utóbbi esetén pedig Java bytekód generálódik)
* széles fejlesztői réteg, közösség, rengeteg kiegészítés.

## Gensim

A gensim[11] egy topic modellezést támogató Python programcsomag. Open source projektként fejlesztették. A fő motiváció egy olyan programcsomag elkészítése volt, mely segítségével dinamikusan felhasználhatjuk a topic modelling adta lehetőségeket saját programokban is.

Napjainkban a gensim az egyik legrobosztusabb eszköz a szemantikus modellezéshez, egyszerű szövegekből. Korábbi feladataim során gyakran kellett a szövegbányászati lépéseket saját algoritmusok segítségével megvalósítanom, majd az eredményen különféle adatbányászati szoftvereket futtatnom. Ezt a lépést spórolja meg nekünk a program, illetve természetesen az automatikus működést teszi lehetővé beépített környezetben.

2. ábra. Egy gensim példakód

### Funkciók

* Corpus és Vektortér kezelés: segítségével több különböző szövegből álló listát tudunk dinamikusan tisztítani, előkészíteni és transzformálni. Első sorban string-listákat használ, hiszen ezeket könnyű végigiterálni például a stop-szavak kiszűréséhez. Automatikusan elvégzi a szavak tokenizációját (összefüggő stringből szóközök mentén szavakra bontását), létrehozza belőlük a szótárt. Egyszerűen előállítja az előfordulás-gyakoriság vektort. Egy ilyen vektor tárolja, hogy egy adott szó milyen gyakran fordul elő a szövegben. A teljes dokumentumtér összes szavával ilyen vektor sorozatok alkotnak egy-egy sort, azaz egy-egy dokumentum reprezentációját.
* Coprus, mint mátrix: hatékony számításokhoz elengedhetetlen, hogy mátrix műveletek felhasználásával végezzük el a későbbi műveleteket. Előnye, hogy a vektorterünkbe átalakított dokumentumokat a gensim mátrixként tárolja.
* NumPy és SciPy támogatás: az egyes matematikai funkciók, műveletek ezekre az ismert programkönyvtárakra épülnek; segítségükkel saját magunk is definiálhatunk műveleteket a corpuson.
* Transzformációk: az elkészült vektorterek módosítására is lehetőség van. A kész vektorterek szótárait könnyedén bővíthetjük, új dokumentumokat vehetünk fel. Lehetőség van vektortér típusok közötti konverzióra is: pl. Bag of words-ből több egyéb mellett tf-idf-be alakításra is.
* Hasonlósági elemzés: a vektorok között definiált távolságok meghatározása is lehetséges. Modellt hozhatunk létre a betanított adatokból, majd ezt felhasználva különböző – például cosinus alapú - távolságértékeket értelmezhetünk a dokumentum-reprezentációk között.
* Elosztott feldolgozás: az igazán nagyméretű állapotterek elemzéséhez lehetőség van elosztott programozási módszereket[12] használni.

## BeautifulSoup

A programcsomag[13] egy weboldal parser programkönyvtár. A parser-ek szintaktikai analízist végeznek valamilyen szimbólumok sorozatain, jellemzően szövegeken. Általában valamilyen nyelvtan szabályai szerint értelmezik, feldolgozzák, struktúrába rendezik annak elemeit. A BeautifulSoup segítségével a weboldalak html tartalmát olvashatjuk fel, és a különböző tag paraméterek segítségével lekéréseket, kereséseket definiálhatunk az adott dokumentumon.

Unicode szöveg kinyerését is támogatja, magyar nyelvű cikkek feldolgozásához ez elengedhetetlen funkció. Első sorban a weboldalakon található cikkek szövegtörzsének, és a cikkek egyéb metaadatainak kinyerésére használom. Például a 3. ábrán látható hogy egy weboldal összes <a> tag-jéből szeretnénk kiszűrni a hivatkozások tartalmát.

3. ábra. Linkek kinyerése egy weboldalból

**for** link **in** soup.find\_all('a'):

**print**(link.get('href'))

## Django

4. ábra - Django architektúra[14]

A Django keretrendszer[15] egy Python programcsomag. Akárcsak a többi keretrendszer, a már megvalósult, gyakran újrafelhasználásra kerülő megoldásokat, komponenseket gyűjtötték benne össze. Python alapú webes alkalmazások fejlesztéséhez készítették és a mai napig folyamatosan fejlesztik. A Django Model-View-Controller[16] alapokra támaszkodik, ahogy a 4. ábrán is látható. Az automatikusan létrehozott alkalmazást a „manage.py” előre elkészített program segítségével tudjuk menedzselni. Az adatbázis és a modell közötti szinkronizáció automatikus, és az alap architektúra felállítását is elvégzi az általunk megadott paramétereknek megfelelően a keretrendszer.

Egy tipikus weboldal létrehozása a következő lépésekből áll:

1. Architektúra, könyvtárszerkezet kialakítása, szerverkonfigurációk élesítése a manage.py segítségével. Ekkor már el tudjuk indítani a webalkalmazást, az agilis módszertanoknak megfelelően azonnal működőképes a webszerverünk, melybe beleintegrálva fogjuk létrehozni a webalkalmazásunkat.
2. Modell létrehozása a models.py-ban. Az általunk létrehozott objektumok adatreprezentációját a manage.py segítségével hozzuk létre az adatbázisban, szinkronizációjuk teljesen automatikus.
3. Adminisztrátor felület kialakítása: admin.py segítségével. A klasszikus adminisztrátori felület előre el van készítve, bővítése egyszerű, kevés munkát igényel.
4. URL-ek beállítása: urls.py-ban fel tudjuk készíteni az alkalmazást, mely kéréshez mely útvonal, mely controller társul.
5. View-k definiálása: a templates mappában létrehozhatjuk azokat a html oldalakat, amelyek a felhasználó oldalán meg fognak jelenni. Hozzáköthetjük a saját változóinkat és később dinamikusan rakhatjuk össze az építő elemeket, de a html azonban első sorban csak a statikus megjelenítésért felel.
6. Controller-ek definiálása: a views.py-ban neve ellenére első sorban controller függvényeket definiálunk. Az url-eknek megfelelően ide futnak be a különböző GET és POST http kérések, az itteni függvények dolgozzák fel ezeket. Ide illeszthetjük be a saját service-ek hívását és az üzleti logikát: tehát mindent, ami az alkalmazás irányítását vezérli. A controller osztályok generálják le a konkrét html oldalakat, irányítják át a kéréseket, menedzselik a modell entitásokat is.

Természetesen ezek mellett rengeteg mást is megtehetünk, a stíluslapok definiálástól egészen az előre elkészített automatikus űrlapok létrehozásáig. Gyakorlatilag majdnem minden, manapság felmerülő, webalkalmazásokkal szemben állított követelményekhez fel van készítve. Az egyszerűbb app-ok létrehozásához első rendű megoldás, a feladatom implementálásához ideális választás volt.

# LabelerApp architektúra

Az elkészült webalkalmazást Python nyelven, a Django webalkalmazás segítségével készítettem el. Az 5. ábrán látható az alkalmazás váza.

## Alkalmazás architektúra

* A LabelerApp mappában találhatóak a webszerver futtatásához szükséges fájlok: a webszerver beállításai, és az URL-ek továbbirányításához szükséges lista.
* Az editor mappában szerepel a belső működés megvalósítása.
  + admins.py: admin felület beállításai.
  + forms.py: előre definiált űrlapok.
  + models: adatbázis szerkezet és logika.
  + tests.py: működés tesztelésének beállítása.
  + urls.py: további url-ek értelmezése.
  + views.py: controller metódusok.
* A migrations mappa tartalmazza az adatbázis szinkronizációjáért felelős fájlokat.
* A templates mappában a megjelenítendő weboldalak statikus sablonjai találhatóak, mint például a cikkek listázása, vagy egy cikk megjelenítése, szerkesztése.
* A topicmodeller mappában a szövegbányászati logika és a parse-olás service-ek megvalósítása található.
* A static mappában a különböző erőforrások találhatóak.

## Topic modeller service

A topic modell service kialakítását gensim segítségével hoztam létre. A service működése úgy valósul meg, hogy a webalkalmazás meghívja a yieldLabels(article, label) függvényét. Az article paraméterhez egy nagyméretű szöveges változót vár, ami a cikk szövege. A label paraméter a cikkhez tartozó címkéket tartalmazza.

5. ábra. Az alkalmazások struktúrája

Visszatérési értéke egy olyan szöveges változó, amely tartalmazza a kapott cikk címkéit, kiegészítve a szövegbányászati elemzés eredményeként születő új címkékkel.

Ennek lépései:

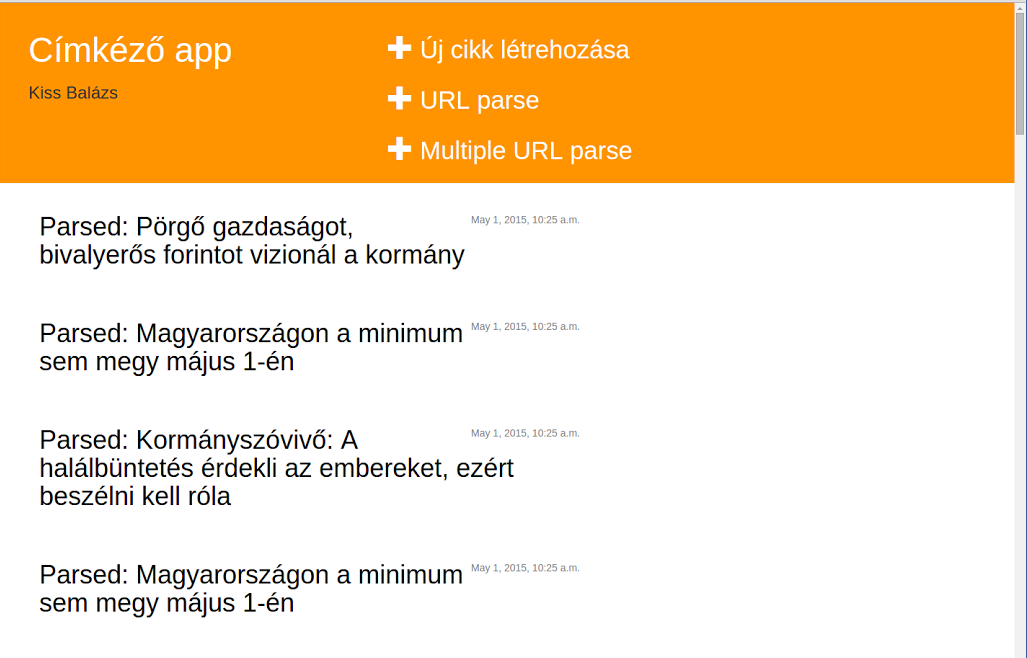
1. Definiálok egy stopszó-listát[17].
2. Létrehozom, vagy ha már létezik, akkor beolvasom a helyi diszken tárolt korábbi dokumentumokat, címkéket, és megtoldom őket a kapott új adatokkal.
3. Ezek után tokenizálom (szavakra bontom) a kapott szövegeket, majd elkészítem a dokumentumtér szótárát.
4. Transzformálom az adatokat Bag of words szerkezetbe.
5. Kétdimenziós vektortér transzformációt hajtok létre a Bag of words-ből Tf-idf-be.
6. Létrehozom az Lsi modellt. A jelenlegi megoldás két dimenziós térképet hoz létre a legfontosabb szavak alapján.
7. Megkeressük a térképen az adott cikkhez legközelebb álló másik cikket a corpusból.
8. A legközelebb álló cikkhez tartozó címkéket hozzáillesztem a bemeneti label paraméterhez, és visszatérek vele.

## A webParser service

BeautifulSoup segítségével több service-t is definiálok

* parse\_content(soup): „soup” tipusú objektumból nyerem ki a <p> tag-ek között lévő szövegeket. Belső eszköz, Clean Code[18] konvenció miatt kiemelve a többi függvényből (a Clean Code elveket a munkám során folyamatosan igyekeztem betartani).
* getWebContent(urlText): segítségével a kapott szövegben megkeresem az összes URL-t, és a mögöttük található HTML tartalmat soup objektumba csomagolva értelmezem, visszaadom a szöveges tartalmukat.
* parse\_title(urlText), getPostLabels(urlText): hasonlóan az előző függvényhez, itt a címkék illetve a cím után keresek, metaadatok és egyéb leírók alapján.

## Működés:

A weboldal kezdőképernyőjén láthatjuk a funkcionalitásokat: Új cikk létrehozásának segítségével egy olyan űrlapot tölthetünk ki, amivel egy új cikk objektumot hozhatunk létre. Az URL parse segítségével egy megadott weboldal szöveges tartalmát nyerhetjük ki, míg a Multiple URL parse több különböző webes cikk tartalmát képes felolvasni, és egyből letárolni

6. ábra. Az alkalmazás kezdőoldala

# Eredmények és további fejlesztési lehetőségek

Az 7. ábrán látható a project static/links\_for\_demo fájl tartalmának felolvasása után az egyik cikk megjelenítése. A „|” vonal jelöli a cikk eredeti, és az ajánlott címkék határát. Látható, hogy a cikk témájából és az adatbázisban található korábbi cikkekből releváns javaslatot tett a program.

## 

7. ábra. Egy példa cikk és a címkéi

A pontosság és használhatóság megítélése a gyakorlat próbájára vár. A project további fejlesztési lehetőségei között szerepel egy olyan webes eszköz megalkotása, amely nem csak az előre definiált weboldal-struktúrákat tudja felolvasni, hanem tetszőlegeset, illetve nagyobb testreszabhatóságot biztosít a címkézésekhez.

Az alkalmazás eme formájában úgy készült el, hogy a service különálló egységet alkot a webalkalmazástól, mobilis és könnyen beépíthető más webalkalmazásokba, online szövegszerkesztőkbe is. Funkcionalitását tovább bővítve elképzelhető esetleg témák kijelölése, vagy csak az adott cikk tartalma alapján témajelölő címkék ajánlására is.

# Összefoglalás

1. Felelevenítettem az adatbányászatról, szövegbányászatról tanult ismereteket a feladathoz szükséges új fogalmakkal kiegészítve azt.
2. Elsajátítottam a Python nyelv alapjait, megtanultam egyszerű webalkalmazást készíteni a Django keretrendszer segítségével
3. Elkészítettem a feladat megvalósításához szükséges webalkalmazást és hozzá a megfelelő szövegbányászatot megvalósító service-eket (nagyjából 700 sor).
4. Éles adatokkal teszteltem az elkészült logikát, többször finomítottam, javítottam rajta, mire elfogadható eredményekre jutottam.
5. Új (parse) funkciókkal bővítettem a webalkalmazást.

# Irodalom, és csatlakozó dokumentumok jegyzéke

1. E-mail postafiók adatbányászati elemzése (2014), Kiss Balázs, Gáspár Csaba, BME TMIT,
2. Dokumentum osztályozás adatbányászati módszerekkel (2014), Kiss Balázs, Nagy Gábor, BME TMIT,
3. K-Monitor közhasznú szervezet   
   <http://k-monitor.hu>
4. Dr. Abonyi J. 2006. *Adatbányászat, a hatékonyság eszköze* ISBN:963-618-432-2, ComputerBooks
5. Tikk D. 2007. *Szövegbányászat* ISBN 978-963-9664-45-6, Typotex
6. Daphna W. „Bags of words model” 2014 Hebrew University of Jerusalem <http://www.cs.huji.ac.il/~daphna/course/student%20lectures/noam%20aigerman.pdf>
7. David M. B. „Introduction to Probabilistic Topic Models” 2011 Princeton University <https://www.cs.princeton.edu/~blei/papers/Blei2011.pdf>
8. Christ M. „Tf-idf: A single-page tutorial”   
   <http://www.tfidf.com>
9. Amy Langville „The Linear Algebra Behind Search Engines – And Andvanced Vector Space Model: LSI”Mathematical Association of America 2005 <http://www.maa.org/publications/periodicals/loci/joma/the-linear-algebra-behind-search-engines-an-advanced-vector-space-model-lsi>
10. Gérard Swinnen 2005 *Tanuljunk meg programozni Python nyelven* <http://mek.oszk.hu/08400/08435/08435.pdf>
11. Radim Ř. 2009 „Gensim”   
    <https://radimrehurek.com/gensim>
12. ETH Zurich „Principles of Distributed Computing (lecture collection)” <http://dcg.ethz.ch/lectures/podc_allstars/>
13. Leonard R. 2015 „BeautifulSoup” <http://www.crummy.com/software/BeautifulSoup>
14. „Django Architecture” MyTARDIS: <https://mytardis.readthedocs.org/en/latest/_images/DjangoArchitecture-JeffCroft.png>
15. „Django” Django Software Foundation 2015   
    <https://www.djangoproject.com>
16. Steve Burbeck, Ph.D „How to use Model-View-Controller (MVC)” Smalltalk Archive 1992 <http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html>
17. „Hungarian stop words” Tutorial.hu. 2009   
    <http://www.tutorial.hu/hungarian-stop-words>
18. Robert C. Martin 2008 *Clean Code: A Handbook of Agile Software Craftsmanship* ISBN-13: 978-0132350884

**A webalkalmazás elérhetősége:**

* Archívum letölthető innen: <http://home.sch.bme.hu/~dczukt/upload/LabelerApp.tar>
* Futtatáshoz szükséges eszközök: Python 3.3.3, Python 2.7.1, Gensim, BeautifulSoup
* Project futtatásához léteznie kell a /tmp/onlab mappának (vagy át kell állítani ezt az elérést)
* Indítás: „./manage.py runserver” ez után a 127.0.0.1 címen elérhető az alkalmazás.