**СЕМИНАРСКА ПО ПРЕДМЕТОТОТ**

**Имплементација на системи со отворен код (програмски практикум)**

**Тема: Имплементација на речник на македонскиот јазик и техники за NLP**

**Кандидат: Филип Димовски**

**Индекс: 131056**

**Насока: КНИ**

***Содржина***

Абстракт ........................................................................................................... 3

Вовед................................................................................................................. 4

Методологија..................................................................................................... 5

Решение на проблемот................................................................................... 15

Резултати...........................................................................................................17

Дискусија............................................................................................................20

Литература.........................................................................................................21

Додатоци............................................................................................................22

***Абстракт***

  Проектот се состои од  crawling (влечење) низ македонски веб страници од различни теми и  преземање на нивната содржина. Содржината на веб страницата е зачувана во посебна датотека, како и сите линкови низ кои нашиот „crawler“ поминал. Баесовиот класификатор понатаму беше истрениран и со помош на тестирачкото множество ги добивме следните заклучоци. За работа искористен е Python, користен e Toolkit (алатката) nltk (natural language Tollkit) за одделување на зборовите и naive bayes во Java за класификација.

Проектот е направен за да може да се препознава суштината на речениците кои се даваат на влез. Со ова полесно би можело да се направи класификација за дадени коментари,текстови, колумни,...

***Вовед***

 NLP (Natural Language Processing) e поле од компјутерските науки, вештачката интелигенција и  лингвистиката која има за цел создавање апликации и информатички техники за анализа на секој аспект од природниот јазик.  Главен предизвик на NLP e  разбирање на природниот јазик, односно дозволување на компјутерите да го дознаат значењето на корисничкиот влез, со што NLP  е длабоко поврзан и со областа интеракција човек-компјутер.

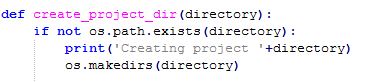
Откако Алан Туринг ја објавил колумната „Компјутери и интелигенција“  во 1950 година, NLP започнува да станува интересна тема. Посебно со „тестот на Туринг“ во кој човек-судија  после комуникација треба да одлучи дали комуницира со човек или со компјутер. Понатаму следуваат разни експерименти, меѓу кои  и на Џорџтаун кои автоматски превел повеќе од 60 реченици од руски во англиски јазик вo 1960тите.

Денес , со посебна заслуга на IBM Research, создадени се покомплексни статистички модели. Овие системи се користат во парламентот на Канада и парламентот на Европската Унија каде официјално се користат повеќе јазици.

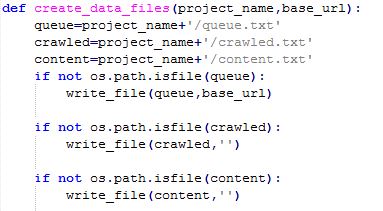
Цел на оваа семинарска е да се извлечат зборови од различни  македонски веб страници, да се истренира алгоритамот и со одредена ефикасност да може да ги погоди значењата на идните реченици кои доаѓаат на влез.

***Методологија***

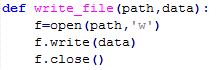
Проектот започнува со прибирање на зборови од македонскиот јазик од различни македонски веб сајтови. Секој веб сајт претставува посебен проект- односно се креира посебна папка каде ќе се чуваат информациите за него. За таа цел, во Python се импортира од OS и дефинираме функција за креирање на папка со соодветно дадено име:



За секој проект треба да имаме три датотеки: *queue.txt* (се чуваат линковите кои треба да се поминат), *crawled.txt (*се чуваат линковите кои се поминати*)*  и *content.txt* (се чува содржината на веб страницата). Креирањето на овие датотеки е дадено со следниот код:

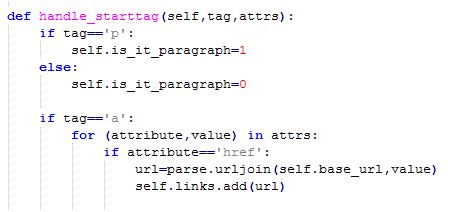


Во оваа функција како параметри се земаат името на проектот и линкот од веб-страницата. Пред креирањето на датотеките се проверува дали датотеката постои. Ако датотеката не постои, тогаш датотеката се креира со повикување на функцијата write\_file која прима два параметри- патеката и содржината која сакаме да ја запишеме внатре во датотеката. Во датотеката queue.txt се сместува линкот до веб страницата, бидејќи тој е првиот линк кој сакаме да го поминеме. Функцијата write\_file e дадена со следниот код:



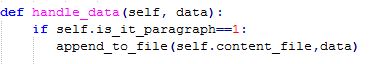
Датотеката се креира и се отвора со наредбата ‘w’(write) – со што се дозволува пристап за запишување во неа и доколку веќе датотеката содржела податоци, тие се бришат. Во датотеката general.py се дефинирани и други функции кои ќе бидат објаснети подолу кога ќе бидат и користени.

Следно, дефинираме класа LinkFinder во датотеката link\_finder.py, која наследува од HTMLParser. За оваа цел, мора да импортираме html.parser од HTMLParser. Исто така, импортираме и urlib од parse и се што имавме од датотеката general. Во конструктурот на класата преземаме три парамтери: линк до основниот веб сајт, линк до моменталната веб страница и името на проектот. Дефинираме и множество *links* каде ќе ги чуваме сите линкови кои се наоѓаат во моменталната веб-страница, знаме за дали елементот е параграф или не и патеката до датотеката каде ја чуваме содржината на веб страницата. Со помош на HTMLParser ние го преземаме HTML кодот на страницата и за ова можеме да ги искористиме двата метода- handle\_starttag и handle\_data. Првиот метод го зема почетокот на елементот, a вториот метод ја зема содржината на елементот. Пример, доколку имаме <a href=”[www.finki.ukim.mk](http://www.finki.ukim.mk)”> Content of the anchor </a> , handle\_starttag ќе го преземе делот <a href=”[www.finki.ukim.mk](http://www.finki.ukim.mk)”> , a handle\_data ќе го преземе делот „Content of the anchor“. Дефиниција на методот handle\_starttag:



Овој метод прима две вредности на влез- името на елементот и сите негови атрибути. На почетокот правиме проверка дали елементот е параграф и соодветно сигнализираме со знамето што претходно го дефиниравме. Потоа правиме проверка дали елементот е линк. Бидејќи линкот е дефиниран во атрибутот href на елементот <a>, правиме итерација низ атрибутите на елементот се додека не го најдеме соодветниот атрибут. Линкот може да биде претставен на два начина: апсолутна и релативна патека. За оваа цел, ја користиме готовата функција parse.urljoin од соодветно искористената библиотека и без разлика на кој начин е даден линкот, се креира линк со апсолутна патека. Функционалниот линк го поставуваме во дефинираното множество од линкови.

Дефиницијата за методот handle\_data e краток и се состои од проверка дали елементот е параграф и доколку е, содржината негова се додава на крајот од датотеката во која се чува содржината – *content.txt*:



Овој метод прима на влез само податокот кој се наоѓа во елементот. За додавање на содржината на крајот на датотеката, се користи функцијата *append\_to\_file* од датотеката general.py:

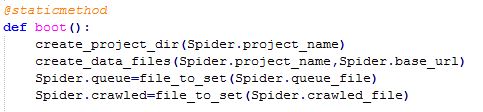
D:\Sedmi_Semestar\PHP\Seminarska\photos_for_the_project\append_to_file.JPG

Оваа функција прима два аргумента: патека до датотеката во која сакаме да запишеме и содржината со која ќе биде потполнета. Датотеката се отвора со наредбата ‘a’- аppend со што содржината се додава на крајот. По додавањето се остава нов ред.

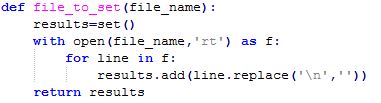
На крајот на класата дефинирани се уште два метода. Првиот метод page\_links со што се враќаат сите линкови кои се наоѓаат на страницата, односно се враќа дефинираното множество links. Вториот метод е error и мора да биде проследен при наследување на HTMLParser. Како аргумент се дава порака од грешката, која може да му се сигнализира на корисникот. Во случајов, методот е празен и затоа се пишува pass.

Следно за пишување е класата Spider кој се наоѓа во датотеката spider.py. Работата на „пајакот“ е дадената веб-страница да ја помине целосно- заедно со сите нејзини линкови кои се дел од истиот проект. Во оваа датотека импортираме urlopen од urlib.request, LinkFinder од link\_finder и сè што имаме од general.py. На почетокот се дефинираат неколку класни променливи кои можеме да ги користиме во сите нејзини инстанци. Конструкторот на оваа класа прима три влезни параметри: име на проектот, основниот линк и име на доменот. URL-от на веб-страницата може да биде составен од неколку делови (пр. mail.yahoo.com), домен името го претставува името на веб-страницата без непотребните екстензии (yahoo.com), па затоа ни е потребен. Во конструкторот го дефинираме и името на потребните датотеките така што на името на проектот (кој ќе биде името на папката) се врзуваат и имињата на соодветните датотеки („queue.txt“, „crawled.txt“, „content.txt“).

Во класата Spider го дефинираме методот boot:



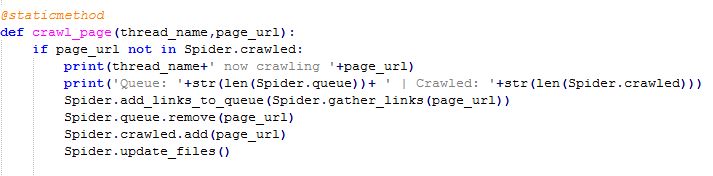
Овој метод се повикува веднаш по креирањето на инстанца и се користи за создавање на потребните датотеки и преземање на потребните податоци. На почетокот се повикува функцијата create\_project\_dir за креирање на папка со исто име како што го задаваме името на проектот. Потоа, се повикува методот create\_data\_files за создавање на потребните датотеки за секој проект. Потоа, се повикува функцијата file\_to\_set со името на датотеката каде се чуваат кои линкови се на ред да се обработат - queue\_file. Кодот на оваа функција се наоѓа во general.py:



Задачата на овој метод е да се преземе содржината на дадена датотека и да се смести во множество. Се мине секоја линија од датотеката, се сместува во резултатно множество, и на крајот ова множество се враќа.

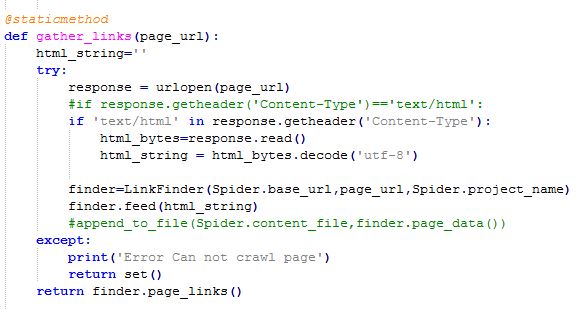
Во нашиот метод boot, методот file\_to\_set се повикува двапати- првиот пат сите линкови кои се на ред да се обработат (доколку постојат) се сместуваат во множеството queue, а вториот пат сите линкови кои се обработени се сместуваат во множеството crawled. Овој метод не зависи од креираната инстанца, па затоа е обележан како статички.

Следно го дефинираме методот crawled\_page:



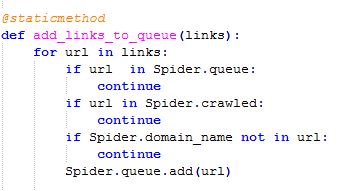
Овој метод се повикува во конструктурот откако ќе се повикува методот boot. Се задава името на „пајакот“ –бидејќи може да имаме повеќе и линкот кои сакаме да го обработиме. На почетокот проверуваме дали линкот е веќе обработен – ако не е тогаш го известуваме корисникот кој линк се обработува од страна на кој пајак. Исто така и колку линкови се останати за обработка и колку од нив се веќе поминати. Со помош на методот gather\_links (дефинирана подолу) се преземаат сите линкови од дадената страница и со помош на функцијата add\_links\_to\_queue (дефинирана подолу) се сместуваат податоците во множеството queue. На крај, линкот се отстранува од queue (множество од линкови кои се следни за обработка) и се сместува во линкови кои се веќе обработени (crawled). Како и множествава, така и соодветните документи се апдејтираат со помош на методот update\_files кој ќе биде објаснет во продолжение.

Како што напоменавме, методот gather\_links се користи за да се преземат сите линкови од дадена страница:



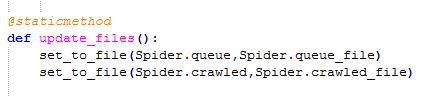
Со помош на библиотеката urllib.request го импортираме urlopen со цел да го отвориме линкот. На почетокот проверуваме дали станува збор за html (може да биде pdf документ на пример) и ја читаме неговата содржина. Резултатот кој се враќа е во бајти, па затоа мора да го декодираме со utf-8. Потоа правиме инстанца од претходно дефинираната класа LinkFinder и со методот feed (достапен бидејќи наследуваме од HTMLParser) ги преземаме соодветните податоци. На крајот го враќаме множеството од сите линкови кои можеме да ги најдеме од оваа веб – страница. Доколку е настаната некаква грешка, му се сигнализира соодветно на корисникот.

Методот add\_links\_to\_queue се користи за додавање на преземаните линкови во дефинираното множество queue:

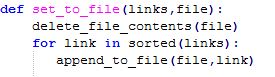


На влез се носи множество од линкови- резултатот од методот gather\_links. Линковите се итератираат и за секој линк се проверува дали веќе се наоѓа во множествата queue и crawled. Исто така, правиме и проверка дали името на доменот се наоѓа во URL-от, бидејќи може да се префрлиме на некоја социјална мрежа и пајакот да почне да собира линкови од таму. Доколку се е во ред, линкот се додава во множеството queue.

Сите измени кои ги направивме во множествата queue и crawled, треба да ги зачуваме и во соодветните документите. Па затоа, го користиме методот update\_files:



Со овој метод, сите линкови кои ги имаме во множествата queue и crawled , се поставуваат во соодветните документи со помош на функцијата set\_to\_file oд документот general.py:



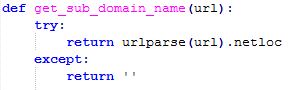
На почетокот, се брише целосно содржината на документот со помош на методот delete\_file\_contents (дефиниран подолу) и потоа секој линк (сортиран) се додава на крајот на соодветниот документ.

Функцијата delete\_file\_content е дефинирана како:

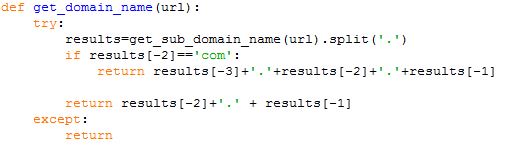
D:\Sedmi_Semestar\PHP\Seminarska\photos_for_the_project\delete_file_content.JPG

Со самото отворање на документот во модул „w“-write, неговата содржина се брише.

Следно за пишување е како да се преземе доменот на страницата за даден url –ова е дефинирано во датотеката domain.py. За оваа цел, импортираме urlparse од urllib.parse и ја дефинираме функцијата get\_sub\_domain\_name:



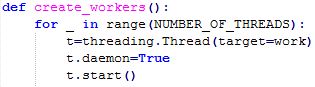
Со оваа функција се враќа основниот URL за дадена веб-страница. Се користи функцијата urlparse и се враќа netloc oд резултатот. Пример, ако на влез ја имаме веб-страницата 'http://www.example.com.mk/asd/dsa/as.html', како резултат ќе имаме ‘example.com.mk’. За да бидеме сигурни дека нема да имаме непотребни екстензии, дефинираме уште една функција get\_domain\_name:



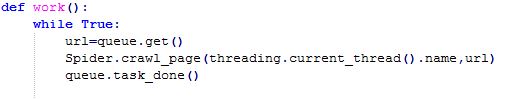
Резултатот кој го добиваме од функцијата get\_sub\_domain\_name го разделуваме на парчиња и враќаме соодветно според форматот на URL-oт: ако завршува на .com.mk или пак само .com или пак само .mk.

На крајот ја дефинираме и датотеката main.py и нејзините функции. Се импортира Queue од queue, Threading, и сè што имаме дефинирано во датотеките domain, spider и general .

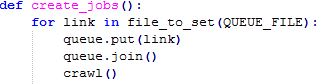
На почетокот, корисникот ги специфицира името на проектот, линкот до веб страницата и бројот на пајаци кои паралелно ќе работат. Името на доменот се дознава од линкот до веб страницата и соодветно се креираат и имињата на датотеката каде се чуваат следните линкови за обработка (queue.txt) и датотеката со линкови кои се обработени(crawled.txt). Се креира првиот пајак и тој започнува со работа. Потоа се креираат дополнителни работници со помош на методот create\_workers:



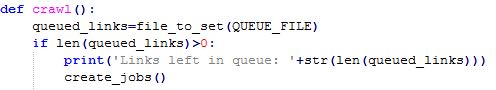
Откако корисникот специфицирал број на пајаци кои ќе работат на прибирањето на податоци, за секој пајак се креира специјален Thread и му се задава работа дефинирана со функцијата work:



Се презема линкот од queue и му се задава на пајакот да ја обработи дадената задача. Queue се апдејтира постојано преку методот:



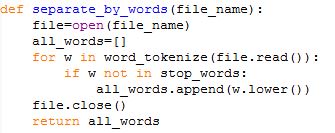
Во овој метод се зема секој линк од датотеката Queue\_file и се поставува во queue. На крајот се повикува методот crawl:



Со овој метод се проверува дали има останато линкови за обработка во соодветната датотека. Ако има, тогаш се креираат соодветни работи за пајаците, ако нема, тоа значи дека сите линкови се поминати и тука завршува апликацијата.

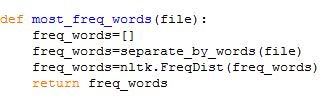
Следниот дел е за обработка на податоците. Кодот се наоѓа во датотеката text\_classificаtion.py каде се импортира од nltk – за обработка на природните јазици, random и word\_tokenize oд nltk.tokenize.

Го дефинираме методот separate\_by\_words:



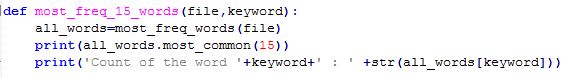
На влез се дефинира името на документот, документот се отвора, се разделува на зборови и секој збор се додава во листата која се враќа на излез. Поделбата на документот на зборови се прави со функцијата word\_tokenize која ја импортиравме од nltk.tokenize. Дефинирана е листа од зборови stop-words – зборови кои немаат никакво влијание врз резултатот, па затоа ги користиме во пресметките.

Потоа дефинираме функција кој ги враќа најфреквентните зборови:



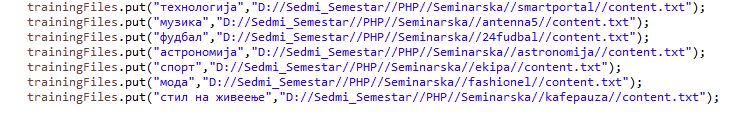
Методот most\_freq\_words на влез го зема името на документот кој треба да го процесира. Ги зема сите зборови од документот и преку готовата функција FreqDist од nltk, создава листа од најфреквентните зборви и ги враќа како излез.

За приказ на корисникот, креирана е функцијата most\_freq\_15\_words , каде се прикажуваат 15те најфреквентни зборови од секоја категорија и за даден збор се прикажува колку пати тој се покажува:



Се земаат најчестите зборови со помош на претходно дефинираната функција и се печатат првите 15 зборови. Исто така, за даден збор се печати колку пати и тој се повторува.

На крај, во проектот во Java ги поставуваме соодветно нашите документи според кои ќе се врши класификација и нивната категорија:



***Решение на проблемот***

   Проектот е поделен на два дела: преземање на податоци од веб-страниците и нивна обработка. Започнато е со преземање на податоци - најпрво одбрани се неколку македонски  веб-страници со различни теми- музика (www.antenna5.mk), спорт (www.ekipa.mk), технологија (www.smartportal.mk), мода ([www.fashionel.mk](http://www.fashionel.mk)), фудбал ([www.24fudbal.com.mk](http://www.24fudbal.com.mk)), стил на живот- lifestyle ([www.kafepauza.mk](http://www.kafepauza.mk)) и астрономија ([www.astronomija.mk](http://www.astronomija.mk)). Темите како фудбал и спорт не се разделни (disjoin) и земени се во предвид да видиме како алгоритамот ќе ги класифицира дадени елементи. Исто, „стил на живот“ може  да има содржина како мода и технологија.

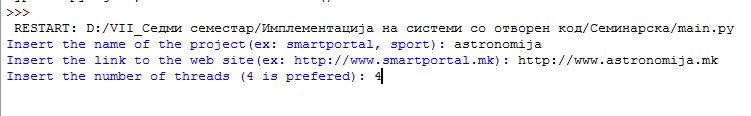
За секоја веб страница се создава посебен проект(посебна папка) и автоматски се креираат три документи- content.txt (се зачувува содржината), queue.txt(се зачувува кои линкови се следни за обработка) и  crawled.txt(се зачувува кои линкови биле обработени). Користена е готовата библиотека во python за обработка на веб-страници - HTMLParser каде од дадена страница го зема нејзиниот HTML код. Од HTML кодот ја земаме содржината од параграфите и ја сместуваме во content.txt, а линковите од anchor елементите ги сместуваме во листа. Откако страницата ќе се обработи, нејзиниот линк ќе се смести во crawled.txt, а преземаните линкови во queue.txt. Во main.py дефиниран е интерфејсот на апликацијата- корисникот внесува име за проектот, линкот до веб страницата и колку „пајаци“  да се искористат- колку работници да работат истовремено - threading. „Пајаците“ читаат од датотеката queue.txt , гледаат кои линкови се на ред и ја обработуваат веб страницата со соодветниот линк.

Вториот дел се состои од обработка на податоците, т.е на content.txt на секоја веб страница што ја одбравме. Со помош на готовата функција word\_tokenize содржината ја разделуваме на зборчиња. Користиме stop-words, односно зборови кои не сакаме да бидат земени во предвид при класификацијата.

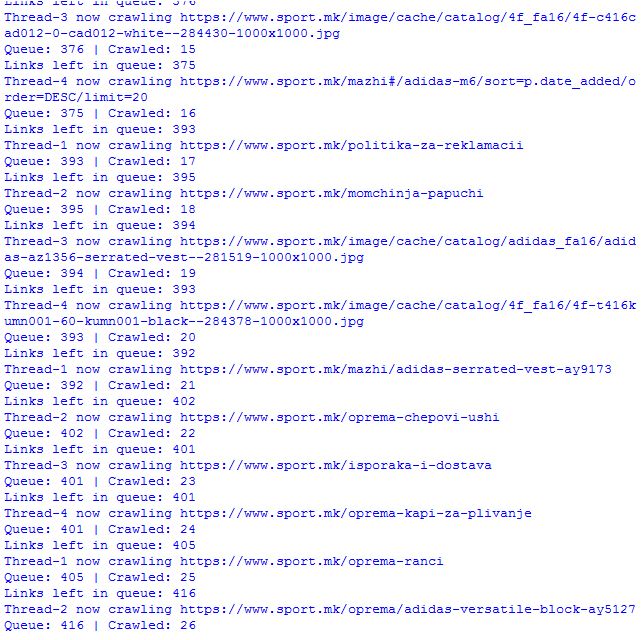
Со помош на функцијата МоstFreq во nltk, ги земаме најфреквентно користените зборови од секоја веб-страница. На екран ги печатиме 15те најчести зборови, и за даден збор можеме да пресметаме колку пати се појавил. Го користиме готовиот naïve bayes класификатор во Java за класификација на зборовите. Целосните содржини од нашите веб-страници ги користиме како тренирачко множество и потоа класификаторот веќе може со одреден процент да ги класифицира идните реченици.

***Резултати***

Создавање на проект за класификација на текст и нејзино користење за препознавање на темата на реченицата. На почетокот, од корисникот се побарува да ги внесе името на проектот, линк до веб страницата и бројот на пајаци кои ќе работат паралелно:

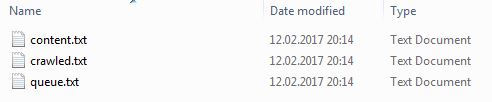


Креиран е соодветен интерфејс во Python за прибирање на соодветните податоци. За внесените податоци од корисникот започнува да се обработува веб страницата:

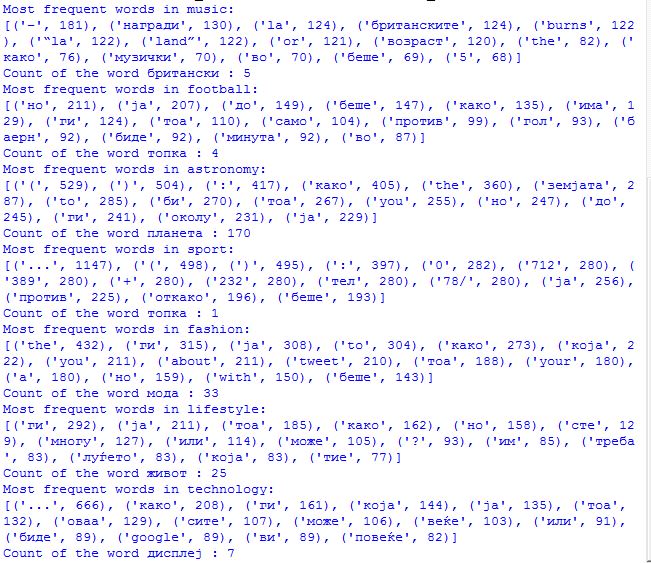


При прибирањето на информации, корисникот е постојано извесуван за тоа кој thread по ред која страница се обработува, како и колку страници се обработени и колку чекаат на ред.

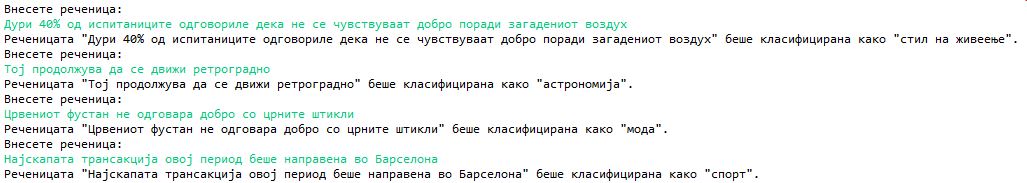
На крајот како резултат се креира посебна папка со името на проектот, и во неа ги има соодветните датотеки:



Потоа, следува класификација на текстовите. Најпрво на корисникот му се прикажуваат најчестите 15 зборови од секоја класа и колку пати даден збор се појавува:



На крај, во naïve bayes проектот во Java, класификаторот е истрениран и ги дава следните резултати за следниве влезови:

******

Но, исто така некои реченици ги класифицира грешно:

D:\Sedmi_Semestar\PHP\Seminarska\photos_for_the_project\primer_gresno.JPG

***Дискусија***

Прибирање на зборови од различни македонски веб-страници и нивна класификација.

На почетокот на корисникот му се дава избор за спецификација за името на проектот, линк до веб страницата, како и да специфицира број на работници кои ќе работат паралелно. Обработката на веб-страницата се одвива брзо и корисникот е постојано известен кој линк се обработува од кој работник, колку линкови се обработени и колку чекаат на ред. Потоа, при класификацијата создаден е повторно интерфејс со што корисникот само ја внесува реченицата и резултатот кој го добива е брз.

Во Python постои scikit-learn алатка која се користи за класификација и машинско учење. Multinomial Naïve bayes алгоритамот што постои во оваа алатка работи на ист начин како и naïve bayes проектот во Java. Класификаторот е целосно преземан од корисникот <https://github.com/datumbox> ( линкот е проследен во делот „Литература“).

Во иднина проектов може да се прошири така што ќе се одберат повеќе македонски веб-страници од различен спектар на теми. Може да се искомбинираат повеќе класифицирачки алгоритми и да се најде нивна средина која би се вратила како резултат. Би можеле да дефинираме и одреден праг на сигурност, со што ако класификаторот не е сигурен, не мора да врати резултат или пак да каже кои теми се најверојатни.

***Литература***

За преземањето на податоците од македонски веб- страници беа искористени веб-страниците:

* <http://www.24fudbal.mk>
* <htttp://www.antenna5.mk>
* <http://www.astronomija.mk/>
* <http://www.ekipa.mk>
* <http://www.kafepauza.mk>
* <http://www.smartportal.mk>
* <http://www.fashionel.mk>

Тие беa обработени со помош на информациите од:

* <https://thenewboston.com/>

За класификација на податоците како помош служеше веб-сајтот:

* <https://github.com/datumbox/NaiveBayesClassifier>
* <http://www.sentdex.com/>

***Додатоци***

Користена е програмата Python 3.5.2 (HTMLParser, nltk,urllib) и Еclipse Mars.