

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 е Toolkit (алатката) nltk (natural language Toolkit) за одделување на зборовите и naïve 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) е 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 е 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 Тјуринг ја објавил колумната „Компјутери и интелигенција“ во 1950 година, NLP започнува да станува интересна тема. Посебно со „тестот на Тјуринг“ 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 во 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 алгоритмот 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:

```
def create_project_dir(directory):  
    if not os.path.exists(directory):  
        print('Creating project ' +directory)  
        os.makedirs(directory)
```

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:

```
def create_data_files(project_name,base_url):  
    queue=project_name+'/queue.txt'  
    crawled=project_name+'/crawled.txt'  
    content=project_name+'/content.txt'  
    if not os.path.isfile(queue):  
        write_file(queue,base_url)  
  
    if not os.path.isfile(crawled):  
        write_file(crawled,'')  
  
    if not os.path.isfile(content):  
        write_file(content,'')
```

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` е дадена со следниот код:

```
def write_file(path,data):  
    f=open(path,'w')  
    f.write(data)  
    f.close()
```

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 `urllib` од `parse` и се што имавме од датотеката `general`. Во конструкторот на класата преземаме три параметри: 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, знаеме 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, а вториот метод ја зема содржината на елементот. Пример, доколку имаме ` Content of the anchor ` , `handle_starttag` ќе го преземе делот `` , а `handle_data` ќе го преземе делот „Content of the anchor“. Дефиниција на методот `handle_starttag`:

```
def handle_starttag(self,tag,attrs):  
    if tag=='p':  
        self.is_it_paragraph=1  
    else:  
        self.is_it_paragraph=0  
  
    if tag=='a':  
        for (attribute,value) in attrs:  
            if attribute=='href':  
                url=parse.urljoin(self.base_url,value)  
                self.links.add(url)
```

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` е 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*:

```
def handle_data(self, data):  
    if self.is_it_paragraph==1:  
        append_to_file(self.content_file,data)
```

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`:

```
def append_to_file(path,data):  
    with open(path,'a') as file:  
        file.write(data+'\n')
```

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'- `append` 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 urllib.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:

```
@staticmethod
def boot():
    create_project_dir(Spider.project_name)
    create_data_files(Spider.project_name, Spider.base_url)
    Spider.queue=file_to_set(Spider.queue_file)
    Spider.crawled=file_to_set(Spider.crawled_file)
```

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:

```
def file_to_set(file_name):
    results=set()
    with open(file_name,'rt') as f:
        for line in f:
            results.add(line.replace('\n',''))
    return results
```

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`:

```
@staticmethod
def crawl_page(thread_name, page_url):
    if page_url not in Spider.crawled:
        print(thread_name+ ' now crawling ' +page_url)
        print('Queue: ' +str(len(Spider.queue))+ ' | Crawled: ' +str(len(Spider.crawled)))
        Spider.add_links_to_queue(Spider.gather_links(page_url))
        Spider.queue.remove(page_url)
        Spider.crawled.add(page_url)
        Spider.update_files()
```

Овој метод се повикува во конструкторот 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 множествата, 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:

```
@staticmethod
def gather_links(page_url):
    html_string=''
    try:
        response = urlopen(page_url)
        #if response.getheader('Content-Type')== 'text/html':
        if 'text/html' in response.getheader('Content-Type'):
            html_bytes=response.read()
            html_string = html_bytes.decode('utf-8')

        finder=LinkFinder(Spider.base_url,page_url,Spider.project_name)
        finder.feed(html_string)
        #append_to_file(Spider.content_file,finder.page_data())
    except:
        print('Error Can not crawl page')
        return set()
    return finder.page_links()
```

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:

```
@staticmethod
def add_links_to_queue(links):
    for url in links:
        if url in Spider.queue:
            continue
        if url in Spider.crawled:
            continue
        if Spider.domain_name not in url:
            continue
        Spider.queue.add(url)
```

На влез се носи множество од линкови- резултатот од методот gather_links. Линковите се итерираат 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:

```
@staticmethod
def update_files():
    set_to_file(Spider.queue, Spider.queue_file)
    set_to_file(Spider.crawled, Spider.crawled_file)
```

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 од документот general.py:

```
def set_to_file(links, file):  
    delete_file_contents(file)  
    for link in sorted(links):  
        append_to_file(file, link)
```

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:

```
def delete_file_contents(path):  
    with open(path, 'w') as file:  
        pass
```

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`:

```
def get_sub_domain_name(url):  
    try:  
        return urlparse(url).netloc  
    except:  
        return ''
```

Со оваа функција се враќа основниот URL за дадена веб-страница. Се користи функцијата `urlparse` и се враќа `netloc` од 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`:

```
def get_domain_name(url):  
    try:  
        results=get_sub_domain_name(url).split('.')  
        if results[-2]=='com':  
            return results[-3]+'.'+results[-2]+'.'+results[-1]  
  
        return results[-2]+'.' + results[-1]  
    except:  
        return ''
```

Резултатот кој го добиваме од функцијата `get_sub_domain_name` го разделуваме на парчиња и враќаме соодветно според форматот на URL-от: 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:

```
def create_workers():
    for _ in range(NUMBER_OF_THREADS):
        t=threading.Thread(target=work)
        t.daemon=True
        t.start()
```

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:

```
def work():
    while True:
        url=queue.get()
        Spider.crawl_page(threading.current_thread().name,url)
        queue.task_done()
```

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

```
def create_jobs():
    for link in file_to_set(Queue_FILE):
        queue.put(link)
        queue.join()
        crawl()
```

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

```
def crawl():
    queued_links=file_to_set(Queue_FILE)
    if len(queued_links)>0:
        print('Links left in queue: '+str(len(queued_links)))
        create_jobs()
```

Со овој метод се проверува дали има останато линкови за обработка во соодветната датотека. Ако има, тогаш се креираат соодветни работи за пајаци, 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_classification.py` каде се импортира од `nltk` – за обработка на природните јазици, `random` и `word_tokenize` од `nltk.tokenize`.

Го дефинираме методот `separate_by_words`:

```
def separate_by_words(file_name):  
    file=open(file_name)  
    all_words=[]  
    for w in word_tokenize(file.read()):  
        if w not in stop_words:  
            all_words.append(w.lower())  
    file.close()  
    return all_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:

```
def most_freq_words(file):  
    freq_words=[]  
    freq_words=separate_by_words(file)  
    freq_words=nltk.FreqDist(freq_words)  
    return freq_words
```

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 зборови 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:

```
def most_freq_15_words(file,keyword):  
    all_words=most_freq_words(file)  
    print(all_words.most_common(15))  
    print('Count of the word '+keyword+' : ' +str(all_words[keyword]))
```

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:


```
trainingFiles.put("технологија", "D://Sedmi_Semestar//PHP//Seminarska//smartportal//content.txt");
trainingFiles.put("музика", "D://Sedmi_Semestar//PHP//Seminarska//antenna5//content.txt");
trainingFiles.put("фудбал", "D://Sedmi_Semestar//PHP//Seminarska//24fudbal//content.txt");
trainingFiles.put("астрономија", "D://Sedmi_Semestar//PHP//Seminarska//astronomija//content.txt");
trainingFiles.put("спорт", "D://Sedmi_Semestar//PHP//Seminarska//ekipa//content.txt");
trainingFiles.put("мода", "D://Sedmi_Semestar//PHP//Seminarska//fashion1//content.txt");
trainingFiles.put("стил на живеење", "D://Sedmi_Semestar//PHP//Seminarska//kafepauza//content.txt");
```

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), фудбал (www.24fudbal.com.mk), стил на живот- lifestyle (www.kafepauza.mk) и астрономија (www.astronomija.mk). Темите како фудбал и спорт не се разделни (disjoin) и земени се во предвид да видиме како алгоритмот 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`. „Пајациите“ 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 `MostFreq` 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:

```
>>>
RESTART: D:/VII_Седми семестар/Имплементација на системи со отворен код/Семинарска/main.py
Insert the name of the project(ex: smartportal, sport): astronomija
Insert the link to the web site(ex: http://www.smartportal.mk): http://www.astronomija.mk
Insert the number of threads (4 is preferred): 4
```




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

```
Links left in queue: 376
Thread-3 now crawling https://www.sport.mk/image/cache/catalog/4f_fa16/4f-c416c
ad012-0-cad012-white--284430-1000x1000.jpg
Queue: 376 | Crawled: 15
Links left in queue: 375
Thread-4 now crawling https://www.sport.mk/mazhi#/adidas-m6/sort=p.date_added/o
rder=DESC/limit=20
Queue: 375 | Crawled: 16
Links left in queue: 393
Thread-1 now crawling https://www.sport.mk/politika-za-reklamaci
Queue: 393 | Crawled: 17
Links left in queue: 395
Thread-2 now crawling https://www.sport.mk/momchinja-papuchi
Queue: 395 | Crawled: 18
Links left in queue: 394
Thread-3 now crawling https://www.sport.mk/image/cache/catalog/adidas_fa16/adid
as-az1356-serrated-vest--281519-1000x1000.jpg
Queue: 394 | Crawled: 19
Links left in queue: 393
Thread-4 now crawling https://www.sport.mk/image/cache/catalog/4f_fa16/4f-t416k
umn001-60-kumn001-black--284378-1000x1000.jpg
Queue: 393 | Crawled: 20
Links left in queue: 392
Thread-1 now crawling https://www.sport.mk/mazhi/adidas-serrated-vest-ay9173
Queue: 392 | Crawled: 21
Links left in queue: 402
Thread-2 now crawling https://www.sport.mk/oprema-chepovi-ushi
Queue: 402 | Crawled: 22
Links left in queue: 401
Thread-3 now crawling https://www.sport.mk/isporaka-i-dostava
Queue: 401 | Crawled: 23
Links left in queue: 401
Thread-4 now crawling https://www.sport.mk/oprema-kapi-za-plivanje
Queue: 401 | Crawled: 24
Links left in queue: 405
Thread-1 now crawling https://www.sport.mk/oprema-ranci
Queue: 405 | Crawled: 25
Links left in queue: 416
Thread-2 now crawling https://www.sport.mk/oprema/adidas-versatile-block-ay5127
Queue: 416 | Crawled: 26
```

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:

Name	Date modified	Type
 content.txt	12.02.2017 20:14	Text Document
 crawled.txt	12.02.2017 20:14	Text Document
 queue.txt	12.02.2017 20:14	Text Document

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

```
Most frequent words in music:
[('-', 181), ('награди', 130), ('ла', 124), ('британските', 124), ('burns', 122), ('"ла', 122), ('land"', 122), ('or', 121), ('возраст', 120), ('the', 82), ('како', 76), ('музички', 70), ('во', 70), ('беше', 69), ('5', 68)]
Count of the word британски : 5
Most frequent words in football:
[('но', 211), ('ја', 207), ('до', 149), ('беше', 147), ('како', 135), ('има', 129), ('ги', 124), ('тоа', 110), ('само', 104), ('против', 99), ('гол', 93), ('баерн', 92), ('биде', 92), ('минута', 92), ('во', 87)]
Count of the word топка : 4
Most frequent words in astronomy:
[('(', 529), (')', 504), (':', 417), ('како', 405), ('the', 360), ('земјата', 287), ('to', 285), ('би', 270), ('тоа', 267), ('you', 255), ('но', 247), ('до', 245), ('ги', 241), ('околу', 231), ('ја', 229)]
Count of the word планета : 170
Most frequent words in sport:
[('...', 1147), ('(', 498), (')', 495), (':', 397), ('0', 282), ('712', 280), ('389', 280), ('+', 280), ('232', 280), ('тел', 280), ('78/', 280), ('ја', 256), ('против', 225), ('откако', 196), ('беше', 193)]
Count of the word топка : 1
Most frequent words in fashion:
[('the', 432), ('ги', 315), ('ја', 308), ('to', 304), ('како', 273), ('која', 222), ('you', 211), ('about', 211), ('tweet', 210), ('тоа', 188), ('your', 180), ('a', 180), ('но', 159), ('with', 150), ('беше', 143)]
Count of the word мода : 33
Most frequent words in lifestyle:
[('ги', 292), ('ја', 211), ('тоа', 185), ('како', 162), ('но', 158), ('сте', 129), ('многу', 127), ('или', 114), ('може', 105), ('?', 93), ('им', 85), ('треба', 83), ('луѓето', 83), ('која', 83), ('тие', 77)]
Count of the word живот : 25
Most frequent words in technology:
[('...', 666), ('како', 208), ('ги', 161), ('која', 144), ('ја', 135), ('тоа', 132), ('оваа', 129), ('сите', 107), ('може', 106), ('веќе', 103), ('или', 91), ('биде', 89), ('google', 89), ('ви', 89), ('повеќе', 82)]
Count of the word дисплеј : 7
```

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

Внесете реченица:

Дури 40% од испитаниците одговориле дека не се чувствуваат добро поради загадениот воздух

Реченицата "Дури 40% од испитаниците одговориле дека не се чувствуваат добро поради загадениот воздух" беше класифицирана како "стил на живеење".

Внесете реченица:

Тој продолжува да се движи ретроградно

Реченицата "Тој продолжува да се движи ретроградно" беше класифицирана како "астрономија".

Внесете реченица:

Црвениот фустан не одговара добро со црните штикли

Реченицата "Црвениот фустан не одговара добро со црните штикли" беше класифицирана како "мода".

Внесете реченица:

Најскапата трансакција овој период беше направена во Барселона

Реченицата "Најскапата трансакција овој период беше направена во Барселона" беше класифицирана како "спорт".

Но, исто така некои реченици ги класифицира грешно:

Компјутерот доаѓа со безжично глумче

Реченицата "Компјутерот доаѓа со безжично глумче" беше класифицирана како "астрономија".

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 алгоритмот 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 проектои 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>
- <http://www.antenna5.mk>
- <http://www.astronomija.mk/>
- <http://www.ekipa.mk>
- <http://www.kafepauza.mk>
- <http://www.smartportal.mk>
- <http://www.fashionel.mk>

Тие беа 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 Eclipse Mars.