

Skripta za vežbe

Skripta u okviru kursa
Istraživanje podataka 2
Matematički fakultet

24. sept 2018.

Sadržaj

1	Uvod	2
1.1	Istraživanje Veba - Web Mining	3
1.1.1	Izvlačenje teksta	5
1.1.2	Latentna semantička analiza	10
2	Eksperimenti	17
3	Zaključak	17
	Literatura	17

1 Uvod

Primer 1.1 *Prvi primer služi kao podsetnik na određene mogućnosti koje nam pruža pandas biblioteka. Najpre, importujemo biblioteku, potom uz pomoć read_csv vršimo učitavanje .csv fajla koji sadrži podatke koje želimo da analiziramo i na kraju vršimo ispile različitih vrednosti.*

```
import pandas as pd

def main():
    # Pravljenje Pandas DataFrame objekta od matrice podataka
    # df = pd.DataFrame([
    #     [1,2,3, 'klasa1'],
    #     [4,2.2,3, 'klasa2'],
    #     [5,3,1, 'klasa3']
    # ], columns=['prva', 'druga', 'treca', 'klasa'],
    #     index=range(1,4))

    # Ucitavanje CSV fajla u Pandas DataFrame
    df = pd.read_csv('iris.csv')

    print(df)

    # Ispis tipova kolona
    print(df.dtypes)

    # Ispis naziva kolona
    print(df.columns)

    # Ispis indeksa redova tabele
    print(df.index)

    # Ispis matrice sa vrednostima
    print(df.values)

    # Ispis reda matrice sa indeksom 0
    print(df.values[0])

    # Ispis vrednosti matrice sa indeksom 2 u redu sa indeksom 0
    print(df.values[0][2])

    # Izdvajanje kolona po nazivima
    # print(df.loc[:,['prva', 'treca']])

    # Izdvajanje kolona po indeksima
    print(df.iloc[:, range(1,3)])

    # Uslovno izdvajanje redova koji zadovoljavaju odgovarajuci uslov
    # print(df[df.prva > 2])
```

```
if __name__ == "__main__":
    main()
```

```

    sepal_length  sepal_width  petal_length  petal_width
species
0              5.1          3.5          1.4
0.2      setosa
1              4.9          3.0          1.4          0.2
setosa
2              4.7          3.2          1.3          0.2
setosa
3              4.6          3.1          1.5          0.2
setosa
4              5.0          3.6          1.4          0.2
setosa
5              5.4          3.9          1.7          0.4
setosa
...
[150 rows x 5 columns]
```

```

sepal_length    float64
sepal_width     float64
petal_length     float64
petal_width     float64
species         object
dtype: object
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
      'species'],
      dtype='object')
RangeIndex(start=0, stop=150, step=1)
[[5.1  3.5  1.4  0.2  'setosa']
 [4.9  3.0  1.4  0.2  'setosa']
 [4.7  3.2  1.3  0.2  'setosa']
```

```

    sepal_width  petal_length
0              3.5          1.4
1              3.0          1.4
2              3.2          1.3
3              3.1          1.5
4              3.6          1.4
...
[150 rows x 2 columns]
```

1.1 Istraživanje Veba - Web Mining

Kada posmatramo distribuirani sistem informacija, vidimo da su dokumenti i objekti najčešće povezani kako bi se omogućio interaktivni pristup. Glavni primer distribuiranog sistema je Veb, gde korisnici, u potrazi za informacijama,

”putuju” od objekta do objekta uz pomoć hiperlinkova i *URL* adresa. Veb konstantno raste i svaki dan se dodaje po nekoliko miliona stranica. Sa ovakvim, neprekidnim, rastom dostupnih stranica i podataka uviđa se da dobijanje informacija iz takvih izvora postaje sve teže i teže. Kao jedan od glavnih problema pri analizi Veb stranica javlja se nedostatak strukture. Pod ”rudarenjem (istraživanjem) Veba” (engl. *web mining*) podrazumevamo korišćenje tehnika za ”rudarenje (istraživanje) podataka” (engl. *data mining*) kako bi se automatski otkrile i izvlačile informacije sa Veb dokumenata i servisa. Web mining možemo podeliti na četiri glavne celine:

1. pronalaženje resursa - prikupljanje informacija preko na primer Web stranica kao što su stranice sa vestima i slično,
2. odabir informacija i pretprocesiranje - različite vrste transformacija nad podacima: uklanjanje stop reči, dobijanje željene reprezentacije i drugo,
3. generalizacija - proces automatskog otkrivanja uobičajenih uzoraka (engl. *pattern*) korišćenjem različitih opštih tehnika mašinskog učenja, istraživanja podataka i raznih veb-orijentisanih metoda i
4. analiza - vrši se validacija i/ili interpretacija istraženih uzoraka (engl. *mined patterns*) .

Jedna od mogućih podela istraživanja Veba odnosi se na to koji deo Veba neko želi da analizira. Postoje tri vida istraživanja Veba:

1. Istraživanje sadržaja Veba (engl. *Web content mining*) - koristi sadržaj Veb stranice kako bi prikupio podatke: tekst, slike, video ili bilo koji drugi sadržaj,
2. Istraživanje strukture Veba (engl. *Web structure mining*) - fokusira se na strukturu veza veb stranice,
3. Istraživanje upotrebe Weba (engl. *Web usage mining*) - kao podatke koristi podatke prikupljene od interakcija korisnika pri korišćenju interneta.

Najčešća upotreba istraživanja sadržaja Veba je u procesu pretrage. Na primer, crawler-i se koriste od strane pretraživača da izvuku sadržaj Veb strane kako bi se odmah dobili traženi rezultati. Isto tako, oni se mogu koristiti tako da im fokus bude na određenoj temi ili oblasti interesovanja, umesto da zahtevaju sve informacije koje se mogu dostići.

Kako bi se kreirao fokusirani crawler, klasifikator se obično trenira na skupu podataka odabranih od strane korisnika, kako bi crawler znao koji tip sadržaja traži. Potom on identifikuje stranice od interesa kako na njih nalazi i prati dalje sve linkove sa te stranice. Ako linkovi sa neke stranice od interesa vode ka nekim drugim stranicama koje su klasifikovane kao stranice koje nisu od interesa onda se linkovi na tim stranama dalje ne posmatraju. Istraživanje sadržaja se može najlakše direktno videti u procesu pretrage. Svi veći pretraživači danas koriste strukturu koja podseća na listu. Ta lista je uređena uz pomoć algoritma za rangiranje stranica.

U *Pythonu* postoji modul *sklearn.feature_extraction* koji služi za izvlačenje objekata (engl. *features*) iz skupova podataka koji sadrže tekst ili slike, u formatu koji je podržan od strane algoritama mašinskog učenja.

1.1.1 Izvlačenje teksta

Analiza teksta je jedna od važnih primena algoritama mašinskog učenja. Međutim, sirovi podaci (engl. *raw data*) predstavljaju prepreku utoliko što ih algoritmima ne možemo dati direktno, već nam je potrebna neka njihova numerička reprezentacija uz pomoć vektora fiksirane dužine. Kako bi se ovaj problem rešio *scikit – learn* nudi nekoliko načina da se iz tekstualnog sadržaja izvuku numerički podaci, princip koji se koristi je sledeći:

- dodeljivanje tokena stringova i davanje id-ja za svaki od tokena, npr. korišćenjem razmaka ili zareza kao odvajajućih elemenata između dva tokena,
- brojanje pojavljivanja tokena u svakom od dokumenata,
- normalizacija i dodeljivanje težina sa smanjenjem bitnosti onih tokena koji se često pojavljuju

Korpus dokumenata se, dakle, može predstaviti uz pomoć matrice koja ima jedan red po dokumentu i jednu kolonu po tokenu (reči) koje se javljaju u korpusu. **Vektorizacijom** nazivamo proces pretvaranja kolekcije tekst dokumenata u numeričke vektore objekata (engl. *numerical feature vectors*). Konkretna strategija navedena gore (tokenizacija, brojanje i normalizacija) naziva se *Bag of Words* reprezentacija. Dokumenti su opisani pojavljivanjima reči ignorišući informacije o poziciji reči u dokumentima.

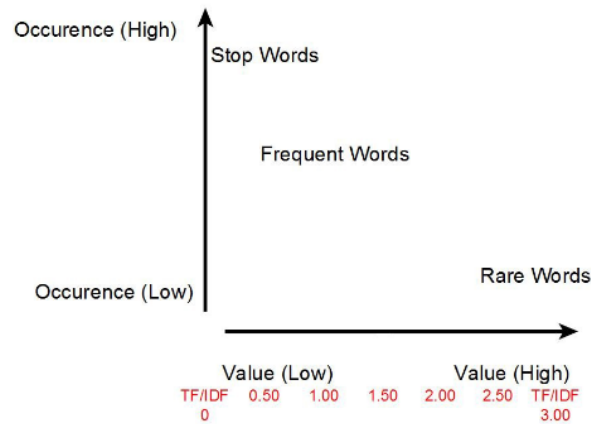
CountVectorizer iz *sklearn.feature_extraction.text* implementira i tokenizaciju i brojanje pojavljivanja u istoj klasi. Korišćenje "stop" reči, poput "and", "the", itd., za koje se pretpostavlja da ne nose nikakve informacije u kontekstu teksta, mogu biti uklonjene. Nekada, međutim, može se desiti da su slične reči korisne za predikciju, tako da sa korišćenjem "stop" reči treba biti oprezan. Jedna od najpoznatijih "stop" lista reči je "english". U velikim korpusima teksta, neke reči će se ponavljati veoma često, samim tim takve reči ni ne nose puno značajnih informacija o stvarnom sadržaju dokumenta. Ako bismo dali podatke o broju reči klasifikatoru direktno, one reči koje se veoma često ponavljaju bi poremetile pojavljivanja onih reči koje se rede ponavljaju, a samim tim su nam i zanimljivije. Kako bismo ponovo izmerili i dodelili težine objektima, u vrednostima u pokretnom zarezu koji je pogodan za klasifikator, veoma često koristićemo *tf – idf* transformaciju.

Tf označava učestalost pojavljivanja nekog termina (engl. *term-frequency*), dok *tf – idf* se odnosi na učestalost pojavljivanja nekog termina puta inverzna učestalost dokumenta (engl. *inverse document frequency*).

$$tfidf(t, d) = tf(t, d) \times idf(t) \quad (1)$$

Korišćenjem osnovnih podešavanja *TfidfTransformer*-a : `TfidfTransformer(norm='l2', use_idf=True, smooth_idf=True, sublinear_tf=False)`, učestalost termina, broj puta koji se on pojavljuje u dokumentu, množi se sa *idf* komponentom, koja se računa kao:

$$idf(t) = \log\left(\frac{1 + n_d}{1 + df(d, t)}\right) + 1 \quad (2)$$



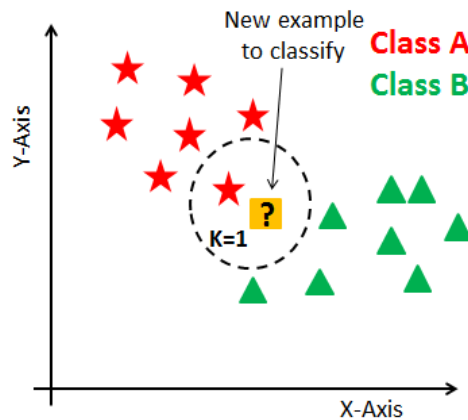
Slika 1: Prikaz odnosa učestalosti pojavljivanja reči i njihovog značaja.

gde je n_d ukupni broj dokumenata, $df(d, t)$ je broj dokumenata koji sadrže termin t . Rezultujući tfidf vektori se potom normalizuju Euklidskom normom. Glavni cilj korišćenja tf-idf umesto sirovih učestalosti pojavljivanja tokena u dokumentu je skaliranje uticaja tokena koji se jako često pojavljuju u korpusu i time su empirijski manje informativni.

Da bismo matricu učestalosti pojavljivanja (engl. *count matrix*) transformisali u normalizovanu *tf* ili *tfidf* reprezentaciju koristimo

`sklearn.feature_extraction.text.TfidfTransformer` Kako ne bismo svaki put radili `CountVectorizer`, pa potom `TfidfTransformer`, postoji ugrađena biblioteka podrška, u vidu:

`sklearn.feature_extraction.text.TfidfVectorizer` koji konvertuje kolekciju sirovih dokumenata u matricu *TF-IDF* objekata.



Slika 2: Prikaz K-najbližih suseda.

Primer 1.2 Pošto nam je cilj da se bavimo obradom prirodnih jezika, za tako nešto možemo iskoristiti *nlTK* (engl. The Natural Language Toolkit). On predstavlja skup biblioteka i programa za simboličko i statističko procesiranje prirodnih jezika, konkretno engleskog jezika. Predstavlja vodeću platformu za izradu *Python* programa za rad sa prirodnim jezikom, pa tako sadrži skup biblioteka za procesiranje teksta pri klasifikaciji, tokenizaciju, izvlačenje korena reči (engl. *stemming*), tagovanje, parsiranje itd. *Stemeri* (engl. *stemmers*) uklanjaju morfološke prefikse, sufikse i infikse iz reči, kako bi ostala samo srž. Postoji više različitih vrsta stemera, od kojih će prvi biti predstavljen *SnowballStemmer*, koji podržava naredne jezike: danski, holandski, engleski, finski, francuski, nemački, mađarski, italijanski, norverški, portugalski, rumunski, ruski, španski i švedski.

Kada hoćemo da vršimo klasifikaciju, to možemo učiniti pomoću više različitih klasifikatora koje nam nudi *sklearn*, kao što su:

- *KNeighborsClassifier* - ima višestruke primene u finansijama, zdravstvu, političkim naukama, detekciji rukopisa, prepoznavanju slika i videa, itd.. Koristi se i za klasifikaciju i za regresiju i bazira se na pristupu sličnosti objekata. K predstavlja broj najbližih suseda. Ako je $K = 1$, onda za algoritam kažemo da je samo algoritam najbližeg suseda (engl. *nearest neighbor algorithm*).
- *SGDClassifier* - *Stohastički gradijentni spust* (engl. *Stochastic Gradient Descent-SGD*) je jednostavan, ali veoma efikasan pristup, čije se prednosti ogledaju u efikasnosti, jednostavnoj implementaciji, a mane su mu što zahteva određeni broj parametara (npr. regularizacioni parametar, broj iteracija) i osetljiv je na skaliranje objekata.
- *MultinomialNB* - Naivni Bajesof klasifikator (engl. *Naive Bayes classifier for multinomial models*) je pogodan za klasifikaciju diskretnih objekata.

Primer 1.3 *Naredni primer vrši klasifikaciju novinskih članaka. Najpre se učitavaju podaci iz fajla `articles.csv`, u kome su podaci smešteni u obliku kategorija, tekst. Kolone koje čine fajl smeštamo u dve promenljive `redom`, `y` i `texts`. Nakon toga, vrši se inicijalizacija vektorizatora za kreiranje TF-IDF matrice.*

```
import pandas as pd
import nltk
import re
from nltk.stem.snowball import SnowballStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.naive_bayes import MultinomialNB

# Preprocesiranje teksta
def preprocess(raw_text):
    # Izdvajanje recenica
    sentences = nltk.sent_tokenize(raw_text)

    words = []

    # Inicijalizacija stemmer-a za engleski jezik
    stemmer = SnowballStemmer('english')

    for sentence in sentences:
        # Izdvajanje reci (tokena) iz recenica
        raw_words = nltk.word_tokenize(sentence)
        for raw_word in raw_words:
            # Rec smatramo validnom ako sadrzi samo karaktere i '
            if re.search('[a-zA-Z\']+$', raw_word):
                # Stem-ovanje reci
                word = stemmer.stem(raw_word)
                words.append(word)

    return words

def main():
    # Ucitavanje CSV fajla sa novinskim clancima
    # kategorija, tekst
    df = pd.read_csv('articles.csv')

    texts = []
    y = []

    # Izdvajanje kolone sa tekstom i kolone sa kategorijom
    for row in df.values:
        texts.append(row[1])
        y.append(row[0])
```



```

# Inicijalizacija vektorizatora za kreiranje TF-IDF matrice
tfidf_vectorizer = TfidfVectorizer(stop_words='english', tokenizer=preprocess, max

# Vektorizacija tekstova iz Bag of Words reprezentacije u matricnu TF-IDF reprezen
tfidf_matrix = tfidf_vectorizer.fit_transform(texts)

# Deljenje podataka na trening i test skup 70%-30%
X_train, X_test, y_train, y_test = train_test_split(tfidf_matrix, y, test_size=0.3

# clf = KNeighborsClassifier()
# clf = SGDClassifier()

# Klasifikacija koriscenjem Naivnog Bajesa
clf = MultinomialNB()

# Fitovanje klasifikacionog modela
clf.fit(X_train, y_train)

# Ukoliko je model zadovoljavajuci,
# finalna verzija modela se dobija fitovanjem
# cele matrice i cele y kolone
# clf.fit(tfidf_matrix, y)

# Racunanje preciznosti modela na trening i test podacima
print('Train acc: {}'.format(clf.score(X_train, y_train)))
print('Test acc: {}'.format(clf.score(X_test, y_test)))

# Ucitavanje tekstualnog fajla za test modela
test_file = open('test.txt')
test_text = test_file.read()

# Transformacija test teksta u TF-IDF
test_text_transformed = tfidf_vectorizer.transform([test_text])

# Predikcija kategorije teksta
print(clf.predict(test_text_transformed))

# Ispis kategorija u sortiranom redosledu
print(sorted(set(y)))

# Ispis vrednosti verovatnoca NB klasifikatora
# vezanih za pripadnosti test teksta nekoj od kategorija
print(clf.predict_proba(test_text_transformed))

if __name__ == "__main__":
    main()

```

Rezultat izvršavanja ovog programa je oblika:

```
Train acc: 0.9871547848426461
Test acc: 0.968562874251497
[ 'business ' ]
[ 'business ', 'entertainment ', 'politics ', 'sport ', 'tech ' ]
[[0.38475301 0.09423106 0.28803524 0.11219421 0.12078647]]
```

1.1.2 Latentna semantička analiza

LSA predstavlja način za particionisanje teksta korišćenjem statističkih modela upotrebe reči koji je sličan dekompoziciji sopstvenih vektora i analizi. Umesto da se samo fokusiramo na površne objekte kao što je učestalost reči, ovaj pristup obezbeđuje kvantitativnu meru semantičkih sličnosti među dokumentima baziranom na kontekstu reči. Dve glavne mane su sinonimi i polisemija. Sinonimija se odnosi na različite reči istog ili sličnog značenja. Sa druge strane, polisemija se odnosi na reči koje imaju više različitih značenja. *LSA* pokušava da razreši ovaj tip problema, bez pomoći rečnika i sredstava na obradu prirodnih jezika, već korišćenjem matematičkih uzoraka koji postoje unutar podataka. To se postiže smanjenjem dimenzije koja se koristi da se dokument predstavi korišćenjem matematičke matrice operacije koja se naziva singularna dekompozicija (engl. SVD - Singular Value Decomposition).

SVD dekompozicija razbija bilo koju matricu A na $A = U * S * V'$. Ako bismo bliže pogledali matricu S , videli bismo da je S matrica forme, takve, da se sastoji od matrice D , koja na dijagonali sadrži sve σ , koje predstavljaju singularne vrednosti. Broj ovih singularnih vrednosti nam govori o rangju matrice A . Možemo pronaći aproksimaciju redukovano rangja (engl. *truncated SVD* ili *LSA*) za A tako što postavimo sve, osim prvih k najvećih singularnih vrednosti na nulu, a potom koristimo samo prvih k kolona matrice U i V . Ovaj postupak vršimo kako bismo izvršili redukciju dimenzionalnosti. Suprotno PCA, ovaj estimator ne centrira podatke pre izračunavanja singularne dekompozicije, što mu omogućava da radi efikasno sa retkim matricama (dostupnim kroz modul *scipy.sparse*). Konkretno, truncated SVD radi nad matricama sa brojevima termova (engl. *term count*) ili nad *tfidf* matricama. U kontekstu ovih drugih, postupak je poznat kao latentna semantička analiza (engl. latent semantic analysis-LSA).

Primer 1.4 Ovaj primer vrši klasifikaciju novinskih članaka. Najpre se učitavaju podaci iz fajla *articles.csv*, u kome su podaci smešteni u obliku kategorija, tekst. Kolone koje čine fajl smeštamo u dve promenljive redom, y i $texts$. Nakon toga, vrši se inicijalizacija vektorizatora za kreiranje *TF-IDF* matrice. Potom se vrši *LSA*.

```
import pandas as pd
import nltk
import re
from nltk.stem.snowball import SnowballStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
```

```

from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.decomposition import TruncatedSVD

# Preprocesiranje teksta
def preprocess(raw_text):
    # Izdvajanje recenica
    sentences = nltk.sent_tokenize(raw_text)

    words = []

    # Inicijalizacija stemmer-a za engleski jezik
    stemmer = SnowballStemmer('english')

    for sentence in sentences:
        # Izdvajanje reci (tokena) iz recenica
        raw_words = nltk.word_tokenize(sentence)
        for raw_word in raw_words:
            # Rec smatramo validnom ako sadrzi samo karaktere i '
            if re.search('[a-zA-Z\']+$', raw_word):
                # Stem-ovanje reci
                word = stemmer.stem(raw_word)
                words.append(word)

    return words

def main():
    # Ucitavanje CSV fajla sa novinskim clancima
    # kategorija, tekst
    df = pd.read_csv('articles.csv')

    texts = []
    y = []

    # Izdvajanje kolone sa tekстом i kolone sa kategorijom
    for row in df.values:
        texts.append(row[1])
        y.append(row[0])

    # Inicijalizacija vektorizatora za kreiranje TF-IDF matrice
    tfidf_vectorizer = TfidfVectorizer(stop_words='english', tokenizer=preprocess, max

    # Vektorizacija tekstova iz Bag of Words reprezentacije u matricnu TF-IDF reprezen
    tfidf_matrix = tfidf_vectorizer.fit_transform(texts)

    # SVD (LSA)
    lsa = TruncatedSVD(n_components=200, n_iter=100)

```

```

# Fitovanje LSA pomocu TF-IDF matrice
lsa.fit(tfidf_matrix)

# Izdvajanje naziva kolona TF-IDF matrice
terms = tfidf_vectorizer.get_feature_names()

# Ispis kolona (reci) sa najvećim tezinama po komponentama
for (i, comp) in enumerate(lsa.components_):
    terms_in_comp = zip(terms, comp)
    sorted_comps = sorted(terms_in_comp, key=lambda x: x[1], reverse=True)[:10]

    print("Component {}".format(i))
    print()
    for term in sorted_comps:
        print(term)
    print()

# Transformisanje TF-IDF matrice koriscenjem LSA
transformed = lsa.transform(tfidf_matrix)

if __name__ == "__main__":
    main()

```

Rezultat izvršavanja ovog programa predstavlja skup reči sa najvećim težinama po komponentama oblika:

Component 182

```

('scottish ', 0.09168672123398844)
('play ', 0.08953887985792805)
('age ', 0.08291233924728324)
('ethnic ', 0.08288040224246893)
('murder ', 0.08153061624591648)
('edward ', 0.07253791078410225)
('develop ', 0.07211813150850585)
('foster ', 0.07116597034292958)
('minor ', 0.07032073815220793)
('sentenc ', 0.06973781343683193)

```

Primer 1.5 Da bismo prikupili neke informacije sa veb stranica, a želimo to da uradimo na jednostavan način, potreban nam je program koji bi automatski izdvojio informacije od interesa i izvršio njihovo izdvajanje za nas. Ovakav program naziva se *skrejper* (engl. *scraper*). On obilazi određenu vezu (ili, još češće, više veza na stranicama) i izdvaja željene informacije. Takav jedan program bio bi nam, na primer, potreban kako bismo popunili neki *.csv* fajl u koji bismo mogli da smestimo neke novinske članke. Da bismo tako nešto postigli, moramo za svaku *.html* stranicu koju posetimo: da izdvojimo relevantne informacije i da nađemo linkove ka ostalim. Kako bismo to uradili moramo koristiti regularne

izraze i *re* biblioteku koja nam omogućava rad sa njima. Da bismo uopšte mogli da "dovučemo" stranice moramo koristiti *urllib.request*.

Urllib je paket koji sadrži nekoliko modula za rad sa URL-ovima:

- *urllib.request* - za otvaranje i čitanje URL-ova,
- *urllib.error* - sadrži izuzetke koji se mogu javiti korišćenjem *urllib.request*,
- *urllib.parse* - za parsiranje URL-ova i
- *urllib.robotparser* - za parsiranje *robots.txt* fajlova.

Osim toga, u slučaju, da je neki sajt na ćirilici, a

```
import re
import urllib.request
from transliterate import translit
from collections import deque

# Čiscenje sadržaja HTML taga od ostalih HTML tagova koji
# se nalaze unutar sadržaja kao i specijalnih HTML karaktera
# oblika &tekst;
def clean(raw_html):
    tags = re.compile('<.*?>|\\&[a-z]+;')
    return tags.sub('', raw_html)

# Prevodjenje teksta iz ćirilice u latinicu.
# Ako je tekst već ćirilicni, tekst će ostati neizmenjen
def transliterate(text):
    return translit(text, 'sr', reversed=True)

# Preuzimanje sadržaja HTML stranice
def get_webpage(URL):
    try:
        # Pripremanje zahteva za traženim URL
        req = urllib.request.Request(URL)
        # Otvaranje konekcije ka stranici
        response = urllib.request.urlopen(req)
        # Čitanje i dekodiranje sadržaja stranice
        data = response.read().decode('utf-8')
        # Transliteracija
        data = transliterate(data)

        # Brisanje karaktera prelaska u novi red,
        # zareza, tabulatora i apostrofa
        data = data.replace('\\n', '')
        data = data.replace('\\t', '')
        data = data.replace('\\r', '')
        data = data.replace(',', '')
        data = data.replace('\\', '')

        # Uklanjanje suvisnih razmaka '___' -> '_'
```

```

        space_regex = re.compile(r'[ ]+')
        data = space_regex.sub(' ', data)
    except:
        data = ''

    return data

# Izdvajanja sadržaja zadatih HTML elemenata
def extract_elements(element, raw_html):
    regex = '<' + element + '(.*)>(<.*>)/'+element+'>'
    raw_elements = re.findall(regex, raw_html)

    for i in range(len(raw_elements)):
        raw_elements[i] = (raw_elements[i][0], clean(raw_elements[i][1]))

    return raw_elements

# Izdvajanje sadržaja zadatih atributa iz HTML elemenata
def extract_attributes(attribute, attribute_string):
    regex = attribute + '="(.*)"'
    attributes = re.findall(regex, attribute_string)

    return attributes

# Izdvajanja sadržaja tekstualnih elemenata sa stranice zadate URL-om
def extract_texts(URL):
    texts = []
    page_html = get_webpage(URL)
    texts += [el[1] for el in extract_elements('h1', page_html)]
    texts += [el[1] for el in extract_elements('h2', page_html)]
    texts += [el[1] for el in extract_elements('h3', page_html)]
    texts += [el[1] for el in extract_elements('h4', page_html)]
    texts += [el[1] for el in extract_elements('h5', page_html)]
    texts += [el[1] for el in extract_elements('h6', page_html)]
    texts += [el[1] for el in extract_elements('p', page_html)]

    return " ".join(texts)

# Izdvajanje vrednosti href atributa svih linkova na stranici zadatoj URL-om
def extract_links(URL):
    raw_links = []
    page_html = get_webpage(URL)

    raw_links = extract_elements('a', page_html)

    links = []

    for link in raw_links:
        hrefs = extract_attributes('href', link[0])

```

```

        if len(hrefs) > 0:
            url = hrefs[0]
            # Filtriranje unutrasnjih linkova (#...) i apsolutnih adresa
            if len(url) > 0 and url[0] == '/' and not re.search('#', url):
                links.append(url)

    return links

# Nalazenje liste susednih strana za zadatu stranu
# base - domen
# path - putanja
def get_neighbors(base, path):
    links = extract_links(base + path)
    return links

# BFS obilazak stranica od zadate pocetne adrese
def bfs(start_page_URL):
    queue = deque(['/'])
    base = start_page_URL

    marked = {}
    marked['/'] = True

    texts = []

    limit = 200
    current_count = 0

    while len(queue) > 0:
        curr = queue.popleft()

        print(curr)

        # Ako je tekst clanak, izdvaja se njegov sadrzaj
        if re.search('clanak', curr):
            print('clanak:')
            texts.append((base + curr, extract_texts(base + curr)))
            current_count += 1
            if current_count == limit:
                return texts

        # U suprotnom, nastavlja se obilazak od trenutne stranice
        else:
            for neighbor in get_neighbors(base, curr):
                if neighbor not in marked:
                    marked[neighbor] = True
                    queue.append(neighbor)

    return texts

```

```

def main():
    article_texts = bfs('http://www.politika.rs')

    # Priprema CSV formata
    header = 'url, text\n'

    content = ''

    # Formatiranje podataka za CSV zapis
    for text in article_texts:
        content += text[0] + ',' + text[1] + '\n'

    # Sastavljanje sadržaja CSV fajla
    articlesCSV = header + content

    # Zapisivanje CSV fajla
    output = open('scraped.csv', 'w')
    output.write(articlesCSV)

if __name__ == "__main__":
    main()

/rss/
/scc/pretraga
/scc/rubrika/1/Svet
/scc/rubrika/2/Politika
/scc/rubrika/3/Drustvo
/scc/rubrika/4/Pogledi
/scc/rubrika/5/Hronika
/scc/rubrika/6/Ekonomija
/scc/rubrika/7/Kultura
/scc/rubrika/9/Srbija
/scc/rubrika/10/Beograd
/scc/rubrika/8/Sport
/scc/rubrika/29/Region
/scc/sarena-strana
/scc/rubrika/396/Magazin
/scc/rubrika/34/Moj-zivot-u-inostranstvu
/scc/satires/index
/scc/rubrika/1060/TV-revija
/scc/rubrika/1073/Tema-nedelje
/scc/clanak/414061/Slucaj-revizora-Sretenovica
clanak:
/scc/autor/913/Jovana-Rabrenovic
/scc/clanak/414064/Vlada-Crne-Gore-Srusicemo-crkvu-na-Rumiji
clanak:
/scc/clanak/414066/SAD-nisane-Rusiju-a-gadaju-Kinu
clanak:
/scc/autor/854/Jelena-Stevanovic
/scc/clanak/414096/Putin-Nas-odgovor-na-americke-rakete-u-Evropi-bice-brz-i-efi

```


clanak :
 /scc/clanak/414059/U-rukama-ucenika-umesto-olovaka-cigarete
 clanak :
 /scc/clanak/414086/Sport/Ubedljiva-pobeda-Liverpula-protiv-Zvezde
 clanak :
 /scc/rubrika/49/Fudbal
 /scc/clanak/414069/Oluja-krivac-za-rakete-ispaljene-iz-Gaze-na-Izrael
 clanak :
 /scc/clanak/414062/Zima-ce-bit-i-blaga-sa-malo-padavina
 clanak :
 /scc/clanak/414057/Humorom-protiv-straha
 clanak :
 /scc/clanak/414071/Pojacana-prodaja-novih-automobila
 clanak :
 /scc/clanak/414097/Sport/Kosarka/Dabl-dabl-Bjelice-u-pobedi-Kingsa-nad-Memfisom
 clanak :
 /scc/rubrika/50/Kosarka
 /scc/clanak/414093/Americki-avion-predvodio-dronove-u-napadu-na-rusku-bazu-u-S

Primer 1.6 Naredni primer je tzv. *PageRank* algoritam. *PageRank* je algoritam originalno objavljen od strane ljudi koji su učestvovali u kreiranju Google-a, Sergeja Brina i Lari Pejđža. *PageRank* se smatra odgovornim za rani uspeh Google-a, funkcioniše tako što obezbeđuje globalno rangiranje čvorova u grafu. Za pretraživače obezbeđuje rangiranje stranica nezavisno od upita. Glavna pretpostavka ovog algoritma je da je svaka veza od stranice a ka stranici b glas od stranice a za stranicu b . Bitno je naglasiti da nije svaki glas jednake težine. Težine dodeljuje *PageRank* algoritam na osnovu početnog čvora.

2 Eksperimenti

3 Zaključak