

# Skripta za vežbe

Skripta u okviru kursa  
Istraživanje podataka 2  
Matematički fakultet

24. sept 2018.

## Sadržaj

<b>1</b>	<b>Uvod</b>	<b>2</b>
<b>2</b>	<b>Istraživanje Veba - Web Mining</b>	<b>4</b>
2.1	Izvlačenje teksta . . . . .	5
2.2	Predikcija kategorije teksta . . . . .	8
2.3	Latentna semantička analiza . . . . .	10
2.4	Web Scraping . . . . .	12
2.5	PageRank algoritam . . . . .	19
<b>3</b>	<b>Furijeova transformacija</b>	<b>25</b>
	<b>Literatura</b>	<b>25</b>

# 1 Uvod

S obzirom na činjenicu da se ovaj kurs direktno nastavlja na kurs Istraživanje podataka 1, kao i da postoji uslovljenost nekim predmetima, podrazumeva se poznavanje osnovnih koncepata. Iako će neki od njih biti ponovno objašnjeni u daljem tekstu, fokus ovog teksta je na novim konceptima.

**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 ispis različitih vrednosti. Kao što vidimo **pandas** biblioteka nam kroz **DataFrame** nudi mnogobrojne načine pregledanja podataka, koje možemo koristiti na različite načine. U nastavku su navedeni osnovni.*

```
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 prvih 5 redova
    print(df.head())
    # 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	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

---

```

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]		

## 2 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"<sup>1</sup> podrazumevamo korišćenje tehnika za "rudarenje (istraživanje) podataka"<sup>2</sup> kako bi se automatski otkrile i izvlačile informacije sa Veb dokumenta 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*<sup>3</sup>.

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

---

<sup>1</sup>engl. web mining

<sup>2</sup>engl. data mining

<sup>3</sup>engl. mined patterns

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<sup>4</sup> iz skupova podataka koji sadrže tekst ili slike, u formatu koji je podržan od strane algoritama mašinskog učenja.

## 2.1 Izvlačenje teksta

Analiza teksta je jedna od važnih primena algoritama mašinskog učenja. Međutim, sirovi podaci<sup>5</sup> 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<sup>6</sup>. 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.

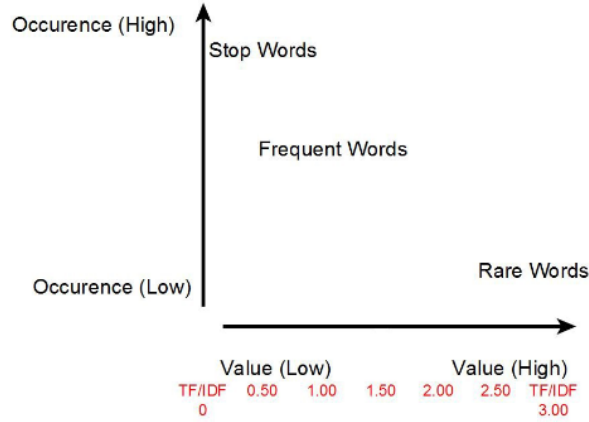
`CountVectorizer` iz `sklearn.feature_extraction.text` implementira i tokenizaciju i brojanje pojavljivanja u istoj klasi. Korišćenje "stop" reči, poput "and", "the", itd., za koje se pretpostavlja da ne nose nikakve informacije u kontekstu teksta, mogu biti uklonjene. Nekada, međutim, može se desiti 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 ređe 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.

---

<sup>4</sup>engl. features

<sup>5</sup>engl. raw data

<sup>6</sup>engl. numerical feature vectors



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

$Tf$  označava učestalost pojavljivanja nekog termina<sup>7</sup>, dok  $tf-idf$  se odnosi na učestalost pojavljivanja nekog termina puta inverzna učestalost dokumenta<sup>8</sup>.

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

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:

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

gde je  $n_d$  ukupni broj dokumenata,  $df(d, t)$  je broj dokumenata koji sadrže termin  $t$ . Rezultujući  $tf-idf$  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<sup>9</sup> transformisali u normalizovanu  $tf$  ili  $tf-idf$  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.

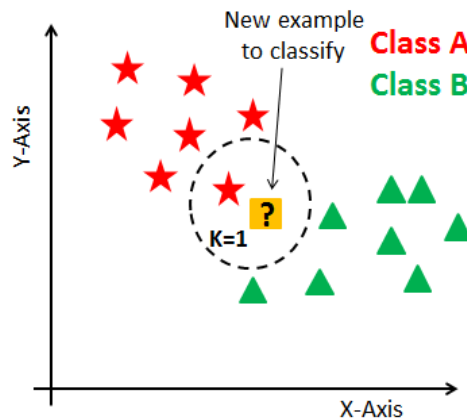
Pošto nam je cilj da se bavimo obradom prirodnih jezika, za tako nešto možemo iskoristiti `nlTK`<sup>10</sup>. On predstavlja skup biblioteka i programa za simboličko i statističko procesiranje prirodnih jezika, konkretno engleskog jezika.

<sup>7</sup>engl. term-frequency

<sup>8</sup>engl. inverse document frequency

<sup>9</sup>engl. count matrix

<sup>10</sup>engl. The Natural Language Toolkit



Slika 2: Prikaz K-najbližih suseda.

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<sup>11</sup>, tagovanje, parsiranje itd. Stemeri<sup>12</sup> 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.

<sup>11</sup>engl. stemming

<sup>12</sup>engl. Stemmers

## 2.2 Predikcija kategorije teksta

**Primer 2.1** *Naredni primer vrši predikciju kategorije teksta. 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, primenjuje se neki od 3 klasifikatora:*

- *`KNeighborsClassifier`*
- *`SGDClassifier`*
- *`MultinomialClassifier`*

*Nakon fitovanja klasifikacionog modela, računaru se preciznosti na trening i test skupu, a potom se učitava test model i vrši se predikcija kategorije teksta.*

```
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 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)
# 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()

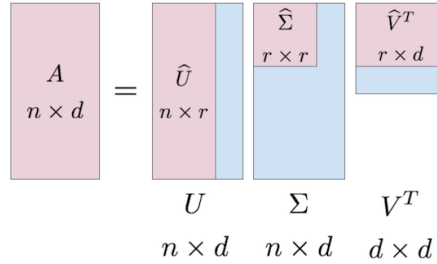
```

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]]

```



Slika 3: Prikaz dekompozicije matrice.

## 2.3 Latentna semantička analiza

**Latentna semantička analiza**<sup>13</sup> 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 sinonimija 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 matrične operacije koja se naziva singularna dekompozicija<sup>14</sup>.

**Singularna 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  $\sigma$ , koje predstavljaju singularne vrednosti. Broj ovih singularnih vrednosti nam govori o rangui matrice  $A$ . Možemo pronaći aproksimaciju redukovano rang<sup>15</sup> 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<sup>16</sup>, ovaj estimator ne centrira podatke pre izračunavanja singularne dekompozicije, što mu omogućava da radi efikasno sa retkim matricama<sup>17</sup> (dostupnim kroz modul `scipy.sparse`). Konkretno, truncated SVD radi nad matricama sa brojevima termova<sup>18</sup> ili nad *tfidf* matricama. U kontekstu ovih drugih, postupak je poznat kao latentna semantička analiza.

**Primer 2.2** *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,*

<sup>13</sup>engl. LSA - latent semantic analysis-LSA

<sup>14</sup>engl. SVD - Singular Value Decomposition

<sup>15</sup>engl. *truncated SVD* ili LSA

<sup>16</sup>engl. PCA - Principal Component Analysis

<sup>17</sup>Retke matrice su one matrice koje sadrže veliki broj nula.

<sup>18</sup>engl. term count

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 (reči) 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 tezinama 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)

```

## 2.4 Web Scraping

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 to izdvajanje za nas. Ovakav program naziva se skrepper<sup>19</sup>. 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:

1. da izdvojimo relevantne informacije i
2. da nađemo linkove ka ostalim stranama.

Kako bismo to uradili moramo koristiti regularne izraze i *re* biblioteku koja nam omogućava rad sa njima. Da bismo uopšte mogli da "dovučemo" stranice

---

<sup>19</sup>engl. scraper

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.

U slučaju da je neki sajt na ćirilici, ne želimo da ga menjamo, ali u slučaju da je na latinici, želimo da, uz pomoć `translit` modula `transliterate`, prevedemo tekst iz latinice u ćirilicu. `Transliterate` je dvosmerni modul za prevođenje u Pythonu. Prevodi niske prema pravilima koja su specifična u svakom jezičnom paketu (izvorni skript <-> ciljni skript). Nudi mogućnost rada sa narednim jezicima: jermenski, bugarski, gruzijski, grčki, makedonski, mongolski, ruski, srpski, ukrajinski.

**Primer 2.3** *Naredni primer upravo vrši opisani postupak.*

```
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 cirilice u latinicu.
# Ako je tekst vec cirilicni, tekst ce 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 unutrašnjih 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 sadrzaja 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-e
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-biti-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-Memfisa
clanak:
/scc/rubrika/50/Kosarka
/scc/clanak/414093/Americki-avion-predvodio-dronove-u-napadu-na-rusku-bazu-u-

```



**Primer 2.4** Nakon što smo prikupili neke podatke sa stranica, želimo sada i na neki način da te podatke analiziramo. U ovom primeru želimo da vršimo pre-tragu stranica na osnovu traženih pojmova, koje unosimo u vidu *query\_string*.

```
import pandas as pd
import numpy as np
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
import stemmer
from sklearn.cluster import KMeans

# Stop reci na srpskom jeziku
stopwords_file = open('stopwords-sr.txt')
stopwords = stopwords_file.read().split('\n')

# Preprocesiranje teksta
def preprocess(raw_text):
    # Stemmer za srpski jezik
    raw_words = stemmer.stem_arr(raw_text)
    words = []
    for raw_word in raw_words:
        # Rec smatramo validnom ako sadrzi samo karaktere i '
        if re.search('[a-zA-ZćžčšđĆŽČŠĐ\']+$', raw_word):
            # Stem-ovanje reci
            words.append(raw_word)
    return words

def main():
    # Ucitavanje CSV fajla sa novinskim clancima
    # kategorija, tekst
    df = pd.read_csv('scraped.csv')
    texts = []
    urls = []
    # Izdvajanje kolone sa tekстом i kolone sa kategorijom
    for row in df.values:
        texts.append(row[1])
        urls.append(row[0])
    # Inicijalizacija vektorizatora za kreiranje TF-IDF matrice
    tfidf_vectorizer = TfidfVectorizer(tokenizer=preprocess, max_df=0.9, ngram_range=(
    # Vektorizacija tekstova iz Bag of Words reprezentacije u matricnu TF-IDF represen
    tfidf_matrix = tfidf_vectorizer.fit_transform(texts)
    # Klasterovanje clanaka K-Means algoritmom
```

```

# n = 11
# km = KMeans(n_clusters=n, n_init=300)
# km.fit(tfidf_matrix)
# clusters = {}
# for i in range(len(urls)):
#     cluster_num = km.labels_[i]
#     if cluster_num not in clusters:
#         clusters[cluster_num] = []
#     clusters[cluster_num].append(urls[i])
# for i in clusters:
#     print('Cluster {}'.format(i))
#     print()
#     for url in clusters[i]:
#         print(url)
#     print()
# Pretraga stranica na osnovu traženih pojmova
query_string = "vesti iz regiona"
query_transformed = tfidf_vectorizer.transform([query_string])
query_column = np.transpose(query_transformed)
product = tfidf_matrix.dot(query_column)
weights = [0 for i in range(len(urls))]
for i in range(len(urls)):
    weights[i] = product[i, 0]
zipped_product = zip(urls, weights)
# Vracaju se prvih 5 najbitnijih stranica za zadate pojmove
sorted_product = sorted(zipped_product, key=lambda x: x[1], reverse=True)
for (url, w) in sorted_product:
    print(url)

if __name__ == "__main__":
    main()

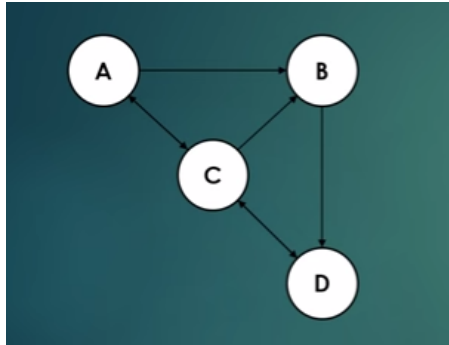
```

Izlaz iz programa predstavlja prvih 5 najbitnijih stranica za zadate pojmove.

```

http://www.politika.rs/scc/clanak/414025/Spanske-firme-sele-centrale-iz-regiona
http://www.politika.rs/scc/clanak/413970/Linta-Da-srpske-zrtve-u-Hrvatskoj-i-
http://www.politika.rs/scc/clanak/413321/Skot-Kosovo-je-suverena-zemlja
http://www.politika.rs/scc/clanak/413988/Bramerc-o-predstojecem-izvestaju-Mla
http://www.politika.rs/scc/clanak/413777/Kosovo-cas-anatomije

```



Slika 4: Prikaz grafa. A, B, C i D predstavljaju stranice.

## 2.5 PageRank algoritam

Naredni primer odnosi se na tzv. PageRank algoritam, odnosno, algoritam za rangiranje stranica. Ovaj algoritam je originalno objavljen od strane ljudi koji su učestvovali u kreiranju Google-a, Sergeja Brina i Lari Pejdzha i smatra se odgovornim za njegov rani uspeh. Funkcioniše tako što stranice posmatra kao čvorove u grafu, a veze između stranica kao grane i potom obezbeđuje globalno rangiranje stranica na webu (čvorova u grafu). Za pretraživače obezbeđuje rangiranje stranica nezavisno od upita. Glavna pretpostvka 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. Iterativna formula za PageRank je sledeća:

$$PR_{i+1}(P_i) = \sum_{P_j} \frac{PR_i(P_j)}{C(P_j)} \quad (3)$$

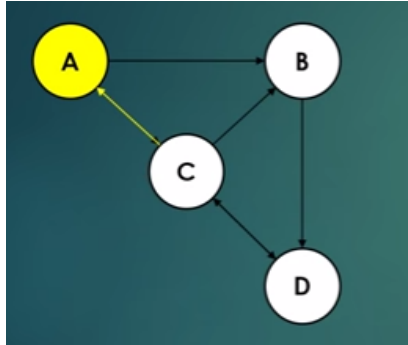
$PR_{i+1}$  predstavlja PageRank neke stranice  $P_i$  u narednoj iteraciji i računa se kao suma količnika PageRank-ova stranica koje pokazuju na tu stranicu u prethodnim iteracijama i broja "odlaznih" stranica. Da bismo bolje razumeli šta nam zapravo navedena formula govori posmatrajmo sliku ??.

Pošto je PageRank iterativan algoritam, u prvoj iteraciji rangove svih stranica postavljamo na  $\frac{1}{\text{ukupanbrojstranica}}$ , u primeru sa slike to bi bilo  $\frac{1}{4}$ .

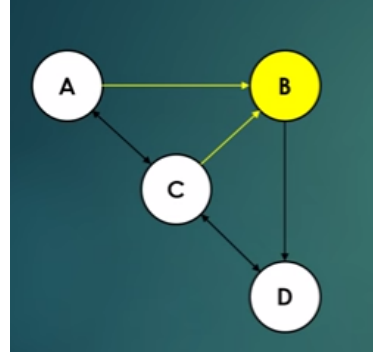
- $PR_0(A) = \frac{1}{4}$
- $PR_0(B) = \frac{1}{4}$
- $PR_0(C) = \frac{1}{4}$
- $PR_0(D) = \frac{1}{4}$

Potom u prvoj iteraciji (prikazanoj na slikama 5), posmatramo nultu i ukupan broj strana na koje pokazuje svaki od čvorova koji pokazuju na stranicu koju analiziramo.

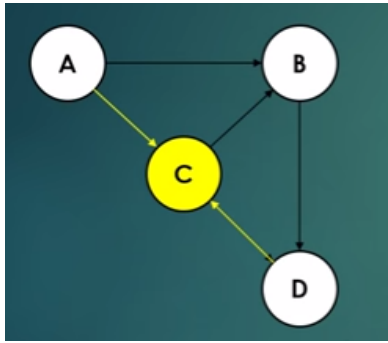
Na primeru, ako posmatramo čvor  $A$ , na njega pokazuje samo čvor  $C$ , koji pokazuje na čvorove  $A$ ,  $B$  i  $D$ . To znači da iz prethodne iteracije imamo  $\frac{1}{4}$ , a potom sve to delimo sa 3 (broj strana na koje se pokazuje sa strane  $C$ ). Analogno, važi za ostale čvorove. Dakle, dobijamo sledeće formule:



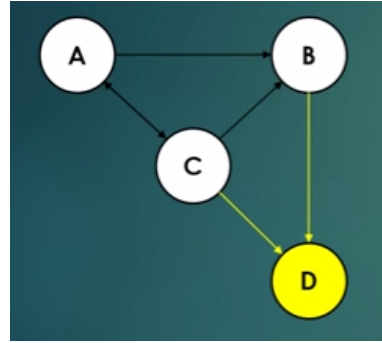
(a) Stranica C ima vezu (pokazuje) na stranicu A.



(b) Stranice A i C imaju vezu (pokazuju) na stranicu B.



(c) Stranice A i D imaju vezu (pokazuju) na stranicu C.



(d) Stranice B i C imaju vezu (pokazuju) na stranicu D.

Slika 5: PageRank prikaz prve iteracije.

- $PR_1(A) = \frac{\frac{1}{4}}{3} = \frac{1}{12}$
- $PR_1(B) = \frac{\frac{1}{4}}{2} + \frac{\frac{1}{4}}{3} = \frac{2.5}{12}$
- $PR_1(C) = \frac{\frac{1}{4}}{2} + \frac{\frac{1}{4}}{1} = \frac{4.5}{12}$
- $PR_1(D) = \frac{\frac{1}{4}}{1} + \frac{\frac{1}{4}}{3} = \frac{4}{12}$

U narednoj iteraciji, ponovo vršimo isti postupak:

- $PR_2(A) = \frac{\frac{4.5}{12}}{3} = \frac{1.5}{12}$
- $PR_2(B) = \frac{\frac{1}{12}}{2} + \frac{\frac{4.5}{12}}{3} = \frac{2}{12}$
- $PR_2(C) = \frac{\frac{1}{12}}{2} + \frac{\frac{4}{12}}{1} = \frac{4.5}{12}$
- $PR_2(D) = \frac{\frac{2.5}{12}}{1} + \frac{\frac{4.5}{12}}{3} = \frac{4}{12}$

Na kraju, dobijamo da je page rank:

- $PR(A) = 1$

- $PR(B) = 2$
- $PR(C) = 4$
- $PR(D) = 3$

Primetimo da se u svakoj iteraciji sume svih rangova sumiraju na 1. Što je viši broj u imeniocu razlomka, to je ta stranica važnija. Tako se uviđa da je na primeru stranica  $C$  najvažnija. Iza nje sledi stranica  $D$ , ako bismo pogledali sliku, možda nam na prvi pogled ne bi bilo odmah jasno zašto je stranica  $D$  druga najvažnija. Ovaj događaj je posledica toga što se važnost stranice određuje u odnosu na broj drugih važnih stranica koje ukazuju na nju. Tako je  $D$ , važnija od stranica  $A$  i  $B$ .

**Matrični pristup** Osim iterativnog pristupa za PageRank algoritam možemo koristiti i matrični pristup. Redom su  $A$ ,  $B$ ,  $C$  i  $D$  vrste i kolone matrice. Po kolonama, redom, smeštamo moguće verovatnoće izbora određenog čvora dostupnog iz tekućeg. Na primer, iz čvora  $A$  možemo doći do čvorova  $B$  i  $C$ , ali ne možemo do  $D$ , tako da u koloni  $A$  na pozicijama  $B$  i  $C$  upisujemo po jednu polovinu. Iz  $B$ , možemo doći samo do  $D$ , itd. Uočimo, da je ponovo suma po kolonama jednaka jedinici. Matrica izgleda na sledeći način:

$$\begin{bmatrix} 0 & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & \frac{1}{3} & 0 \\ \frac{1}{2} & 0 & 0 & 1 \\ 0 & 1 & \frac{1}{3} & 0 \end{bmatrix}$$

Ovakvu matricu nazivamo "stohastička" matrica, obeležavamo je sa  $H$  i množimo sa nekim vektorom  $v$ , dimenzija  $1 \times n$ , gde  $n$  predstavlja broj čvorova u grafu, odnosno, broj stranica. Na osnovu navedenog možemo izvesti formulu:

$$P_{i+1} = Hv \quad (4)$$

Vektor  $v$  je vrednost PageRank-a u prethodnim iteracijama, pa prethodnu formulu možemo predstaviti kao:

$$P_{i+1} = HP_i \quad (5)$$

Naredna iteracija vektora  $v$  je:

$$v_2 = Hv \quad (6)$$

$$v_3 = Hv_2 = H(Hv) = H^2v \quad (7)$$

Ako bismo nastavili da iteriramo, svaki put dodavajući  $H$ , posle nekog vremena došli bismo do stanja ekvilibrijuma, koje označava da ne treba dalje da iteriramo, jer smo našli vrednosti koje smo tražili. U vektoru  $v$  nam se nalaze rangovi stranica. Ovaj metod naziva se "**power method**".

Potencijalni problemi koji se mogu javiti prilikom primene PageRank algoritma:

- "viseći" čvorovi - algoritam neće raditi u situaciji kada imamo čvorove na koje niko ne pokazuje,
- ako graf sadrži komponente koje nisu povezane.

**Primer 2.5** Ovaj primer upravo implementira PageRank algoritam. Izlaz ovog algoritma neće biti predstavljen.

```
import numpy as np
from numpy.linalg import matrix_power
import urllib.request
from collections import deque
import re
from transliterate import translit
from googletrans import Translator

# Formiranje ulazne matrice za PageRank algoritam
# od dobijenog grafa
def generatePRMatrix(G):
    keys = {}
    num = 0
    for key in G:
        keys[key] = num
        num += 1
        for key2 in G[key]:
            if key2 not in keys:
                keys[key2] = num
                num += 1
    A = [[0 for i in range(num)] for j in range(num)]
    for v in G:
        neighbors = G[v]
        num = len(neighbors)
        for w in neighbors:
            A[keys[w]][keys[v]] = 1/num
    return np.matrix(A)

# Izracunavanje znacajnosti stranica pomocu PageRank algoritma.
# Matrica A je dobijena od grafa stranica
def page_rank(A):
    n = A.shape[0]
    # Pocetne vrednosti svake stranice su jednake i iznose 1/n,
    # gde je n ukupan broj stranica
    v = np.matrix([1/n for i in range(n)])
    return matrix_power(A, n).dot(np.transpose(v))

def transliterate(text):
    return translit(text, 'sr', reversed=True)

def get_webpage(URL):
    req = urllib.request.Request(URL)
    response = urllib.request.urlopen(req)
    data = response.read().decode('utf-8')
    data = data.replace('\n', '')
    data = data.replace('\r', '')
    data = data.replace('\t', '')
```

```

        data = data.replace("'", ' ')
        data = data.replace('\\', '\\\\')
        space_regex = re.compile(r'[ ]+')
        data = space_regex.sub(' ', data)
        return transliterate(data)

def clean(rawHTML):
    tags = re.compile(r'<.*?>|\&.+;')
    return tags.sub('', rawHTML)

def extract_elements(element, rawHTML):
    regex = '<' + element + '(.*)>(.*?)</' + element + '>'
    raw_elements = re.findall(regex, str(rawHTML))
    elements = []
    for i in range(len(raw_elements)):
        elements.append((raw_elements[i][0], clean(raw_elements[i][1])))
    return elements

def extract_attributes(attribute, element):
    regex = attribute + '="(.*?)"'
    attributes = re.findall(regex, element[0])
    return attributes

def get_neighbors(base, URL, relative_only=True):
    if URL[0] == '/':
        URL = base + URL
    pageHTML = get_webpage(URL)
    elements = extract_elements('a', pageHTML)
    links = []
    for element in elements:
        links += extract_attributes('href', element)
    if relative_only:
        links = list(filter(lambda x: x[0] == '/', links))
        return [base + link for link in links]
    else:
        return links

def addEdge(G, v_from, v_to):
    if v_from not in G:
        G[v_from] = []
    if v_to not in G[v_from]:
        G[v_from].append(v_to)

def bfs(startPage):
    marked = {}
    marked[startPage] = True
    queue = deque(['/'])
    base = startPage
    G = {}
    limit = 300

```

```

currNum = 0;
while len(queue) > 0:
    if (currNum == limit):
        break;
    curr = queue.popleft()
    print(curr)
    neighbors = get_neighbors(base, startPage, True)
    for neighbor in neighbors:
        addEdge(G, curr, neighbor)
        if neighbor not in marked and not re.search('#', neighbor):
            marked[neighbor] = True
            queue.append(neighbor)
        currNum += 1

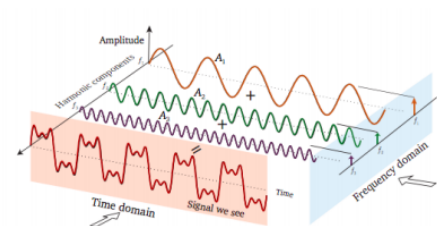
return G

def main():
    G = bfs('http://www.politika.rs')
    A = generatePRMatrix(G)
    print(page_rank(A))

if __name__ == "__main__":
    main()

```





Slika 6: Ilustracija analize signala uz pomoć Furijeove transformacije.

### 3 Furijeova transformacija

Furijeovi koeficijenti funkcije omogućavaju analizu signala u odnosu na frekvencije koje su u njemu zastupljene, odnosno spektar signala. Furijeova transformacija omogućava prevođenje reprezentacije funkcije iz vremenskog domena u frekvencijski domen. Obrnuto se postiže inverznom Furijeovom transformacijom.