



Mašinsko učenje

Klasterovanje

Nemanja Maček

- Uvodne napomene
- Merenje povezanosti postova
- Predprocesiranje
- Klaster analiza
- Skup podataka
- Klasterovanje postova
- Rešavanje zadatog problema
- Osvrt na šum
- Zaključne napomene

U čemu je razlika između klasifikacije i klasterovanja?

- Na prethodnom predavanju bavili smo se klasifikacijom kao vidom nadgledanog učenja.
- Podaci za obuku modela su bili obeleženi, a obučeni model se na dalje koristio za klasifikaciju prethodno nepoznatih instanci.
- Postavljamo sledeće pitanje: šta možemo da uradimo ukoliko nemamo obeležene instance za obuku modela?
- Očigledno, model klasifikacije ne možemo da izgradimo. Ali možemo da pronađemo određene šablone u podacima.

Formalno rečeno (podsetnik) ...

- U savremenoj oblasti mašinskog učenja dominiraju sledeći pristupi:
 - induktivno učenje,
 - analitičko učenje (analogija sa logikom),
 - učenje na slučajevima (engl. *case-based learning* – analogija sa ljudskim pamćenjem),
 - neuralne mreže (analogija sa neurobiologijom),
 - genetski algoritmi (analogija sa evolucijom) i
 - hibridni modeli (kombinacija više pristupa).
- Od najvećeg značaja za tekuću praksu u domenu računarstva i veštačke inteligencije je induktivno učenje.
- Suština ovog tipa učenja je učenje na osnovu raspoloživih primera. U svakodnevnom jeziku to možemo da nazovemo: učenje iz iskustva (drugih).
- S obzirom na objekat učenja najopštije je učenje funkcionalnih, tj. ulazno-izlaznih preslikavanja.

Formalno rečeno (podsetnik) ...

- Ključni elementi u induktivnom učenju funkcionalnih preslikavanja su: nepoznato preslikavanje, obučavajući skup ulazno – izlaznih parova i skup hipoteza unutar koga bираmo finalnu hipotezu putem algoritma obučavanja.
- Ulazi su po pravilu n -dimenzioni vektori. U literaturi se pominju pod nazivima: vektori obeležja, uzorci, primeri i instance.
- Komponente ulaznih vektora se nazivaju obeležja ili atributi i mogu biti: kontinualni (beskonačan broj vrednosti) i diskretni (konačan broj vrednosti).
- Izlazni prostor može biti: kategorijalan sa K distiktnih vrednosti, kada obučeni sistem obavlja klasifikaciju (izlaz se naziva oznaka, klasa, kategorija ili odluka) ili realan, kada obučeni sistem realizuje regresiju (funkcionalni estimator).

Formalno rečeno (podsetnik) ...

- Grubo, vrste obučavanja možemo da podelimo na: obučavanje sa učenjem, obučavanje bez učitelja i obučavanje sa podsticajem (engl. *reinforcement learning*).
- Za obučavanje sa učiteljem, tj. nadgledano obučavanje (engl. *supervised learning*) važi:
 - obučavajući skup je oblika $\{x_i, f(x_i)\}$, gde je $f(x_i)$ ciljna vrednost vektora obeležja x_i ,
 - zadatak obučavanja je nalaženje aproksimacije za f , a
 - mera performansi je kvalitet aproksimacije u tačkama koje ne pripadaju obučavajućem skupu.
- Za obučavanje bez učitelja, tj. nenadgledano obučavanje (engl. *unsupervised learning*) važi da su u obučavajućem skupu su prisutne samo ulazne instance x_i .
- Tipičan pristup nenadgledanom učenju je klasterovanje, odnosno grupisanje raspoloživih podataka u manji broj grupa unutar kojih su podaci sličniji u poređenju sa podacima iz ostalih grupa. Imenovanjem klastera se dolazi posrednim putem do označenih uzoraka.

Šta ćemo raditi na ovom predavanju?

- Transformisaćemo postove iz grupa vesti (engl. *newsgroup*) u vektore pogodne za dalju obradu koristeći tzv. *bag of words* pristup,
- grupisati postove u klastere koristeći K-sredina (engl. *K-means*) algoritam i
- izdvojiti postove koji su povezani (dovoljno slični) sa novim postom, tj. postom koji nije bio prisutan u početnom skupu postova.

Da li je običan, neobrađen tekst od koristi?

- Sa stanovišta mašinskog učenja, običan, odnosno nepredprocesiran tekst, nam je u najvećem broju slučajeva beskoristan.
- Međutim, tekst koji smo na neki način pretvorili u numeričke vrednosti od značaja je u nekim slučajevima upotrebljiv – na takve podatke mogu se primeniti određeni algoritmi, poput klasterovanja.
- Ovo važi za neke operacije nad tekstom, kao što je merenje sličnosti.

Kako ne treba rešavati problem?

- Minimalno rastojanje između dva stringa (engl. *minimum edit distance*) jednako je minimalnom broju operacija umetanja znaka, brisanja znaka ili zamene jednog znaka drugim, potrebnih da se jedan string transformiše u drugi.
- Što je minimalno rastojanje između dva stringa manje, stringovi su sličniji.
- Na primer, minimalno rastojanje između stringova „mirna“ i „igra“ je 3 jer su potrebne tri operacije za transformaciju jednog stringa u drugi:
 - brisanje znaka „m“: „mirna“ $\rightarrow$ „irna“,
 - zamena znaka „r“ slovom „g“: „irna“ $\rightarrow$ „igna“,
 - zamena znaka „n“ slovom „r“: „irna“ $\rightarrow$ „igra“.
- U opštem slučaju, postoji više načina da se jedan string transformiše u drugi koristeći isti broj operacija.
- Za prethodni primer postoje još dva načina transformacije koristeći minimalni broj operacija.

Kako ne treba rešavati problem?

- Minimalno rastojanje između stringova koje smo razmatrali u prethodnom primeru se naziva i Levenštajново rastojanje (engl. *Levenshtein distance*).
- Levenštajново rastojanje je specijalni slučaj rastojanja između stringova, u kom se podrazumeva da operacije umetanja, brisanja i zamene znakova podjednako doprinose rastojanju između stringova, i da je dovoljno samo utvrditi minimalni broj različitih operacija potrebnih da se jedan string transformiše u drugi.
- Treba imati na umu da određivanje Levenštajnovog rastojanja može biti računski zahtevno ukoliko su stringovi vrlo dugi, zato što zavisi od dužine prvog i drugog stringa.

Kako ne treba rešavati problem?

- Imajući to na umu možemo da primenimo sledeći štos – tretiraćemo cele reči u postovima kao karaktere i određivaćemo minimalno rastojanje na nivou reči.
- Zarad jednostavnosti, u sledećem primeru ćemo analizirati samo naslove postova: „How to format my hard disk“ i „Hard disk format problems“.
- U ovom slučaju minimalno rastojanje je 5 – potrebno je izbaciti reči „How“, „to“ i „format“, a zatim dodati reči „format“ i „problems“.
- Na ovaj način se razlika između dva posta može izraziti kao broj reči koje treba dodati ili izbrisati, tako da se jedan tekst pretvori u drugi.
- Međutim, treba imati na umu da, iako smo postupak ubrzali, vremenska složenost i dalje ostaje problem.

Kako ne treba rešavati problem?

- Prilikom računanja minimalnog rastojanja javlja se još jedan problem.
- U prethodnom primeru smo najpre izbacili reč „format“, a zatim je dodali, što je povećalo minimalno rastojanje za 2, ali se reč i dalje nalazi u oba naslova.
- To ukazuje na činjenicu da jednostavno računanje minimalnog rastojanja nije dovoljno robusno da u obzir uzme promenu redosleda reči.

Kako treba rešavati problem?

- Bolji pristup od minimalnog rastojanja je princip *bag of words*.
- U ovom pristupu se koristi broj pojavljivanja date reči u postu, a red reči ignoriše.
- Za svaku reč u postu određuje se broj pojavljivanja koji se unosi u vektor.
- Vektor je obično ogroman jer sadrži onoliko elemenata koliko različitih reči imamo u celom skupu podataka.
- Ovaj korak se naziva vektORIZACIJA.

Kako treba rešavati problem?

- Za prethodni primer (post 1: “How to format my hard disk”, post 2: “Hard disk format problems”) dobijamo:

Reč	Broj pojavljivanja u postu 1	Broj pojavljivanja u postu 2
disk	1	1
format	1	1
how	1	0
hard	1	1
my	1	0
problems	0	1
to	1	0

Kako treba rešavati problem?

- Kolone „Broj pojavljivanja u postu 1“ i „Broj pojavljivanja u postu 2“ su vektori.
- Nadalje se može računati Euklidsko rastojanje između datog vektora i vektora svih postova i pronaći najmanje. Međutim, ovaj postupak takođe može biti spor.
- Ove vektore korisimo kao vektore obeležja u postupku klasterovanja i pronalaženja povezanih postova na sledeći način:
 - ekstrahuju se istaknuta obeležja iz svakog posta i čuvaju se kao vektor za dati post,
 - primeni se klasterovanje (grupisanje) na vektore,
 - odredi se klaster kome određeni post pripada,
 - iz datog klastera, uzima se nekoliko postova koji imaju različitu, ali dovoljno malu sličnost sa datim postom, čime se povećava raznolikost odgovora.

Konverzija neobrađenog teksta u *bag of words*.

- Za brojanje reči i predstavljanje u obliku vektora koristićemo SciKit `CountVectorizer()` metodu.

```
>>> from sklearn.feature_extraction.text import CountVectorizer  
>>> vectorizer = CountVectorizer(min_df=1)
```

- Parametar `min_df` određuje kako CountVectorizer tretira retke reči.
- Ukoliko je u pitanju celobrojna vrednost, sve reči koje se pojavljuju manji broj puta od te vrednosti, biće odbačene.
- Ukoliko je u pitanju razlomak, sve reči koje se pojavljuju manje od razlomkom određenog dela celokupnog skupa podataka, biće odbačene.
- Parametar `max_df` se ponaša na sličan način.

Konverzija neobrađenog teksta u *bag of words*.

- Izlaz komande `print(vectorizer)` ispisuje na ekranu podrazumevane vrednosti.
- Za sada su od značaja sledeće:

```
analyzer='word'                # brojanje na nivou reči  
token_pattern='(?u)\\b\\w\\w+\\b' # šablon za tokenizaciju
```

- To znači da se:
 - brojanje radi na nivou reči,
 - obavlja tokenizacija, tj. deljenje teksta na tokene (nizove znakova koji predstavljaju semantičke jedinice pogodne za dalju obradu).
- Intuitivno, tokenizacija se može posmatrati kao deljenje teksta na pojedine reči, pri čemu se zanemaruju znakovi interpunkcije.
- Na primer, tekst „cross-validated“ biće razdvojen u dva tokena – „cross“ i „validated“.

Konverzija neobrađenog teksta u *bag of words*.

- Posmatramo dve rečenice koje predstavljaju naslove postova.

```
>>> content = ["How to format my hard disk", "Hard disk format problems"]
```

- Funkcija `fit_transform()` obavlja vektorizaciju.

```
>>> X = vectorizer.fit_transform(content)
```

```
>>> vectorizer.get_feature_names()
```

```
[u'disk', u'format', u'hard', u'how', u'my', u'problems', u'to']
```

Konverzija neobrađenog teksta u *bag of words*.

- Dakle, otkriveno je sedam reči, a njihova pojavljivanja u rečenicama možemo videti i „ručno“ izbrojati.

```
>>> print(X.toarray().transpose())
```

```
[[1 1]  
 [1 1]  
 [1 1]  
 [1 0]  
 [1 0]  
 [0 1]  
 [1 0]]
```

- To znači da se u prvoj rečenici nalaze sve reči osim reči „problems“, a u drugoj sve reči osim „how“, „my“ i „to“.

Brojanje reči.

- Kao primer, tražimo post u sledećem skupu najsličniji kratkoj rečenici „imaging databases“.
- Datoteke se nalaze u dodatku knjige, direktorijum data/ch03/data/toy.

Ime datoteke	Sadržaj posta
01.txt	This is a toy post about machine learning. Actually, it contains not much interesting stuff.
02.txt	Imaging databases can get huge.
03.txt	Most imaging databases save images permanently.
04.txt	Imaging databases store images.
05.txt	Imaging databases store images. Imaging databases store images. Imaging databases store images.

Brojanje reči.

- Pod pretpostavkom da se postovi nalaze u direktorijumu DIR, uvešćemo ih u CountVectorizer.
- Pri tome moramo da naglasimo da se radi o celom skupu kako bi imao uvid u reči koje može da očekuje.

```
>>> posts = [open(os.path.join(DIR, f)).read() for f in os.listdir(DIR)]
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> vectorizer = CountVectorizer(min_df=1)
>>> X_train = vectorizer.fit_transform(posts)
>>> num_samples, num_features = X_train.shape
>>> print("#samples: %d, #features: %d" % (num_samples, num_features))
#samples: 5, #features: 25
```

Brojanje reči.

- Ukupno imamo 5 postova sa 25 reči. Sledeće reči koje su tokenizovane biće brojane.

```
>>> print(vectorizer.get_feature_names())  
[u'about', u'actually', u'capabilities', u'contains', u'data',  
u'databases', u'images', u'imaging', u'interesting', u'is', u'it',  
u'learning', u'machine', u'most', u'much', u'not', u'permanently',  
u'post', u'provide', u'save', u'storage', u'store', u'stuff',  
u'this', u'toy']
```

Brojanje reči.

- Sledeće što radimo je vektorizacija nove rečenice „imaging databases“.
- Vektor vraćen metodom transformacije ne čuva vrednosti broja za svaku reč (tzv. *sparse* format), pošto će većina tih brojeva biti nula (post ne sadrži reč).
- Ukoliko želite da vidite sve vrednosti, koristi se metod `toarray()`. Ovo nam je potrebno za računanje sličnosti između postova.

```
>>> new_post = "imaging databases"
>>> new_post_vec = vectorizer.transform([new_post])
>>> print(new_post_vec)
(0, 7) 1
(0, 5) 1
>>> print(new_post_vec.toarray())
[[0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
```

Brojanje reči.

- Kao meru sličnosti možemo koristiti Euklidsko rastojanje između novog posta i starih postova.
- Definisaćemo funkciju `dist_raw` koja računa Euklidsko rastojanje koristeći `norm()` funkciju.

```
import scipy as sp
def dist_raw(v1, v2):
    delta = v1-v2
    return sp.linalg.norm(delta.toarray())
```

- Koristeći definisanu funkciju, potrebno je proći kroz sve postove i naći najbliži.
- Deo koda kojim se to radi dat je na str. 25, za kompletan kod v. dodatak knjige.

Brojanje reči.

```
import sys
best_doc = None
best_dist = sys.maxint
best_i = None
for i, post in enumerate(num_samples):
    if post == new_post:
        continue
    post_vec = X_train.getrow(i)
    d = dist_raw(post_vec, new_post_vec)
    print("=== Post %i with dist=%.2f: %s"%(i, d, post))
    if d < best_dist:
        best_dist = d
        best_i = i
print("Best post is %i with dist=%.2f"%(best_i, best_dist))
```

Brojanje reči.

- Kada se prethodni kod pokrene, dobija se sledeći izlaz:

=== Post 0 with dist=4.00: This is a toy post about machine learning.
Actually, it contains not much interesting stuff.

=== Post 1 with dist=1.73: Imaging databases provide storage capabilities.

=== Post 2 with dist=2.00: Most imaging databases save images permanently.

=== Post 3 with dist=1.41: Imaging databases store data.

=== Post 4 with dist=5.10: Imaging databases store data. Imaging databases store data. Imaging databases store data.

Best post is 3 with dist=1.41

Brojanje reči.

- Šta možemo da zaključimo?
- Post 0 je najmanje sličan, zato što nema nijednu zajedničku reč sa novom rečenicom.
- Post 1 je vrlo sličan, ali, u odnosu na post 3 sadrži jednu reč viška koje nema u novoj rečenici.
- Šta je u izlazu prethodnog koda čudno?
- Post 4 možemo da posmatramo kao utrostručeni post 3 (tri puta ponovljen).
- Iako neko može da pomisli da bi sličnost nove rečenice sa postovima 3 i 4 trebala biti ista, prikazivanjem njihovih vektora se može zaključiti da je brojanje reči isuviše jednostavno i da je potrebno normalizovati vektore brojanja reči.

```
>>> print(X_train.getrow(3).toarray())  
[[0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0]]  
>>> print(X_train.getrow(4).toarray())  
[[0 0 0 0 3 3 0 3 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0]]
```

Normalizovanje vektora brojanja reči.

- Izмениćemo funkciju `dist_raw` tako da računa rastojanje između normalizovanih vektora.

```
def dist_norm(v1, v2):  
    v1_normalized = v1/sp.linalg.norm(v1.toarray())  
    v2_normalized = v2/sp.linalg.norm(v2.toarray())  
    delta = v1_normalized - v2_normalized  
    return sp.linalg.norm(delta.toarray())
```

- U ovom slučaju se, prilikom poređenja nove rečenice sa svim postovima dobija drugačiji rezultat.

Normalizovanje vektora brojanja reči.

- Iz izlaza programa se vidi da su sada post 3 i post 4 podjednako slični novoj rečenici.

=== Post 0 with dist=1.41: This is a toy post about machine learning.

Actually, it contains not much interesting stuff.

=== Post 1 with dist=0.86: Imaging databases provide storage capabilities.

=== Post 2 with dist=0.92: Most imaging databases save images permanently.

=== Post 3 with dist=0.77: Imaging databases store data.

=== Post 4 with dist=0.77: Imaging databases store data. Imaging databases store data. Imaging databases store data.

Best post is 3 with dist=0.77

Uklanjanje stop reči.

- Ukoliko ponovo pogledamo post 2, primetićemo da se reči „most“, „save“, „images“ i „permanently“ ne nalaze u novoj rečenici.
- Uticaj ovih reči na značenje posta, odnosno količina informacije koju nose, je različita.
- Reči poput „most“ (u prevodu: većina) se često pojavljuju u svim različitim kontekstima i nazivaju se stop reči (engl. *stop words*).
- Stop reči ne sadrže toliko informacija i ne treba im pridavati isti značaj kao rečima poput „images“, koje se ne pojavljuju često u različitim kontekstima.
- Najbolja opcija bila bi uklanjanje svih reči koje su tako česte da ne pomažu u razlikovanju različitih tekstova.

Uklanjanje stop reči.

- Pošto je uklanjanje stop reči tipičan korak u obradi prirodnih jezika, dovoljno je parametar `stop_words` podesiti na jezik čije stop reči želimo da uklonimo.
- Za engleski jezik uklanja se 318 stop reči, od kojih je prvih 20 dato u listing (radi ilustracije).

```
>>> vectorizer = CountVectorizer(min_df=1, stop_words='english')
>>> sorted(vectorizer.get_stop_words())[0:20]
['a', 'about', 'above', 'across', 'after', 'afterwards', 'again',
'against', 'all', 'almost', 'alone', 'along', 'already', 'also',
'although', 'always', 'am', 'among', 'amongst', 'amoungst']
```

- Nakon uklanjanja stop reči, uklanja se 7 tokenizovanih stop reči iz našeg primera: „about“, „is“, „it“, „most“, „much“, „not“ i „this“.

Uklanjanje stop reči.

- Nakon uklanjanja stop reči dobićemo sledeće mere sličnosti (primetite da je udaljenost postova 1 i 2 od nove rečenice sada ista, jer imaju identičan broj reči kojih nema u novoj rečenici):

=== Post 0 with dist=1.41: This is a toy post about machine learning.

Actually, it contains not much interesting stuff.

=== Post 1 with dist=0.86: Imaging databases provide storage capabilities.

=== Post 2 with dist=0.86: Most imaging databases save images permanently.

=== Post 3 with dist=0.77: Imaging databases store data.

=== Post 4 with dist=0.77: Imaging databases store data. Imaging databases store data. Imaging databases store data.

Best post is 3 with dist=0.77

Stemming.

- Do sada smo različite pojavne oblike reči i slične reči posmatrali kao različite reči.
- Na primer, post 2 sadrži reči „imaging“ i „images“.
- Pošto reči ukazuju na isti ili vrlo slični koncept, možemo da ih brojimo zajedno.
- Dakle, potrebna nam je funkcija koja će reči svesti na stem oblik koji je pogodan za brojanje.
- SciKit ovakvu funkciju za sada nema, ali se zato može preuzeti besplatan alat Natural Language Toolkit (NLTK) – ovaj alat obezbeđuje stemer koji se može jednostavno koristiti uz CountVectorizer.
- NLTK obezbeđuje više stemera – ovo je logično zato što svaki jezik ima različita pravila dovođenja reči u osnovni oblik.
- Za Engleski jezik koristićemo SnowballStemmer. Uzmite u obzir da rezultat ove operacije ne mora uvek da rezultuje ispravnim engleskim rečima.

* Za detalje o instalaciji NTLK, v. knjigu str. 60.

Stemming.

- Primer:

```
>>> import nltk.stem
>>> s = nltk.stem.SnowballStemmer('english')
>>> s.stem("graphics")
u'graphic'
>>> s.stem("imaging")
u'imag'
>>> s.stem("image")
u'imag'
>>> s.stem("imagination")
u'imagin'
>>> s.stem("imagine")
u'imagin'
```

Stemming.

- Primer:

```
>>> s.stem("buys")
```

```
u'buy'
```

```
>>> s.stem("buying")
```

```
u'buy'
```

- Napomena – ova operacija ponekad ne vraća stem oblik. Na primer, stemovanje reči „bought“ trebalo bi da rezultuje rečju „buy“.

```
>>> s.stem("bought")
```

```
u'bought'
```

Stemming.

- Proširivanje CountVectorizer klase NLTK funkcionalnošću:

```
import nltk.stem
english_stemmer = nltk.stem.SnowballStemmer('english')
class StemmedCountVectorizer(CountVectorizer):
    def build_analyzer(self):
        analyzer = super(StemmedCountVectorizer, self).build_analyzer()
        return lambda doc: (english_stemmer.stem(w) for w in analyzer(doc))
vectorizer = StemmedCountVectorizer(min_df=1, stop_words='english')
```

- Šta dati kod radi?

Stemming.

- Prethodni kod:
 - pretvara sva slova u mala slova u fazi predprocesiranja,
 - izdvaja sve reči u procesu tokenizacije,
 - izvacuje stop reči i
 - pretvara reči u stem oblik.
- Nakon pokretanja ovog koda izbačena je još jedno obeležje, odnosno stem oblik reči, zato što su nakon stemming-a oblici reči „images“ i „imaging“ identični.

```
print(vectorizer.get_feature_names())  
[u'actual', u'capabl', u'contain', u'data', u'databas', u'imag',  
u'interest', u'learn', u'machin', u'perman', u'post', u'provid',  
u'save', u'storag', u'store', u'stuff', u'toy']
```

Stemming.

- Koji je sad post najbližiji rečenici „imaging databases“?

=== Post 0 with dist=1.41: This is a toy post about machine learning. Actually, it contains not much interesting stuff.

=== Post 1 with dist=0.86: Imaging databases provide storage capabilities.

=== Post 2 with dist=0.63: Most imaging databases save images permanently.

=== Post 3 with dist=0.77: Imaging databases store data.

=== Post 4 with dist=0.77: Imaging databases store data. Imaging databases store data. Imaging databases store data.

Best post is 2 with dist=0.63

Term frequency – inverse document frequency (TF-IDF).

- Za sada možemo da kažemo da smo pronašli dobar način da formiramo kompaktni vektor na osnovu zašumljenog tekstualnog posta.
- Međutim, postavićemo pitanje: šta vrednosti obeležja zapravo znače u ovom slučaju?
- Vrednosti obeležja se dobijaju na osnovu broja pojavljivanja termina u postu, a naša pretpostavka je da veća vrednost znači da je termin od većeg značaja.
- Međutim, može se izvesti zaključak da duži dokumenti imaju prednost nad kraćima.
- Dakle, potrebno je normalizovati brojanje na osnovu dužine dokumenta.
- Takođe, potrebno je izbaciti beskorisne termine (na primer, termin „subject“, koji se nalazi u skoro svakom postu).

Term frequency – inverse document frequency (TF-IDF).

- Jednostavna implementacija TF-IDF:

```
import scipy as sp
def tfidf(term, doc, corpus):
    tf = doc.count(term) / len(doc)
    num_docs_with_term = len([d for d in corpus if term in d])
    idf = sp.log(len(corpus) / num_docs_with_term)
    return tf * idf
```

Term frequency – inverse document frequency (TF-IDF).

- Za dati korpus D koji sadrži tokenizovane dokumente a, abb i abc prethodna funkcija vraća sledeće:

```
>>> a, abb, abc = ["a"], ["a", "b", "b"], ["a", "b", "c"]
```

```
>>> D = [a, abb, abc]
```

```
>>> print(tfidf("a", a, D))
```

```
0.0
```

```
>>> print(tfidf("a", abb, D))
```

```
0.0
```

```
>>> print(tfidf("a", abc, D))
```

```
0.0
```

- Na osnovu toga zaključujemo da je termin a neznačajan jer se nalazi svugde.

Term frequency – inverse document frequency (TF-IDF).

- Za dati korpus D koji sadrži tokenizovane dokumente a, abb i abc prethodna funkcija vraća:

```
>>> print(tfidf("b", abb, D))
```

```
0.270310072072
```

```
>>> print(tfidf("b", abc, D))
```

```
0.135155036036
```

```
>>> print(tfidf("c", abc, D))
```

```
0.366204096223
```

- Na osnovu toga zaključujemo da je termin b značajniji za dokument abb nego za document abc jer se u njemu pojavljuje dva puta.

Term frequency – inverse document frequency (TF-IDF).

- SciKit implementacija (koristimo TfidfVectorizer koji nasleđuje CountVectorizer) je data sledećim kodom:

```
from sklearn.feature_extraction.text import TfidfVectorizer
class StemmedTfidfVectorizer(TfidfVectorizer):
    def build_analyzer(self):
        analyzer = super(TfidfVectorizer, self).build_analyzer()
        return lambda doc: (english_stemmer.stem(w) for w in analyzer(doc))
vectorizer = StemmedTfidfVectorizer(min_df=1, stop_words='english', decode_error='ignore')
```

Koji su nedostaci pristupa *bag of words*?

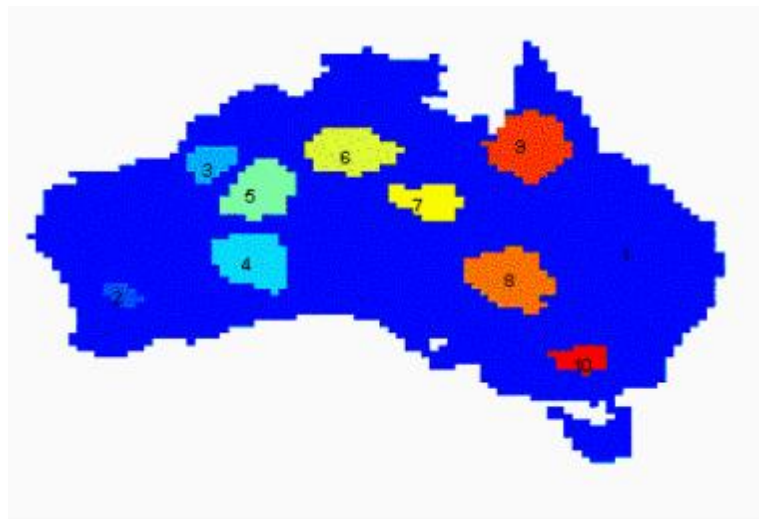
- Relacije između reči se ne uzimaju u obzir.
 - Na primer, rečenice „cars hits wall“ i „wall hits car“ imaće identičan vektor obeležja.
- Postoji problem sa negacijama.
 - Na primer, rečenice „I will eat ice cream“ i „I will not eat ice cream“ imaće vrlo sličan vektor obeležja i ako imaju suprotno značenje.
 - Ovaj problem je rešiv ukoliko se prilikom brojanja reči u obzir ne uzmu samo pojedinačne reči, već i bigrami (parovi reči) ili trigrami (tri reči za redom).
- Postoji problem sa pogrešno napisanim rečima.
 - Iako je čitaocu jasno da reči „database“ i „databas“ imaju isto značenje, odnosno da je u pitanju greška u pisanju, pristup koji mi koristimo će ih tretirati kao dve različite reči.

Koji su nedostaci pristupa *bag of words*?

- Međutim, fokus ovog izlaganja je na klasterovanju, a ne na obradi prirodnih jezika, tako da ćemo pristup *bag of words* koristiti za dalju klaster analizu.

Šta je klasterovanje?

- Klasterovanje, tj. klaster analiza je pronalaženje grupa objekata takvih da su objekti u grupi slični (ili povezani), i da su objekti u različitim grupama različiti (ili nepovezani).
- Klasterovanje se koristi za pregledanje dokumenata iz iste grupe, grupisanje gena i proteina koji imaju iste funkcionalnosti, grupisanje akcija sa sličnim promenama cena itd.
- Primer: klasterovanje padavina u Australiji.



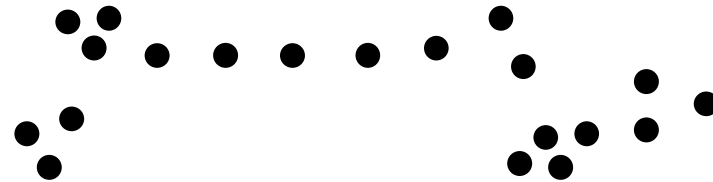
Dvosmislenost pojma klastera.

- Koliko klastera imamo u ovom slučaju?



Dvosmislenost pojma klastera.

- Koliko klastera imamo u ovom slučaju?



Koji tipovi klasterovanja postoje?

- Postoji značajna razlika između particionog i hijerarhijskog skupa klastera.
- Particiono klasterovanje je podela skupa podataka u nepreklapajuće podskupove (klastere) takve da je svaki podatak tačno u jednom podskupu.
- Hijerarhijsko klasterovanje je skup ugnježdenih klastera organizovan u obliku hijerarhijskog drveta.
- Hijerarhijsko klasterovanje može biti tradicionalno i netradicionalno, što rezultuje tradicionalnim i netradicionalnim dendogramom.

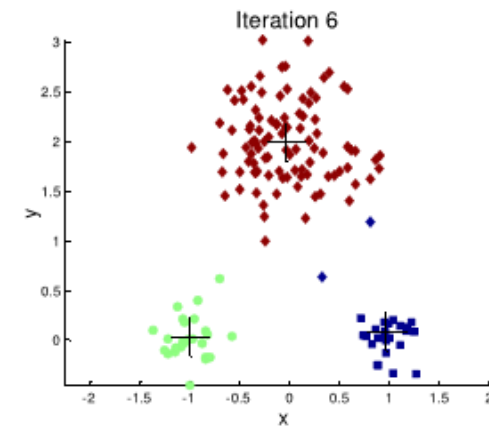
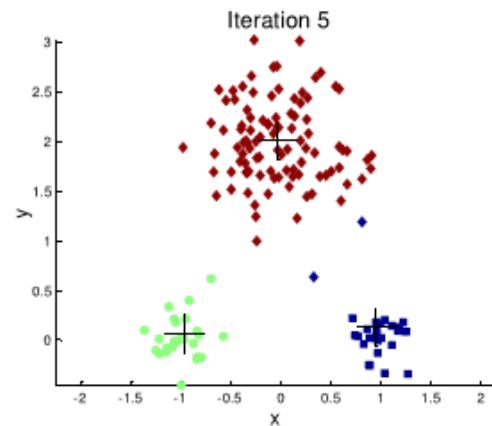
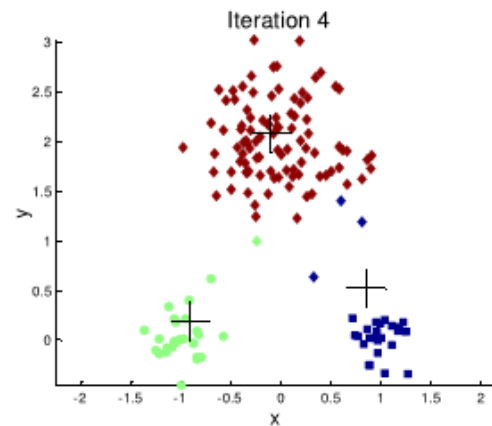
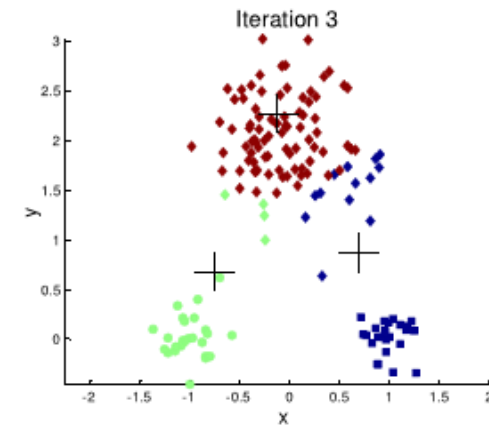
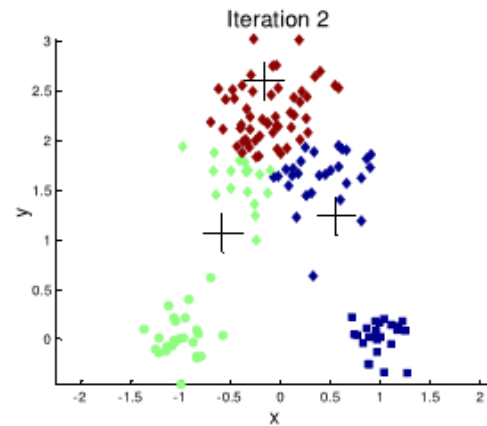
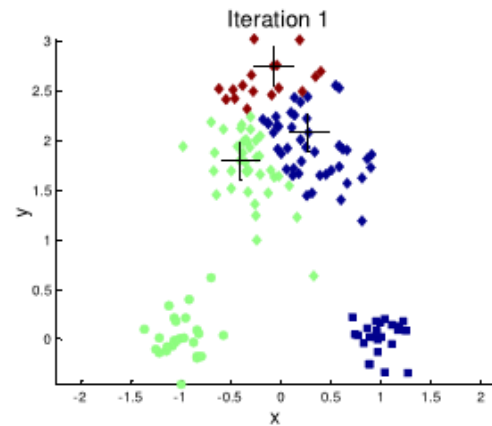
Klasterovanje pomoću algoritma K-sredina.

- K-sredina (engl. *K-means*) je pristup particionom klasterovanju u kome je svakom klasteru pridružen centroid (centralna tačka) a svaka tačka je dodeljena klasteru sa najbližim centroidom.
- Broj klastera K mora da se navede.
- Osnovni algoritam je krajnje jednostavan:
 1. *Select K points as initial centroids.*
 2. **repeat**
 3. *Form K clusters by assigning all points to the closest centroid.*
 4. *Recompute centroid for each cluster.*
 5. **until** *the centroids don't change*

Klasterovanje pomoću algoritma K-sredina.

- Početni centroid se često bira na slučajni način.
- Dobijeni klasteri mogu da se razlikuju u uzastopnim izvršenjima programa. Rezultati mogu da budu relativno loši.
- Uobičajeno je da je centroid srednja vrednost tačaka u klasteru.
- „Najbliže“ se meri kao Euklidsko rastojanje, kosinusno rastojanje, korelacija, itd.
- K-sredine konvergiraju ka prethodno pomenutim merama sličnosti.
- Najveći deo konvergencije se dešava u prvih nekoliko iteracija.
- Često se uslov zaustavljanja menja na „sve dok relativno malo tačaka ne promeni klaster“.

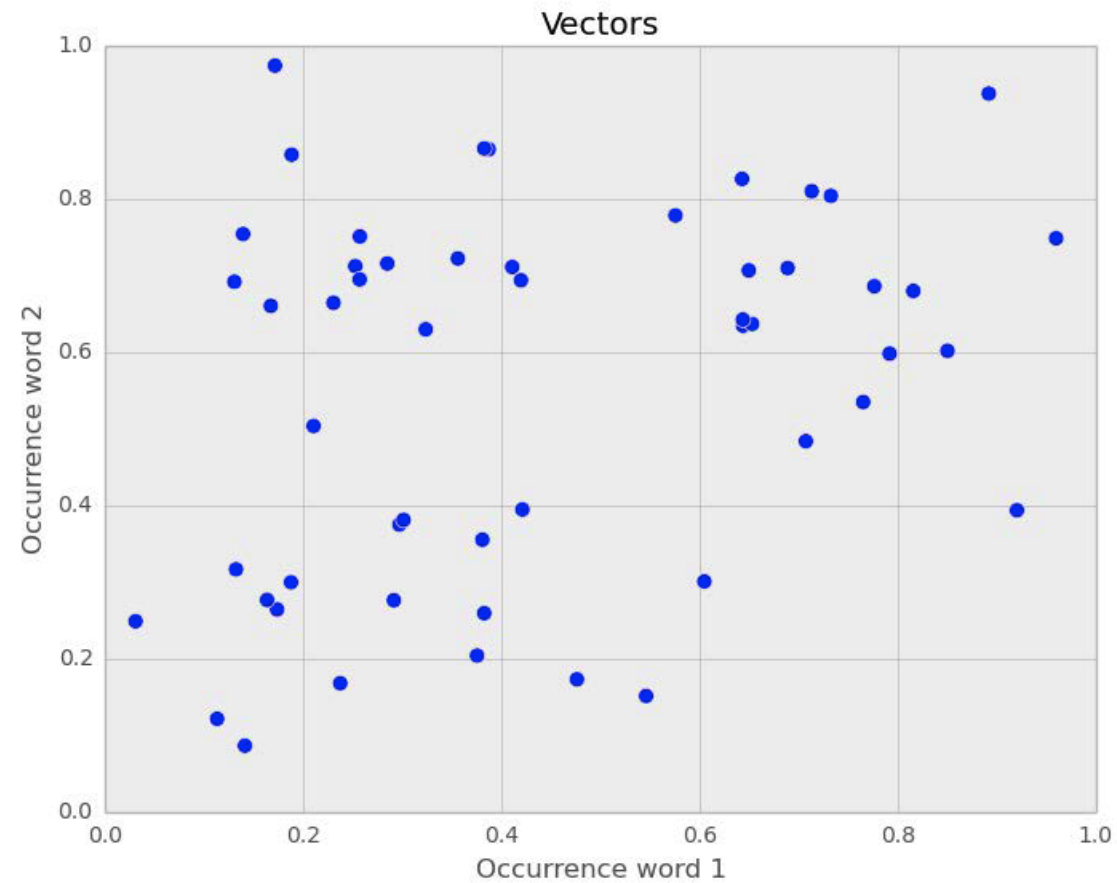
Klasterovanje pomoću algoritma K-sredina.



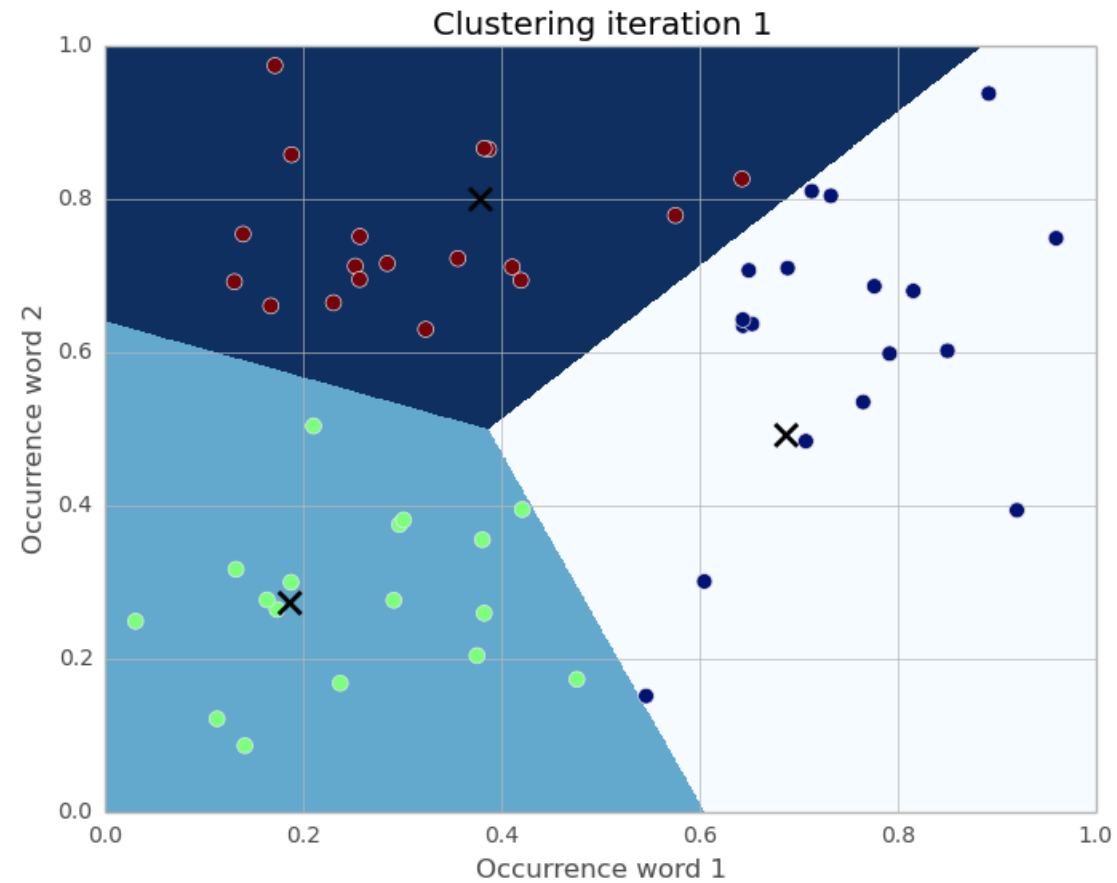
Jednostavan primer.

- Klaster analizu tekstualnih podataka možemo da ilustrujemo klasterovanjem postova koji sadrže dve reči.
- Strelice na grafikonu na str. 56 označavaju pomeraj centroida.
- Nakon pet iteracija na ovom primeru, centroidi se „neprimetno pomeraju“ (SciKit tolerancija, odnosno prag je u podrazumevanom stanju 0,0001).
- Nakon što je klasterovanje završeno, potrebno je da zabeležimo centroide i identitet klastera.
- Svaki novi dokument se vektorizuje i poredi sa svim centroidima.
- Centroid sa najmanjim rastojanjem od novog vektora posta pripada klasteru koji ćemo dodeliti novom postu.

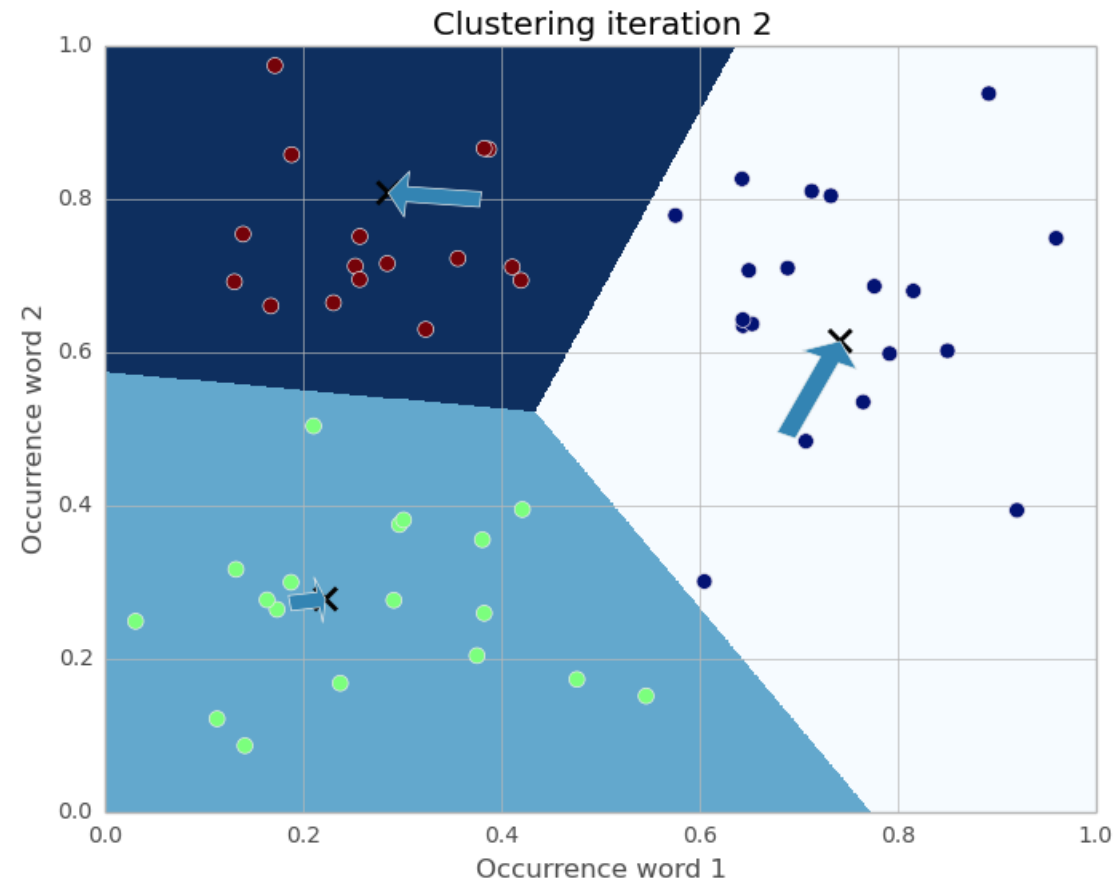
Jednostavan primer.



Jednostavan primer.



Jednostavan primer.



O skupu podataka.

- Skup podataka 20newsgroup je jedan od „standardnih“ skupova podataka u oblasti mašinskog učenja koji se često koristi za ilustrovanje klaster analize (slično kao što se iris koristi za ilustrovanje klasifikacije).
- Sadrži 18.826 postova iz 20 različitih grupa vesti (engl. *newsgroup*), od kojih su neke tehničkog karaktera (comp.sys.mac.hardware, sci.crypt i slične), dok se neke odnose na politiku ili religiju. U našem eksperimentu ograničićemo se na tehničke.
- Pretpostavićemo da je svaka grupa jedan klaster i ispitati da li naš pristup pronalaženju povezanih postova radi.

* Za detalje o skupu podataka i lokacijama za preuzimanje, v. knjigu str. 70.

Uvoz skupa.

- Modul `sklearn.datasets` sadrži funkciju `fetch_20newsgroups` koja automatski uvozi ove podatke:

```
>>> import sklearn.datasets
>>> all_data = sklearn.datasets.fetch_20newsgroups(subset='all')
>>> print(len(all_data.filenames))
18846
>>> print(all_data.target_names)
['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc',
'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware',
'comp.windows.x', 'misc.forsale', 'rec.autos', 'rec.motorcycles',
'rec.sport.baseball', 'rec.sport.hockey', 'sci.crypt',
'sci.electronics', 'sci.med', 'sci.space', 'soc.religion.christian',
'talk.politics.guns', 'talk.politics.mideast', 'talk.politics.misc',
'talk.religion.misc']
```

Uvoz skupa.

- Ukoliko želimo da uvezemo odvojeno obučavajući i test skup, radimo sledeće:

```
>>> train_data = sklearn.datasets.fetch_20newsgroups(subset='train', categories=groups)
```

```
>>> print(len(train_data filenames))
```

```
11314
```

```
>>> test_data = sklearn.datasets.fetch_20newsgroups(subset='test')
```

```
>>> print(len(test_data filenames))
```

```
7532
```

Uvoz skupa.

- Ograničimo se na određeni broj grupa vesti korišćenjem parametra `categories` na sledeći način:

```
>>> groups = ['comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware',  
'comp.sys.mac.hardware', 'comp.windows.x', 'sci.space']  
>>> train_data = sklearn.datasets.fetch_20newsgroups(subset='train', categories=groups)  
>>> print(len(train_data filenames))  
3529  
>>> test_data = sklearn.datasets.fetch_20newsgroups(subset='test', categories=groups)  
>>> print(len(test_data filenames))  
2349
```

Kreiranje vektora i uklanjanje šuma.

- Jednu stvar uvek trebamo da uzmemo u obzir – stvarni podaci su zašumljeni.
- Skup podataka koji koristimo nije izuzetak jer sadrži karaktere koji mogu dovesti do greške tipa `UnicodeDecodeError`.
- Dakle, treba naznačiti da se prilikom vektorizacije ovi karakteri ignorišu.

```
>>> vectorizer = StemmedTfidfVectorizer(min_df=10, max_df=0.5,  
...                                   stop_words='english', decode_error='ignore')  
>>> vectorized = vectorizer.fit_transform(train_data.data)  
>>> num_samples, num_features = vectorized.shape  
>>> print("#samples: %d, #features: %d" % (num_samples, num_features))  
#samples: 3529, #features: 4712
```

Formiranje klastera.

- To znači da smo imamo skup od 3.529 posta i da smo za svaki post formirali vektor od 4.712 obeležja.
- Ovo su ulazni podaci za algoritam K-sredina.
- Postavićemo broj klastera na 50 (za vežbu možete da isprobate druge vrednosti za broj klastera).

```
>>> num_clusters = 50
>>> from sklearn.cluster import KMeans
>>> km = KMeans(n_clusters=num_clusters, init='random', n_init=1, verbose=1, random_state=3)
>>> km.fit(vectorized)
```

Informacije o klasteru.

- Nakon klastera, za svaki vektorizovani post možemo da izdvojimo broj klastera kome pripada (celobrojna vrednost) koristeći `km.labels_`.
- Centroidi su dostupni preko `km.cluster_centers_`.

```
>>> print(km.labels_)  
[48 23 31 ..., 6 2 22]  
>>> print(km.labels_.shape)  
3529
```

- Nove postove možemo da dodeljujemo klasterima koristeći `km.predict`.

Novi post.

- Sve što smo do sada napravili iskoristićemo na sledećem postu koji je dodeljen promenljivoj `new_post`.

*„Disk drive problems. Hi, I have a problem with my hard disk.
After 1 year it is working only sporadically now.
I tried to format it, but now it doesn't boot any more.
Any ideas? Thanks.“*

Vektorizacija i klasterovanje.

- Kao što je rečeno, pre predviđanja klastera moramo da vektorizujemo post.

```
>>> new_post_vec = vectorizer.transform([new_post])  
>>> new_post_label = km.predict(new_post_vec)[0]
```

- Vektor novog posta ne poredimo sa svim postovima, već samo sa onim koji pripadaju istom klasteru. Preuzećemo njihove indekse iz originalnog skupa podataka.

```
>>> similar_indices = (km.labels_==new_post_label).nonzero()[0]
```

- Poređenje u zagradi rezultuje nizom Bulovskih vrednosti, a funkcija `nonzero()` pretvara rezultujući niz u manji niz koji sadrži samo vrednosti True.

Računanje sličnosti (povezanosti) postova.

- Kreiraćemo listu svih postova u klasteru i mere sličnosti postojećih postova sa novim.

```
>>> similar = []
>>> for i in similar_indices:
...     dist = sp.linalg.norm((new_post_vec - vectorized[i]).toarray())
...     similar.append((dist, dataset.data[i]))
>>> similar = sorted(similar)
>>> print(len(similar))
```

131

Prikazivanje relevantnih postova.

- Nakon računanja sličnosti može se prikazati najbliži post (smešten u promenljivu `show_at_1`), a da bi stekli uvid u rezultate računanja sličnosti, prikazaćemo i dva manje slična posta iz istog klastera.

```
>>> show_at_1 = similar[0]
>>> show_at_2 = similar[int(len(similar)/10)]
>>> show_at_3 = similar[int(len(similar)/2)]
```

Prikazivanje relevantnih postova.

- Sličnost posta smeštenog u promenljivu `show_at_1` sa novim postom je 1,038 a sadržaj je:

BOOT PROBLEM with IDE controller

Hi,

I've got a Multi I/O card (IDE controller + serial/parallel interface) and two floppy drives (5 1/4, 3 1/2) and a Quantum ProDrive 80AT connected to it. I was able to format the hard disk, but I could not boot from it. I can boot from drive A: (which disk drive does not matter) but if I remove the disk from drive A and press the reset switch, the LED of drive A: continues to glow, and the hard disk is not accessed at all. I guess this must be a problem of either the Multi I/o card or floppy disk drive settings (jumper configuration?) Does someone have any hint what could be the reason for it. [...]

Prikazivanje relevantnih postova.

- Sličnost posta smeštenog u promenljivu `show_at_2` sa novim postom je 1,150 a sadržaj je:

Booting from B drive

I have a 5 1/4" drive as drive A. How can I make the system boot from my 3 1/2" B drive? (Optimally, the computer would be able to boot: from either A or B, checking them in order for a bootable disk. But: if I have to switch cables around and simply switch the drives so that: it can't boot 5 1/4" disks, that's OK. Also, boot_b won't do the trick for me. [...])

Prikazivanje relevantnih postova.

- Sličnost posta smeštenog u promenljivu `show_at_3` sa novim postom je 1,280 a sadržaj je:

IBM PS/1 vs TEAC FD

Hello, I already tried our national news group without success. I tried to replace a friend s original IBM floppy disk in his PS/1-PC with a normal TEAC drive. I already identified the power supply on pins 3 (5V) and 6 (12V), shorted pin 6 (5.25"/3.5" switch) and inserted pullup resistors (2K2) on pins 8, 26, 28, 30, and 34. The computer doesn't complain about a missing FD, but the FD s light stays on all the time. The drive spins up o.k. when I insert a disk, but I can't access it. The TEAC works fine in a normal PC. Are there any points I missed? [...]

Analiza rezultata.

- Interesantno je kako postovi odražavaju rezultat merenja sličnosti.
- Prvi post sadrži sve istaknute reči iz našeg novog posta.
- Drugi post se takođe odnosi na problem sa pokretanjem, ali se radi o flopi diskovima, a ne čvrstim diskovima.
- Konačno, treći post se ne odnosi ni na problem sa pokretanjem niti na čvrste diskove.
- Međutim, možemo da primetimo da sva tri posta (uključujući i najmanje sličan) pripadaju istom problemskom domenu kao i novi post.

Da li je realno da očekujemo savršeno klasterovanje?

- Ne možemo očekivati savršeno klasterovanje u kontekstu da su postovi iz iste grupe vesti (na primerm, comp.graphics) smešteni u isti klaster.
- Jedan od razloga za to je šum.
- Obratite pažnju na sledeći post:

```
>>> post_group = zip(train_data.data, train_data.target)
>>> all = [(len(post[0]), post[0], train_data.target_names[post[1]]) for post in post_group]
>>> graphics = sorted([post for post in all if post[2]=='comp.graphics'])
>>> print(graphics[5])
(245, 'From: SITUNAYA@IBM3090.BHAM.AC.UK\nSubject:
test....(sorry)\nOrganization: The University of Birmingham, United
Kingdom\nLines: 1\nNNTP-Posting-Host: ibm3090.bham.ac.uk<...snip...>',
'comp.graphics')
```

Da li postoje pokazatelji da ovaj post pripada grupi u kojoj se nalazi?

- Nakon predprocesiranja (tokenizacija, uklanjanje stop reči i svođenje na mala slova) dobićemo reči koje nisu dovoljan pokazatelj da dati post pripada grupi comp.graphics.

```
>>> noise_post = graphics[5][1]
>>> analyzer = vectorizer.build_analyzer()
>>> print(list(analyzer(noise_post)))
['situnaya', 'ibm3090', 'bham', 'ac', 'uk', 'subject', 'test',
'sorri', 'organ', 'univers', 'birmingham', 'unit', 'kingdom', 'line',
'nntp', 'post', 'host', 'ibm3090', 'bham', 'ac', 'uk']
```

- Ukoliko uvedemo dodatno filtriranje pomoću parametara `min_df` i `max_df`, situacija postaje još gora jer dobijamo sledeće reči: „ac“, „birmingham“, „host“, „kingdom“, „nntp“, „sorri“, „test“, „uk“, „unit“ i „univers“.

1. Proces obrade kolekcije jezičkih dokumenata nije trivijalan. U našem slučaju predprocesiranje podataka se svodi na konverziju tekstualnih dokumenata u numeričke vektore pogodne za klasterovanje i merenje sličnosti.
 2. Klaster analiza je pokazala da se vektorizovani postovi iz istog problemskog domena (uglavnom) mogu grupisati u iste klastere, iz kojih se merenjem sličnosti mogu izdvojiti postovi povezani sa novim.
 3. Šum u podacima može negativno da utiče na performanse klasterovanja.
- Eksperimentišite sa brojem klastera, izborom početnih centroida, pristupima klasterovanju zasnovanim na drugim metrikama!

- Beleške pripremljene prema knjizi – Luis Pedro Coelho, Willi Richert (2015): „Building Machine Learning Systems with Python, Second Edition“. Packt Publishing.
- Dodatni material korišćen u pripremi – Milan Milosavljević (2018): beleške sa predavanja iz predmeta mašinsko učenje, Univerzitet Singidunum.

Pitanja su dobrodošla.