Curs 8. Lucrul cu date de tip text

Nota: prezentarea si codul sun conform Introduction to Machine Learning with Python: A Guide for Data Scientists (https://www.bookdepository.com/Introduction-Machine-Learning-with-Python-Andreas-C-Mueller-Sarah-Guido/9781449369415)

O clasa aparte de probleme este cea data de procesarea de informatie de tip text. Exemple de probleme:

- clasificarea mailului ca fiind de tip spam sau legitim
- analiza sentimentelor pe baza elementelor scrise
- regasirea de texte similare

Exista urmatoarele 4 categorii de date text:

1. date categoriale
2. text liber reductibil la date categoriale
3. text structurat
4. text liber

Datele categoriale rpovin dinstr-o lista predefinita (ex: genul unei persoane, culoarea unei masini). Totusi, in functie de modul de prelaure a valorii de la utilizator, este inca posibil sa apara erori de scriere: blak in loc de black, sau particularitati de scriere: gray/grey, neighbor/neighbour. Se impune deci o determinare a formelor corecte, de exemplu prin distanta Levenshtein sau algoritmi de spell-checking.

Textul liber reductibil la date categoriale se refera la exprimari care pot fi reduse la date text categoriale: culoara ierbii -> verde, sau definirea unor categorii de ocupatii iar la final "Altele"; un exemplu bun este Color Survey Results (https://blog.xkcd.com/2010/05/03/color-survey-results/).

Textul structurat poate fi exemplificat prin documente XML sau fisiere JSON, care respecta o anumita sintaxa si eventual o schema. Procesarea lor este un subiect separat.

Ultima categorie este data de fisiere email, carti, tweets, scrisori de intentie etc. Procesarea lor intra in zona Natural Language Processing (NLP) si information retrieval (IR). Un exemplu este determinarea plagiatelor, prin care se detecteaza preluari masive din alte surse text, sau atribuirea autorului - pentru o bucata de text se cere determinarea autorului cel mai probabil.

Sursele de date sunt diverse:

- paginile Wikipedia
- cartile din proiectul Gutenberg (http://www.gutenberg.org/), la ora actuaal 56000 ebooks
- mesaje newgroups (http://kdd.ics.uci.edu/databases/20newsgroups/20newsgroups.html)
- ziare (http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html)

Exemplu de aplicatie: analiza sentimentor din recenzii de filme

Exista un set de recenzii de filme facute de catre utilizator, disponibil aici (http://ai.stanford.edu/~amaas/data/sentiment/). Pentru fiecare recenzie exista o valoare numerica intre 1 si 10, reprezentand o indicatie: daca este un review pozitiv sau negativ. Setul de date este impartit in subset de antrenare si de testare, iar pentru fiecare subset se gasesc doua directoare, respectiv pentru aprecieri pozitive si negative.

In [1]:
```
!dir "data/aclImdb"
```

```
 Volume in drive D is data
 Volume Serial Number is 503D-AB7C

 Directory of D:\work\school\cursuri\cursuri_github\cursuri\Introducere_in_da
ta_science\cursuri\Curs8\data\aclImdb

06/26/2011  04:08 AM    <DIR>          .
06/26/2011  04:08 AM    <DIR>          ..
04/12/2011  08:14 PM           845,980 imdb.vocab
06/12/2011  01:54 AM           903,029 imdbEr.txt
06/26/2011  03:18 AM             4,037 README
04/12/2011  08:22 PM    <DIR>          test
06/26/2011  04:09 AM    <DIR>          train
               3 File(s)      1,753,046 bytes
               4 Dir(s)  1,542,084,423,680 bytes free
```

In [2]:
```
path = './data/aclImdb/'
path_train = path + 'train/'
path_test = path + 'test/'
```

In [3]:
```
from sklearn.datasets import load_files
```

In [4]:
```
reviews_train = load_files(path_train)
```

In [5]:
```
text_train, y_train = reviews_train.data, reviews_train.target
print('How are texts organized: ', type(text_train))
print('How many texts in train subset', len(text_train))
print('The first text: ', text_train[0])
print('Associated review:', y_train[0])
```

```
How are texts organized:  <class 'list'>
How many texts in train subset 75000
The first text:  b'Full of (then) unknown actors TSF is a great big cuddly ro
mp of a film.<br /><br />The idea of a bunch of bored teenagers ripping off t
he local sink factory is odd enough, but add in the black humour that Forsyth
& Co are so good at and your in for a real treat.<br /><br />The comatose van
driver by itself worth seeing, and the canal side chase is just too real to b
e anything but funny.<br /><br />And for anyone who lived in Glasgow it\'s a
great "Oh I know where that is" film.'
Associated review: 2
```

In [6]:
```
reviews_test = load_files(path_test)
```

In [7]:
```
text_test, y_test = reviews_test.data, reviews_test.target
print('How are texts organized: ', type(text_test))
print('How many texts in train subset', len(text_test))
print('The first text: ', text_test[0])
print('Associated review:', y_test[0])
```

```
How are texts organized:  <class 'list'>
How many texts in train subset 25000
The first text:  b"Don't hate Heather Graham because she's beautiful, hate he
r because she's fun to watch in this movie. Like the hip clothing and funky s
urroundings, the actors in this flick work well together. Casey Affleck is hy
sterical and Heather Graham literally lights up the screen. The minor charact
ers - Goran Visnjic {sigh} and Patricia Velazquez are as TALENTED as they are
gorgeous. Congratulations Miramax & Director Lisa Krueger!"
Associated review: 1
```

Se remarca existenta elementului html `<br />` ce poate fi inlaturat, fara a afecta continutul mesajului:

In [8]:
```
text_train = [text.replace(b'<br />', b' ') for text in text_train]
text_test = [text.replace(b'<br />', b' ') for text in text_test]
```

Numarul de elemente din fiecare clasa:

In [9]:
```
import numpy as np
print('Classes in train set: ', np.unique(y_train))
print('Classes in test set: ', np.unique(y_test))
```

```
Classes in train set:  [0 1 2]
Classes in test set:  [0 1]
```

In [10]:
```
#Filtering out items of class "2" in train set
text_train = [text_train[i] for i in range(len(y_train)) if y_train[i] < 2 ]
y_train = [y_train[i] for i in range(len(y_train)) if y_train[i] < 2 ]
```

In [11]:
```
print('Samples per class in training set: {0}'.format(np.bincount(y_train)))
print('Samples per class in test set: {0}'.format(np.bincount(y_test)))
```

```
Samples per class in training set: [12500 12500]
Samples per class in test set: [12500 12500]
```

Reprezentarea textului sub forma bag of words

Pentru un text, reprezenatrea bag of word se obtine astfel: se pleaca de la un vocabular = multimea tuturor cuvintelor care pot aparea in texte - si se contorizeaza pentru fiecare cuvant din dictionar de cat ori apare in text. Daca vocabularul are N cuvinte distincte, atunci pentru fiecare document rezulta un vector de N componente. Majoritatea componentelor din vectorul asociat unui document va fi zeri, deci avem de-a face cu vectori rari.

Pasii pentru obtinerea unei reprezentari bag of words sunt:

1. despartirea in cuvuinte (tokenizing)
2. construirea dictionarului
3. construirea documentului bag of word pentru document

Exemplu de obtinere de bag of words pe un text simplu

In [12]:
```
bards_words =[
    "The fool doth think he is wise,",
    "but the wise man knows himself to be a fool"]
```

Clasa `CountVectorizer` din `sklearn.feature_extraction.text` serveste la selectarea cuvintelor din text si calculul frecventei de aparitie:

In [13]:
```
from sklearn.feature_extraction.text import CountVectorizer
vect = CountVectorizer()
vect.fit(bards_words)
print(vect.vocabulary_)
```

```
{'the': 9, 'fool': 3, 'doth': 2, 'think': 10, 'he': 4, 'is': 6, 'wise': 12,
'but': 1, 'man': 8, 'knows': 7, 'himself': 5, 'to': 11, 'be': 0}
```

Obtinerea unui vector bag of words se obtine prin aplicarea metodei `transform`:

In [14]:
```
bow = vect.transform(bards_words)
```

In [15]:
```
print('Reprezentarea ca vectori rari:', bow)
print('Reprezentarea ca vectori:\n', bow.toarray())
```

```
Reprezentarea ca vectori rari:   (0, 2)	1
  (0, 3)	1
  (0, 4)	1
  (0, 6)	1
  (0, 9)	1
  (0, 10)	1
  (0, 12)	1
  (1, 0)	1
  (1, 1)	1
  (1, 3)	1
  (1, 5)	1
  (1, 7)	1
  (1, 8)	1
  (1, 9)	1
  (1, 11)	1
  (1, 12)	1
Reprezentarea ca vectori:
 [[0 0 1 1 1 0 1 0 0 1 1 0 1]
 [1 1 0 1 0 1 0 1 1 1 0 1 1]]
```

Transformarea celor doua subseturi de date in BOW

In [16]:
```
vect = CountVectorizer()
X_train = vect.fit_transform(text_train)
print(repr(X_train))
X_test = vect.transform(text_test)
```

```
<25000x74849 sparse matrix of type '<class 'numpy.int64'>'
	with 3431196 stored elements in Compressed Sparse Row format>
```

In [17]:
```
#obtinerea vocabularului
feature_names = vect.get_feature_names()
print('Dimensiunea vocabularului:', len(feature_names))
```

```
Dimensiunea vocabularului: 74849
```

In [18]:
```
print(feature_names[:20])
```

```
['00', '000', '0000000000001', '00001', '00015', '000s', '001', '003830', '00
6', '007', '0079', '0080', '0083', '0093638', '00am', '00pm', '00s', '01', '0
1pm', '02']
```

In [19]:
```
print(feature_names[-20:])
```

```
['är', 'ääliöt', 'äänekoski', 'åge', 'åmål', 'æsthetic', 'écran', 'élan', 'ém
igré', 'émigrés', 'était', 'état', 'étc', 'évery', 'êxtase', 'ís', 'ísnt', 'ø
stbye', 'über', 'üvegtigris']
```

```
In [20]: print(feature_names[20000:20020])
```

```
['draper', 'draperies', 'drapery', 'drapes', 'draskovic', 'drastic', 'drastic
ally', 'drat', 'dratch', 'dratic', 'dratted', 'draub', 'draught', 'draughts',
'draughtswoman', 'draw', 'drawback', 'drawbacks', 'drawer', 'drawers']
```

Antrenarea si evaluarea unui model logistic regression

```
In [21]: from sklearn.model_selection import cross_val_score
         from sklearn.linear_model import LogisticRegression
```

```
In [22]: scores = cross_val_score(LogisticRegression(), X_train, y_train, cv = 5, n_job
         s=4)
         print('Acuratetea medie de clasificare: {0}'.format(np.mean(scores)))
```

```
Acuratetea medie de clasificare: 0.8819600000000001
```

Aplicare de cross validation pentru parametrul de regularizare

Exista un parametru de regularizare care determina penalizarea L_2 a ponderilor modelului. Efectum cautarea valorii potrivite a acestui hiperparametru, al carui nume de argument este `C`:

```
In [23]: from sklearn.model_selection import GridSearchCV
```

```
In [24]: param_grid = {'C':[0.001, 0.01, 0.1, 1, 10]}
         grid = GridSearchCV(LogisticRegression(), param_grid=param_grid, cv=5, n_jobs=
         4)
         grid.fit(X_train, y_train)
         print('best cross validation score:', grid.best_score_)
         print('best params:', grid.best_params_)
```

```
best cross validation score: 0.88812
best params: {'C': 0.1}
```

```
In [25]: print(grid.score(X_test, y_test))
```

```
0.87892
```

Modificarea parametrilor pentru obtinerea BOW

Putem cere retinerea doar a acelor cuvinte care apar intr-un numar minim de documente, de ex 5. In acest fel speram ca se elimina cuvintele care nu sunt larg partajate.

In [26]:
```
vect = CountVectorizer(min_df=5)
X_train = vect.fit_transform(text_train)
X_test = vect.transform(text_test)
print('Dimensiune vocabular: ', len(vect.get_feature_names()))
```

```
Dimensiune vocabular:  27271
```

In [27]:
```
grid = GridSearchCV(LogisticRegression(), param_grid=param_grid, cv=5, n_jobs=
4)
grid.fit(X_train, y_train)
print('best cross validation score:', grid.best_score_)
print('best params:', grid.best_params_)
```

```
best cross validation score: 0.8874
best params: {'C': 0.1}
```

In [28]:
```
print(grid.score(X_test, y_test))
```

```
0.87784
```

Eliminare cuvinte neinformative

Exista cuvinte care nu poarta multa informatie: prepozitii, conjunctii, sau cuvinte care se repeta des in toate documentele. Exista doua posibiliatti de eliminare a acestora:

1. Se foloseste o lista predefiita de stopwords, specifica limbii respective
2. Se estimeaza valoarea informationala a cuvintelor

Eliminarea de stopwords

Pentru limba engleza sunt definite stopwords in sklearn.feature_extraction.text:

In [29]:
```
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS
print(len(ENGLISH_STOP_WORDS))
```

```
318
```

In [30]:
```
print(list(ENGLISH_STOP_WORDS)[:20])
```

```
['itself', 'nevertheless', 'five', 'others', 'sometimes', 'also', 'thin', 'wh
at', 'than', 'sometime', 'formerly', 'themselves', 'thereafter', 'never', 'st
ill', 'whenever', 'because', 'anywhere', 'me', 'please']
```

In [31]:
```
vect = CountVectorizer(min_df=5, stop_words='english')
X_train = vect.fit_transform(text_train)
X_test = vect.transform(text_test)
print('Dimensiune vocabular: ', len(vect.get_feature_names()))
```

```
Dimensiune vocabular:  26966
```

In [32]:
```
grid = GridSearchCV(LogisticRegression(), param_grid=param_grid, cv=5, n_jobs=
4)
grid.fit(X_train, y_train)
print('best cross validation score:', grid.best_score_)
print('best params:', grid.best_params_)
```

```
best cross validation score: 0.88364
best params: {'C': 0.1}
```

In [33]:
```
print(grid.score(X_test, y_test))
```

```
0.87252
```

Calculul valorii tf-df

Term frequency - inverse document frequency da pondere unui cuvant care apare frecvent intr-un document, dar nu in multe documente - deci apre putere descriptiva pentru documentul in care apare frecvent. Pentru un document d si un cuvant w, valoarea tfidf se calculeaza ca:

$$tfidf(w,d) = tf(w,d) \cdot \log\left(\frac{N+1}{N_w+1}\right) + 1$$

unde: $tf(w,d)$ este numarul de aparitii ale lui w in docuemntul d, N e numarul total de documente din corpus, N_w este numarul de documente din corpus care contin cuvantul w. Pentru fiecare document d se calculeaza un astfel e de vector, care in final este normalizat in norma L_2 la 1.

In [35]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import make_pipeline
pipe = make_pipeline(TfidfVectorizer(min_df=5, norm=None), LogisticRegression
())
param_grid = {'logisticregression__C':[0.001, 0.01, 0.1, 1, 10]}

grid = GridSearchCV(pipe, param_grid=param_grid, cv=5, n_jobs=4)
grid.fit(text_train, y_train)
print('Best cross validation score:', grid.best_score_)
```

```
Best cross validation score: 0.89416
```

In [36]:
```
grid.score(text_test, y_test)
```

Out[36]:
```
0.88504000000000005
```

Utilizarea de n-grams

BOW pierde succesiunea cuvintelor. Se pot considera toate succesiunile de n cuvinte - n-grams.

In [37]:
```
print(bards_words)
```
```
['The fool doth think he is wise,', 'but the wise man knows himself to be a f
ool']
```

In [39]:
```
cv = CountVectorizer(ngram_range=(2, 2)).fit(bards_words)
cv.get_feature_names()
```
Out[39]:
```
['be fool',
 'but the',
 'doth think',
 'fool doth',
 'he is',
 'himself to',
 'is wise',
 'knows himself',
 'man knows',
 'the fool',
 'the wise',
 'think he',
 'to be',
 'wise man']
```

In [44]:
```
pipe = make_pipeline(TfidfVectorizer(min_df=5), LogisticRegression())
param_grid = {"logisticregression__C": [0.001, 0.01, 0.1, 1, 10, 100],
"tfidfvectorizer__ngram_range": [(1, 1), (1, 2), (1, 3)]}
grid = GridSearchCV(pipe, param_grid, cv=5, n_jobs=4)
```

In [ ]:
```
grid.fit(text_train, y_train)
```

In [ ]:
```
print("Best cross-validation score: {:.2f}".format(grid.best_score_))
print("Best parameters:\n{}".format(grid.best_params_))
```

In [ ]:
```
grid.score(text_test, y_test)
```