Curs 8. Lucrul cu date de tip text

Nota: prezentarea si codul sunt 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

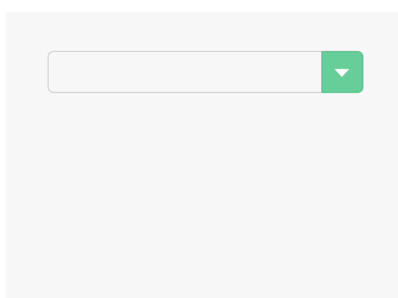
Datele categoriale provin dintr-o lista predefinita (ex: genul unei persoane, culoarea unei masini):

- "red", "green", "blue", "yellow", "black", "white", "purple", "pink"; "multicolo(u)red", "other"
- "male", "female", "not declared".

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 (https://en.wikipedia.org/wiki/Levenshtein_distance) sau algoritmi de spell-checking (https://norvig.com/spell-correct.html) - atentie la dialectul ales. In alte cazuri este nevoie de reducerea dictionarului, pentru notiuni care sunt foarte similare sau identice:

- "culoarea ierbii" -> "verde",
- definirea unor categorii de ocupatii iar la final "Altele"
- Colours (http://www.thedoghousediaries.com/1406)

In alte cazuri, exprimari libere (texte de dimensiuni medii) pot fi reduse la **categorii cunoscute**. Pentru simplificarea procesarii, se recomanda evitarea de introducere de text liber si sa se permita alegerea dintr-o multime predefinita de valori:



Textul structurat poate fi exemplificat prin documente XML sau fisiere JSON, care respecta o anumita sintaxa si eventual o schema. De exemplu, datele de contact au o structura (adresa fizica, persoana de contact, telefon), interpretarea lor facandu-se cu cunostinte de domeniu. Procesarea lor este un subiect separat.

Ultima categorie - **textul liber** - este data de documente de intindere mai mare: 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; Character-level and Multi-channel Convolutional Neural Networks for Large-scale Authorship Attribution (https://arxiv.org/abs/1609.06686); detectarea autorului unui cod compilat Even Anonymous Coders Leave Fingerprints (https://www.wired.com/story/machine-learning-identify-anonymous-code/), articol la When Coding Style Survives Compilation: De-anonymizing Programmers from Executable Binaries (https://arxiv.org/pdf/1512.08546.pdf).

Sursele de date sunt diverse:

- pagini Wikipedia
- cartile din proiectul Gutenberg (http://www.gutenberg.org/), la ora (aprilie 2020) peste 60000 de ebooks
- mesaje newgroups (http://kdd.ics.uci.edu/databases/20newsgroups/20newsgroups.html)
- ziare (http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html)
- Alphabetical list of free/public domain datasets with text data for use in Natural Language Processing (NLP) (https://github.com/niderhoff/nlp-datasets)
- IMDB Movie Review Sentiment Classification (stanford) (http://ai.stanford.edu/~amaas/data/sentiment/)
- Movie Review Data (http://www.cs.cornell.edu/people/pabo/movie-review-data/)
- Resursa 1 (https://stats.stackexchange.com/questions/18905/where-to-find-a-large-text-corpus), Colectia 2 (https://lionbridge.ai/datasets/the-best-25-datasets-for-natural-language-processing/), Colectia 3 (https://blog.cambridgespark.com/50-free-machine-learning-datasets-natural-language-processing-

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 (rating > 5) si negative (rating <= 5).

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

```
 Volume in drive C is system
 Volume Serial Number is DEBB-6518

 Directory of c:\temp\Curs8\data\aclImdb

04/13/2020  07:51 PM    <DIR>          .
04/13/2020  07:51 PM    <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
04/13/2020  07:49 PM    <DIR>          test
04/13/2020  07:51 PM    <DIR>          train
               3 File(s)      1,753,046 bytes
               4 Dir(s)  64,399,880,192 bytes free
```

In [2]:
```
!tree "data/aclImdb" /A
```

```
Folder PATH listing for volume system
Volume serial number is DEBB-6518
C:\TEMP\CURS8\DATA\ACLIMDB
+---test
|   +---neg
|   \---pos
\---train
    +---neg
    +---pos
    \---unsup
```

In [3]:
```
import os
path = './data/aclImdb/'
path_train = os.path.join(path, 'train')
path_test = os.path.join(path, 'test')
```

In [4]:
```
from sklearn.datasets import load_files
```

In [5]:
```
####### !!!16 mins running time!!!
reviews_train = load_files(path_train)
```

In [6]:
```
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('A text: ', text_train[50100])
print('Associated review:', y_train[50100])
```

```
How are texts organized:  <class 'list'>
How many texts in train subset 75000
A text:  b"Having watched all of the Star Trek TV series episodes many times
each since the 1960s, most being quite good to superb, and only very few bein
g mediocre, my opinion is that this one is the worst of all.<br /><br />In fa
ct, I think it's so poorly executed as to be an embarrassment to the series.
It's not that the story is so bad, although it's not particularly outstanding
in any way, but the acting is just abysmal on the part of the two lead charac
ters, meaning those other than the regulars in this case. Barbara Anderson gi
ves her weakest performance ever as the daughter of a mass killer, and who is
on a mission of a sort. She practically calls in the role from a phone, and s
hows no real emotive abilities here. Although usually she's never used as mor
e than a pretty face in most of her film/TV roles,usually small parts, she ha
s done much better.<br /><br />Arnold Moss as her father gives new meaning to
the term 'Ham' and is the only actor ever on a 1960s Star Trek episode that o
utdid William Shatner in this area, and actually makes Shatner look superb by
comparison. And he gets to play a Shakespearian actor no less, which gives hi
m more impetus to overact, and he does so.<br /><br />Other than these two le
ads being so weak, the story is such that anybody with any sense at all can t
ell who the killer is within the first 15 minutes. I say this because I told
my brother the whole plot ending at the first commercial break when we were w
atching the original 1966 broadcast as pre-teens. His reply was, Yeah, you're
right.<br /><br />Skip this one and watch the much superior Menagerie episode
s which were originally televised right before."
Associated review: 2
```

In [7]:
```
reviews_test = load_files(path_test)
```

```
In [8]: 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[70])
        print('Associated review:', y_test[70])
```

```
How are texts organized:  <class 'list'>
How many texts in train subset 25000
The first text:  b'A great gangster flick, with brilliant performances by wel
l-known actors with great action scenes? Well, not this one.<br /><br />It\'s
rather amazing to see such a wide cast of well-known actors, that have many g
ood movies in their filmographies in such a movie, without doubt this may be
one of the worst they could possibly appear in.<br /><br />First of all, the
plot is as you\'d expect it from your average gangster biography, nothing ne
w, nothing fancy in it. The way it is told makes the movie look a LOT longer
than it is (when i thought the two hours should be almost over, i was quite s
urprised that only 45 minutes had passed).<br /><br />The action scenes look
a lot like those from 80ies TV series - the A-Team, for example. It\'s just t
hat in the 80ies (esp. with the A-Team) those scenes were far more sophistica
ted than those in "El Padrino". It\'s especially fun to see the guys point th
eir guns in the air and still hit something (not to talk about people that ta
ke cover behind car doors which later look like they\'ve been shot through).<
br /><br />The acting fits quite nicely to the action. Either you get the sam
e reaction to everything that happens (Dolph Lundgren style), or it\'s so ove
racted that you may think it\'s a parody (but unfortunately it\'s not).<br />
<br />My advise is to stay away from this movie, any other gangster movie is
better than this one.'
Associated review: 0
```

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

```
In [9]: 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 [10]: 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 [11]: #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 [12]: 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)))
```

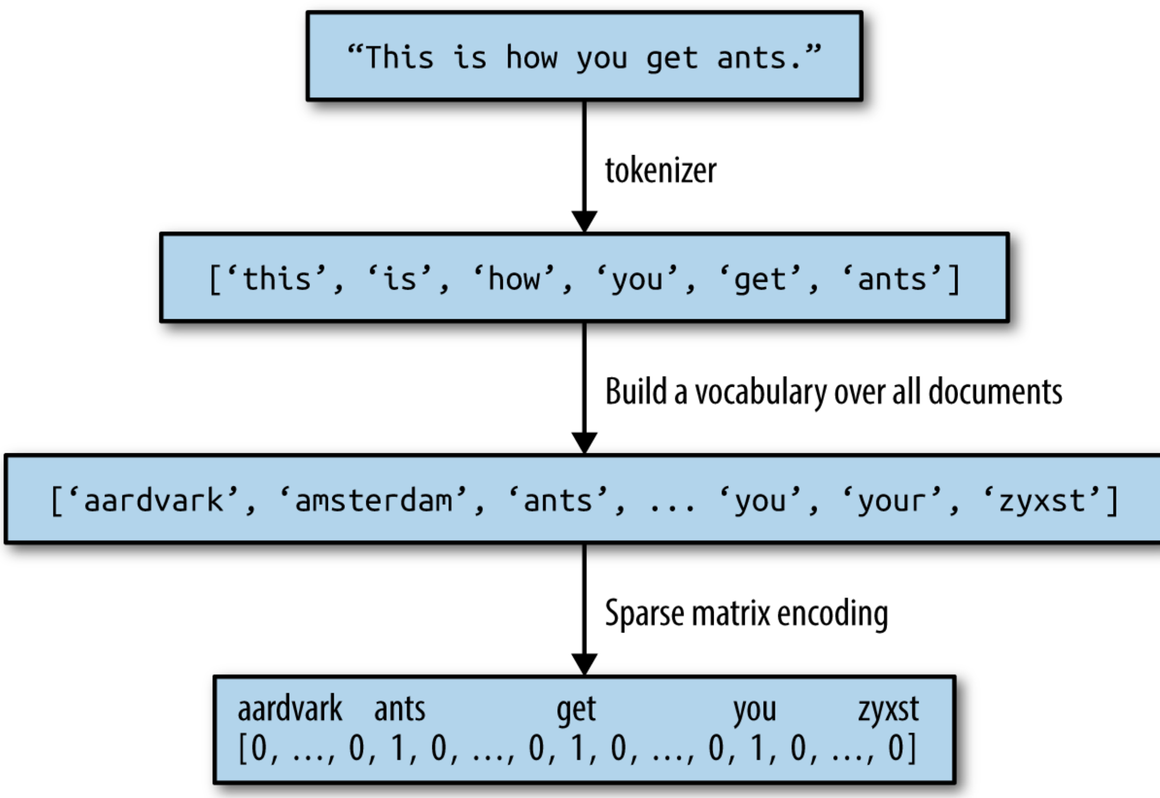
```
Samples per class in training set: [12500 12500]
Samples per class in test set: [12500 12500]
```

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 k cuvinte distincte, atunci pentru fiecare document rezulta un vector de k componente. Majoritatea componentelor din vectorul asociat unui document va fi zero, deci avem de-a face cu vectori rari.

Pasii pentru obtinerea unei reprezentari bag of words sunt:

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



Exemplu de obtinere de bag of words pe un text simplu

In [13]:
```
bards_words =[
    "The fool doth think he is wise,",
    "but the wise man knows himself to be fool"]
```

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

In [14]:
```
from sklearn.feature_extraction.text import CountVectorizer
vect = CountVectorizer()
vect.fit(bards_words)
print(vect.vocabulary_) # cuvintele distincte din texte

# vocabular sortat in ordine crescatoare
print(sorted(vect.vocabulary_.items(), key = lambda item: item[1]))
```

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

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

In [15]:
```
bow = vect.transform(bards_words)
```

In [16]:
```
print(f'Reprezentarea ca vectori rari\n:\b{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 [17]:
```
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 [18]:
```
print(f'Dimensiune vocabular: {len(vect.vocabulary_)}')
```

```
Dimensiune vocabular: 74849
```

In [19]:
```
#obtinerea vocabularului
feature_names = vect.get_feature_names()
print('Dimensiunea vocabularului:', len(feature_names))
```

```
Dimensiunea vocabularului: 74849
```

In [20]:
```
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 [21]:
```
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 [22]:
```
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 de logistic regression

In [23]:
```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
```

In [24]:
```
scores = cross_val_score(LogisticRegression(), X_train, y_train, cv = 5, n_job
s=4)
print('Acuratetea medie de clasificare pe setul de instruire: {0}'.format(np.m
ean(scores)))
```

```
Acuratetea medie de clasificare pe setul de instruire: 0.88272
```

Aplicare de cross validation pentru parametrul de regularizare

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

In [25]:
```
from sklearn.model_selection import GridSearchCV
```

In [26]:
```
# param_grid = {'C':[0.001, 0.01, 0.1, 1, 10]}
param_grid = {'C':[0.01, 0.1, 1]}
grid = GridSearchCV(LogisticRegression(solver='lbfgs', max_iter=1000), 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: 0.8881599999999998
best params: {'C': 0.1}
```

In [27]:
```
print(grid.score(X_test, y_test))
```

```
0.87884
```

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 [28]:
```
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 [29]:
```
grid = GridSearchCV(LogisticRegression(solver='lbfgs', max_iter=1000), 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: 0.88744
best params: {'C': 0.1}
```

In [30]:
```
print(grid.score(X_test, y_test))
```

```
0.8778
```

Eliminare cuvinte neinformative (stopwords)

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

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

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

In [31]:
```
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS
print(len(ENGLISH_STOP_WORDS))
```

```
318
```

In [32]:
```
print(list(ENGLISH_STOP_WORDS)[:20])
```

```
['get', 'anywhere', 'must', 'seeming', 'across', 'their', 'about', 'whereafte
r', 'nevertheless', 'next', 'hers', 'seems', 'we', 'noone', 'itself', 'eigh
t', 'are', 'above', 'be', 'cant']
```

In [33]:
```
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 [34]:
```
grid = GridSearchCV(LogisticRegression(solver='lbfgs', max_iter=1000), param_g
rid=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: 0.8837200000000001
best params: {'C': 0.1}
```

In [35]:
```
print(grid.score(X_test, y_test))
```

```
0.87256
```

Calculul valorii tf-idf

Term frequency - inverse document frequency da pondere unui cuvant care apare frecvent intr-un document, dar nu in multe documente - deci are 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 document 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 de vector, care in final este normalizat in norma L_2 la 1.

In [36]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import make_pipeline
pipe = make_pipeline(TfidfVectorizer(min_df=5, norm='l2'),
                     LogisticRegression(solver='lbfgs', max_iter=1000))
# param_grid = {'logisticregression__C':[0.001, 0.01, 0.1, 1, 10]}
param_grid = {'logisticregression__C':[0.01, 0.1, 1]}

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: 0.8886
```

In [37]:
```
grid.score(text_test, y_test)
```

Out[37]:
```
0.8832
```

Utilizarea de n-grams

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

In [38]:
```
print(bards_words)
```

```
['The fool doth think he is wise,', 'but the wise man knows himself to be foo
l']
```

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 [40]:
```
pipe = make_pipeline(TfidfVectorizer(min_df=5), LogisticRegression(solver='lbf
gs', max_iter=1000))
# param_grid = {"logisticregression__C": [0.001, 0.01, 0.1, 1, 10, 100],
param_grid = {"logisticregression__C": [0.01, 0.1, 1, 10],
"tfidfvectorizer__ngram_range": [(1, 1), (1, 2), (1, 3)]}
grid = GridSearchCV(pipe, param_grid, cv=5, n_jobs=4)
```

In [41]:
```
grid.fit(text_train, y_train)
```

Out[41]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('tfidfvectorizer',
                                        TfidfVectorizer(analyzer='word',
                                                        binary=False,
                                                        decode_error='stric
t',
                                                        dtype=<class 'numpy.f
loat64'>,
                                                        encoding='utf-8',
                                                        input='content',
                                                        lowercase=True,
                                                        max_df=1.0,
                                                        max_features=None,
                                                        min_df=5,
                                                        ngram_range=(1, 1),
                                                        norm='l2',
                                                        preprocessor=None,
                                                        smooth_idf=True,
                                                        stop_words=None...
                                                           max_iter=1000,
                                                           multi_class='aut
o',
                                                           n_jobs=None,
                                                           penalty='l2',
                                                           random_state=None,
                                                           solver='lbfgs',
                                                           tol=0.0001,
                                                           verbose=0,
                                                           warm_start=Fals
e))],
                                verbose=False),
             iid='deprecated', n_jobs=4,
             param_grid={'logisticregression__C': [0.01, 0.1, 1, 10],
                         'tfidfvectorizer__ngram_range': [(1, 1), (1, 2),
                                                          (1, 3)]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [42]:
```
print("Best cross-validation score: {:.2f}".format(grid.best_score_))
print("Best parameters:\n{}".format(grid.best_params_))
```

```
Best cross-validation score: 0.91
Best parameters:
{'logisticregression__C': 10, 'tfidfvectorizer__ngram_range': (1, 3)}
```

In [43]:
```
grid.score(text_train, y_train)
```

Out[43]:
```
0.99928
```

In [44]:
```
grid.score(text_test, y_test)
```

Out[44]:
```
0.90364
```

Alte modalitati de procesare a textelor

- Word2Vec: King - Man + Woman = Queen: The Marvelous Mathematics of Computational Linguistics (https://www.technologyreview.com/s/541356/king-man-woman-queen-the-marvelous-mathematics-of-computational-linguistics/); A Beginner's Guide to Word2Vec and Neural Word Embeddings (https://skymind.ai/wiki/word2vec); Understanding Word Embeddings (https://hackernoon.com/understanding-word-embeddings-a9ff830403ce)
- Latent Dirichlet allocation (https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation)

```
In [ ]:
```