

Romanian sub-dialect identification

1 Descriere si preprocesare a datelor

Datele date constau in 5 fisiere: train_samples.txt, validation_samples.txt si test_samples.txt unde pe fiecare linie a acestor fisiere se afla un numar(indice) urmat de un text encrypted, iar in fisierele train_labels.txt si validation_labels.txt se afla pe fiecare linie indexul corespunzator unei propozitii urmat de o valoare, 0 sau 1, 0 fiind reprezentativ pentru dialectul moldovenesc si 1 fiind reprezentativ pentru dialectul din Romania.

Cerinta este sa cream un clasificator, sa il “antrenam” folosind train_samples.txt si train_labels.txt si sa vedem ce eficienta are folosind fisierele validation_samples.txt si validation_labels.txt, iar dupa sa incercam sa prezicem din ce categorii fac parte textele din fisierul test_samples.txt, iar rezultatele sa le salvam intr-un fisier submissions.csv(submissions.txt).

Mai intai salvam intr-o lista fiecare label din train_labels.txt in felul urmator:

```
label_train = [] #in aceasta lista vor fi salvate

for line in document_labels_train:
    for word in line.split()[1]: #scapam de
        label_train += [word]
```

Similar procedam si pentru validation_labels.txt

Dupa,folosind urmatorul procedeu

```
rows_train = []    #in aceasta lista vor fi salvate
toate_cuvinte_train = []    #in aceasta lista vor
index_train = 0
for line in document_train:    #luam fiecare linie
    linie_train = ' '.join(line.split()[1:])    #despa
    linie_train = linie_train.split();    #despa
    rows_train += [[linie_train, label_train[index_train]]
    index_train = index_train + 1
    for cuvint in linie_train:    #fiecare cuvant
        toate_cuvinte_train += [cuvant]
```

construim o lista formata dintr-o alta lista de cuvinte si label-ul corespunzator textului.

Lista formata arata asa:

	Text	Label (0 sau 1)
0	[;%fE, mr#&, crmx, temjc@m, %'wb:, }hHAm@@m, y...	1
1	[sAFW, K#xk}t, fH@ae, m&Xd, >h&, @#, l@Rd}a, @...	1
2	[zgHy%, @kA, qCrw, h@@m, he %WA, Eh}W@m, mkZrm...	1
3	[!ck&, g@eAh, =F:, me, @Hc, Zk&}, mk@eAhH, jmj...	1
4	[zpW, hjreaek, egae, h:, (AvnY, }e, m@p:, Ejfm...	0
...
7752	[Kopv, rSum', EhfZm}a@m, Zh@a%p, (r=*, hZ, mgA...	0
7753	[E%<Hh, \$&FW, m*#, m*#, m*#, m*#H, ga@#&e, }m}...	1
7754	[SZU:\$, r(<@, k:, HHf@, re:@, Hhrfa@m, oEgAa, ...	1
7755	[.ZeEq, jpxdrl, :q, H>w:, n>IAX, c'h\$V, aAx>=,...	1
7756	[Bo=@*(, eglc, gq>, Zh#, e@Aah, B*kiY, D:(sTE,...	1

7757 rows x 2 columns

2 Modele folosite

Metodele pe care le-am folosit sunt modelul bag-of-words si clasificatorul naive-Bayes.

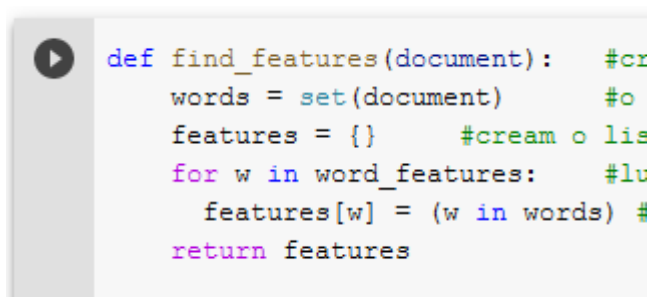
Modelul bag-of-words este o metoda de simplificare a reprezentarii folosita in procesarea limbajului natural. Un text este despartit in unitati(words) si reprezentat pe un multiset(bag),netinandu-se cont de ordinea cuvintelor sau de gramatica,ci doar de multiplicitatea fiecarui cuvant.

Naive-Bayes este un clasificator probabilistic bazat pe teorema lui Bayes. Naive-Bayes reprezinta o tehnica simpla de a construi un clasificator: repartizeaza o clasa unui vector(sau lista) de 'feature_values',unde o clasa poate fi aleasa dintr-o multime finita.

3 Rezultat

In implementarea proiectului am folosit Natural Language Toolkit.

Pentru gasirea a 'feature_values' am folosit urmatoarea functie

```
A screenshot of a code editor showing a Python function named 'find_features'. The function takes a 'document' as input and returns a dictionary of features. The code is as follows:  
def find_features(document):  
    words = set(document)  
    features = {}  
    for w in word_features:  
        features[w] = (w in words)  
    return features
```

unde fiecare cuvant din lista cuvintelor cele mai frecvente este verificat daca apare in cuvintele unei linii de text.Daca apare este marcat ca 'True' daca nu apare este marcat ca 'False'.

Construim o lista formata din 'feature_values' si label-urile corespunzatoare fiecărei linii de text:

```
featuresets_train = [(find_features(d), c) for (d,c) in rows_train]
```

Dupa, construim si antrenam clasificatorul prin metoda naive-Bayes,folosind train_set – ul ,

```
train_set = featuresets_train
test_set = featuresets_test
classifier = nltk.NaiveBayesClassifier.train(train_set)
```

si testam acuratetea predictiilor folosind test_set-ul.

4 Concluzie

In concluzie,dupa antrenarea clasificatorului cu 7757 linii de cuvinte,testam clasificatorul pe 2656 linii de cuvinte si obtinem acuratetea de 0.6027,

```
print(nltk.classify.accuracy(classifier, test_set))
print(classifier.show_most_informative_features())
```

0.6027861445783133

Most Informative Features

ahk#@m = True	0 : 1	=	26.0 : 1.0
Ehrpe@m = True	0 : 1	=	15.0 : 1.0
Ehahk#@m = True	0 : 1	=	14.0 : 1.0
)me@rH@m = True	0 : 1	=	13.8 : 1.0
cZmka@m = True	0 : 1	=	12.0 : 1.0
EhchcA = True	0 : 1	=	10.9 : 1.0
)#ehk = True	0 : 1	=	10.6 : 1.0
amkc@@ = True	0 : 1	=	9.2 : 1.0
rpe = True	0 : 1	=	8.8 : 1.0
ahkpa@m = True	0 : 1	=	8.6 : 1.0

,obtinem urmatoarea confusion-matrix si f1 score:

```

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

conf_matr = confusion_matrix(label_test, test_result)
print('Confusion Matrix :')
print(conf_matr)
print('Acuracy :', accuracy_score(label_test, test_result))
print('Report : ')
print(classification_report(label_test, test_result))

```

```

Confusion Matrix :
[[ 553  748]
 [ 307 1048]]
Acuracy : 0.6027861445783133
Report :

```

	precision	recall	f1-score	support
0	0.64	0.43	0.51	1301
1	0.58	0.77	0.67	1355
accuracy			0.60	2656
macro avg	0.61	0.60	0.59	2656
weighted avg	0.61	0.60	0.59	2656

Observam ca daca folosim clasificatorul MultinomialNB obtinem o acuratete de 0.6257, iar daca folosim clasificatorul BernoulliNB obtinem o acuratete de 0.60052

```
[28] MNB_Classifier = SklearnClassifier(MultinomialNB())  
MNB_Classifier.train(train_set)
```

```
↳ <SklearnClassifier(MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True))>
```

```
[29] print("Multinomial naive-Bayes accuracy:", nltk.classify.accuracy(MNB_Classifier, test_set))
```

```
↳ Multinomial naive-Bayes accuracy: 0.6257530120481928
```

```
[31] BernoulliNB_Classifier = SklearnClassifier(BernoulliNB())  
BernoulliNB_Classifier.train(train_set)
```

```
↳ <SklearnClassifier(BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True))>
```

```
▶ print("Bernoulli naive-Bayes accuracy:", nltk.classify.accuracy(BernoulliNB_Classifier, test_set))
```

```
↳ Bernoulli naive-Bayes accuracy: 0.6005271084337349
```