DOCUMENTAȚIE

Proiect Inteligență Artificială – Machine Learning Martie / Aprilie 2020

OVERVIEW

Scopul acestui proiect este de clasificare a tweeturilor criptate în funcție de dialect: în limba moldovenească sau în limba română. Aceasta este o problemă de învățare supravegheată, deoarece vom introduce în modelul nostru un set de date etichetate, de la care calculatorul poate învăța, pentru a face predicții viitoare. Vom încerca diferite abordări pentru a observa care este mai bună pentru setul nostru de date.

[redacted]

Grupa: 241

PASUL 1.1: ÎNȚELEGEREA SETULUI DE DATE

Pentru modelul nostru, vom folosi:

1. train_samples.txt - date de train (conține ID-urile și tweeturile criptate);
2. train_labels.txt - etichetele de train (conține ID-urile și valorile 0, alocate pentru moldovenește sau 1, pentru limba română);
3. validation_samples.txt - date de validare (conține ID-urile și tweeturile criptate);
4. validation_labels.txt - etichetele de validare (conțin ID-urile și valorile 0, alocate pentru limba moldovenească, sau 1, pentru limba română);
5. test_samples.txt - datele de testare.

După cum vedeți, fiecare fișier .txt conține 2 coloane, care nu sunt etichetate, dar care reprezintă: ID-ul și tweet / etichetă.

PASUL 1.2: CITIREA / IMPORTAREA SETULUI DE DATE

Vom scrie o funcție care ne va ajuta să importăm setul nostru de date în program:

```
# Funcție pt a citi fișierele .txt
def citeste_fila(fila):
    with open(fila, mode="r", encoding="latin-1") as f:
        aux = f.readlines()
    return aux

# Citire Propriu-zisă
train_samples = citeste_fila('train_samples.txt')
train_labels = citeste_fila('train_labels.txt')
validation_samples = citeste_fila('validation_samples.txt')
validation_labels = citeste_fila('validation_labels.txt')
test_samples = citeste_fila('test_samples.txt')
```

PASUL 2.1: BAG OF WORDS

Conceptul „*Bag of Words*” este un termen folosit pentru a specifica problemele în care există un „sac de cuvinte” sau o colecție de date text cu care trebuie lucrat. Ideea de bază a BoW este de a lua o bucată de text și de a număra frecvența cuvintelor din acel text. Este important de menționat că acest concept tratează fiecare cuvânt individual și ordinea în care apar cuvintele nu contează.

Folosind acest proces, putem converti o colecție de documente într-o matrice, fiecare document fiind un rând și fiecare cuvânt fiind coloana, iar valorile corespunzătoare (rând, coloană) fiind frecvența apariției fiecărui cuvânt în documentul respectiv.

Pentru a îndeplini acest lucru, vom folosi metoda "*countVectorizer*" din sklearns, care face următoarele:

- Acesta separă șirul în cuvinte individuale și dă un ID de tip Integer fiecărui simbol.
- Numără apariția fiecăruia dintre aceste cuvinte.

IMPORTANT:

- Metoda "*countVectorizer*" convertește automat toate majusculele în litere mici. Tweeturile noastre sunt criptate, așa că, în scopul acestui proiect, trebuie să setăm parametrul „*lowercase*” pe „*False*”, deoarece literele majuscule și cele normale reprezintă lucruri diferite în cadrul cuvintelor criptate.
- De asemenea, ignoră orice semn de punctuație, astfel încât cuvintele urmate de un semn de punctuație (de exemplu: "salut!") nu sunt tratate diferit de aceleași cuvinte care nu sunt prefixate sau sufixate de un semn de punctuație (de exemplu: "salut"). Pentru proiectul nostru, aceasta este o caracteristică utilă, deoarece cuvintele criptate conțin caractere alfanumerice.
- Al treilea parametru de luat în considerare este parametrul „*stop_words*”. „Stop Words” se referă la cele mai utilizate cuvinte dintr-o limbă. Pentru setul nostru de date, această caracteristică nu este potrivită (deoarece datele sunt criptate), așa că o vom seta pe „*None*”.

PASUL 2.2: PUNEREA ÎN APLICARE A CONCEPTULUI DE BOW ÎN SCIKIT-LEARN

```
# CountVectorizer pt BoW
from sklearn.feature_extraction.text import CountVectorizer
# Tweets codate -> nu e nevoie de lowercase sau stop_words
CV = CountVectorizer(max_features = 50000, lowercase = False,
stop_words = None)
```

După cum ne amintim, fiecare fișier .txt conține 2 coloane: ID-uri și tweet / etichetă; deci trebuie să le separăm pentru a putea folosi funcția „countVectorizer”. Deoarece vom avea nevoie mai târziu de etichete, o vom face aici.

```
# Extragere labels (0/1) și Tweets pt countVectorizer
train_labels = [train_labels[x].split('\t')[1].replace("\n", '
') for x in range(len(train_labels))]
train_sentences = [train_samples[x].split('\t')[1] for x in ra
nge(len(train_samples))]

# print(train_labels[0:10])
# print(train_sentences[0:10])


# Aplicăm countVectorizer pe Datele noastre
'''
fit_transform = fit + trasnform   AKA
Învățăm vocabularul dicționarului și o transformăm în matrice
(cuvinte-val)
'''
sentences_CV = CV.fit_transform(train_sentences).toarray()
```

PASUL 3.1: TRAIN / TESTARE

Acum putem continua cu proiectul nostru. Avem deja setul de date împărțit în train, validare și test; deci nu trebuie să ne amestecăm. În schimb, trebuie să editățm datele testului astfel încât să le putem prelucra (vom separa ID-urile de tweet-uri).

```
# Preprocesarea Datelor de Test

# Extragere ID-uri din Setul de date pentru Test
test_IDs = [test_samples[x].split('\t')[0] for x in range(len(
test_samples))]
# print(test_IDs[0:10])

# Extragere Tweets din Setul de date pentru Test
test_sentences = [test_samples[x].split('\t')[1] for x in rang
e(len(test_samples))]
# print(test_sentences[0:10])


# Transformăm setul de date pt Test și întoarcem matricea. (!!
! NU facem fit)
test_CV = CV.transform(test_sentences).toarray()
```

PASUL 3.2: ALEGEREA CELUI MAI BUN ALGORITM

Nu ne rămâne decât să stabilim care algoritm are cea mai bună precizie pentru setul nostru de date.

Vom începe cu:

PASUL 3.2.1: MULTINOMIAL NAIVEBAYES

Acest tip de clasificator este potrivit pentru clasificarea cu caracteristici discrete (cum ar fi, în cazul nostru, numărul de cuvinte pentru clasificarea textului). Primește ca input numere de tip Integer, care reprezintă numărul de apariții ale cuvintelor.

```
# Antrenare pe datele de Train cu Multinomial NB

from sklearn.naive_bayes import MultinomialNB
NB = MultinomialNB()
NB.fit(sentences_CV, train_labels)

# Facem prezicerile pentru setul de date pt Test
prediction = NB.predict(test_CV)


# Salvare Predicții în fișier .csv separat
import pandas as pd
rezultat = pd.DataFrame(data={"id":test_IDs, "label": predicti
on} )
rezultat.to_csv("Clasificator.csv", index = False)
```

Acum că am făcut predicții cu privire la setul de teste, următorul nostru obiectiv este să evaluăm cât de eficient este modelul nostru. Pentru aceasta, ne vom antrena modelul pe setul de validare și vom compara predicția cu etichetele reale. Repetăm astfel pașii de mai sus, dar pentru setul de date de validare.

```
# Extragere labels (0/1) și Tweets pt countVectorizer
validation_labels = [validation_labels[x].split('\t')[1].repla
ce("\n", '') for x in range(len(validation_labels))]
validation_sentences = [validation_samples[x].split('\t')[1] f
or x in range(len(validation_samples))]


# Transformăm setul de date pt Test și întoarcem matricea
validation_CV = CV.transform(validation_sentences).toarray()


# Facem prezicerile pentru setul de date pt Test
prediction_validation = NB.predict(validation_CV)
```

Și acestea sunt rezultatele legate de acuratețea modelului nostru:

```
# Pentru a verifica acuratețea și f1-score
from sklearn.metrics import accuracy_score, f1_score
print('Accuracy score: ', accuracy_score(validation_labels, pr
ediction_validation))
print('F1 score: ', f1_score(validation_labels, prediction_val
idation, average='macro'))
```

```
Accuracy score:  0.6701807228915663
F1 score:  0.6648165222201733
```

```python
# Matricea de Confuzie și Raport

from sklearn.metrics import confusion_matrix, classification_r
eport
cm = confusion_matrix(validation_labels, prediction_validation
)
r = classification_report(validation_labels, prediction_valida
tion)


# Afișare Matrice Confuzie
print('Confusion Matrix :\n')
print(cm)

# Afișare Raport
print('\n\nReport :')
print(r)
```

```
Confusion Matrix :

[[ 722  579]
 [ 297 1058]]


Report :
              precision    recall  f1-score   support

           0       0.71      0.55      0.62      1301
           1       0.65      0.78      0.71      1355

    accuracy                           0.67      2656
   macro avg       0.68      0.67      0.66      2656
weighted avg       0.68      0.67      0.67      2656
```

PASUL 3.2.2: RANDOM FOREST CLASSIFIER

Un algoritm RFC creează mulți arbori de decizie și îi îmbină pentru a obține o predicție mai stabilă și mai precisă. În general, cu cât sunt mai mulți copaci în pădure, cu atât mai puternică e predicția și deci vom obține o precizie mai bună. În RFC, fiecare arbore de decizie preconizează un răspuns pentru o întâmplare, iar răspunsul final este decis prin vot. Pentru problemele de clasificare, răspunsul primit cu votul majorității din arborii de decizie este răspunsul final. Într-un RFC, chiar dacă există mai mulți copaci, modelul nu va permite să se întâmple un caz de overfitting.

```
# Antrenare pe datele de Train cu RFC
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=20000)
rfc.fit(sentences_CV, train_labels)

# Facem prezicerile pentru setul de date pt Test
prediction = rfc.predict(test_CV)
```

Acum vom analiza acuratețea modelului ales.

```
Accuracy score:  0.6558734939759037
F1 score:  0.6557415361436898
```

```
Confusion Matrix :

[[897 404]
 [510 845]]


Report :
              precision    recall  f1-score   support

           0       0.64      0.69      0.66      1301
           1       0.68      0.62      0.65      1355

    accuracy                           0.66      2656
   macro avg       0.66      0.66      0.66      2656
weighted avg       0.66      0.66      0.66      2656
```

După cum se poate remarca, modelul Multinomial NB rămâne cea mai bună decizie deoarece oferă o acuratețe mai mare.