Documentatia Proiectului

Metoda I: Prima mea abordare pentru rezolvarea problemei a fost compusa din metoda bag-of-words, normalizarea datelor si masini cu vectori support (SVM) prin incercarea mai multor parametri si functii.

1. Prima incercare a fost si cea care a obtinut cel mai bun scor si a constat in:

-am citit datele sub urmatoarea forma: fiecare tweet era memorat intr-o lista ce continea cuvintele separate.

```
# load data
f = open('train_samples.txt', 'r')
train_samples = [list(line.split()) for line in f]
```

print (train_samples) => ['107933', 'sgAl', 'NsU', '$DE', '}$%.', '.my', 'qx&A', 'Eh}hxf@Z@m', 'j;}.', 't=Tm#', 'gmr&n|', '%A.', 'bgoK', 'jtFsW', 'Ejmhc@@', 'EAef', '=ZqR', 'qH*y', '(jg', 'A@BRf', 'am@mhaWm', 'Cyt%', '@eAa', 'eA', 'h@@m', '*@m<Z', 'hg', 'AT@', 'mryY', 'me'] …

```
train_labels = np.loadtxt('train_labels.txt', 'int')
```

```
[[104661      1]
 [112752      1]
 [108151      1]
 ...
 [107088      1]
 [105724      1]
 [103551      1]]
```

print (train_labels), dar elimin indexul pentru fiecare linie si raman cu un vector de forma: [1 1 1 ... 1 1 1]

-pentru SVC am folosit in mare parte parametrii impliciti: svm_model = svm.SVC(C=1, kernel='linear', gamma='scale')

Scorul obtinut a fost: pe 80%: 0.70320 pe 20%: 0.67303

2. La a doua incercare nu am modificat nimic fata de primul model in ceea ce priveste paramtri, insa am antrenat si pe datele de validare. Rezultatul obtinut a fost mai prost si m-am decis sa nu mai fac asta in continuare.

Scorul obtinut a fost: pe 80%: 0.67718 pe 20%: 0.65181

3. La a treia incercare am modificat svm_model = svm.SVC(C=2, kernel='linear', gamma='auto'). Scorul a scazut. Scorul obtinut a fost: pe 80%: 0.68218 pe 20%: 0.65073

4. A patra oara m-am decis sa folosesc prima varianta pentru ca pana atunci a obtinut cel mai bun scor, insa sa elimin indexul fiecarui tweet.

```
for i in train_samples:
    i.pop(0)
```

print (train_samples) => ['107933', 'sgAl', 'NsU', '$DE', '}$%.', '.my', 'qx&A', 'Eh}hxf@Z@m', 'j;}.', 't=Tm#', 'gmr&n|', '%A.', 'bgoK', 'jtFsW', 'Ejmhc@@', 'EAef', '=ZqR', 'qH*y', '(jg', 'A@BRf', 'am@mhaWm', 'Cyt%', '@eAa', 'eA', 'h@@m', '*@m<Z', 'hg', 'AT@', 'mryY', 'me'] …

Scorul obtinut a fost mai bun doar pe 20% din cazuri, pe restul a fost mai mic. Scorul obtinut a fost: pe 80%: 0.70129 pe 20%: 0.67477

5. Am incercat si LinearDiscriminantAnalysis() cu parametri default si scorul obtinut a fost: pe 80%: 0.60261 pe 20%: 0.63930

6. Am incercat sa aduc imbunatatiri incercarii cu numarul 4 si am decis sa incerc si cu parametric svm.SVC(C=2, decision_function_shape='ovo') insa pe nici un procent din teste nu a adus vreo diferenta.

Pentru toate incercarile acestea am folosit acelasi bag-of-words si normalize_data care au fost prezentate la laboratorul 5.

```
class Bag_of_words:

    def __init__(self):
        self.vocabulary = {}
        self.words = []
        self.vocabulary_length = 0

    def build_vocabulary(self, data):
        for document in data:
            for word in document:
                if word not in self.vocabulary.keys():
                    self.vocabulary[word] = len(self.vocabulary)
                    self.words.append(word)

        self.vocabulary_length = len(self.vocabulary)
        self.words = np.array(self.words)

    def get_features(self, data):
        features = np.zeros((len(data), self.vocabulary_length))

        for document_idx, document in enumerate(data):
            for word in document:
                if word in self.vocabulary.keys():
                    features[document_idx, self.vocabulary[word]] += 1
        return features
```

```
def normalize_data(train_data, test_data, type=None):
    scaler = None
    if type == 'standard':
        scaler = preprocessing.StandardScaler()

    elif type == 'min_max':
        scaler = preprocessing.MinMaxScaler()

    elif type == 'l1':
        scaler = preprocessing.Normalizer(norm='l1')

    elif type == 'l2':
        scaler = preprocessing.Normalizer(norm='l2')

    if scaler is not None:
        scaler.fit(train_data)
        scaled_train_data = scaler.transform(train_data)
        scaled_test_data = scaler.transform(test_data)
        return (scaled_train_data, scaled_test_data)
    else:
        print("No scaling was performed. Raw data is returned.")
        return (train_data, test_data)
```

“Modelul bag-of-words

➔ este o metodă de reprezentare a datelor de tip text, bazată pe frecvența de apariție a cuvintelor în cadrul documentelor

➔ algoritmul este alcătuit din 2 pași:

1. definirea unui vocabular prin atribuirea unui id unic fiecărui cuvânt regăsit în setul de date (setul de antrenare)

2. reprezentarea fiecărui document ca un vector de dimensiune egală cu lungimea vocabularului, definit astfel: $features[word\_idx] = numărul\ de\ apariții\ al\ cuvântului\ cu\ id - ul\ word\_idx$”

Masina cu vectori suport (SVM) - modele de învățare supravegheate cu algoritmi de învățare asociați care analizează datele utilizate pentru analiza de clasificare (in cazul nostru clasificare de text 0/1 in functie de dialectul tweet-ului) și regresie.

```
Confusion matrix:
[[827 474]
 [383 972]]

F1_score:
0.694037843627276
```

Metoda II: Am avut o abordare gresita a proiectului si nu am incercat si alte metode pana la termenul limita al competitiei, am tot incercat sa obtin scoruri mai bune doar prin modificarea parametrilor de la prima metoda de mai sus. Asa ca in dimineata de 05.04.2020 dupa finalul competitiei am vazut ca puteam trimite in continuare “Late Submission”. Asa ca am decis sa testez ce scoruri as putea obtine folosind Naive Bayes. Am folosit aceleasi date normalizate ca la metoda I si acelasi Bag_of_words.

Prima diferenta dintre Naive Bayes si SVM este ca Naive Bayes are un timp de compilare mult mai mic (~15 secunde in loc de ~1.5 - 2 ore)

1. Prima incercare cu aceasta metoda a fost facuta pe datele memorate ca la Metoda I punctul 4 (fara index la inceput de tweet).

```
from sklearn.naive_bayes import MultinomialNB

naive_bayes_model = MultinomialNB()
naive_bayes_model.fit(train_features, a)
predicted_labels = naive_bayes_model.predict(test_features)
```

```
Confusion matrix:

[[ 787  514]
 [ 262 1093]]

F1_score:

0.738014854827819
```

Scorul obtinut a fost: pe 80%: 0.70659 pe 20%: 0.70017.

2. A doua oara am vrut sa incer si cu datele exact asa cum sunt in fisier (adica fara sa scot indexul).

Scorul obtinut a fost: pe 80%: 0.71358 pe 20%: 0.70670.

In concluzie, indiferent de cum am decis sa prelucrez datele (cu sau fara index) naive_bayes obtine un scor mai bun decat SVM. Cel mai mare scor pe care l-am obtinut in general a fost pentru Metoda II punctul 2.

Variantele pe care le-am luat in considerare pentru evaluarea finala sunt Metoda I punctele 4 si 6. Puteam sa obtin un scor mai bun pe restul de 80% din date doar daca foloseam Metoda I punctul 1, insa pe initialul set de 20% aveam un scor mai mic.

```
Confusion matrix:

[[ 829  472]
 [ 283 1072]]

F1_score:

0.7395653673680579
```