DOCUMENTATIE PROIECT
INTELIGENTA ARTIFICIALA

Introducere

Scopul proiectului este acela de a construi un model care sa diferentieze intre dialectul limbii romane si al limbii moldovenesti (datele au fost initial vizibile, sub forma unor cuvinte separate prin spatii; ulterior, acestea au fost criptate, insa spatiile au fost pastrate)

Am primit ca set de date:

- train_samples (propozitii din limba romana si limba moldoveneasca, fiecare avand un id; acestea sunt folosite pentru antrenarea modelului)
- train_labels (id-urile propozitiilor din train_samples, urmate de label-ul fiecarei propozitii: 0 – limba moldoveneasca; 1 – limba romana)
- validation_samples (la fel ca train_samples, contine propozitii din ambele limbi, fiecare cu cate un id; acestea sunt folosite pentru testarea modelului)
- validation_labels (id-urile propozitiilor din validation_samples cu label-urile corespunzatoare)
- test_samples (propozitii din ambele limbi, fiecare cu cate un id; acestea sunt folosite pentru prezicere)

Dupa antrenarea (train_samples) si testarea modelului (validation_samples), vom prezice label-urile propozitiilor din train_samples si vom pune prezicerile intr-un fisier de tip CSV, pe care mai apoi il postam pe pagina competitiei (Kaggle) pentru a verifica corectitudinea.

Functii si clase folosite pentru implementare:

- def read_data(cale)

```
9   # functie pentru citirea datelor din fisiere
10  def read_data(cale):
11      data = []
12      iduri = []
13      with open(cale, 'r', encoding='utf-8') as fin:
14          line = fin.readline()
15          while line:
16              cuvinte_text = line.split()
17              iduri.append(cuvinte_text[0])
18              data.append(cuvinte_text[1:])
19              line = fin.readline()
20      return iduri, data
```

- este o functie pentru citirea datelor din fisiere
- functia returneaza doi vectori: iduri (toate id-urile propozitiilor din fisier) si data (toate cuvintele din fiecare propozitie a fisierului curent)

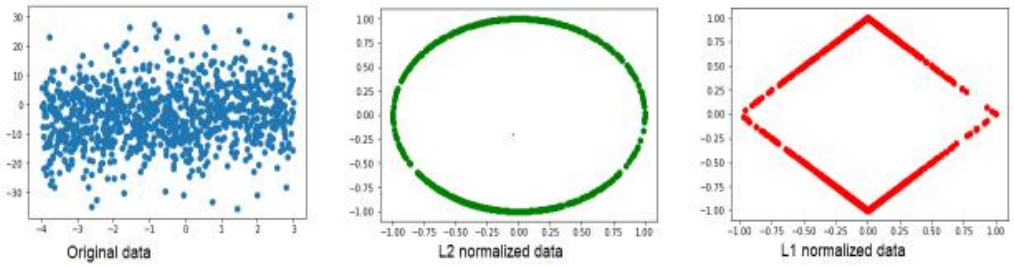
- am citit cate o linie din fisier, iar pentru fiecare linie am format un vector cu toate cuvintele separate prin spatiu (cu functia split())
- am pus id-ul fiecarei propozitii in vectorul iduri (id-ul este mereu pe prima pozitie)
- am pus toate celelalte cuvinte separate prin spatiu in vectorul data

- def normalize_data(train_data, test_data, norm = None)

```
46      # functie pentru normalizarea datelor
47      def normalize_data(train_data, test_data, norm=None):
48          if norm == 'L1':
49              train_data /= np.sum(abs(train_data), axis=1, keepdims=True)
50              test_data /= np.sum(abs(test_data), axis=1, keepdims=True)
51
52          if norm == 'L2':
53              train_data /= np.sqrt(np.sum(train_data ** 2, axis=1, keepdims=True))
54              test_data /= np.sqrt(np.sum(test_data ** 2, axis=1, keepdims=True))
55
56          return train_data, test_data
```

- este o functie pentru normalizarea datelor
- am folosit norma L1 si norma L2

2.2. Normalizarea L1. Normalizarea L2



În partea **stângă** sunt reprezentate datele 2D originale. În **mijloc,** sunt reprezentate datele normalizate folosind norma L_2.
În partea **dreaptă** , sunt reprezentate datele normalizate folosind norma L_1.

- scalarea individuală a vectorilor de caracteristici corespunzători fiecărui exemplu astfel încât norma lor să devină 1.

Folosind norma L_1: $x\_scaled = \frac{x}{||X||_1}, ||X||_1 = \sum_{i=1}^{n} |x_i|$

Folosind norma L_2: $x\_scaled = \frac{x}{||X||_2}, ||X||_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$

- def write_submission(nume_fisier, predictii, iduri)

```
# functie pentru scriere in fisier
def write_submission(nume_fisier, predictii, iduri):
    with open(nume_fisier, 'w') as fout:
        fout.write("id,label\n")
        for id_text, pred in zip(iduri, predictii):
            fout.write(str(id_text) + ',' + str(int(pred)) + '\n')

```

  - este o functie pentru a scrie in fisierul CSV
  - pe prima linie se gaseste “id,label”
  - pe urmatoarele linii se gasesc id-urile propozitiilor din test_samples, urmate de label-urile prezise de model

- Class BagOfWords

```
class BagOfWords:
    def __init__(self):
        self.words = []
        self.vocabulary = {}

    def build_vocabulary(self, sentences):
        for sentence in sentences:
            for word in sentence:
                if word not in self.words:
                    # print(word)
                    self.words.append(word)
                    self.vocabulary[word] = self.words.index(word)

    def get_features(self, sentences):
        features = np.zeros((len(sentences), len(self.words)))
        for index, sentence in enumerate(sentences):
            for word in sentence:
                # print(word)
                if word in self.words:
                    features[index][self.vocabulary[word]] += 1
        return features
```

- def __init__(self)
  - constructorul clasei BagOfWords
  - initializeaza vectorul de cuvinte (words) si dictionarul (vocabulary)
- def build_vocabulary(self, sentences)
  - functie ce construieste vocabularul

    - pentru fiecare cuvant din propozitie, verific daca cuvantul exista deja in lista de cuvinte (words) a clasei
    - in caz negativ, adaug cuvantul in lista si in vocabular (in vocabular, pe pozitia unde am salvat cuvantul, adaug un index – acesta corespunde cu pozitia cuvantului in words)
- def get_features(self, sentences)
    - functie ce construieste matricea de features a datelor (frecventa aparitiei cuvintelor din vocabular in fiecare propozitie)
    - cand este apelata, initializeaza o matrice de dimensiunea nr_propozitii x nr_cuvinte (len(sentences) x len(words))
    - pentru fiecare cuvant din propozitie, daca cuvantul se regaseste in lista de cuvinte, vom creste valoarea in matricea de features, contorizand astfel frecventa fiecarui cuvant

<u>Detalii despre implementare:</u>

Cum datele au fost sub forma unor propozitii in care exista cuvinte criptate, dar separate prin spatii, am considerat ca cea mai buna varianta pentru a antrena modelul ar fi sa folosesc Bag of Words.

Astfel:

- am citit datele (train_samples, train_labels, validation_samples, validation_labels, train_samples) cu ajutorul functiei read_data
- am initializat un obiect de tip BagOfWords
- am format vocabularul (build_vocabulary) cu toate cuvintele unice din toate propozitiile din train_samples
- am construit matricele de features (get_features) pentru train_samples si validation_samples (astfel, am doua matrice de dimensiunea nr_propozitii_fisier x nr_cuvinte_vocabular)
- am normalizat datele (matricile de features) cu ajutorul functiei normalize_data (am incercat diferite variante: fara normalizare, cu norma L1, cu norma L2, insa cel mai bun scor l-am avut cu norma L2)

Am incercat sa antrenez modelul cu diferite variante ale clasificatorilor:

- MultinomialNB cu alpha = 1 / 0.1 / 0.01 / 0.001 / 0.0001
- MultinomialNB cu fit_prior = True / False
- LinearSVC()
- SVC()
- SVC(C=100, kernel='linear')
- SVC cu C = 1 / 10 / 100

Am ales ca submisii finale doua dintre incercari: MultinomialNB(alpha=.01) si LinearSVC()

Initial, am antrenat modelul pe train_samples si am testat pe validation_samples. Am folosit classification_score si confusion_matrix din libraria sklearn pentru a verifica acuratetea prezicerilor. Mai jos sunt detalii despre submisiile finale:

- LinearSVC()

```
Rezultate classification report:

              precision    recall  f1-score   support

           0       0.68      0.66      0.67      1301
           1       0.68      0.70      0.69      1355

   micro avg       0.68      0.68      0.68      2656
   macro avg       0.68      0.68      0.68      2656
weighted avg       0.68      0.68      0.68      2656


Matricea de confuzie:

[[856 445]
 [408 947]]
```

- MultinomialNB

```
Rezultate classification report:

              precision    recall  f1-score   support

           0       0.74      0.69      0.71      1301
           1       0.72      0.76      0.74      1355

   micro avg       0.73      0.73      0.73      2656
   macro avg       0.73      0.73      0.73      2656
weighted avg       0.73      0.73      0.73      2656


Matricea de confuzie:

[[ 898  403]
 [ 322 1033]]
```

Bibliografie:

- https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
- https://scikit-learn.org/stable/modules/svm.html
- https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
- https://scikit-learn.org/stable/modules/naive_bayes.html
- https://scikit-learn.org/stable/modules/preprocessing.html
- https://scikit-learn.org/stable/modules/preprocessing.html#preprocessing-normalization
- https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html
- https://fmi-unibuc-ia.github.io/ia/ (cursurile + laboratorele 3, 4 si 5)