Veronica Chirut, 241

Romanian sub-dialect identification

Discriminate between the Moldavian and the Romanian dialects across different text genres

Descrierea proiectului

Scenariul care trebuie implementat in acest proiect suna astfel: Participantii trebuie sa antreneze un model pe un set de date care contine tweet-uri, urmand sa construiasca un model pentru o clasificare binara in-genre, in functie de dialect, in care un model de clasificare este obligat sa discrimineze intre dialectele moldovenesti (eticheta 0) si dialectele romanesti (eticheta 1).

Ni s-au pus la dispozitie 3 tipuri de date: de antrenare, validare si de testare. Datele de antrenare contin 7757 esantioane, in timp ce datele de validare si de testare contin 2656, respective 2623.

Descrierea fisierelor:

- train_samples.txt - datele de antrenare, cate un esantion pe fiecare rand
- train_labels.txt - etichetele datelor de antrenare, cate o eticheta pe fiecare rand
- validation_samples.txt – datele de validate, cate un esantion pe fiecare rand
- validation_labels.txt – etichetele datelor de validare, cate o eticheta pe fiecare rand
- test_samples.txt – datele de testare, cate un esantion pe fiecare rand
- sample_submission.txt – un fisier de transmitere a rezultatelor finale in formatul correct

Fiecare linie a fisierelor samples reprezinta un esantion de date in care:

- prima coloana reprezinta ID-ul probei de date
- a doua coloana este esantionul de date efectiv

Fiecare linie a fisierelor labels reprezinta o eticheta asociata unui esantion de date in care:

- prima coloana reprezinta ID-ul probei de date
- a doua coloana este eticheta in sine

Abordare

Pentru rezolvarea acestui proiect am incercat diferite modele de clasificatori (Naïve Bayes, Metoda celor mai apropiati vecini, Masini cu vectori suport, Multi-layer Perceptron) pentru a vedea care se potriveste cel mai bine pe setul de date, dar in continuare voi dezvolta

doar doua dintre ele si anume Naïve Bayes si Multi-layer Perceptron, intrucat acuratetea oferita de aceste modele a fost mai mare.

1. **Citirea datelor:**

Pentru citirea datelor am definit functia load_data(path) in care citesc fiecare rand al documentului si memorez in cate un vector indecsii, respectiv esantioanele, pe care le returnez la final.

```
def load_data(path):
    data = []
    ids = []
    with open(path, mode = 'r', encoding = 'utf-8') as fin:
        document = fin.readlines()
    for line in document:
        id, sentence = line.split('\t')
        sentence = sentence.rstrip()
        data.append(sentence)
        ids.append(id)
    data = np.array(data)
    ids = np.array(ids)
    return (ids, data)
```

2. **Manipularea datelor**

Pentru manipularea datelor am folosit metoda Bag of Words. Acest model se bazeaza pe extragerea feature-urilor dintr-un text si folosirea lor, ulterior, in modelare. Este o reprezentare a textului care descrie aparitia cuvintelor in cadrul unui document.

Abordarea pe care am folosit-o este urmatoarea, am definit clasa BagOfWords cu urmatorul constructor, in care vocabulary reprezinta vocabularul de cuvinte, iar vocabulary_length lungimea acestuia.

```
def __init__(self):
    self.vocabulary = []
    self.vocabulary_length = 0
```

In continuare, am definit functia *build_vocabulary* care construieste vocabularul efectiv cu toate cuvintele cunoscute si distincte. Deoarece datele sunt criptate nu le-am transformat in lowercase, gandindu-ma ca exista posibilitatea ca ele sa nu aiba acelasi sens in datele necriptate.

Initial am luat cuvintele cu tot cu duplicate, dupa care, folosindu-ma de functia set() am eliminat duplicatele si cu ajutorul functiei list() am construit o lista de cuvinte. Am memorat in vocabulary_length lungimea vocabularului si folosind functia array() din biblioteca numpy am transformat lista de cuvinte intr-un vector.

```
self.vocabulary = [word for sentence in data
                   for word in sentence.split(' ')]
self.vocabulary = list(set(self.vocabulary))
```

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

Pentru a extrage feature-urile documentelor am definit funtia *get_features*, care calculeaza frecventa cuvintelor in fiecare document. Scopul acestei functii este de a transforma fiecare document text intr-un vector pe care il putem folosi pentru un model de invatare automata.

Stiind numarul cuvintelor distincte din document, construiesc un dictionar cu fiecare cuvant, pe care, ulterior o sa il folosesc in reprezentarea documentelor cu vocabulary_length lungimi fixe.

Folosind ordinea arbitrara a cuvintelor enumerate in vocabulary, parcurg fiecare document si il transform intr-un vector. Pentru fiecare linie a documentului voi avea vocabulary_length pozitii, fiecare reprezentand numarul de aparitii a cuvintelor din esantionul curent in dictionarul construit anterior.

La final, lungimea vectorului de feature-uri va fi egal cu (numarul de samples din setul de date, vocabulary_length).

```
    dictionary = dict(zip(self.vocabulary, range(self.vocabulary_length)))
    features = []
    for sentence in data:
        features_sentence = np.zeros([self.vocabulary_length])
        for word in sentence.split(' '):
            if word in dictionary:
                features_sentence[[dictionary[word]]] += 1
        features.append(features_sentence)
    features = np.array(features)
```

3. Clasificatori

Cum am zis si mai sus, in continuare voi prezenta doi dintre clasificatorii pe care i-am folosit, si anume Naïve Bayes si Multi-layer Perceptron.

- Naïve Bayes:

Pentru ca datele noastre sunt valori continue, le-am transformat in valori discrete cu ajutorul unei histograme. Am stabilit numarul de intervale la care am impartit lungimea intervalului valorilor continue si am asignat fiecarei valori continue din features indicele intervalului corespunzator.

Pe parcurs am dat mai multe valori num_bins-ului pentru a observa care obtine o acuratete mai mare.

- In variabila stop am memorat valoarea cea mai mare din toate feature-urile, adica numarul de aparitii a celui mai frecvent cuvant din vocabular in documente.
- Functia linspace returneaza intervalele, care sunt in numar de num_bins, incepand de la start = 0 pana la stop.
- Am definit functia values_to_bins(matrix, bins) care returneaza pentru fiecare element intervalul corespunzator. Deoarece indexarea elementelor incepe de la 1, pentru ca nu avem valori mai mici decat 0, scad un 1 la final.

```
    num_bins = 200
    stop = int(max(train_features.max(), validation_features.max()))
    bins = np.linspace(start = 0, stop = stop, num = num_bins)
```

```
    def values_to_bins(matrix, bins):
        matrix = np.digitize(matrix, bins)
        return matrix - 1
```

Clasificatorul pe care l-am definit este MultinomialNB din modulul sklearn.baive_bayes care este potrivit pentru clasificarea cu caracteristici discrete. Modelul folosit este unul simplu fara parametri sau attribute, intrucat forma default era mult mai eficienta si ducea la o acuratete mai mare.

```
    from sklearn.naive_bayes import MultinomialNB
    naive_bayes_model = MultinomialNB()
```

Pentru antrenarea modelului am folosit functia fit din modulul mentionat anterior. Antrenarea am facut-o si pe datele de antrenare, dar si pe datele de antrenare+validare, pentru a observa diferenta. Urmeaza prezicerea etichetelor pentru care am folosit functia predict, dupa care, folosind functia *compute_accuracy*, pe care am definit-o, calculez acuratetea.

```
naive_bayes_model.fit(train_features_to_bins, train_labels)
predicted_validation_labels = naive_bayes_model.predict(validation_features_to_bins)
def compute_accuracy(gt_labels, predicted_labels):
    accuracy = np.sum(predicted_labels == gt_labels) / len(predicted_labels)
    return accuracy
```

Pentru a calcula F1 score am transformat valorile din etichetele datelor de validare si etichetele prezise in int pentru a avea accelasi format, dupa care am folosit functia f1_score din biblioteca sklearn.metrics.

F1 score se calculeaza astfel:

```
F1 = 2 * (precision * recall) / (precision + recall)
```

unde: precision = true positive / (true positive + false positive)

recall = true positive / (true positive + false negative)

Pentru un numar de 200 de interval in care am impartit lungimea intervalului valorilor continue, am obtinut urmatoarea acuratete:

```
from sklearn.metrics import f1_score
validation_labels = [int(label) for label in validation_labels]
predicted_validation_labels = [int(label) for label in predicted_validation_labels]
f1_score = f1_score(np.asarray(validation_labels), predicted_validation_labels))
print('F1 score:', f1_score)

>> F1 score: 0.7408637873754154
```

Matricea de confuzie pentru datele de validare am calculat-o definind functia confusion_matrix(true, pred) care numara cate exemple din clasa i au fost clasificate ca fiind in clasa j.

Pentru datele pe care le-am introdus, matricea de confuzie arata ca mai jos.

- 761 - numarul de exemple din clasa 0 care au fost clasificate ca fiind in clasa 0
- 540 - numarul de exemple din clasa 0 care au fost clasificate ca fiind in clasa 1
- 240 - numarul de exemple din clasa 1 care au fost clasificate ca fiind in clasa 0
- 1115 - numarul de exemple din clasa 1 care au fost clasificate ca fiind in clasa 1

```
 def confusion_matrix(true, pred):
     num_classes = 2
     conf_matrix = np.zeros((num_classes, num_classes))

     for i in range(len(true)):
         conf_matrix[int(true[i]), int(pred[i])] += 1
     return conf_matrix
 print(confusion_matrix(validation_labels, predicted_validation_labels))

>> [[ 761.  540.]
   [ 240. 1115.]]
```

- Multi-layer Perceptron:

Pentru normalizarea datelor am folosit Normalizer care normalizeaza esantioanele individual in conformitate cu norma de unitate. Fiecare esantion, adica fiecare rand al matricei de features, cu cel puțin o componentă zero este rescalat independent, astfel incat norma sa l2 este egală cu 1.

Am facut statisticile pe datele de antrenare, folosind fit, dupa care am scalat datele de antrenare si de validare.

```
 from sklearn import preprocessing
 scaler = preprocessing.Normalizer(norm='l2')
 scaler.fit(train_features)
 scaled_train_samples = scaler.transform(train_features)
 scaled_validation_samples = scaler.transform(validation_features)
```

Am definit clasificatorul MLPClassifier din biblioteca sklearn.neural_network.

- Am setat dimensiunea minibatches-urilor pentru optimizatoare la 500.
- Am luat un numar maxim de 200 de interatii. Se itereaza pana la convergenta sau pana la acest numar.
- Hidden_layer_sizes reprezinta numarul de neuroni din stratul ascuns.

```
 from sklearn.neural_network import MLPClassifier
 clf = MLPClassifier(batch_size = 500, max_iter = 200, hidden_layer_sizes =
(100,100,100,100,100))

 clf.fit(scaled_train_samples, train_labels)
 predicted_validation_labels = clf.predict(scaled_validation_samples)
 accuracy = compute_accuracy(np.asarray(validation_labels),
predicted_validation_labels)

 print(accuracy)

>> 0.692394578313253
```

Pentru calcularea F1 score si a matricei de confuzie am folosit aceeasi metoda ca mai sus.

```
 validation_labels = [int(label) for label in validation_labels]
 predicted_validation_labels = [int(label) for label in predicted_validation_labels]
 f1_score = f1_score(np.asarray(validation_labels), predicted_validation_labels))
```

```
print(f1_score)

>> 0.7021509296390812
```

- 876 – numarul de exemple din clasa 0 care au fost clasificate ca fiind in clasa 0
- 425 – numarul de exemple din clasa 0 care au fost clasificate ca fiind in clasa 1
- 392 – numarul de exemple din clasa 1 care au fost clasificate ca fiind in clasa 0
- 963 – numarul de exemple din clasa 1 care au fost clasificate ca fiind in clasa 1

```
def confusion_matrix(true, pred):
    num_classes = 2
    conf_matrix = np.zeros((num_classes, num_classes))

    for i in range(len(true)):
        conf_matrix[int(true[i]), int(pred[i])] += 1
    return conf_matrix
print(confusion_matrix(validation_labels, predicted_validation_labels))

>> [[876. 425.]
  [392. 963.]]
```

4. Submisii

Dupa ce am antrenat mai multi clasificatori cu diferite metode de normalizare, diferiti hiperparametri, am ales cele mai bune metode, care dadeau acuratete cat mai mare pe datele de validare.

Astfel, folosind acelasi clasificator, am prezis etichetele pentru datele de test, asemanator predictiilor facute pe datele de validare, pe care le-am incarcat intr-un fisier cu extensia .csv, astfel: am definit functia submission(file, ids, predictions), unde:

- file este numele fisierului in care vreau sa incarc predictiile
- ids sunt ID-urile esantioanelor
- predictions sunt predictiile effective

Acest fisier trebuie sa aiba formatul exemplului incarcat in sample_submission.txt, adica pe prima linie id,label, dupa care pe cate o linie ID-ul si predictia despartite printr-o virgula

```
def submission(file, ids, predictions):
    with open(file, 'w') as fout:
        fout.write("id,label\n")
        for id, pred in zip(ids, predictions):
            fout.write(id + ',' + str(int(pred)) + '\n')
submission("submission.csv", id_test, predicted_test_labels)
```