Modelul bag-of-words. Kernel Ridge Regression

1. Modelul bag-of-words

➔ reprezintă o metodă de reprezentare a datelor de tip text, care descrie prezența 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:

$$features(word\_idx) = numarul\ de\ aparitii\ al\ cuvantului\ cu\ id - ul\ word\_idx$$

Movie Review Data[1]

În cadrul laboratorului vom folosi setul de date “Movie Review Data”, care conține recenzii de filme, împărțite în două clase: pozitive și negative.

În folderul “*txt_sentoken*” se regăsesc 2.000 de fișiere, 1.000 conținând recenzii pozitive și 1000 conținând recenzii negative. 50% din aceste date vor fi folosite pentru antrenare, 20% pentru validare și 30% pentru testare.

Codul care citește datele și le împarte astfel este prezentat în continuare:

```
import os
import re
from sklearn.utils import shuffle

class Data_Loader:
  def __init__(self):
      dataset_path = "data/txt_sentoken/"
      self.positive_samples_path = dataset_path + "pos/"
      self.negative_samples_path = dataset_path + "neg/"

  # functie care primeste path-ul catre un folder si
  # citeste toate fisierele din el, salvand recenziile intr-o lista
  def read_folder(self, folder_path):
      reviews = []

      for file_name in os.listdir(folder_path):
          file_path = folder_path + file_name
          file = open(file_path, 'r')
          file_content = file.read()
          reviews.append(re.sub("[-.,;:!?\"\'\/()_*=`]", "", file_content).split())
      return reviews
```

[1] http://www.cs.cornell.edu/people/pabo/movie-review-data/

```
  def build_dataset(self):
      positive_samples = self.read_folder(self.positive_samples_path)
      negative_samples = self.read_folder(self.negative_samples_path)

      # impartim setul de date in 50% pentru antrenare, 20% pentru validare si 30% pentru
test
      num_training_samples_per_class = int(0.5 * len(positive_samples))
      num_test_samples_per_class = int(0.3 * len(positive_samples))
      num_eval_samples_per_class = int(0.2 * len(positive_samples))
      # in setul de antrenare salvam exemplele pozitive si negative cu indecsii 0:499
      train_data = positive_samples[0:num_training_samples_per_class] + \
                   negative_samples[0:num_training_samples_per_class]

      # exemplele pozitive vor avea eticheta 1, iar cele negative -1
      train_labels = [1] * num_training_samples_per_class + \
                     [-1] * num_training_samples_per_class

      # in setul de validare salvam exemplele pozitive si negative
      eval_data =
positive_samples[num_training_samples_per_class:num_training_samples_per_class +
num_eval_samples_per_class] + \

negative_samples[num_training_samples_per_class:num_training_samples_per_class +
num_eval_samples_per_class]

      # exemplele pozitive vor avea eticheta 1, iar cele negative -1
      eval_labels = [1] * num_eval_samples_per_class + \
                    [-1] * num_eval_samples_per_class

      # in setul de test salvam exemplele pozitive si negative
      test_data = positive_samples[num_training_samples_per_class +
num_eval_samples_per_class:len(positive_samples)] + \
                  negative_samples[num_training_samples_per_class +
num_eval_samples_per_class:len(negative_samples)]
      test_labels = [1] * num_test_samples_per_class + \
                    [-1] * num_test_samples_per_class

      # amestecam datele
      self.train_data, self.train_labels = shuffle(train_data, train_labels)
      self.test_data, self.test_labels = shuffle(test_data, test_labels)
      self.eval_data, self.eval_labels = shuffle(eval_data, eval_labels)
```

```
from Data_Loader import *
data_loader = Data_Loader()
data_loader.build_dataset()
print(f"Training data length: {len(data_loader.train_data)}")     # => 1000
print(f"Test data length: {len(data_loader.test_data)}")          # => 1000
```

2. Kernel Ridge Regression

➔ Algoritmul combină regresia ridge cu funcțiile nucleu

➔ Algoritmul minimizează funcția:

$$|| y_{hat} - y||_2^2 + alpha * ||w||_2^2$$, unde

$y_{hat} = sign(\sum_{i=1}^{n} y_i w_i K(x_i, x'))$ și

y - etichetele corecte

w - ponderile învățate de model

y_{hat} - etichetele prezise

K - funcție nucleu

Importarea modelului:

```
from sklearn.kernel_ridge import KernelRidge
```

Detalii de implementare:

1. **Definirea modelului:**

```
class sklearn.kernel_ridge.KernelRidge(alpha, kernel, gamma)
```

Parametri:

alpha (float)

- ponderea termenului de regularizare al modelului
  - alpha mare => regularizare puternică
  - alpha = 0 => nu se face regularizare

kernel (string, default = ‘rbf’)

- tipul de kernel folosit: în cadrul laboratorului vom lucra cu ‘linear’ și ‘rbf’

Kernel linear:

$$K(u, v) = u^T v$$

Kernel RBF:

$$K(u, v) = exp(-gamma * ||u - v||^2)$$

gamma (float, default =None)

- coeficient pentru kernelul ‘rbf’

2. **Antrenarea:**

```
krr_model.fit(train_data, train_labels)
```

Parametri:

train_data

- setul de antrenare având exemplele stocate pe linii => dimensiune ($num\_samples$ x $num\_features$)

train_labels

- etichetele corespunzătoare fiecărui exemplu de antrenare

3. **Predicția**

```
krr_model.predict(test_data)
```

Parametri:

test_data

- setul de test având exemplele stocate pe linii => dimensiune ($num\_test\_samples$ x $num\_features$)

Funcția întoarce un vector cu $num_test_samples$elemente de tip *float*, fiecare reprezentând valoarea prezisă de model pentru respectivul exemplu. Pentru a stabili eticheta finală se aplică funcția ***sign*** definită astfel:

$$sign(x) = \begin{matrix} 1 & ,daca\ x \geq 0 \\ -1 & ,daca\ x < 0 \end{matrix}$$

Exerciții

1. Încărcați setul de date “Movie Review Data” folosind clasa Data_Loader.

2. Definiți clasa **Bag_of_Words** în al cărui constructor se inițializează vocabularul (un dicționar gol). În cadrul ei implementați metoda **build_vocabulary(self, data)** care primește ca parametru o listă de documente (listă de liste de strings) și construiește vocabularul pe baza acesteia. Cheile dicționarului sunt reprezentate de cuvintele din documente, iar valorile de id-urile unice atribuite cuvintelor.
   Afișati vocabularul construit.

   OBS. Vocabularul va fi construit doar pe baza datelor din setul de antrenare.

3. Definiți metoda **get_features(self, data)** care primește ca parametru o listă de documente de dimensiune $num\_samples$(listă de liste de strings) și returnează o matrice de dimensiune ($num\_samples$ x $dictionary\_length$) definită astfel:

$$features(sample\_idx, word\_idx) = numarul\ de\ aparitii\ al\ cuvantului\ cu\ id-ul\ word\_idx\ \ in\ documentul\ sample\_idx$$

4. Antrenați un SVM pe setul de date “Movie Review Data” folosind vectorii de caracteristici obținuți prin aplicarea modelului bag-of-words și calculați acuratețea pe mulțimea de testare.

OBS. Vectorii trebuie normalizați.

5. Antrenați un model KRR folosind kernel-ul ‘linear’ și ‘rbf’, calculați acuratețea ambelor modele pe mulțimea de validare, apoi reantrenați clasificatorul cel mai performant pe mulțimea de antrenare și mulțimea de validare și calculați acuratețea lui pe mulțimea de testare.