TD8._sujet

March 4, 2023

1 TD8 Identification des langues avec des bigrams

1.1 Partie1

[ ]:
```
import pickle
import string
import os
from nltk import ngrams, FreqDist, word_tokenize
from numpy import arange
import matplotlib.pyplot as plt
%matplotlib inline
```

1.1.1 Jeu de données

Pour ce TD, on va travailler avec un corpus en 3 langues. L'idée est de prédire la langue de chaque ligne dans un fichier `LangId.test` en utilisant le modèle de ngrams.

Le jeu de données se trouve ici : https://github.com/chuyuanli/TAL-L3-TD-language-detection-ngram

Dans le folder `data/`, vous trouvez 5 fichiers: - `LangID.train.English`, `LangID.train.Italian`, `LangID.train.French` : ces trois fichiers sont vos fichers d'entreinement. - `LangID.test` : ce ficher est le ficher de test, vous devez lire chaque ligne et prédire quelle langue c'est - `LangID.sol` : ce ficher est le **gold**, cela veut dire qu'il contient la bonne réponse.

Vous pouvez faire un `clone` de ce github, ou bien directement télécharger le zip. Mettre le zip sous le même répertoire que votre notebook pour avoir le même chemin.

Pour cloner un git : `git clone [HTTPS://]`, vous pouvez copier coller l'adresse https en cliquant sur le bouton verte `Code`.

1.1.2 Exo1 A l'oral : est-ce que vous arrivez à identifier ?

Parmi l'anglais, le français et l'italien, à quelle langue ces mots appartiennent-ils ?

- `menu`
- `restaurant`
- `si`
- `idea`

Et maintenant ?

- le menu
- a restaurant
- mais si
- nessuna idea

Si on regarde qu'un mot, on a 50% de chance d'avoir la bonne réponse (ou moins, si on élargit le choix des langues). Si on regarde le bigram, la chance augmente.
Tout est sur la probabilité !

1.1.3 Exo2 Construction des données d'apprentissages en français, italien et anglais

Vous allez lire les fichiers `LangId.train.[lang]`, pour produire les unigrams et les bigrams de chaque des langues. Stocker ces résultats en utilisant **le module `pickle`** : un module pour **serializing** des objets en Python.
Le principe est d'enregistrer un objet Python directement. Cela est particulièrement utile lorsque la construction de l'objet est longue ou que sa taille est importante.

Vous pouvez suivre ces étapes pour créer les fichiers `.pickle` (Ou bien écrire les scripts à votre façon !)

(1) Ecrire une fonction `get_unigram_bigram_dicts()` qui prend un fichier et retourne un dictionnaire de unigrams et un dictionnaire de bigrams. Il prend comme entrée le chemin d'un ficher.

Par exemple :

```
>>> file = 'chemin2fichier'
>>> file.read()
'je aime tal'
>>> unigrams, bigrams = get_unigram_bigram_dicts(file)
>>> unigrams
{'je': 1, 'aime': 1, 'tal': 1}
>>> bigrams
{'_ je': 1, 'je aime': 1, 'aime tal': 1, 'tal _': 1}
```

Plus précisément, dans cette fonction, vous pouvez : - lire le contenu du fichier `read()` et le stocker dans une variable, - tokenizer le contenu, - obtenir les unigrams et bigrams grâce à la méthode `ngrams` de NLTK

Ces deux dictionnaires constituent un tuple `(dico_unigram, dico_bigram)`, et ce tuple est le retour de votre fonction.

Pour lire un fichier, voici des exemples : https://www.w3schools.com/python/python_file_open.asp

Pour appeler une fonction dans une autre fonction, voici un exemple :

```
def plus1(a):
    return a+1

def affichage_plus1(a):
    result = plus1(a)
    print(result)
```

```
>>> affichage_plus1(3)
4
```

[ ]:
```python
# TODO


# uncomment for testing
# for lang in ['English', 'French', 'Italian']:
#     file = f"TAL-L3-TD-language-detection-ngram/data/LangID.train.{lang}"
#     uni, bi = get_unigram_bigram_dicts(file)
#     print(len(uni), len(bi))
```

(2) Ecrire la fonction `dump_pickle()` qui prend comme argument un langage et qui dump (décharge) l'objet `dico_unigram` et `dico_bigram` dans les fichiers `[lang].unigram.pickle` et `[lang].bigram.pickle`.

Pour dump un dictionnaire, voici un exemple :

```
dico = {'je': 1, 'aime': 1, 'tal': 1}'
with open(path2file + language + '.unigram.pickle', 'wb') as handle:
    pickle.dump(dico, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

- `path2file` est le chemin vers le répertoire `data/`
- `language`: {'English', 'French', 'Italian' }
- `.pickle` est l'extension du fichier

Votre fonction ne retourne rien, mais elle doit créer 2 fichiers `.pickle` pour chaque langue.

```
data/
    English.unigram.pickle
    English.bigram.pickle
    French.unigram.pickle
    French.bigram.pickle
    ...
```

Ca y est ! Vous venez de créer des corpus pour l'entraînement !

Plus d'exemples pour l'utilisation de pickle : https://www.geeksforgeeks.org/understanding-python-pickling-example/

[ ]:
```python
# TODO

def dump_pickle():
    pass

# uncommet for testing, modify the datapath on your pc accordingly
# datapath = 'TAL-L3-TD-language-detection-ngram/data/'
# for lang in ['English', 'French', 'Italian']:
#     dump_pickle(datapath, lang)
```

1.1.4 Exo3 Un peu de stats

C'est important de savoir les stats de votre corpus :)

(1) Obtenir le nombre des phrases dans chaque fichier d'entrainnement `LangId.train.[language]`, et afficher le résultat :

```
NUMBER OF SENTENCES IN TRAINING DATA
English: 2980
French: 2980
Italian: 2821
```

[ ]:
```
# TODO
import re


# uncommet for testing
# print(f"English: {number_sents_en}\nFrench: {number_sents_fr}\nItalian:
 ↪{number_sents_it}")
```

(2) On souhaite également savoir la **taille du vocabulaire**.

Pour cela, vous pouvez compter la tailler des unigrams de chaque langue.

Lire les fichiers `pickle` que vous venez de créer dans l'exo2 et compter la taille du dictionnaire unigram. Pour lire le pickle, vous pouvez utiliser cette commande :

```
with open(path2file+'English.unigram.pickle', 'rb') as handle:
    unigram_english_dict = pickle.load(handle)
```

Les résultats :

```
English: 6128
French: 7894
Italian: 8683
```

[ ]:
```
# TODO

path2file = 'TAL-L3-TD-language-detection-ngram/data/'


# uncomment for testing
# print("Unigram size: ")
# print(f"English: {len(unigram_english)}")
# print(f"French: {len(unigram_french)}")
# print(f"Italian: {len(unigram_italian)}")

# print("\nBigram size: ")
# print(f"English: {len(bigram_english)}")
# print(f"French: {len(bigram_french)}")
# print(f"Italian: {len(bigram_italian)}")
```

1.1.5 Exo4 Identifier la langue en utilisant les probabilités des bigrams

(1) On commence par calculer la probabilité qu'un bigram.

Un exemple :
La probabilité du bigram `il est` dans un corpus français. On regarde l'occurence du `il est` et l'occurrence du premier mot de bigram `il` :

- `il est`: 88 fois
- `il` : 428 fois

La probabilité que ce bigram appartienne dans un texte français est donc :

`88 / 428 = 20.6%`

Ecrire une fonction `get_bigram_probability()` qui prend comme argument : - un bigram: un tuple avec deux string ('a', 'b') - un dictionaire des bigrams - un dictionaire des unigrams

```
def get_bigram_prob(bigram, bigram_dict, unigram_dict):
    pass
```

[ ]:
```
# TODO

def get_bigram_prob(bigram, bigram_dict, unigram_dict):
    pass
```

(2) Maintenant écrire une fonction `get_language_probability` qui donne la probabilité d'un liste des bigrams appartiennent à une certaine langue.

Un exemple :

`Le chien mange une pomme.`

Les bigrams dans cette phrase sont : `('Le', 'chien')`, `('chien', 'mange')`, `('mange', 'une')`, `('une', 'pomme')`.

Deux questions : - Comment calculer la probabilité d'une liste des bigrams ? On fait la somme, le produit, ou autres combinaisons de ces bigrams ? - Si vous multipliez les probabilités de bigrams. Quand la probabilité d'un bigram est 0, la probabilité de toute la phrase serait 0. Comment gérer dans ce cas ? - Hint : add-1 smoothing - More info about smoothing techiques in NLP: https://www.codingninjas.com/codestudio/library/smoothing-in-nlp

```
def get_language_probability(bigram_list, bigram_dict, unigram_dict):
    pass
```

[ ]:
```
# TODO

# Get probability that a given bigram list is of a language (specified by its␣
↪bigram_dict)
def get_language_probability(bigram_list, bigram_dict, unigram_dict):
    pass


# uncomment for testing
```

```
# test1 = "je suis heureux de venir"
# tokens = tokenize(test1)
# bigrams = list(ngrams(tokens, 2))
# print(get_language_probability(bigrams, bigram_french, unigram_french))
# print(get_language_probability(bigrams, bigram_english, unigram_english))
# print(get_language_probability(bigrams, bigram_italian, unigram_italian))
```

1.2 Partie II

1.2.1 Exo5 On regarde le Gold !

Comment évaluer votre résultat ?

Pour cela, il faut comparer avec le **gold** : le fichier `LangId.sol`.

Voici les étapes : - créer un variable vide `solution_dict = dict()` - lire le fichier `LangId.sol` ligne par ligne : la première partie est le numéro de ligne, et la deuxième partie est la bonne réponse. - stocker ces deux parties dans le dictionnaire `solution_dict`. - afficher le résultat

```
>>> solution_dict
{1: 'Italian',
 2: 'English',
 3: 'Italian',
 4: 'French',
 ...
}
```

[ ]:
```
# TODO
```

1.2.2 Exo6 Mettre tout ensemble

Vous y est presque ! La dernière étape est de regrouper les différents éléments.

Pour calculer l'exactitude de prédiction des langues dans le fichier `LangId.test`, vous allez faire les étapes suivantes : - ouvrir ce fichier et lire ligne par ligne - pour chaque ligne, obtenir une liste de bigrams et calculer `language_probability` de fr, en, it - prendre la probabilité la plus haute comme la langue prédite - comparer avec le "gold" (celui dans `solution_dict`) et compter le nombre de fois où la prédiction est correcte - afficher l'exactitude - (optionnel) Calculer le f-score pour chacune des langues (n'oubliez pas il faut calculer la précision et le rappel pour chacune des langues). - (optionnel) extraire les lignes où la prédiction est erronée. Combien de phrases sont concernées ?

[ ]:
```
# initiate variables
result_dict = dict()
correct = 0
incorrect_lines = []

# need to add padding symbol in unigram because I'm using padding for bigrams
unigram_english['_'] = number_sents_en
unigram_french['_'] = number_sents_fr
```

```
unigram_italian['_'] = number_sents_it

test_file = 'TAL-L3-TD-language-detection-ngram/data/LangId.test'

# TODO



# Uncomment for testing
# with open('TAL-L3-TD-language-detection-ngram/data/LangId.myresult', 'w') as␣
 ↪f:
#     for (key, val) in result_dict.items():
#         f.write(' '.join([str(key), val]) + '\n')

# print('Accuracy: {:2.2f}%'.format(correct * 100 / len(solution_dict)))
# print('Line numbers for incorrectly classified languages: {}'.
 ↪format(str(incorrect_lines)))
```

1.2.3 /! Submission instructions

You will work on this lab for two TDs: td8 and td9.
You will need to submit **Partie 1** of this lab on Arche before 9:59am on **Friday, 17rd March**.

Submit either a `.py` or an `.ipynb` file containing the functions you wrote for all the exercises and name it `td8_firstname_lastname.py` or `td8_firstname_lastname.ipynb` accordingly, where `firstname` should be your first name and `lastname` should be your last name (e.g. Jane Doe's submission should be called `td8_jane_doe.py` or `td8_jane_doe.ipynb`).