TD10._sujet

March 17, 2023

1 TD10 TF-IDF from scratch

1.1 Partie 1 Introduction

Le TF-IDF (term frequency-inverse document frequency) est une méthode de pondération souvent utilisée en recherche d'information et en particulier dans la fouille de textes. Cette mesure statistique permet d'évaluer l'importance d'un terme contenu dans un document, relativement à une collection ou un corpus. Le poids augmente proportionnellement au nombre d'occurrences du mot dans le document. Il varie également en fonction de la fréquence du mot dans le corpus. https://fr.wikipedia.org/wiki/TF-IDF

Dans cette partie, on va regarder étape par étape comment représenter un doc via TF-IDF.

Tout d'abord, importer des librairies :

[ ]:
```
import pandas as pd
import math
```

1.1.1 Terminologie

t - terme (mot)
f - frequency
i - inversed
d - document

tf - term frequency
idf - inversed document frequency

corpus - collection des documents
N - nombre des documents

1.1.2 Warm-up, calculer ensemble

```
>>> corpus = [
...     'This is the first document.',
...     'This document is the second document.',
...     'And this is the third one.',
...     'Is this the first document?',
... ]
```

Calculer sur papier le TF-IDF de la 1er et 2eme phrases dans ce corpus, en suivant les étapes suivantes : - les termes - tf de s1 et tf de s2 - df du corpus - idf du corpus - tfidf(s1) = tf_s1 x idf - tfidf(s2) = td_s2 x idf

Discuter avec vos camarades, est-ce que vous obtenez le même résultat ?

N = 4

	'and'	'document'	'first'	'is'	'one'	'second'	'the'	'third'	'this'
tf_s1	0	1/5	1/5	1/5	0	0	1/5	0	1/5
tf_s2	0	2/6	0	1/6	0	1/6	1/6	0	1/6
df	1/4	**3/4**	2/4	4/4	1/4	1/4	4/4	1/4	4/4
idf	4/1	4/3	4/2	4/4	4/1	4/1	4/4	4/1	4/4
tf_s1 * idf	0	4/15	2/5	...					
tf_s2 * idf	0	4/9	0	...					

Maintenant transformer le calcul en python :

[ ]:
```
s1 = "Natural language processing (NLP) is a subfield of linguistics, computer␣
↪science, and artificial intelligence concerned with the interactions between␣
↪computers and human language."
s2 = "NLP is so cool :D Challenges in natural language processing frequently␣
↪involve speech recognition, natural language understanding, and␣
↪natural-language generation."
```

1.1.3 Exo1 Chargement des données et calcul de l'union

Vous comprenez le calcul pour tf-idf. Maintenant vous pouvez soit implémenter tdidf de votre manière, soit suivre les étapes proposées suivantes.

Prendre ces deux phrases `s1` et `s2` et les séparer en tokens, ensuite créer une union d'ensemble (un **set**) à partir de ces strings. Nommer une variable `total` pour l'union des mots. `total` est le vocabulaire de votre corpus :

`{'intelligence', 'computers', 'natural', 'cool', 'natural-language', 'between', 'processing',`

Hint : l'objet **set** a une fonction `union`: `seta.union(setb)`

Afficher le résultat.

[ ]:
```
# TODO


# uncommet for testing
# print(total)
```

1.1.4 Préprocessing

Maintenant qu'on a une union des mots, observer les résultats :

Des mots comme “with”, “the”, “is” sont des “stopwords” qui n’ont pas de vraie valeurs. Enlever ces mots améliorait notre calcul. - Autres améliorations ? Par exemple les maj, les ponctuations, les emojis, etc. - Certains mots ont le même lemme (rappel du TD5), comme “computers” et “computer”, est-ce qu’on les compte comme 2 mots différents ? Pensez à utiliser le lemmatizer de NLTK pour améliorer votre résultat.

Vous pouvez écrire votre propre function pour le préprocessing.

Ou alors faire tourner le code et obtenir `s1_clean`, `s2_clean` et l’union `total`.

[ ]:
```
# TODO

import nltk
from nltk.corpus import stopwords
from nltk import WordNetLemmatizer
wnl = WordNetLemmatizer()

def preprocess(sent):
    sent_clean = []
    for word in sent:
        if str(word) not in set(stopwords.words('english')):
            word = word.lower()
            word = wnl.lemmatize(word)
            word = word[:-1] if word[-1] in ",.!?/'*" else word
            sent_clean.append(word)
    return sent_clean

s1_clean = preprocess(s1)
s2_clean = preprocess(s2)

print(s1_clean)
print(s2_clean)

total= set(s1_clean).union(set(s2_clean))
print(len(total))
```

1.1.5 Exo2 Word-dict représentation

Maintenant on va compter les mots à l’aide d’un dictionnaire. Ecrire une fonction `get_word_dict()` qui prend une phrase et l’uninon des mots `total` comme arguments, et renvoie un dictionnaire (key-value pair) avec comme clé le mot, et comme valeur l’occurence de ce mot. Nommer le résultat de s1 et s2 avec `wordDict_s1` et `wordDict_s2`.

Attention Le dictionnaire sortant contient tous les mots présents dans l’union des mots. Pour les mots qui ne sont pas dans la phrase, attribuer la valeur 0.

Exemple :

```
>>> a = ['un', 'chat', 'dort', 'dort']
>>> union = {'un', 'chat', 'dort', 'chien', 'mange'}
```

```
>>> get_word_dict(a, union)
{'un': 1, 'chat':1, 'dort':2, 'chien': 0, 'mange': 0}
```

[ ]:
```python
# TODO

def get_word_dict(sent, word_union):
    pass

# Uncomment for testing
# wordDict_s1 = get_word_dict(s1_clean, total)
# wordDict_s2 = get_word_dict(s2_clean, total)
# print(wordDict_s1)
```

On peut visualiser `wordDict_s1` et `wordDict_s2` avec pandas.

Exécuter le code suivant, normalement vous allez voir un tableau avec 2 lignes et ~24 colonnes (dépend de votre préprocess). Chaque ligne représente un wordDict, et chaque colonne représente un mot.

[ ]:
```python
pd.DataFrame([wordDict_s1, wordDict_s2])
```

1.1.6 Exo3 TF (term frequency)

Ecrire une fonction `compute_tf()` qui calcule la valeur TF d'un document. Cette fonction prend comme argument un `word_dict`. Vous pouvez - calculer la somme de termes dans ce document (n) - boucler sur chaque mot dans `word_dict` et diviser par n - stocker et retourner le résultat de la division `tfDict`

De la même manière, visualier avec pandas `DataFrame`.

Rappel : `tf(t,d) = count of t in d / number of words in d`

[ ]:
```python
# TODO

def compute_tf(wordDict):
    pass

# Uncomment for testing
# tf_s1 = compute_tf(wordDict_s1)
# tf_s2 = compute_tf(wordDict_s2)

# #Converting to dataframe for visualization
# pd.DataFrame([tf_s1, tf_s2])
```

1.1.7 Exo4 IDF (inverse document frequency)

Maintenant écrire une fonction `compute_idf()` qui calcule IDF. La fonction prend comme argument une liste de documents sous forme de dictionnaire `[wordDict_s1, wordDict_s2, ...]` et renvoie un dictionnaire `idfs`.

Par exemple, si votre corpus contient 2 documents. Le mot “science” apparaît dans doc 1 et pas doc 2, le df du mot “science” est 1. Si un mot “is” appraît dans tous les documents, il aurait une valeur de df 2.

On inverse la valeur du df pour obtenir idf `idf = N / df`. Le idf du mot “science” est 2/1 = 2; l’idf du mot “is” est 2/2=1.

Rappel :

```
df(t) = occurrence de documents où apparaîssent le terme t: {0, N}
idf(t) = N/df
```

NB Dans le cas où N est très grand, on calcule idf en utilisant le log. Pour cet exercise, vous pouvez rester sur la formule initiale.

[ ]:
```python
# TODO

def compute_idf(docList):
    pass

# Uncomment for testing
# idfs = compute_idf([wordDict_s1, wordDict_s2])
# pd.DataFrame([idfs])
```

1.1.8 Combiner ensemble

Vous y est presque ! Vous allez écrire une fonction qui combine la valeur tf et idf de chaque document. Comme arguments : - tf d’un document - idf du corpus

Et comme sortie retourner un dictionnaire qui stocke la valeur tfidf de chaque mot dans ce document.

[ ]:
```python
def compute_tfidf(tfs, idfs):
    pass

# Uncommet for testing
#running our two sentences through the IDF:
# tfidf_s1 = compute_tfidf(tf_s1, idfs)
# tfidf_s2 = compute_tfidf(tf_s2, idfs)

# #putting it in a dataframe
# pd.DataFrame([idf_s1, idf_s2])
```

1.2 Partie 2 (optionnel) Pour aller plus loin

Vous venez d’implémenter le calcul pour TF-IDF from scratch, bravo ! Maintenant explorer la fonction implémentée dans sklearn. Pour cela, il vous faut importer `TfidfVectorizer`.

Plus info : https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.htm

[ ]:
```python
# TODO
```

1.2.1 /! Submission instructions

You will work on this lab for TD10.
You will need to submit this lab on Arche before 9:59am on **Friday, 31st March**.

Submit either a `.py` or an `.ipynb` file containing the functions you wrote for all the exercises and name it `td10_firstname_lastname.py` or `td10_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 `td10_jane_doe.py` or `td10_jane_doe.ipynb`).