TD5 : word embedding

LO17

Fatma Chamekh

2021-2022

Objectifs :

L'objectif de ce TD est d'explorer le word embedding en utilisant l'algorithme word2vec. Il comporte deux parties :

- Partie 1 : découverte, manipulation de l'algorithme word2vec.
- Partie 2 : Appliquer le word2vec pour créer des indexes pour la base citation..

Partie 1 :

Le dossier découverte word2vec contient un notebook et les données. Le notebook est en anglais.

Partie 2 :

Exercice1 : Acquisition des données

Le jeu de données considère est tiré du Citation Network Dataset qui rassemble plusieurs millions d'articles accessible via le lien suivant https://www.aminer.org/citation. Vous utiliserez en particulier la version 10 qui comprend : 3 079 007 articles et 25 166 994 citations, extraits le 27 octobre 2017. L'archive contient plusieurs (gros) fichiers au format .json. En utilisant la librairie Python du même nom, chargez ces données afin de les transformer dans un format .csv plus digeste en ne conservant que quelques informations : identifiant, titre et résumé (s'il existe).

Au vu de la taille et suivant les capacités de la machine que vous employez, vous pouvez vous restreindre `a une année ou filtrer d'une manière de votre choix (par ex. sur la base d'un mot-clef dans le titre). L'objectif est d'avoir un jeu de données de taille suffisante mais traitable en temps raisonnable, imaginons entre 10k et 200k articles. Sauvegardez ce jeu de données dans un fichier .csv que vous pourrez ensuite facilement charger en mémoire pour les analyses futures. Vous pouvez rencontrer des problèmes avec le caractère Unicode nul ('\$x00'). Pour résoudre ce problème, vous pouvez le remplacer par un caractère vide grâce `a la commande replace.

Exercice2 : création d'index

Il s'agit de construire un index sur un vocabulaire de mots extraits du corpus. Construisez ce vocabulaire en pensant à :

• supprimer les mots-outils,

• supprimer les mots trop fréquents,

• supprimer les mots trop rares.

Si vous le souhaitez, vous pouvez passer par une étape de normalisation des mots (stemming, lemmatisation).

Suite à cela, vous pouvez construire la matrice Documents x Termes en utilisant le modèle de sac de mots. Afin d'améliorer les résultats, vous pouvez adopter le schéma de pondération de votre choix. L'idéal serait de pouvoir tester plusieurs options (par ex. TF et TFxIDF) afin de pouvoir faire étudier l'impact de ce choix par la suite. Pour vous aider à réaliser cette partie, vous pouvez vous référer au code suivant :

Prenons à présent deux documents, tels que :

In [1]:
```
X = ["Some say the world will end in fire,",
     "Some say in ice."]
```

In [2]:
```
len(X)
```

Out[2]:
```
2
```

La librairie contient une fonction qui permet de "vectoriser" un ensemble de textes en prenant en compte un certain nombre de prétraitements (*preprocessing*) couramment employés : mis en minuscule, utilisation de mots-outils, etc.

In [3]:
```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
vectorizer.fit(X)
```

Out[3]:
```
CountVectorizer()
```

Observons le vocabulaire automatiquement construit à partir de ces deux textes :

In [4]:
```
vectorizer.vocabulary_
```

Out[4]:
```
{'some': 5,
 'say': 4,
 'the': 6,
 'world': 8,
 'will': 7,
 'end': 0,
 'in': 3,
 'fire': 1,
 'ice': 2}
```

Matrice documents x termes

On peut maintenant construire la matrice documents * termes :

In [5]:
```
X_bag_of_words = vectorizer.transform(X)
```

In [6]:
```
X_bag_of_words
```

Out[6]:
```
<2x9 sparse matrix of type '<class 'numpy.int64'>'
	with 12 stored elements in Compressed Sparse Row format>
```

In [7]:
```
X_bag_of_words.shape
```

Out[7]:
```
(2, 9)
```

Jetons un oeil au contenu de la matrice :

In [8]:
```
X_bag_of_words.toarray()
```

Out[8]:
```
array([[1, 1, 0, 1, 1, 1, 1, 1, 1],
       [0, 0, 1, 1, 1, 1, 0, 0, 0]])
```

On observe que la valeur indiquée dans une cellule est le nombre d'occurrences d'un terme dans un document (TF pour *Term Frequency*)

On peut également retrouver le nom des termes du vocabulaire :

On observe que la valeur indiquée dans une cellule est le nombre d'occurrences d'un terme dans un document (TF pour *Term Frequency*)

On peut également retrouver le nom des termes du vocabulaire :

In [9]:
```
vectorizer.get_feature_names()
```

```
/Users/jvelcin/arm/envs/cours21_spacy/lib/python3.9/site-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

Out[9]:
```
['end', 'fire', 'ice', 'in', 'say', 'some', 'the', 'will', 'world']
```

On peut retrouver les attributs (features) utilisés dans les documents :

In [10]:
```
vectorizer.inverse_transform(X_bag_of_words)
```

Out[10]:
```
[array(['end', 'fire', 'in', 'say', 'some', 'the', 'will', 'world'],
       dtype='<U5'),
 array(['ice', 'in', 'say', 'some'], dtype='<U5')]
```

Représentation TFxIDF

A la place du nombre d'occurrences (TF), on peut utiliser une autre mesure qui prend en compte la rareté d'un mot dans le corpus :

$$tf_{t,d} \times idf_t$$

avec $tf_{t,d}$ le nombre d'occurrences de t dans d

et $idf_t = \log \frac{N}{df_t}$ (N est le nombre total de documents)

In [11]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer()
tfidf_vectorizer.fit(X)
```

Out[11]:
```
TfidfVectorizer()
```

In [12]:
```
import numpy as np
np.set_printoptions(precision=2)

X_TFxIDF = tfidf_vectorizer.transform(X)
print(X_TFxIDF.toarray())
```

```
[[0.39 0.39 0.   0.28 0.28 0.28 0.39 0.39 0.39]
 [0.   0.   0.63 0.45 0.45 0.45 0.   0.   0.  ]]
```

Ensuite créez des vecteurs de mots en utilisant l'algorithme Word2vec. Comparez les résultats avec celles obtenus via la méthode bag of words.