Programmation Avancée : Python Projet de fin de module

PROF : MADANI Abdellah

REALISEE PAR : EL IRAOUI Amine

Partie 1 :

similarité entre deux documents en utilisant la distance euclidienne, le coefficient de Jaccard et la similarité cosinus

Etant donné un corpus (ensemble de documents textuels), effectuez les étapes suivantes :

- **préprocessing:**
  - lowercase
  - tokenizing
  - stopwords & ponctuations
  - stemming
- **TF ou TF-IDF**
- **Calcul de la similarité en utilisant les trois distances**

Les Imports

In [ ]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import string
import glob
from nltk import stem
import pandas as pd
import nltk
nltk.download()
```

```
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/inde
x.xml (https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml)
```

FONCTION : préprocessing

J'ai creer cette fonction pour résoudre :

- lowercase
- tokenizing
- stopwords & ponctuations
- stemming

In [ ]:

```
def préprocessing(nom_fichier):
    """cette fonction permet d'éviter les stopwords et permet de rendre le contenu d'un fic
        et permet de supprimer les ponctuation, et permet de limmatiser les tokens.
    """
    file=open(nom_fichier,"r")#read le fichier
    contenu = file.read()#read le contenu de fichier
    file.close()
    stop_words = set(stopwords.words('english'))#les stopwords
    mots = word_tokenize(contenu)#rendre le contenu sous forme des thermes
    liste_mots = []
    for word in mots:
        if word.lower() not in stop_words:#permet de transformer les thermes lowercase et e
            liste_mots.append(word)#liste des thermes lowercases et sans stopwords
    mots_final=[]
    for token in liste_mots:
        if token not in string.punctuation:#eliminer les ponctuations
            mots_final.append(token)
    for i in range(len(mots_final)):
        mots_final[i]=stem.WordNetLemmatizer().lemmatize(mots_final[i])#limatizer les mots
    return mots_final#retour d'un liste des thermes
```

Ensuite j ai creer une repertoire files qui contient une corpus des fichiers texts

puis, j ai fait appeler a la fonction préprocessing pour chaque fichier de notre corpus

In [ ]:

```
nom_fichiers = glob.glob('./files/corpus/*.txt')#les noms des fichiers de corpus
corpuslematizer = []#initialiser une liste pour les contenu des fichiers lematizer
for file in nom_fichiers:#chaque fichier de corpus
    chaine_limatizer=""
    doc_limatizer=préprocessing(file)#permet de rendre le fichier lowercase lowercase,token
    #et stemming
    for mot in doc_limatizer:
        chaine_limatizer=chaine_limatizer+" "+mot
    corpuslematizer.append(chaine_limatizer)
```

Ensuite la demande de la requete el l ajout dans le corpuslematizer

In [ ]:

```
req=input("donnez votre requete\n")   #la requete saisi par user
corpuslematizer.append(req)# on ajoute la requete dans le corpus
```

- **TF-IDF**

In [ ]:

```
vect =TfidfVectorizer()#remplir le DTM avec des nombres de fois qu'un token apparaît dans u
dtm = vect.fit_transform(corpuslematizer)
pd.DataFrame(dtm.toarray(), columns=vect.get_feature_names())
```

- **Calcul de la similarité en utilisant la similarité cosinus**

In [ ]:

```
for sim in cosine_similarity(dtm[len(corpuslematizer)-1],dtm)[0]:
    dict_similarite[i]=sim
    i=i+1
dict_items_simil = dict_similarite.items()
similarite_final_desc = sorted(dict_items_simil, key=lambda x: x[1],reverse=True)
similarite_final_desc.pop(0)
for s in similarite_final_desc:
    print("doc",s[0],"--->",nom_fichiers[s[0]],"---> similarite =",s[1])
```

- **Calcul de la similarité en utilisant la similarité de le coefficient de Jaccard**

In [ ]:

```
def coeficientJaccard(ch1,ch2):
    ch11=ch1.split(' ')
    ch22=ch2.split(' ')
    ch11=set(ch11)
    ch22=set(ch22)
    ch3 = ch11.intersection(ch22)
    ch4 = ch11.union(ch22)
    n=len(ch3)
    m=len(ch4)
    return (n/m)
    #s=[value for value in ch1 if value in ch2]
nom_fichiers = glob.glob('./files/corpus/*.txt')#les noms des fichiers de corpus
corpuslematizer = []#initialiser une liste pour les contenu des fichiers lematizer
for file in nom_fichiers:#chaque fichier de corpus
    chaine_limatizer=""
    doc_limatizer=préprocessing(file)#permet de rendre le fichier lowercase lowercase,token
    #et stemming
    for mot in doc_limatizer:
        chaine_limatizer=chaine_limatizer+" "+mot
    corpuslematizer.append(chaine_limatizer)
req=input("donnez votre requete\n")
i=0
for mot in corpuslematizer:
    print("doc",i,"--->",nom_fichiers[i],"---> similarite =",coeficientJaccard(req,mot))
    i=i+1
```

- **Calcul de la similarité en utilisant la similarité de la distance euclidienne**

In [ ]:

```
from sklearn.metrics.pairwise import euclidean_distances
for sim in euclidean_distances(dtm[len(corpuslematizer)-1],dtm)[0]:
    dict_similarite[i]=sim
    i=i+1
dict_items_simil = dict_similarite.items()
similarite_final_desc = sorted(dict_items_simil, key=lambda x: x[1],reverse=True)
similarite_final_desc.pop(0)
for s in similarite_final_desc:
    print("doc",s[0],"--->",nom_fichiers[s[0]],"---> similarite =",s[1])
```

Partie 2 :

Word Embedding : principes et implémentation

Principe

Word Embedding est une technique de modélisation utilisée pour mapper des mots sur des vecteurs de nombres réels. Il représente des mots ou des phrases dans un espace vectoriel à plusieurs dimensions. Les intégrations de mots peuvent être générées à l'aide de différentes méthodes telles que les réseaux de neurones, la matrice de cooccurrence, les modèles probabilistes, etc.

Parmis les types les plus utilisé de word embedding le Word2Vec est constitué de modèles permettant de générer l'intégration de mots. Ces modèles sont des réseaux neuronaux à deux couches peu profondes comportant une couche d entrée, une couche cachée et une couche de sortie.

Word2Vec utilise deux architectures:

v

- **CBOW (sac continu de mots) :**

  le modèle CBOW prédit le mot actuel en fonction du contexte dans une fenêtre spécifique. La couche d'entrée contient les mots de contexte et la couche de sortie contient le mot actuel. La couche masquée contient le nombre de dimensions dans lesquelles nous voulons représenter le mot actuel présent dans la couche en sortie.

- **Skip-gram :**

  Ignorer le gramme prédit les mots de contexte environnants dans une fenêtre donnée en fonction du mot actuel. La couche d'entrée contient le mot actuel et la couche de sortie contient les mots de contexte. La couche masquée contient le nombre de dimensions dans lesquelles nous voulons représenter le mot actuel présent dans la couche en entrée.

L'idée de base de l'incorporation de mots est que les mots qui apparaissent dans un contexte similaire ont tendance à être plus proches les uns des autres dans l'espace vectoriel. Pour générer des vecteurs de mots en Python, les modules nécessaires sont nltk et gensim.

In [ ]:

```
#Programme Python pour générer des vecteurs de mots à l'aide de Word2Vec
# importer tous les modules nécessaires
from nltk.tokenize import sent_tokenize, word_tokenize
import warnings
warnings.filterwarnings(action = 'ignore')
import gensim
from gensim.models import Word2Vec
#  lire le fichier ‘amin.txt’
sample = open("C:/Users/AM-EL/Desktop/amin.txt", "r")
s = sample.read()
# Remplace le caractère d'échappement par un espace
f = s.replace("\n", " ")
data = []
# parcourir toutes les phrases du fichier
for i in sent_tokenize(f):
    temp = []
    # tokenize la phrase en mots
    for j in word_tokenize(i):
        temp.append(j.lower())
    data.append(temp)
# Créer le modèle CBOW
model1 = gensim.models.Word2Vec(data, min_count = 1, size = 100, window = 5)

# Afficher les resultats
print("Cosine similarity between 'alice' " + "and 'wonderland' - CBOW : ", model1.similarit

print("Cosine similarity between 'alice' " +"and 'machines' - CBOW : ", model1.similarity('

# Créer le modèle Skip Gram
model2 = gensim.models.Word2Vec(data, min_count = 1, size = 100, window = 5, sg = 1)
# Afficher les resultats
print("Cosine similarity between 'alice' " +"and 'wonderland' - Skip Gram : ", model2.simil

print("Cosine similarity between 'alice' " +"and 'machines' - Skip Gram : ", model2.similar
```

Visualisation de Word Embedding

In [ ]:

```
from gensim.models import Word2Vec
from sklearn.decomposition import PCA
from matplotlib import pyplot
```

Après avoir appris l’incorporation de mots pour vos données textuelles, il peut être agréable de les explorer avec la visualisation.

Vous pouvez utiliser des méthodes de projection classiques pour réduire les vecteurs de mots de grande dimension à des graphiques en deux dimensions et les représenter sur un graphique.

Les visualisations peuvent fournir un diagnostic qualitatif pour votre modèle appris.

Nous pouvons récupérer tous les vecteurs d un modèle formé comme suit:

Avec CBOW MODEL:

In [ ]:

```
X = model1[model1.wv.vocab]
```

Nous pouvons ensuite former une méthode de projection sur les vecteurs, telles que celles proposées dans scikit-learn, puis utiliser matplotlib pour tracer la projection sous forme de diagramme de dispersion.

Tracer des vecteurs de mots à l'aide de PCA

Nous pouvons créer un modèle PCA en deux dimensions des vecteurs de mots à l aide de la classe scikit-learn PCA, comme suit.

In [ ]:

```
pca = PCA(n_components=2)
result = pca.fit_transform(X)
```

La projection résultante peut être tracée en utilisant matplotlib comme suit, en extrayant les deux dimensions en coordonnées x et y.

In [ ]:

```
pyplot.scatter(result[:, 0], result[:, 1])
```

Nous pouvons aller plus loin et annoter les points sur le graphique avec les mots eux-mêmes. Une version brute sans aucun décalage agréable ressemble à ce qui suit.

In [ ]:

```
words = list(model1.wv.vocab)
for i, word in enumerate(words):
    pyplot.annotate(word, xy=(result[i, 0], result[i, 1]))
```

Partie 3 :

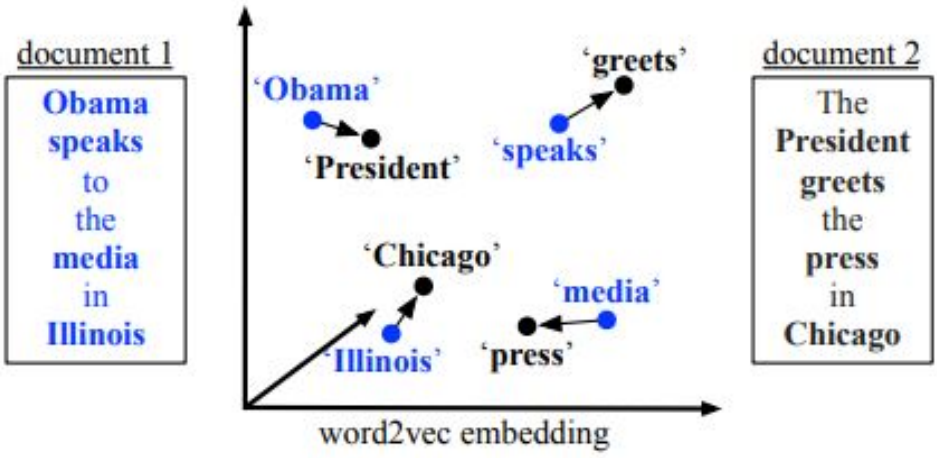
Word Embedding avec la distance WMD pour calculer la similarité entre deux documents

Dans le blog précédent, nous avons expliqué comment utiliser un moyen simple de rechercher la «similarité» entre deux documents (ou phrases). A cette époque, la distance euclidienne, la distance en cosinus et la similarité de Jaccard sont introduites, mais elles ont certaines limites. Les WMD sont conçues pour surmonter le problème des synonymes.

L'exemple typique est :

- Sentence 1: Obama speaks to the media in Illinois
- Sentence 2: The president greets the press in Chicago

À l'exception des mots vides, il n'y a pas de mots communs entre deux phrases, mais toutes deux abordent le même sujet (à ce moment-là).



In [ ]:

```
# Import et telecharger stopwords de NLTK.
from nltk.corpus import stopwords
from nltk import download
import gensim
from gensim.models import Word2Vec
download('stopwords')  # liste de stopwords.
model = Word2Vec.load_word2vec_format('https://s3.amazonaws.com/dl4j-distribution/GoogleNew
sentence_obama = 'Obama speaks to the media in Illinois'
sentence_president = 'The president greets the press in Chicago'
sentence_obama = sentence_obama.lower().split()
sentence_president = sentence_president.lower().split()
#Supprimer les stopwords.
stop_words = stopwords.words('english')
sentence_obama = [w for w in sentence_obama if w not in stop_words]
sentence_president = [w for w in sentence_president if w not in stop_words]
distance = model.wmdistance(sentence_obama, sentence_president)
print ('distance = %.4f' % distance)
```