idl_16 Système de recommandation de films

Pour ce TP, le fichier `fims.json` est fourni. Il contient des entrées de la forme suivante :

```
  {
    "Title": "The Shawshank Redemption",
    "Plot": "Two imprisoned men bond over a number of years, finding solace
and eventual redemption through acts of common decency."
  },
  {
    "Title": "The Godfather",
    "Plot": "The aging patriarch of an organized crime dynasty transfers
control of his clandestine empire to his reluctant son."
  },
```

Comme vous pouvez le constater les `plots` (résumés) sont très courts, mais nous allons voir dans ce TP ce qu'on peut déjà développer sur cette base.

I. Préparation du dataset

Étape 1 : lecture du jeu de données

Implémentez une fonction `json_to_dic()` qui prend en argument le nom d'un fichier `json` de la forme ci-dessus, et renvoie un dictionnaire dont les `clés` sont les titres de films, et les `valeurs` les résumés.

```
{'The Shawshank Redemption': 'Two imprisoned men bond over a number of
years, finding solace and eventual redemption through acts of common
decency.', 'The Godfather': 'The aging patriarch of an organized crime
dynasty transfers control of his clandestine empire to his reluctant son.',
[...]}
```

Étape 2 : chaîne de traitement en TAL

Nous devons effectuer un certain nombre de pré-traitements sur nos résumés

❓ quelles sont les étapes à réaliser ?

Implémentez une fonction `nlp` qui prend en argument une phrase et renvoie la liste des tokens pertinents pour la recommandation.

```
sentence = "Two imprisoned men bond over a number of years, finding solace
and eventual redemption through acts of common decency"

print(nlp(sentence))
```

```
['Two', 'imprison', 'man', 'bind', 'number', 'year', 'find', 'solace',
'eventual', 'redemption', 'act', 'common', 'decency']
```

Vous vous appuierez pour cela sur des librairies existantes.

Étape 3 : application de la chaîne de traitement au dictionnaire

Implémentez une fonction `nlp_dico` qui prend en argument un dictionnaire et renvoie un dictionnaire où le traitement ci-dessus a été appliqué à chacune des valeurs.

II. Calcul de la similarité de Jaccard

Implémentez une fonction `jaccard_similarity` qui prend en argument deux listes de tokens et renvoie la valeur flottante correspondant au coefficient de similarité entre ces deux listes.

Implémentez une fonction `jaccard_similarity_from_title` qui prend en argument deux titres de films et qui renvoie le coefficient de similarité entre les deux résumés correspondants, et affiche le résultat sous la forme :

```
le Score de simiarité de Jaccard des films `The Godfather` et `The
Godfather: Part II` est 0.1111111111111111
```

III. Calcul de la représentation vectorielle

Nous allons ici construire une représentation vectorielle de notre corpus basée sur le calcul des coefficients dits `TD-IDF`.

❓ quelle doit-être la taille des vecteurs représentant chaque document de notre corpus ? Cette valeur peut-elle être obtenue en aditionnant toutes les longueurs des valeurs du dictionnaire obtenu à l'étape 3 du I ?

Etape 1 : Calcul de TF et d'IDF

- Implémentez une fonction `TF` qui calcule, pour un terme donné, la fréquence de ce terme dans un résumé de film donné.

$$ TF(token, document) = { Nb_occurrences_token \over Nb_total_tokens} $$

- Implémentez une fonction `IDF` qui prend en argument un dictionnaire (clé : titre, valeur : résumé) et qui renvoie un dictionnaire dont les clés sont les tokens du vocabulaire de l'ensemble du corpus, et les valeurs sont le score idf pour chacun de ces tokens. Vous pourrez utiliser la fonction `log10` de la librairie `maths`.

$$ IDF(token, corpus) = log10({ Nb_docs_du_corpus \over Nb_docs_contenant_token}) $$

Exemple de résultat :

```
{'Two': 1.6989700043360187,
 'imprison': 2.0969100130080562,
 'man': 1.7958800173440752,
 'bind': 2.0969100130080562,
 'number': 2.0969100130080562,
 etc.
}
```

Étape 2 : Calcul des vecteurs

Implémentez une fonction `TFIDF` qui prend en argument le dictionnaire des films et qui renvoie un dictionnaire dont les clés sont le titre des films et les valeurs sont le vecteur de coefficients TF-IDF correspondants à ce film.

$$ TFIDF(token, document, corpus) = TF(token, corpus) * IDF(token, corpus) $$ Exemple de résultat :

```
{'The Shawshank Redemption':
    [
      0.07079041684733411, // token Two
      0.08737125054200234, // token imprison
      0.07482833405600313, // token man
      0.08737125054200234, // token bind
      etc.
    ],
    etc.
}
```

IV. Recommandation

Dans cette dernière section, on considère que le profil d'un.e utilisateurice se résume à un titre de film qu'il/elle dit avoir aimé.

Implémentez-donc deux fonctions `recommend_Jaccard` et `recommend_TFIDF` qui, pour un titre de film donné, recommandent 3 films.

On utilisera pour calculer la similarité entre deux films (dans le cadre de la représentation vectorielle) la fonction `cosine_similarity` ci-dessous. Cette fonction requiert l'import suivant `import numpy as np`.

```
def cosine_similarity(list1, list2):
    dot = np.dot(list1, list2)
    norm1 = np.linalg.norm(list1)
    norm2 = np.linalg.norm(list2)
    cos = dot / (norm1 * norm2)
    return(cos)
```

Exemple de résultat :

```
Les recommandations basées sur la mesure de similarité de Jaccard pour le
film "The Nightmare Before Christmas" sont :
[('Finding Nemo', 0.07692307692307693), ('A Christmas Story',
0.06451612903225806), ("Hachi: A Dog's Tale", 0.047619047619047616)]

Les recommandations basées sur la distance cosinus et sur TF-IDF pour le
film "The Nightmare Before Christmas" sont :
[('A Christmas Story', 0.10757874998175111), ('Life of Brian',
0.09473294779049923), ('Finding Nemo', 0.09460119447812901)]
```