RAPPORT DU PROJET
MACHINE LEARNING NLP

LE FOURN CONSTANCE

FOUZARI SANDRA

A5-DIA1

INTRODUCTION

Le présent rapport détaille le développement d'un système original de recherche d'information sur un corpus spécifique appelé NFCorpus. Ce corpus médical est constitué d'abstracts de publications médicales provenant de PubMed, accompagnés de requêtes vulgarisées liées aux sujets de certains articles de PubMed. La complexité de ce corpus nécessite des approches sophistiquées, et il a été observé que les approches de deep learning modernes ne surpassent pas significativement le modèle traditionnel BM25

L'objectif principal de ce projet est de concevoir un système de recherche d'information innovant sur le NFCorpus. Pour ce faire, nous sommes autorisés à utiliser tout type de prétraitement et à manipuler le vocabulaire des documents. L'utilisation de modèles Word2Vec pré-entrainés ou l'apprentissage de de modèles Word2Vec est priorisée.

Le modèle BM25 est établi comme référence de base, et notre objectif est d'améliorer ses performances. Ce rapport sert de documentation détaillée pour expliquer les démarches entreprises et le code implémenté. Il vise à décrire comment relier de manière efficace les résumés de documents avec les documents scientifiques, en mettant en perspective les différentes méthodes explorées et les résultats obtenus.

Le notebook NFCorpusBM25.ipynb, disponible à cette adresse : https://colab.research.google.com/drive/1zOsx2n_JdHAwvtLRNdsaaShoVIr5yCLq?usp=sharing , permet de charger le jeu de données NFCorpus, d'entraîner un modèle BM25, et enfin, d'évaluer ses performances à l'aide de la métrique NDCG. Notre rapport s'inscrit dans une démarche comparative avec ce code de référence sur BM25, visant à présenter et expliquer les démarches prises pour améliorer la corrélation entre les résumés de documents et les documents scientifiques de manière optimale.

I. Importation des données

```
import urllib.request as re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import numpy as np
from collections import defaultdict
import nltk
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
nltk.download('stopwords')
nltk.download('punkt')
```

Source : Obtenue à partir de jupyter 'Projet_Essai_3.ipynb'

Les lignes ci-dessus sont des commandes utilisées pour télécharger des ressources linguistiques spécifiques à la bibliothèque Natural Language Toolkit en python. Le modèle 'stopwords' télécharge des mots qui n'ont pas beaucoup de signification contextuelle tel que « et », « ou », « le ». Le modèle 'punkt' télécharge est utilisé pour la tokenisation des mots en anglais.

```
#Code prof Import des 3 fichiers

def loadNFCorpus():
    dir = "./project1-2023/"
    filename = dir +"dev.docs"

    dicDoc={}
    with open(filename) as file:
        lines = file.readlines()
    for line in lines:
        tabLine = line.split('\t')
        #print(tabLine)
        key = tabLine[0]
        value = tabLine[1]
        #print(value)
        dicDoc[key] = value
    filename = dir + "dev.all.queries"
    dicReq={}
    with open(filename) as file:
        lines = file.readlines()
    for line in lines:
        tabLine = line.split('\t')
        key = tabLine[0]
        value = tabLine[1]
        dicReq[key] = value
    filename = dir + "dev.2-1-0.qrel"
    dicReqDoc=defaultdict(dict)
    with open(filename) as file:
        lines = file.readlines()
    for line in lines:
        tabLine = line.strip().split('\t')
        req = tabLine[0]
        doc = tabLine[2]
        score = int(tabLine[3])
        dicReqDoc[req][doc]=score

    return dicDoc, dicReq, dicReqDoc
```

```
dicDoc, dicReq, dicReqDoc = loadNFCorpus()
```

Source : Obtenue à partir de jupyter 'Projet_Essai_3. ipynb'

La fonction ci-dessus, permet d'importer le corpus que l'on souhaite traiter :

- Chaque ligne de ce fichier est formatée avec une clé et une valeur séparée par une tabulation. La clé est extraite de la première partie de la ligne (tabLine[0]), et la valeur est extraite de la deuxième partie (tabLine[1]). Ces paires clé-valeur sont stockées dans le dictionnaire dicDoc.
- Le chargement des requête est ensuite effectuée à partir du fichier « dev.all.queries». De manière similaire aux documents, chaque ligne semble contenir une clé et une valeur séparées par une tabulation. Les paires clé-valeur sont stockées dans le dictionnaire « dicReq ».
- La suite du code consiste à charger des associations entre requêtes et documents, ainsi que les scores associés. Chaque ligne semble contenir des informations sur une requête, un document et un score. Ces informations sont stockées dans le dictionnaire dicReqDoc. Les clés de ce dictionnaire sont les requêtes, et les valeurs sont des dictionnaires où les clés sont les documents associés et les valeurs sont les scores.
- Enfin, la fonction retourne les trois dictionnaires « dicDoc », « dicReq », et « dicReqDoc ».

II. Mise en Forme de la donnée

A. Formatage des mots

```
# Mise en forme des article, enlèvement des mots inutiles
def text2TokenList(text):
    stopword = stopwords.words('english')
    #print("LEN DE STOPWORD=",len(stopword))
    word_tokens = word_tokenize(text.lower())
    word_tokens_without_stops = [word for word in word_tokens if word not in stopword and len(word)>2]

    return word_tokens_without_stops
```

Source : Obtenue à partir de jupyter 'Projet_Essai_3. ipynb'

La première étape de mise en forme, consiste à rendre le texte plus adapté à certaines tâches d'analyse en convertissant l'ensemble du texte en minuscule. Ensuite, on effectue une tokenization qui consiste à diviser le texte en unités plus petites appelées "tokens", qui peuvent être des mots, des phrases, ou d'autres éléments significatifs. Cela est effectué avec « word_tokenize(text.lower()) », qui renvoie une liste de mots.

```
docsToKeep=[]
reqsToKeep=[]
dicReqDocToKeep=defaultdict(dict)

#Construction du dictionnaire de Req à garder
i=0
for reqId in dicReqDoc:
    if i >= len(dicDoc) :  #nbDocsToKeep:
        break
    for docId in dicReqDoc[reqId]:
        dicReqDocToKeep[reqId][docId] = dicReqDoc[reqId][docId]
        docsToKeep.append(docId)
        i = i + 1
    reqsToKeep.append(reqId)
docsToKeep = list(set(docsToKeep))
```

Source : Obtenue à partir de jupyter 'Projet_Essai_3.ipynb'

Le code ci-dessus crée un dictionnaire rassemblant les requêtes et documents avec la valeur qui leurs a été attribué dans le dicReqDoc. Dans ce même temps, docsToKeep va lister tous les noms de documents présents dans le dictionnaire dicReqDoc et ceux même s'ils sont associés à plusieurs requêtes. Une liste similaire sera complétée pour les requêtes. Au cas où les nombre de documents associé aux requêtes est supérieur au nombre de documents existants, le processus s'arrête. En d'autres termes, on limite la sélection pour s'assurer qu'on ne prend pas plus de documents que ceux présents dans le dictionnaire « dicDoc ».

B. Construction des corpus MED et PLAIN

```
allVocab ={}
#Construction corpus des MED
for k in docsToKeep: #Pour chaque MED
    docTokenList = text2TokenList(dicDoc[k]) #Transforme les texte de MED en liste
    #print(docTokenList)
    for word in docTokenList: #Pour chaque mot dans la liste
        if word not in allVocab: #S'il n'y est pas déjà
            allVocab[word] = word #l'ajoute dans le dictionnaire

allVocabListDoc = list(allVocab)
#print("doc vocab=",allVocabListDoc)


allVocab ={}

#Construction corpus des PLAIN
for k in reqsToKeep: #Pour chaque PLAIN
    docTokenList = text2TokenList(dicReq[k]) # Transforme les texte de PLAIN en liste
    #print(docTokenList)
    for word in docTokenList: # Pour chaque mot dans la liste
        if word not in allVocab: #S'il n'y est pas déjà
            allVocab[word] = word #l'ajoute dans le dictionnaire
allVocabListReq = list(allVocab)
```

Source : Obtenue à partir de jupyter 'Projet_Essai_3.ipynb'

Ce code construit un vocabulaire global à partir de deux corpus distincts : le corpus des MED (documents médicaux) et le corpus des PLAIN (résumé des documents médicaux). L'objectif des algorithmes sera de vérifier que PLAIN qui sont des résumés d'articles sont bien reliés à MED (les articles médicaux).

Avant de commencer à exécuter des modèles, nous avons donc :

- Une liste de tous les noms de MED « docsToKeep »
- Une liste de tous les mots dans les MED sans distinction sans doublons « allVocabListDoc »
- Une liste de tous les mots dans les PLAIN sans distinction sans doublons « allVocabListReq »
- Une liste de tous les mots dans les MED sans distinction avec doublons « corpusDocTokenList »,
- La numérotation de chaque MED « corpusDicoDocName »
- Une liste de tous les mots dans les PLAIN sans distinction avec doublons « corpusReqTokenList »
- Une liste de tous les noms de PLAIN « corpusReqName »
- La numérotation de chaque PLAIN « corpusDicoReqName »

III. Les tests des algorithmes sur les corpus

A. Les tentatives et échecs

```
%%time
TableTFIDF=pd.DataFrame(columns=['MED','PLAIN','score'])
c=0
CumulTFIDF=0
for plain in corpusReqTokenList: #Pour chaque mot dans le plain avec doublons
    j=0
    reqTokenList = corpusReqTokenList[plain]  #liste de mots de PLAIN n°req
    #print(req)
    #print(len(doc_scores)) #nombre de mot dans chaque PLAIN ?
    trueDocs = np.zeros(len(corpusDocTokenList)) #Ressort un array de la forme de corpusDocTokenList (1779,)
    for med in corpusDicoDocName: #pour chaque nom de MED
        if med in dicReqDocToKeep[plain]: # et si MED n°docId est dans les recommendation de PLAIN
            vectorizer=TfidfVectorizer() # Déclare un nouveau vecteur
            pos=corpusDicoDocName[med] # Recupère la position du MED
            textMED= ' '.join(corpusDocTokenList[med]) # Recrée un cropus str à partir de la liste de MED
            textPLAIN=' '.join(corpusReqTokenList[plain]) # Recrée un cropus str à partir de la liste de PLAIN
            score=vectorizer.fit_transform([textMED,textPLAIN]) # Cherche le score TF IDF des MED et PLAIN associés
            trueDocs[corpusDicoDocName[med]] = score.mean() # Somme des moyennes de TFIDF de MED pour PLAIN n°plain
    CumulTFIDF=CumulTFIDF + score.mean()
    c = c + 1
moytfidf=CumulTFIDF/c
print(moytfidf)
```

```
0.03620726767190228
Wall time: 17.2 s
```

Source : Obtenue à partir de Jupyter 'Projet_Essai_3.ipynb'

L'objectif est de relier les résumés (PLAIN) et les documents médicaux correspondant (MED). Le score TF-IDF est utilisé comme mesure de cette similarité, ce qui permet de quantifier à quel point les termes d'un résumé sont similaires à ceux d'un document médical. Il s'agit de la moyenne des scores est calculée pour chaque résumé et pour l'ensemble des résumés. Ce processus donne une idée de la pertinence des résumés par rapport aux corpus médicaux recommandés. Le résultat du score est de 0.036 ce qui est très faible (étant donné que l'on cherche un score supérieur à 0.43, qui est le résultat obtenu dans le lien de référence collab).

```
def modelWord2vec(corpus):
    model = word2vec.Word2Vec(sentences=corpus, vector_size=1000, window=5, min_count=1, workers=4)
    corpus_embeddings = []
    for corp in corpus:
        embeddings = [model.wv[word] for word in corp if word in model.wv]
        corpus_embeddings.append(sum(embeddings) / len(embeddings))
    return corpus_embeddings
```

Source : Obtenue à partir de Jupyter 'Projet_1_Word2vec.ipynb'

La fonction ci-dessous, sert à entraîner un modèle Word2Vec sur la liste des tokens de chaque document médical dans le corpus. Les embeddings générés par ce modèle sont utilisés comme représentations vectorielles pour chaque document médical. Ces embeddings sont ensuite utilisés comme entrée pour l'indice BM25.

Le modèle BM25 avec les embeddings Word2Vec sont utilisés pour évaluer la similitude entre les termes de la requête et les termes des documents médicaux. Cela contribue à améliorer la précision de la recherche d'information car les embeddings capturent des informations sémantiques sur les

mots, permettant au modèle de prendre en compte la signification des mots plutôt que de simplement compter les occurrences.

En résumé, l'utilisation de Word2Vec dans ce modèle vise à améliorer la représentation des documents médicaux, ce qui peut conduire à des scores de similarité plus précis lors de l'application de l'indice BM25 pour la recherche d'information.

```
def run_bm25_only(startDoc,endDoc):

    dicDoc, dicReq, dicReqDoc = loadNFCorpus()
    #print(dicReqDoc)

    docsToKeep=[]
    reqsToKeep=[]
    dicReqDocToKeep=defaultdict(dict)

    #150
    ndcgTop=10
    print("ndcgTop=",ndcgTop,"nbDocsToKeep=",endDoc - startDoc)
```

Source : Obtenue à partir de Jupyter 'Projet_1_Word2vec.ipynb'

```
from sklearn.metrics import ndcg_score
for req in corpusReqTokenList: #Pour chaque mot dans le plain avec doublons
    j=0
    reqTokenList = corpusReqTokenList[req]  #liste de mots de PLAIN n°req
    doc_scores = bm25.get_scores(reqTokenList) #Computes and returns BM25 scores in relation to every item in corpus.
    #print(req)
    #print(len(doc_scores)) #nombre de mot dans chaque PLAIN ?
    trueDocs = np.zeros(len(corpusDocTokenList)) #Ressort un array de la forme de corpusDocTokenList (1779,)

    for docId in corpusDicoDocName: #pour chaque nom de MED
        if req in dicReqDocToKeep: #si PLAIN n°req est dans les recommendation
            if docId in dicReqDocToKeep[req]: # et si MED n°docId est dans les recommendation de PLAIN
                #get position docId
                posDocId = corpusDicoDocName[docId] #ressort le numéro du MED n°docId
                #print(posDocId)
                #print(req)
                trueDocs[posDocId] = dicReqDocToKeep[req][docId] #arret de la position du numéro de MED est le score déjà donné dans reqdoc ?!
                #print("TOKEEP=",docId)
                #print(trueDocs)
    ndcgBM25Cumul = ndcgBM25Cumul + ndcg_score([trueDocs], [doc_scores],k=ndcgTop)
    nbReq = nbReq + 1
ndcgBM25Cumul = ndcgBM25Cumul / nbReq
print("ndcg bm25=",ndcgBM25Cumul)
return ndcgBM25Cumul
```

Source : Obtenue à partir de Jupyter 'Projet_1_Word2vec.ipynb'

Le modèle BM25 est utilisé pour mesurer la similarité entre les termes des résumés et des documents, le calcul du score NDCG quant à lui évalue la qualité des recommandations. NDCG mesure donc la pertinence des résumés recommandés par rapport aux résumés réels associés aux documents médicaux. Le score est ensuite affiché.

La valeur « ndcgTop=10 » dans le code correspond au paramètre k dans la métrique NDCG (Normalized Discounted Cumulative Gain). Le paramètre k spécifie le nombre de documents les plus

pertinents à considérer lors du calcul de la métrique. En d'autres termes, c'est le nombre de documents pris en compte dans le calcul de la pertinence cumulative normalisée et actualisée.

Cependant, avec ces paramètres, le score est beaucoup trop faible comme on peut le voir ci-dessous :

```
nb_docs = 3192 #all docs
#nb_docs = 150 #for tests
cumul=run_bm25_only(0,nb_docs)
```

```
ndcgTop= 10 nbDocsToKeep= 3192
bm25 doc indexing...
ndcg bm25= 0.018439409472351828
```

Source : Obtenue à partir de Jupyter 'Projet_1_Word2vec.ipynb'

L'objectif ici est donc de jouer avec les paramètres d'entrée du modèle BM25 pour obtenir un résultat optimal.

IV. Le score et le test final

```
: nb_docs = 3192 #all docs
  #nb_docs = 150 #for tests
  cumul=run_bm25_only(0,nb_docs)
```

```
0.20210734650054754
0.5087403079104241
0.0
0.5
0.8687949224876581
0.2139862647345275
0.0
0.2463023887407299
0.639945385422766
0.639945385422766
0.5147714448836773
0.3391602052736161
0.38356636737133554
0.5413996682199069
0.0
0.0
0.9999999999999999
ndcg bm25= 0.46335714418774215
TFIDF :  0.03647733911334044
```

Source : Obtenue à partir de Jupyter 'Projet_1_code_prof_nbcgTop=5.ipynb'

Après quelques essais, le score le plus fructueux obtenu est celui avec un ndcg_top = 5 avec un score de 0.46 et sans utiliser la méthode Word2Vec qui semble jusqu'ici peu fructueuse sur nos données.

Conclusion :

L'objectif principal de ce projet est d'établir une corrélation précise entre les résumés et les documents scientifiques. La première étape cruciale consiste à formater les données de manière à faciliter l'application ultérieure des algorithmes. Cela implique la suppression de mots qui ne sont pas considérés comme des "mots clés" et qui, par conséquent, n'apportent pas une grande utilité à la tâche. De plus, la mise en minuscules de tous les mots est effectuée afin de traiter les termes de manière équivalente.

Dans la suite de l'analyse, deux corpus distincts, PLAIN et MED, sont créés pour l'application de différents modèles. Cependant, malgré l'utilisation initiale de la mesure TF-IDF, le score obtenu est inférieur à celui utilisé comme référence (<0.43), justifiant ainsi son rejet. En cherchant une approche alternative, le modèle BM25 combiné à un calcul de score NDCG est adopté.

Néanmoins, avec un paramètre d'initialisation de "ndcgTop=10", le score demeure faible (0.018). Pour optimiser l'algorithme, des ajustements sont effectués sur ce paramètre. Après plusieurs essais, le meilleur résultat est obtenu avec "ndcgTop=5", générant un score significativement amélioré de 0.46.

En conclusion, la démarche de formatage des données et l'exploration des modèles, en particulier le passage du TF-IDF au BM25 avec un ajustement du paramètre NDCG, ont abouti à des résultats prometteurs pour la tâche de relier les résumés aux documents scientifiques.

Afin d'améliorer les scores, nous aurions aussi pu utiliser un model plus adapté tel que Sent2Vec ou Bio2Vec mais les fichiers pour importer c'est algorithme étaient trop lourd pour nos espaces de stockage.

Vous pourrez retrouver nos travaux de rechercher sur le GitHub suivants : Const679/NLP: Projet1-NLP (github.com)

Bibliographie :

https://colab.research.google.com/drive/1zOsx2n_JdHAwvtLRNdsaaShoVIr5yCLq?usp=sharing : lien collab du code de référence

https://github.com/cr-nlp/project1-2023/blob/main/dev.all.queries : liens des corpus PLAIN

https://github.com/cr-nlp/project1-2023/blob/main/dev.docs_/ : Liens des corpus MED

https://github.com/cr-nlp/project1-2023/blob/main/dev.2-1-0.qrel : lien du dictionnaire reliant les corpus MAIN au corpus MED