UNIVERSITÉ NATIONALE DU VIETNAM
INSTITUT FRANCOPHONE INTERNATIONAL

FOUILLE DE DONNÉES

RAPPORT DES TRAVAUX PRATIQUES SUR LA RECHERCHE D'INFORMATION

Jeu de données : **"stories"**

Octobre 2019

Étudiants du Groupe 9 :

Mike Arley MORIN

Afi Elolo Gisèle DEKPE

Professeur : **Nguyen Thi Minh Huyen**

P23 RSC, Année Académique : 2019 - 2020

Table des matières

Table des figures

Introduction

La fouille de texte est une approche analytique de texte, adaptée et utilisée dans un large nombre de domaine d'activités. C'est une discipline qui vise à extraire les informations pertinentes d'un grand ensemble de documents. Dans le but de nous entraîner à l'utilisation des techniques de recherche d'informations, dont l'objectif est de concevoir des systèmes capables de retrouver parmi un ensemble de documents, ceux qui répondent précisément au besoin d'un utilisateur, il nous a été donné un travail pratique. Ce document fait office du rapport de notre travail. Dans les prochaines lignes, nous allons décrire les étapes parcourues pour accéder au résultat final.

1 Le jeu de données

Un jeu de données nous a été donné dans le tutoriel qui nous a été demandé de suivre. Ce jeu de données est titré "**stories**" et comporte un lot de documents de divers formats contenant des archives d'histoires. Il est disponible ici : http ://archives.textfiles.com/stories.zip.

2 Méthode, Outil et Langage utilisés

2.1 La méthode utilisée : TF-IDF

TF-IDF qui signifie "**Term Frequency — Inverse Document Frequency**" est une méthode ou technique pour quantifier un mot dans les documents, nous calculons généralement un poids pour chaque mot qui signifie l'importance du mot dans le document et le corpus. Cette méthode est une technique largement utilisée dans la recherche d'information et l'exploration de texte.

2.2 Outil utilisé : Jupyter Notebook

Jupyter Notebook est une application Web open-source qui permet de créer et de partager du code interactif, des visualisations et bien d'autres. Cet outil peut être utilisé avec plusieurs langages de programmation, notamment Python, R et Ruby, etc... Il est souvent utilisé pour travailler avec des données, la modélisation statistique et l'apprentissage automatique.
Le langage utilisé est Python.

3 La chaîne complète de traitements

Dans cette section, nous allons décrire les étapes à suivre pour aboutir au résultat final.

3.1 Analyse du jeu de données

La première chose à faire dans tout traitement de données surtout dans l'apprentissage automatique est d'analyser les données. Notre première remarque est tous les documents comportent des mots en anglais. Cela nous amène à identifier le titre et le corps du document. La seconde remarque est que le titre dans chaque document est à un emplacement différent. Mais la plupart des titres sont centrés. Nous pouvons maintenant passer à l'extraction du titre et du corps des textes.

3.2 Extraction des titres et des noms de fichiers

3.2.1 Sélection des dossiers

Avant de commencer à extraire les titres et les noms de fichiers, nous allons explorer les dossiers pour pouvoir lire plus tard leur contenu.

```
Entrée [4]: folders = [x[0] for x in os.walk(str(os.getcwd())+'/Documents/M1/ModuleFD/TP/'+'stories')]

Entrée [7]: folders

    Out[7]: ['/home/gisele/Documents/M1/ModuleFD/TP/stories/',
             '/home/gisele/Documents/M1/ModuleFD/TP/stories/SRE',
             '/home/gisele/Documents/M1/ModuleFD/TP/stories/FARNON']

Entrée [8]: folders[0] = folders[0][:len(folders[0])-1]

Entrée [9]: folders

    Out[9]: ['/home/gisele/Documents/M1/ModuleFD/TP/stories',
             '/home/gisele/Documents/M1/ModuleFD/TP/stories/SRE',
             '/home/gisele/Documents/M1/ModuleFD/TP/stories/FARNON']
```

FIGURE 1 – Chargement du jeu de données

3.2.2 Collecte des titres et des noms de fichiers

Les fichiers **index.html** contenus dans les dossiers comportent les titres que nous recherchons. Nous allons les parcourir pour en extraire les titres. Pour ce faire, nous procédons par l'extraction des titres. Ceci rendra le travail un peu plus facile.
Étant donné que les noms des fichiers sont contenus dans des balises html du genre $><A$

HREF=" " et les titres dans *
<TD> n*, nous allons pouvoir facilement avoir recupérer des index et ainsi obtenir le titre et le nom du fichier dans tous les fichiers index.html.

Entrée [10]:
```
dataset = []

c = False

for i in folders:
    file = open(i+"/index.html", 'r')
    text = file.read().strip()
    file.close()

    file_name = re.findall('><A HREF="(.*)">', text)
    file_title = re.findall('<BR><TD> (.*)\n', text)

    if c == False:
        file_name = file_name[2:]
        c = True

    print(len(file_name), len(file_title))

    for j in range(len(file_name)):
        dataset.append((str(i) +"/"+ str(file_name[j]), file_title[j]))
```

```
452 452
15 15
0 0
```

FIGURE 2 – Collecte des titres et des noms de fichiers

Ce bout de code prépare les index de l'ensemble de données et supprime le dossier racine index.html car il contient aussi des dossiers et ses liens, qui ne seront pas utile à notre travail.

3.3 Pré-traitement des données

Le pré-traitement est une étape importante dans le traitement de n'importe quel type de modèle de texte. Ici, nous vérifions la distribution de nos données, quelles techniques sont nécessaires et à quelle profondeur nous devons épurer notre jeu de données. La plupart du temps, cette phase consiste à la conversion en minuscules, la suppression de la ponctuation, la suppression des mots vides et la lemmatisation / radicalation.
Les captures suivantes montrent les pré-traitements que nous avons effectués sur les données.

3.3.1 Conversion en minuscules

La conversion en minuscules est une étape de pré-traitement obligatoire car en raison du caractère encodant, deux mots "**Cette**" et "**cette**" sont considérés comme des symboles différents. La méthode suivante permet de faire ce traitement :

```
Entrée [11]: def convert_lower_case(data):
                 return np.char.lower(data)
```

FIGURE 3 – Conversion en minuscules

3.3.2 Les stop-words

Ce sont des mots qui n'ajoute aucune valeur supplémentaire au vecteur de document. De ce fait, leur suppression diminuera le temps de calcul et augmente l'efficacité de l'espace. La méthode suivante permet de faire ce traitement :

```
Entrée [12]: def remove_stop_words(data):
                 stop_words = stopwords.words('english')
                 words = word_tokenize(str(data))
                 new_text = ""
                 for w in words:
                     if w not in stop_words and len(w) > 1:
                         new_text = new_text + " " + w
                 return new_text
```

FIGURE 4 – Suppression des stop-words

3.3.3 Les ponctuations

Les signes de ponctuation sont les symboles majoritairement inutiles sauf pour les cas où ils apportent un sens à un mot. Nous stockons tous nos symboles dans une variable et itérer cette variable en supprimant ce symbole particulier dans l'ensemble de données. La méthode suivante permet de faire ce traitement :

```
Entrée [13]: def remove_punctuation(data):
                 symbols = "!\"#$%&()*+-./:;<=>?@[\]^_`{|}~\n"
                 for i in range(len(symbols)):
                     data = np.char.replace(data, symbols[i], ' ')
                     data = np.char.replace(data, "  ", " ")
                 data = np.char.replace(data, ',', '')
                 return data
```

FIGURE 5 – Suppression des ponctuations

3.3.4 Les apostrophes

les apostrophes ne sont pas considéré comme des caractères de ponctuation. Ils sont donc traités séparément.

```
Entrée [14]: def remove_apostrophe(data):
                 return np.char.replace(data, "'", "")
```

FIGURE 6 – Suppression des apostrophes

3.3.5 Le stemming

Le stemming est une technique qui réduit le mot essentiellement à sa racine. Nous allons l'appliquer à notre jeu de données. Rappelons qu'il existe également l'opération de Lemmatisation mais nous n'allons pas l'utiliser dans notre cas.

```
Entrée [15]: def stemming(data):
                 stemmer= PorterStemmer()

                 tokens = word_tokenize(str(data))
                 new_text = ""
                 for w in tokens:
                     new_text = new_text + " " + stemmer.stem(w)
                 return new_text
```

FIGURE 7 – Stemming

3.3.6 La conversion des nombres en lettres

La librairie **num2words** sera utilisée afin de traduire le nombres en lettres. Nous faisons ceci parce que le modèle de recherche d'informations traite de façon différente les nombres en chiffres et les nombres en lettres.

Entrée [16]:

```
def convert_numbers(data):
    tokens = word_tokenize(str(data))
    new_text = ""
    for w in tokens:
        try:
            w = num2words(int(w))
        except:
            a = 0
        new_text = new_text + " " + w
    new_text = np.char.replace(new_text, "-", " ")
    return new_text
```

FIGURE 8 – La conversion des nombres en lettres

3.3.7 Exécution des méthodes de pré-traitement

Pour finir cette phase de pré-traitement, nous appliquons toutes ces méthodes dans une méthode globale qui les prendra en compte.

Entrée [17]:

```
def preprocess(data):
    data = convert_lower_case(data)
    data = remove_punctuation(data)
    data = remove_apostrophe(data)
    data = remove_stop_words(data)
    data = convert_numbers(data)
    data = stemming(data)
    data = remove_punctuation(data)
    data = convert_numbers(data)
    data = stemming(data)
    data = remove_punctuation(data)
    data = remove_stop_words(data)
    return data
```

FIGURE 9 – Exécution des méthodes de pré-traitement

Remarquons qu'après la conversion de nombre en lettres, il y a eu une seconde application des méthodes **stemming**, **remove_punctuation** et **remove_stop_words**. Ceci est fait parce que la conversion de nombre en lettres peut également regénérer des ponctuation, des stop-words et il va falloir trouver la racine des mots générés.

3.4 Calcul du TF-IDF

Ceci se fait grâce à la formule : **TF-IDF = Term Frequency (TF) * Inverse Document Frequency (IDF)** ; où **TF** est la fréquence d'un mot ou d'un terme dans un document et **DF** est l'unité de mesure de l'importance d'un document dans un ensemble donné ou encore le nombre de documents dans lequel le mot est présent. D'où **IDF** étant l'inverse du DF est considéré comme l'informativité d'un terme.
Pour passer aux différents calculs permettant de réaliser cette opération, nous allons d'abord récupérer la taille du jeu de données épuré puis l'utiliser pour extraire les données :

```
Entrée [19]: N = len(dataset)
```

FIGURE 10 – Récupération de la taille du jeu de données

```
Entrée [32]: processed_text = []
             processed_title = []

             for i in dataset[:N]:
                 file = open(i[0], 'r', encoding="utf8", errors='ignore')
                 text = file.read().strip()
                 file.close()

                 processed_text.append(word_tokenize(str(preprocess(text))))
                 processed_title.append(word_tokenize(str(preprocess(i[1]))))
```

FIGURE 11 – Extraction de données

3.4.1 Calcul du DF

Voici ci-dessous la procédure pour le cacul du DF :

```
Entrée [33]: DF = {}

for i in range(N):
    tokens = processed_text[i]
    for w in tokens:
        try:
            DF[w].add(i)
        except:
            DF[w] = {i}

    tokens = processed_title[i]
    for w in tokens:
        try:
            DF[w].add(i)
        except:
            DF[w] = {i}
for i in DF:
    DF[i] = len(DF[i])
```

FIGURE 12 – Calcul du DF

Nous pouvons maintenant afficher la taille du DF ainsi que les mots qu'il contient.

```
Entrée [34]: total_vocab_size = len(DF)

Entrée [31]: total_vocab_size

    Out[31]: 32350

Entrée [35]: total_vocab = [x for x in DF]

Entrée [36]: print(total_vocab[:20])

['unsupport', 'josiah', 'mutter', 'criteria', 'interpret', 'mannikin', 'marin', 'vengeanc', 'cept', 'fucker', 'for
g', 'bothnia', 'inspir', 'molest', 'wheez', 'umpir', 'starless', 'destwd', 'declin', 'olden']
```

Nous rappelons que nous devons maintenir des poids différents pour le titre et le body. Pour calculer le TF-IDF du body ou du titre, nous devons considérer à la fois le titre et le body. Nous avons juste besoin de parcourir tous les documents, nous pouvons utiliser **Coutner** qui peut nous donner la fréquence des jetons, calculer TF et IDF et enfin stocker en tant que paire (doc, jeton) dans TF_IDF.

Calculons d'abord celui du body.

3.4.2 Calcul du TF-IDF pour le body

Entrée [38]:
```
doc = 0

tf_idf = {}

for i in range(N):

    tokens = processed_text[i]

    counter = Counter(tokens + processed_title[i])
    words_count = len(tokens + processed_title[i])

    for token in np.unique(tokens):

        tf = counter[token]/words_count
        df = doc_freq(token)
        idf = np.log((N+1)/(df+1))

        tf_idf[doc, token] = tf*idf

    doc += 1
```

FIGURE 13 – Calcul du TF-IDF pour le body

On calcule pour le titre.

3.4.3 Calcul du TF-IDF pour le body

Entrée [39]:
```
doc = 0

tf_idf_title = {}

for i in range(N):

    tokens = processed_title[i]
    counter = Counter(tokens + processed_text[i])
    words_count = len(tokens + processed_text[i])

    for token in np.unique(tokens):

        tf = counter[token]/words_count
        df = doc_freq(token)
        idf = np.log((N+1)/(df+1))

        tf_idf_title[doc, token] = tf*idf

    doc += 1
```

FIGURE 14 – Calcul du TF-IDF pour le titre

Ensuite on procède au calcul suivant :

3.4.4 Calcul du TF-IDF pour le body

```
Entrée [40]: tf_idf[(0,"go")]
    Out[40]: 0.0002906893990853149

Entrée [41]: tf_idf_title[(0,"go")]
    Out[41]: 0.0002906893990853149
```

3.5 Fusion de la TF-IDF en fonction des poids

Nous avons ici une valeur **alpha**, qui est le poids pour le corps, puis évidemment, Premièrement, nous devons maintenir une valeur alpha, qui est le poids pour le corps, puis évidemment, 1-alpha sera le poids pour le titre. sera le poids pour le titre.

```
Entrée [43]: alpha = 0.3

Entrée [44]: for i in tf_idf:
                 tf_idf[i] *= alpha

Entrée [45]: for i in tf_idf_title:
                 tf_idf[i] = tf_idf_title[i]

Entrée [46]: len(tf_idf)
    Out[46]: 344378
```

FIGURE 15 – Fusion de la TF-IDF

3.6 Classement en utilisant le score correspondant

Pour déterminer la similarité, le score de correspondance est le moyen le plus approprié et simple. Ici, il va falloir ajouter les valeurs tf_idf des jetons en requête pour chaque document.

Entrée [47]:
```
def matching_score(k, query):
    preprocessed_query = preprocess(query)
    tokens = word_tokenize(str(preprocessed_query))

    print("Matching Score")
    print("\nQuery:", query)
    print("")
    print(tokens)

    query_weights = {}

    for key in tf_idf:

        if key[1] in tokens:
            try:
                query_weights[key[0]] += tf_idf[key]
            except:
                query_weights[key[0]] = tf_idf[key]

    query_weights = sorted(query_weights.items(), key=lambda x: x[1], reverse=True)

    print("")

    l = []

    for i in query_weights[:10]:
        l.append(i[0])

    print(l)


matching_score(10, "Without the drive of Rebeccah's insistence, Kate lost her momentum. She stood next a slatted oak
```

FIGURE 16 – Fusion de la TF-IDF

Nous obtenons le résultat suivant :

```
Matching Score

Query: Without the drive of Rebeccah's insistence, Kate lost her momentum. She stood next a slatted oak bench, cani
sters still clutched, surveying

['without', 'drive', 'rebeccah', 'insist', 'kate', 'lost', 'momentum', 'stood', 'next', 'slat', 'oak', 'bench', 'ca
nist', 'still', 'clutch', 'survey']

[166, 200, 352, 433, 211, 350, 175, 187, 188, 294]
```

3.7 Classement utilisant la similarité Cosinus

Il est nécessaire de réaliser la similarité Cosinus également car bien que le score de correspondance fonctionne parfaitement, il n'est pas adapté au cas de longues requêtes. Le cosinus marque tous les documents en tant que vecteurs des jetons TF-IDF et les trace au centre. Même si la longueur de la requête est petite, elle pourrait être étroitement liée au document.

3.7.1 La vectorisation

Pour ce faire, on procède à la vectorisation puis on calcule la similarité en cosinus. Nous allons utiliser la variable **total_vocab** calculé plus haut, qui contient toute la liste des jetons uniques pour générer un index pour chaque jeton.

Entrée [49]:
```
D = np.zeros((N, total_vocab_size))
for i in tf_idf:
    try:
        ind = total_vocab.index(i[1])
        D[i[0]][ind] = tf_idf[i]
    except:
        pass
```

Entrée [50]:
```
def gen_vector(tokens):

    Q = np.zeros((len(total_vocab)))

    counter = Counter(tokens)
    words_count = len(tokens)

    query_weights = {}

    for token in np.unique(tokens):

        tf = counter[token]/words_count
        df = doc_freq(token)
        idf = math.log((N+1)/(df+1))

        try:
            ind = total_vocab.index(token)
            Q[ind] = tf*idf
        except:
            pass
    return Q
```

FIGURE 17 – Vectorisation

Dans ce cas, la similarité cosinus est le meilleur moyen de trouver la pertinence. Nous avons eu à calculer les valeurs de TF-IDF, TF nous avons pu calculer à partir de la requête elle-même.
Il ne reste plus qu'à calculer la similarité en cosinus pour tous les documents et à renvoyer le nombre maximal de **k** documents.

3.7.2 Calcul de la similarité en cosinus

Entrée [54]:
```
def cosine_similarity(k, query):
    print("Cosine Similarity")
    preprocessed_query = preprocess(query)
    tokens = word_tokenize(str(preprocessed_query))

    print("\nQuery:", query)
    print("")
    print(tokens)

    d_cosines = []

    query_vector = gen_vector(tokens)

    for d in D:
        d_cosines.append(cosine_sim(query_vector, d))

    out = np.array(d_cosines).argsort()[-k:][::-1]

    print("")

    print(out)

Q = cosine_similarity(10, "Without the drive of Rebeccah's insistence, Kate lost her momentum. She stood next a slat
```

FIGURE 18 – Calcul de la similarité en cosinus

Les résultats sont les suivants :

```
Cosine Similarity

Query: Without the drive of Rebeccah's insistence, Kate lost her momentum. She stood next a slatted oak bench, cani
sters still clutched, surveying

['without', 'drive', 'rebeccah', 'insist', 'kate', 'lost', 'momentum', 'stood', 'next', 'slat', 'oak', 'bench', 'ca
nist', 'still', 'clutch', 'survey']

[200 166 433 175 169 402 211  87 151 369]
```

La requête nous donne le résultat suivant :

Entrée [56]:
```
print_doc(200)
```

```
('/home/gisele/Documents/M1/ModuleFD/TP/stories/ghost', 'Time for Flowers, by Gay Bost')
TIME FOR FLOWERS
  by Gay Bost

They'd put flowers up. She hadn't noticed. Time wouldn't hold still.
She remembered, quite clearly, that time had been a simple thing; one
moment following the previous one, seconds strung out neatly like her
mother's pearls laid out on the dark mahogany vanity each Sunday
morning. But there had been a catch . . .

Hung around Mother's neck the catch clicked and the tidy little line
of seconds became a never ending circle with only the catch in the
middle. For some reason the thought of pearls gathered from the sea,
naturally nested within the confines of oyster shells, scattered
haphazardly about the ocean floor disturbed her.

Now they'd put up the flowers in the same careless groupings. This,
too, disturbed her. Bright yellow trumpets, their collars spread to
catch the sun, dotted the front yard in clusters of two or three, five
or six. Bunches laid carelessly and forgotten. In a moment she'd
```

FIGURE 19 – Affichage du résultat de la requête

Comme nous pouvons le voir ci-dessus, le document 200 (**print_doc(200)**) est tou-

jours très coté en méthode cosinus par rapport à la méthode d'appariement, car la similarité des cosinus apprend davantage le contexte.

Conclusion

Pour ce travail de recherche d'informations, nous avons utilisé les librairies telles que **nltk, numpy, re, mat, num2words** et l'outil **jupyter notebook**. Ce travail pratique nous a, non seulement amené à travailler avec un outil comme **jupyter notebook** mais aussi à nous frôler au **Machine Learning**. Nous avons rencontrés des difficultés concernant la manipulation des fichiers mais les recherches nous ont aidé à parvenir à arriver à bout de ce travail.

Références

— https ://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-world-dataset-796d339a4089

— https ://github.com/williamscott701/Information-Retrieval/blob/master/2.%20TF-IDF%20Ranking%20Cosine%20Similarity%2C%20Matching%20Score/TF-IDF.ipynb

— https ://tel.archives-ouvertes.fr/file/index/docid/524514/filename/these_Fabienne_Moreau.pdf