TATIA

COMPTE RENDU DE PROJET

SAMAK Maxime
MUNIER Remy

Sujet du projet

Dans le cadre du projet de Traitement automatique du texte en IA nous avons décidé de concevoir un mini moteur de recherche destiné à trouver des articles d'information à partir d'un flux RSS, dans le cas présent le journal "Le Monde" sera la source de ce flux.

Introduction

Mener à bien ce projet nous a mis face à différents problèmes que nous nous sommes efforcés de traiter de la manière la plus optimale possible. Dans la mesure du possible nous avons essayé d'avoir le moins possible recours à des librairies offrant une solution clé en main dans le but de conserver un aspect d'apprentissage et de découverte du Traitement Automatique du Langage Naturel (TALN).
Ainsi au cours de ce compte rendu, il sera exposé les difficultés rencontrées de même les solutions ou tentatives de solutions apportées lors de l'élaboration de ce projet.

Gestion des données

Pour pouvoir démarrer un tel projet il est en premier lieu nécessaire de se constituer un jeu de données et de mettre en forme ces dernières pour faciliter leur utilisation.
La norme de stockage de l'information des flux RSS étant les fichiers au format XML, il a été dans premier temps notre priorité de pouvoir lire et traiter ce type de fichier d'une manière simple et efficace.

```
−<title>
    Le Monde.fr - Actualités et Infos en France et dans le monde
  </title>
−<description>
    Le Monde.fr - 1er site d'information. Les articles du journal et toute l'actualité en continu : International, France, Société, Economie, Culture, Environnement, Blogs ...
  </description>
−<copyright>
    Le Monde - L'utilisation des flux RSS du Monde.fr est réservée à un usage strictement personnel, non professionnel et non collectif. Toute autre exploitation doit faire l'ob
  </copyright>
  <link>https://www.lemonde.fr/rss/une.xml</link>
  <pubDate>Tue, 31 Dec 2019 18:02:52 +0000</pubDate>
  <language>fr</language>
  <atom:link href="https://www.lemonde.fr/rss/une.xml" rel="self" type="application/rss+xml"/>
−<item>
  −<title>
      Policiers, pilotes, SNCF... Les concessions de l'exécutif sur la réforme des retraites
    </title>
    <pubDate>Tue, 31 Dec 2019 14:06:43 +0100</pubDate>
  −<description>
      Le gouvernement est accusé de renoncer petit à petit à son ambition initiale d'universalité du système de retraites.
    </description>
  −<guid isPermaLink="true">
      https://www.lemonde.fr/politique/article/2019/12/31/reforme-des-retraites-des-accrocs-a-l-universalite_6024487_823448.html
    </guid>
  −<link>
      https://www.lemonde.fr/politique/article/2019/12/31/reforme-des-retraites-des-accrocs-a-l-universalite_6024487_823448.html
    </link>
    <enclosure url="https://img.lemde.fr/2019/12/18/480/0/2362/1179/644/322/60/0/ecbd1fb_TGgZU9eQ_JKa9GqK56LjTDPs.jpg" length="30963" type="image/jpeg"/>
  </item>
```

Nous avons donc commencé par créer un parseur permettant d'isoler les éléments pertinents de ces fichiers, à savoir le contenue des différentes balises <item>. En effet ces balises constituent l'ensemble des données utiles, c'est à dire le titre de l'article d'information, une description courte de son contenu et enfin le lien vers la page web de l'article en question.

Une fois ces balises isolées pour l'ensemble des fichiers xml de notre jeu de données, des objets sont créés avec pour attributs les 3 informations présentant un intérêt.

```python
class Article:
    def __init__(self, title, description, link):
        self.title = title
        self.description = description
        self.link = link
```

Une collection constituée de l'ensemble de tous les objets Article ainsi obtenue devient alors la base de notre moteur de recherche.

Acquisition des mots clés

Pour effectuer une recherche il est impératif d'avoir des mots clés constituant l'objet de la recherche à effectuer. Pour cela une simple lecture d'un fichier ".in" est la solution qui nous a paru être la plus optimale.

Génération de synonymes

La langue Française étant très riche les mots clés saisies peuvent ne pas trouver d'homologue dans les différents articles d'information à notre disposition tout en ayant un sens presque similaire voir totalement identique à ceux présents dans la description / titre de ces dits articles.

Une solution à ce problème est de générer une liste de synonymes pour chaque mot de notre recherche de manière à étendre les chances de trouver un mot similaire dans les articles.
Pour cela l'utilisation de la librairie nltk wordnet s'est révélé très importante.

Ainsi lors d'une recherche avec le mot clé "voiture" on peut générer quelques synonymes susceptibles de nous intéresser : "wagon, entraîneur, véhicule, voiture, auto, carrosse, automobile"
On peut alors couvrir un étendu plus large lors de notre recherche

Cet exemple illustre l'utilisation que nous avons fait de cet outil durant le projet. Et même si cette utilisation présente des bénéfices elle s'accompagne également d'un problème assez important. Il s'agit d'un effet qu'on pourra qualifier de "dérive de la recherche" qui provoquera parfois la découverte d'un article peu corrélé à l'intention de notre recherche. Cet effet est très sensible à la qualité ainsi qu'à la quantité des synonymes obtenues avec wordnet et il s'est avéré que l'utilisation du Français ne nous est ici pas favorable.

TF-IDF et COSINE SIMILARITY

Pour procéder à la recherche et la découverte un article pertinent il est important de mesurer le "poids" des mots par rapport à l'article dans son ensemble de même que la totalité des articles.
Nous avons donc fait usage de la méthode TF-IDF (term frequency-inverse document frequency) permttant de pondérer les termes contenus dans un document.

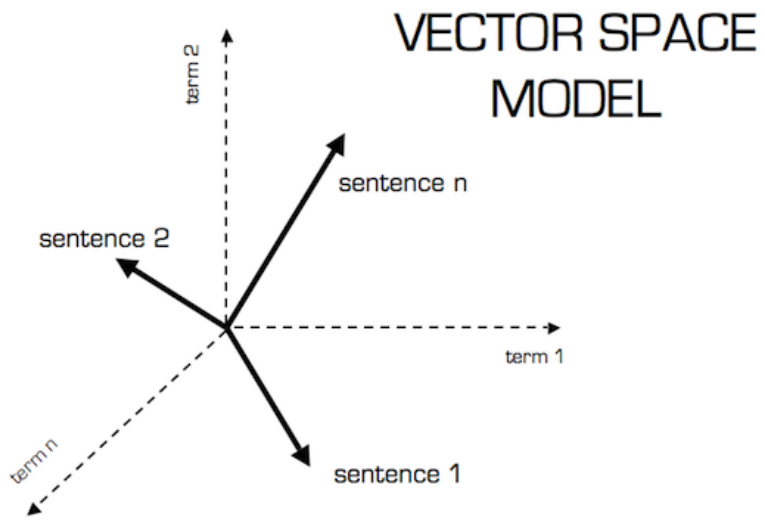
$$idf(t) = 1 + \log(\frac{n_{total}}{n_{docs(t)}})$$

Nous avions dans un premier temps fait usage de de la méthode NTF (normalized term frequency) :

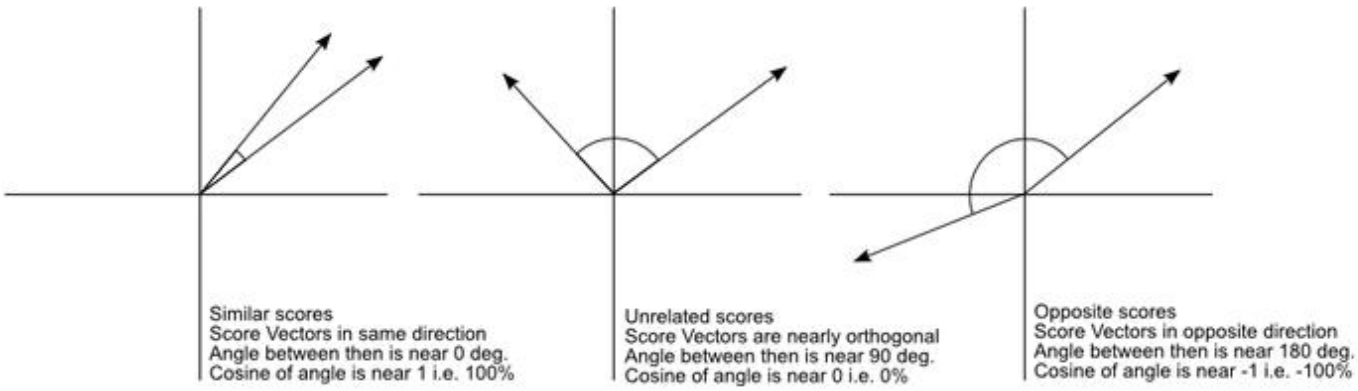
$$tf_{norm} = 1 + \log(tf_{raw})$$

Mais avons vite réaliser notre erreur et transitionner vers une méthode plus correcte et prenant en compte la différence de taille entre nos documents et notre entrée.

Une fois cette étape complétée on peut alors transformer chaque description et titre d'article (et notre entrée) en un vecteur représenté par nos pondérations TF-IDF. La suite consiste simple à trouver le vecteur de document le plus proche de notre vecteur de recherche.



Sickit learn nous offre alors la solution la plus efficace avec la méthode de mesure de distance cosinus (cosine_similarity())



On peut alors identifier quel article est le plus pertinent par rapport à notre recherche initiale.
On retourne alors le lien menant à sa page web.

Une observation intéressante est qu'il existe des méthodes plus poussées (et plus rapide) au sein de SpaCy cependant leur utilisation nous a été rendu difficile voire impossible au vu de la langue choisie pour ce projet.

Conclusion

Ce projet s'est révélé être des plus instructifs notamment en ce qui concerne les méthodes de recherche d'information et d'étude des langages naturels.
De même de nombreuses méthodes et outils ayant été exploré mais n'ayant pas abouti à un résultat qui nous paraissait satisfaisant pour ce projet nous ont permis d'acquérir des connaissances plus vastes que ce compte rendu de projet seul pourrait le laisser entendre. Et nous avons donc pu nous familiariser avec des outils, méthodes et librairies tel que : SpaCy, TreeTagger, Numpy, Scipy, NTF, RFrequency, …

Dans une autre mesure il faut toutefois rappeler que même si dans l'ensemble nous sommes assez satisfaits du stade auquel nous sommes arrivés pour ce projet, il existe une marge d'amélioration assez grande.
En effet on pourrait améliorer le modèle actuel en y intégrant l'identification des entités nommés ainsi que de la désambiguïsation (l'utilisation du Français représentant toutefois un frein à ces deux améliorations).

Utilisation du projet et setup

Avant de pouvoir lancer le projet il faut avant tout s'assurer que nltk ainsi que sickit-learn soient installés.
Il faut pour cela exécuter les lignes de commande suivantes à la racine du projet :

```
1. pip install -U scikit-learn
2. pip install ntlk
```

Une fois ces étapes complétées on peut alors ouvrir le fichier input.in et y écrire les mots clés souhaité pour la recherche.

Exemples :

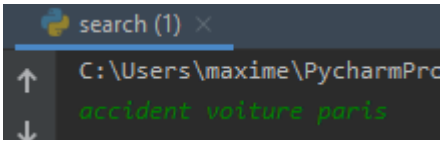
- Séisme catastrophe naturel
- Accident voiture paris
- Manifestation accident police

A noter que seule la première ligne du fichier input.in est prise en compte. Il est également préférable d'entrer entre 2 et 5 mots clés lors de la recherche pour minimiser les phénomènes de "dérive".

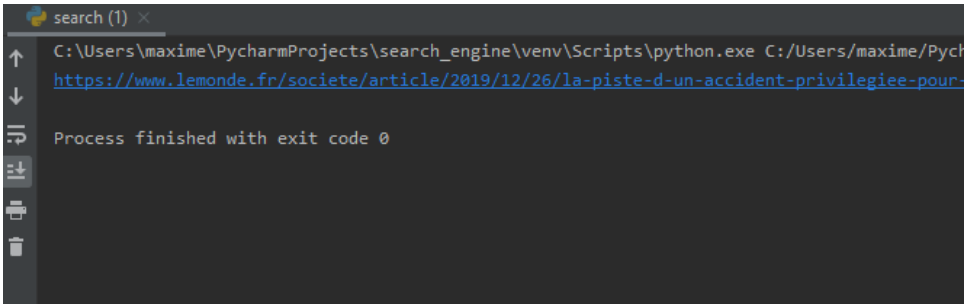
Il faut alors exécuter les lignes de commandes suivantes :

1. cd engine
2. Python3 search.py < input.in

Il est également possible de lancer directement search.py sans indiquer de fichier .in, il est alors requis de renseigner les mots clés à utiliser directement dans le terminal.



Le lien correspondant à l'article trouvé s'affiche alors dans le terminal.


search (1)
C:\Users\maxime\PycharmProjects\search_engine\venv\Scripts\python.exe C:/Users/maxime/Pych
https://www.lemonde.fr/societe/article/2019/12/26/la-piste-d-un-accident-privilegiee-pour-
Process finished with exit code 0