L3 Informatique 2019 / 2020

Projet de Linguistique Informatique
Indexation de documents et moteur de recherche

Auteurs:

MIRET Blanche
TRAN THACH Linh

SOMMAIRE

Introduction

Le but de ce projet était de prendre en main des outils d'analyse et de traitement automatique de la langue étudiés en cours de Linguistique Informatique. Intéressées par les méthodes d'indexation de documents, nous avons souhaité créer un moteur de recherche très simple afin de comprendre les premiers enjeux et étapes de l'indexation et de la recherche de documents. Ce rapport présente quelques éléments théoriques sur ces sujets ainsi que notre processus de travail.

Le cours[1]

Indexation de documents

Représentation à l'aide de matrices

En Traitement Automatique de la Langue il est nécessaire d'imaginer une façon de représenter les documents sur ordinateur pour que la machine puisse les traiter. Une solution populaire est de penser les documents comme des "sacs de mots", sans signification ou ordre particulier, et de les modéliser à l'aide de *Vector Space Model* - matrice termes par document. À chaque document est attribué une coordonnée par terme présent dans la matrice, autour de 50 000 si l'on se base sur un dictionnaire simple. La coordonnée est alors le nombre de fois que le terme apparaît dans le document.

TF - IDF

La méthode TF-IDF s'utilise sur une matrice termes par document et pondère les coordonnées pour les rendre plus représentatives de la proximité réelle entre deux contenus de texte. En effet traiter les rapports des documents entre eux en se basant uniquement sur la fréquence des termes n'est pas assez subtil : la taille des documents, ainsi que l'"importance" d'un terme parmi le corpus doivent entrer en compte.

La TF - *Term Frequency* - répond à la première problématique : pour un terme t, si deux textes contiennent chacun 10 fois ce terme et que le premier a une longueur de 100 mots tandis que le second mesure 100 000 mots, il est certain que t a plus d'importance dans la représentation du contenu du premier texte que du second. En conséquence de quoi, le poids du terme doit être grand pour le premier texte, beaucoup moins pour le second. C'est pourquoi, et selon l'utilisation faite du modèle TF-IDF, il existe plusieurs calculs définis pour mesurer la *Term Frequency* d'un document[2] :

[1] Cf Transparents Extractions d'informations et indexation de documents : http://damien.nouvels.net/cours/introtal/03_ExtractionIndexation.pdf
[2] Calculs à retrouver sur la TF-IDF Wikipedia Page : https://fr.wikipedia.org/wiki/TF-IDF

- Binaire : 1 ou 0 selon si le terme est présent ou non.
- Fréquence brute : $tf_{i,j} = f_{t_i,d_j}$
- Normalisation par le max : $tf_{i,j} = \frac{f_{t_i,d_j}}{\#mots\ dans\ d_j}$:
- Normalisation logarithmique : $tf_{i,j} = 1 + log(f_{t_i,d_j})$

tf : Term Frequency	t : term	i : indice d'un terme
f : frequency	d : document	j : indice d'un document

La deuxième problématique apparaît dans le choix des termes : les termes de la langue courante, les déterminants ou les déclinaisons des auxiliaires par exemple, sont très peu représentatifs du contenu d'un document. Comment identifier l'importance d'un mot ? C'est là qu'intervient l'IDF : *Inverse Document Frequency*, attribuée à chaque terme par rapport à un corpus : plus le nombre de documents contenant le terme est faible, plus le terme est susceptible de représenter un concept fortement représentatif du thème d'un document, c'est pourquoi son poids doit augmenter. D'où la formule suivante :

$$idf_i = log \frac{|D|}{|d_j : t_i \in d_j|} = log \frac{\#\ docs}{\#docs\ contenant\ le\ terme\ t_i}$$

Et la coordonnée finale d'un document d'indice *j* par rapport à un terme *i* est ainsi :

$$tfidf_{i,j} = tf_{i,j} \times idf_i$$

Distance et recherche de documents

Grâce à ces représentations mathématiques il est possible de traiter ces documents avec des opérations algébriques, comme n'importe quels vecteurs, et notamment de mesurer leur distance. C'est ce que permet l'opération de similarité cosinus reproduite ici :

$$cos\ \theta = \frac{A.B}{||A||||B||} = \frac{\Sigma x_{Aj} \times x_{Bj}}{\sqrt{\Sigma x_{Aj}^2} \times \sqrt{\Sigma x_{Bj}^2}}$$

Cette méthode doit pouvoir rapprocher des documents par leur contenu, en se basant sur la similarité des mots présents entre les différents textes d'un corpus, et permet de répondre à une requête entrée dans un moteur de recherche par exemple. Il s'agit alors de créer une matrice *termes par document* à partir des termes de la requête entrée par l'utilisateur, de mesurer la distance de chaque élément de la base de donnée avec cette requête - prise en compte comme un document en elle-même, puis de retourner les résultats les plus proches de la requête.

Deux critères permettent d'évaluer la pertinence d'une réponse de recherche dans un moteur : la précision, qui évalue le bruit : c'est la quantité de bonne réponse sur le nombre total de réponses ; et le rappel, qui évalue le silence : c'est la quantité de réponses ayant été trouvées sur le nombre total de bonnes réponses qui auraient dû être trouvées.

Le projet

Méthode

1. Base de données

La première étape fut de constituer une base de données sur laquelle tournerait le moteur de recherche. Le choix fut fait fait de travailler sur le thème de la littérature, et dans la continuité des exemples vus en cours, sur les pages Wikipedia des oeuvres choisies, leur contenu résumant de façon idéale les documents à représenter. Le module python *wikipediaapi* [3] fournissant tous les éléments nécessaires à l'extraction d'une page wikipedia, il fut utilisé pour écrire un script python *wiki_copy.py* permettant la constitution rapide d'une petite base de donnée.

2. Recherche

C'est avec le module python *sklearn*, et en particulier *TfidfVectorizer*[4] qu'a été construit le moteur de recherche. Ce module permet de construire une matrice TF-IDF à partir d'un corpus de documents donné, éventuellement d'un vocabulaire. Le calcul de la *Term Frequency* implémenté par le modèle se fait par la fréquence brute des termes par document.

[3] https://pypi.org/project/Wikipedia-API/
[4] TfidfVectorizer Documentation Page : https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html#sklearn.feature_extraction.text.TfidfVectorizer

Le script *research.py* implémente une première méthode de recherche : après la construction de la liste des documents de la base de données contenue dans le répertoire *bdd_utf8* puis l'extraction du vocabulaire de la requête, la matrice TF-IDF est créée. Afin d'obtenir les coordonnées qui représente la requête entrée par l'utilisateur, une deuxième matrice est créée, calculée à partir du même vocabulaire que la précédente et contenant uniquement ce "document". Cette méthode, améliorable, facilitait l'utilisation d'entrées de types différents dans la création des matrices via *sklearn* : par nom de document ou contenu brut. Il fallait ensuite calculer la distance de chaque document de la base de données à la requête et déterminer un critère de filtrage de résultat : nous avons choisi d'établir un seuil à 0,8 de distance avec la requête pour que le document soit retenu dans les résultats. Les résultats sont enfin triés par proximité avec la requête, et affichés sur la console.

Bien que satisfaisants dans l'ensemble, quelques tests utilisant ce premier script ne donnaient pas les résultats attendus. Par exemple, la recherche des mots "victor" et "hugo" donnaient d'abord "L'étranger" puis "Les fleurs du mal" avant les oeuvres écrites par Victor Hugo lui-même. Après quelques recherches, il semble que ces résultats étaient renvoyés par ordre d'inverse de fréquence brute d'apparition… Le script *v2_research.py* implémente donc une seconde méthode de recherche avec une matrice TF-IDF partiellement construite manuellement : le compte des termes par documents est pris en charge par le module *CountVectorizer*[5] de *sklearn*, le calcul et l'application de la pondération par l' *Inverse Document Frequency* est ré-implémentée. En clair, cette deuxième méthode reproduit la création de la matrice *TfidfVectorize*r moins l'application de la norme d'Euclide, comme décrit dans le *User Guide sklearn*[6]. Dans ce second script, plutôt que de recréer une matrice pour la requête, cette dernière est représentée en ayant systématiquement 1 pour chacune de ses coordonnées. Les résultats peuvent être plus probants : les résultats pour la recherche "victor hugo" sont en effet au moins triés par nombre décroissant d'apparition des termes dans chaque document.

Code

L'intégralité du code source et la base de données constituée est à retrouver sur le dépôt git hébergé à l'adresse suivante :
https://gaufre.informatique.univ-paris-diderot.fr/miret/projet_li_2020

[5] CountVectorizer Documentation Page :
https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html#sklearn.feature_extraction.text.CountVectorizer

[6] TfidfVectorizer User Guide :
https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

Description des modules et scripts

- *wiki_copy.py* : script permettant d'extraire le contenu de pages Wikipédia. Les pages sont copiés dans le répertoire bdd_utf8. Usage :
  `% ./wiki_copy.py "page_name1" ["page_name_2"] [...]`

- *search_engine.py* : module fournissant les fonctions utilisées dans les deux scripts de recherche fournis. Entre autres, construction de la liste de documents contenus dans le répertoire *bdd_utf8*, construction d'un vocabulaire à partir des arguments passés en ligne de commande par l'utilisateur, calcul de distance entre documents, implémentation de l'algorithme de tri rapide, construction d'une matrice TF-IDF.

- *research.py* : premier script de recherche dans la base de données contenue dans le répertoire *bdd_utf8*. Usage :
  `% ./research.py word1 [word2] [...]`

- *v2_research.py* : second script de recherche dans la base de données contenue dans le répertoire *bdd_utf8*. Usage :
  `% ./v2_research.py word1 [word2] [...]`

Exemples d'exécution

```
%./research.py magie

The documents corresponding to your research are :
Le Silmarillion : 1.0
Harry Potter À L'école Des Sorciers : 1.0
Les Fleurs Du Mal : 1.0

% ./research.py victor hugo

The documents corresponding to your research are :
L'étranger : 0.9948775137905812
Les Fleurs Du Mal : 0.9948775137905812
Notre Dame De Paris : 0.9948775137905812
Les Misérables : 0.9943538524494836
Hernani : 0.9707624692005832
```

```
% ./v2_research.py victor hugo

The documents corresponding to your research are :
Hernani : 0.9899494936611665
Les Misérables : 0.9551731226472869
Les Fleurs Du Mal : 0.9486832980505139
L'étranger : 0.9486832980505138
Notre Dame De Paris : 0.9486832980505138
```

Perspectives d'amélioration

En commençons par des améliorations légères, remarquons que la mesure de Term *Frequency* utilisée par le module *sklearn* ne prend pas en compte la taille des documents, ce qui pourrait être un premier point afin de poursuivre les tests d'affinage. Les accents ne sont pas encore tout à fait géré non plus.

Une amélioration possible plus concrète serait d'étendre les recherches aux mots similaires à ceux entrés dans la requête. Dans un premier temps en travaillant par exemple avec les mots ayant les mêmes racines que ceux donnés par l'utilisateur. La tâche n'est pas évidente, mais des outils existent déjà pour extraire le lemme des mots et leurs déclinaisons, ce qui donnerait un peu plus de consistance au moteur de recherche implémenté. Ensuite, en travaillant avec un dictionnaire, construire des groupes de synonymes pour chaque mot et étendre la recherche aux documents semblant rester proches de la requête sans contenir les mots eux-mêmes.

Conclusion

Cette expérience de construction d'un moteur de recherche nous a permis d'aller à la rencontre de quelques problématiques concrètes du traitement de la langue : qu'est-ce qu'un document pour une machine ? Comment traiter les mots indépendamment de leur sens ? Comment se servir intelligemment des capacités de calculs de la machine pour qu'elle soit au service l'homme, en conservant du sens ? Nous avons pris une certaine mesure des subtilités du domaine et de la question passionnante qu'est la traduction de la langue utilisée par l'homme à celle traitée par des machines et sommes ravies d'avoir fait cette découverte à travers l'option de Linguistique Informatique cette année !