Université de Caen Normandie
Licence informatique 2ème année

11 septembre 2020
Données Web et multimédia

Compte-rendu de TP

A.L & MALBEC Elie

Table des matières

1 Introduction **2**
1.1 Introduction . . . 2
1.2 Organisation . . . 2

2 Réalisation du TF-IDF **2**
2.1 Définitions . . . 2
2.2 TF . . . 2
2.3 DF . . . 3
2.4 TF-IDF . . . 4
2.4.1 tfidf.sh . . . 5
2.4.2 tfidf.awk . . . 5
2.5 Conclusions . . . 5

3 Expérimentations **5**
3.1 Temps d'exécution . . . 5
3.1.1 Chronométrage du TF . . . 5
3.1.2 Chronométrage du DF . . . 5
3.1.3 Chronométrage du TF-IDF . . . 5
3.2 Conclusions . . . 5

4 Index de recherche **5**
4.1 Réalisation de l'index . . . 5
4.2 Interrogation de l'index . . . 6

1 Introduction

1.1 Introduction

Dans le cadre du cours Données Web et multimédia, nous avons pu comprendre et réaliser les préliminaires d'un moteur de recherche sur un corpus de texte. Pour cela, nous avons étudié la méthode de pondération sur la fouille de textes ***TF-IDF***. Ce rapport est un compte-rendu des étapes qui ont menés à la mise en place de ce moteur de recherche ainsi que les difficultés rencontrées. Notre binôme est composé de Elie Malbec et A.L .

1.2 Organisation

Nous avons à disposition un corpus de 3655 textes avec un sujet et un contenu. Nous avons utilisé le langage `shell` pour travailler sur ce corpus car les outils à disposition sont nombreux et rapides à utiliser. Des outils génériques comme `sed`, `awk` ou `grep` manipulent les chaînes de caractères avec de nombreuses possibilités.

Le but de ces TP est de répondre à une requête de mots donnés par l'utilisateur et lui renvoyer les documents les plus pertinents. Pour arriver à cette étape, l'élaboration du **TF-IDF** est nécessaire.

2 Réalisation du TF-IDF

2.1 Définitions

La méthode TF-IDF permet d'évaluer statistiquement la pertinence d'un mot dans un document comparativement au corpus de texte auquel il appartient. C'est une méthode de pondération qui se base sur le nombre d'occurrences d'un mot dans le corpus de textes.

Le TF-IDF est composé de deux parties, le TF qui signifie "*Term-Frequency*". Le TF évalue pour chaque mot d'un document, le nombre de fois qu'il apparaît.

La seconde partie, DF, signifie "*Document-Frequency*". Le DF évalue quand à lui, le nombre de documents du corpus de texte qui contiennent un mot donné.

Les deux mesures précédentes forment le score TD-IDF.

2.2 TF

La mesure TF, est le nombre d'occurrences d'un mot dans un document. C'est à dire, cela correspond au nombre de fois qu'un mot apparaît dans un texte. Il peut être représenté de la façon suivante : $TF(word, doc)$.

Dans notre cas, les textes sont des contenus de mails. Or nous pouvons y trouver des chiffres, signes de ponctuations, lignes vides et des majuscules. Ces caractéristiques posent problème car il ne s'agit pas seulement de mots, mais aussi un mot peut s'écrire de plusieurs façons différentes avec ou sans majuscules.

Le TP1 nous a permis de nettoyer les textes pour les transformer en listes de mots. Cette liste sera ensuite triée et les occurrences de chaque mot comptées. Nous obtiendrons ainsi la mesure du TF pour chaque mot présent dans le document.

Les textes contenus dans le dossier `content` portent pour nom les numéros de 1 à 3655. Nous pouvons donc parcourir l'ensemble des fichiers avec une boucle `for`.

Pour nettoyer les textes, il faut utiliser le système de pipes du `shell`, la sortie d'une commande devient l'entrée de la suivante, il n'y a pas besoin de stocker le résultat intermédiaire dans un fichier. Dans un premier temps, il faut lire le contenu du fichier puis passer l'ensemble des caractères en minuscules. Ensuite, avec l'outil `sed`, tous les caractères qui ne sont pas des lettres sont retirés. Nous avons donc la commande suivante :

```
cat content/i | tr [:upper:] [:lower:] | sed 's/[^a-z]/ /g'
```

À la sortie de cette étape, le document est constitué de lignes avec plusieurs mots en minuscules et sans ponctuation. Il faut désormais séparer les mots sur des lignes différentes pour les trier par la suite. Une commande `sed` va substituer tous les espaces par des sauts de ligne. Puis les lignes vides supprimées avec une commande `grep` inversée.

```
sed 's/ /\n/g' | grep -v '^$'
```

La commande `grep` fonctionne de la façon suivante : l'option v sélectionne les lignes qui ne correspondent pas au pattern suivant. Le pattern sélectionne toutes les lignes ne comportant aucun caractère. Nous avons donc une commande qui signifie, récupération de toutes les lignes qui ne sont pas vides. D'autres variantes de cette commande existent avec d'autres outils, par exemple :

```
sed '/^$/d'
awk 'NF!=0{print}'
```

Désormais, chaque ligne du résultat contient un mot du document. Pour trouver le nombre d'occurrences de chaque mot, il faut trier la sortie avec la commande `sort`. Il ne reste qu'à compter les occurrences. L'outil `uniq` avec son option *c* nous donne la réponse mais il est aussi possible de l'effectuer avec un script `awk`.

Nous avons donc :

```
sort | awk 'NR == 1 {mot = $1}{if (mot == $1) tf++; else {print mot, tf; mot =
    $1; tf = 1}} END {print mot, tf}'
```

`awk` parcourt le fichier ligne par ligne, si le mot de la ligne correspond à celui de la ligne précédente, le compteur est incrémenté. Sinon le mot est affiché avec ses occurrences avant de passer au mot suivant.

La commande complète pour calculer le TF du texte numéro 1 est donc la suivante :

```
cat content/1.txt | tr [:upper:] [:lower:] | sed 's/[^a-z]/ /g' | sed 's/ /\n/g'
    | grep -v '^$' | sort | uniq | awk 'NR ==1 {mot = $1}{if (mot == $1) tf++;
    else {print mot, tf; mot = $1; tf = 1}} END {print mot, tf}' > content/1.tf
```

La sortie est redirigée vers le fichier `1.tf` pour ne pas écraser le fichier original. Le contenu du fichier `1.tf` est le suivant :

```
amghar 1
applications 1
au 1
bonjour 1
calabretto 3
cnrs 1
complète 1
concours 1
...
```

Nous pouvons voir sur ce résultat que chaque mot contenu dans ce document est accompagné de son nombre d'occurrences. Le calcul du TF est effectué pour chaque document du corpus grâce à une boucle `for`.

2.3 DF

La mesure DF est le nombre de fois qu'un mot est présent dans un corpus de textes. Il convient de mettre bout à bout la liste de tous les mots de chaque document. Chaque mot n'étant présent qu'une fois par document. Cette mesure peut s'écrire de la façon suivante : $DF(word)$.

Pour réaliser cette mesure, il faut nettoyer le corpus comme pour le TF. Nous allons donc réutiliser les commandes du TP1 :

```
cat content/i | tr [:upper:] [:lower:] | sed 's/[^a-z]/ /g' | sed 's/ /\n/g'
```

Nous obtenons un mot par ligne, non triés pour le document *i*. Nous allons ensuite trier et simplifier le résultat avec `sort` et `uniq` dans le but d'avoir une seule fois chaque mot du document.

Pour un mot donné au DF, c'est le nombre de fois qu'il apparaît dans tous les documents, il nous faut donc compiler à la suite les listes de mots pour tous les textes du corpus. Une boucle nous permet de réaliser cela. Nous avons choisi de stocker le résultat dans un fichier intermédiaire pour la prochaine étape.

```
for j in $(seq $i); do
  cat content/$j | tr [:upper:] [:lower:] | sed 's/[^a-z]/ /g' | sed 's/ /\n/g'
      | sort | uniq | grep -v '^$' >> tmp.txt
```

Le fichier `tmp.txt` contient la liste des mots de tous les documents, où pour chaque document un mot est présent qu'une seule fois. Le nombre d'occurrences du mot "de" dans cette liste est le nombre de documents dans lequel il est présent.

Nous allons générer le fichier `df.txt` qui à chaque mot de cette liste, lui associe le nombre de documents dans lequel il est présent. Autrement dit, le nombre de fois qu'on le trouve dans `tmp.txt`. Il faut trier la liste et compter avec une des méthodes vues pour le calcul du TF.

```
cat tmp.txt | sort | uniq -c > df.txt
```

Le contenu du fichier `df.txt` ressemble à :

```
1797 a
3369 à
   6 aa
   2 aaaaj
   9 aaai
   1 aaas
   5 aac
   3 aachen
   1 aae
   6 aafd
   ...
```

Dans ce résultat, certains mots comme "à" sont présents dans presque la totalité des documents et d'autres moins présents. Nous pouvons également remarquer que certains mot apparaissent dans les résultats mais n'existent pas comme "aac",cela est dû aux les règles de substitution de caractères, notamment avec la suppression des caractères qui ne sont pas des lettres. Ces mots n'ont pas d'impact sur les futures recherches.

Notons que le calcul du DF est à refaire à chaque changement de texte dans le corpus contrairement au TF qui change uniquement pour le fichier qui lui est associé.

Le calcul du DF est désormais terminé, il n'est pas nécessaire de le générer à chaque nouvelle recherche car les textes ne seront pas modifiés.

2.4 TF-IDF

Le TF-IDF est la composante du TF et du DF expliqués précédemment, il s'agit d'un score de pertinence donné à un mot dans un document par rapport à un corpus de textes. Plus le score est haut, plus le mot est pertinent pour rechercher ce document dans le corpus.

Un mot est d'autant plus pertinent dans un document si sa fréquence est élevée et le nombre de documents qui contiennent ce mot est faible. Avec les deux mesures précédentes, cela signifie que le TF-IDF va croître avec le TF mais décroître avec le DF.

La formule utilisée pour le calcul est la suivante :

$$TFiDF(word, doc) = TF(word, doc) * log\frac{nDocuments}{DF(word)}$$

Pour réaliser ce calcul sur tous les mots du corpus, un script `awk` est réalisé, il applique à chaque mot de chaque fichier des TF, la formule pour calculer son score. Il est nécessaire d'utiliser le fichier `df.txt` et les fichiers `num.tf` pour le trouver.

Ce script récupère pour chaque mot du fichier TF, le nombre d'occurrences dans le `df.txt` et affiche le score calculé. Ce processus est généralisé à l'ensemble des fichier TF. Une règle de tri place les mots les plus pertinents en premier. La sortie est redirigée vers le fichier `num.tfidf` associé au TF. Chaque document du corpus possède désormais deux fichiers qui lui sont associés : `i.tf` et `i.tifdf`.

Un exemple de fichier obtenu avec le `tfidf` pour le document 3 est le suivant :

```
smis 45.7516
pucheral 38.4726
uvsq 37.1944
saclay 29.5439
protection 25.404
versailles 18.8294
david 17.4268
embarquées 17.7038
inria 15.3279
quentin 15.3384
...
```

Le score de chaque mot est présent à droite. Une recherche sera pertinente pour ce document si les mots clés sont ceux situés en haut du fichier.

Nous pouvons noter que le mot "pucheral", qui est un nom de famille, permettra de retrouver rapidement ce fichier, si il est combiné au mot "protection", il y a de grandes chances que ce document soit le seul avec ces mots. Les commandes nécessaires sont les fichiers 2.4.1 et 2.4.2.

2.4.1 tfidf.sh

```
for j in $(seq $i); do
  awk -f tfidf.awk content/$j.tf df.txt  | sort -k 2,2nr > content/$j.tfidf
done
```

2.4.2 tfidf.awk

```
BEGIN {
  fileDF = ARGV[--ARGC]
  while ((getline < fileDF) > 0)
    df[$2] = $1
}
{
  print $1, $2 * log (3655 / df[$1])
}
```

2.5 Conclusions

Avec l'aide de ces outils statistiques, nous pouvons dégager certaines conclusions. Par exemple, les mots les plus pertinents pour un document sont ceux présents à de nombreuses reprises dans ce dernier mais peu dans le reste du corpus.

À l'inverse, les mots les moins pertinents sont présents dans tous les documents, les mots courts principalement.

Certaines améliorations permettraient de rejeter les mots incohérents, des règles plus sophistiquées de filtrage pourraient être mise en place. particulièrement

3 Expérimentations

3.1 Temps d'exécution

3.1.1 Chronométrage du TF

3.1.2 Chronométrage du DF

3.1.3 Chronométrage du TF-IDF

3.2 Conclusions

À l'aide des résultats obtenus précédemment, il est possible de dégager des conclusions. Le temps d'exécution est en lien avec le nombre de fichiers du corpus. Son évolution est linéaire. De plus, plus le nombre de documents est important, plus le TF-IDF est précis.

4 Index de recherche

Nous avons désormais à disposition les outils nécessaires pour évaluer statistiquement la pertinence des mots d'un document dans un corpus de textes. Pour effectuer une recherche, il faut créer un `index` de recherche.

Ce dernier doit permettre de retrouver les documents qui contiennent un mot donné. C'est à dire, si l'index est interrogé pour le mot "poste", le nom de tous les documents qui le contiennent seront retournés. Les deux parties suivantes exposent sa réalisation.

4.1 Réalisation de l'index

Pour créer un index, il faut se servir des fichiers TF générés précédemment. Chaque mot contenu dans les TF est présent une seule fois, avec son nombre d'occurrences. Ce nombre est remplacé par le nom du document dans lequel il se trouve. Dans notre cas, il s'agit de nombres entre 1 et 3655.

Cette opération est appliquée sur les 3655 fichiers et les résultats mis à la suite. Une règle de tri ordonne selon les mots puis les numéros des documents. Enfin, un script `awk` récupère tous les noms de fichiers pour un mot, puis écrit pour le mot donné tous ses documents sur une ligne du fichier `index`. Le code complet est le suivant :

```
for i in $(seq 3655); do cat $i.tf |
  sed "s/ .*/ $i/"; done |
  sort -k1,1 -k2,2n |
  awk '{if ($1 != last){if (last!="") print last, tab[last]; last = $1; tab[last
      ] = $2} else tab[last] = tab[last] " " $2} END {print last, tab[last]}' >
      index
```

Le contenu du fichier `index` ressemble à :

```
...
abadie 158 283 284 286 434 435 452 505 832 1027 1485 2522 3400
abandon 2366
abascal 2168
abbaci 1588
abbadi 3457
abbaye 637
abbes 647 2302
abbès 647 1967
abboud 396 1399
abbreviations 929 2622
...
```

4.2 Interrogation de l'index

Une recherche est une liste de mots, aussi appelés mots clés. Cette liste de mots doit être considérée comme un document à comparer avec les autres. La pertinence des résultats correspond à la similarité de ce document avec les autres du corpus.

Chaque mot de la requête va être analysé un par un, et un index correspondant au mot est créé. La commande suivante permet de créer le fichier `i.index` pour le mot `i` de la requête.

```
for i in $(cat query); do
    grep "^$i " index |
  sed 's/[^ ]* //; s/ /\n/g' | sort > $i.index;
done;
```

Après cette étape, il faut créer la réponse de l'utilisateur, elle est contenue dans le fichier `answer`. Le mot de recherche le plus important est le premier, tous les documents dans lesquels il est présents serons retenus si le document en question comporte un des autres mots de la requête. La commande `comm -1 -2` permet de retenir les lignes qui apparaissent dans deux fichiers.

```
# comm -1 -2 : show lines that appear in both files
cp $(head -1 query).index answer;
for i in $(sed 1d query); do comm -1 -2 answer $i.index > tmp; mv tmp answer;
    done;
```

Le fichier `answer` ressemble à :

```
1344
139
1730
1739
1752
...
```

Le document 1344 contient le premier mot de la requête mais aussi un autre parmi "machine" et "learning".

Nous avons donc un document qui rassemble les textes avec des ressemblances à la requête. Il faut désormais calculer le score TF-IDF de chaque mot dans ces documents. Les résultats sont envoyés dans le fichier `query.tfidf`.

```
for i in $(cat ../query); do grep " $i$" ../df.txt; done | awk '{print $2, log
    (3655/$1)}' | awk '{print $2}' > query.tfidf
```

Le contenu de `query.tfidf` est le suivant :

```
1.99325
2.32612
2.477
```

Enfin, l'évaluation des meilleurs documents est réalisée en comparant les documents de la réponse `answer` avec le vecteur des scores TF-IDF des mots de la requête. La pertinence entre les deux documents est réalisée sur la base du calcul d'une norme de vecteurs. Plus elle se rapproche de 1, plus les documents sont similaires.

```
for i in $(sort answer); do
    echo -n "$i ";

    for j in $(cat query); do
      grep "^$j " $i.tfidf | awk '{print $2}';
  # Le tout est mis dans le fichier doc.pert pour pertinence
    done > $i.pert;

  # paste -d" " : merge lines of files with separator " "
    paste -d" " query.tfidf $i.pert | awk '{sum+=$1*$2;norm1+=$1*$1;norm2+=$2*$2
        }END{print sum/sqrt(norm1)/sqrt(norm2)}';
    rm -f $i.pert;
done
```

Pour finir, si la requête comporte les mots "poste machine learning", la sortie de cette commande est :

```
1344 0.926838
139 1
1730 0.943741
1739 0.943741
1752 0.98003
1758 0.897486
1777 0.90769
...
```

Le nom des documents est sur la gauche et sa similarité par rapport au vecteur de la recherche à droite. Le texte 139 à une similarité de 1, tous les mots y sont présents, il est très pertinent.