Corpus

Données Web

HUET Bryan, SAHIN Tolga

Année universitaire 2020

Table des matières

1 Introduction **2**
1.1 Le TF-IDF . . . 2
1.2 Indexage . . . 2

2 Calcul du TF **2**
2.1 Le TF . . . 2
2.2 Calcul du TF . . . 2
2.3 Temps d'exécution avec time . . . 3
2.4 Temps d'exécution avec usr/bin/time . . . 3
2.5 Graphique avec gnuplot . . . 3

3 Le DF **4**
3.1 Définition . . . 4
3.2 Calcul du DF . . . 4
3.3 Temps d'exécution . . . 4
3.4 Temps d'exécution avec usr/bin/time . . . 4
3.5 Graphe avec gnuplot . . . 5

4 Indexage et calcul du TF-IDF **5**
4.1 Premier script . . . 5
4.2 Second script . . . 5
4.3 Troisième script . . . 6
4.4 Quatrième script . . . 6
4.5 Temps d'exécution avec time . . . 7

5 Conclusion **7**

1 Introduction

Le projet de notre corpus est de réaliser un moteur de recherche via des indexage et à l'aide la méthode de pondération TF-IDF.

1.1 Le TF-IDF

Le tf-idf est le poids d'un mot dans un ensemble de documents. Ce poids augmentera en fonction de la fréquence brute des mots (TF) et de la fréquence des mots dans tous les documents. (DF) Si le tf-idf est grand alors plus le mot est fréquent. Tandis que, moins il l'est le mot est peu fréquent.

1.2 Indexage

Le second objectif ce ce projet est de rechercher la fréquence (TF-IDF) de différents mots dans un paquet de documents. De ce fait-là, nous procéderons à un indexage à l'aide de différentes requêtes. Puis, nous calculerons le TF-IDF des mots.

2 Calcul du TF

2.1 Le TF

Le TF est la fréquence des mots dans un document. (Term Frequency)

2.2 Calcul du TF

On commence par récupérer l'ensemble de nos documents (3655 dans notre cas).

Listing 1 – Bouclage

```
for i in $(seq 3655) ; do cat $i | ...
```

On convertit tous nos String majuscule en miniscule.

Listing 2 – Miniscule

```
tr [:upper:] [:lower:] | ...
```

On transforme tous les mots qui utilise des caractères spéciaux, des espaces et des sauts de ligne.

Listing 3 – Caractères spéciaux

```
sed 's/[^a-z]/ /g' | sed 's/ /\n/g' | ...
```

Si le nombre de fields est différent de 0 alors on print les champs de texte. Puis, on pipe l'output avec la requête sort qui permet de trier. Donc, on obtient une liste de mot ordonnée.

Listing 4 – Liste de mots triés

```
awk 'NF != 0 {print}'| sort | ...
```

La requête awk permet de calculer le TF. On calcule depuis notre liste de mots. Le nombre d'occurrences de chacun de ces mots. On retourne le mot et son TF. La requête sed permet de supprimer la première ligne avec le paramètre '1d'. Puis, de stocker les résultats du script (les TF des mots) dans des fichiers .tf jusqu'à la fin de notre boucle.

Listing 5 – Calcul du TF

```
awk '{if (mot == $1) tf ++;
else {print mot, tf; mot = $1; tf = 1}} END {print mot, tf}'
| sed 1d >$i.tf;done
```

2.3 Temps d'exécution avec time

Le script TF prend en moyenne à l'aide la requête time ce délai :

Listing 6 – Suppression

```
time bash df.sh

real 1m18,654s
user 1m0,027s
sys 0m28,940s
```

2.4 Temps d'exécution avec usr/bin/time

Temps d'exécution cette fois-ci calculer avec usr/bin/time.
Pour que cela marche, on passe sur nos script sur un bouclage variable :

Listing 7 – Bouclage variable

```
j=$1
for i in ($(seq $((j * 500))); do ...
```

En terminal nous lancerons de cette manière notre script. Les temps d'exécutions vont être écrites sur un fichier "resultat.txt". Ainsi, on pourra utiliser gnuplot pour observer ceci depuis un graphe.

Listing 8 – Requête d'exécution

```
for j in $(seq 7); do usr/bin/time -p -o ../resultat.txt -a bash tf.sh $j; done
```

2.5 Graphique avec gnuplot

On récupère uniquement les valeurs de real avec la requête awk suivante.

Listing 9 – Fichier à plot

```
awk '/real/{print $2}' resultat.txt > tf.txt
```

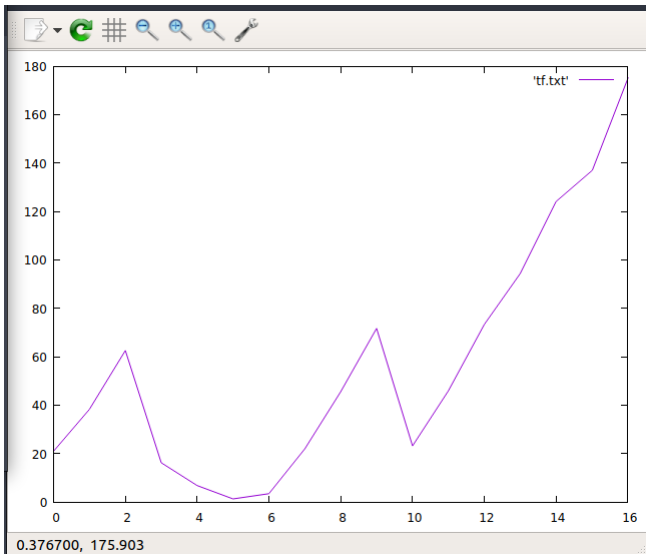

FIGURE 1 – Graphique du temps d'exécution du script tf.sh

Au début, mon temps d'exécution est variable mais sur la fin de la boucle il est presque linéaire.

3 Le DF

3.1 Définition

Le DF (Document Frequency) est le nombre d'occurence de tous les mots dans un ensemble de documents.

3.2 Calcul du DF

Le début du script suit exactement le même principe que pour le TF.

Listing 10 – Bouclage

```
for i in $(seq 3655); do cat $i
| tr [:upper:] [:lower:]
| sed 's/[^a-z]/ /g'
| sed 's/ /\n/g' | ...
```

sort -u est aussi équivalent à sort — uniq. Ce paramètre permet de trier plus efficacement notre ensemble de documents. Les mots dupliqués sont éliminés. Donc, notre output devient la liste de mot de l'ensemble des documents. De plus, "done ;" est important car il ferme notre boucle pour ne garder en sortie que cette liste de mot. On trie cette liste de mot cette fois-ci via la requête sort. On obtient une liste de mots par ordre alphabétique.

Listing 11 – Tris

```
awk 'NF != 0 {print}'| sort -u; done | sort | ...
```

Enfin, on termine le script de la même manière que pour le TF. On compte le nombre d'occurrence de notre liste de mot. Sauf que cette fois-ci notre output est stocké dans un seul fichier texte "df.txt" car on ne boucle plus. Cette output contient le mot et la fréquence de mot par ligne. Donc, l'output est le DF des mots dans nos documents.

Listing 12 – Occurences

```
awk '{if (mot == $1) tf ++;
else {print mot, tf; mot = $1; tf = 1}} END {print mot, tf}'
| sed 1d > ../df.txt
```

3.3 Temps d'exécution

Listing 13 – Temps d'exécution du script df.sh

```
time bash df.sh

real 1m1.042s
user 0m51,006s
sys 0m28,940s
```

3.4 Temps d'exécution avec usr/bin/time

On fait la même chose que dans la partie 2.4 pour le TF.

3.5 Graphe avec gnuplot

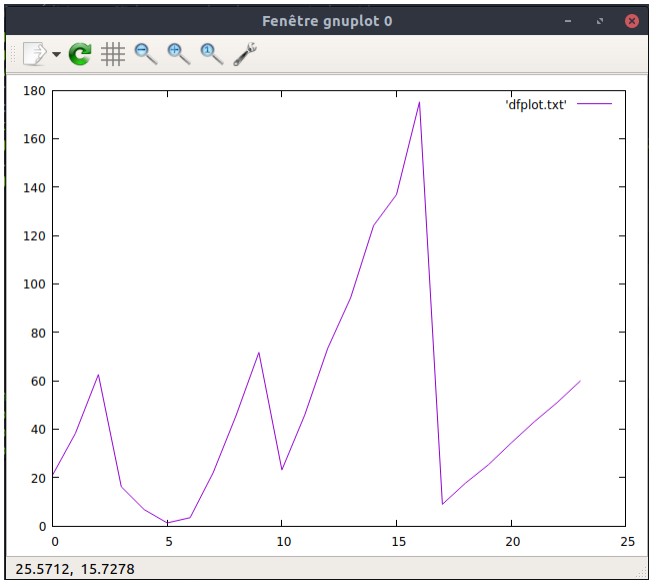

FIGURE 2 – Graphique du temps d'exécution du script df.sh

Les temps d'exécutions sont variables sur certains instants. Mais, il y'a de la linéarité comme pour le tf.

4 Indexage et calcul du TF-IDF

L'indexage et le calcul du TF-IDF va s'effectuer en quatre script.

4.1 Premier script

On récupère le contenu des fichiers .tf (liste de mots avec le TF). On lie chaque mots à une valeur qui correspond au numéro(s) de document(s) dans lequel il s'y présente.

Listing 14 – Requête 1

```
#!/bin/bash
for i in $(seq 3655); do cat "content/"$i.tf
| sed "s/ .*/ $i/"; done | ...
```

Tri les mots par ordre alphabétique et croissant de documents.

Listing 15 – Requête 2

```
sort -k1,1 -k2,2n | ...
```

On supprime les duplications de mots. Puis, on regroupe en une ligne tous les numéros de documents dans lequel se présente le mot. La liste de mots et de numéros est stocké dans le fichier index.

Listing 16 – Requête 3

```
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
```

4.2 Second script

On récupère les mots et les numéros de documents depuis le fichier index de nos mots dans query.

Listing 17 – Requete 1

```
for i in $(cat query.txt); do grep "^$i " index | ...
```

Recupère les numéros de documents.

Listing 18 – Requête 2

```
sed 's/[^ ]* //; s/ /\n/g' | ...
```

Tri des numéros et stockage dans des fichiers mot.index

Listing 19 – Requête 3

```
sort > $i.index; done;
```

On copie les valeurs de chaque fichier index et le stockons dans answer.

Listing 20 – Requête 4

```
cp $(head -1 query.txt).index answer;
```

On compare le fichier "answer" et les fichiers "i.index". Le paramétrage -1 supprime les lignes uniques d'answer et -2 ceux des fichiers index. Puis, stockons les lignes restantes dans tmp. On renomme tmp en answer.

Listing 21 – Requête 5

```
for i in $(sed 1d query.txt);
do comm -1 -2 answer $i.index > tmp; mv tmp answer; done;
```

4.3 Troisième script

On cat query. La requête grep permet de récupérer le DF correspondant à chaque mot de query.

Listing 22 – Requête 1

```
for i in $(cat query.txt); do grep "^$i " df.txt; done | ...
```

On effectue le calcul de TF-IDF avec log(nombres de fichiers / valeurs de df). On stocke les tf-idf dans query.tfidf.

Listing 23 – Requête 2

```
awk '{print $1, log(3655/$2)}' | awk '{print $2}' > query.tfidf
```

4.4 Quatrième script

La première boucle récupère les numéros de documents sans duplication depuis answer. La seconde boucle, à l'aide de la requête grep permet de relier les numéros de documents leur mot et leur tf-idf en une ligne.

Listing 24 – Requête 1

```
for i in $(sort answer); do echo -n "$i ";
for j in $(cat query); do grep "^$j " $i.tfidf | ...
```

On récupère la pertinence des mots de chaque mots dans les précédents numéros de documents (i). Puis, on stocke ce résultat dans des fichiers "i.pert". Paste permet d'écrire les lignes tabulairement (-d) de query.tfidf et des fichiers i.pert correspondant.

Listing 25 – Requête 2

```
awk '{print $2}'; done > $i.pert; paste -d" " query.tfidf $i.pert | ...
```

On effectue le calcul sur nos valeurs comme sur la formule du TF-IDF. Puis on supprime tous les fichiers i.pert pour de la lisibilité.

Listing 26 – Requête 3

```
awk '{sum+=$1*$2;norm1+=$1*$1;norm2+=$2*$2}END{print sum/sqrt(norm1)/sqrt(norm2)}';
rm -f $i.pert; done
```

L'output sera vu sous terminal, on observe ainsi chaque numéro de documents (où nos query apparaissent) et sa valeur de pertinence sur chaque ligne.

4.5 Temps d'exécution avec time

Script 1 :

Listing 27 – Temps d'exécution du script 1

```
time bash step1.sh

real 0m4,744s
user 0m6,020s
sys  0m1,207s
```

Script 2 :

Listing 28 – Temps d'exécution du script 2

```
time bash step2.sh

real 0m0,265s
user 0m0,039s
sys  0m0,008s
```

Script 3 :

Listing 29 – Temps d'exécution du script 3

```
time bash step3.sh

real 0m0,013s
user 0m0,007s
sys  0m0,012s
```

Script 4 :

Listing 30 – Temps d'exécution du script 4

```
time bash step4.sh

real 0m0,197s
user 0m0,231s
sys  0m0,071s
```

Total :

Listing 31 – Temps d'exécution en tout

```
time bash tfidf.sh

real 0m5,219s
user 0m6,297s
sys  0m1,298s
```

5 Conclusion

Notre moteur de recherche permet bien de récupérer depuis des mots les TF, DF, TF-IDF. Les temps d'exécutions eux varient considérablement en fonction des requêtes que l'on utilise. Notre temps d'exécution moyen nous semble correct