RAPPORT PROJET JAVA

Text-Mining

Hoang Khanh LE - Mathieu LIVEBARDON
Master 1 Informatique

Sommaire

Introduction

Afin de pratiquer notre compétence de Java, on devrait créer un interface qui fait une analyse sur la fouille de texte de tweets sur Twitter.

Notre programme/ interface est au but de faire un fouille de texte en analysant les comportements des utilisateurs selon leur contenu. Donc ça permet de faire un tri sur les dates, utilisateur et nombre de retweets. Notre programme propose aussi un bar de recherche sur l'utilisateur et le contenu de tweet. Pour la partie statistique, l'algorithme permet d'informer sur les termes les plus important (nombre d'occurrences, TF-IDF)

Pour expliquer ce projet nous devons livrer un rapport pour renseigner le contexte de travail et les spécifications du projet.

I. L'analyse

L'environnement de travail

IntelliJ

Au lieu de programmer Java sur Eclipse ou NetBeans, IntelliJ était notre choix pour son interface et son pratique de synchronisation. IntelliJ IDEA également appelé « IntelliJ », « IDEA » ou « IDJ » est un environnement de développement intégré de technologie Java destiné au développement de logiciels informatiques.

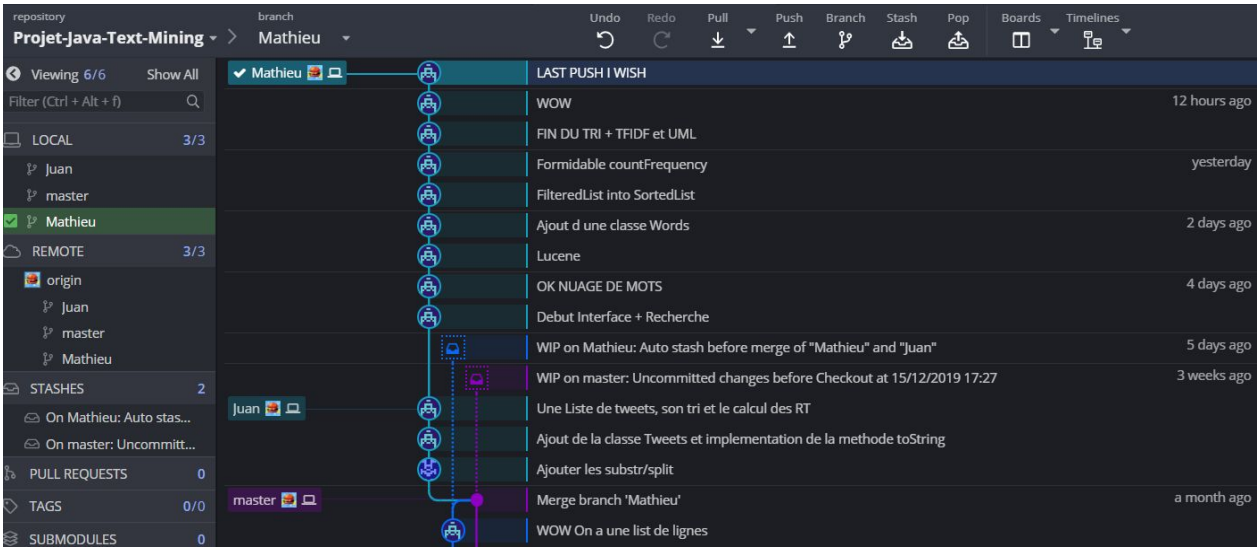
Remarque : IntelliJ nous donne la possibilité d'exporter le projet pour Eclipse.

GitHub – GitKraken

Afin de faciliter la synchronisation de travail en équipe, GitHub était conseillé par les enseignants. Il s'agit d'un service en ligne open source qui permet de sauvegarder ses répertoires et fichiers de code tout en les versionnant. L'ensemble des personnes autorisées peuvent ensuite y avoir accès.

GitKraken, une autre l'interface de Github, nous avait beaucoup aidé de suivre la progresse de travail de chacun ainsi les taches partagées/communes.

Il y a 3 branches, 1 chacun et 1 commun. Chaque fois une tache de travail est fini, il faut pusher ces taches sur la branche de chacun puis fusionner cette branche avec la branche commune. Parfois, c'est possible d'avoir les conflits quand on modifie les mêmes fichiers, GitKraken nous averti et nous demande ce que nous voulons garder ou supprimer.



Les données

On travaille sur les deux bases de données de tweets des utilisateurs Twitter. La première est une base de climat et la deuxième est une base de foot.

Ignorer les fichiers

Comme les bases de données sont assez grands, le climat.txt est à 531MB et le foot.txt est à 186MB, malheureusement on ne peut pas push sur l'environnement commun Git. Pour que le programme Java marche, on devait faire en sorte que le Git ne prend pas en charge ces fichiers, autrement dit, il faillait ignorer ces fichiers. Pour faciliter le temps d'exécution de programme, nous avons créé une base de données de 100 lignes.

Nettoyage de données

L'idée est d'enlever les lignes en mauvais formats comme ça fait arrêter le lancement de programme ou ça dérange le résultat obtenu.

Pour la base de données de Climat, on a supprimé les lignes 37380 et 37381

La base de données de Foot, on a supprimé les caractères « 000 » sur les lignes 37 et 38 et aussi les lignes 717677 et 717678 pour leurs mauvais formats

Structure d'un tweet

Donc pour comprendre concrètement de ce que chaque colonne présente dans chaque ligne de la base de donnée.

Par exemple :

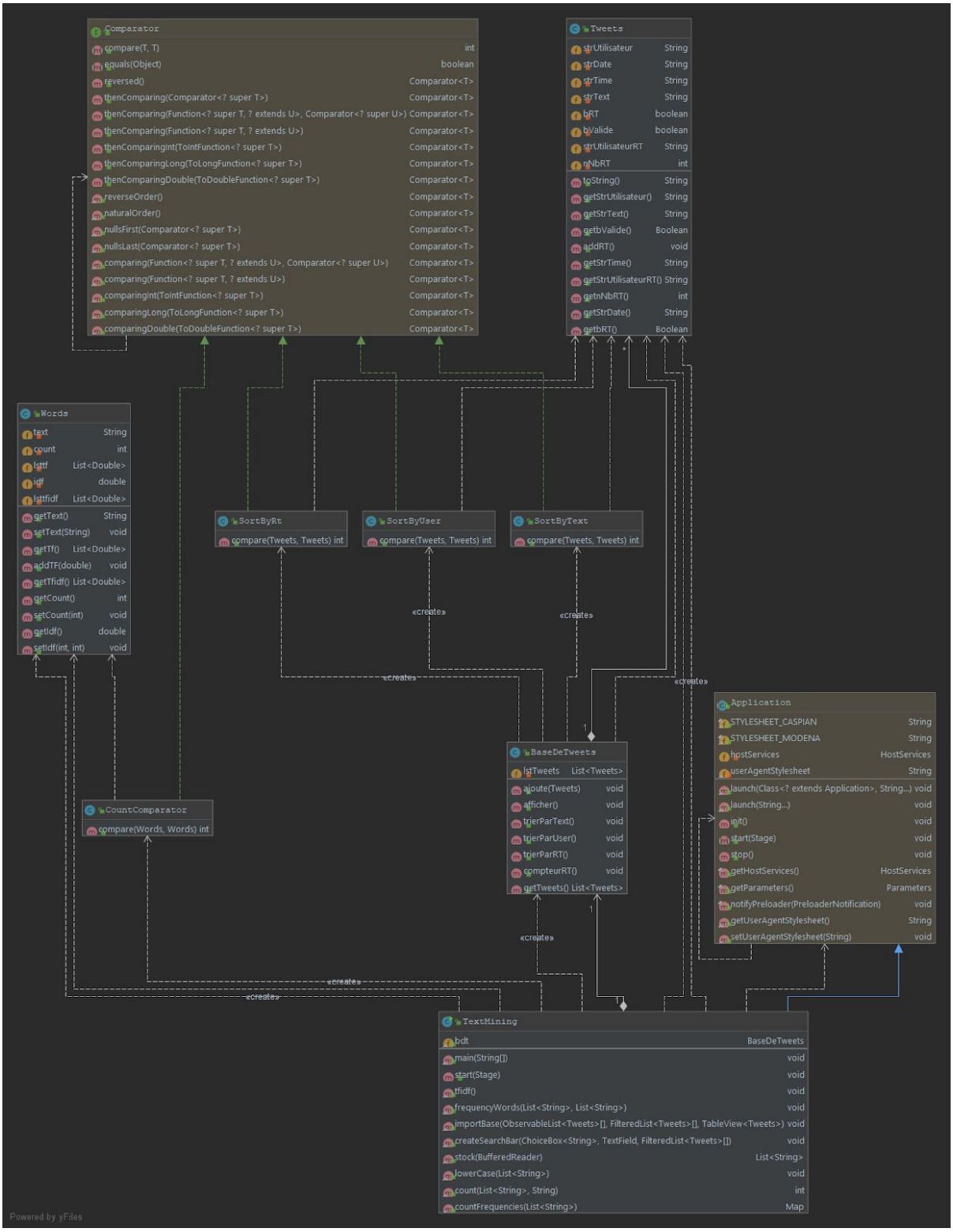
000 Pastore__ 2019-06-21 11:46:19.686000 Un de mes joueurs préférés, merci pour tout El Niño ❤
000 r0msev 2019-06-21 11:47:25.203950 Un de mes joueurs préférés, merci pour tout El Niño ❤ https://t.co/g3asQwWFaJ Pastore__

On voit bien que le premier tweet est le tweet original de Pastore__ et la deuxième est le retweet de compte r0msev

Ici, ce qui nous concerne est le nom de l'utilisateur, la date, le contenu de tweet et l'utilisateur original si c'est un retweet. On va enlever les colonnes inutiles comme le temps exact, le liens URL hors contenu, ect.

User ID	Date	Texte	RTID
Pastore__	2019-06-21	Un de mes joueurs préférés, merci pour tout El Niño ❤	
r0msev	2019-06-21	Un de mes joueurs préférés, merci pour tout El Niño ❤	Pastore__

Diagramme de classe



Remarque : Les méthodes et champs publics sont en vert et les privés sont en rouge.

Spécification de chaque classe

TextMining :

- C'est la classe principale, elle hérite d'Application. C'est ici que le programme est lancé et que l'interface est crée (*main(), start()*). Elle s'occupe de l'importation de données (*start(), stock(), importBase(), createSearchBar()*).
- Méthodes :
    - tfidf() : Elle découpe tous les tweets en mot. Pour chaque tweet elle crée un objet Words pour chaque mot et l'ajoute dans une liste de mot. Avant de l'ajouter elle vérifie si le mot est déjà créé, si c'est le cas on le met à jour (compteur d'occurrences et calcul du tf du tweet en cours). Ensuite une fois que la liste est crée, elle parcourt celle-ci pour calculer l'idf et le ou les tf-idf).
    - frequencyWords() : Elle est très similaire à tfidf() seulement elle ne va qu'informer sur le nombre d'occurrences des mots.
    - importBase() : Elle importe la base dans le TableView dans l'interface.
    - createSearchBar() : Elle crée la barre de recherche dans l'interface.
    - stock() : Elle parcourt la base de données et nous retourne une liste de chaine de caractères (une chaine de caractère par ligne). **Attention** elle s'arrête à 100 000 lignes (une base plus grande pose des problèmes dans le calcul du tf-idf, c'est trop long).
    - lowerCase() : Elle permet de transformer tous les éléments d'une liste de chaine de caractère en minuscule.
    - count() : C'est le premier compteur du nombre d'occurrences que nous utilisions, il fonctionne bien sur les petites bases mais sa complexité algorithmique est trop grande.
    - countFrequencies() : C'est finalement la solution que nous avons préférée pour compter la fréquence des mots dans une liste. **BEAUCOUP** plus rapide à l'exécution.

Tweets :

- Une classe qui permet surtout de bien diviser les différentes informations d'un tweet dans différents champs :
    - Utilisateur : une chaine de caractères avec le nom de l'utilisateur qui tweet.
    - Date : une chaine de caractère avec la date d'envoi du tweet.
    - Time : une chaine de caractère avec l'heure d'envoi du tweet.
    - Text : une chaine de caractère avec le contenu du tweet.
    - RT : un booléen qui informe sur le type de tweet (si c'est un retweet ou pas)
    - Valide : un booléen qui vérifie dans le constructeur si la chaine de caractère respecte les conditions d'un tweet valide (nombre de champs suffisant).
    - UtilisateurRT : s'il s'agit d'un retweet, une de caractère avec le nom de l'auteur du tweet original.
    - NbRT : un nombre entier qui contient le nombre de RT d'un tweet.

BaseDeTweets :

- Une classe qui contient une liste de Tweets. Elle s'occupe des différents tris et du compteur de retweet. Nous n'instancierons qu'une fois cette classe dans TextMining.

Words :

- Une classe pour les mots. Pour chaque mot unique de chaque tweet de la base de donnée, nous allons stocker plusieurs informations dans autant de champs :
    - text : une chaine de caractère qui contient le mot.
    - count : un entier qui contient le nombre d'occurrences du mot dans la base de donnée.
    - lsttf : une liste de nombres décimaux qui stock tous les coefficients TF du mot dans les tweets où il apparait.
    - idf : un nombre décimal qui contient le coefficient IDF du mot dans la base.
    - lsttfidf : une liste de nombres décimaux qui stock tous les résultats des calculs TFIDF du mot.

SortByRT, SortByUser, SortByText, CountComparator :

- Ce sont des classes qui implémentent la classe Comparator. Elles vont nous servir pour différents tris. Notamment le tri de la base par Utilisateur, Contenu du Tweet et Nombre de Retweets. Mais aussi le tri de la liste de Words en fonction du nombre d'apparitions du mot dans la base.

Normalisation

La normalisation de texte est une méthode de pondération utilisée en text mining. Cette méthode permet d'évaluer l'importance d'un terme contenu dans un document, relativement à une collection ou un corpus. Le poids augmente proportionnellement au nombre d'occurrences du mot dans le document. Il varie également en fonction de la fréquence du mot dans le corpus. Des variantes de la formule originale sont souvent utilisées dans des moteurs de recherche pour apprécier la pertinence d'un document en fonction des critères de recherche de l'utilisateur.

Fréquence des termes – TF

La Term Frequency décrit la fréquence d'un certain terme apparaît dans un document par rapport à tous les autres termes contenus dans le document. Autrement dit, c'est un indicateur d'importance du terme dans le document. La formule est :

$$TF(i) = \frac{log_2(Freq(i,j) + 1}{log_2(L)}$$

i	Terme dont le TF dans le document doit être déterminé
j	Document analysé
Lj	Nombre total de mots dans le document « j »
Freq(i,j)	Fréquence d'un mot « i » dans le document « j »
log2	Logarithme du nombre x en base 2

Ça aide à prendre en charge la répétition d'un terme dans le document est prise en compte afin de déterminer les mots clés.

Inverse document frequency - IDF

Un terme présent dans presque tout le corpus (D) influe peu quand il apparaît dans un document. A l'inverse, un terme rare apparaissant dans un document doit retenir notre attention. L'IDF mesure l'importance d'un terme dans un corpus. La formule est :

$$IDF(i) = \log\left(\frac{N_D}{f_i} + 1\right)$$

i	Terme dont l'inverse Document Frequency doit être déterminé
log	Logarithme du nombre x en base 10 ou en toute autre base b
ND	Nombre de tous les documents dans le corps du document (qui contiennent les termes pertinents)
fi	Nombre de tous les documents dans lesquels le terme « i » apparaît

Pondération TF-IDF

Comme l'IDF reflète le rôle d'un terme par rapport à tous les documents d'un corpus, et le TF représente la pertinence d'un terme dans un document donné, la combinaison des deux nous permet de bien comprendre la fréquence réelle des termes et le potentiel de chaque terme pour optimiser le texte

existant. Ça indique l'importance d'un terme dans un document (TF) par son importance dans le corpus (IDF)

$$\mathrm{TF(i,j)} = TF_{i,j} * IDF_i$$

II. La conception

Les taches partagées

En fait, chacun dans le binôme a une compétence différente. Khanh a plus des compétences sur partie statistique, donc text mining, c'est aussi pourquoi le sujet nous intéresse le plus. Mathieu a déjà appris Java donc il est beaucoup plus à l'aise à coder les programmes plus complexes. Donc au début, Khanh a fait les recherches sur l'importation et ainsi coder celle-ci. Dans le même temps, Mathieu commençait faire le début de l'interface. Ensuite, nous avons travaillé ensemble sur le découpage de données et le tri. Ces parties sont implémentées sur l'interface par Mathieu. Pour la partie analyse statistique, Khanh était chargé de comprendre la problématique de chaque tache puis Mathieu faisait la traduction de tous ça en Java.

L'idée était d'apprendre l'un à l'autre, chaque fois Khanh fait des recherches des bouts de codes et essayer d'appliquer dans le programme, qui sera vérifié et évalué par Mathieu. En revanche, grâce aux connaissances de Khanh, Mathieu a aussi bien compris c'est quoi Text Mining, TF IDF...

Relations d'utilisations

Pour les relations d'utilisations nous pouvons les distinguer dans le diagramme de classe. La structure est basée sur les TDs que nous avons fait pendant le semestre :

- Tweets a une relation d'utilisation avec BaseDeTweets qui contient une liste de Tweets.
- BaseDeTweets elle-même a une relation d'utilisation avec TextMining qui contient un seul et unique objet BaseDeTweets.

Ces relations nous ont permis de séparer la partie stockage et la partie traitement de données.

Remarques :

- Il aurait été logique que BaseDeTweets soit un Singleton.
- Words aurait pu subir le même traitement que Tweets (avec une BaseDeWords).

Problèmes rencontrés

L'apprentissage d'outils comme GitHub, effectivement il n'est pas évident à prendre en main.

Au début on a choisi d'utiliser Treeset pour stocker les chaine de caractères des tweets, en se basant sur ce qu'on a fait pendant les TDs. Malgré le côté pratique d'apparence pour trier, seulement il n'accepte pas les doublons donc les RT. Nous avons choisi d'utiliser essentiellement des ArrayList<>.

Travailler avec des bases de données si grande nous a souvent bloqué dans l'exécution de certaines fonction. Nous avons cherché à optimiser le plus possible la complexité de notre code. Cela c'est surtout ressentit lors du calcul du TF-IDF, lors du calcul du nombre d'apparitions des mots l'algorithme était en complexité exponentielle. Nous avons donc trouver une façon plus rapide pour ce calcul. Cependant, pour calculer la fréquence d'un mot dans chaque tweet ça reste encore maintenant très long. C'est pourquoi nous avons limité à 100 000 lignes de données la taille de la base.

Nous avons décidé malheureusement d'abandonner la partie sur le Clustering. Nous ne savions aucunement comment l'implémenter et le temps nous a manqué.

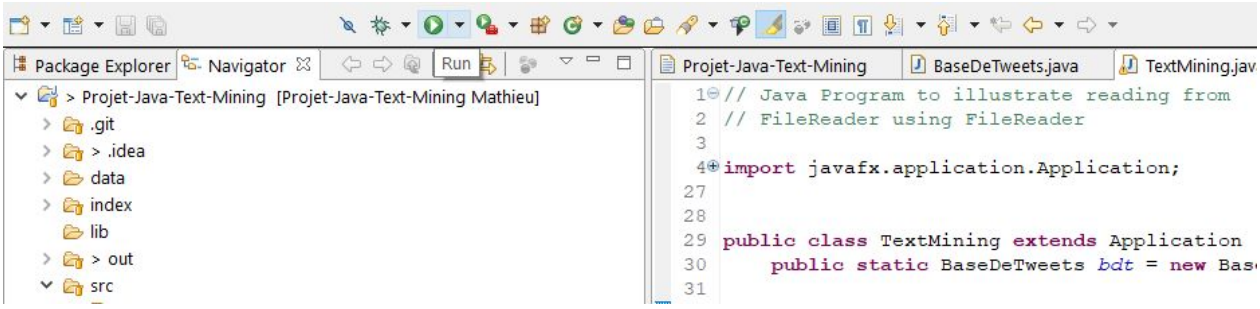
Remarque : Nous avons essayé d'utiliser des librairies, mais celle-ci nous empêchait de comprendre le corps du code. Nous avons décidé de coder toutes les fonctions nous même avec des morceaux de codes que nous comprenions, cela a sans doute eu un impact sur l'optimisation des algorithmes.

Exemple commenté

Prérequis :

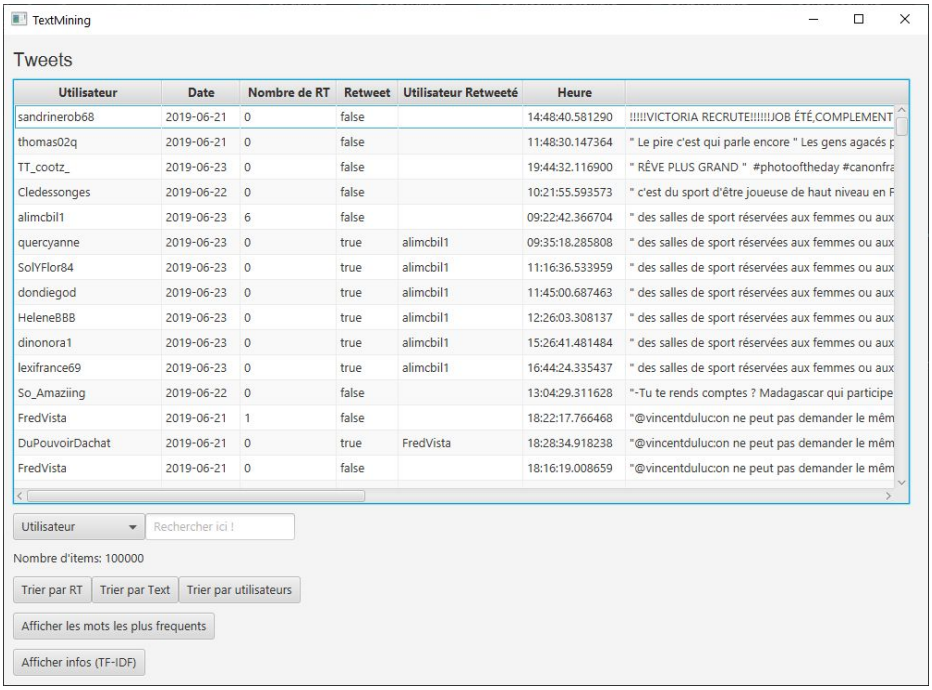
- Assurez vous d'avoir bien les fichiers StopWords.txt et Foot.txt dans le dossier data du projet.
- Certaines informations seront écris dans la console de l'IDE que vous utiliserez.

LANCEMENT DE L'APPLICATION



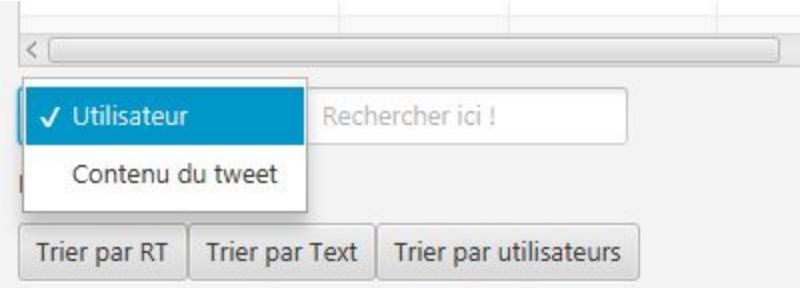
Une fois le projet ouvert sur votre IDE Java favoris, il ne vous reste plus qu'à exécuter le fichier TextMining.java. L'application va se lancer.

L'APPLICATION

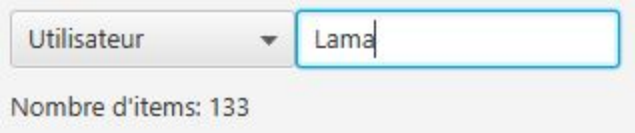


L’application crée une interface qui permet l’importation des données dans un tableau. Chaque Tweets de la base sont dans ce tableau trié par défaut par contenu.

LA RECHERCHE

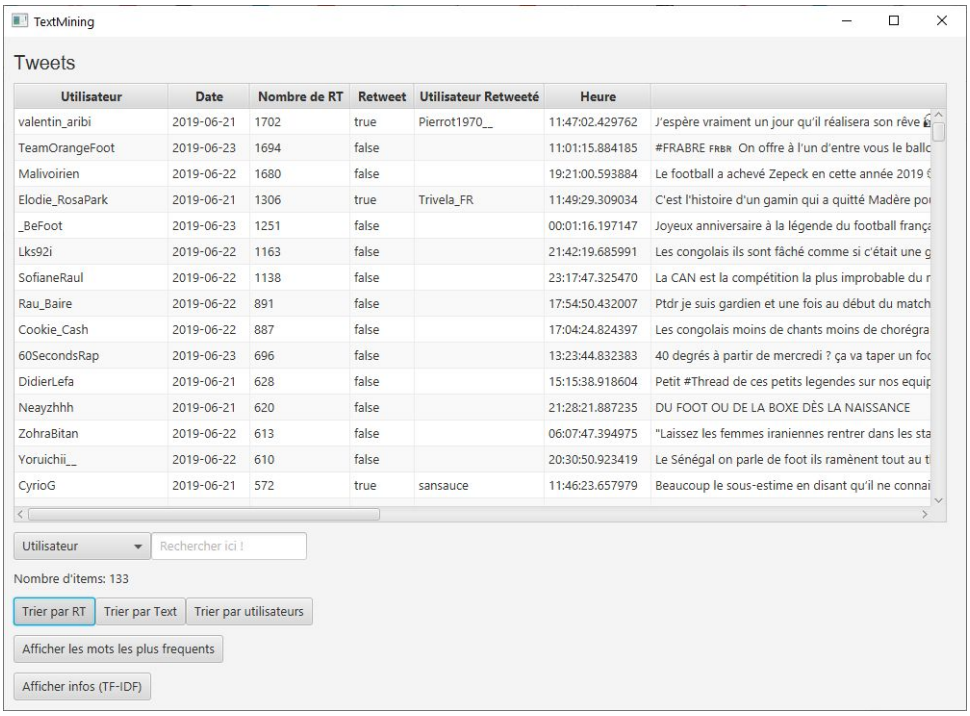


La première fonctionnalité disponible est la barre de recherche avec la possibilité de rechercher par utilisateur ou par contenu du tweet.



Nous connaissons aussi le nombre de résultats, par exemple il y a exactement 133 utilisateurs qui contienne la suite de lettre lama dans leur nom.

LES TRIS

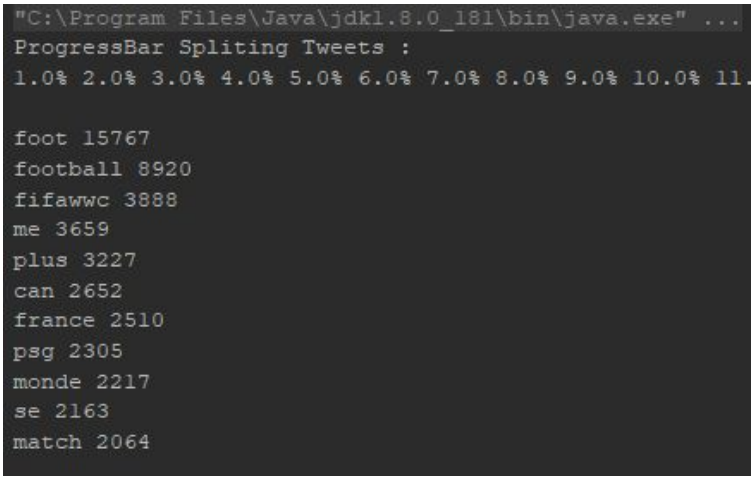


Les boutons tris permettent comme leur nom l'indique de trier la base de donnée en fonction du nombre de RT, du contenu du tweet ou de nom de l'utilisateur.

Remarques : Les tris ne sont exécuter que de manière ascendante et les boutons tris recharge la base de donnée, par conséquent elle ne garde pas en mémoire une recherche précédente.

AFFICHER LES MOTS LES PLUS FREQUENTS

ATTENTION : Tout se passe malheureusement dans la console.



Lorsque le bouton associés est actionné, un indicateur de progression commence à défiler dans la console (il indique le pourcentage de Tweets traités dans la boucle de la fonction frequencyWords()).

Ensuite il affiche le classement des 11 premiers mots les plus utilisés dans la base de données.

Remarques : Seul les mots ne figurant pas dans le StopWords.txt sont comptés. Il ne prend pas en compte s'il y a une recherche en cours. Il ne compte que les mots des tweets originaux.

AFFICHER INFOS (TF-IDF)

Meme consigne que pour le précédent bouton. **Tout se passe dans la console.**

```
ProgressBar Spliting Tweets :
1.0% 2.0% 3.0% 4.0% 5.0% 6.0% 7.0% 8.0% 9.0% 10.0% 11.0% 12.0% 13.0% 14.0% 15.0% 16.0% 17.0% 18.0% 1

est
c : 11941
idf : 4.1109
tf : [0.0263, 0.04, 0.0333, 0.037, 0.0208, 0.0909, 0.0154, 0.0417, 0.0179, 0.0625, 0.0192, 0.0455, 0
tf-idf : [0.10811667, 0.164436, 0.13689297, 0.1521033, 0.08550672, 0.37368081, 0.06330786000000001,

les
c : 10873
idf : 4.2047
tf : [0.0244, 0.0476, 0.0769, 0.0465, 0.0638, 0.0217, 0.04, 0.0227, 0.0227, 0.0185, 0.0208, 0.0303,
tf-idf : [0.10259468000000001, 0.20014372, 0.32334143, 0.19551854999999999, 0.26825985999999996, 0.0

et
c : 10000
idf : 4.2905
tf : [0.0244, 0.0263, 0.0323, 0.0556, 0.0233, 0.0213, 0.0185, 0.0208, 0.0909, 0.0154, 0.0536, 0.0208
tf-idf : [0.1046882, 0.11284015, 0.13858315, 0.23855179999999998, 0.09996865, 0.09138764999999999, 0

du
c : 9544
idf : 4.3438
tf : [0.0417, 0.0526, 0.0161, 0.0233, 0.1, 0.0426, 0.0385, 0.0217, 0.0175, 0.0185, 0.0208, 0.0303, 0
tf-idf : [0.18113646, 0.22848388, 0.06993518, 0.10121054, 0.43438, 0.18504588, 0.1672363, 0.09426046

football
c : 8920
idf : 4.4067
tf : [0.0667, 0.0417, 0.0476, 0.0263, 0.0769, 0.0233, 0.0217, 0.0303, 0.0263, 0.0172, 0.0417, 0.0217
tf-idf : [0.29392689, 0.18375939, 0.20975892000000002, 0.11589621, 0.33887522999999997, 0.10267611,
```

Il y a encore un indicateur de progression. Cette fois-ci cela risque d'être **beaucoup** plus long.

Une fois les calculs terminés, nous affichons encore une fois les 11 mots les plus utilisés dans les tweets de la base, mais cette fois accompagner de leurs TF-IDF des tweets où ils apparaissent.

Remarques : Cette fois-ci tout les mots sont pris en compte, sinon nous retrouvons les mêmes conditions que pour le précédent bouton.

III. La validation

Nettoyage

Pour avoir une analyse plus effective sur une base de mots pertinents, on doit supprimer les mots inutiles. C'est pourquoi on a récupéré une base de données de mots vide français, ce qui porte aucun sens dans un document (ex. préposition, pronoms, etc), pour éviter l'erreur de la normalisation. Au fur à mesure, on ajoute les mots vides selon le résultat pour obtenir une analyse plus concrète.

L'autre côté, on devait aussi supprimer tous les caractères spéciaux.

Tests

Pour tous ce qui est test du code, la première chose que nous avons faite c'est de créer un base de tweets plus petites (FootLight.txt avec les 100 premiers tweets de Foot.txt). Ensuite pour résoudre la plupart des problèmes c'était du débogage classique (points d'arrêts...) ou des infos écrits dans la console (System.out).

IV. La maintenance

L'application que nous avons produite répond dans l'ensemble aux objectifs de l'axe. Cependant l'objectif principal étant de faciliter l'exploration des données, cet objectif reste partiellement rempli.

En effet, il aurait été plutôt appréciable d'avoir une fonctionnalité qui permette de naviguer d'un tweet à un autre grâce à une sélection. Exemple accéder à tous les tweets d'un utilisateur en cliquant dessus ...

Ensuite, nous sommes arrivés à calculer le TF-IDF de chaque mots, c'est bien. mais nous ne nous en servons pas.

Et pour finir le Clustering, même si ne devons avouer ne pas trop savoir comment l'implémenter dans notre code actuel, c'était un objectif tout de même.

Pour ce qui est de la faisabilité, je pense que le code est assez modulable, même si la séparation de responsabilité des classes n'est pas parfaite, le code laisse la place pour ajouter les éléments manquants. Pour répondre à la question si ce sera difficile, alors oui mais nous pourrons compter sur les compétences que nous avons acquises jusqu'ici.

Table des matières