Johann Mitrail
Anthony Rossi

Rapport de projet extraction de données

Introduction

Ce projet d'extraction de connaissances et de données a été réalisé par Johann Mitrail et Anthony Rossi, étudiants de la filière AIGLE.
Son but était de nous faire travailler sur un corpus comprenant un nombre important de textes et structuré de manière complexe. Une fois ce corpus préparé et "décortiqué", nous devions y appliquer différents algorithmes de fouille pour en tester l'efficacité et pouvoir comparer les résultats obtenus.

Préparation du corpus de textes

Notre corpus de texte est celui donné en exemple par l'enseignant composé de commentaires d'hôtels et autres lieux touristiques récoltés sur le site de TripAdvisor. Ces textes sont donc organisés de manière similaire et comprennent toujours les mêmes informations. Parmi ces dernières, nous retiendrons celles qui nous intéressent et qui seront utiles pour évaluer la "positivité" (ou non) d'un avis : le commentaire et la note. Celle-ci est effectuée sur cinq et se compose de huit critères dont nous ferons à chaque fois la moyenne. Notre but était donc d'évaluer l'opinion des internautes, pour cela il faut donc classer ces commentaires selon qu'ils sont positifs, neutre ou négatifs. Nous faisons cette opération tout d'abord en nous appuyant seulement sur les notes, positif étant supérieur ou égal à 4 et négatif inférieur ou égal à 2.

Nettoyage des données

Dans un premier temps, ils nous a fallu analyser l'ensemble de notre corpus. Chaque fichier et commentaire ayant une structure semblable, nous avons mis leur contenu dans une hashmap (car nous avons fait toute cette partie de préparation des textes en Java) en prenant soin de réunir chaque mot avec son nombre d'occurrences. Comme énoncé précédemment, les huit notes sont moyennées et ce chiffre est aussi placé dans la hashmap. Dans un souci d'efficacité, les ponctuations et majuscules ont aussi été retirées car elle sont négligeables quant à l'utilité pour l'analyse de la sensation que renvoie le texte.

C'est maintenant qu'interviennent les notions de positif, neutre et négatif vues dans la partie d'avant : chaque commentaire est placé dans un dossier différent selon la catégorie dans laquelle il est considéré en fonction du critère de notation explicité ci-dessus. On regroupe ensuite toutes les hashmaps classées par "positivité" dans trois ArrayLists, un pour les hashmaps positives, neutres et négatives.
Arrive maintenant la phase où il faut lemmatiser nos mots c'est-à-dire les simplifier au maximum. Un mot pluriel sera réduit au singulier, un verbe sera toujours placé à l'infinitif par exemple. Cette étape se fait en même temps que le remplissage des ArrayLists de la partie précédente. Pour ce faire, nous avons utilisé la librairie Lucene intégrée sur la plate-forme Eclipse.

A partir de là intervient un nouveau concept : les mots vides. Ces derniers sont en fait des mots qui ne portent pas vraiment de sens intervenant dans l'analyse et la fouille de textes, le plus souvent il s'agit d'articles ou de mots de liaison comme des conjonctions de coordination (mais, une, ou, …). Nous avons donc récupéré un fichier contenant une liste de mots vides anglais, puisque tous les commentaires sont dans cette langue. On parcourt ensuite nos ArrayLists à la recherche de mots vides contenus dans ce fichier. Lorsque l'on vient à tomber sur l'un d'eux, nous le supprimons.

La partie mathématique se révèle ensuite quand vient l'application de la formule de TF-IDF (Term Frequency-Inverse Document Frequency). Rappelons la formule vue dans le cours :

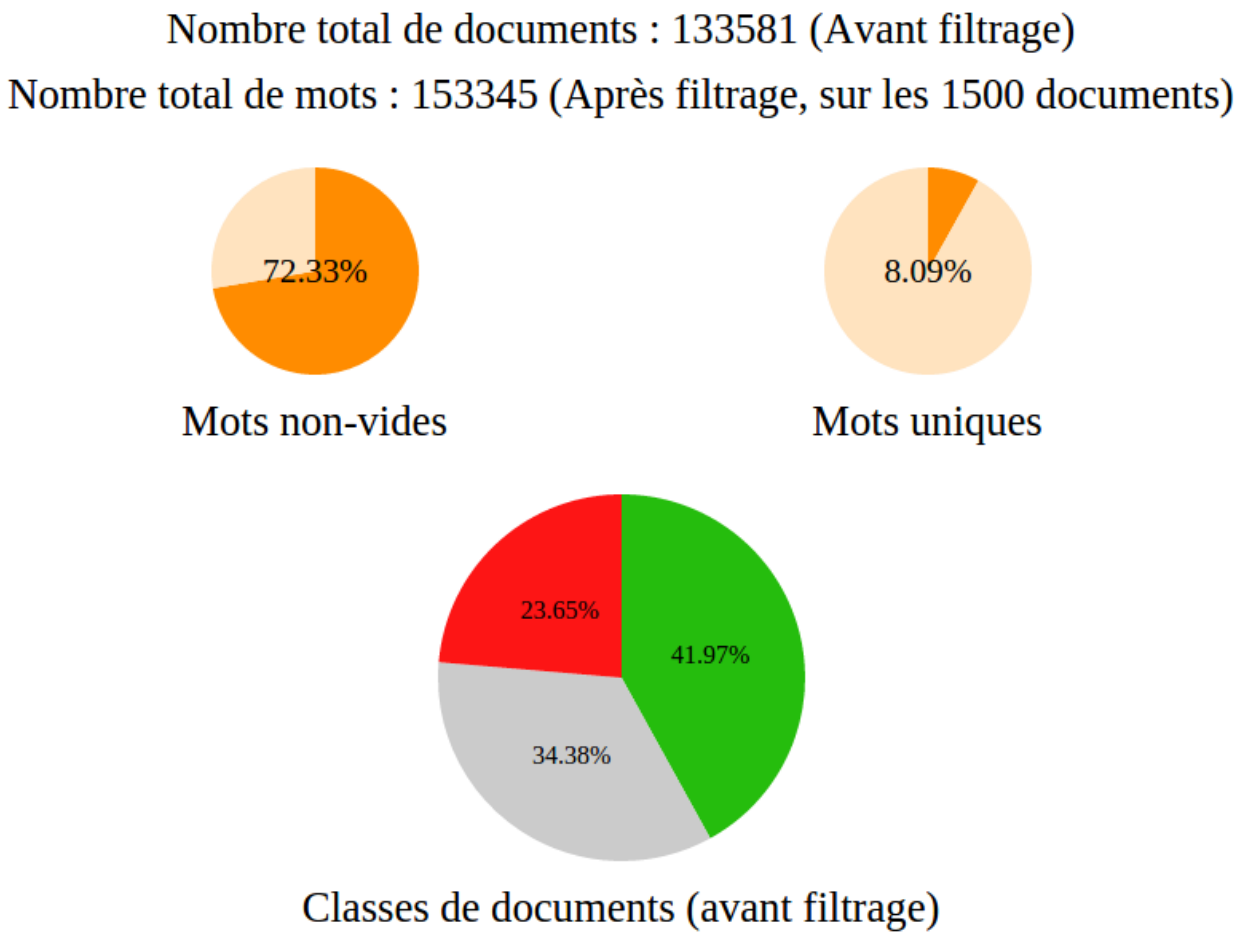
```
TF-IDF(a) =
(frequence_de_a_dans_le_doc_x / frequence_max_de_a_dans_le_doc_x )
* log(nombre_de_docs / nombre_de_docs_contenant_a)
```

Grâce au logiciel Weka, nous avons appliqué cette formule qui permet de garder les mots à plus faible fréquence qui sont estimés plus représentatifs selon cette théorie. Le résultat est compris entre zéro et un. Cette méthode est donc mise en pratique sur tous nos textes pour chaque mot. On extrait ensuite ces résultats dans un fichier .arff, format pouvant être lu par le logiciel Weka. Ce fichier est en fait un énorme tableau à deux dimensions comprenant d'un

côté les mots des documents et d'un autre côté la liste des documents dont la jonction donnera le fameux taux TF-IDF.

Visualisation des données

Après l'application de ces étapes, le programme Java était en mesure de nous retourner des informations numériques comme le nombre de documents positifs, neutres et négatifs ou encore le nombre de mots total, de mots vides et le nombre de mots uniques et lemmatisés. Ces informations ont été visualisées grâce à la librairie javascript d3.js déjà utilisée dans une autre UE l'année précédente et conforme à ce que nous voulions en faire. Ainsi, sur un camembert, nous observerons la "positivité" des documents en voyant quelle opinion ressort le plus majoritairement. Nous pouvons aussi observer le pourcentage de mots uniques ou vides sur deux autres camemberts.



Application des algorithmes

L'un des buts principaux de ce projet étant d'appliquer différents algorithmes de fouille de données pour les comparer et voir leurs résultats, nous avons utilisé Weka qui offre cette possibilité tout en nous fournissant des statistiques pertinentes sur le contenu de nos textes.

Nous allons donc commencer par cerner les mots les plus pertinents grâce au fichier .arff produit précédemment car, en effet, la pertinence est proportionnelle au taux TF-IDF. Suite à cette étape, Weka appliquera alors divers algorithmes d'analyse supervisée (K plus proches voisins, arbres de décisions, naïve bayes, ensemble classifieur) comme non supervisée (K moyenne, clustering hiérarchique ascendant) pour nous fournir des résultats analysables et comparables.

Sans sélection des attributs

J48 : Arbre de décision tree

Instances classées correctement : **59.33%**
Instances mal classées : 40.67%

Documents\Calculés	Négatif	Positif	Neutre
Négatif	**241**	174	85
Positif	144	**322**	34
Neutre	110	63	**327**

SVM

Instances classées correctement : **64.26%**.

Instances mal classées : 35,74%.

Documents\Calculés	Négatif	Positif	Neutre
Négatif	**278**	141	81
Positif	116	**336**	48
Neutre	106	44	**350**

K plus proches voisins

K=1

Instances classées correctement : **27.93%**

Instances mal classées : 72.06%

Documents\Calculés	Négatif	Positif	Neutre

Négatif	**180**	314	6
Positif	273	**220**	7
Neutre	224	257	**19**

K=9

Instances classées correctement : **33.93%**

Instances mal classées : 66.06%

Documents\Calculés	Négatif	Positif	Neutre
Négatif	**1**	497	2
Positif	1	**499**	0
Neutre	1	489	**9**

NaiveBayes

Instances classées correctement : **61.8%**

Instances mal classées : 38.2%

Documents\Calculés	Négatif	Positif	Neutre
Négatif	**272**	131	97
Positif	145	**332**	23
Neutre	92	85	**323**

Observations

Les résultats des différents algorithmes de fouille de donnée sont clairement distincts.
Les algorithmes de machine à états de vecteurs et ceux l'arbre de décision et Bayes sont clairement plus performants contrairement à la méthode K des plus proches voisins qui a beaucoup plus de mal à classer correctement les instances.

Avec sélection des attributs

trees/J48

Instances classées correctement : **75%**

Instances mal classées : 25%

Documents\Calculés	Négatif	Positif	Neutre
Négatif	**374**	83	43
Positif	63	**397**	40
Neutre	73	73	**354**

SVM

Instances classées correctement : **84.93%**

Instances mal classées : 15,07%

Documents\Calculés	Négatif	Positif	Neutre
Négatif	**429**	55	16
Positif	31	**449**	20
Neutre	59	45	**396**

K plus proches voisins

K=1

Instances classées correctement : **61,66%**

Instances mal classées : 38.33%

Documents\Calculés	Négatif	Positif	Neutre
Négatif	**363**	97	40
Positif	148	**309**	43
Neutre	140	107	**253**

K=9

Instances classées correctement : **61,06%**

Instances mal classées : 38.93%

Documents\Calculés	Négatif	Positif	Neutre

Négatif	**414**	68	18
Positif	180	**296**	24
Neutre	197	97	**206**

NaiveBayes/Bayes

Instances classées correctement : **84%**

Instances mal classées : 16%

Documents\Calculés	Négatif	Positif	Neutre
Négatif	**424**	50	26
Positif	42	**442**	16
Neutre	56	50	**394**

Observations

Sélectionner les attributs les plus pertinents a eu un grand impact sur les résultats des algorithmes de fouilles de données sur nos documents sans pour autant altérer notre classement. Bayes, la machine à état de vecteurs et l'arbre à décision sont toujours en tête, on remarquera tout de même un plus grand écart entre les performances entre ces deux derniers.
L'algorithme des plus proches voisins est toujours à la traîne mais avec des résultats satisfaisants cette fois ci.

Analyse des résultats

La lemmatisation a vraiment permis de réduire le bruit ce qui facilite donc la mise en relief des attributs pertinents.
Après la sélection des attributs on remarque une amélioration significative des résultats des algorithmes sur nos documents. Les attributs sélectionnés sont en effet pertinents pour notre classification d'opinion. Pour finir on note que les meilleurs résultats sont obtenus grâce à l'algorithme de la machine à état de vecteurs.