Rapport

Extraction de connaissances avancée

CARBONNEL Jessie, Nguyen Daniel & Pibre Lionel

Jeudi 11 décembre 2014

1 Présentation

Ce projet s'inscrit dans le cadre de l'UE Extraction de connaissances avancée et a pour but d'appliquer les différentes techniques de fouille et d'analyse de données vues en cours.

Contraintes : L'analyse doit s'effectuer sur des données textuelles hétérogènes et doit porter sur des sentiments ou des opinions.

2 Outils utilisés

Voici la liste des différents outils utilisés lors de ce projet :
- Hadoop
- D3.js
- TreeTagger
- API Weka
- Langage de programmation : Java

TODO : Expliquer rapidement à quoi sert chaque outils et à quoi ils vont nous servir.

3 Constitution du corpus

Nous avons décidé de travailler sur un corpus basé sur des commentaires d'applications android.

Nous avons choisit de les extraire depuis les applications disponnibles sur le Google Play Store (*https ://play.google.com/store/apps*) : les applications y sont nombreuses, variées et le site possède une communauté importante. Les informations relatives à un commentaire sont les suivantes : le contenu, le titre, la note de l'application et la date. Nous pouvons aussi utiliser des informations relatives à l'application pour les mettre en parallèle avec les commentaires : la note globale, le nombre de votants, la catégorie ... On peut trouver diverses API qui permettent de récupérer les informations relatives aux applications de Google Store. Nous avons choisi une API non officielle, *Android Market API*, qui est rapide à prendre en main et permet d'extraire facilement les commentaires.

Chaque API possède un identifiant unique. Nous avons donc dans un premier temps récupérer les identifiants de 135 applications dans des sections variées. Puis nous avons récupéré les informations des 40 derniers commentaires de chacune de ces applications.

Chaque commentaire va faire l'objet d'un fichier. Voici l'exemple des informations relatives au premier commentaire d'une application permettant de lire des PDF (*books.ebook.pdf.reader#1.txt*) :

```
NomApplication:Ebook et PDF Reader
IdApplication:books.ebook.pdf.reader
CategorieApplication:Livres et références
NoteApplication:4,3
NombreVotants:43 379
TitreCommentaire:Ebook Pelerin
Commentaire: Super installation, ai acheté un ebook chez Bayard. Suis pas déçu.
DateCommentaire:26 juillet 2014
NoteCommentaire:5
```

Nous avons extrait un total de 5346 commentaires sur ce format.

4 Préparation des données

4.1 TreeTagger

Afin de réaliser le prétraitement sur notre corpus, nous avons utilisé TreeTagger. Ce logiciel nous permet de connaître la classe grammaticale des mots et d'obtenir la forme lemmatisée de ces derniers. Cet outils nous a permis de nettoyer le corpus en supprimant par exemple les pronoms et les déterminants.

La figure 1 présente un exemple de TreeTagger.

Mot	Classe grammaticale	Mot lemmatisé
dès	PRP	dès
que	KON	que
je	PRO :PER	je
lance	VER :pres	lancer
l'	DET :ART	le
application	NOM	application
ça	PRO :DEM	cela
marche	NOM	marche
pas	ADV	pas
sinon	KON	sinon
j'	PRO :PER	je
adore	VER :pres	adorer
cyprien	ADJ	cyprien
...	...	...

FIGURE 1 – Exemple de TreeTagger

4.2 Génération des fichiers ARFF avec un parser Java

Nous utilisons l'API java.nio, développée pour JDK7, qui facilite la manipulation d'entrées/sorties de système de fichiers et qui apporte une importance à la scalabilité.

En entrée les données étiquetées par TreeTagger, en sortie un fichier de format ARFF. Un fichier ARFF est un simple fichier texte structuré de façon à être lisible par le programme Weka.

On récupère la ligne *NoteCommentaire*, et le texte entre *TitreCommentaire* et *DateCommentaire*. La valeur associée à *NoteCommentaire* est entre 1 et 5, représentée sous forme de nombre d'étoiles dans le Play Store, et va nous permettre de catégoriser le commentaire dans une des trois classes d'opinion :

- [1,2] = mauvais
- 3 = neutre
- [4,5] = bien

Une fois le commentaire catégorisé dans une des classes, les mots contenu dans ce commentaire seront utilisés afin de créer une caractéristique ou bien une règle à la classe pour, ensuite, bien classer un nouveau commentaire entrant. Cette étape sera détaillée par la suite dans la section 6.

Pour le commentaire etiqueté dans la figure 1, avec une valeur de *NoteCommentaire* à 3, nous aurons la ligne suivante :

```
neutre, 'bug lancement lancer application marche help adorer cyprien ...'
```

Notre fichier ARFF contiendra 5346 lignes dans ce format et sera utilisé pour effectuer une première analyse visualisable, mais aussi pour mettre en pratique les algorithmes de fouilles de données.

Nous avons décidé de créer huit fichiers ARFF différents afin de voir l'impact qu'a la composition du corpus sur les algorithmes de classification.

- Le texte brut (sans aucune modification)
- Le texte brut en utilisant la forme lemmatisée des mots
- Le texte uniquement composé d'adjectifs, de noms et de verbes
- Le texte uniquement composé d'adjectifs, de noms et de verbes en utilisant la forme lemmatisée des mots

- Ensuite, nous avons repris ces quatre fichiers ARFF mais en corrigeant l'orthographe des verbes aimer et adorer, nous avons aussi remplacer “kiffer” par aimer.

5 Analyse et visualisation des données

5.1 Statistiques sur les données préparées

Nous allons exécuter deux processus

Compter la part de chaque classe avec une méthode qui compte le nombre total de commentaire classé "bien", "neutre" et "mauvais" : on calcule le pourcentage que représente chaque classe on peut observer une majorité absolue pour la classe "bien" (3279 commentaires positifs pour 5346, soit 61%).

Lister les mots les plus fréquents de chaque classe en utilisant le framework Hadoop MapReduce que nous avons au préalable mis en place dans une autre UE pour exécuter cette même tâche : nous avons un programme qui va se charger de lire, dans le fichier ARFF précédent, les lignes qui commencent par "bien", "neutre" et "mauvais" puis

de compter l'occurrence de chaque mot et les trier par ordre décroissant dans chacune des classes.

5.2 Résultats obtenus avec Hadoop

Nous avons réussi à extraire la liste des 5 mots les plus utilisés par classe dans les commentaires.

Ils sont présentés dans la figure 2.

Classe	Mot	Valeur
bien	cool	578
bien	super	481
bien	bon	455
bien	adorer	25
bien	merci	219
neutre	jour	96
neutre	bon	72
neutre	mise	61
neutre	mettre	58
neutre	dommage	54
mauvais	jour	-271
mauvais	nul	-242
mauvais	impossible	-189
mauvais	fonctionner	-182
mauvais	mettre	-174

FIGURE 2 – Top 5 des mots les plus utilisés dans chaque classe

5.3 Visualisation avec D3.js

Nous allons visualiser les statistiques de la section 5.1 en utilisant la librairie D3.js pour produire des formes à partir des données récupérées.

Une première visualisation en camembert pour représenter le pourcentage des commentaires appartenant à chaque classe : *bien*, *mauvais* ou *neutre*. Ces résultats sont représentés dans la figure 3.

Une seconde en histogramme pour mettre en évidence les mots les plus présents dans chaque classe, et donc voir les mots les plus caractéristiques d'une opinion dans notre corpus. Ces résultats sont représentés dans la figure 4.

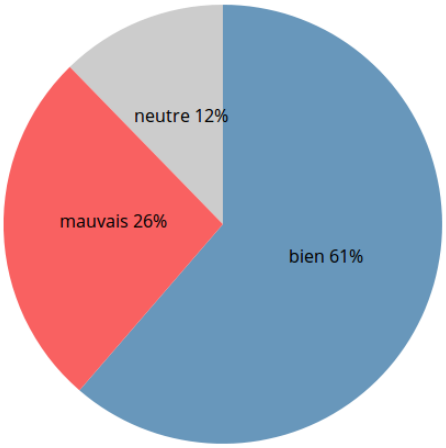

FIGURE 3 – Pourcentage des commentaires en fonction de leur classe

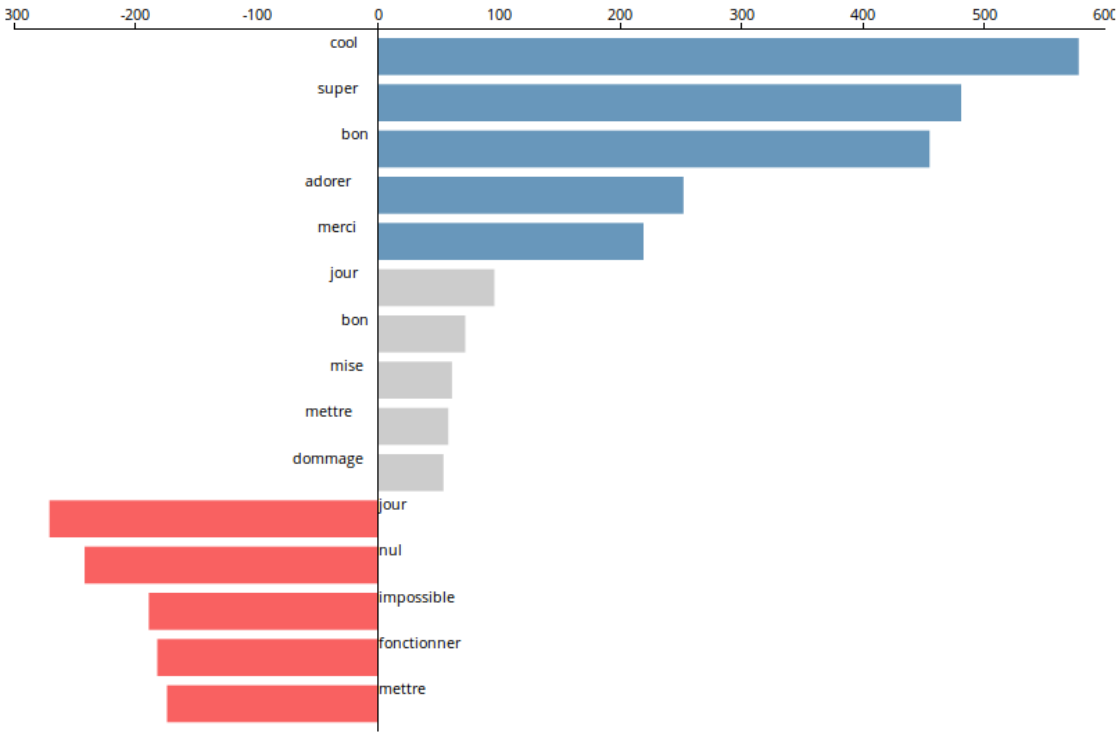

FIGURE 4 – Top 5 des mots les plus utilisés dans chaque classe et leur score

6 Fouille de données

6.1 Application d'algorithmes avec Weka

Voici les différents algorithmes utilisés dans la partie fouille de données :

NaiveBayes : la classification naïve bayésienne est un type de classification probabiliste simple basée sur le théorème de Bayes avec une forte indépendance d es hypothèses.

Arbre de décision : cet algorithme utilise une structure d'arbre. L'extrémité de chaque branche représente les différents résultats possibles en fonction des décisions prises à chaque étape. Cet algorithme répartit une population d'individus en groupes homogènes, selon un ensemble de variables discriminantes en fonction d'un objectif fixé et connu.

SVM : Les SVM sont des classificateurs qui reposent sur deux idées clés :

- La notion de marge maximale. La marge est la distance entre la frontière de séparation et les échantillons les plus proches. Ces derniers sont appelés vecteurs supports. Dans les SVM, la frontière de séparation est choisie comme celle qui maximise la marge. Le problème est de trouver la frontière séparatrice optimale, à partir d'un ensemble d'apprentissage. Cependant il existe déjà des algorithmes pour résoudre ce problème
- Transformer l'espace de représentation des données d'entrées en un espace de plus grande dimension, dans lequel il est probable qu'il existe une séparatrice linéaire.

Méthode par règles d'association : La méthode par règles d'association s'apparente à celle des arbres de décision. Chaque règle pouvant être vue comme le parcours dans un arbre de décision binaire. Cependant les arbres de décisions essayent de réduire le nombre de parcours à explorer tandis qu'ici on explore tout.

K plus proche voisin : L'algorithme KNN figure parmi les plus simples algorithmes d'apprentissage artificiel. Dans un contexte de classification d'une nouvelle observation x, l'idée fondatrice simple est de faire voter les plus proches voisins de cette observation. La classe de x est déterminée en fonction de la classe majoritaire parmi les k plus proches voisins de l'observation x. La méthode KNN est donc une méthode à base de voisinage, non-paramétrique ; Ceci signifiant que l'algorithme permet de faire une classification sans faire d'hypothèse sur la fonction $y = f(x_1, x_2, \ldots x_p)$ qui relie la variable dépendante aux variables indépend antes.

6.2 Recherche des mots les plus fréquents par classe

TODO : Partie Daniel sur Hadoop

7 Résultats

TODO : A quoi correspondent les données dans les tableaux des figures 2 à 9 ?

7.1 Résultats obtenus avec Weka

Les tableaux suivant représentent les résultats des instances correctement classifiées (en pourcentage). Pour obtenir ces résultats nous avons utilisé une représentation de type "sac de mots". Nous avons donc utilisé le filtre de Weka "StringToWordVector", sans compter le nombre de fois que le mot apparaît la première fois (les résultats de la colonne binaire), et une autre fois en comptant le nombre de fois que le mot apparaît (la colonne fréquence).

7.1.1 Texte non corrigé

	Binaire	Fréquence
NaiveBayes	64.1788	63.6551
SMO	73.1201	73.6438
IBk	65.563	64.3659
J48	67.8451	67.8638
JRip	64.7961	65.2637

FIGURE 5 – Texte brut

	Binaire	Fréquence
NaiveBayes	65.8997	65.376
SMO	73.6438	74.3172
IBk	65.9371	64.4407
J48	68.9113	68.7991
JRip	67.7703	68.3315

FIGURE 6 – Texte brut lemmatisé

	Binaire	Fréquence
NaiveBayes	66.517	65.4321
SMO	72.5776	72.5963
IBk	64.5342	64.1227
J48	69.2854	69.0797
JRip	66.33	65.9559

FIGURE 7 – Texte nettoyé

	Binaire	Fréquence
NaiveBayes	67.9012	66.4796
SMO	72.8021	73.2697
IBk	65.9746	62.1212
J48	69.0423	68.9862
JRip	67.5832	67.5645

FIGURE 8 – Texte nettoyé lemmatisé

7.1.2 Texte corrigé

	Binaire	Fréquence
NaiveBayes	64.2724	63.6177
SMO	73.1949	73.569
IBk	65.6004	64.3659
J48	67.8077	67.8451
JRip	65.2263	65.7127

FIGURE 9 – Texte brut

	Binaire	Fréquence
NaiveBayes	66.0307	65.5256
SMO	73.4755	74.6914
IBk	66.0307	64.3285
J48	69.5099	69.2667
JRip	67.4336	67.3588

FIGURE 10 – Texte brut lemmatisé

	Binaire	Fréquence
NaiveBayes	66.835	65.4134
SMO	72.1848	72.4654
IBk	64.7774	64.7026
J48	69.3229	69.3042
JRip	66.3113	65.9746

FIGURE 11 – Texte nettoyé

	Binaire	Fréquence
NaiveBayes	67.8077	66.7228
SMO	73.6064	73.251
IBk	64.4407	63.3184
J48	69.2854	69.5099
JRip	67.7142	67.3588

FIGURE 12 – Texte nettoyé lemmatisé

8 Analyse des résultats

8.1 Analyse des résultats de Weka

Comment expliquer que les résultats obtenus soient aussi bas ?

Lorsque nous avons fait le prétraitement, nous avons vu que deux éléments risquaient de limiter nos résultats.

Les fautes d'orthographe et de frappe : "Je kiff grave car jadore cyprien et jaimefai lui poser un question : est ce que tu connais squeezie (ca c oui c sur) norman kihouu tal blackm ...."

Dans ce cas, peu de mots sont reconnus par TreeTagger. Par exemple les mots "kiff" et "jadore" ne sont pas reconnus alors qu'ils déterminent l'opinion de l'utilisateur.

Les commentaires en anglais : Certains commentaires sont écrits en anglais et ne sont donc pas reconnus par TreeTagger.

On peut voir d'après les résultats obtenus que l'algorithme de classification le plus efficace est SMO. Cet algorithme est très robuste au bruit et comme on a pu le constater au vu des résultats, aux fautes d'orthographe.

Les algorithmes JRip et J48 ne sont pas efficaces car ils se basent sur les mots pour créer des règles afin de déterminer la classe d'un commentaire. En effet, les fautes d'orthographe empêchent de faire des règles précises et efficaces, car un même mot peut être écrit de plusieures manières différentes dans ces commentaires. On peut aussi voir que le fait que les mots soient représentés par la présense ou l'occurence importe peu, car les résultats diffèrent très peu. Étant donné que notre corpus contient un nombre incalculable de fautes d'orthographe, la représentation des données n'impacte pas les résultats.

NaiveBayes est un algorithme de classification probabiliste, encore une fois les fautes d'orthographe jouent un rôle important dans les résultats de cet algorithme. En effet, les fautes l'empêchent de créer des caractéristiques spécifiques aux différentes classes, puisque les mêmes mots sont écrits de différentes façon dans le corpus.

Pour l'algorithme IBk, le fait que des mots apparaissent dans des classes différentes (par exemple le mot "dommage", qui apparaît dans toutes les classes), empêche d'obtenir de meilleurs résultats. De plus on peut remarquer que cet algorithme est plus efficace

lorsque la représentation des mots est binaires, cela vient du fait que les mots vides ont moins d'importance dans ce cas là.

8.2 Analyse des mots les plus fréquents

L'extraction du top 5 des mots les plus utilisés dans chaque classe semble cohérente.

Le mot "*bon*" se retrouve dans la classe bien et neutre, mais pas dans la classe mauvais.

Les personnes satisfaites ont tendance à exprimer leur satisfaction avec des adjectifs ou des verbes positifs (*"cool", "super", "bon", "adorer"*) ainsi que leur gratitude ("*merci*").

Celles qui ne sont pas satisfaites n'expriment pas seulement leur désaprobation avec des adjectifs négatifs ("*nul*"). Ils ont aussi tendance à expliquer ce qui ne leur convient pas et les problèmes rencontrés avec le fonctionnement de l'application ("*fonctionner*" et "*impossible*"). Les mots "*mettre*" et "*jour*" apparaissent tous deux dans les classes neutre et mauvais. On peut faire l'hypothèse que les individus qui ne laissent pas de bons commentaires tendent à proposer ou demander des "mises à jours" pour l'application. Ce qui n'est pas le cas des personnes qui en sont satisfaites.

D'après ces résultats, il semble que si les developpeurs d'une application ont besoins de retours utiles sur le fonctionnement de l'application, ils devraient regarder les commentaires les plus négatifs en premier.

9 Conclusion et perspectives

9.1 Conclusion

Nous avons travaillé sur un corpus de plus de 5000 commentaires portant sur un ensemble d'applications variées.

TODO : Ce qu'a fait Lionel + CCL Lionel et Daniel

Nous avons extrait pour chaque classe de commentaire (bon, neutre ou mauvais) les cinq mots les plus fréquemments utilisés. Nous avons émit des hypothèses sur l'importance de la dimension critique des commentaires rédigés par des utilisateurs qui ont une mauvaise opinion de l'application par rapport à ceux rédigés par des utilisateurs satisfaits.

Nous avons utilisé des techniques de visualisation pour montrer la proportion des trois classes de commentaires ainsi que la fréquence des mots les plus utilisés.

9.2 Perspectives

Utiliser des outils de TALN afin de corriger le corpus ainsi que traduire les commentaires en anglais. En effet, on peut remarquer dans les résultats que juste le fait de corriger l'orthographe de trois verbes, les algorithmes obtiennent pour la majorité d'entre eux de meilleurs résultats.

Tester d'autres algorithmes afin de voir s'il n'y a pas un ou plusieurs algorithmes plus performant que ceux testés au cours de ce projet.