PJE:Analyse de Comportements avec Twitter

Alexis Pernet

Table des matières

1 Introduction

1.1 Problématique du projet

L'objectif de ce projet est de réaliser une analyse de comportements sur des tweets. C'est à dire qu'on veut dégager une tendance, soit positive, négative ou neutre.

Pour ce faire on va utiliser diverses méthodes permettant de classer automatiquement des tweets à l'aide d'une base de tweets qui aura été notée manuellement.

On pourra également voir si ces méthodes sont efficaces ou non, comparer les résultats qu'on obtient, et essayer de détermine une façon plus efficace de faire cette analyse de comportements.

1.2 Architecture de l'application

Le dépot git pour l'application est situé à l'adresse https://github.com/levanh/twitter.git

L'application est donc programmée en java, et contenue dans un fichier .jar, accompagnée de fichier CSV pour les bases d'apprentissage, et d'un fichier twitter4j.properties contenant la configuration pour les recherches sur l'API Twitter.

On a un package interfaceBuilder contenant les éléments de l'interface graphique. Celui-ci contient les sous-packages table contenant les classes pour le tableau contenant les tweets, listener contenant les ActionListener utilisés par les boutons, et itemListener pour les cases à cocher.

On a des packages keyWord, knn, et bayes, contenant respectivement les classes nécessaires permettant d'éxécuter les algorithmes.

Il y a également un package search contenant la classe utilisant l'API pour la recherche, et un package expAnalyse qui gère l'éxécution de la validation croisée.

Enfin, un package utilty contient diverses classes utilisées en général dans l'application.

2 Travaux réalisés

2.1 Récupération des tweets avec l'API Twitter

Pour la récupération des tweets, on s'est limité à une fonction basique de recherche. On utilise l'API "search" pour récupérer des tweets sur un sujet en fonction des mots-clés choisis. On récupère une page de tweets, en passant via l'api twitter4j pour java.

2.2 Préparation de la base d'apprentissage

2.2.1 Nettoyage des tweets

Les tweets récupérés avec l'API ne sont pas forcément dans un format qui est adéquat pour les analyser. C'est pourquoi avant de les annoter, que ce soit manuellement ou avec utilisation des algorithmes de classification, on va les "nettoyer", c'est à dire modifier le contenu pour ne garder que les données qui soient utiles et exploitables.

L'objectif ici est qu'on puisse facilement identifier la classe des tweets à partir des mots qui les composent. Et si on retrouve certains de ces mots dans une liste/base d'apprentissage etc...

On devra donc comparer des mots pour vérifier s'ils sont identiques, ils faut donc éviter que les petites différences fassent que les mots ne soient pas reconnus.

Au final, on a appliqué les règles suivantes:

- On retire les sauts de lignes et les tabulations, car ils n'apportent pas d'informations et seraient gênants pour l'enregistrement des données.
- On retire toutes les ponctuations et autres symboles, dans notre cas les espaces suffisent à délimiter les mots. De plus la virgule est le délimiteur pour le fichier CSV qu'on utilise.
- On retire complètement les groupes de caractères contenant des chiffres. On n'a ici pas de méthode pour traiter correctement les nombres.
- On change les caractères accentués en leur version sans accent. Les tweets n'étant pas toujours correctement accentués, il sera ainsi plus simple de les comparer sans accents.

2.2.2 Construction de la base

Pour construire la base d'apprentissage, on effectue diverses recherches pour récupérer des tweets, et on les nettoie puis on les note manuellement.

Quand cela est fait, on les enregistre au format CSV grâce à une fonctionnalité de notre programme. On a également une fonctionnalité permettant de les charger ultérieurement.

2.3 Présentation des algorithmes de classification

2.3.1 Classification par mots-clefs

Pour cette classification, on utilise un dictionnaire de mots-clefs. C'est une liste de mots à laquelle on a associé une classe, indiquant si le mot est positif ou négatif.

Ici, ces liste sont contenues dans des fichier CSV différents, qui sont lus quand cette classification est utilisée. (Un fichier positive.txt et un fichier negative.txt)

A partir de cette liste, on classe les nouveaux tweets, en regardant si ils contiennent des mots de cette liste, et on compte le nombre de fois ou on a un mot positif, et le nombre de fois ou on a un

2.3.2 Classification avec KNN

On a donc réalisé la classification des tweets par méthode KNN. On peut entrer un eniter k correpondant aux nombre de voisins qui seront considérés. On peut ensuite lancer l'algorithme avec une liste de tweets à annoter, et une base d'apprentissage.

La méthode consiste à calculer une distance entre le tweet à noter et chacun de la base d'apprentissage, afin d'avoir une mesure pour dire desquels il se rapproche le plus. Ainsi on peut attribuer une classe en fonction de cela.

Pour ce faire on a une fonction assez simple permettant de calculer la distance entre 2 tweets, en utilisant le nombre de mots communs entre les 2. La formule donnant la distance est:

$$\frac{n_t + n_c}{n_t}$$

Où n_t est le nombre total de mots différents présents dans les 2 tweets, et n_c le nombre de mots en commun dans les 2 tweets.

On a également implémenté le calcul des distances avec la distance de Levenstein, qui correspond au nombres de changements à faire pour passer d'un tweet à l'autre.

A l'aide d'une de ces 2 formules, pour chaque tweet à noter:

- On calcule la distance entre ce tweet et chacun des tweets de la base d'apprentissage
- On considère les k tweets de la base d'apprentissage ayant la distance la moins élevée.
- On regarde la classe présente en majorité dans ces tweets, et c'est la classe qu'on donne au tweet que l'on voulait noter.

On répète ensuite ce processus pour chaque tweet de la liste à annoter.

2.3.3 Classification bayesienne

La classification bayesienne permet, à partir de formules probabilistes, d'obtenir la probabilité que le tweet appartienne à une classe, pour chacune des classes. On peut ainsi attribuer la classe en fonction de la probabilité la plus élevée.

Les formules utilisent des probabilités d'apparition pour chaque mot, et pour calculer celle-ci, il faut compter la présence des mots dans la base d'apprentissage.

Ainsi, pour annoter une liste de tweets, on commence par faire ce comptage avant tout et on utilise les résultats pour chaque tweet. On crée 3 tables de hachage, une pour chaque classe. Celles-ci contiendront une association entre un mot et un nombre d'occurrences dans la base pour la classe considérée.

Pour les remplir, on parcours la base d'apprentissage. A chaque mot de chaque tweet, on travaille dans la tables dont la classe correspond. Puis si le mot n'y est pas présent, on met une entrée avec ce mot et l'entier 1. Sinon on incrémente l'entier auquel le mot est associé.

Après avoir fait ce comptage, on note les tweets de note liste un à un. Pour ce faire, on calcule les probabilités de chaque classe à partir de la formule suivante:

$$(\prod_{m \in t} p(m|c)) * p(c)$$

Et on utilise l'approximateur de Laplace pour éviter une probabilité nulle dans le formule, avec:

$$p(m|c) = \frac{n_m + 1}{n_c + n_t}$$

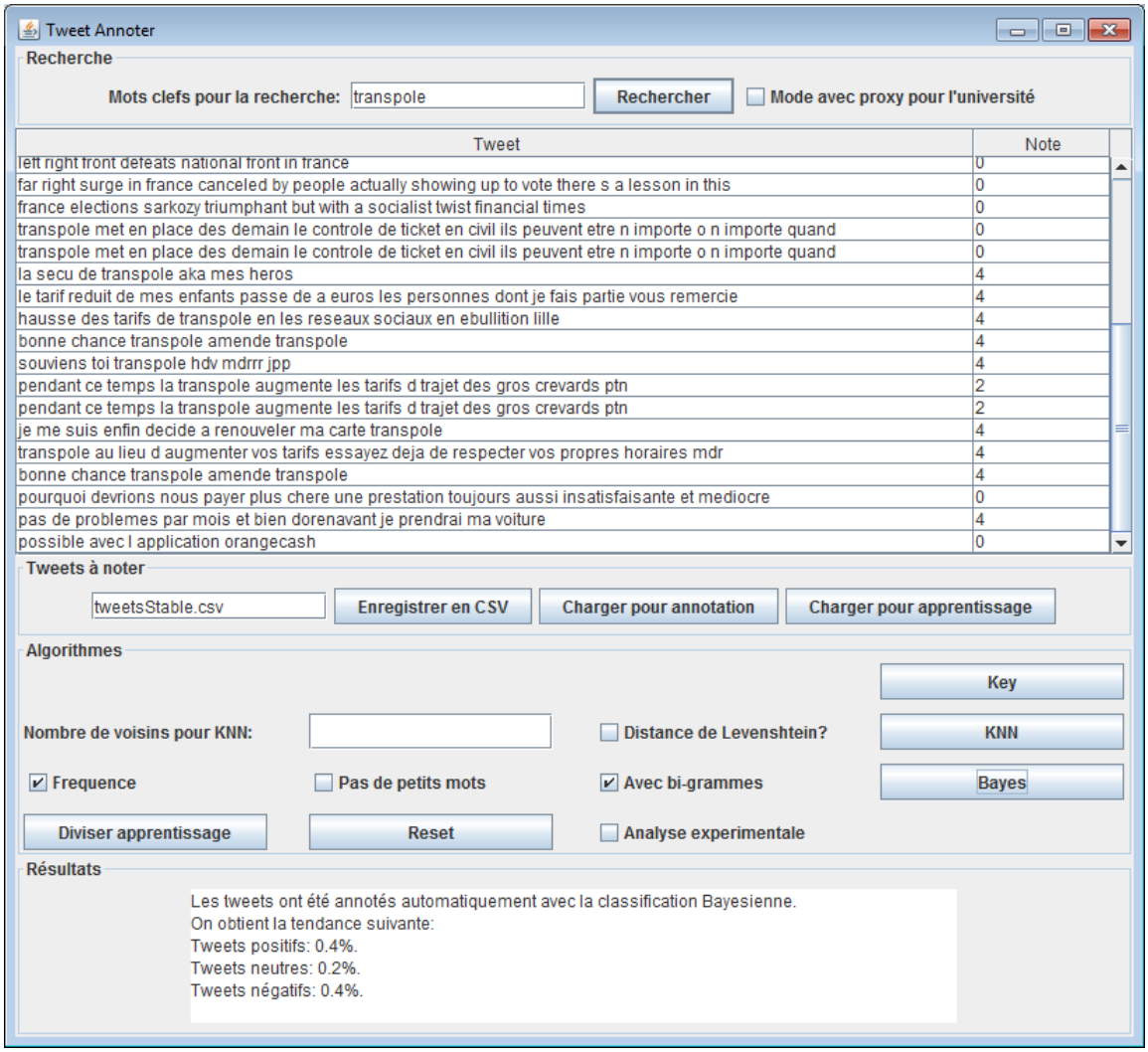
Où n_m est le nombre d'occurence du mot m dans les tweets de classe c, n_c le nombre de mots total dans les tweets de classe c, et n_t le nombre total de mot dans les tweets de la base d'apprentissage.

Avec les résultats qu'on obtient, on choisit la probabilité la plus élevée pour noter le tweet.

On a également réalisé plusieurs variantes de cette classification Bayesienne:

- On a une version par fréquence plutôt que présence, où si un mot apparaît plusieurs fois dans un tweet à noter, alors on le prends autant de fois en compte, en mettant plusieurs fois sa probabilité dans le produit.
- On a une version où les mots plus courts (moins de 3 lettres inclus) ne sont pas pris en compte dans les calculs des probabilités. En effet, la majorité de ces mots courts n'ajoutent généralement pas beaucoup de sens à la phrase, et donc ne sont pas utiles pour décider si le tweet est positif, négatif ou neutre. On pourra voir si cela impacte les résultats.
- Enfin, au lieu de compter les mots un par un, on ajoute la présence des bi-grammes à cette formule, c'est à dire des ensembles de deux mots, qu'on compte de la même façon qu'un unique mot dans la formule.

2.4 Interface graphique



2.4.1 Utilisation

Il y a d'abord une section "recherche", ou on peut taper un mot-clé puis cliquer sur rechercher, pour obtenir de nouveaux tweets avec l'API Twitter. On a une option a cocher, qu'il faut utiliser pour utiliser le programme à l'université. Il rajoute les lignes nécessaires dans le fichier de configuration pour le proxy.

La section suivante contient une liste de tweets à noter qui contient les résultats de la recherche, où de l'ouverture d'un fichier CSV. On peut ici changer manuellement les notes attribuées au tweets dans la colonne "Note".

Ensuite une partie contenant les options pour enregistrer et charger des fichiers CSV. Il faut taper le nom du fichier dans lequel on veut sauvegarder ou à partir duquel on veut charger des tweets. Le bouton "Enregistrer en CSV" sauvegarde les tweets du tableau en format CSV. Le bouton "Charger pour annotation" charge les tweets d'un fichier CSV pour les mettre dans le tableau. Le bouton "Charger pour apprentissage" permet de charger les tweets annotés d'un fichier CSV pour être utilisés comme base d'apprentissage pour l'annotation automatique.

On a le cadre contenant les boutons pour lancer les algorithmes de classification. Le bouton "Mots-clés" pour simplement lancer la classification par mots-clés. Pour KNN on peut rentrer un nombre pour k, ainsi qu'utiliser ou non la distance de Levenshtein en cochant une option. Puis on lance l'algorithme en cliquant sur "KNN". Pour Bayes on a 3 options qu'on peut cocher, pour utiliser la version avec fréquence, retirer les petits mots ou ajouter les bi-grammes. Et on lance l'algorithme en cliquant sur "Bayes".

On a un bouton "Diviser Apprentissage", utile pour réaliser des tests courts. Il prends la base d'apprentissage qui est chargée, la divise et en mets une partie dans le tableau pour être annotée. Le bouton "Reset" permet de vider le tableau et la base d'apprentissage. Enfin l'option "Analyse expérimentale" fait en sorte que, au lieu de noter les tweets du tableau en utilisant la base d'apprentissage, on lance la validation croisée sur la base d'apprentissage, pour obtenir un taux d'erreur moyen.

La zone de texte en bas contient quelques résultats après l'exécution des algorithmes, et des messages après le chargement des fichiers.

3 Analyse expérimentale des résultats

3.1 Protocole suivi

Pour l'analyse expérimentale, on a réalisé la validation croisée sur nos différentes méthodes de classification. On va effectuer cette validation en utilisant des bases d'apprentissage, qu'on va découper en 10, où on va annoter chaque partie en utilisant les 9 autres. On récupérera ainsi le pourcentage d'erreurs obtenu en moyenne. Ici, on a utilisé 2 bases d'apprentissage différentes pour avoir un peu plus de résultats:

- tweetsBetter.csv qui contient environ 125 tweets, et la proportion tends un peu vers les tweets négatifs.
- tweetsStable.csv qui contient un peu plus de 300 tweets, avec exactement autant de tweets dans chaque classe.

3.2 Résultats obtenus

3.2.1 Mots-clés

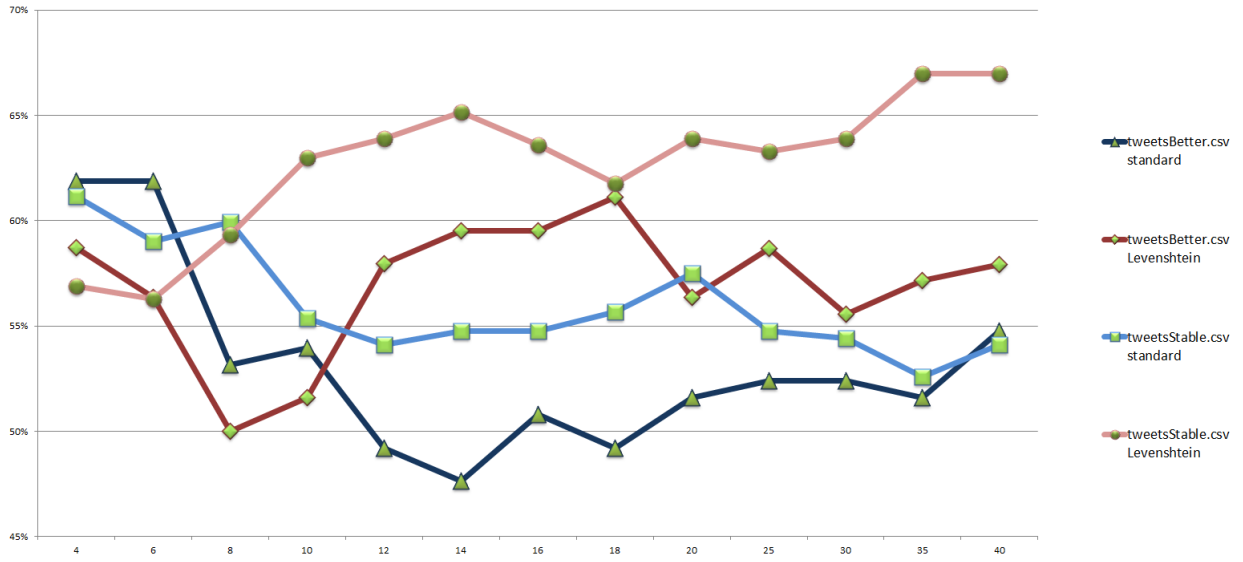
Pour la classification par mots-clefs, on n'a pas de base d'apprentissage contenant des tweets, mais on a quand même suivi le même protocole, en ayant le même dictionnaire de mots utilisés pour noter une liste de tweets. On a donc récupérés le taux d'erreur obtenu pour chacune des bases.

Pour tweetsBetter.csv, on a un taux d'erreur de **55,56%**

Pour tweetsStable.csv, on a un taux d'erreur de **60,86%**

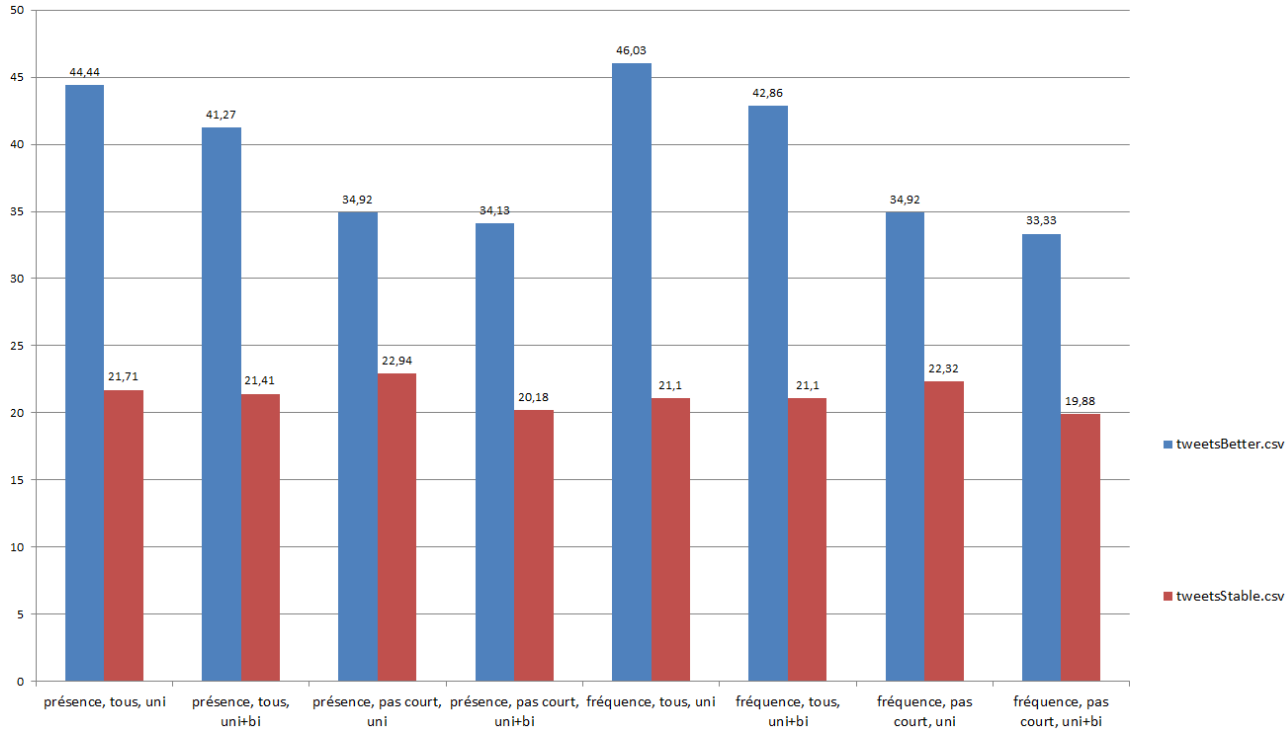
3.2.2 KNN

On a effectué divers tests sur chaque base, en utilisant le calcul des distances avec la fonction simple, puis la distance de Levenshtein. On a également fait varier le nombre de voisins k pour voir comment les résultats étaient influencés. Voici les résultats regroupés au sein d'un petit graphe, où on a le taux d'erreurs en fonction de la valeur de k, pour les 2 bases.



3.2.3 Classification Bayesienne

On a implémenté 3 différents choix, calcul par présence ou fréquence, prise en compte des mots courts ou non, et utilisation de bi-grammes ou non. On va donc effectuer les tests avec les 8 combinaisons possibles, pour chaque base. Les résultats sont regroupés dans ce diagramme en barres, qui indique le taux d'erreur selon la variante utilisée, et selon la base utilisée.



3.3 Interprétation des résultats

On peut déjà comparer les résultats obtenus en moyenne pour chaque méthode. Pour les mots clefs on a 55-60% d'erreurs, pour knn on oscille entre 45 et 65%, avec la majorité proche des 50%. Enfin, pour Bayes on obtient entre 35 et 45% pour une base et autour de 20% pour l'autre.

On voit donc que la classification Bayesienne nous donne des résultats bien meilleurs, malgré que ceux-ci ne sont pas très bons. A l'inverse les résultats qu'on obtient ici pour les mots-clés et knn sont très mauvais, ce qui donne l'impression que l'algorithme est beaucoup moins intéressant.

Cela dit, on pourrait probablement grandement améliorer la performance de l'algorithme des mots-clés en agrandissant les dictionnaires de mots positifs et négatifs, pour qu'ils prennent en compte plus de cas. En effet, la majorité des faux résultats proviennent de tweets notés neutre car aucun mot positif ni négatif n'est trouvé.

Pour knn, dans notre cas, la distance de Levenshtein nous donne des résultats moins bons, ce calcul de distance n'est peut-etre pas approprié pour la comparaison de phrases plus longues, ou alors il faudrait de plus grandes bases d'apprentissage. On voit aussi que la base tweetsBetter.csv donne des résultats légèrement meilleurs, malgré qu'elle soit plus petite, probablement dû au déséquilibre des classes de tweets au sein de celle-ci.

On peut aussi remarquer que faire varier la valeur de k ne change pas beaucoup les résultats obtenues, les valeurs donnant les meilleurs résultats dépendent des tests, même si les résultats semblent bons entre 8 et 16, et assez irréguliers pour des valeurs plus faibles que ça.

Finalement, pour Bayes, la différence des taux d'erreurs obtenues entre les deux bases est importante, et la base tweetsStable.csv, contenant plus de tweet et plus équilibrée donne les meilleurs résultats.

Il n'y a pas beaucoup de différence ou d'amélioration entre fréquence et présence. Par contre, quand on considère les bi-grammes plutôt que juste les uni-grammes, on obtient toujours une amélioration, bien que légère. De la même manière, on obtient des meilleurs résultats en ne prenant pas en compte les mots de moins de 3 lettres, mais cela ce voit plus sur une des bases que sur l'autre.

Pour les 2 bases, le taux d'erreurs le plus faible est obtenue avec la variante par fréquence, sans compter les mots cours, et en comptant unigramme et bi-gramme.

4 Conclusion sur le projet

Ce projet a permis de mettre en oeuvre plusieurs façons de classer automatiquement des données, de les tester sur de nouvelles données, et également de réaliser une analyse expérimentale. On a obtenu plusieurs résultats, qui ont permis d'identifier quelle méthode semble être la meilleure, et d'essayer de comprendre les problèmes de chaque méthode, et de qui pourrait être mise en oeuvre pour l'améliorer.

Il faudrait probablement avoir des bases d'apprentissage plus grandes et mieux construites pour pouvoir pousser cette analyse plus loin, et obtenir des résultats plus fiables.

Personnellement, j'ai pu découvrir ces classifications automatiques que je ne connaissais pas, ainsi que travailler sur le traitement et l'analyse de données, qui est un sujet qui m'intéresse mais sur lequel je n'avais jamais vraiment travaillé.