Rapport PJE

Lucas Sauvage

15 Decembre 2025

Contents

1 Description du projet **2**
1.1 Problématique 2
1.2 Architecture logicielle 2
1.3 Organisation 2

2 Détails des travaux réalisés **3**
2.1 Préparation et nettoyage des données 3

3 Algorithmes de classifications et clustering **3**
3.1 Dictionnaires 3
3.2 KNN 3
3.3 Bayésien 4
3.4 Clustering 5

4 Interface graphique **5**

5 Résultats de la classification avec les différentes méthodes et analyse **7**
5.1 Comparaison 8
5.2 Variables 8

6 Conclusion **10**

1 Description du projet

1.1 Problématique

Ce projet vise à concevoir et implémenter une application capable d'automatiser la classification des sentiments (i.e. positifs, neutres ou négatifs) de base de données de tweets grâce à différents algorithmes de classification, qui seront expliqués plus bas, notamment l'algorithme KNN et l'algorithme de classification Bayésien.

1.2 Architecture logicielle

Les outils de classifications, c'est à dire tous les petits algorithmes nécessaires pour aboutir aux 2 algos de classifications, sont les suivants :

- Tools : Le package "tools" définit tous les algorithmes / fonctions utilisés dans l'application
  - Outils CSV : "Sortir" les tweets de la base de données, préparer la base d'apprentissage et la base d'entrainement.
  - Outils KNN : Calcul de la distance de Jaccard, algorithme pondéré ou non.
  - Outils Bayésien : Entraînement du modèle, cross-validation.
- Graphic : Le package "graphic" définit les pages, grâce à PyQt5.
  - MainWindow : La fenêtre qui s'exécute en premier.
  - DashBoard : La page de présentation.
  - HandleDatabase : Permet de nettoyer, auto-évaluer et tester les algorithmes sur des bases de données.
  - KNN Algorithm : Permet d'utiliser l'algorithme KNN sur un unique tweet (que l'utilisateur peut renseigner).
  - Clusters : Permet de tester les clusters.

Finalement, chaque page de l'application (chaque vue) vient chercher dans les fichiers du package tools. Chaque fichier du package graphic définit les composants graphiques de l'application (pages de la stack, Pop-Up et messages d'erreur).

1.3 Organisation

Comme je ne suis pas en binôme, j'ai développé tout seul l'application. J'ai toujours commencé par implémenter la partie back-end à chaque nouvel algorithme ou outils, avant de finir par le front-end.

2 Détails des travaux réalisés

2.1 Préparation et nettoyage des données

Afin de pouvoir utiliser les algorithmes de classification, il est important d'uniformiser les données. Chaque base de données peut être différente, même lorsque les attributs (les noms de colonnes) ne sont pas organisés pareil, voir même différents.

La première difficulté était donc d'identifier la colonne du tweet et la colonne de la note. Après avoir testé plusieurs fonctionnalités, notamment prendre le premier élément de plus de 20 lettres, il se trouve que l'algorithme était très fragile et dépendait beaucoup de la première entrée de la base de donnée. Pour pallier ce problème, j'ai choisi de reconnaître les mêmes patterns (i.e. un pattern de plus de 20 caractères alphabétiques et un minimum de 3 mots) mais de faire une moyenne sur les 50 premières entrées de la base de donnée.

Pour préparer la base d'apprentissage (pour les tests de performances), j'ai découpé une base de donnée en 2/3 réservés à l'apprentissage et 1/3 pour le test. J'ai également mélangé chaque base nouvellement créé, ce qui permet de tester plusieurs fois l'algorithme KNN avec différents voisinages tout en gardant la même base de donnée.

3 Algorithmes de classifications et clustering

3.1 Dictionnaires

Avant de passer à de réels algorithmes, il était demandé de faire une implémentation naïve, qui comptait seulement le nombre de mots négatifs, neutres ou positifs d'un tweet en fonction d'un dictionnaire préalablement déclaré contenant des mots de la classe désignée (e.g. le dictionnaire neg ne contient que des mots négatifs).

3.2 KNN

L'algorithme KNN est un algorithme de classification qui s'appuie sur 2 principes : La distance de Jaccard et la comparaison avec un voisinnage. A chaque tour de boucle, KNN va analyser le tweet fournit et calculer sa distance avec le tweet actuel (la boucle se fait sur chaque tweet de la base d'apprentissage) présents dans le voisinage. Celui qui a la distance la plus éloignée avec le tweet sort du voisinage, et est remplacé par le nouveau tweet (celui de la boucle). Finalement, on calcule la note moyenne du voisinage à la fin de la boucle, et le score est renvoyé.

Pour cette algorithme, il peut être judicieux d'essayer de trouver un nombre de tweets dans le voisinage, assez élevé pour ne pas avoir de faux-résultats, mais pas trop élevé pour éviter le bruit et ralentir l'exécution de l'algorithme. Mon application possède un outil qui permet de faire varier le nombre de tweets dans le voisinage, appelons le k. Il est également possible de changer la base de données utilisée comme base pour l'algorithme KNN. Après plusieurs tests,

$k = 11$ semble être un bon nombre de voisins. (Voir section "Résultats", le k optimum varie en fonction de la base de donnée analysée.)

Cet algorithme est également implémenté en version pondérée. La version pondérée prend en compte la distance avec le tweet. C'est-à-dire qu'un tweet très peu distant du tweet fourni aura plus de poids qu'un tweet plus distant.

3.3 Bayésien

La classification Bayésienne est une classification **supervisée** qui se base sur le théorème de Bayes. Contrairement au KNN qui s'appuie sur la distance, l'algorithme Bayésien estime la probabilité qu'un tweet t appartienne à une classe c (i.e. "positif", "neutre" ou "négatif"). On cherche donc à maximiser :

$$c(t) = \arg\max_{c \in C} P(c|t)$$

En décomposant cette probabilité le calcul revient à estimer :

$$P(c|t) \propto P(t|c) \cdot P(c)$$

Avec :

- $P(c)$: Proportion de tweets de la classe c dans l'apprentissage
- $P(t|c)$: Vraisemblance du tweet sachant la classe.

Pour estimer $P(t|c)$ on utilise :

$$P(m|c) = \frac{n(m,c)+1}{n(c)+N}$$

Avec :

- $n(m,c)$: Nombre d'occurrences du mot m dans la classe c.
- $n(c)$: Nombre total de mots dans la classe c.
- N : Nombre total de mots des tweets de l'ensemble d'apprentissage.

De plus, cet algorithme possède des variantes, qui sont plus où moins efficaces en fonction de la taille du jeu de donnée, son uniformité ...

Dans sa version **Presence**, les attributs sont des mot du vocabulaire, et sont tous associés à une valeur booléenne, en fonction de leur présence dans le vocabulaire. L'ordre et le nombre d'occurrences ne sont pas pris en compte. La probabilité pour une classe donnée est le produit des probabilités des mots. Il faut multiplier la probabilité du mot une unique fois, même s'il est plusieurs fois présent dans le vocabulaire. La formule utilisée est la suivante :

$$P(t|c) = \prod_{m \in t} P(m|c)$$

La représentation par fréquence prend en compte le nombre d'occurrences n_m de chaque mot dans le tweet, ce qui ajoute un poids à chaque mot du vocabulaire. Plus il est répété plus il a du poids. La différence avec la formule juste au dessus, c'est qu'il faut multiplier $P(m|c)$ par son poids soit :

$$P(t|c) = \prod_{m \in t} P(m|c)^{n_m}$$

Il existe également différentes manière de sélectionner les mots du vocabulaire. Un premier filtrage est fait pour supprimer tous les mots de moins de 3 lettres. En effet, ces mots sont souvent des articles ou des pronoms, qui ne sont donc pas utiles pour notre algorithme.

Ensuite, il reste 2 modes à utiliser (3 si l'on compte la fusion des 2), qui définissent la taille des attributs

- Uni-grammes : Les mots sont considérés un par un.
- Bi-grammes : Les mots sont considérés par paires
- Fusion : Utilise les mots seuls ainsi que les paires.

3.4 Clustering

Le clustering, à la différence de KNN, est un algorithme de classification **non-supervisée**. L'objectif de cet algorithme est de regrouper les tweets en cluster en fonction de leur distance. Cet algorithme ne vient pas définir la note du tweet, mais regroupe juste les tweets qui ont une distance faible, i.e. par "thème".

4 Interface graphique

Lors du démarrage de l'application, la page "DashBoard" est chargée. Elle permet d'afficher le titre de l'application, mon nom, et un paragraphe d'introduction.

Figure 1: DashBoard

Il y a ensuite 2 pages utilisée pour exécuter les algorithmes.

La première permet de classer une base de données entière.

De plus, elle permet également de tester les 2 algorithmes en faisant varier les variantes

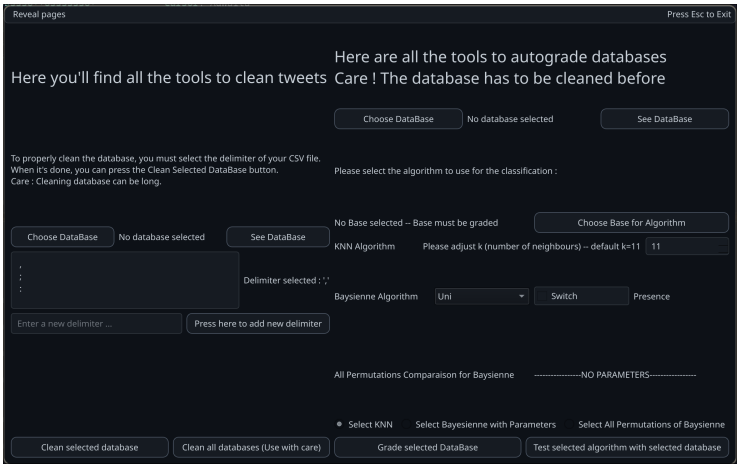

Figure 2: Handle Database

Elle permet également de nettoyer les fichiers CSV. Un fichier CSV nettoyé est un fichier avec au maximum 2 colonnes (une pour la note et une pour le tweet en lui même). De plus, les tweets sont vidés de dates, mentions (@), les "RT" etc. afin de n'avoir que des mots utilisables dans l'application.

La seconde permet d'utiliser l'algorithme KNN sur un unique tweet, toujours en faisant varier les paramètres.

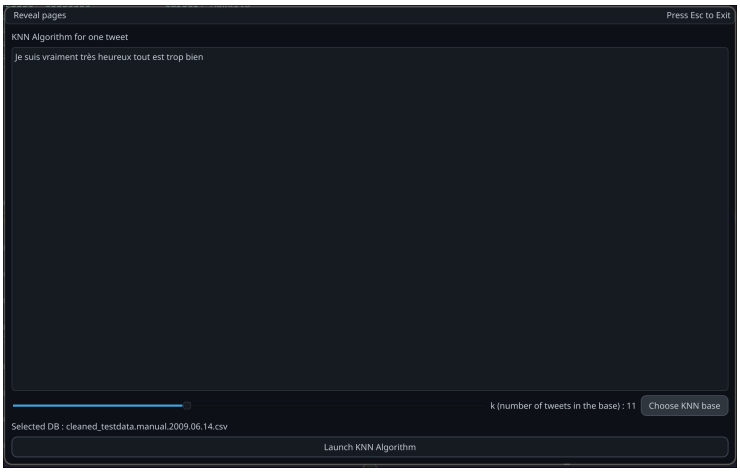

Figure 3: KNN Algorithm

Le résultat est renvoyé immédiatement :

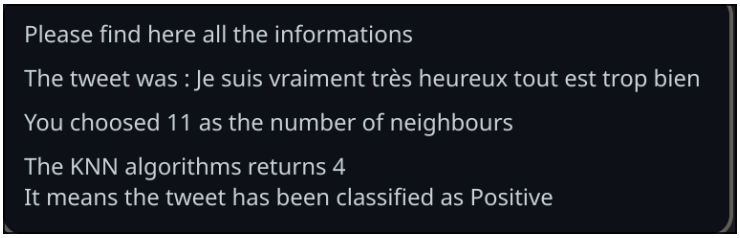

Figure 4: KNN Algorithm Result

La page utilisée pour les clusters étant très sommaire (3 boutons, juste pour switch entre IDF et Ward, sélectionner la base de données et lancer la création de clusters), je n'inclus pas de capture d'écran.

5 Résultats de la classification avec les différentes méthodes et analyse

Afin d'évaluer la précision des algorithmes, il fallait tester les dits algorithmes. Au préalable, la base de données analysée va se décomposer en 2 sous-listes. 2/3 de la base de donnée sera utilisé comme base (de voisinage pour KNN ou comme modèle pour Bayès). Cette évaluation passe par la construction d'une matrice de confusion. Pour chaque classe (0 = "Neutre", 2 = "Négatif" et 4 = "Positif"), les prédictions correctes sont comptabilisées, mais on garde également les erreurs, selon leur classement (par exemple, un tweet positif prédit comme neutre donne "but_neu", négatif donne "but_neg". Il est ensuite facile de calculer

des pourcentages de réussite et de sortir des informations, notamment grâce à des Heatmap :

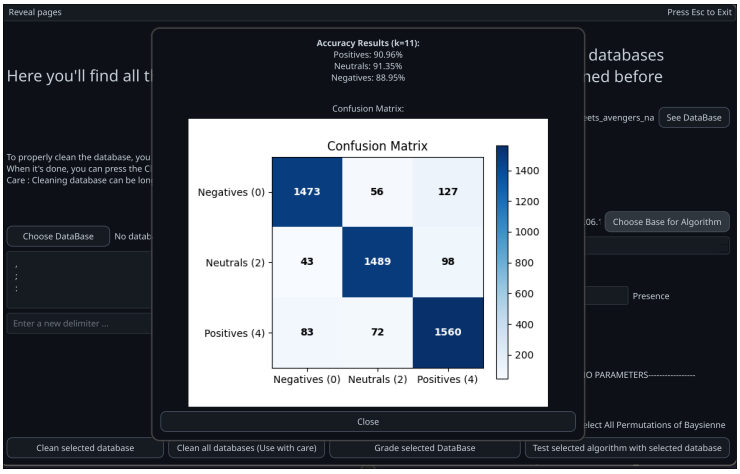

Figure 5: Heatmap

Les différents paramètres et leurs variations ayant été expliqués plus haut, le bilan peut être fait.

5.1 Comparaison

Le vainqueur ici est clairement l'algorithme Bayésien. En effet, même sur des grosses bases de donnée l'algorithme reste relativement rapide. A contrario, l'algorithme KNN montre ses limites très vite. En plus d'être lent (plusieurs minutes si l'on dépasse les 10 000 tweets, beaucoup trop pour le million). De plus, l'algorithme KNN compare à chaque fois le tweet analysé avec toute la base, alors que Bayésien entraîne un modèle et ensuite prédit quasi instantanément. De plus, probablement dû à une faute d'optimisation de ma part, mais l'algorithme KNN fait planter l'application (tout en continuant de classifier) et repart quand la classification est terminée. L'algorithme KNN est donc meilleur pour de l'analyse au cas par cas, là où Bayésien doit entraîner le modèle pour chaque analyse unique. Pour des bases de données entières, on préférera Bayésien.

5.2 Variables

Pour l'algorithme KNN j'ai testé différentes valeurs pour k. J'ai créé et exécuté un algorithme qui, en fonction des matrices de confusion générées pour chaque n, affiche la précision. Voici le graphique :

Figure 6: Variation of k

On voit que pour ces bases de données, le meilleur k est 19.

Pour l'algorithme Bayésien, il y a 2 variables : Le mode (uni, bi ou les 2) ainsi que l'association ou non des mots par paire. Le graphique de chaque combinaisons est le suivant :

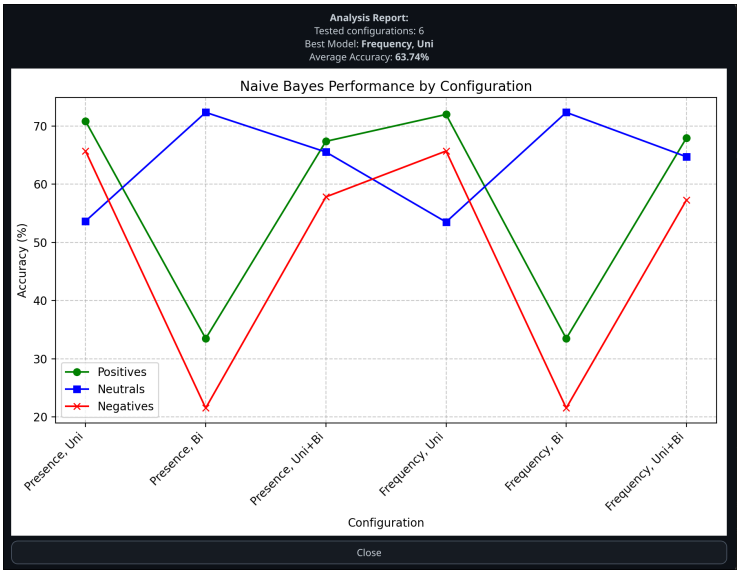

Figure 7: All permutations of Bayès parameters

L'analyse nous donne bien le model idéal pour cette base de données analysée et la base utilisée : Fréquence + Uni. La plupart des analyses retournaient souvent "Bi-Uni + Fréquence".

6 Conclusion

J'aurais aimé créé un outil permettant de comparer réellement l'algorithme KNN avec Bayès, selon un raisonnement mathématique (temps d'exécution, précision, variation selon la taille de l'entrée ...). Sinon tout ce qui a été demandé à été implémenté. Cette application permet la classification, au choix, de tweets uniques ou de base de donnée entière. Il existe aussi des outils pour nettoyer les bases de données **avant** de l'analyser (ou de l'utiliser pour l'analyse).