Rapport PJE : Analyse de sentiments sur Twitter

GERUSSI Nathan
BARRO Amadou

A) Description générale du projet

1) La problématique

La problématique de ce PJE est de classifier des tweets selon le sentiment qu'ils dégagent via des algorithmes supervisés. Tout le travail du PJE réside donc dans la récupération de données, réussir à les nettoyer et les mettre sous une bonne forme pour pouvoir bien travailler dessus puis coder différents algorithmes supervisés tels que des KNN ou des classifieurs Bayésiens et les comparer entre eux pour voir lequel est le meilleur.

2) L'architecture de l'application

Au niveau de la structure nous avons un fichier principal qui va gérer l'interface, gérer l'import et le nettoyage des données et appeler les fonctions de classification puis un dossier à part où se situe tout les différents algos de classification.

Pour l'interface nous avions commencé en utilisant tkinter mais celle-ci était trop vieille et pas assez flexible donc nous sommes passés sur streamlit.

3) L'organisation

Au vu du nombre de choses à faire l'organisation était assez simple car il y avait toujours des choses à effectuer. La seule règle qu'on suivait est qu'on ne se mettait jamais à deux sur une même fonction pour ne pas se marcher dessus. Si l'un travaillait sur l'interface l'autre travaillait sur l'import/nettoyage des données, si l'un codait un des algorithmes l'autre codait son implémentation dans l'interface et la méthode pour l'évaluer et inversement pour un autre algorithme ... . En suivant cette organisation on ne s'est jamais retrouvé sans rien à faire.

B) Détails des différents travaux réalisés

1) Import/Nettoyage des données

Pour cette partie nous avons divisé le travail en 3 fonctions.

La première permet d'uploader le fichier dans son état de base.

La deuxième va venir structurer le dataframe en enlevant les colonnes inutiles et en ne gardant que “target” et “texte” et en associant les valeurs de chaque ligne aux bonnes colonnes.

Enfin la troisième va venir nettoyer chaque tweet en enlevant les caractères inutiles et en re-structurant le tweet pour qu’il soit optimal à une analyse d’algorithme supervisé.

Au niveau des bases de données utilisés, nous avons utilisé celles mis à disposition sur moodle et nous en avons cherché une grande sur kaggle que nous avons réduit qui fait actuellement 20 000 lignes.

Pas d’énormes problèmes rencontrés pour cette partie, seule petite difficulté pour la fonction qui structure le dataframe car les fichiers csv ne sont pas forcément exactement les mêmes et il fallait trouver une méthode générique qui marche pour tous plus bien gérer l’assignation des colonnes aux bonnes valeurs c’était le plus compliqué.

2) Algorithmes de classification

Pour trouver le meilleur classifieur de tweets nous avons du implémenter différentes variantes d’algorithmes supervisés notamment 3 qui sont le dictionnaire, le knn et le Bayes. Dans ces 3 variantes nous avons encore pu faire des sous-variantes et également modifier certains paramètres.

Méthode du dictionnaire

Le premier algrothime supervisé est donc la technique du dictionnaire. Cette technique est relativement simple et basique. Nous avons 2 fichiers de mots, un qui ne contient que des mots positifs et un autre qui ne contient que des mots négatifs. A partir de ça, pour chaque tweet que l’on veut classifier, on regarde combien de mots il contient qui sont dans le fichier de mots positifs puis combien de mots il contient qui sont dans le fichier de mots négatifs. Si le tweet contient plus de mots positifs que de mots négatifs, on le classe positif. Si c’est l’inverse, on le classe négatif, et s'il y a autant de mots positifs que de mots négatifs, on le classe neutre.

Cet algorithme dû à sa simplicité ne possède pas vraiment de variantes ni de paramètres à modifier. La seule chose sur laquelle on pourrait jouer serait le critère selon lequel on classifie le tweet. Par exemple au lieu de dire que le mot est neutre seulement s’il y’a autant de mots positifs que négatifs on pourrait dire qu’il est neutre également s’il n’y a qu’un mot d’écart entre le nombre de mots positifs et le nombre de mots négatifs ce qui fait qu’il serait classé positif seulement s’il y’a au moins 2 mots positifs de plus que de mots négatifs et inversement.

Méthode KNN

Le deuxième algorithme supervisé est celui du KNN appelé aussi la méthode des plus proches voisins. Cette méthode est plus compliqué à mettre en place que le dictionnaire mais le principe est assez simple. La méthode consiste pour

un tweet à classer, de calculer sa distance avec tout les tweets de la base d'apprentissage puis de prendre les k-tweets les plus proches avec le tweet à classer et de classer le tweet selon le sentiment majoritaire des k-tweets les plus proches. Par exemple si k = 3 et qu'on a un tweet T à classer, si dans les 3 tweets les plus proches de T il y'en a deux classés positifs et un classé neutre et bien alors on va classer T en tant que tweet positif.

Ici même si le principe restera le même il existe des variantes du knn en modifiant la façon dont on calcule la distance entre les tweets. Nous avons implémenté 3 différentes variantes pour calculer les distances qui sont les suivantes : la distance dite par-défaut, la distance jaccard et la distance Cosine-Similarity.

La distance par-défaut se calcule en multipliant par 2 le nombre de mots de mots communs puis en divisant ce nombre par le nombre de mots totaux. Par exemple "Il a une pomme" et "Tu aimes la pomme" ont un mot-commun (pomme) et il y a au total 8 mots. Le calcul sera donc (2*1) / 8 = 2 / 8 = 0,25.

La distance jaccard se calcule en faisant le nombre de mots communs diviser par le nombre de mots uniques. Par exemple "Il a une pomme" et "Tu aimes la pomme" ont un mot-commun et il y a au total 7 mots uniques. Le calcul sera donc (1/7) ≈ 0,142.

La distance Cosine-Similarity est une méthode complexe venant de scikit-learn. Dans l'idée cette méthode crée deux vecteurs A et B de fréquence des termes pour chaque phrase qu'on va venir normaliser en divisant chaque composant par la norme euclidienne du vecteur lui-même. Une fois les 2 vecteurs de chaque phrase normalisé, on va calculer la distance en utilisant la formule de la simiralité du cosinus qui est : (A . B) / (|| A || . || B ||) où . représente le produit scalaire et || || représente la norme euclidienne.

Enfin cet algorithme du knn possède un paramètre sur lequel on peut jouer c'est le nombre de voisins les plus proches qu'on utilise pour classifier le tweet. Dans l'exemple d'au-dessus j'avais utilisé k = 3 ce qui fait qu'on prenait les 3 tweets les plus proches pour classifier le tweet mais on peut très bien prendre seulement le voisin le plus proche ou alors les 2 plus proches, les 4 plus proches … .

Méthode Bayes

Le troisième algorithme est celui du Bayes. La méthode Bayes est une méthode probabiliste qui va comparer les différentes probabilités d'un tweet d'être dans une certaine classe. Pour calculer cette probabilité, on multiplie 2 probabilités qui sont : la probabilité de la classe elle-même P(c) et la probabilité du tweet sachant la classe P(t|c).

P(c) est donc le nombre de tweet de classe c divisé par le nombre de tweet total. Par exemple si j'ai une base d'aprentissage de 300 tweets et qu'il y'a 100 tweets positifs, P(positif) = 100/300 = 1/3.

P(t|c) s'écrit aussi comme la multiplication des P(m|c) pour tout les mots appartenant à t. P(m|c) se calcule en comptant le nombre d'occurence du mot

m dans les tweets de classe c divisé par le nombre de mot total des tweets de classe c. Ce calcul amène un inconvénient qui est que si le nombre d'occurence du mot m est 0 cela amène P(t|c) à 0 donc pour contrer cela on utilise l'estimateur de Laplace qui fait que P(m|c) est dorénavant le nombre d'occurence du mot m dans les tweets de classe c + 1 divisé par le nombre total de mot dans les tweets de classe c + le nombre de mot total de tout les tweets réunis.

La probabilité du tweet d'être dans la classe c est donc P(c) * P(t|c). On calcule donc pour chaque tweet P(positif | t), P(neutre | t) et P(négatif | t) et la classe qui à la plus haute probabilité sera associé au tweet t.

Cette méthode offre énormément de variantes différentes dans la manière de sélection des mots ou dans le calcul des probabilités lui-même.

Pour le calcul des probabilités il existe une deuxième variante qui est la réprésentation par fréquence. Celle que j'ai présenté au-dessus est la représentation par présence, la différence que fréquence va mettre est que désormais on rajoute une puissance à chaque P(m|c) correspondant au nombre de fois que le mot est dans le tweet à classifier. Par exemple pour t = "A B A", en présence P(t|c) = P(A|c) * P(B|c) * P(A|c) alors qu'en fréquence P(t|c) = P(A|c)² * P(B|c) * P(A|c)² car A est deux fois dans le tweet.

Pour la sélection des mots voici les 4 combinaisons possibles : Tout les mots + Uni-gramme , Tout les mot + bi-gramme , Mot d'au moins 3 lettres + bi-gramme et Mot d'au moins 3 lettres + bi-gramme.

Tout les mots correspond au fait que l'on prend tout type de mot en compte, mot d'au moins 3 lettres correpond au fait qu'on ne prend en compte que les mots d'au moins 3 lettres, uni-gramme correspond au fait qu'on prend les mots unitairement et bi-gramme correspond au fait qu'on prend les mots 2 par 2. Par exemple si t = "A B C D", en uni-gramme on a donc 1*A ,1*B, 1*C et 1*D tandis qu'en bi-gramme on a 1*AB, 1*BC et 1*CD.

Ces 4 combinaisons combiné au choix présence ou fréquence fait qu'on a 8 types de Bayes différents.

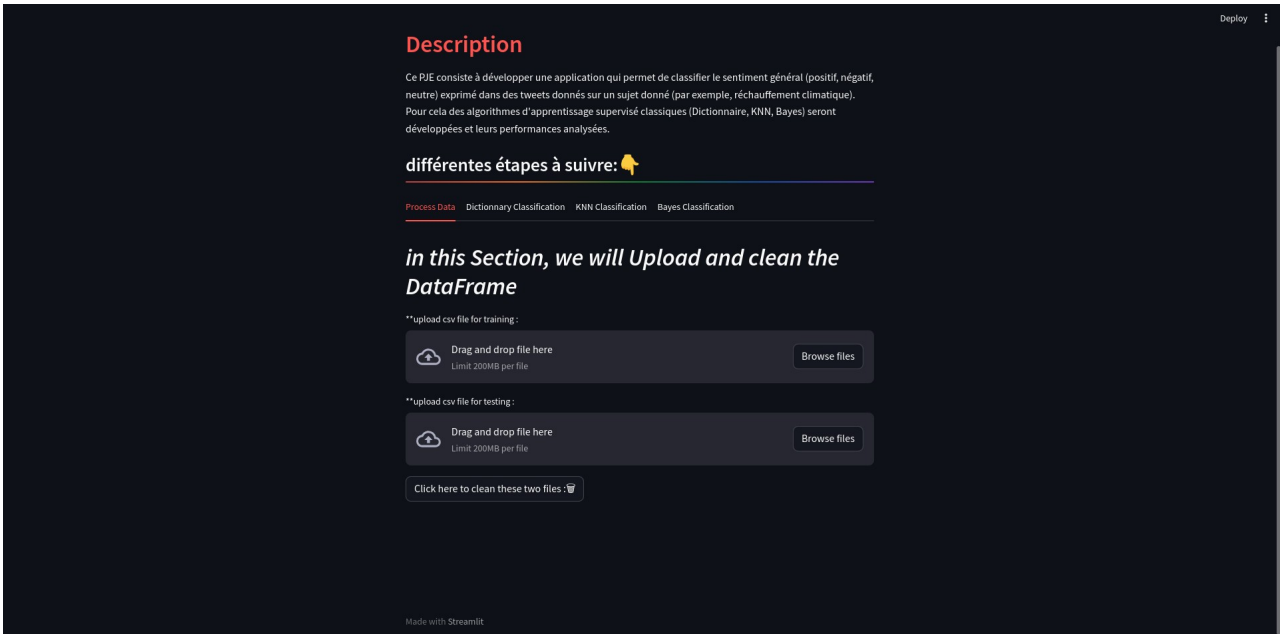
3) Interface graphique

Etape 1: Lancer l'application

Pour lancer l'application ouvrez dans le terminal le dossier contenant l'application et tapez "streamlit run NomDuFichier.py" où NomDuFichier est le nom du fichier conteant l'implémentation de l'interface graphique.

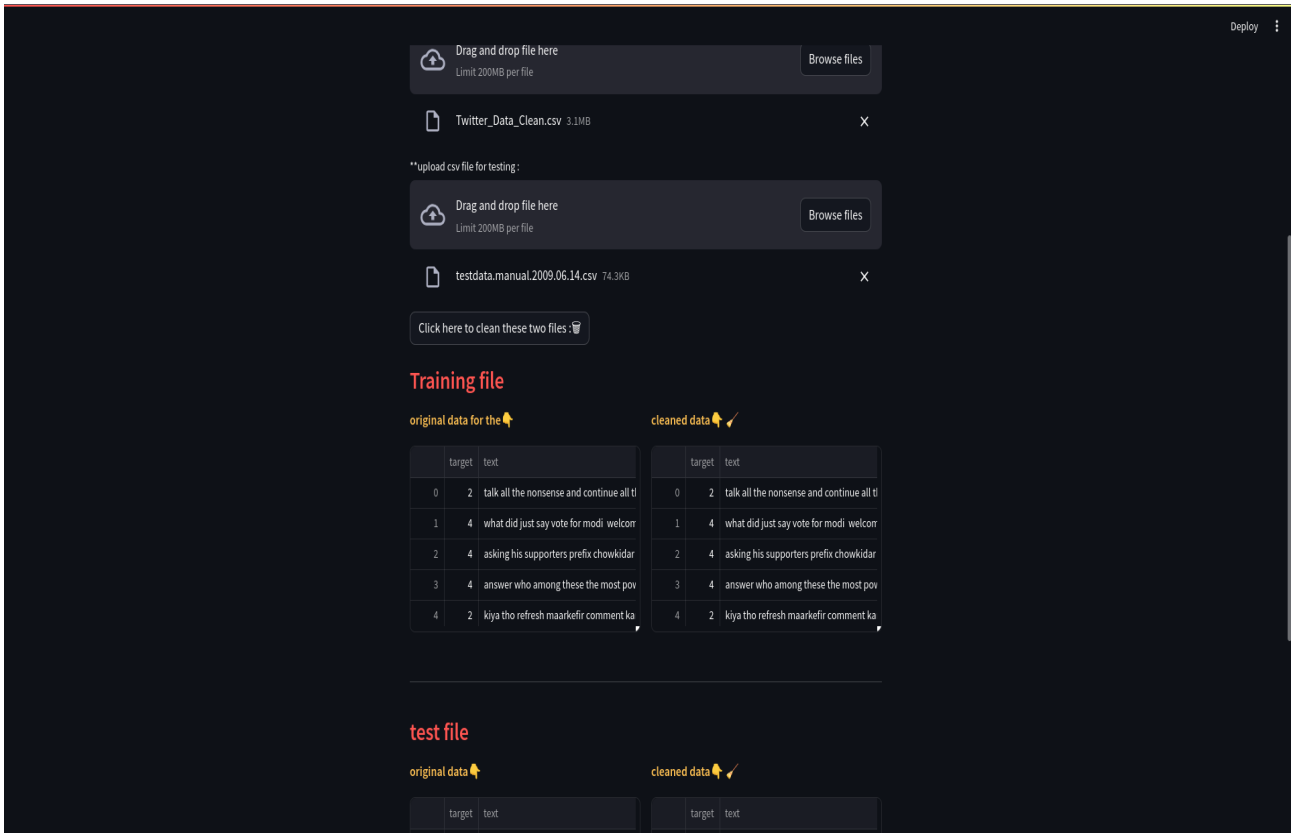
Etape 2: Importez et nettoyer les données

L'application est lancé vous arrivez donc sur cette page d'accueil.



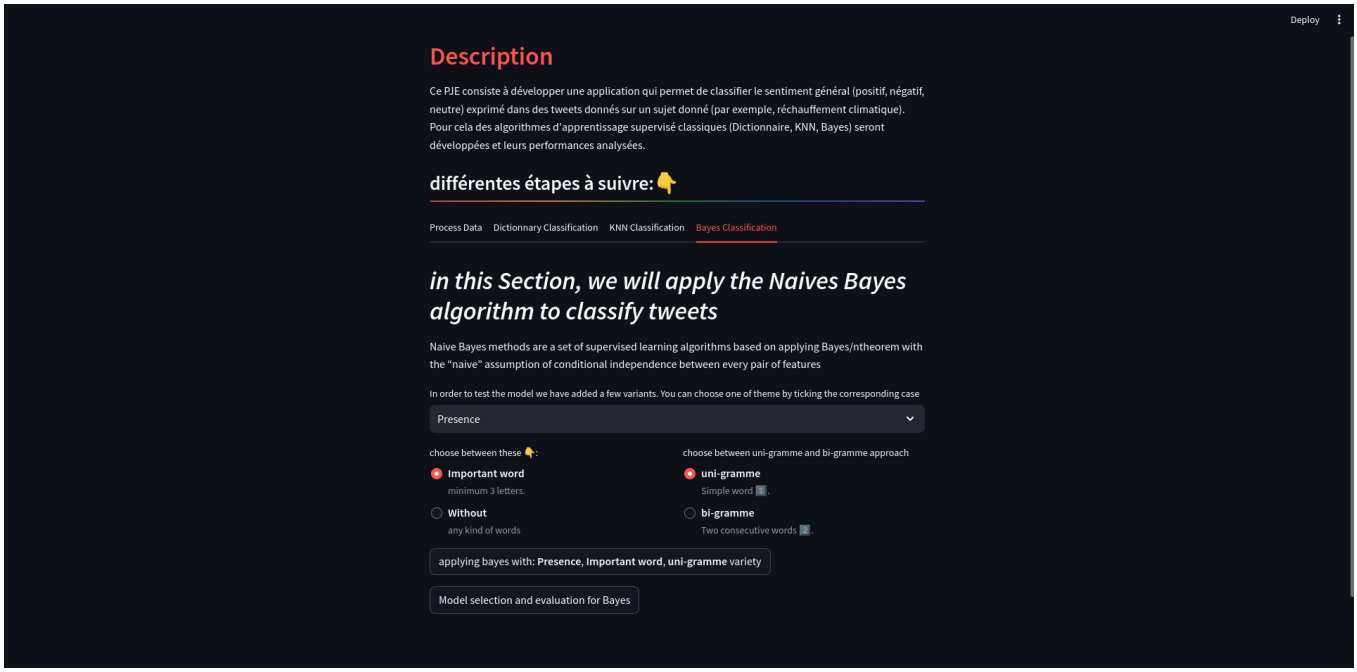
Ici cliquez sur le premier browse files pour importer votre fichier de train et sur le deuxième pour importer votre fichier de test que vous voulez classifier.

Une fois importé cliquez sur “Click here to clean these two files”, cela va nettoyer les fichiers et vous devriez voir en bas afficher un avant/après nettoyage des 2 fichiers comme ceci.



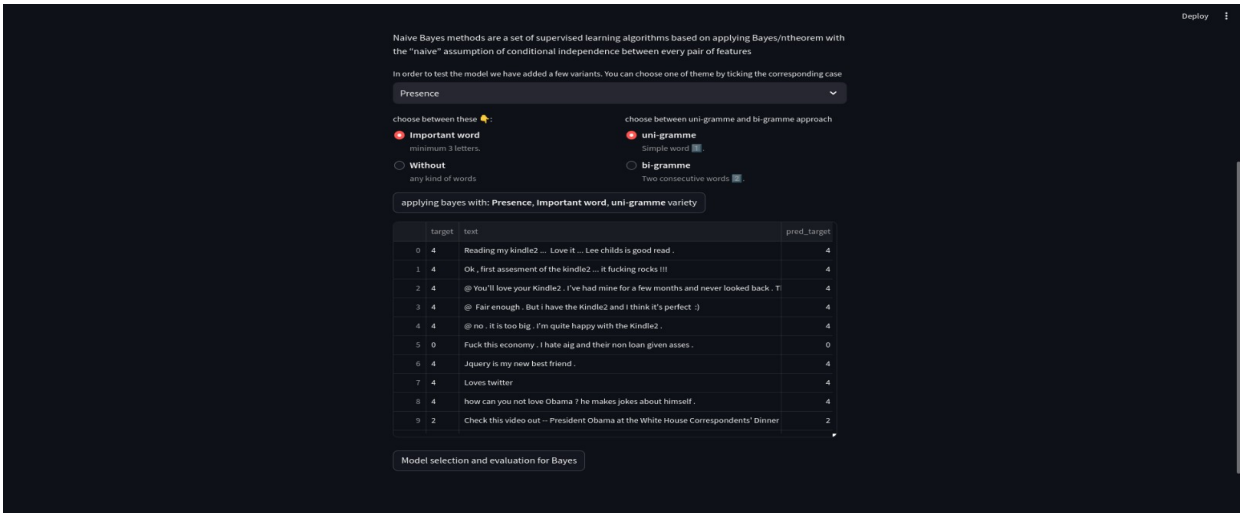
Etape 3: Appliquer un algorithme et regarder son évaluation

Comme vous le voyez il y'a différents onglets : Process Data, Dictionnary Classification, KNN Classification et Bayes Classification. Quand vous ouvrez l'application et que vous traitez les données vous êtes sur Process Data, maintenant si vous voulez utiliser un algorithme il faut cliquer sur l'onglet correspondant à celui que vous voulez utilisez. Disons que l'on veut utiliser un algorithme Bayes cliquez sur l'onglet Bayes Classification vous devriez arriver sur cette page.

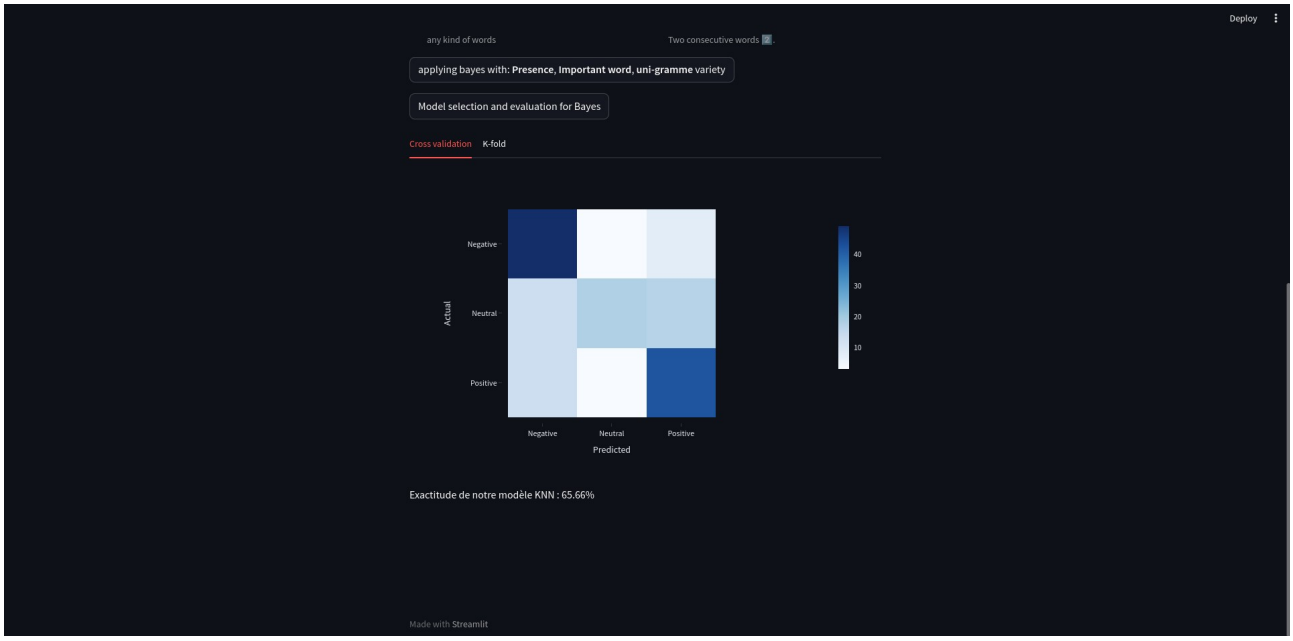


On peut voir que les différentes options du Bayes sont sélectionnables. Choisissez donc présence ou fréquence en déroulant le menu de presence, et choisissez les paramètres de sélection des mots en cliquant sur ceux que vous voulez.

Une fois cela fait vous pouvez cliquez sur Applying Bayes ce qui va sortir le dataframe avec la colonne pred_target qui corresond à la classification de l'algo cela doit donner quelque chose comme ça.



Mais vous pouvez aussi cliquer sur “Model selection and evaluation for Bayes” et vous obtiendrez la matrice de confusion.



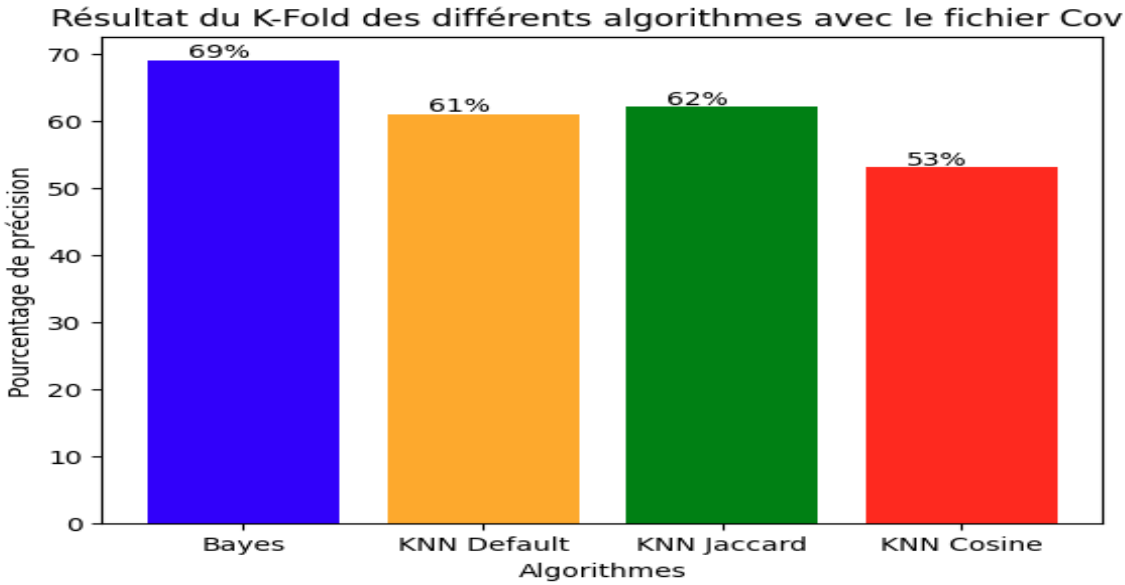
Vous pouvez également cliquez sur K-fold pour avoir une évaluation du modèle plus générale de plus vous n’aurez pas besoin d’avoir les valeurs réelles dans le fichier de test car k-fold utilise uniquement le fichier de train.

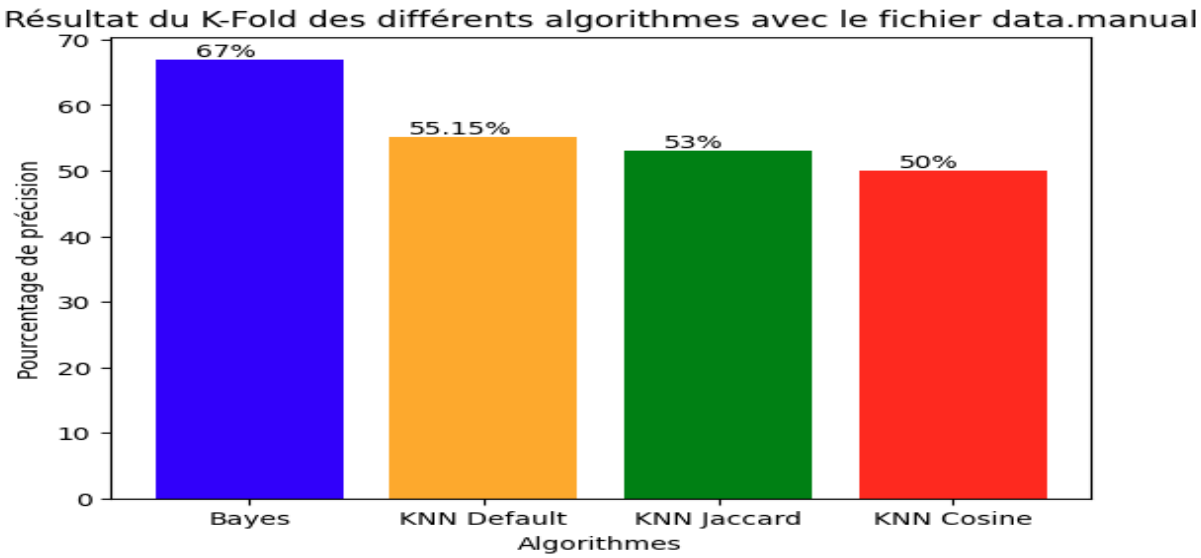
Vous pouvez changer les paramètres et relancer l’algorithme, vous pouvez également retourner sur process data changer les fichiers de train ou test sans avoir besoin de relancer l’application.

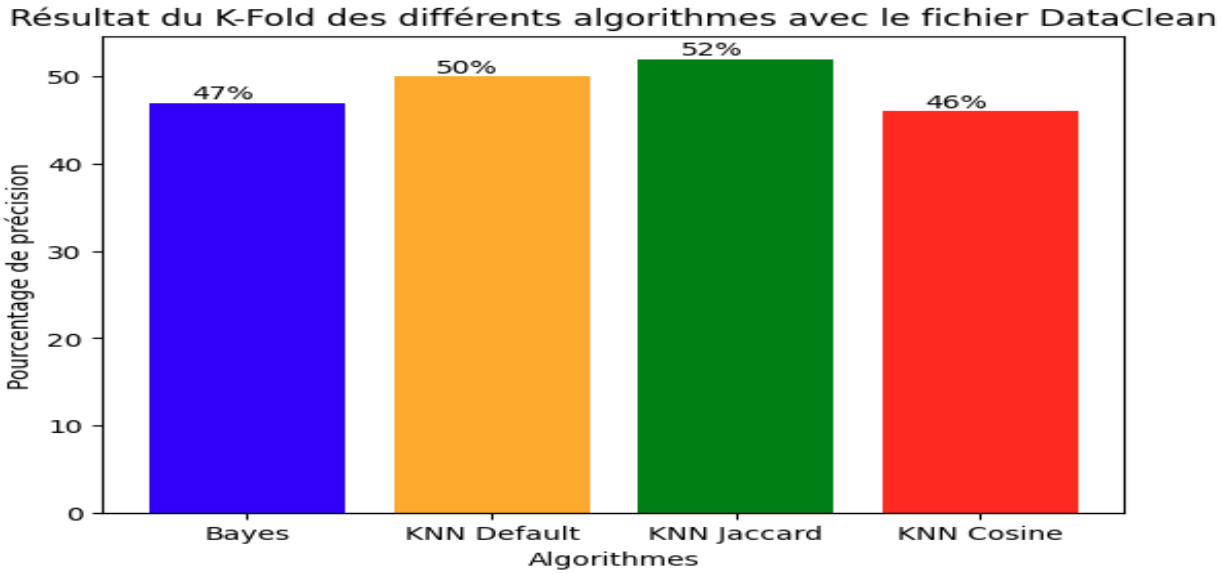
C) Résultats de la classification avec les différentes méthodes et analyse

Pour comparer nos différents algorithmes, nous avons mis en place une cross-validation de type k-fold avec k = 5 (nous avons aussi fait avec k = 10 mais cela donne les mêmes résultats). Nous avons donc sélectionner 3 fichiers que nous avons divisé en 5 parties (p1,p2,p3,p4,p5) et pour chaque algorithme nous les avons entrainé sur 4 des 5 parties et les avons évalué sur la dernière partie et ça 5 fois en changeant à chaque fois la partie à évaluer. Leur score k-fold est donc la moyenne de leur 5 scores. Les fichiers sélectionnés sont le fichier data.manual, covid et le fichier kaggle que nous avons importé.

Voici les résultats trouvés pour chaque fichier. A noter que nous avons représenté les résultats des différents Bayes par un seul Bayes car et ça pour les 3 fichiers le score minimum et le score maximum des différents Bayes ne varie d’à peine 2%.


Résultat du K-Fold des différents algorithmes avec le fichier Covid
Pourcentage de précision
69%
61%
62%
53%
0
10
20
30
40
50
60
70
Bayes
KNN Default
KNN Jaccard
KNN Cosine
Algorithmes


Résultat du K-Fold des différents algorithmes avec le fichier data.manual
Pourcentage de précision
67%
55.15%
53%
50%
0
10
20
30
40
50
60
70
Bayes
KNN Default
KNN Jaccard
KNN Cosine
Algorithmes


Résultat du K-Fold des différents algorithmes avec le fichier DataClean
Pourcentage de précision
47%
50%
52%
46%
0
10
20
30
40
50
Bayes
KNN Default
KNN Jaccard
KNN Cosine
Algorithmes

On peut voir que le Bayes domine aisément sur les fichiers data.manual et Covid mais est beaucoup moins performant sur le fichier importé de Kaggle. Nous pensons que cela vient du fait que les fichiers data.manual et covid ont des thèmes précis ce qui permet à Bayes de mieux exprimer l'efficacité de ses probabilités et d'en avoir des plus fiables. Concernant les KNN, on sent que le Default et Jaccard font à peu près jeu égale par contre Cosine est clairement légèrement en-dessous.

Pour ce qui est de l'accuracy en elle-même sur les fichiers de tests, les algorithmes n'ont pas été très performant. Bayes a avoisiné 33% de précision sur tout les fichiers tout comme le dictionnaire et les KNN ont avoisiné les 40%. Nous pensons que cela vient de la manque de taille des données et notamment pour Bayes de la manque de ressemblance entre les fichiers car quand on le train sur un fichier covid et qu'on le test sur le fichier data.manual ils ne retrouvent pas ces mots et a donc très peu de probabilités fiables.

D) Conclusions

Notre conclusion de ce projet est que la classification de tweet n'est pas chose facile mais pas impossible non plus. Nous pensons qu'avec des bases de données bien plus importante (100 000+), rien que des algorithmes comme Bayes pourrait être rudement efficace alors des algorithmes puissants venant de scikit-learn tels que des arbres de décision, du gradient boosting ou des réseaux de neuronnes nous pensons que ce serait proche du 100%.

Nos algorithmes ont obtenu des scores assez satisfaisant en k-fold mais un peu moins en accuracy réel malgré tout l'on reste satisfait de ce qu'on a pu faire et le projet était vraiment interessant car il touchait un peu à pas mal de choses que ce soit au niveau de l'interface graphique, le traitement des données ou la conception des algos.