Rapport final
présenté par :
NGAUV Axel

Projet de classification de textes

28 décembre 2021

Master 1 de Sciences du Langage
parcours Langue et Informatique
Méthodologie de la Recherche en Informatique

Table des matières

1 Introduction **2**

2 État de l'art **3**

3 Jeu de données (corpus) **4**

4 Méthode **6**

5 Résultats **8**

6 Conclusion et Perspectives **13**

Liste des tableaux

1 Premier jeu de données, *spam* ou *ham* 5
2 Second jeu de données, tweet *négatif* ou *positif* 6

Table des figures

1 Perceptron appliqué sur le premier jeu de données avec pondération TF-IDF et sans caractéristiques 11
2 SVM appliqué sur le premier jeu de données avec pondération TF-IDF et sans caractéristiques 11
3 Perceptron appliqué sur le second jeu de données sans pondération TF-IDF et en allant jusqu'aux trigrammes 12

1 Introduction

Ce projet consiste en la mise en place d'une chaîne de traitement de classification automatique de textes et en une analyse des résultats.
Pour cela, nous avons choisi deux jeux de données. Le premier jeu concerne la détection de SPAM et le second concerne la détection de polarité dans les tweets. Chacun de ces jeux de données est conservé dans un fichier avec l'extension « csv ». Plus de détails sur le corpus seront données dans la partie 3 page 4.

Pour mener à bien ce projet nous utiliserons pour la rédaction du rapport Overleaf, un éditeur LaTeX en ligne, collaboratif en temps réel. Pour la conception du code, ce sera avec Jupyter Notebook. Les notebooks de Jupyter sont « des cahiers électroniques qui, dans le même document, peuvent rassembler du texte, des images, des formules mathématiques et du code informatique exécutable »[Fuchs and Poulain, 2019].
Le langage de programmation que nous utiliserons pour ce projet est Python. Il a été introduit par un programmeur néerlandais, Guido van Rossum, en 1991. Il s'agit d'un langage de programmation orienté objet [Doyle, 2019].
Python est le langage informatique le plus populaire pour le traitement Big Data, l'exécution de calculs mathématiques ou le Machine Learning. De manière générale, il s'agit du langage de prédilection pour la Data Science. Ce langage est extrêmement polyvalent est peut également être utilisé pour le script et l'automatisation (deux principaux cas d'usage), ainsi que pour la programmation d'application et la création de services web par exemple [Margot, 2021].
Citons tout de même quelques inconvénients : la vitesse d'exécution n'est pas son point fort (contrairement aux langages compilés, un interpréteur Python doit convertir l'application ligne par ligne en code machine pour l'exécuter) et étant un langage de haut niveau il n'est pas recommandé pour la programmation système [Doyle, 2019].
Dernière précision : pour ce projet nous utiliserons Scikit-learn, une bibliothèque libre Python destinée à l'apprentissage automatique.

Dans un premier temps, nous présenterons l'état de l'art concernant la mise en place d'une chaîne de traitement de classification automatique de texte. Puis nous donnerons des détails sur les jeux de données choisis et sur la méthode que nous avons mise en place. Dans un troisième temps, nous discuterons des résultats obtenus, et tenterons de les expliquer. Enfin, nous conclurons brièvement avant de nous exprimer sur les perspectives.

2 État de l'art

La classification automatique consiste en une identification automatique de groupes de données similaires dans un ensemble de données et est une composante importante de la fouille de données. « La classification automatique (cluster analysis ou clustering en anglais) cherche à regrouper les données de façon à obtenir des groupes tels que les données sont plus similaires entre elles à l'intérieur d'un même groupe (cluster en anglais) qu'entre groupes »[Crucianu et al., 2016].
Prenons l'exemple de la classification automatique de textes. Elle consiste à attribuer une catégorie (une classe) à chaque texte (instance). L'ensemble des catégories/classes peut être donné au départ, on parle d'approche supervisée. Mais cet ensemble peut ne pas être donné au départ, il s'agit de le *créer* en regroupant les textes en classes qui possèdent une certaine similitude : on parle alors d'approche non supervisée [Devillers and Lejeune, 2021]. La classification automatique « fait couramment appel à l'apprentissage automatique et est largement utilisée en reconnaissance de formes. »[Wikipédia, 2021b].

La reconnaissance des formes est un ensemble de techniques et méthodes visant à identifier des formes à partir de données brutes afin de prendre une décision dépendant de la catégorie attribuée à cette forme. On considère que c'est une branche de l'intelligence artificielle qui fait largement appel aux techniques d'apprentissage automatique et aux statistiques. L'apprentissage consiste en l'estimation des paramètres de modèles à partir d'observations sur un ensemble d'objets, d'individus... La stratégie d'apprentissage peut être décomposée en deux étapes : l'exploration des données et codage, puis apprentissage ou modélisation statistique [Devillers and Lejeune, 2021]. Une situation d'apprentissage typique se présente comme suit :

- Ensemble de données observées
- Nouveau cas, prise de décision (classer le nouveau cas, dire s'il ressemble aux cas déjà vus)

L'objectif est la généralisation. Pour pouvoir généraliser à de nouvelles situations, il peut être nuisible d'apprendre par cœur. Pour procéder à une classification automatique, plusieurs méthodes (ou modèles) existent. Aucune n'est meilleure que les autres. Le choix de la méthode se fait en fonction du problème (une méthode sera plus adaptée qu'une autre selon le problème).
Nous pouvons également préciser que plus un modèle est complexe et plus il intègre de paramètres (et plus il est capable de s'ajuster aux données). En revanche un tel modèle peut s'avérer défaillant lorsqu'il s'agira de prévoir ou de généraliser, c'est à dire de s'appliquer à des données qui n'ont pas parti-

cipé à son estimation [Devillers and Lejeune, 2021].

Parmi les méthodes, citons tout d'abord le Perceptron. Dans le domaine du Machine Learning, le Perceptron est un algorithme d'apprentissage supervisé de classifieurs binaires (séparant deux classes). Il s'agit alors d'un type de classifieur linéaire, et du type de réseau de neurones artificiels le plus simple. Frank Rosenblatt inventa l'algorithme en 1957 [Bastien, 2019].
Les machines à vecteurs de support ou séparateurs à vaste marge (en anglais support-vector machine, SVM) sont un ensemble de techniques d'apprentissage supervisé. Les SVMs sont une famille d'algorithmes d'apprentissage automatique qui permettent de résoudre des problèmes tant de classification que de régression ou de détection d'anomalie. « Ils sont connus pour leurs solides garanties théoriques, leur grande flexibilité ainsi que leur simplicité d'utilisation même sans grande connaissance de data mining. »[Vandeginste, 2021b].
Les séparateurs à vaste marge ont été développés dans les années 1990 à partir des considérations théoriques de Vladimir Vapnik sur le développement d'une théorie statistique de l'apprentissage : la théorie de Vapnik-Chervonenkis [Wikipédia, 2021c].
Les arbres de décisions sont des outils d'aide à la décision qui représentent la situation plus ou moins complexe à laquelle nous devons faire face sous la forme graphique d'un arbre de façon à faire apparaître à l'extrémité de chaque branche les résultats possibles en fonction des décisions prises à chaque étape [Devillers and Lejeune, 2021]. Il s'agit de plus d'une représentation calculable automatiquement par des algorithmes d'apprentissage supervisé [Wikipédia, 2021a].
Nous finissons cette présentation (non exhaustive) des méthodes par la méthode des K plus proches voisins (KNN pour *k-nearest neighbors*). Il s'agit d'une méthode d'apprentissage supervisée [Wikipédia, 2021e]. C'est un algorithme qui repose exclusivement sur le choix de la métrique de classification. L'idée est la suivante : « à partir d'une base de données étiquetées, on peut estimer la classe d'une nouvelle donnée en regardant quelle est la classe majoritaire des k données voisines les plus proches (d'où le nom de l'algorithme) » [Vandeginste, 2021a].

3 Jeu de données (corpus)

Mon corpus est constitué de deux jeux de données, chacun constituant un sous-corpus. Ces deux sous-corpus sont des fichiers « csv » (pour *comma-separated values*, c'est à dire « valeurs séparées par des virgules »). Il s'agit d'un format texte ouvert représentant des données tabulaires sous forme de

valeurs séparées par des virgules.

Le premier sous-corpus pèse 8,53 Mo et concerne la détection de spam. Il est consitué de deux colonnes et de 5729 lignes. La première colonne contient du texte en anglais (un texte par ligne à partir de la deuxième ligne, donc 5728 textes différents). En ce qui concerne la seconde colonne, chaque ligne (à partir de la deuxième) contient soit le caractère 0 si le texte correspondant n'est pas du spam, soit le caractère 1 si le texte est bien du spam.
Ainsi, le premier jeu de données est constitué de deux classes (ham et spam) et de 5728 instances (les textes). La première classe (ham) comporte 4360 instances, la seconde classe (spam) comporte 1368 instances. La taille des jeux de *train* et de *test* sont respectivement de 0,7 et 0,3.
Exemples d'instances pour ce jeu de données :

— « Subject : security alert - confirm your national credit union information - - » est une instance de la classe *spam*
— « Subject : presentation george , this is the presentation i promised . vince » est une instance de la classe *ham*

Poids	**Taille**	**Instances par classes**	**Taille *train* et *test***
8,53 Mo	5728 instances 2 classes	4360 (*ham*) 1368 (*spam*)	*train* = 0,7 *test* = 0,3

TABLE 1 – Premier jeu de données, *spam* ou *ham*

Le second sous-corpus pèse 123 Mo et concerne la détection de polarité par rapports aux sentiments dans les tweets. Il est consitué de deux colonnes et de 1 526 725 lignes. En ce qui concerne la première colonne, chaque ligne (à partir de la deuxième) contient soit le caractère 0 si le texte correspondant dans la seconde colonne est un tweet dont le sentiment est négatif, soit le caractère 1 si le texte correspondant dans la seconde colonne est un tweet dont le sentiment est positif. La seconde colonne contient du texte, il s'agit de tweets en français (un tweet par ligne).
Ainsi, le second jeu de données est constitué de deux classes (négatif et positif) et de 1 526 724 instances (les tweets). La première classe (négatif) comporte 771 604 instances, la seconde classe (positif) comporte 755 120 instances. La taille des jeux de *train* et de *test* sont respectivement de 0,7 et 0,3.
Exemples d'instances pour ce jeu de données :

— « Il est la raison de la larme sur ma guitare, le seul qui a assez de moi pour me casser » est une instance de la classe *négatif*
— « Un autre bon, merci ! Mais il fallait écouter à haute voix à travers

les héros comme toujours pas encore vu jusqu'ici. En arrivant. » est une instance de la classe *positif*

Poids	**Taille**	**Instances par classes**	**Taille *train* et *test***
123 Mo	1 526 724 instances	771 604 (*négatif*)	*train* = 0,7
	2 classes	755 120 (*positif*)	*test* = 0,3

TABLE 2 – Second jeu de données, tweet *négatif* ou *positif*

4 Méthode

Afin de pouvoir mesurer par la suite l'influence des caractéristiques (features) que nous avons choisi de tester (et leur combinaison entre eux), nous avons tout d'abord effectué un test sans aucune caractéristique. Cette étape est essentielle afin d'avoir une *baseline* pour pouvoir par la suite constater les changements apporter par les paramètres.
Puis, nous avons utilisé différentes caractéristiques :

- `Ngram_range` afin de voir l'influence que cela pouvait avoir sur les résultats pour les mots contigus (n-grammes) jusqu'aux bigrammes (1, 2) et jusqu'aux trigrammes (1, 3).
- `Stop_words` afin d'enlever les mots vides. Pour l'anglais, une liste est directement disponible avec Scikit-learn, mais pour le français il a été nécessaire de passer par nltk (Natural Language Toolkit) pour avoir une liste de mots outils adapté au français.
- Une combinaison des deux caractéristiques précédentes (jusqu'aux bigrammes seulement pour le second jeu de données par manque de temps, mais il aurait été intéressant de tester également jusqu'aux trigrammes afin de constater l'évolution qu'apporte la combinaison de ces deux caractéristiques pour ces deux cas).

Quelques définitions pour éclaircir tout cela...
En recherche d'information, un mot vide (ou *stop word*, en anglais) est un mot qui est tellement commun qu'il est inutile de l'indexer ou de l'utiliser dans une recherche. Il s'agit donc d'un mot non significatif figurant dans un texte. On l'oppose au mot plein. La signification d'un mot s'évalue à partir de sa distribution (au sens statistique) dans une collection de textes. [Wikipédia, 2021d].
Les n-grammes sont des « séquences de mots, des chaînes de caractères qui se suivent et qui forment des sacs de mots. »[Duong, 2011].

Pour mener à bien ce projet, de nombreux outils ont été utilisés :

— Pandas pour lire les fichiers « csv » (Pandas est une bibliothèque écrite pour le langage de programmation Python permettant la manipulation et l'analyse des données) [Wikipédia, 2021f].
— Le module *time* de Python qui nous a permis de mesurer le coût en temps pour chacun des tests.
— Le package *json* de Python afin de pouvoir enregistrer les résultats dans des fichiers « json ». L'intérêt de ces fichiers est qu'ils sont directement lisible par un langage de programmation, le format étant conçu pour l'échange de données (transférer des données qui soient parsables). Comme nous l'avons vu durant le cours, ce format trouve un équilibre entre flexibilité et interopérabilité. Sa structure est légère à mettre en place, cependant elle est lourde à mettre à jour.
— Le module *os* de Python afin de pouvoir vérifier par la suite si un test a déjà était fait (si le fichier de résultat existe déjà) pour ne pas avoir à perdre du temps à répéter plusieurs fois la même opération.
— La bibliothèque *nltk* afin d'avoir une liste de mots vides pour la langue française car le second jeu de données est en français.
— Le module `sklearn.model_selection` pour séparer les corpus en corpus d'entraînement et corpus de test.
— Enfin, pour l'évaluation nous avons utilisé le module *sklearn.metrics* afin de calculer la précision, le rappel, la f-mesure, l'*accuracy* (l'exactitude), la *macro avg* (macro moyenne) ainsi que la *weighted avg* (micro moyenne). Ce module nous a également permis de produire des matrices de confusion.

Les classifieurs que nous avons choisi de tester sont le Perceptron, le *DecisionTreeClassifier* (arbre de décision), le *KNeighborsClassifier* (K plus proches voisins ou KNN) et le SVC/*Support Vector Classification* (machine à vecteurs de support ou SVM).
Concernant ce dernier, après plus de 40h d'attente, nous nous voyons contraint d'arrêter son test sur le deuxième sous-corpus (jeu de données correspondant aux tweets en français) pour nous concentrer sur la suite des tests.
Une des difficultés les plus grandes que nous rencontrons dans ce projet, c'est la contrainte de temps : le jeu de données sur les tweets est énorme, et le classifieur SVM a un coût en temps considérable. Comme nous n'effectuons plus exactement la même chaîne de traitement selon les corpus, il a été nécessaire de défactoriser une partie de mon code.

Les différentes étapes de ma chaîne de traitement :
— Lecture des fichiers « csv »
— Récupération des instances (X) et des classes (y) pour chaque corpus

— Sélection des caractéristiques et de l'outil pour l'extraction (CountVectorizer, ou TfidfVectorizer pour la pondération : donner une importance plus grande à certains mots)
— Pré-traitement des instances avec la sélection de l'outil et des caractéristiques
— Définitions d'une liste des classifieurs que nous allons utiliser
— Enfin, on effectue l'expérience pour chaque classifieur et pour chacun des deux jeux de données. A chaque fois, les résultats sont enregistrés dans un fichier portant le nom du classifieur suivi du nom de l'outil d'extraction avec les paramètres sélectionnées. Sur le même principe, une matrice de confusion est également créée. Si l'expérience a déjà été faite, nous passons directement à la suite (chaque expérience n'est réalisée qu'une fois, ce qui constitue un gain de temps non négligeable).

Ainsi, nous pourrons comparer l'évolution des résultats selon les caractéristiques sélectionnées, les différents classifieurs, l'effet de la pondération, mais nous serons également en mesure de constater si les résultats évoluent de la même manière sur les deux jeux de données.

5 Résultats

Quelques définitions afin de mieux comprendre les résultats...
La précision est le nombre de documents pertinents retrouvés rapporté au nombre de documents total proposé pour une requête donnée. Le rappel est défini par le nombre de documents pertinents retrouvés au regard du nombre de documents pertinents que possède la base de données. La F-mesure ou F-score et la moyenne harmonique entre la précision et le rappel.
La précision et le rappel peuvent être appréciés avec une matrice de confusion qui va permettre de mettre en évidence les vrais positifs, les faux positifs, les faux négatifs et les vrais négatifs.
Un résultat est dit *vrai positif* lorsqu'un item est correctement détecté par le test. Il est dit *faux positif* lorsqu'un item est déclaré positif alors qu'il ne l'était pas. Il est dit *faux négatif* lorsqu'un item est déclaré négatif alors qu'il était en réalité positif. Il est dit *vrai négatif* lorsqu'un item est correctement déclaré comme négatif.
Ainsi, dans le premier jeu de données par exemple, les vrais positifs sont les instances spam qui ont été prédites comme spam, les faux positifs les instances ham qui ont été prédites commes spam, les faux négatifs sont les instances spam qui ont été prédites comme ham, et les vrais négatifs ont les instances ham qui ont été prédites comme ham. Le rappel et la précision se

calculent comme suit :

— Rappel = Vrai positif/(Vrai positif + Faux négatif)

— Précision = Vrai positif/(Vrai positif + Faux positif)

Pour les résultats, nous avons également calculé l'exactitude mais aussi la macro moyenne et la micro moyenne, comme expliqué dans la partie précédente concernant la méthode. En effet, l'exactitude ne représente pas forcément une bonne évaluation. Le fait de compter toutes les erreurs dans le même sac peut masquer la cause et leur importance parce que nous ne tenons pas compte du fait que les classes soient équilibrées ou non.
Avec la micro moyenne de la F-mesure qui est une moyenne pondérée des F-mesure, une classe compte en fonction de sa taille, c'est à dire que nous comptons par instance et cela représente donc mieux la classe majoritaire.
Avec la macro moyenne de la F-mesure qui est une moyenne des F-mesure de chaque classe indépendamment de leur taille, la classe minoritaire généralement plus difficile à trouver (car il y a moins d'exemples, et le classifieur, cherchant la plus grande probabilité va en cas de doute mettre le cas dans la classe majoritaire : il a donc moins tendance à prédire la classe minoritaire) est mieux représentée. Nous choisisons donc de nous attarder plutôt sur cette moyenne.

Dans les fichiers où sont écrits les résultats, le terme de *support* représente le nombre d'instances concernées. Nous remarquons en lisant les résultats contenus dans les fichiers json et en regardant les matrices de confusions associées que concernant le premier jeu de données :

— Pour le classifieur Decision Tree nous obtenons les meilleurs résultats en allant jusqu'aux bigrammes et en enlevant les mots outils : 1263 vrais négatifs et 373 vrais positifs avec une macro moyenne de la F-mesure de 0,9340. Et lorsque nous faisons une pondération avec TF-IDF, les résultats s'améliorent encore (les meilleurs résultats sont obtenus en allant jusqu'aux bigrammes mais sans enlever les mots vides : 1277 vrais négatifs et 375 vrais positifs avec une macro moyenne de la F-mesure de 0,9462).

— Pour le classifieur KNN nous obtenons les meilleurs résultats en ne mettant aucune des caractéristiques utilisées dans ce projet (peut-être que des caractéristiques que nous n'avons pas utilisées seront susceptibles d'améliorer les résultats) : 1299 vrais négatifs et 259 vrais positifs pour une macro moyenne de la F-mesure de 0,8523. Nous pouvons également noter qu'en enlevant les mots vides, nous obtenons exactement les mêmes résultats (que cela soit en combinaison avec les n-grammes ou non, seule la combinaison trigramme en enlevant les mots vides modifie légèrement les résultats en faisant perdre un faux

négatif par rapport aux seuls trigrammes). En pondérant avec TF-IDF, les résultats évoluent positivement : les meilleurs résultats sont données par le même cas de figure que nous venons juste de voir, mais les résultats sont cette fois-ci plus élevés avec 1290 vrais négatifs et 385 vrais positifs pour une macro moyene de la F-mesure de 0,9646. Pour ce jeu de données, le classifieur KNN est plus performant que le Decision Tree.

— Pour le classifieur Perceptron nous obtenons les meilleurs résultats en ne mettant là encore aucune des caractéristiques utilisées dans ce projet et la suppresion des mots vides n'a pas d'importance : 1299 vrais négatifs et 392 vrais positifs pour une macro moyenne de la F-mesure de 0,9774. En pondérant avec TF-IDF, les résultats s'améliorent (toujours sans aucune caractéristique) : 1310 vrais négatifs et 393 vrais positifs pour une macro moyenne de la F-mesure de 0,9870. La suppression des mots vides ne change rien ici aussi sauf pour les trigrammes ou la suppression des mots vides améliore un peu les résultats : nous perdons 7 faux négatifs mais nous gagnons 2 faux positifs comparé aux résultats pour les trigrammes avec mots vides. Pour ce jeu de données, le Perceptron est donc pour le moment le meilleur classifieur.
— Pour le classifieur SVM nous obtenons les meilleurs résultats encore une fois sans caractéristique et la suppresion des mots vides ne fait pas évoluer les résultats : 1307 vrais négatifs et 333 vrais positifs pour une macro moyenne de la F-mesure de 0,9323. En pondérant avec TF-IDF, la suppression des mots vides ne fait toujours pas évoluer les résultats et ces derniers sont les meilleurs sans caractéristiques, encore une fois : 1313 vrais négatifs et 390 vrais positifs pour une macro moyenne de la F-mesure de 0,9869. Pour ce premier jeu de données, le meilleur classifieur est donc le Perceptron, suivit de très (très) près par le classifieur SVM.

Pour illustrer ces résultats, nous avons créer des matrices de confusions. Une matrice de confusion (*confusion matrix*) est un outil permettant de mesurer les performances d'un modèle de Machine Learning en vérifiant notamment à quelle fréquence ses prédictions sont exactes par rapport à la réalité dans des problèmes de classification. Il s'agit d'un résumé des résultats de prédictions sur un problème de classification. Les prédictions correctes et incorrectes sont mises en lumière et réparties par classe. Les résultats sont ainsi comparés avec les valeurs réelles. Cette matrice permet de comprendre de quelle façon le modèle de classification est confus lorsqu'il effectue des prédictions. Ceci permet non seulement de savoir quelles sont les erreurs commises, mais surtout le type d'erreurs commises.

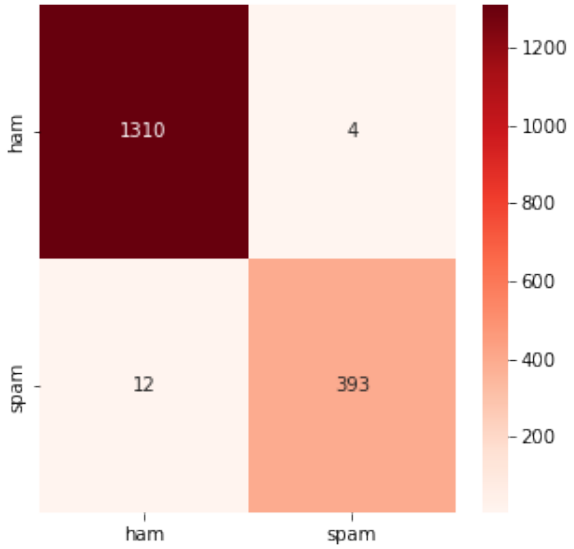

FIGURE 1 – Perceptron appliqué sur le premier jeu de données avec pondération TF-IDF et sans caractéristiques

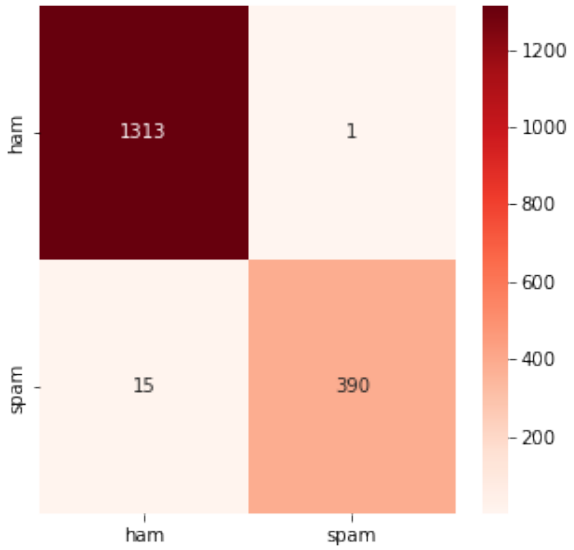

FIGURE 2 – SVM appliqué sur le premier jeu de données avec pondération TF-IDF et sans caractéristiques

Passons maintenant au second jeu de données :

— Pour le classifieur Decision Tree nous obtenons les meilleurs résultats en allant jusqu'aux trigrammes (et on constate que les résultats sont moins bons lorsque nous supprimons les mots vides ou que nous pondérons les résultats) : 169 078 vrais négatifs et 163 795 vrais positifs pour une macro moyenne de la F-mesure de 0,7267.

— Pour le classifieur KNN nous obtenons les meilleurs résultats sans pondération et sans caractéristiques (et les résultats sont moins bons lorsque nous supprimons les mots vides ou que nous pondérons les résultats) : 155 018 vrais négatifs et 165 939 vrais positifs pour une macro moyenne de la F-mesure de 0,7006. Le classifieur Decision Tree est donc meilleur sur ce jeu de données que KNN.

— Pour le classifieur Perceptron nous obtenons les meilleurs résultats en allant jusqu'aux trigrammes (et on constate que les résultats sont moins bons lorsque nous supprimons les mots vides ou que nous pondérons les résultats) : 174 112 vrais négatifs et 181 813 vrais positifs pour une macro moyenne de la F-mesure de 0,7770. Le Perceptron est donc sur ce jeu de données également, le meilleur classifieur.

Illustrons cela avec une matrice de confusion :

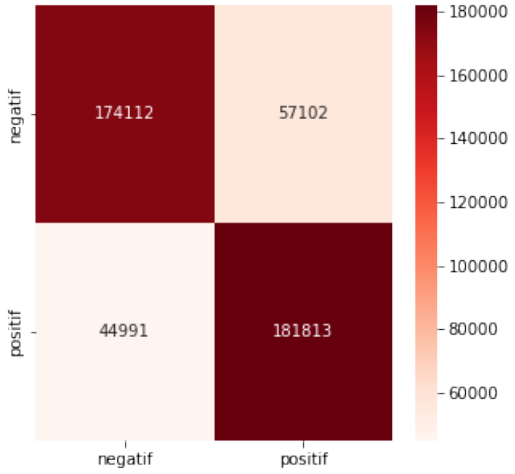

FIGURE 3 – Perceptron appliqué sur le second jeu de données sans pondération TF-IDF et en allant jusqu'aux trigrammes

6 Conclusion et Perspectives

Ainsi, pour le premier jeu de données, les meilleurs résultats sont obtenus avec le Perceptron, en effectuant une pondération avec TF-IDF et sans caractéristiques. Nous avons également constaté que la pondération améliorait toujours les résultats et que la suppression des mots vides n'influençait que très peu (voire pas du tout dans la plupart des cas) les résultats.
Pour le second jeu de données en revanche, même si les meilleurs résultats sont toujours obtenus avec le Perceptron, la pondération ainsi que la suppression des mots vides font évoluer les résultats de manière négative.
Enfin, nous avons pu remarquer que les résultats sont moins bons pour le second jeu de données que le premier, et cela peu importe le classifieur, les caractéristiques (ou leur absence) et cela est encore plus vrai lorsque nous effectuons une pondération avec TF-IDF.

Pour aller plus loin, nous aurions pu effectuer tous les tests que nous avons fait avec le premier jeu de données pour le second (nous rappelons que nous n'avons pas pu tester le SVM sur le second jeu de données) afin de pouvoir constater le plus justement possible quels changements (une caractéristique, une combinaison de caractéristiques, un classifieur en particulier, la valeur ajoutée ou non de la pondération avec TF-IDF, ou une combinaison de tout cela) impliquent une amélioration des résultats et si ces changement n'améliorent les résultats que pour un seul des deux jeux de données, ou pour les deux. En ce qui concerne le premier jeu de données, le SVM a été moins bon que le Perceptron, mais de très peu seulement. Qu'en aurait-il été avec le second jeu de données ?
Nous pourrions également essayer de faire varier la taille des jeux de *train* et de *test* ou encore tester d'autres caractéristiques tels que *lowercase* qui convertit tous les caractères en minuscules par exemple. Nous ne sommes pas allé jusqu'au bout en explorant toutes ces possibilités par manque de temps, mais c'est une piste intéressante qui mérite d'être creusée.

Références

[Bastien, 2019] Bastien, L. (2019). Tout savoir sur le plus vieil algorithme de machine learning.

[Crucianu et al., 2016] Crucianu, M., Fournier-S'niehotta, R., Ferecatu, M., and Audebert, N. (2016). Classification automatique.

[Devillers and Lejeune, 2021] Devillers, L. and Lejeune, G. (2021). Méthodologie de la recherche en langue et informatique.

[Doyle, 2019] Doyle, K. (2019). Python : un langage avantageux, mais pas pour tout le monde.

[Duong, 2011] Duong, V. (2011). Définition de n-grammes.

[Fuchs and Poulain, 2019] Fuchs, P. and Poulain, P. (2019). Jupyter et ses notebooks.

[Margot, 2021] Margot, P. (2021). Python : Focus sur le langage le plus populaire.

[Vandeginste, 2021a] Vandeginste, P. (2021a). k-nearest neighbours.

[Vandeginste, 2021b] Vandeginste, P. (2021b). SVM.

[Wikipédia, 2021a] Wikipédia (2021a). Arbre de décision.

[Wikipédia, 2021b] Wikipédia (2021b). Classement automatique.

[Wikipédia, 2021c] Wikipédia (2021c). Machine à vecteurs de support.

[Wikipédia, 2021d] Wikipédia (2021d). Mot vide.

[Wikipédia, 2021e] Wikipédia (2021e). Méthode des k plus proches voisins.

[Wikipédia, 2021f] Wikipédia (2021f). Pandas.