Rapport du Projet de Réseau de Neurones

Master 1 Systèmes Informatiques Intelligents

Enseignante : Mme H. MOULAI

SAYOUD Maissa **191931040670**

BOULKABOUL Amira **202031043294**

Mai 2024

***

Table des matières

Introduction

Dans le cadre de ce projet, nous avons fait une analyse approfondie des données suivie d'une phase d'apprentissage supervisé à l'aide de techniques de deep learning. Notre objectif était de développer un modèle capable de classifier efficacement des données dans un contexte spécifique.

Nous avons commencé par une phase de nettoyage des données afin de garantir la qualité et la cohérence des informations utilisées pour l'apprentissage.Une fois les données nettoyées, nous avons procédé à une exploration approfondie pour comprendre la distribution des variables, les relations entre les caractéristiques et la variable cible. Après avoir exploré les données, nous avons sélectionné plusieurs modèles de machine learning et de deep learning pour notre tâche de classification. Nous avons ensuite entraîné les modèles sélectionnés et enfin, nous avons comparé les performances des différents modèles pour sélectionner celui offrant les meilleures performances pour notre tâche.

Partie I :
Préparation des Données

1. Introduction

Avant de se lancer dans l'apprentissage automatique, nous devons préparer les données. Dans cette section, nous nettoyons et normalisons les tweets pour créer un ensemble de données cohérent. Nous convertissons le texte en minuscules, supprimons les balises HTML, réduisons les mots à leur forme radicale, supprimons la ponctuation et les mots vides.

Nous abordons aussi deux étapes essentielles de la préparation des données : la construction du vocabulaire et l'extraction de caractéristiques. Après le prétraitement des tweets, nous sélectionnons les mots pertinents pour notre modèle et les stockons dans un fichier de vocabulaire. Ensuite, nous convertissons chaque tweet en un vecteur numérique en utilisant deux méthodes : la représentation binaire et la représentation par comptage.

A la fin de cette partie, nous aurons un ensemble de données structuré et prêt à être utilisé pour entraîner un modèle d'analyse de sentiment.

2. Description des Données du « sentiment140 »

Le jeu de données « sentiment140 » comprend 1.6M d'informations relatives aux caractéristiques des tweets de différents utilisateurs collectés en 2009 dans la période du Lundi 06 Avril à 22:19:45 PDT jusqu'au Mardi 16 Juin à 08:40:50 PDT. Nous remarquons que les tweets sont divisés en deux grandes classes annotées avec les valeurs '0' et '4' (800000 tweets négatifs et 800000 tweets positifs).

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1600000 entries, 0 to 1599999
Data columns (total 6 columns):
 #   Column  Non-Null Count    Dtype
---  ------  --------------    -----
 0   target  1600000 non-null  int64
 1   ids     1600000 non-null  int64
 2   date    1600000 non-null  object
 3   flag    1600000 non-null  object
 4   user    1600000 non-null  object
 5   text    1600000 non-null  object
dtypes: int64(2), object(4)
memory usage: 73.2+ MB
```

Figure 1. Informations sur les types de données dans « sentiment140 »

La figure 1 montre comment les données sont caractérisées par différentes colonnes : 'target' la polarité du tweet (0 = négatif, 4 = positif), 'ids' l'identifiant du tweet, 'date' la date du tweet,

‘flag’ la requête (les tweets ne contiennent aucune requête, les champs de cette colonne pour tous les tweets sont: NO_QUERY), ‘user’ l’utilisateur qui a tweeté et ‘text’ le texte du tweet que l’utilisateur a écrit. Chaque ligne représente des tweets différents de différents utilisateurs.

3. Importation des Librairies Nécessaires

Pour cette étape, nous avons fait appel à différentes bibliothèques Python, notamment :

- **Pandas** : pour avoir un ‘pandas dataframe’ dans lequel nous stockons les données du dataset.
- **Random** : pour générer des nombres aléatoires.
- Re : pour pouvoir utiliser les regex et préciser les chaines de caractères à chercher dans le dataset.
- **Numpy** : pour la vectorisation principalement et pour convertir les données en tableaux de format approprié pour le traitement par Keras.
- **String** : pour manipuler les chaines de caractères.
- **Nltk** : dont nous citons :
    - nltk.corpus et nltk.tokenize pour l’importation les stopwords et les word_tokenize respectivement.
    - nltk.stem pour appeler l’algorithme de radicalisation « PorterStemmer ».
- **Collections** : pour compter le nombre d'occurrences de chaque mot dans les tweets du dataset.
- **Spacy :** pour diviser le texte en tokens.
- **sklearn.feature_extraction.text :** pour utiliser CountVectorizer qui sert dans la vectorisation.
- **matplotlib.pyplot :** pour générer des graphiques.
- **wordcloud :** pour mettre en évidence les mots les plus fréquents dans le texte en format visuel.
- **Seaborn :** pour créer un graphique de distribution des cibles.

4. Nettoyage des Tweets

Nous avons chargé toutes les données du « sentiment140 » dans un dataset (pandas dataframe) de taille (1600000x6) et nous avons visualisé trois données aléatoires comme montré dans la figure 2.

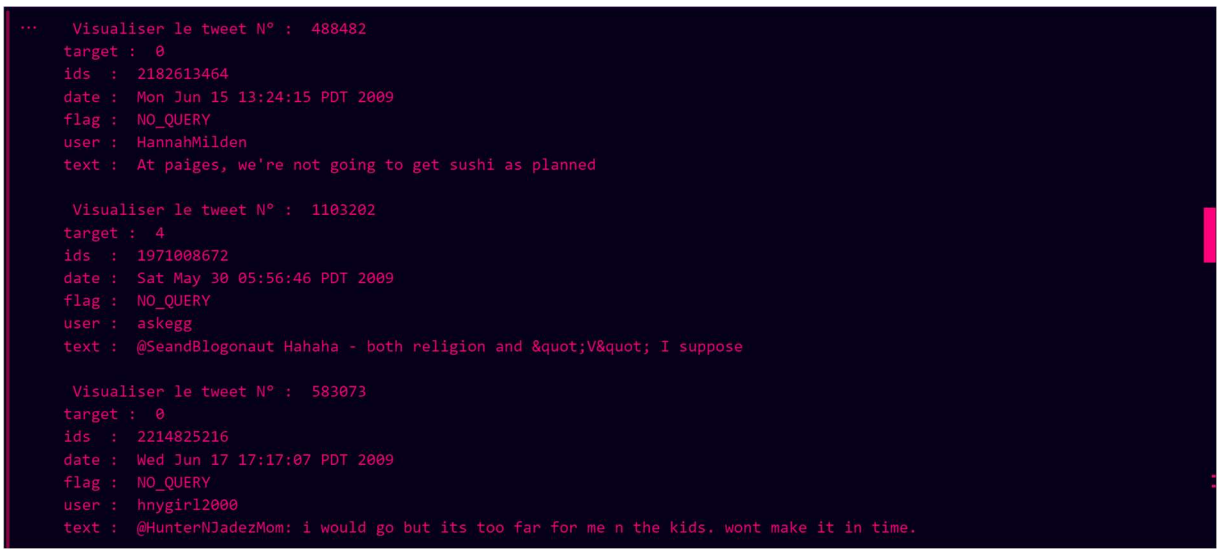

Figure 2. Exemple de trois tweets aléatoires

Cependant les colonnes comme "ids" (la grande majorité des identifiants des utilisateurs sont distincts), "date" aussi, "flag" (NO_QUERY pour tous les tweets), et "user" (comme les identifiants, pratiquement distincts) ne sont pas nécessaires pour l'analyse de sentiment, et donc nous les éliminons dans l'étape de suppression des colonnes inutiles.

4.1. Suppression des Colonnes Inutiles

Nous supprimons les colonnes "ids", "date", "flag" et "user", pour simplifier l'ensemble de données et réduire sa dimension. Le nouveau dataset est devenu donc de taille (1600000x2). Les deux colonnes restantes sont 'target' (polarité) et 'text'.

4.2. Suppression des URLs

Nous avons implémenté la fonction `remove_url(input_text)` dont nous supprimons les URLs des textes en utilisant une expression régulière pour remplacer les URLs par une chaîne vide, et ceci pour chaque case de la colonne 'text' du dataset.

Remarque : nouvelles colonnes ont été créées puis supprimées à la fin du nettoyage pour stocker les nouveaux tweets nettoyés dans le but de pouvoir suivre et faire une comparaison entre les tweets originaux et les tweets apres les modifications apportées.

4.3. Suppression des Balises HTML

Bien que notre dataset ne contient pas de balises HTML, nous implémentons quand même la fonction `remove_html_tags(input_text)` qui supprime les balises HTML en utilisant une expression régulière pour rechercher et remplacer les balises par une chaîne vide.

4.4. Suppression de tags (@user)

Nous avons implémenté pour ce cas la fonction ‘def remove_pattern(input_text, pattern)’ qui prend en entrée un texte (input_text) et un motif (pattern) à rechercher dans ce texte. Elle utilise la bibliothèque regex ‘re’ pour rechercher toutes les occurrences du motif dans le texte en parcourant chaque occurrence trouvée et la supprimer du tweet.

4.5. Suppression des Non-Mots et de la Ponctuation

Pour supprimer les caractères spéciaux et la ponctuation nous avons implémenté la fonction ‘remove_non_words’ qui supprime tous les caractères non alphabétiques (tel que "ðµð¼") et numériques du texte du tweet.

4.6. Suppression des Mots Courts

Nous filtrons aussi les mots très courts des tweets nettoyés car ces mots sont souvent peu ou non significatifs. Nous avons donc supprimé les mots de moins de trois caractères pour conserver uniquement les termes qui ont un sens pertinent.

4.7. Conversion en Minuscule

Nous convertissons l’intégralité des tweets en minuscules pour qu’il n’y est pas des confusions lors de l’apprentissage et que le modèle ne confondra pas par exemple entre le mot ‘feeling’ et le mot ‘Feeling’.

4.8. Suppression des Mots Vides (Stopwords)

Les mots vides (stopwords) sont des mots qui sont souvent des prépositions, des conjonctions et des articles. Ils sont nombreux et n’ont pas un sens pertinent dans l’analyse, d’autant plus qu’ils apparaissent fréquemment dans les textes ce qui conduit à une lenteur dans le traitement et une mauvaise concentration sur les mots qui ont une signification importante.

Grâce aux stopwords anglais fournis par la bibliothèque nltk, nous comparons chaque mot des tweets de l’ensemble de données et vérifions son appartenance à la liste des stopwords. Nous le supprimons s’il y existe.

4.9. Suppression des Blancs

Nous avons implémenté la fonction ‘remove_gaps’ pour nettoyer le texte en supprimant les espaces multiples et en les remplaçant par un seul espace pour faciliter les prochaines étapes du traitement.

4.10. Tokenisation

Avant de passer aux traitements critiques des mots du tweet et pour faciliter l'analyse linguistique nous passons par la tokenisation. La tokenisation est le processus de division d'un texte en unités appelées tokens qui sont les mots du texte du tweet. Par exemple, dans la phrase "just woke up having no school is the best feeling ever", tokenisation divisera cette phrase en onze tokens: "just", "woke", "up", "having", "no", "school", "is", "the", "best", "feeling", "ever".

4.11. Radicalisation de Mots (Stemming) et Lemmatisation

La transformation des mots en leur forme radicale est essentielle pour normaliser les mots et réduire les variations causées par les différentes formes d'un même mot. Par exemple, les mots ‘running’ et ‘runs’ seront tous les deux réduits à leur racine ‘run’ mais le processus du stemming se concentre sur le préfixe et suffixe du mot pour avoir sa forme racine.

La lemmatisation, est un autre processus plus sophistiqué et plus précise qui ramène le mot en sa forme de dictionnaire (forme canonique). Par exemple : ‘better’ deviendra ‘good’.

Pour la radicalisation des mots nous avons conçu la fonction ‘Stemming’ pour réduire chaque mot d'un texte à sa racine en utilisant l'algorithme de stemming de Porter de la bibliothèque nltk.

4.12. Binarisation des labels

Comme nous adoptons une convention de classification binaire où la classe positive est représentée par 1 et la classe négative par 0, nous remplaçons les étiquettes 4 par 1 pour faciliter le processus d'analyse et de prise de décision.

Ainsi, les tweets comme montré dans la figure 3 sont bien nettoyés et stocké dans le fichier ‘sentiment140_preprocessed.csv’ comme montré dans la figure 4.

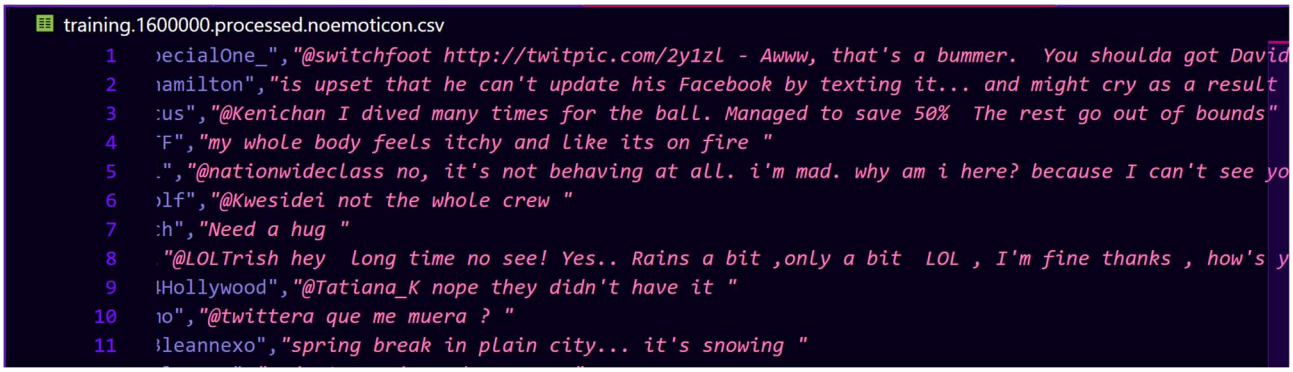

Figure 3. Tweets avant le prétraitement

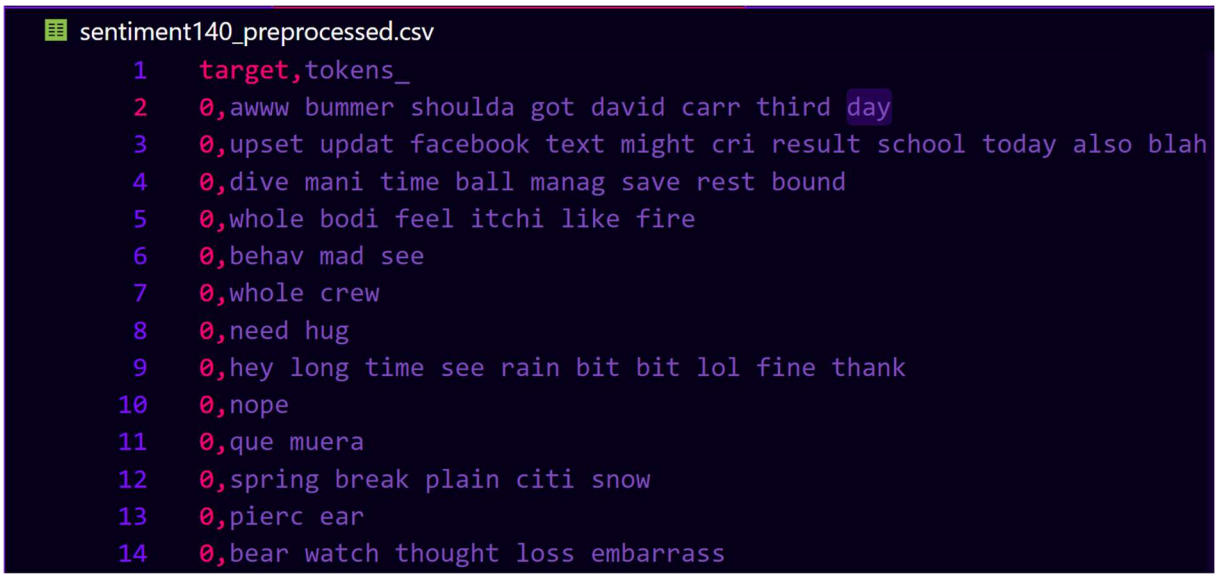

Figure 4. Tweets après le prétraitement

5. Construction du vocabulaire

Dans cette étape, notre but était de réduire la dimension des données en ne conservant que les mots les plus significatifs et les plus fréquents dans notre ensemble de données.

Pour cela, nous avons compté les occurrences des mots en parcourant chaque tweet dans le dataset prétraité, nous avons divisé chaque tweet en mots, compter son nombre d'occurrences et mettre à jour son compteur dans le dictionnaire.

Seuls les mots qui apparaissaient au moins ‘K’ fois sont conservés dans une liste de vocabulaire (limite de K = 7777).

Nous avons choisi le seuil d'occurrence minimale ‘K’ selon trois critères :

1. **Taille du vocabulaire**

- Certes, plus la valeur de K est élevée, moins de mots sont considérés pour former notre vocabulaire et plus la valeur de K est minimisée, une variété de mots, y compris les mots qui n’ont pas un sens pertinent, sont stockés dans le vocabulaire.

2. **Impact sur la représentation des données**

- Une valeur de K très élevée signifie que seuls les mots les plus fréquents seront inclus dans le vocabulaire et cela affecte l’interprétation des données par nos modèles. Or que des valeurs de K plus petites incluent une plus grande variété de mots, et ceci capture des nuances et du bruit dans nos données.

3. **Considérations de performance**

- Le calcul des fréquences de mots et la sélection des mots les plus fréquents sont coûteux en termes de temps de calcul surtout avec 1,6 Millions de tweets. Donc, choisir une valeur de K plus élevée réduit le temps d’exécution de ces opérations, mais aussi cause une perte d'informations importantes car des mots importants sont exclus.

Alors, à cause des contraintes de mémoire, nous avons opté pour un K moyen (K=7777) qui, à la fois, **satisfait les performances de notre machine** et **fournit une certaine diversité** dans le vocabulaire pour maintenir l’équilibre entre **la pertinence** et **la capacité du système**.

Enfin, nous avons trié la liste de vocabulaire par occurrence des mots (244 mots), du plus grand au plus petit et stocké le vocabulaire dans un fichier ‘vocab.txt’ et dans un autre fichier appelé ‘vocab_occ.txt’ qui contient, de plus des mots choisis, le nombre d'occurrences dans le corpus.

L’histogramme de la figure 5 montre la distribution des 20 mots les plus fréquents du vocabulaire en fonction de leur occurrence dans le dataset nettoyé.

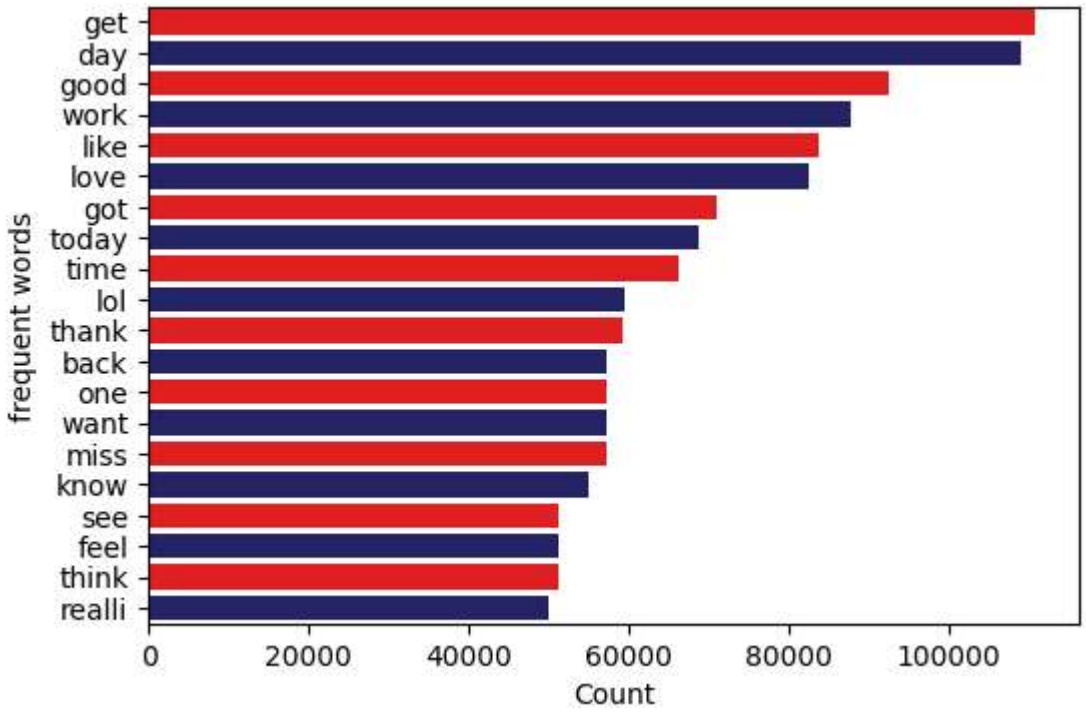

Figure 5. Histogramme d'aperçu de top 20 mots les plus fréquents du vocabulaire

- **Mappage des Mots aux Indices dans le Vocabulaire**

Dans cette étape nous avons converti les tweets en une forme numérique en représentant chaque tweet par une liste d'indices de mots pour conserver la structure et le sens du langage naturel dans les données et faciliter l'analyse et l'entraînement des modèles d'apprentissage.

Pour ceci, nous avons associé chaque mot d'un tweet prétraité à son index dans le vocabulaire à partir de l'ensemble de données de tweets. Nous avons d'abord extrait les mots du vocabulaire. Ensuite, pour chaque tweet, nous avons recherché chaque mot dans le vocabulaire. Si le mot était présent, son index était ajouté à une liste d'indices pour ce tweet ; sinon, il était ignoré.

```
Indices des mots du tweet N° 1     : [6, 1]
Indices des mots du tweet N° 89405 : [23, 32, 41, 60, 76]
```

Figure 6. Aperçu de deux tweets convertis en des indices de mots

Depuis la figure 7, Nous remarquons que pour le tweet N° 1 qui est construit des mots : [‘awww’, ‘bummer’, ‘shoulda’, **‘got**’, ‘david’, ‘carr’, ‘third’ et **‘day**’]. Les indices **6** et **1** font référence aux mots ‘**got**’ et ‘**day**’ respectivement, les autres mots ne sont pas choisis.

Pour le tweet N° 89405 qui est composé des mots : [‘realli’, ‘realli’, ‘realli’, ‘realli’, ‘realli’, ‘realli’, ‘realli’, ‘want’, ‘onlin’, ‘shop’, ‘seen’, ‘niic’, ‘stuff’, ‘card’, ‘adress’, ‘tho’, ‘poopi’]. Les indices **23**, **32**, **41**, **60**, **76** ont été repéré.

6. Extraction de Caractéristiques

Après le prétraitement, une transition des données textuelles vers des représentations numériques est essentielle pour les modèles d'apprentissage. C’est pourquoi nous procédons à **la vectorisation** en convertissant les mots alphabétiques en vecteurs numériques, ceci nous permet non seulement l'extraction de caractéristiques nécessaires mais aussi la réduction de la dimension des données (représentation binaire (ou par comptage) c’est un seul bit (ou un octet) pour chaque mot VS chaîne de caractères (plusieurs octets) pour chaque mot). Cela nous facilite l'analyse et l'application des différents algorithmes d'apprentissage (format compréhensible par les machines).

6.1. Représentation Binaire des Caractéristiques

L’idée dans la représentation binaire des caractéristiques est de traiter les tweets en détectant les informations essentielles sur la présence des mots ou non dans chaque tweet.

la caractéristique $x_i \in \{0,1\}$ d'un tweet correspond à savoir si le i-ème mot du dictionnaire apparaît dans le tweet. Autrement dit, x_i=1 si le i-ème mot est dans le tweet et x_i=0 si le i-ème mot n'est pas présent dans le tweet.

Chaque tweet est transformé en un vecteur où chaque élément définit si le mot est présent dans le tweet, la valeur est 1 ; sinon, elle est 0.

Dans notre cas, chaque tweet est représenté par un vecteur de longueur égale au nombre total de mots dans notre vocabulaire qui est de taille de 244 mots.

Pour le premier tweet par exemple seuls les mots avec les indices ‘6’ et ‘1’ sont mis à 1 les autres sont mis à 0.

Indices des mots du tweet N° 1 : [**6, 1**]

Sa représentation binaire :

0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0…,0

6.2. Représentation des Caractéristiques par Comptage

Avec cette méthode nous comptons la fréquence d'apparition des mots dans le tweet, pour l'exploiter dans l'entrainement de nos modèles d'apprentissage. Cependant, nous avons remarqué que les mots qui ont une fréquence différente de 1 et 0 sont limités. Dans cette représentation nous convertissons des données textuelles en vecteurs numériques où chaque tweet est représenté par un vecteur dont les dimensions correspondent aux mots du vocabulaire, et les valeurs correspondent au nombre d'occurrences de chaque mot dans le tweet.

La caractéristique $x_i \in \{0, . . , m\}$ d'un tweet correspond au nombre d'apparitions du i-ème mot du dictionnaire dans le tweet.

Penons pour exemple le tweet N° 89 405 :

Indices des mots du tweet N° 89 405 : [**23**, **32**, **41**, **60**, **76**]

Sa représentation par comptage :

0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0

Le mot 'realli' (indice 32) est répété 7 fois dans le tweet.

Remarque

Certains tweets, après le nettoyage des données sont devenus des tweets vides, c'est-à-dire, qu'ils sont inutiles et ne contiennent aucune information dont laquelle nous pouvons effectuer des traitements. Ils causaient des problèmes de compilation car ils retournaient un valeur

‘NAN’. Alors pour des raisons d’optimisation, nous avons procédé à les éliminer de l’ensemble de données et stocké le nouveau dataset dans ‘sentiment140_preprocessed_reduced.csv’.

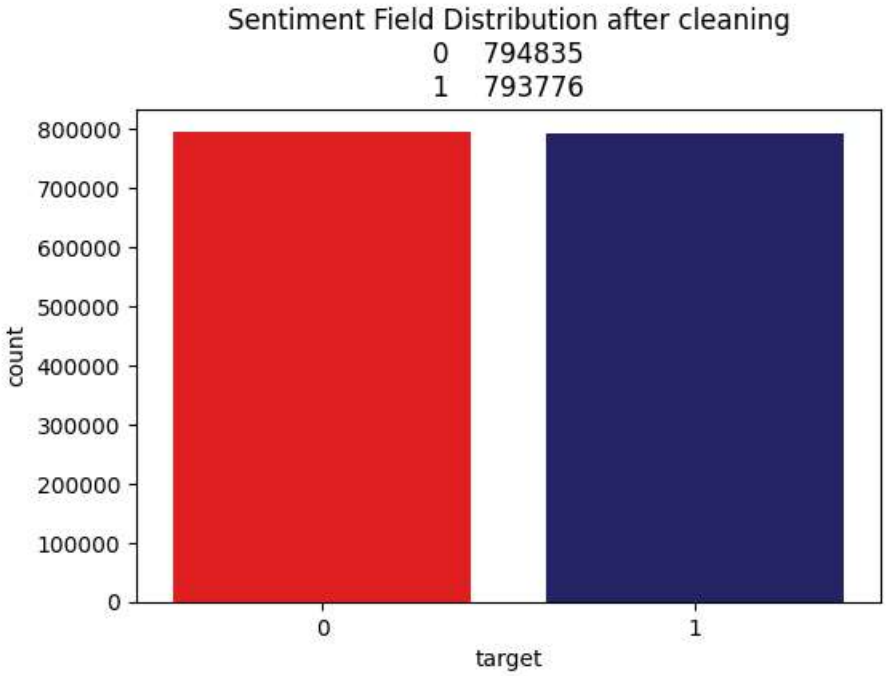

Figure 7. Distribution des sentiments après le nettoyage

La figure 8 de Distribution des sentiments après le nettoyage montre que la taille est réduite de 1 600 000 tweets à 1 588 612 lignes de tweets dont nous trouvons : 794 835 tweets négatifs et 793 776 tweets positifs.

7. Conclusion

À la fin de la partie de prétraitement et de construction du vocabulaire, ainsi que de l'extraction des caractéristiques, nous avons obtenu :

1. Un jeu de données nettoyé et prétraité, prêt à être utilisé pour l'analyse de sentiment stocké dans ‘sentiment140_preprocessed_reduced.csv’.

2. Un fichier de vocabulaire ‘vocab.txt’ contenant les mots pertinents sélectionnés pour notre modèle, basé sur leur fréquence d'apparition dans l’ensemble de données de tweets nettoyés.

3. Chaque tweet du corpus est représenté sous forme de vecteur numérique, en utilisant deux représentations : une représentation binaire qui indique la présence ou l'absence de mots dans le tweet, et une représentation par comptage qui indique le nombre d'apparitions de chaque mot dans le tweet. Chacune stockée dans un fichier ‘binary_features.csv’ et ‘count_features.csv’ respectivement.

Partie II :
Classification et Résultats

1. Introduction

Dans cette section, nous explorerons plusieurs algorithmes de classification largement utilisés pour traiter les données textuelles et évaluer leur performance. Pour ce faire, nous commencerons par diviser notre ensemble de données en ensembles d'entraînement et de test pour évaluer l'efficacité des modèles.

Ensuite, nous plongerons dans différents algorithmes de classification, notamment : KNN, Random Forest, Decision Tree, SVM, Logistic Regression, Naive Bayes, Neural Network et Deep Learning tout en interprétons leurs resultats.

2. Diviser l'Ensemble de Données

La première étape de notre analyse consiste à diviser notre ensemble de données en ensembles d'entraînement et de test pour pouvoir entraîner nos modèles sur une partie de données tout en réservant une autre partie d'évaluation de performance.

Nous avons deux représentations différentes des caractéristiques : la représentation binaire et la représentation par comptage. Pour chaque représentation, nous avons défini notre matrice des caractéristiques X et notre vecteur cible Y.

Les caractéristiques binaires et les caractéristiques par comptage ont été chargées à partir des fichiers CSV dont lesquels nous avons stockés nos vecteurs binaires et de comptage. Les cibles correspondantes ont été aussi extraites du fichier des données résultant du prétraitement.

Nous avons ensuite divisé les données en ensembles d'entraînement et de test, en prenant d'abord les deux tiers (2/3) pour l'entraînement. Mais pendant l'entraînement nous l'avons augmenté à (80%) car ceci a donné de meilleurs résultats. Les 20% qui restent ont été consacré pour le test. Cette division a été réalisée à l'aide de la fonction `train_test_split` de la bibliothèque `scikit-learn`, en fixant un état aléatoire (42) pour assurer la reproductibilité des résultats.

Les dimensions des ensembles résultants pour les deux représentations sont les suivantes :

Entraînement : X_train_count et X_train_binary (1270888, 240)

Test: X_test_count et X_test_binary (317723, 240)

Cibles entraînement: y_train_binary et y_train_count: (1270888,)

Cibles test: y_test_binary et y_test_count (317723,)

3. Matrice de confusion

Nous avons interprété les résultats avec des matrices de confusion car elle sont essentielles pour comprendre les performances d'un modèle de classification. En ce qui suit, comment nous avons interprété les différentes parties de la matrice :

Nous avons deux classes positive (P) et négative (N), notre matrice de confusion a donc la forme suivante :

	Predicted Negative	Predicted Positive
Actual Negative	TN	FP
Actual Positive	FN	TP

où :

- TN (True Negative) : Le nombre d'observations correctement classées comme négatives.

- FP (False Positive) : Le nombre d'observations incorrectement classées comme positives (faux positifs).

- FN (False Negative) : Le nombre d'observations incorrectement classées comme négatives (faux négatifs).

- TP (True Positive) : Le nombre d'observations correctement classées comme positives.

Avec la matrice de confusion nous pouvons tirer les informations suivantes :

- Rappel/Sensibilité (True Positive Rate) : C'est la capacité du modèle à trouver les exemples positifs. Elle se calcule comme TP / (TP + FN)

- Spécificité (True Negative Rate) : C'est la capacité du modèle à trouver les exemples négatifs. Elle se calcule comme TN / (TN + FP). Une spécificité élevée signifie que le modèle trouve bien les exemples négatifs.

- Précision : C'est la capacité du modèle à ne pas identifier à tort un exemple négatif comme positif. Elle se calcule comme TP / (TP + FP).

- Faux positifs (False Positives) : Le nombre d'exemples négatifs incorrectement classés comme positifs.

- Faux négatifs (False Negatives) : Le nombre d'exemples positifs incorrectement classés comme négatifs.

4. Application des Classifieurs

4.1. K-Nearest Neighbors (KNN)

C'est un algorithme d'apprentissage supervisé simple qui classe les points de données en fonction des voisins les plus proches dans l'espace des caractéristiques. En ce qui suit une explication de son fonctionnement, son application dans notre projet, ainsi que les raisons de son choix et les paramètres utilisés.

Fonctionnement

Il fonctionne en trouvant les k points de données les plus proches d'un nouveau point, puis en attribuant la classe majoritaire parmi ces voisins au nouveau point. Il est facile à implémenter mais il est aussi sensible aux valeurs aberrantes et nécessite de stocker l'ensemble de données en mémoire, ce qui est coûteux comme dans notre cas avec plus de 1.5 Millions de données. Malgré ses limitations, il reste une méthode efficace pour la classification des données, en particulier pour les ensembles de données de petite à moyenne taille.

Paramètres Clés

- K (nombre de voisins) : Ce paramètre détermine le nombre de voisins à considérer pour la classification. Une valeur plus petite de K peut rendre le modèle sensible au bruit, tandis qu'une valeur plus grande peut rendre le modèle trop général.

- Métrique de Distance : comme la distance Euclidienne qui est couramment utilisée ou encore la distance de Manhattan.

Application de KNN sur les Données Binaire et par Comptage

Dans notre projet, nous avons appliqué l'algorithme KNN à deux types de représentations de nos tweets : la représentation binaire et la représentation par comptage.

Justification des Choix des Paramètres Clés

1. Choix de la Bibliothèque :

- **scikit-learn** : est une bibliothèque bien documentée et contient des implémentations optimisées des algorithmes de machine learning. Et donc nous l'avons utilisé pour implémenter KNN.

2. Choix des Paramètres :

- K = **777** : Nous avons choisi un nombre impair k=777 voisins car après plusieurs expérimentations, c'est la valeur qui nous a donné le meilleur résultat. Pour les valeurs entre 1 et 10 elles sont considérées comme basse pour une telle taille si énorme. Les valeurs au de-là de 1000 risquent de lisser les frontières de décision et nous induire en erreur.

- Distance Euclidienne : la distance euclidienne est choisie par défaut, et comme cette mesure est simple et efficace pour les espaces de caractéristiques numériques nous l'avons choisie.

Remarque

- Nous étions dans l'obligation d'apporter des modifications à l'ensemble de l'entraînement, nous l'avons réduit à 200000 tweets pour des raisons de capacité de mémoire et de dépassement des performances de nos machines.

Résultats et Évaluation

L'algorithme KNN a été évalué sur les ensembles de test modifiés pour les deux types de représentations de caractéristiques. Les résultats que nous avons trouvés ont montré les performances suivantes :

- Avec une représentation binaire :

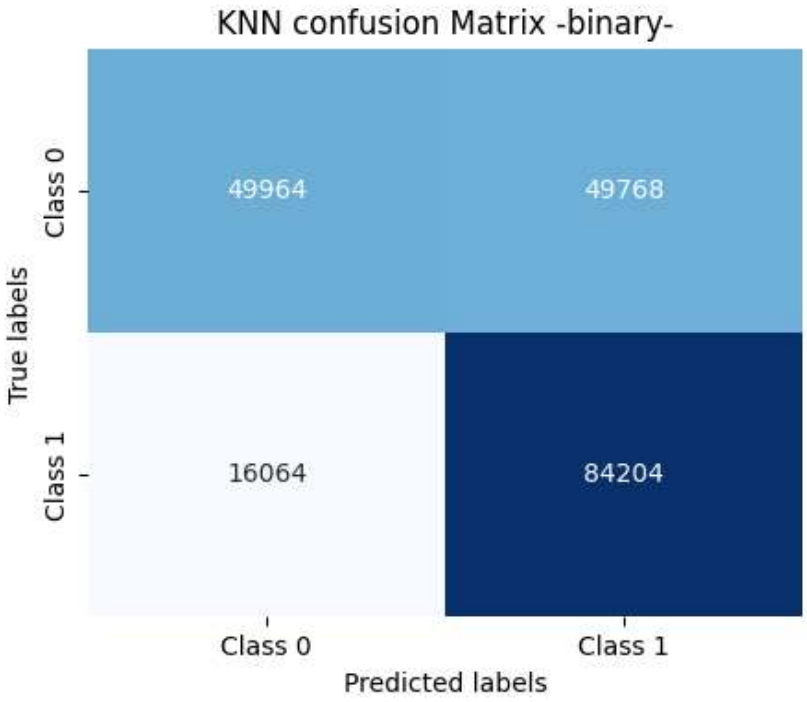

Figure 7. Matrice de confusion de la représentation binaire en KNN

Depuis la matrice de la figure7, nous pouvons conclure les résultats suivants :

	precision	recall	f1-score	support
0	0.76	0.50	0.60	99732
1	0.63	0.84	0.72	100268
accuracy			0.67	200000
macro avg	0.69	0.67	0.66	200000
weighted avg	0.69	0.67	0.66	200000

- Interprétation des résultats

Le modèle a une performance globale modérée, avec des résultats légèrement meilleurs pour la classe 1 par rapport à la classe 0 en termes de précision, de rappel et de F1-score. Cependant, le rappel pour la classe 0 est assez bas, ce qui indique que le modèle a eu du mal à identifier correctement toutes les instances de cette classe.

- Avec une représentation par comptage :

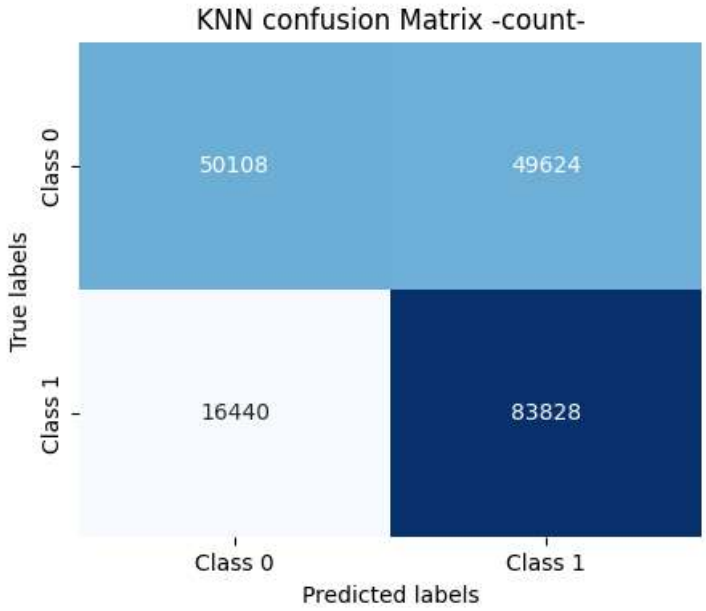

Figure 8. Matrice de confusion de la représentation par comptage en KNN

Depuis la matrice de la figure 8, nous pouvons conclure les résultats suivants :

	precision	recall	f1-score	support
0	0.75	0.50	0.60	99732
1	0.63	0.84	0.72	100268
accuracy			0.67	200000
macro avg	0.69	0.67	0.66	200000
weighted avg	0.69	0.67	0.66	200000

Encore une fois, ces résultats indiquent une performance globale modérée du modèle, avec des performances légèrement meilleures pour la classe 1 par rapport à la classe 0 en termes de précision, de rappel et de F1-score, mais le rappel pour la classe 0 reste bas.

Ces résultats nous permettent de les exploiter plus tard pour comparer l'efficacité de KNN avec les autres algorithmes de classification.

4.2. Arbre de Décision

L'arbre de décision est un algorithme utilisé pour les tâches de classification et de régression. C'est un modèle basé sur des règles qui divise récursivement l'espace des caractéristiques en sous-ensembles homogènes en fonction des valeurs des caractéristiques. Chaque nœud interne

de l'arbre représente une caractéristique et un seuil de décision, alors que chaque feuille de l'arbre représente une classe cible ou une valeur de sortie.

Fonctionnement

- L'arbre de décision commence par l'ensemble de données complet à la racine.

- À chaque nœud, il sélectionne la caractéristique et le seuil qui divisent le mieux les données selon un certain critère.

- Ce processus est répété de manière récursive pour chaque sous-ensemble jusqu'à ce qu'un critère d'arrêt soit atteint.

- Pour classer un nouvel échantillon, l'arbre de décision suit le chemin des nœuds, en appliquant les décisions de seuil jusqu'à atteindre une feuille, ce qui donne la prédiction de la classe.

Justification des Choix des Paramètres Clés

1. Choix de la Bibliothèque:

- **scikit-learn** : est une bibliothèque bien documentée et contient des implémentations optimisées des algorithmes de machine learning. Et donc nous l'avons utilisé pour implémenter l'algorithme de Decision Tree.

2. Paramètres Utilisés :

- random_state=42 : Pour assurer la reproductibilité des résultats.

- Critère de Divison : nous avons laissé le critère par défaut qui est l'indice de Gini.

Résultats et Évaluation

L'algorithme Decision Tree a été évalué sur les ensembles de test pour les deux types de représentations de caractéristiques. Les résultats que nous avons trouvés ont montré les performances suivantes :

- Avec une représentation binaire :

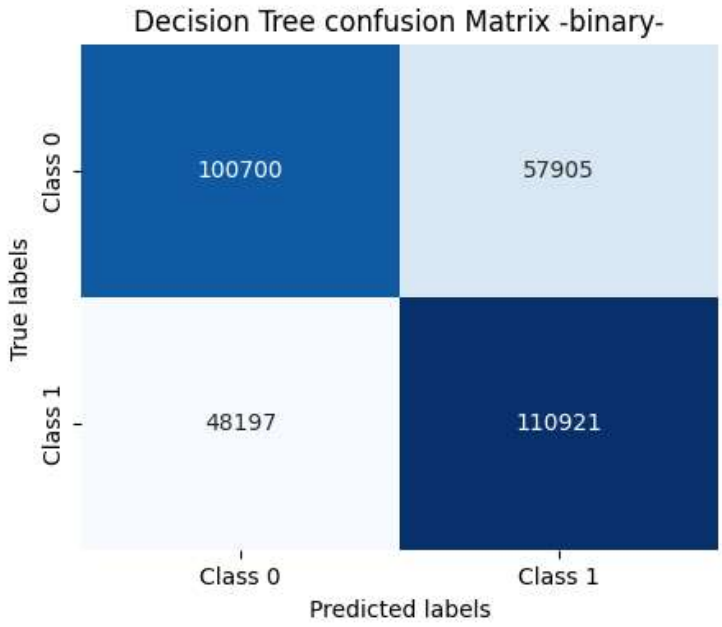

Figure 9. Matrice de confusion de la représentation binaire avec Decision Tree

Depuis cette matrice, nous pouvons conclure les résultats suivants :

	precision	recall	f1-score	support
0	0.68	0.63	0.65	158605
1	0.66	0.70	0.68	159118
accuracy			0.67	317723
macro avg	0.67	0.67	0.67	317723
weighted avg	0.67	0.67	0.67	317723

Les résultats indiquent une performance modérée du modèle, avec des performances relativement similaires entre les deux classes en termes de précision, de rappel et de F1-score. L'exactitude est également de 67%, ce qui montre que le modèle est capable de classifier les instances avec une précision raisonnable.

- Avec une représentation par comptage :

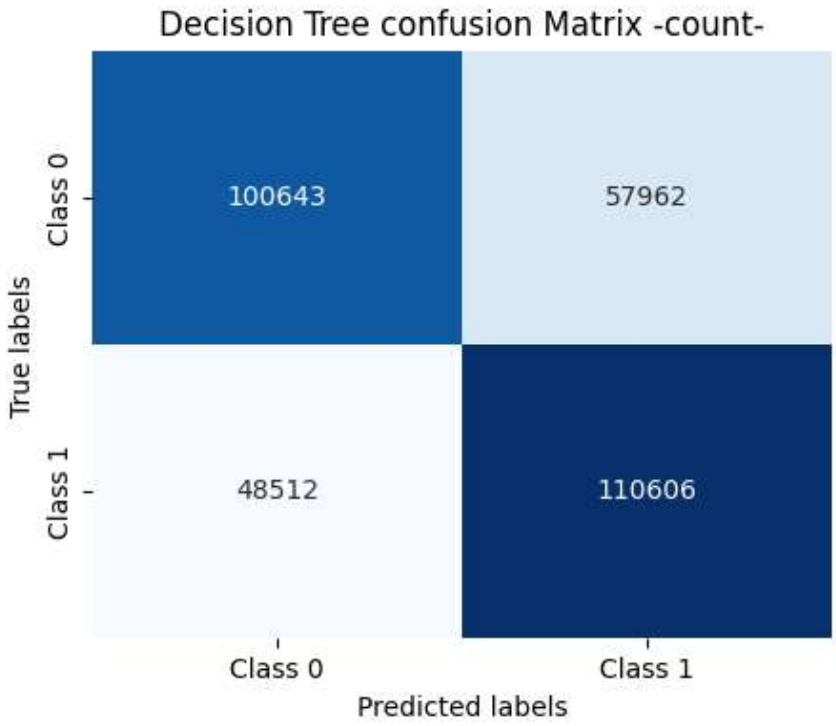

Figure 10. Matrice de confusion de la représentation par comptage avec Decision Tree

	precision	recall	f1-score	support
0	0.67	0.64	0.65	158605
1	0.66	0.69	0.68	159118
accuracy			0.67	317723
macro avg	0.67	0.67	0.66	317723
weighted avg	0.67	0.67	0.66	317723

Les résultats montrent une performance modérée et relativement équilibrée du modèle, avec des performances similaires pour les deux classes en termes de précision, de rappel et de F1-score. L'exactitude est également cohérente à 67%.

Ces résultats nous permettent de les exploiter plus tard pour comparer l'efficacité de cet algorithme avec les autres algorithmes de classification.

4.3. Random Forest

Random Forest est un ensemble d'arbres de décision qui combine les prédictions de plusieurs arbres pour améliorer les performances et réduire le sur-ajustement. Il est basé sur basé sur le principe d'ensemble learning. Chaque arbre dans la forêt est construit à partir d'un échantillon

aléatoire du jeu de données et utilise une sélection aléatoire des caractéristiques pour chaque division, ce qui permet de réduire la variance et de prévenir le surapprentissage.

Fonctionnement

- Random Forest utilise la méthode du bagging pour créer plusieurs jeux de données d'entraînement en échantillonnant avec remplacement à partir du jeu de données d'origine. Chaque arbre de décision est ensuite formé sur un de ces jeux de données.

- Lors de la construction de chaque arbre, Random Forest sélectionne de manière aléatoire un sous-ensemble des caractéristiques à chaque nœud pour déterminer la meilleure division. Cela augmente la diversité entre les arbres et contribue à la réduction de la corrélation entre eux.

- Pour une nouvelle observation, chaque arbre de la forêt donne une prédiction de classe. La classe finale prédite par la forêt est déterminée par la majorité des votes des arbres (classification) ou par la moyenne des prédictions (régression).

Justification des Choix des Paramètres Clés

1. Choix de la Bibliothèque :

- **scikit-learn** : est une bibliothèque bien documentée et contient des implémentations optimisées des algorithmes de machine learning. Et donc nous l'avons utilisé pour implémenter Random Forest aussi.

2. Paramètres Utilisés :

- n_estimators=100 : Nombre d'arbres dans la forêt. Un très grand nombre induit à une lenteur d'exécution et une petite valeur induit à des résultats non suffisants, nous avons choisi 100 parce qu'il est le plus optimal.

- random_state=42 : Pour assurer la reproductibilité des résultats.

Résultats et Évaluation

Les performances des modèles Random Forest ont été évaluées sur les ensembles de test pour les deux types de représentations de caractéristiques. Les résultats obtenus sont les suivants :

- Avec une représentation binaire :

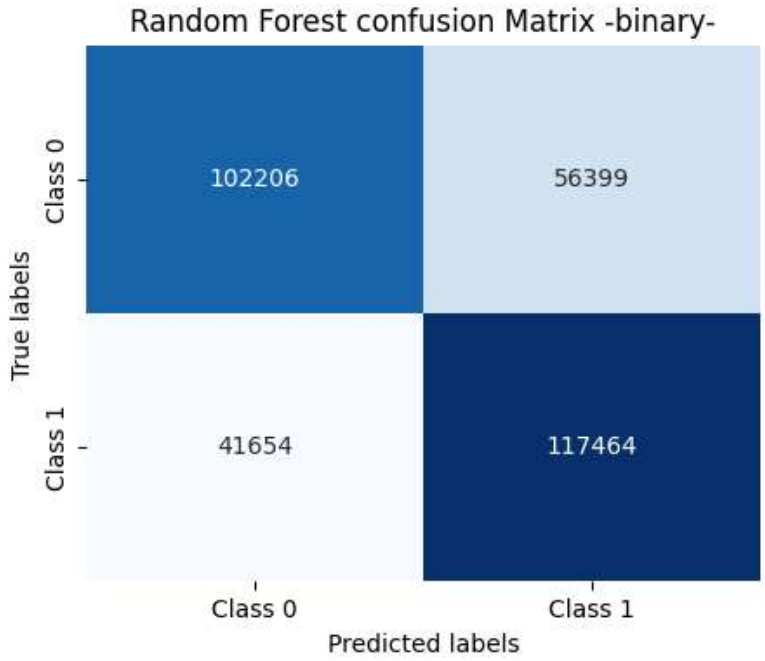

Figure 11. Matrice de confusion de la représentation binaire avec Random Forest

	precision	recall	f1-score	support
0	0.71	0.64	0.68	158605
1	0.68	0.74	0.71	159118
accuracy			0.69	317723
macro avg	0.69	0.69	0.69	317723
weighted avg	0.69	0.69	0.69	317723

Les résultats de Random Forest avec la représentation binaire montrent une amélioration dans la capacité du modèle à distinguer les deux classes, avec des performances légèrement meilleures pour la classe 1 par rapport à la classe 0. L'exactitude globale est également améliorée à 69%.

- Avec une représentation par comptage :

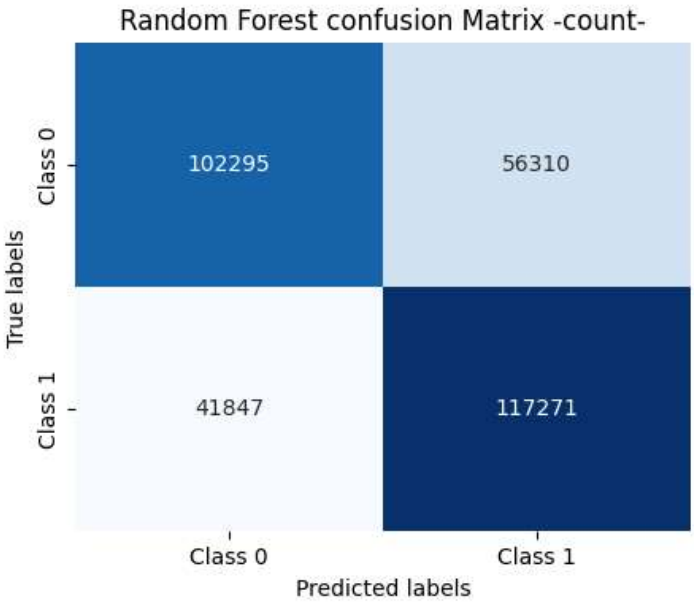

Figure 12. Matrice de confusion de la représentation par comptage avec Random Forest

	precision	recall	f1-score	support
0	0.71	0.64	0.68	158605
1	0.68	0.74	0.70	159118
accuracy			0.69	317723
macro avg	0.69	0.69	0.69	317723
weighted avg	0.69	0.69	0.69	317723

Encore une fois, les résultats montrent une performance modérée et relativement équilibrée du modèle, avec des performances similaires pour les deux classes en termes de précision, de rappel et de F1-score. L'exactitude est également cohérente à 69%.

Ces résultats permettent de comparer l'efficacité de Random Forest avec d'autres algorithmes de classification appliqués à notre jeu de données de tweets.

4.4. SVM (Support Vector Machine)

SVM (Support Vector Machine) est un algorithme d'apprentissage supervisé qui trouve un hyperplan optimal pour séparer les classes dans un espace des caractéristiques de grande dimension.

Fonctionnement

- Le SVM cherche à maximiser la marge entre les classes, c'est-à-dire la distance entre l'hyperplan séparateur et les points de données les plus proches de chaque classe.

- Dans un problème de classification binaire, un hyperplan est une frontière de décision linéaire qui sépare deux classes. Pour les problèmes non linéaires, SVM utilise des noyaux pour transformer les données dans un espace de caractéristiques plus élevé où un hyperplan linéaire peut être utilisé pour la séparation.

- Les noyaux couramment utilisés incluent le noyau linéaire, polynomial, et radial basis function (RBF). Les noyaux permettent à SVM de résoudre les problèmes de classification non linéaire.

Justification des Choix des Paramètres Clés

1. Choix de la Bibliothèque :

- **scikit-learn** : c'est une bibliothèque bien documentée, contient des implémentations optimisées des algorithmes de machine learning et des capacités de manipulation de noyaux variés. Alors nous l'avons utilisé pour implémenter SVM.

2. Paramètres Utilisés :

- kernel='linear' : Nous avons choisi le noyau linéaire pour simplifier l'interprétation des résultats.

- random_state=42 : Pour assurer la reproductibilité des résultats.

Remarque

- Comme dans le cas du traitement avec KNN, nous nous sommes retrouvées dans l'obligation d'apporter des modifications à l'ensemble de l'entraînement, nous l'avons réduit à 50000 tweets pour des raisons de capacité de mémoire et de dépassement des performances de nos machines.

Résultats et Évaluation

L'algorithme de la régression logistique a été évalué sur les ensembles de test pour les deux types de représentations de caractéristiques. Les résultats que nous avons trouvés ont montré les performances suivantes :

- Avec une représentation binaire :

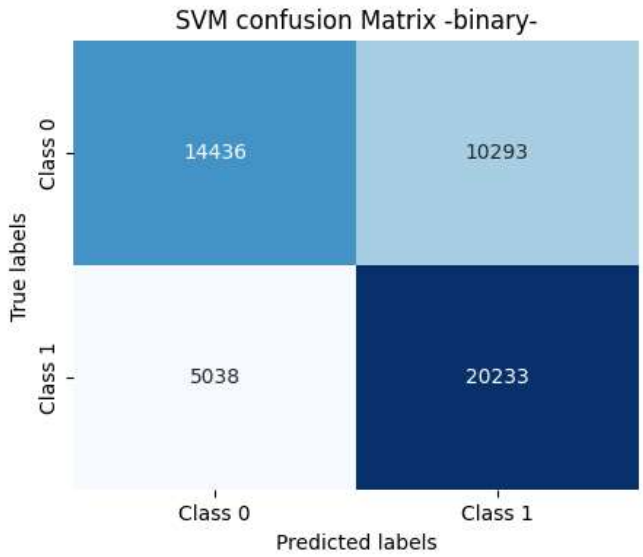

Figure 13. Matrice de confusion de la représentation binaire avec SVM

	precision	recall	f1-score	support
0	0.74	0.58	0.65	24729
1	0.66	0.80	0.73	25271
accuracy			0.69	50000
macro avg	0.70	0.69	0.69	50000
weighted avg	0.70	0.69	0.69	50000

le modèle a mieux identifié les instances de la classe 1 par rapport à la classe 0, avec une précision et un F1-score globalement modérés. Il serait bénéfique de chercher à améliorer le rappel pour la classe 0 sans trop sacrifier la précision pour améliorer la performance globale du modèle.

- Avec une représentation par comptage :

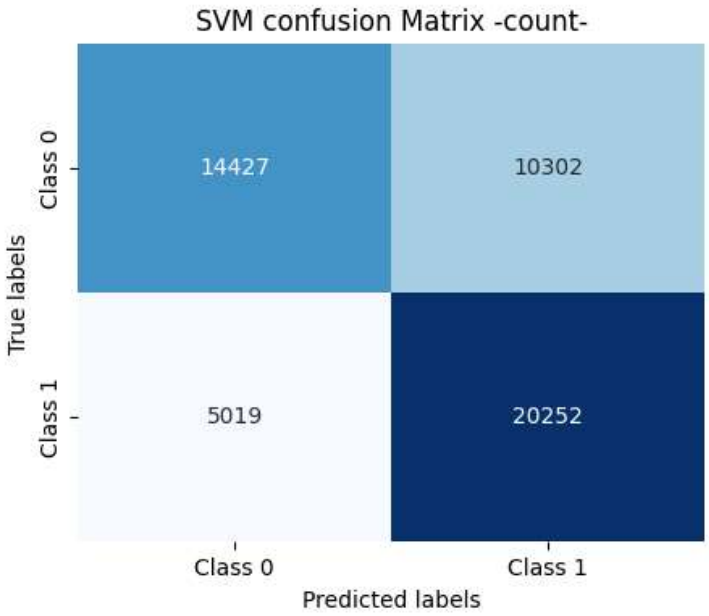

Figure 14. Matrice de confusion de la représentation par comptage avec SVM

	precision	recall	f1-score	support
0	0.74	0.58	0.65	24729
1	0.66	0.80	0.73	25271
accuracy			0.69	50000
macro avg	0.70	0.69	0.69	50000
weighted avg	0.70	0.69	0.69	50000

Les résultats montrent que le modèle de classification binaire a une performance modérée avec une exactitude globale de 69%. La classe 0 présente une précision de 74%, un rappel de 58% et un F1-score de 0.65, indiquant que le modèle est assez précis mais pas autant. Pour la classe 1, la précision est de 66%, le rappel de 80% et le F1-score de 0.73 montrent que le modèle détecte bien les instances de cette classe, mais avec plus de faux positifs.

Ces résultats nous permettent de les exploiter plus tard pour comparer l'efficacité de cet algorithme avec les autres algorithmes de classification.

4.5. Régression Logistique

La régression logistique est un algorithme de classification supervisé utilisé pour prédire la probabilité qu'une observation appartienne à l'une des deux classes possibles.

Fonctionnement

- La régression logistique utilise la fonction sigmoïde pour transformer la sortie d'une combinaison linéaire de caractéristiques en une probabilité comprise entre 0 et 1.

- La fonction sigmoïde est définie comme suit :

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- Une fois les paramètres estimés, la régression logistique prédit la probabilité qu'une observation appartienne à la classe positive. Si cette probabilité dépasse un certain seuil (généralement 0.5), l'observation est classée dans la classe positive.

Justification des Choix des Paramètres Clés

Nous n'avons pas spécifié de paramètres particuliers, car par défaut, la régularisation L2 (Ridge) est appliquée qui aide à prévenir le surapprentissage en ajoutant une pénalité sur la taille des coefficients.

Solver 'lbfgs' : C'est l'algorithme d'optimisation par défaut. Il est efficace et souvent suffisant pour des tâches courantes de classification.

Paramètre C (inverse de la régularisation) : Par défaut, C=1.0. Cela signifie que la régularisation est appliquée avec une force modérée.

Maximisation de la vraisemblance : La régression logistique maximise la vraisemblance de la distribution observée des classes.

Résultats et Évaluation

L'algorithme de la régression logistique a été évalué sur les ensembles de test pour les deux types de représentations de caractéristiques. Les résultats que nous avons trouvés ont montré les performances suivantes :

- Avec une représentation binaire :

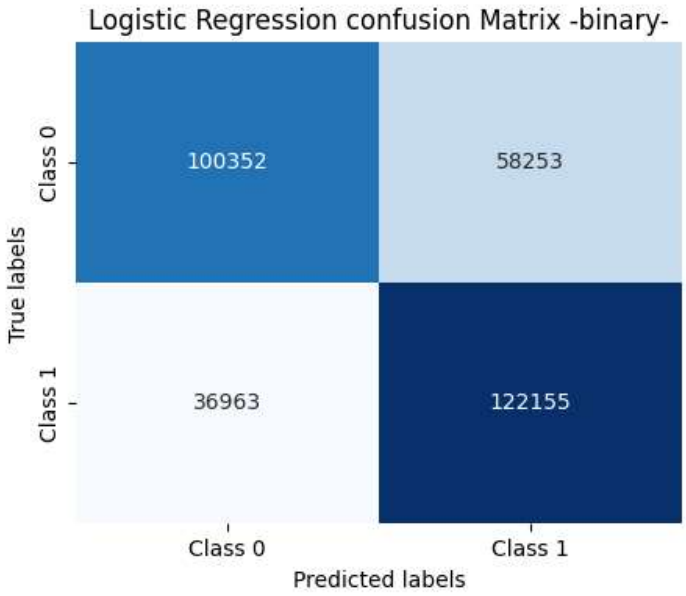

Figure 15. Matrice de confusion de la représentation binaire avec Logistic Regresison

	precision	recall	f1-score	support
0	0.73	0.63	0.68	158605
1	0.68	0.77	0.72	159118
accuracy			0.70	317723
macro avg	0.70	0.70	0.70	317723
weighted avg	0.70	0.70	0.70	317723

Les résultats de la régression logistique en représentation binaire montrent une performance globale modérée avec une exactitude de 70%. La classe 0 a une précision de 73%, un rappel de 63%, et un F1-score de 0.68, ce qui indique que le modèle est assez précis mais manque une partie notable des cas réels de cette classe. La classe 1 présente une précision de 68%, un rappel de 77%, et un F1-score de 0.72 ce qui montre que le modèle identifie bien les instances de cette classe mais avec un certain nombre de faux positifs.

- Avec une représentation par comptage :

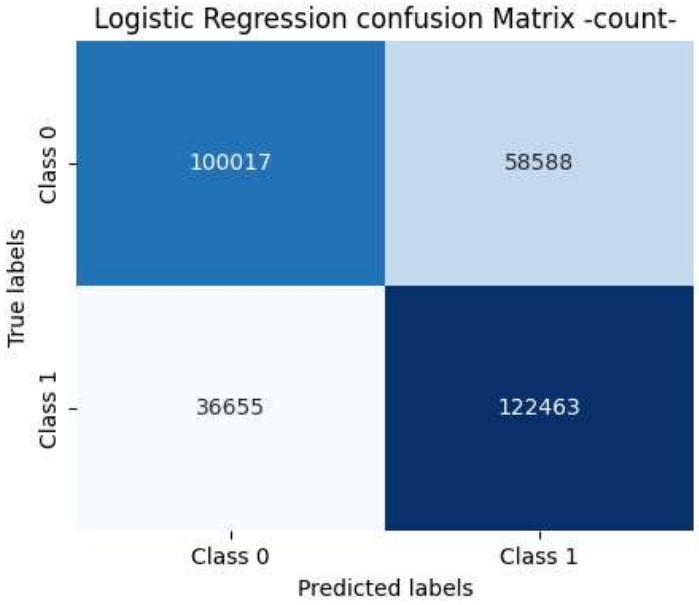

Figure 16. Matrice de confusion de la représentation par comptage avec Logistic Regresison

	precision	recall	f1-score	support
0	0.73	0.63	0.68	158605
1	0.68	0.77	0.72	159118
accuracy			0.70	317723
macro avg	0.70	0.70	0.70	317723
weighted avg	0.70	0.70	0.70	317723

Les résultats de la régression logistique par comptage montrent une performance globale modérée avec une exactitude de 70%. Pour la classe 0, le modèle obtient une précision de 73%, un rappel de 63%, et un F1-score de 0.68, indiquant une bonne précision mais une capacité limitée à capturer tous les cas réels de cette classe. Pour la classe 1, la précision est de 68%, le rappel de 77%, et le F1-score de 0.72, ce qui montre une meilleure détection des cas réels de cette classe malgré la présence de quelques faux positifs.

Ces résultats nous permettent de les exploiter plus tard pour comparer l'efficacité de cet algorithme avec les autres algorithmes de classification.

4.6. Naive Bayes

Naive Bayes est un ensemble d'algorithmes de classification basés sur le théorème de Bayes avec une hypothèse d'indépendance naïve entre les caractéristiques. Les classificateurs Naive Bayes ont prouvé leur efficacité dans de nombreux domaines de classification.

Fonctionnement

- Son fonctionnement repose sur l'hypothèse de l'indépendance conditionnelle des caractéristiques, ce qui simplifie le calcul des probabilités.

- L'algorithme commence par estimer les probabilités a priori de chaque classe, puis calcule les probabilités conditionnelles des caractéristiques pour ces dernières.

- Enfin, il combine ces probabilités pour estimer la classe la plus probable pour une instance donnée. Bien que simpliste dans son principe.

Justification des Choix des Paramètres Clés

1. Choix de la Bibliothèque :

- **scikit-learn** : Nous l'avons utilisé pour l'implémentation des modèles Naive Bayes en raison de sa facilité d'utilisation et de ses outils de modélisation performants.

2. Types de Naive Bayes :

- **BernoulliNB** car il est conçu pour les caractéristiques binaires et les autres comme GaussianNB ou MultinomialNB sont conçus pour des fonctionnalisées continues ou discrètes non binaires.

Résultats et Évaluation

L'algorithme Naive Bayes a été évalué sur les ensembles de test pour les deux types de représentations de caractéristiques. Les résultats que nous avons trouvés ont montré les performances suivantes :

- Avec une représentation binaire :

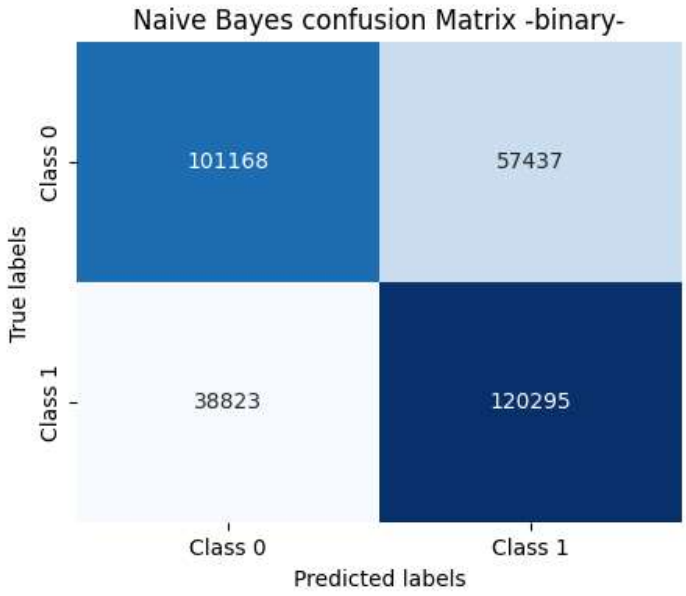

Figure 17. Matrice de confusion de la représentation binaire avec Naïve Bayes

	precision	recall	f1-score	support
0	0.72	0.64	0.68	158605
1	0.68	0.76	0.71	159118
accuracy			0.70	317723
macro avg	0.70	0.70	0.70	317723
weighted avg	0.70	0.70	0.70	317723

Les résultats de la classification binaire avec Naive Bayes utilisant une représentation binaire montrent une performance globale modérée avec une exactitude de 70%. Pour la classe 0, le modèle obtient une précision de 72%, un rappel de 64%, et un F1-score de 0.68, ce qui indique une bonne précision mais une capacité limitée à capturer tous les cas réels de cette classe. Pour la classe 1, la précision est de 68%, le rappel de 76%, et le F1-score de 0.71, ce qui montre une meilleure détection des cas réels de cette classe malgré quelques faux positifs.

- Avec une représentation par comptage :

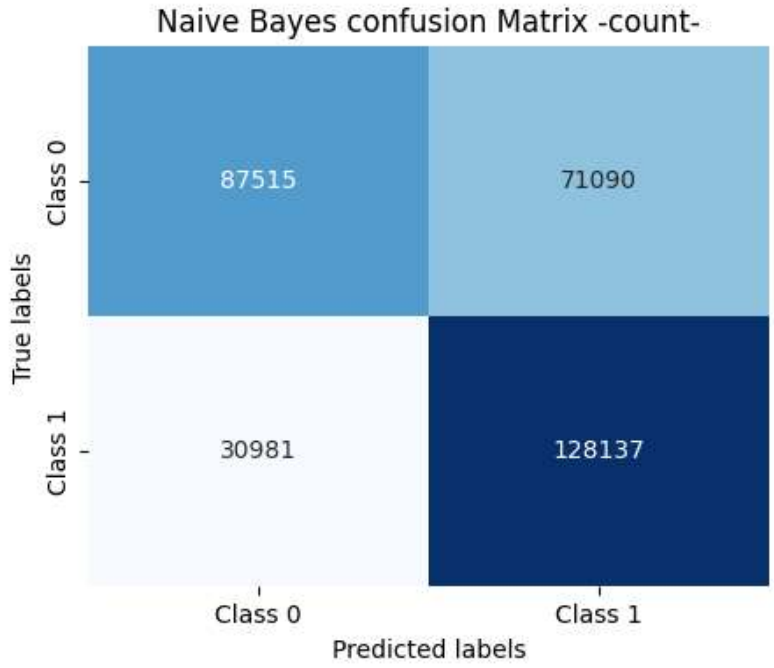

Figure 18. Matrice de confusion de la représentation par comptage avec Naïve Bayes

	precision	recall	f1-score	support
0	0.74	0.55	0.63	158605
1	0.64	0.81	0.72	159118
accuracy			0.68	317723
macro avg	0.69	0.68	0.67	317723
weighted avg	0.69	0.68	0.67	317723

Les résultats de la classification binaire avec Naive Bayes utilisant une représentation par comptage montrent une performance globale modérée avec une exactitude de 68%. Pour la classe 0, le modèle obtient une précision de 74%, un rappel de 55%, et un F1-score de 0.63, ce qui indique une bonne précision mais une capacité limitée à capturer tous les cas réels de cette classe. Pour la classe 1, la précision est de 64%, le rappel de 81%, et le F1-score de 0.72, montrant une meilleure détection des cas réels de cette classe malgré quelques faux positifs.

Ces résultats nous permettent de les exploiter plus tard pour comparer l'efficacité de cet algorithme avec les autres algorithmes de classification.

4.7. Réseau de Neurones

Les réseaux de neurones sont des modèles d'apprentissage profond inspirés du fonctionnement du cerveau humain qui utilise des couches de neurones artificiels pour apprendre des motifs dans les données. Ils sont composés de couches de neurones artificiels qui sont interconnectés.

Fonctionnement

Chaque connexion de couches de neurones artificiels possède un poids ajusté lors de l'entraînement du réseau. Dans le contexte de la classification binaire, le réseau de neurones utilise des couches denses (fully connected : tous connectées) où chaque neurone de la couche précédente est connecté à chaque neurone de la couche suivante.

Le modèle prend en entrée les caractéristiques des tweets, passe ces caractéristiques à travers plusieurs couches non linéaires, et génère une sortie qui représente la probabilité que le tweet appartienne à une classe spécifique (par exemple, tweet positif ou négatif).

Justification du Choix des Paramètres et des Bibliothèques

Dense Layers et Neurons (128, 64):

128 et 64 Neurones: Notre choix de 128 et 64 neurones pour les deux couches denses est basé sur des pratiques courantes où des puissances de 2 sont utilisées pour la taille des couches, permettant ainsi une bonne capacité de modélisation.

Activation ReLU: Nous avons utilisé la fonction d'activation ReLU (Rectified Linear Unit) pour introduire la non-linéarité, permettant au modèle de capturer des relations complexes dans les données.

0.5 Dropout Rate: pour prévenir le surapprentissage (overfitting) en désactivant aléatoirement 50% des neurones durant l'entraînement. Cela force le réseau à apprendre des représentations plus robustes.

Binary Crossentropy: Nous avons utilisé la perte binaire crossentropy pour les tâches de classification binaire, mesurant la différence entre les prédictions du modèle et les véritables étiquettes.

Adam Optimizer: Adam est un optimiseur adaptatif bien connu pour sa rapidité de convergence et sa capacité à gérer les grands espaces de paramètres efficacement, c’est pour cela que nous l’avions choisi.

En ce qui concerne les bibliothèques utilisées, nous avons choisi :

Keras: une API de haut niveau pour la construction et l'entraînement de modèles de réseaux de neurones. Elle est choisie pour sa simplicité et sa facilité d'utilisation, ainsi que pour son intégration avec TensorFlow, qui offre des capacités puissantes pour le calcul de réseau de neurones.

Sans oublier la bibliothèque numpy, utilisée pour convertir les données en tableaux de format approprié pour le traitement par Keras.

Résultats et Évaluation

Les réseaux de neurones ont été évalué sur les ensembles de test pour les deux types de représentations de caractéristiques. Les résultats que nous avons trouvés ont montré les performances suivantes :

- Avec une représentation binaire :

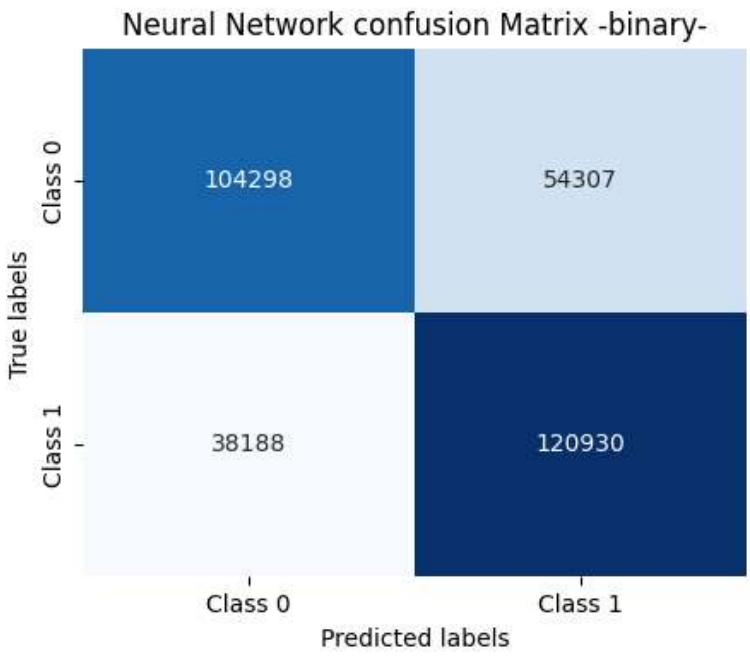

Figure 19. Matrice de confusion de la représentation binaire avec les réseaux de neurones

	precision	recall	f1-score	support
0	0.73	0.66	0.69	158605
1	0.69	0.76	0.72	159118
accuracy			0.71	317723
macro avg	0.71	0.71	0.71	317723

weighted avg	0.71	0.71	0.71	317723

Les résultats de la classification binaire avec un réseau de neurones utilisant une représentation binaire montrent une performance globalement solide avec une exactitude de 71%. Pour la classe 0, le modèle atteint une précision de 73%, un rappel de 66%, et un F1-score de 0.69, indiquant une bonne précision mais une capacité modérée à capturer tous les cas réels de cette classe. Pour la classe 1, la précision est de 69%, le rappel de 76%, et le F1-score de 0.72, ce qui montre une meilleure détection des cas réels de cette classe, bien que quelques faux positifs.

- Avec une représentation par comptage :

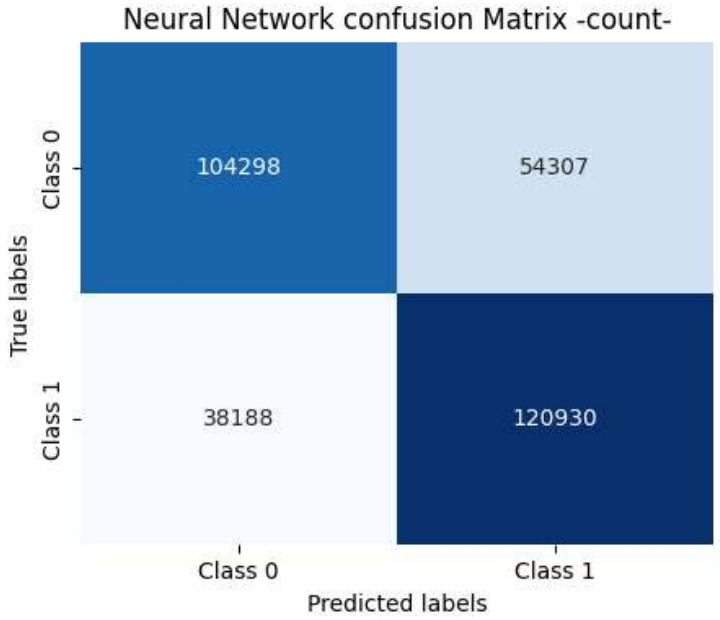

Figure 20. Matrice de confusion de la représentation par comptage avec les RNN

	precision	recall	f1-score	support
0	0.74	0.63	0.68	158605
1	0.68	0.78	0.73	159118
accuracy			0.71	317723
macro avg	0.71	0.71	0.71	317723
weighted avg	0.71	0.71	0.71	317723

Les résultats de la classification binaire avec des réseaux de neurones récurrents (RNN) utilisant une représentation par comptage montrent aussi une performance globalement solide avec une exactitude de 71%. Pour la classe 0, le modèle obtient une précision de 74%, un rappel de 63%, et un F1-score de 0.68, indiquant une bonne précision mais une capacité modérée à capturer tous les cas réels de cette classe. Pour la classe 1, la précision est de 68%, le rappel de 78%, et

le F1-score de 0.73, ce qui montre une meilleure détection des cas réels de cette classe malgré quelques faux positifs.

Ces résultats nous permettent de les exploiter plus tard pour comparer l'efficacité des réseaux de neurones avec les autres algorithmes de classification.

4.8. Deep Learning

Le deep learning est une sous-catégorie du machine learning qui utilise des réseaux de neurones artificiels avec de multiples couches, c'est un réseau de neurones profond. Dans le but de modéliser des représentations abstraites et complexes des données. Ces modèles sont particulièrement efficaces pour traiter des tâches comme la reconnaissance d'images, le traitement du langage naturel (comme dans notre cas), etc.

Fonctionnement

- Un réseau de neurones profond comporte une couche d'entrée, plusieurs couches cachées, et une couche de sortie.

- Chaque neurone dans une couche est connecté à tous les neurones de la couche suivante, formant une structure de réseau dense.

Propagation Avant (Forward Propagation):

- Les données d'entrée traversent le réseau couche par couche jusqu'à atteindre la couche de sortie.

- Chaque neurone applique une fonction d'activation (par exemple : Sigmoïde) à la somme pondérée de ses entrées.

Rétropropagation (Backpropagation) :

- L'erreur entre les prédictions et les valeurs réelles est calculée.

- Cette erreur est propagée en arrière à travers le réseau pour ajuster les poids des connexions afin de minimiser l'erreur globale.

Optimisation :

- Des algorithmes comme la descente de gradient stochastique (SGD) ou Adam sont utilisés pour mettre à jour les poids et optimiser le modèle.

Résultats et Évaluation

Le Deep Learning a été évalué sur les ensembles de test pour les deux types de représentations de caractéristiques. Les résultats que nous avons trouvés ont montré les performances suivantes :

- Avec une représentation binaire :

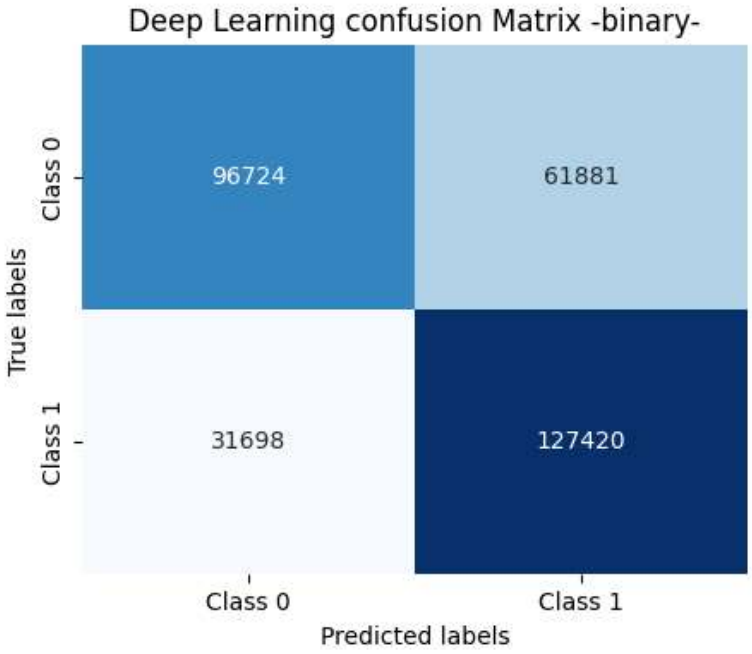

Figure 21. Matrice de confusion de la représentation binaire avec Deep Learning

	precision	recall	f1-score	support
0	0.75	0.61	0.67	158605
1	0.67	0.80	0.73	159118
accuracy			0.71	317723
macro avg	0.71	0.71	0.70	317723
weighted avg	0.71	0.71	0.70	317723

Pour la classe 0 :

La précision est de 0,75, ce qui signifie que sur tous les exemples classés comme appartenant à la classe 0, 75 % d'entre eux sont réellement de la classe 0.

Le rappel (recall) est de 0,61, ce qui signifie que sur tous les exemples réellement de la classe 0, le modèle a réussi à en identifier 61 %.

Le f1-score est de 0,67, une moyenne harmonique de la précision et du rappel pour la classe 0.

Cela montre que le modèle a une précision relativement élevée pour la classe 0, mais qu'il a tendance à manquer certains exemples qui sont réellement de cette classe (rappel plus faible).

Pour la classe 1 :

La précision est de 0,67, ce qui signifie que sur tous les exemples classés comme appartenant à la classe 1, 67 % d'entre eux sont réellement de la classe 1.

Le rappel est de 0,80, ce qui signifie que sur tous les exemples réellement de la classe 1, le modèle a réussi à en identifier 80 %.

Le f1-score est de 0,73 pour la classe 1.

Ces résultats nous permettent de les exploiter plus tard pour comparer l'efficacité des réseaux de neurones avec les autres algorithmes de classification.

Partie III :
Comparaison des Résultats & Analyse

1. Introduction

Dans cette section, nous comparons les classifieurs implémentés en termes de différents modèles d'évaluations, avec les approches classiques elles-mêmes et avec celles de l'apprentissage profond, en mentionnant les forces et les faiblesses de chaque méthode. Enfin, nous analysons et discutons les résultats.

2. Modèles d'Evaluations

2.1. Courbe ROC

Nous avons généré deux courbes ROC pour les deux représentations de données, pour cela nous avons :

- Calculé les probabilités prédictives de chaque approche pour les utiliser par la suite dans la génération de la courbe ROC ;
- Calculé les taux de faux positifs (fpr) et des taux de vrais positifs (tpr) avec roc_curve ;
- Calculé l'aire sous la courbe (AUC) avec auc ;
- Et tracé la courbe ROC avec matplotlib.

Les résultats de ce traitement ont montré les deux courbes suivantes :

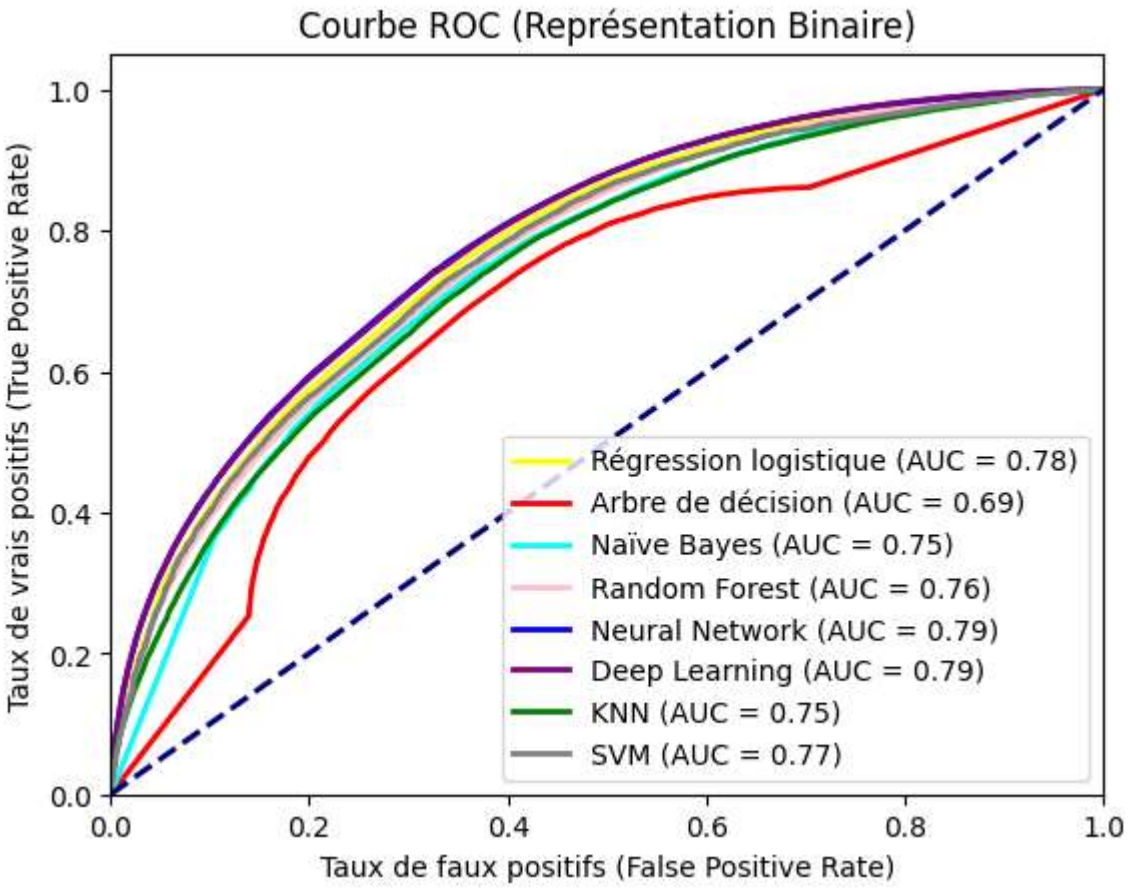

Figure 22. Courbe ROC de la représentation binaire

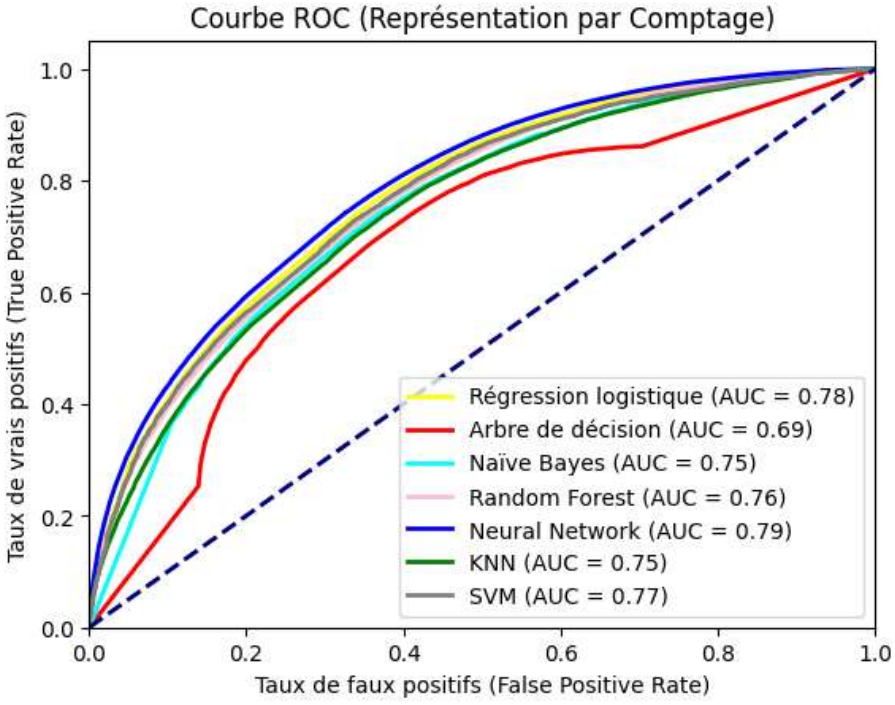

Figure 23. Courbe ROC de la représentation par comptage

Représentation Binaire

Les valeurs AUC pour chaque algorithme sont les suivantes classées du meilleur au pire:

Deep Learning (Deep Learning): AUC = 0.79

Réseau de Neurones (Neural Network): AUC = 0,79

Régression logistique (Logistic Regression): AUC = 0,78

Naïve Bayes: AUC = 0,77

Forêt Aléatoire (Random Forest): AUC = 0,76

Arbre de décision (Arbre de décision): AUC = 0,69

Apprentissage profond (Deep Learning): AUC = 0,75

Le Deep learning et le réseau de Neurones ont la valeur AUC la plus élevée, ce qui en fait les algorithmes le plus performants dans ce contexte et dans cette représentation, suivi de la Régression Logistique, Naïve Bayes, Random Forest et l'Arbre de Décision. Il est toutefois important de considérer que les Algorithmes KNN et SVM n'étaient pas entraînés sur le même dataset mais d'autres qui sont relativement réduits.

Représentation par comptage

Les valeurs AUC obtenus sont les suivantes :

Régression logistique (AUC = 0.78)

Arbre de décision (AUC = 0.69)

Naïve Bayes (AUC = 0.75)

Forêt aléatoire (AUC = 0.76)

Réseau de neurones (AUC = 0.79)

Apprentissage profond (AUC = 0.79)

KNN (AUC = 0.75)

SVM (AUC = 0.77)

Ces algorithmes diffèrent dans leur taux de faux positifs (FPR). Un FPR plus faible est généralement plus souhaitable, car cela signifie que le modèle fait moins d'erreurs en prévoyant les instances négatives.

En termes d'AUC, le réseau de neurones et l'apprentissage profond se classent au premier rang avec un AUC de 0.79. Ils sont suivis de près par la régression logistique, la forêt aléatoire, Naïve Bayes, SVM et KNN, qui ont tous un AUC entre 0.75 et 0.78. L'arbre de décision se classe au dernier rang avec un AUC de 0.69.

3. Comparaison des Classifieurs Classiques

La régression logistique a démontré une performance équilibrée, avec une précision, un rappel et un score F1 assez élevés, bien que légèrement inférieurs à ceux du Random Forest et du SVM. Mais, sa courbe ROC a montré une performance globalement bonne.

Les classifieurs basés sur les arbres de décision : Decision Tree et le Random Forest, ont montré des performances similaires en termes d'exactitude, de rappel et de précision. Mais, le Random Forest a légèrement surpassé le Decision Tree en termes de score F1 et d'AUC ROC, ce qui suggère une meilleure capacité à généraliser sur des données non vues.

Le SVM a produit les performances les plus élevées en termes d'exactitude, de rappel, de précision et de score F1. Sa courbe ROC a également montré une performance remarquable, indiquant une excellente capacité à discriminer les classes.

4. Comparaison Entre les Approches Classiques et Celles de l'Apprentissage Profond

Les approches classiques de machine learning sont préférés lorsque l'interprétabilité du modèle est critique, lorsque les données sont relativement petites et lorsque des caractéristiques spécifiques sont bien comprises par des experts en domaine. En revanche, l'apprentissage profond est souvent utilisé lorsque les données sont de grande taille et complexes, et lorsque des performances élevées sont nécessaires, même si cela se fait au détriment de l'interprétabilité du modèle.

5. Analyse et Discussion des Résultats

Tous les classifieurs ont montré des performances proches et respectables. Nous concluons donc que le choix de KNN et SVM était mauvais pour une classification qui se base sur un dataset si énorme car, tout en étant conscientes du fait, la réduction du dataset influence le résultat du knn, surtout qu'il repose sur le principe de calcul de chacune des instances dans le dataset et la comparer avec les k plus proches voisines mais nous n'avions pas un autre choix pour tester leur efficacité dans ce contexte. Qu'il n'y a pas grande différence entre la représentation binaire et la représentation par comptage, ceci est dû à la faible différence entre les deux dataset résultants de ces deux représentations.

Conclusion

Ce projet nous a permis d'acquérir une meilleure maîtrise des différents algorithmes d'apprentissage, ainsi que des compétences pratiques telles que l'extraction de résultats à partir de courbes ROC, la manipulation de grands ensembles de données et l'exploration du domaine du deep learning.

Dans ce projet, nous avons fait une analyse approfondie des techniques de classification en utilisant à la fois des algorithmes d'apprentissage classiques allant au deep learning.

Nous avons exploré plusieurs méthodes, y compris la régression logistique, les k plus proches voisins, Naive Bayes, Random Forest, l'arbre de décision ainsi que les réseaux de neurones, en évaluant leur performance à l'aide de métriques telles que la précision, le rappel et le F1-score.

En analysant les résultats obtenus, nous avons pu comparer les forces et les faiblesses de chaque méthode, montrant les cas où les modèles d'apprentissage comme le réseau de neurones et le deep learning ont surperformé les approches classiques dans la représentation binaire, ainsi que les situations où les classifieurs classiques ont donné des résultats compétitifs dans la représentation par comptage.

En fin de compte, cette expérience du projet et des différents travaux pratiques réalisés durant le semestre nous a fourni une base solide pour comprendre et appliquer efficacement les techniques de classification dans divers contextes.