Classification de Texte Binaire avec des Algorithmes de Machine Learning

Amine OUAKIB[1] *

[1] UDEM,Université de Montréal, Montréal, Canada
[2]DIRO,Département d'informatique et de recherche opérationnelle,UDEM,Université de Montréal, Montréal, Canada.

Introduction

Cette compétition Kaggle propose un défi de classification binaire de texte, une tâche classique en traitement du langage naturel (NLP). Le but est de concevoir un algorithme de machine learning qui, à partir de vecteurs de comptage de termes représentant les documents, pourra prédire la catégorie (0 ou 1) des nouveaux documents.

L'enjeu est de développer un modèle précis et capable de généraliser, en évitant le surajustement aux données d'entraînement. La performance est mesurée par le F1-score macro, qui assure un équilibre entre précision et rappel pour chaque classe, permettant ainsi de traiter équitablement les déséquilibres de classe.

Méthodologie de Classification de Texte Binaire

Feature Design

Pour le Feature Design, il est important de noter que les données fournies dans le cadre de cette compétition sont déjà bien structurées et prêtes à l'emploi sous la forme de vecteurs de comptage de termes (bag-of-words). Cette représentation est déjà optimale pour une tâche de classification binaire de texte, car elle encode efficacement chaque document en fonction de la fréquence des termes, ce qui est particulièrement adapté aux modèles de machine learning supervisés.

Division de l'ensemble de données

La division des données est une étape cruciale dans les méthodologies de machine learning et de deep learning. Elle consiste à diviser le jeu de données en deux ensembles. Dans les tâches de classification, le jeu de données est divisé en un ensemble d'entraînement (80 pourcents des données) et un ensemble de test (les 20 pourcents restants). La performance du modèle et sa capacité à se généraliser à des données inconnues peuvent être évaluées grâce à cette division.

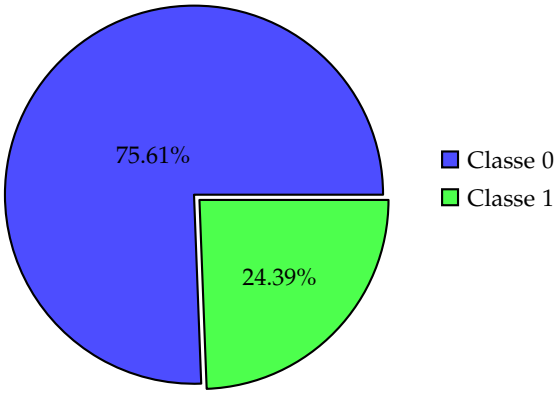

Figure 1: *Répartition des classes de texte*

Sous-échantillonnage

J'ai réduit la taille de la classe majoritaire pour équilibrer la distribution des classes. Cette méthode consiste à supprimer des échantillons de la classe dominante afin que les deux classes aient un nombre équivalent d'exemples Cependant, après avoir effectué ces traitements et réévalué les performances du modèle, j'ai constaté que cette méthode n'a eu d'impact significatif sur les résultats du modèle. Les performances sont restées globalement stables.

validation croisée

La validation croisée est une méthode pour tester la précision d'un modèle de machine learning. Elle consiste à diviser les données en plusieurs parties : on entraîne le modèle sur certaines parties, puis on le teste sur les autres. Cela permet de mieux évaluer la performance du modèle en utilisant toutes les données disponibles pour l'entraînement et le test, mais de manière alternée.

*Corresponding author: amine.ouakib@umontreal.ca

BayesSearchCV

BayesSearchCV est une méthode d'optimisation des hyperparamètres utilisée en machine learning. Contrairement à une recherche brute de grille (grid search), qui teste toutes les combinaisons possibles d'hyperparamètres, BayesSearchCV utilise une approche probabiliste (basée sur les probabilités de Bayes) pour explorer plus intelligemment l'espace des hyperparamètres. Cela permet de trouver plus rapidement les meilleurs paramètres en se concentrant sur les zones prometteuses plutôt que d'explorer toutes les combinaisons possibles.

Classification Binaire de Texte

Pour la classification de texte, divers algorithmes permettent de catégoriser des documents, des avis ou d'autres types de données textuelles en classes spécifiques, comme la détection de sentiments, la catégorisation thématique ou le filtrage de spam. Ce modèle de classification textuelle utilise des caractéristiques clés, telles que la fréquence des mots, la signification sémantique et la structure syntaxique, pour identifier les catégories. Des algorithmes d'apprentissage automatique les machines à vecteurs de support (SVM), les K plus proches voisins (KNN), les forêts aléatoires, les perceptrons multicouches (MLP), les arbres de décision, la régression logistique, XGBoost et Naive Bayes permettent d'identifier efficacement des distinctions subtiles entre les classes. En particulier, Naive Bayes est bien adapté pour la classification de texte grâce à sa capacité à gérer des données textuelles en utilisant des probabilités pour prédire la classe la plus probable en fonction des caractéristiques observées. Ces approches permettent une catégorisation précise des données textuelles, utile dans l'analyse des réseaux sociaux, le traitement des retours clients, et l'organisation de contenu textuel.

Méthodes

Naive Bayes : Le **Naive Bayes** est une famille de modèles de classification probabilistes basés sur le théorème de Bayes, avec une hypothèse clé de naïveté, d'où le nom "naive". Cette hypothèse simplifie le calcul en supposant que les caractéristiques (ou attributs) d'un ensemble de données sont indépendantes les unes des autres, conditionnellement à la classe cible.

Principe du Naive Bayes

Le modèle de Naive Bayes repose sur la règle de Bayes :

$$P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)}$$

où :

- $P(C|X)$ est la probabilité qu'une observation appartienne à la classe C étant donné les caractéristiques X,
- $P(X|C)$ est la probabilité d'observer X si la classe est C,
- $P(C)$ est la probabilité a priori de la classe C,
- $P(X)$ est la probabilité totale d'observer X, calculée pour normaliser les probabilités.

Hypothèse d'indépendance

Dans le cas de Naive Bayes, on suppose que chaque caractéristique X_i d'un vecteur $X = (X_1, X_2, \ldots, X_n)$ est indépendante des autres, conditionnellement à la classe C. Cela donne :

$$P(X|C) = P(X_1|C) \cdot P(X_2|C) \cdot \ldots \cdot P(X_n|C)$$

Cette hypothèse simplifie énormément les calculs, permettant de traiter chaque caractéristique séparément.

Multi-layer Perceptron (MLP) : Le perceptron multicouche (MLP) est un type de réseau de neurones artificiel composé de plusieurs couches de neurones, qui permet d'apprendre des relations complexes dans les données. Il est couramment utilisé pour des tâches variées, comme la classification d'images, la détection de spam et la prédiction de tendances, grâce à sa capacité à modéliser des liens non linéaires entre les données d'entrée et de sortie.

Support Vector Machine (SVM) : est un algorithme d'apprentissage supervisé utilisé principalement pour la classification. Il cherche à trouver l'hyperplan qui sépare les différentes classes de manière optimale, c'est-à-dire celui qui maximise la marge entre les points de données les plus proches de chaque classe (les vecteurs de support). En cas de données non linéairement séparables, SVM utilise un noyau (kernel) pour projeter les données dans un espace de dimensions supérieures où elles deviennent séparables.

K-nearest neighbor (KNN) : est un algorithme d'apprentissage supervisé utilisé pour la classification et la régression. Dans le cadre de la classification, pour un point de données donné, l'algorithme examine les k voisins les plus proches de ce point dans l'ensemble d'entraînement, puis attribue la classe la plus fréquente parmi ces voisins. La mesure de la proximité (distance) est souvent basée sur des métriques comme la distance Euclidienne. KNN est un algorithme paresseux, ce qui signifie qu'il ne nécessite pas de phase d'entraînement, mais effectue des calculs à chaque prédiction.

Random Forest : est un algorithme d'apprentissage automatique qui combine plusieurs arbres de décision pour améliorer la précision et éviter le surapprentissage (overfitting). Il crée une "forêt" en générant de nombreux arbres de décision sur des sous-ensembles aléatoires de données et de caractéristiques. Chaque arbre vote pour une prédiction, et la classe ou la valeur finale est déterminée par un vote majoritaire (dans le cas de la classification) ou une moyenne (pour la régression). Cet ensemble d'arbres permet de réduire la variance et d'obtenir de meilleures performances par rapport à un arbre de décision individuel.

XGBoost: (Extreme Gradient Boosting) est un algorithme de machine learning basé sur le boosting, une méthode qui combine plusieurs modèles faibles (généralement des arbres de décision) pour créer un modèle puissant. XGBoost optimise la performance en minimisant l'erreur en ajoutant des arbres successifs qui corrigent les erreurs des modèles précédents. Il utilise une approche de gradient boosting, où les arbres sont ajoutés en se concentrant sur les erreurs des arbres précédents. XGBoost est connu pour sa rapidité, son efficacité et ses performances exceptionnelles sur de nombreux types de problèmes, notamment ceux impliquant de grandes quantités de données et des modèles complexes. Il inclut également des techniques de régularisation pour éviter le surapprentissage (overfitting).

Performance Measurement

L'analyse statistique, notamment l'exactitude, a été utilisée pour évaluer la robustesse des modèles développés pour classifier la qualité des textes. Cependant, la spécificité, la sensibilité, la précision et la matrice de confusion ont également été prises en compte.

Accuracy : L'exactitude est une statistique qui décrit la performance globale du modèle sur toutes les classes. Elle est utile lorsque toutes les classes ont la même importance. Elle est déterminée en divisant le nombre de prédictions correctes par le nombre total de prédictions.

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{1}$$

Precision : La précision indique combien de cas correctement prédits ont été trouvés positifs.

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

Recall : Le rappel indique combien de cas réellement positifs (classe 1) nous avons pu prédire correctement avec notre modèle.

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

F1 score : Le score F1 est une moyenne harmonique de la précision et du rappel, fournissant ainsi une synthèse de ces deux mesures. Il est maximal lorsque la précision est égale au rappel.

$$F1 - score = \frac{(2 * Precision * Recall)}{(Precision + Recall)} \tag{4}$$

où,
TP: représente le nombre de textes correctement classés comme appartenant à la classe positive (classe 1).
FP: représente les textes négatifs classés comme appartenant à la classe positive (classe 1).
TN: représente les textes négatifs correctement classés comme négatifs (classe 0).
FN: représente les textes positifs classés comme négatifs (classe 0).

Phase I: Algorithme Naive Bayes Multinomial avec Pondération des Classes

Dans cette phase de la compétition, j'ai utilisé un classifieur Naive Bayes multinomial pour effectuer la classification de texte binaire. Ce modèle est adapté aux données de comptage, comme celles qu'on trouve souvent dans la classification de texte (par exemple, les fréquences d'apparition de mots).

Justification du Choix de l'Algorithme

Le Naive Bayes Multinomial est particulièrement efficace pour les problèmes de classification de texte, car il repose sur une hypothèse simple, à savoir que les caractéristiques (dans ce cas, les mots) sont conditionnellement indépendantes étant donné la classe. Cette approche est particulièrement adaptée lorsque les données sont discrètes et que les classes sont déséquilibrées.

Modèle et Lissage

Le modèle de Naive Bayes Multinomial repose sur la formule :

$$P(C \mid X) \propto P(C) \prod_{i=1}^{n} P(x_i \mid C)$$

Où $P(C)$ est la probabilité a priori d'une classe, et $P(x_i \mid C)$ est la probabilité conditionnelle de chaque caractéristique x_i (mot) donnée la classe C. Afin de traiter les caractéristiques absentes dans certaines classes, j'ai appliqué un lissage de Laplace (additif) pour éviter les probabilités nulles. Ce lissage est effectué en ajoutant une constante α (ici, $\alpha = 1.0$) aux comptages de chaque caractéristique.

Table 1: *Performance du Naive Bayes multinomial avec différentes pondérations.*

Class Weights	Accuracy	Precision	Recall	F1-score
0: 0.25, 1: 0.75	0.7427	0.4961	0.6687	0.6930
0: 0.1, 1: 0.9	0.7363	0.4875	0.6937	0.6910
0: 0.7, 1: 0.3	0.7591	0.5226	0.625	0.7010
0: 0.15, 1: 0.85	0.7368	0.4880	0.6833	0.6899
0:0.4,1: 0.6	0.7448	0.4992	0.6541	0.6927
0: 0.5, 1: 0.5	0.7549	0.5148	0.6520	0.7015

Pondération des Classes

Dans le cadre de ce projet, les classes étaient déséquilibrées, ce qui peut affecter la performance du modèle en favorisant la classe majoritaire. Pour y remédier, j'ai intégré une pondération des classes, où chaque classe a une pondération inversement proportionnelle à sa fréquence dans les données d'entraînement. Cela permet de donner plus d'importance aux classes minoritaires lors de l'entraînement du modèle. Par exemple, les pondérations utilisées étaient : Pondérations=0:valeur1,1:valeur2 Ces pondérations ont été appliquées lors du calcul des probabilités a priori de chaque classe et ont permis d'améliorer l'équilibre entre les classes.

Résultats et interprétation

Interprétation

- Optimisation des Pondérations : Les pondérations affectent directement l'équilibre entre précision et rappel. En particulier, les modèles avec des pondérations équilibrées comme 0: 0.7, 1: 0.3 et 0: 0.5, 1: 0.5 montrent de bons résultats en termes de score F1.
- Modèles Favorisant la Classe Minoritaire : Les pondérations comme 0: 0.1, 1: 0.9 favorisent la classe 1 (la classe minoritaire) et entraînent un rappel plus élevé, mais la précision et l'exactitude en souffrent.
- Résultats Remarquables avec la Pondération 0: 0.5, 1: 0.5 : L'algorithme avec la pondération 0: 0.5, 1: 0.5 a également donné d'excellents résultats sur l'ensemble de texte avec une exactitude de 0.7426, offrant ainsi un bon compromis entre les différentes métriques tout en maintenant une performance équilibrée.

Phase II:Sélection et Évaluation des Algorithmes pour la Classification Binaire

Dans cette phase, une approche de recherche bayésienne (BayesSearchCV) a été adoptée afin d'identifier les meilleurs hyperparamètres pour chaque algorithme utilisé. Contrairement à la recherche sur grille (Grid Search), BayesSearch explore l'espace des hyperparamètres de manière probabiliste, en se concentrant sur les zones les plus prometteuses. Cette méthode permet de trouver plus rapidement les paramètres optimaux en évaluant la performance de chaque combinaison sur l'ensemble de validation. L'objectif était d'optimiser les performances des modèles en ajustant des paramètres clés, tels que la régularisation ou les taux d'apprentissage, et ainsi garantir une comparaison équitable et robuste des différents algorithmes.

3.

Recherche d'hyperparamètres pour le Naive Bayes multinomial avec BaesSearchCV

- J'ai utilisé BayesSearchCV pour effectuer une recherche bayésienne des hyperparamètres du modèle, spécifiquement sur les paramètres alpha (paramètre de lissage), fitprior (activation ou non des priorités de classe), et classprior0 (priorité de classe explicite pour la première classe). L'espace de recherche a été défini comme suit :
  alpha : Valeurs réelles entre 0.01 et 1.0, distribué de façon log-uniforme pour explorer des valeurs de lissage fines. fitprior : Valeurs booléennes, pour déterminer si le modèle ajuste automatiquement les priorités de classe ou utilise des priorités explicites. classprior0 : Valeurs réelles entre 0.1 et 0.9, pour ajuster la priorité de classe explicitement. Le modèle a été évalué sur 50 itérations de recherche.
- Suite à l'intégration de cette recherche d'hyperparamètres dans mon code, j'ai observé une amélioration des performances du modèle. Un nouveau score a été affiché sur le tableau de bord, atteignant 74,274. De plus, j'ai retrouvé exactement ce même score lorsque j'ai appliqué la validation croisée sur l'ensemble du modèle. Cette cohérence montre que la recherche des hyperparamètres a effectivement amélioré la performance globale du modèle.
- Après avoir appliqué le sous-échantillonnage sur les données, j'ai observé des résultats prometteurs sur l'ensemble de validation, comme le montrent les figures suivantes. Cette technique a permis de mieux équilibrer la distribution des classes, ce qui a entraîné une amélioration significative des scores de performance, notamment en termes de F1-score. Cependant, malgré ces bons résultats sur l'ensemble de validation, j'ai choisi de ne pas inclure cette approche dans ma soumission pour la compétition. La raison de cette décision réside dans le fait que le sous-échantillonnage pourrait ne pas être représentatif des données réelles et pourrait conduire à une perte d'information importante, surtout dans le cas d'une grande asymétrie entre les classes. En conséquence, j'ai préféré soumettre les résultats obtenus sans cette étape de sous-échantillonnage, afin de garantir la robustesse du modèle face à des données non équilibrées en conditions réelles.

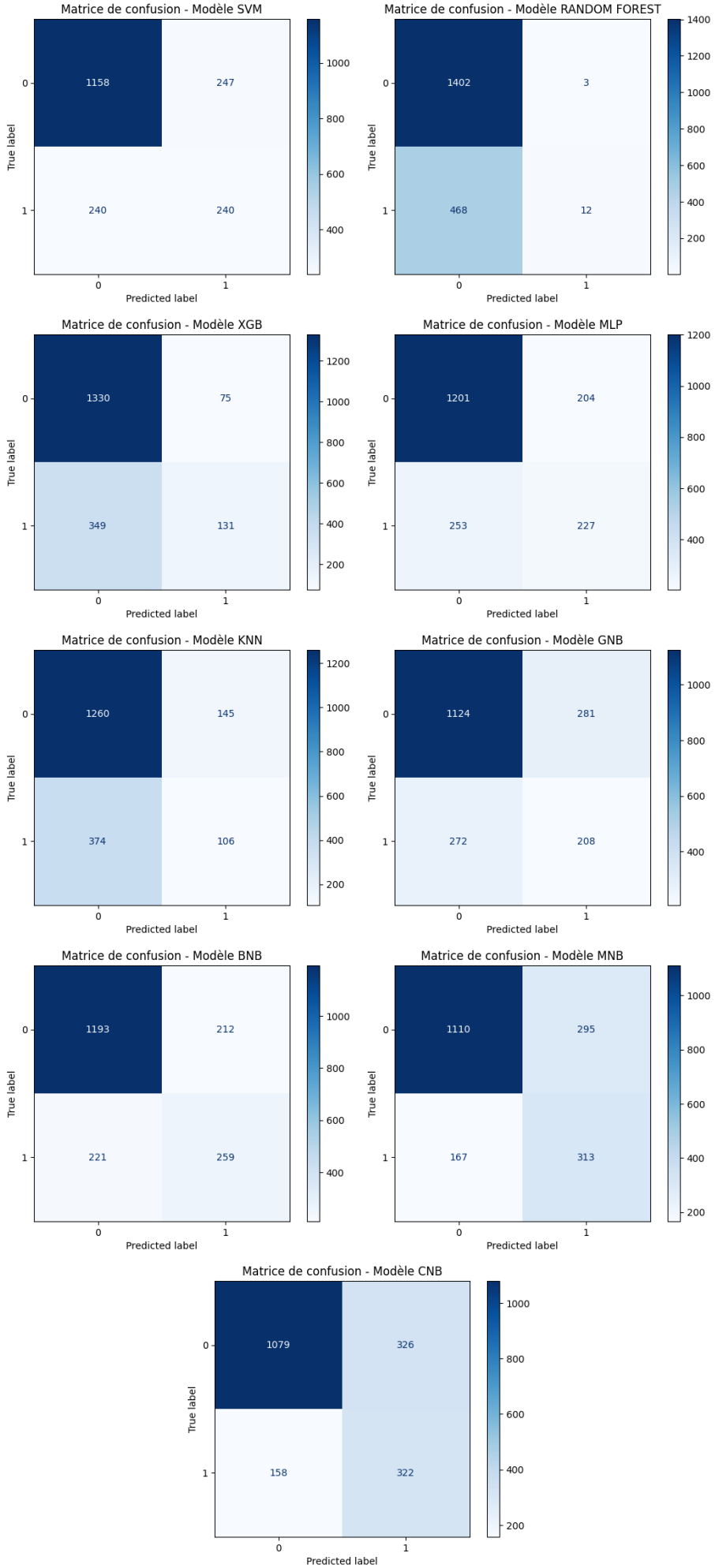

Figure 2: *Confusion Matrix for all models*

Table 2: *Performance des modèles de machine learning utilisés pour prédire la classe des textes.*

Algorithms	Accuracy	Precision	Recall	F1-score
SVM C=8.0, gamma='auto', kernel='linear	0.74	0.66	0.66	0.66
MLP $\text{hidden}_{l}ayer_{s}izes = (100,), activation = 'relu', learning_{r}ate_{i}nit = 0.001, max_{i}ter = 100$	0.76	0.68	0.66	0.67
Random Forest	0.75	0.77	0.51	0.45
KNN $n_{n}eighbors = 5$	0.72	0.60	0.56	0.56
XGBoost $\max_{d}epth = 6, min_{c}hild_{w}eight = 1, n_{e}stimators = 100, learning_{r}ate = 0.1, subsample = 0.8, colsample_{b}ytree = 0.8$	0.78	0.71	0.61	0.62
Gaussian NB	0.71	0.62	0.62	0.62
Bernoulli NB	0.77	0.70	0.69	0.70
Multinomial NB alpha=1.0	0.75	0.69	0.72	0.70
Complement NB	0.74	0.68	0.72	0.69

	precision	recall	f1-score	support
0.0	0.77	0.70	0.73	460
1.0	0.72	0.79	0.75	460
accuracy			0.74	920
macro avg	0.75	0.74	0.74	920
weighted avg	0.75	0.74	0.74	920

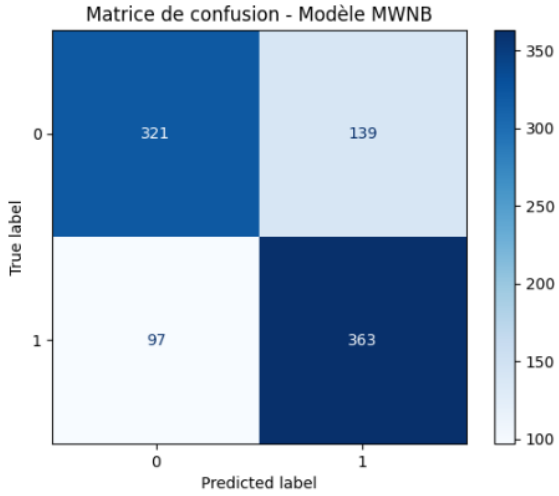

Figure 3: *Confusion Matrix for BWNB model after undersampling*

3.

Conclusion

Ce projet a permis d'explorer plusieurs algorithmes de machine learning pour la classification binaire de texte, avec un focus sur l'équilibrage des classes via le sous-échantillonnage et la pondération. Bien que ces techniques aient équilibré les classes, elles n'ont pas amélioré de manière significative les performances finales du modèle.

Perspectives

Pour aller plus loin, l'utilisation de représentations textuelles avancées (comme les embeddings) et de modèles d'ensemble pourrait offrir une meilleure généralisation. Tester des approches de sur-échantillonnage synthétique pourrait également être bénéfique pour traiter les déséquilibres de classe.

Références

- https://shs.cairn.info/revue-document-numerique-2013-1-page-73?lang=frs2n2
- https://shs.cairn.info/revue-document-numerique-2018-3-page-33?lang=frs2n1
- https://www.kaggle.com/code/sharatsk/nlp-text-documentMachine-Learning-Models
- https://www.kaggle.com/code/victornicofac/naive-bayes