Compétition Kaggle - Rapport

Guillaume Genois, 20248507
Kamen Damov, 20102811

13 novembre 2024

1 Introduction

La présente compétition consiste à trouver $f \in F$ avec F une famille riche de fonctions (linéaire, arbres de décisions, bayésiennes, etc.) pour un problème de classification binaire sur un corpus de texte qui est représenté par des vecteurs de compte. Les données sont déjà séparées pour le test et l'entraînement tel que, $X_{train} \in \mathbb{N}^{k \times d}$, et $X_{test} \in \mathbb{N}^{m \times d}$. Étant donné la sparsité des données, et le fait que les vecteurs d'entrées sont des fréquences, nous avons prioriser les approches bayésiennes qui sont robustes face au problème de sparsité, sont bien adaptées aux données de fréquences, et sont relativement rapides à implémenter. Pour le premier jalon, nous avons implémenter un modèle Naif de Bayes avec validation croisée pour le lissage de Laplace, sans prétraitement des données. Cette méthode nous a donné sur un score de 0.5893 sur la validation et de 0.719 sur l'ensemble de test privés. Pour le deuxième jalon, nous avons fait un prétraitement plus exhaustif des données, en explorant les techniques pour réduire la dimensionnalité, balancer les données en sur-échantillonant ou en sous-échantillonant, retirer les mots sans importance, et appliquer des transformations TF-IDF à nos ensembles d'entraînement et de test indépendamment. De plus, nous avons élargi l'ensemble de fonctions à tester allant de familles de modèles bayésiens (tel que la Naïve de Bayes et le Complément de Bayes), modèles à base d'arbres (XGBoost), modèles linéaires (SVM et régression logistique) et modèles ensemblistes (ensemble models en anglais) par vote. Nous avons également utilisé la procédure de validation croisée pour trouver les meilleurs hyperparamètres de chacun de ces modèles. Le modèle le plus performant était un modèle ensembliste qui avait un score F1 de 0.5983 sur l'ensemble de validation et 0.7233 sur l'ensemble de test.

2 Conception de fonctionnalité

Nous avons constater plusieurs axes de prétraitement qui pourraient faciliter l'apprentissage de nos algorithmes. Nous énumérons les techniques utilisées dans cette section, mais il est important de noter que ces méthodes ne sont pas nécessairement utilisées ensembles. Comment mentionné, nous n'avons pas appliqué de prétraitement de données lors du premier jalon, nous avons directement appliqué la Naïve de Bayes à nos données. Cela dit, pour le deuxième jalon, nous avons tenté d'appliquer certaines méthodes mentionnées ci-après.

2.1 Rééchantillonage

Nous avons tout d'abord constaté le débalancement de classe dans X_{train} était d'environ 76% pour la classe 0 et de 14% pour la classe 1. Les sous-sections suivantes discutent de nos choix de rééchantillonage.

2.1.1 SMOTE

Nous avons d'abord tenter de suréchantilloner notre jeu de données avec une technique de génération d'échantillon synthétique, soit SMOTE. Cette méthode se base sur l'algorithme de K plus proche voisins pour générer des nouveaux points de données dans le voisinage d'un point de la classe minoritaire. Pour un point donné x_i et x_{voisin} un des k points les plus proches de x_i, on génère x_{synth} ainsi:

$$x_{synth} = x_i + \lambda(x_{voisin} - x_i)$$

avec $\lambda \in [0, 1]$. Nous générons 1000 observations dans la classe minoritaire.

2.1.2 Sous-échantillonage

Si nous tentons plutôt de sous-échantilloner, il nous est possible de retirer des observations aléatoirement dans la classe majoritaire jusqu'à un seuil choisi. Pour la compétition, nous avons choisi de retirer la moitié des données dans la classe majoritaire ce qui nous permet de garder nos classes débalancées, mais nous passons d'un ratio de 76/14 à environ 60/40. Par ce fait même, nous retirons environ 3000 lignes du jeu d'entraînement, mais nous considérons qu'une taille d'environ 7000 lignes reste raisonnable pour l'entrainement. Voir annexes pour ces graphiques (Figure 3 à 6).

2.2 Réduction de dimensions

Étant donné que nos vecteurs sont de très hautes dimensions, nous avons penser à retirer des attributs (qui sont les mots inclus dans le corpus de texte fournis) pour accélérer l'entraînement et enlever les attributs qui n'apportent aucune information quant à la discrimination dans la classe 0 ou 1. Nous avons testé deux méthodes pour réduire les dimensions selon un critère calculés. Une autre méthode a aussi été essayées en prenant plutôt en compte le sens des mots.

2.2.1 Réduction à base d'arbres

Nous utilisons un arbre de décision avec le critère de Gini pour évaluer l'importance des attributs. Le coefficcient de Gini est calculé ainsi: Gini $= 1 - \sum_{i=1}^{C} \mathbb{P}[\text{un points choisi aléatoirement appartient à } C_i]^2$ avec $C_i \in C$. À chaque division, l'arbre choisit l'attribut qui réduit le plus l'impureté de Gini, indiquant ainsi les attributs les plus discriminants pour la prédiction. En analysant l'importance cumulative de chaque attribut dans l'ensemble de l'arbre, nous extrayons les k attributs les plus importants, ceux-ci étant les plus déterminants pour la prédiction.

2.2.2 Réduction à base de fréquences cumulatives

Sachant que le jeu de données est très éparse, nous avons tenter de retirer les mots qui surviennent très rarement à travers le corpus de texte. Nous détectons ces mots en calculant la somme des fréquences d'un mot à travers tout le corpus, en triant le corpus en ordre croissant, et en retirant une proportion des mots du jeu de données en tronquant le vecteurs de fréquences cumulatives, à l'indice qui couvre la proportion voulue du corpus. En somme, nous retirons les mots qui ne sont pas fréquents dans le corpus. En résultat, nous enlevons plus de 20000 mots en gardant 95% des mots les plus fréquents. Voir Algorithme 2 pour plus de détails sur la méthode, et Figure 1 et 2 pour une représentation visuelle de la transformation.

2.2.3 Réduction en retirant les mots vides

En langage naturel, certains mots n'apportent pas de sens à une phrase. Ils ne sont présents que pour faire les liaisons et les transitions fluides entre les mots. Ceux-ci se font appeler les mots vides ("stopwords"). En anglais, des exemples de ces mots sont "the", "of", "a", etc. En dessinant le graphique de la fréquence de ces mots, il est possible de remarquer que certains apparaissent très souvents tels que juste qu'à 10000 fois à travers les documents. Il apparait alors intéressant de retirer les colonnes de ces mots dans nos jeux d'entrainement et de test. La dimensionnalité diminue alors d'environ 40 colonnes. Nous avons donc utilisé la liste de mots vides que la bibliothèques Natural Language Toolkit (nltk) utilise. Cette liste est présente en tant que fichier dans notre remise du jalon 2 (Bird et al., YEAR). Voir graphiques dans l'annexe (Figure 11 à 14).

2.3 Transformation TF-IDF

Afin de mesurer l'importance d'un mot selon sa fréquence à travers les documents, il est possible de le transformer avec TF-IDF. La formule est la suivante :

$$m_{x,y} = tf_{x,y} * log(\frac{n}{1 + df_x})$$

où $tf_{x,y}$ est la fréquence du mot x dans le document y, n est le nombre de documents et df_x est le nombre de documents dans lesquels le mot x apparait.
Ainsi, les mots apparaissant souvent à travers tous les documents auront un moins gros poids ce qui devrait faire diminuer leur bruit.

3 Algorithmes utilisés

Nous avons effectué multiples expériences avec multiples algorithmes d'apprentissage différents. Les algorithmes qui ont été les plus efficaces sont ceux présentés ci-après.

Pour l'implémentation du premier jalon, nous avons choisi d'implémenter un classifieur de Bayes Naif. Pour ce premier jalon, nous n'avons apporté de prétrairement aux données. Nous nous sommes uniquement concentré sur l'implementation du modèle, et de la validation croisée de celui-ci.

3.1 Classifieur de Bayes Naif

D'abord et avant tout, notre argument pour utiliser ce modèle était que nous avions des vecteurs de compte (et donc de fréquence) en haute dimension. La Naïve de Bayes prend moins d'effort de régularisation contrairement à un modèle comme la régression logistique qui est plus prône d'être affecté par la haute dimensionnalité sans critère de régularisation bien défini. Lors de la phase d'entraînement, on estime le postérieur de Bayes (probabilités conditionnelles) par le jeu de donnés d'entraînement. Posons, $C \in \{0, 1\}$ étant la variable réponse 0 ou 1, et $x_i \in X_{train}$. Une hypothèse cruciale à cet algorithme est l'indépendance des attributs (ou des mots dans le présent problème). Nous avons donc par le théorème de Bayes:

$$\mathbb{P}[C|X = x_i] = \frac{\mathbb{P}[X = x_i|C]\mathbb{P}[C]}{\mathbb{P}[X = x_i]} = \frac{\prod_{j=1}^{d} \mathbb{P}[X = x_{i,j}|C]\mathbb{P}[C]}{\mathbb{P}[X = x_i]}$$

Le critère de classification est le suivant:

$$\underset{C_i \in C}{\text{argmax}} \quad P[C_i|X = x_i] = \underset{C_i \in C}{\text{argmax}} \frac{\prod_{j=1}^{d} \mathbb{P}[X = x_{i,j}|C]\mathbb{P}[C]}{\mathbb{P}[X = x_i]}$$

De façon équivalente, nous pouvons écrire le critère en fonction de la vraisemblance (le numérateur de l'expression ci-haut), car maximiser la probabilité, est équivalent à maximiser la vraisemblance, qui est aussi équivalent à maximiser la log vraisemblance, car les deux sont des fonctions croissantes. Voici le critère réécrit en fonction de la log vraisemblance:

$$\underset{C_i \in C}{\text{argmax}} \log \prod_{j=1}^{d} \mathbb{P}[X = x_{i,j}|C]\mathbb{P}[C] = \underset{C_i \in C}{\text{argmax}} \sum_{j=1}^{d} \log \mathbb{P}[X = x_{i,j}|C] + \log \mathbb{P}[C]$$

En pratique, il est important de voir que toutes les probabilités dans les expressions ci-haut sont calculables par le biais de fréquences observées dans le jeu de données d'entraînement. $\mathbb{P}[C]$ est la probabilité à priori d'être dans la classe 0 ou 1, donc $\mathbb{P}[C] = \frac{\text{Nombre d'observations dans la classe } i}{\text{Nombre d'observations dans } X_{train}}$. Similairement, $\mathbb{P}[X = x_{i,j}|C]$ est la probabilité que $x_{i,j}$, sachant la classe C, qui représente $\mathbb{P}[X = x_{i,j}|C] = \frac{\text{Nombre de } j \text{ dans la classe } C_i}{\text{Nombre total de mots dans la classe } C_i}$. Cela dit, il est important de noter qu'il faut lisser cette probabilité afin d'éviter d'annuler la vraisemblance qui se produit lorsque $\mathbb{P}[X = x_{i,j}|C] = 0$, qui survient lorsque un mot j ne se trouve pas dans la classe C_i. Posons $\alpha \in \mathbb{R}^+$ et $\alpha > 0$ l'hyperparamètre de lissage. Nous calculons dorénavant la vraisemblance,

$$\mathbb{P}[X = x_{i,j}|C] = \frac{\text{Nombre de } j \text{ dans la classe } C_i + \alpha}{\text{Nombre total de mots dans la classe } C_i + \alpha \times \text{ nombre de mots dans le vocabulaire}}$$

. Une grande valeur pour α applique un fort lissage sur toute les probabilités, alors qu'une faible valeur de α produit un lissage plus dur. Nous pouvons trouver le α optimal par validation croisée.

3.2 Classifieur Complément de Bayes naïf

Contrairement au classifieur Naïve Bayes décrit précédemment, le Classifieur Complément de Bayes calcule les statistiques des caractéristiques (comme les fréquences des mots) en fonction des classes autres que celle que l'on cherche à prédire. C'est pour cela qu'on parle de "complément" : pour chaque classe c, on prend en compte les informations des classes complémentaires, ce qui permet de compenser les erreurs d'estimation dans les classes minoritaires. Cette approche tend à réduire l'impact des caractéristiques qui sont très spécifiques à certaines classes et qui pourraient biaiser le modèle. Pour ce qui est des hyperparamètres de ce modèle, c'est le même α que dans la Naive de Bayes vanille expliqué ci-haut.

3.3 Classifieur XGBoost

Nous avons également exploré l'application d'un modèle basé sur des arbres au problème. Cet algorithme crée une série d'arbres de décision, chaque arbre subséquent étant formé pour corriger les erreurs des arbres précédents, en utilisant les gradients et Hessians de l'erreur pour orienter ses subdivisions. Ces gradients et Hessians sont calculé par la derivée seconde et première de la fonction de perte d'entropie croisée. Voir annexe pour plus de détails sur les dérivations mathématiques. Contrairement au vote majoritaire utilisé dans des méthodes comme la forêt aléatoire, XGBoost produit une prédiction finale en sommant les prédictions pondérées de chaque arbre. Comme pour le critère de Gini mentionné plus haut, XGBoost utilise un critère de gain de subdivision pour déterminer les branchements de chaque arbre de décision.

XGBoost utilise ainsi la somme des gradients et des Hessians pour guider la construction de chaque arbre, en choisissant les divisions qui maximisent le gain de subdivision pour minimiser l'erreur totale.

3.4 Classifieur perte de Huber modifié avec descente de gradient stochastique (SGD)

Le Classifieur SGD (Stochastic Gradient Descent) avec la perte Modified Huber est une méthode de classification linéaire qui utilise la descente de gradient stochastique pour minimiser une fonction de coût basée sur la distance entre les prédictions et les étiquettes cibles. La fonction de perte Huber modifiée est une variante de la perte Hinge, utilisée dans les SVMs, mais avec une modification qui la rend lisse et plus tolérante aux valeurs aberrantes (outliers). L'algorithme SGD utilise un échantillonnage aléatoire des données pour mettre à jour les paramètres du SVM à chaque itération, au lieu d'utiliser l'ensemble des données d'apprentissage, ce qui le rend rapide et efficace, même pour des jeux de données volumineux.

3.5 Classifieur SVM (SVC)

Le principe d'un SVM est de trouver un hyperplan qui sépare les données de différentes classes avec la plus grande marge possible, c'est-à-dire la distance maximale entre les points de chaque classe les plus proches de l'hyperplan. Une fois l'hyperplan trouvé, le SVM classe un nouvel exemple en fonction de son côté par rapport à l'hyperplan : si le point est d'un côté, il appartient à la classe positive, sinon il appartient à la classe négative.

3.6 Classifieur par régression logistique

La régression logistique utilise une fonction sigmoïde pour transformer la sortie linéaire en une probabilité comprise entre 0 et 1, ce qui permet de modéliser la probabilité d'appartenance à une classe. Le modèle cherche à optimiser les coefficients associés à chaque caractéristique pour minimiser la différence entre les prédictions et les étiquettes réelles. Si la probabilité est assez haute d'être dans une classe, cette classe lui est attribuée.

3.7 Classifieur d'apprentissage par ensembles

Nous avons tenter de combiner plusieurs modèles de nature différentes et d'aggréger leurs prédictions dans le but d'augmenter la performance sur le score F1. Cette procédure consiste à entraîner indépendamment des classifieurs et procéder à un vote lisse en extrayant et aggrégeant les probabilités de classification de chaque modèle dans l'ensemble. Dans notre cas, nous avons entraîner un classifieur complémentaire de Bayes (modèle fréquentiste), un classifieur XGBoost (modèle à base d'arbres de décision), et un classifieur SVM sous perte Huber modifiée à descente de gradient stochastique (modèle linéaire). Il est important de noter que nous avons choisi ces modèles pour avoir une frontière de décision de nature différente, et avoir des estimations plus robustes, qui ne sont pas biaisées par la similarité de la frontière de décision apprise entre les modèles. Voir algorithme 2 dans l'annexe, pour une description plus rigoureuse de la méthode.

4 Méthodologie

4.1 Répartition pour l'entrainement et la validation

Pour tout les algorithmes et prétraitement choisis, nous avons appliqué une validation croisée k-fold avec $k = 5$ stratifiées. La stratification dans la séparation des données en ensemble d'entraînement et de validation est cruciale, car nous avons un jeu de données débalancés (malgré le rebalancement mentionné plus haut). Ainsi, nous avons une représentation proportionelle des étiquettes de la classe 0 et 1 dans l'ensemble d'entraînement et de validation.

4.2 Ajustement des hyperparamètres

Nous utilisons la procédure de recherche aléatoire pour trouver la meilleure combinaison d'hyperparamètres. Celle-ci est choisie en prenant la combinaison ayant la meilleure moyenne du score F1 à travers les 5 divisions. Cette méthode s'applique à tout les algorithmes du deuxième jalon, par soucis computationnel et de production de résultats rapides. Pour le modèle bayésien du premier jalon, nous appliquons une recherche en grille, vu que nous avons qu'un seul hyperparamètre à ajuster. Nous implémentons une validation croisée (k-fold cross validation) avec $k = 7$ pour trouver l'hyperparamètre α optimal. Nous avons établi un espace hyperparamétrique de valeur espacée uniformément, allant de 0.4 à 1.05 avec des sauts de 0.05. Dû au fait que l'espace est relativement restreint, cet espace est visité séquentiellement par une recherche en grille (grid-search). Comme mentionné ci-haut, autre que le modèle XGBoost, nous avons opté pour un espace restreint d'hyperparamètres, en ajustant prioritairement les poids du terme de régularisation. Pour le modèle SVC, nous avons le coefficient du noyau RBF, et le poids attribué au terme de régularisation de la pénalité. Pour le classifieur SGD, nous avons utilisé le terme de régularisation ElasticNet, qui est une combinaison de terme de pénalité $l1$ et $l2$. Il s'en suit naturellement, que la proportion de chaque perte dans la régularisation, $l1_{ratio}$ est un hyperparamètre, que nous avons borné dans un espace de valeur linéairement espacée dans $[0.001, 1]$. Le poids de cette régularisation ElasticNet est modulée par un hyperparamètre qu'on trouve également par validation croisée qui prend des valeurs dans un espace séparé logarithimquement entre $[0.01, 1]$. Pour XGBoost, nous avons décider de restreindre l'espace hyperparamétrique à quatres hyperparamètres cruciaux. Le premier est le taux d'apprentissage qui est la taille du pas lors de la descente de gradient sur la perte d'entropie croisée pour calculer le gain de subdivision $\gamma \in \mathbb{R}$, $\gamma > 0$. Celui-ci prend des valeurs uniformément distribué entre $[0.01, 0.2]$. Le deuxième hyperparamètre est le nombre d'arbres que nous créons séquentiellement, prenant des valeurs de espacées de 100 entre $[200, 500]$. Ensuite, nous avons la profondeur maximale de chaque arbres prenant des valeurs dans $[3, 5, 7, 10]$. Et finalement, nous avons un hyperparamètre pour la proportion de données à utiliser pour produire un arbre, ainsi contrôlant le sur-apprentissage. Cet hyperparamètre prend des valeurs uniformément distribuées entre $[0.4, 0.6]$.

4.3 Astuces d'optimisation

Nous avons opté pour une recherche aléatoire plutôt qu'une recherche en grille qui est plus chronophage pour un résultat comparable (Bengio et Bergstra, 2012). Une autre optimisation a été de convertir les données de train et de test en int8. Ceci a permis de contourner plusieurs erreurs de mémoire et a permis d'accélérer le temps computationnel tout en conservant l'étendu de toutes les valeurs présentes dans les jeux de données. En effet, la fréquence maximale dans les matrices est de 38 ce qui est inférieure à $2^8 = 256$. Cependant, l'algorithme du SVC est une exception, car il a besoin d'avoir les données encodées en float32 ou float64 afin de faire ses calculs. Nous avons donc simplement rechanger les types. Une dernière astuce d'optimisation a été d'utiliser la bibliothèque cuML de RAPIDS créé par NVIDIA qui fournit une grande partie des mêmes méthodes que scikit-learn mais en utilisant le GPU au lieu du CPU (NVIDIA, 2023). Ceci nous a permi de principale accélérer les temps de calculs pour les SVC.

5 Résultats

Par soucis de présentation, nous renommons le modèle Complément Naive de Bayes CNB, XGBoost par XGB, Régression Logistique par LogReg. Les modèles ensemblistes figurent sur la même ligne. Les colonnes sont le type de prétraitement appliqué.

Model	SMOTE	Under sampl.	Tree	Cum Sum	Stopwords	TF-IDF	F1 Val	F1 Test
NB vanille							0.5893	0.7196
CNB						x	0.5611	0.7127
CNB			x		x	x	0.5636	0.6324
CNB	x		x		x	x	0.6556	0.6870
CNB		x		x	x		0.7041	N/A
LogReg		x		x	x		0.6308	N/A
SVM		x		x	x		0.6415	N/A
SGD avec Huber mod.		x		x	x		0.6294	0.6629
XGBoost		x		x	x		0.6592	0.6115
CNB, XGB, LogReg		x		x	x		0.5983	0.7233
CNB, XGB, LogReg		x			x		0.5956	0.7164
CNB, XGB, SVC, SGD		x		x	x		0.6841	0.7194

Table 1: Comparaison entre prétraitement et modèles

En analysant le score F1, on peut voir que les méthodes d'apprentissages par ensemble performent mieux que les modèles uniques sur l'ensemble de test privé. De plus, les modèles ensemblistes ont aussi une plus petite variance en nous basant sur la Table 1. Étant donné l'entraînement indépendant de chaque modèle, les valeurs des hyperparamètres dans les ensembles de modèles, sont les mêmes que les hyperparamètres optimaux trouvés dans les modèles individuels. Les modèles de bayésiens priorisent des valeurs arbitrairement petite ou grande de α ce qui nous indique que le lissage des probabilités dépend fortement des subdivisions aléatoires de validation pour la recherche du α optimal. Pour ce qui est des modèles à base d'arbres, ceux-ci priorisent plus d'estimateurs profonds, retournant les bornes maximales pour les hyperparamètres du nombre d'arbre = 500, et profondeur maximale des arbres = 10. Cela nous indique qu'un modèle plus complexe est favorable pour avoir des prédictions plus justes à partir de notre ensemble de données. Pour ce qui est des modèles linéaires (SGD, SVC, régression logistique), on peut observer un biais systématique pour un plus gros poids des termes de régularisations qui témoigne d'une préférence pour pour les modèles plus simples. Voir l'annexe pour des détails sur les valeurs exactes des hyperparamètres en lien avec la performance des modèles. Il est intéressant de noter que malgré la performance souvent plus faible sur l'ensemble de validation, les modèles ensemblistes produisent des résultats moins variables, quel que soit le choix de prétraitement de données, et modèles choisis. Ceci témoigne du lissage des erreurs de prédictions des modèles individuels ce qui diminue la variance des prédictions.

6 Discussion

6.1 Avantages et inconvénients

Un inconvénient de notre méthodologie a été que nous nous sommes lancés sur trop de modèles différents, alors qu'il aurait été plus efficace de se concentrer sur une poignée tels que seulement ceux décrits dans le rapport. La même méthodologie aurait dû être appliquée pour identifier clairement quelles méthodes de prétraitement étaient les meilleures.

Un avantage de notre méthodologie est la robustesse du classifieur par ensemble, car ils ont une plus petite variance que les modèles individuels. Un autre avantage est le travail sur la visualisation des données qui a permis une meilleure compréhension du matériel. Un dernier avantage est la structure du code en orienté objet qui a permis une plus grande facilité à adapter le code rapidement pour tester différentes méthodes de prétraitement (ce qui en a constitué un inconvénient aussi).

6.2 Améliorations futures

Une amélioration future à nos résultats serait de faire rouler notre modèle d'apprentissage par ensemble sur plus d'itérations de recherches d'hyperparamètres. En effet, celui-ci semble pouvoir converger vers un meilleure score F1 plus il y a d'itérations. De plus, comme mentionné précédemment, nous avions accès aux ressources pour lancer des calculs plus lourds avec le GPU ce qui nous aurait aidé. Une autre amélioration possible serait d'identifier plus rigoureusement et exhaustivement quelles méthodes de prétraitement des données sont les plus efficaces au lieu de trop s'éparpiller sur plusieurs différentes.

7 Références

1. Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *The Journal of Machine Learning Research, 13*, 281–305.

2. Brownlee, J. (2020, August 20). *Voting ensembles with Python*. Machine Learning Mastery. Tiré de https://machinelearningmastery.com/voting-ensembles-with-python/

3. Scikit-Learn developers. (2024). *Document classification using 20 newsgroups*. Scikit-Learn. Tiré de https://scikit-learn.org/1.5/auto_examples/text/plot_document_classification_20newsgroups.html

4. NVIDIA. (2024). *RAPIDS cuML*. NVIDIA Corporation. Tiré de https://rapids.ai

5. Bird, S., Klein, E., & Loper, E. (2024). *Natural Language Toolkit (nltk)*. Tiré de https://www.nltk.org

8 Annexes

8.1 Graphiques

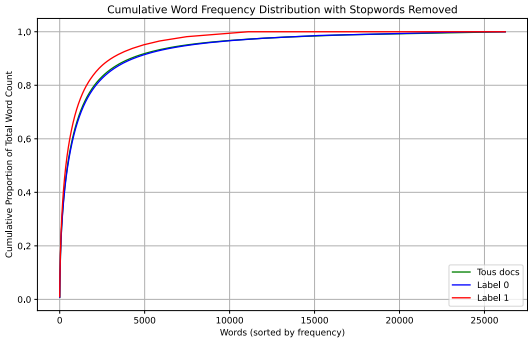

Figure 1: Somme cumulative de fréquences des mots

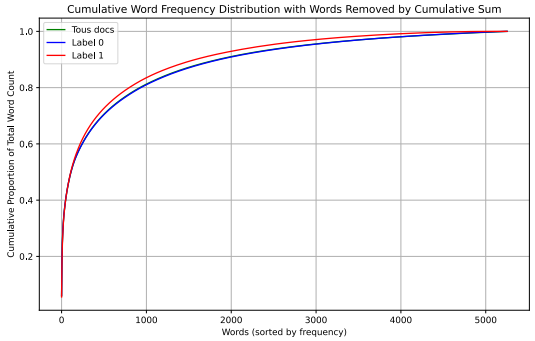

Figure 2: Somme cumulative de fréquences des mots - avec mots vide retirés

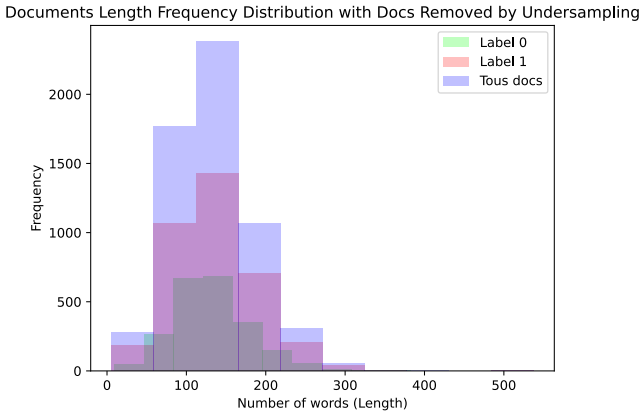

Figure 3: Fréquence de longueur des documents après sous échantillonnage

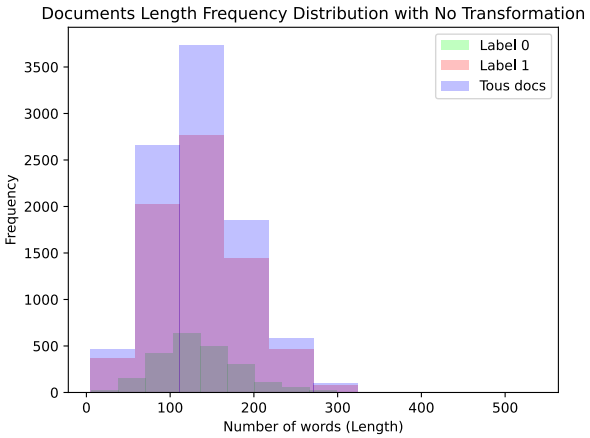

Figure 4: Fréquence de longueur des documents sans transformation

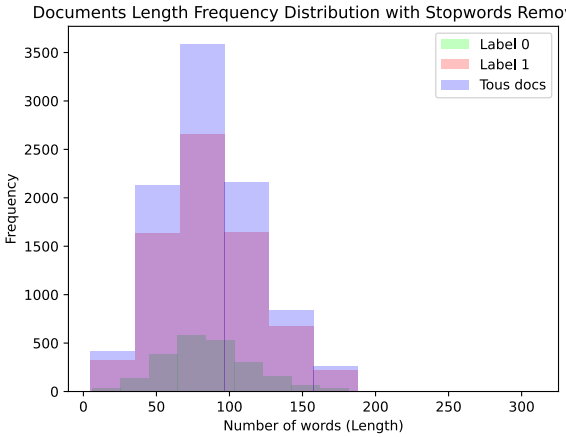

Figure 5: Fréquence de longueur des documents avec mots vides retirés

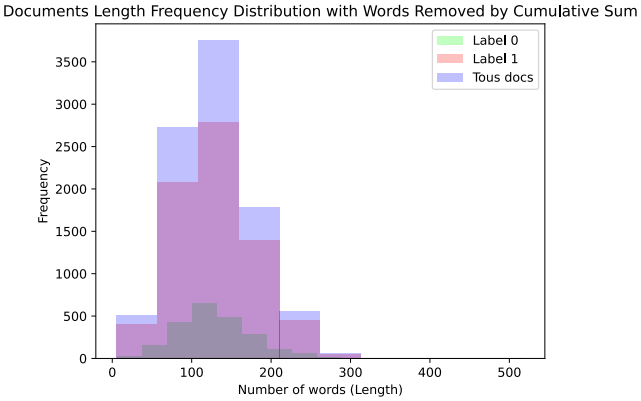

Figure 6: Fréquence de longueur des documents avec fréquences relatives retirées

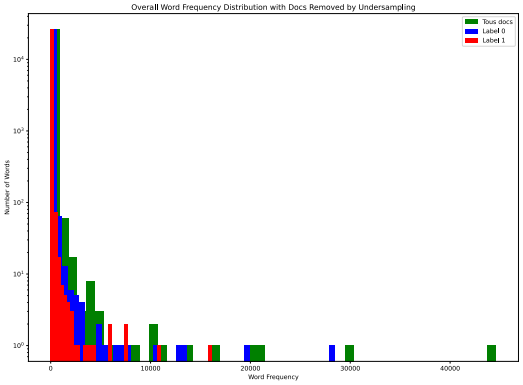

Figure 7: Distributions de fréquences globales des mots avec sous-échantillonnage

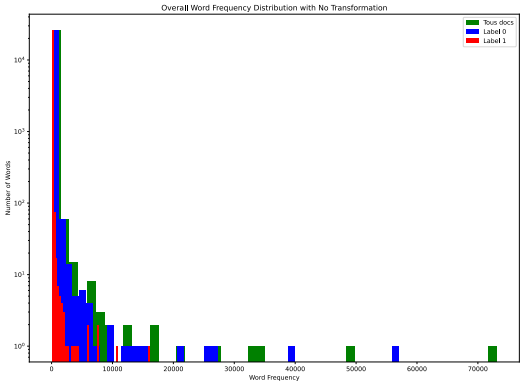

Figure 8: Distributions de fréquences globales des mots sans transformation

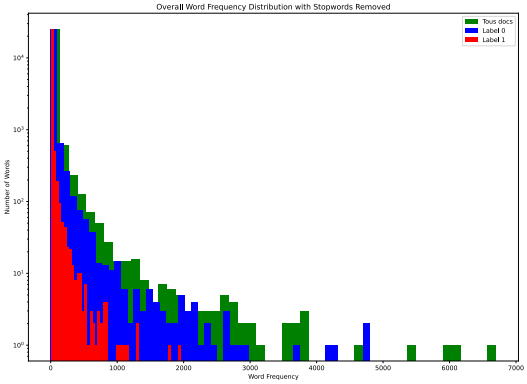

Figure 9: Distributions de fréquences globales des mots avec mots vides retirés

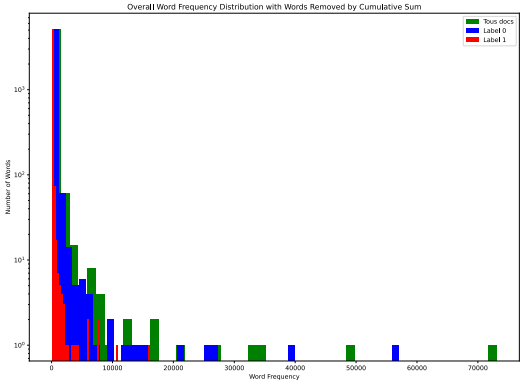

Figure 10: Distributions de fréquences globales des mots avec somme cumulative retirée

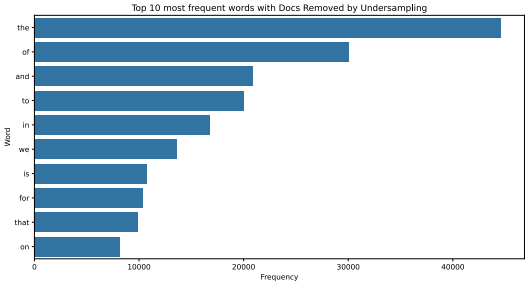

Figure 11: Top mots les plus fréquents après sous échantillonnage

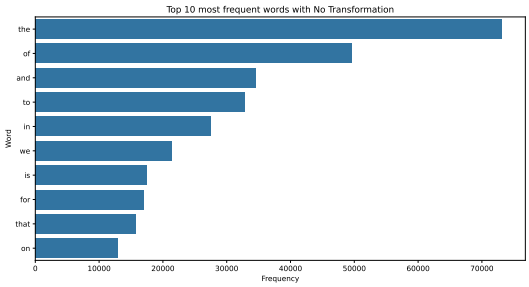

Figure 12: Top 10 mots les plus fréquents sans transformation

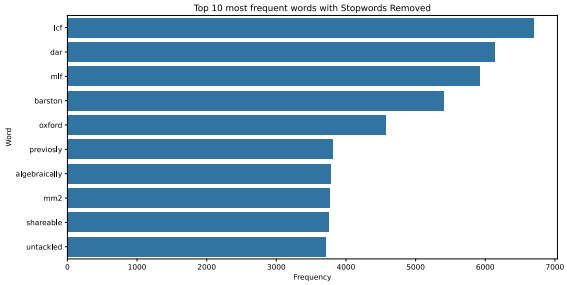

Figure 13: Top 10 mots les plus fréquents avec mots vides retirés

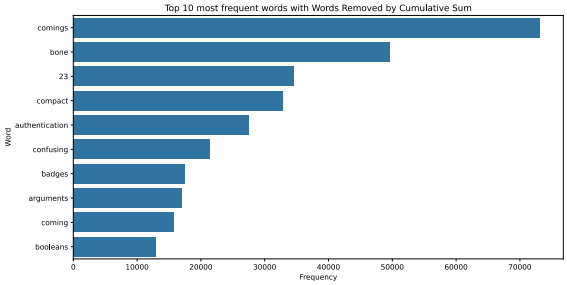

Figure 14: Top 10 mots les plus fréquents retirés par somme cumulative.

8.2 Tables

Model	α	**F1 Val**	**F1 Test**
Bayes naïf vanille	0.45	0.5893	0.7196
CNB	0.0872	0.5611	0.7127
CNB	1.0980	0.5636	0.6324
CNB	0.0872	0.6556	0.6870
CNB	1.4261	0.7041	N/A

Table 2: Comparaison entre prétraitement et valeurs d'hyperparamètres pour Les modèles bayésien

Model	**Coefficient RBF**	**Poids régularisation**	**F1 Val**	**F1 Test**
SVM	0.005	53	0.6415	N/A

Table 3: Comparaison entre prétraitement et valeurs d'hyperparamètres pour SVM

Model	**l1 ratio**	**Poids régularisation**	**F1 Val**	**F1 Test**
SGD avec Huber modifié	0.223	0.01	0.6294	N/A

Table 4: Comparaison entre prétraitement et valeurs d'hyperparamètres pour SVM

Model	**Learning rate**	**Nombre d'arbres**	**Profondeur maximale**	**subsample**	**F1 Val**	**F1 Test**
XGBoost	0.0378	500	10	0.7465	0.6592	0.6629

Table 5: Comparaison entre prétraitement et valeurs d'hyperparamètres pour XGBoost

Model	Poids régularisation	Régularisation	Solveur	F1 Val	F1 Test
LogReg	0.6808	l1	liblinear	0.6308	N/A

Table 6: Comparaison entre prétraitement et valeurs d'hyperparamètres pour régression logistique utilisée dans les modèles ensemblistes

8.3 Algorithmes et fondements mathématiques

8.3.1 XGBoost, perte entropie croisée, et gain de subdivision

Le gain de subdivision (critère pour produire les branchements de l'arbre de décision) est défini par la formule suivante :

$$\text{Gain} = \frac{1}{2}\left(\frac{(\sum \text{gradients}_{\text{gauche}})^2}{\sum \text{Hessians}_{\text{gauche}} + \lambda} + \frac{(\sum \text{gradients}_{\text{droite}})^2}{\sum \text{Hessians}_{\text{droite}} + \lambda} - \frac{(\sum \text{gradients}_{\text{total}})^2}{\sum \text{Hessians}_{\text{total}} + \lambda}\right) - \gamma$$

Avec $\sum$gradients étant la somme des gradients pour le sous-ensmeble à gauche ou droite de la subdivision, Avec $\sum \text{gradients}_{\text{total}}$ somme des gradients pour l'ensemble complet avant la subdivision. $\sum$Hessians est la somme des Hessiens (dérivées secondes). Notons que les gardients et Hessiens sont calculés à partir de la perte d'entropie croisée:

$$\text{Log Loss} = -\frac{1}{N}\sum_{i=1}^{N}(y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i))$$

Avec respectivement la dérivée première pour le gradient et seconde pour l'Hessien.

$$\frac{\partial \text{Log Loss}}{\partial \hat{y}_i} = \frac{\hat{y}_i - y_i}{\hat{y}_i(1 - \hat{y}_i)}$$

$$\frac{\partial^2 \text{Log Loss}}{\partial \hat{y}_i^2} = \frac{1 - y_i}{\hat{y}_i^2} + \frac{y_i}{(1 - \hat{y}_i)^2}$$

Avec $\hat{y}_i$ est la classe prédite et y_i est la vraie classe.

Algorithm 1 Calcul de la fréquence cumulative et filtrage des mots les moins fréquents

Input: Liste des mots *words* et leur fréquence $freqs$, seuil de fréquence cumulative *threshold*
Output: Liste des mots filtrés $filtered\_words$

1. Trier *words* par ordre décroissant de $freqs$.
2. Initialiser $cumulative\_sum \leftarrow 0$ et $filtered\_words \leftarrow []$.
3. Pour chaque indice i de 1 à length(*words*):
   - Calculer $cumulative\_sum \leftarrow cumulative\_sum + freqs[i]$.
   - Si $cumulative\_sum \geq threshold$, alors **arrêter la boucle**.
   - Sinon, ajouter $words[i]$ à $filtered\_words$.
4. Retourner $filtered\_words$.

Algorithm 2 Algorithme de Vote Lisse pour un Classifieur Ensembliste

Input: Un ensemble de modèles $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_k\}$, des poids associés $\{w_1, w_2, \ldots, w_k\}$, un échantillon à classifier x
Output: Classe prédite pour l'échantillon x

1. Pour chaque modèle γ_i dans Γ :
   - Obtenir la probabilité prédite pour chaque classe c: $P(c|x, M_i)$.
2. Pour chaque classe c :
   - Calculer la probabilité agrégée pour la classe c :

$$P(c|x) = \frac{\sum_{i=1}^{k} w_i \cdot P(c|x, M_i)}{\sum_{i=1}^{k} w_i}$$

3. Retourner la classe avec la probabilité agrégée maximale :

$$\hat{y} = \arg\max_{c} P(c|x)$$