Compétition Kaggle - Novembre 2024 : Classification de texte avec un modèle Bag of Words

Thierry POEY - N°20307205

1. Introduction

Le projet donné est une classification de textes, où l'objectif est de concevoir un algorithme capable de trier automatiquement des documents textuels. Les données fournies sont des vecteurs de comptage de mots, avec des étiquettes cibles pour l'entraînement. L'objectif final est d'optimiser la performance du modèle sur un ensemble de tests, mesuré par le macro F1-score.

En combinant plusieurs algorithmes tels que Naïve Bayes, Perceptron et Perceptron Multi-couches (avec une ou deux couches), ainsi que différentes techniques de traitement du texte, j'ai pu observé des performance significatives. Le modèle de baseline a été optimisé par un simple Grid Search. L'utilisation d'une approche bayésienne pour l'optimisation des hyper-paramètres a permis d'atteindre un F1-score de 0.7385 pour le modèle le plus performant selon le score privé de la compétition et 0.73069 sur le modèle sélectionné.

2. Conception des fonctionnalités

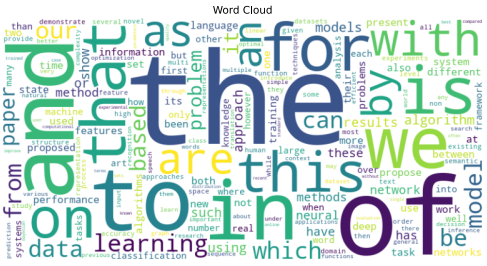

Figure 1. Word cloud des données d'entraînement

Plusieurs techniques de prétraitement et d'extraction ont été appliquées afin de réduire le bruit causé par des caractéristiques non pertinentes. L'objectif visent à créer une représentation optimale des données textuelles tout en réduisant la dimensionnalité. Les résultats observés seront discutés plus en détail dans une section ultérieure.

2.1. Remove stopwords

La première étape du prétraitement consiste à éliminer les mots courants (stopwords) à l'aide de la bibliothèque NLTK. Les mots vides, sont des mots très fréquents (”the”, ”is”, ”and”, etc.). Généralement ils ne portent pas de significations pour la classification. En les supprimants, on concentre le modèle sur les mots les plus pertinents. Grâce au vocabulaire, on peut filtrer les mots correspondant à une liste de stopwords.

2.2. TF-IDF (Term Frequency-Inverse Document Frequency)

Cette méthode pondère chaque terme en fonction de sa fréquence dans le document (TF) et de son inverse de fréquence dans le corpus (IDF). Cela permet de réduire l'importance des mots très fréquents dans l'ensemble des documents tout en conservant leur importance locale. Le TF est calculé en divisant les occurrences d'un mot par le total des mots dans le document, et l'IDF est obtenu en prenant le logarithme du ratio entre le nombre total de documents et le nombre de documents contenant le mot.

2.3. Lemmatisation

La lemmatisation réduit les mots à leur ”forme canonique, ou lemme. Cette méthode se base sur le contexte du mot pour produire une forme valide et unique. Souvent le nom singulier ou le verbe à l'infinitif. Par exemple, les mots ”programs”, ”programming” et ”programmer” seront tous convertis en ”program”. Cette étape permet de normaliser les mots en conservant leur signification. Dans notre approche, j'ai utilisé un lemmatiseur pour transformer chaque mot du vocabulaire en sa forme canonique. Les mots lemmatisés ont ensuite été filtrés pour supprimer les stopwords.

2.4. Stemming

Le stemming est une autre méthode qui consiste à réduire les mots à leur racine ou base commune. Généralement par suppression de leurs suffixes. Le stemming produit souvent des formes de mots qui ne sont pas nécessairement valides, mais qui partagent une racine commune. Par exemple, ”fishing”, ”fished” et ”fisher” seraient tous réduits à ”fish”. Cette

technique est plus rapide que la lemmatisation mais est plus approximative.

2.5. Lemmatisation vs Steeming

L'utilisation de la lemmatisation et du stemming permet de tester différentes représentations des données textuelles. La lemmatisation tend à conserver plus de précision et de sens. Tandis que le stemming offre une simplification rapide. Selon les exigences de la tâche de classification, l'une ou l'autre des méthodes, peut être avantageuse pour maximiser les performances du modèle tout en réduisant la complexité des données.

2.6. Suppression des mots à faible et forte fréquence

Pour réduire le bruit et améliorer la capacité de généralisation du modèle, on applique un filtrage basé sur les fréquences. Les mots ayant une différence d'occurrence faible ou élevée entre les classes sont supprimés. Ainsi, plus un mot est courant dans un document et moins dans l'autre, plus il a un score de fréquence élevé.

La suppression à basse fréquence (low threshold) supprime les mots les moins courants dans les deux documents et ceux qui sont utilisés de manière neutre sans distinctions. L'avantage de cette méthode est qu'elle ne repose pas directement sur le nombre absolu d'occurrences d'un mot, mais plutôt sur sa rareté relative dans le corpus.

D'autre part, la suppression à haute fréquence (high threshold) permet de supprimer les mots ayant trop d'importance pour distinguer les classes pouvant conduire à une mauvaise compréhension pour la classification d'un document.

Ces approches permettent d'éliminer les mots peu informatifs ou omniprésents.

3. Algorithmes

3.1. Naïve Bayes

Le modèle Naïve Bayes exploite le théorème de Bayes pour estimer la probabilité $P(y|x)$ d'une classe y donnée un vecteur de compteur de mots $x = (x_1, x_2, \ldots, x_n)$ tel que :

$$P(y|x) \propto P(y) \prod_{i=1}^{n} P(x_i|y),$$

En raison de l'hypothèse d'indépendance conditionnelle, le modèle reste efficace même avec un grand nombre de dimensions, d'où mon premier choix d'algorithme.

L'optimisation bayésienne des hyper-paramètres a été utilisée pour ajuster α et d'autres paramètres, maximisant ainsi la performance sur les données de validation de manière échantillonnée.

3.2. Perceptron

Le Perceptron cherche à trouver un hyperplan qui sépare linéairement les classes. L'algorithme ajuste les poids en minimisant une fonction de coût basée sur les erreurs de classification. Cet algorithme, bien que simple, arrive à distinguer les représentations de texte pondérées.

3.3. Multi-Layer Perceptron (MLP)

Un Perceptron multi-couches (MLP) ajoute de la non linéarité en introduisant des couches cachées. L'entraînement du MLP utilise la rétropropagation de l'erreur pour ajuster les poids :

$$\nabla w_i \leftarrow \nabla w_i - \eta \frac{\partial \mathcal{L}}{\partial w_i}$$

où $\mathcal{L}$ est la fonction de perte d'entropie croisée binaire. Le dropout et la régularisation L_2 ($\lambda|w||^2$) ont été intégrés, pénalisant les poids excessivement larges.

4. Méthodologie

J'aborderais ici les choix adoptés pour l'entraînement du modèle. Notamment la répartition des données, les stratégies de régularisation, les optimiseurs, les hyper-paramètres sélectionnés et l'optimisation bayésienne.

4.1. Répartition des données

Pour commencer, j'ai utilisé une stratégie de validation croisée en k-folds pour maximiser la robustesse des évaluations du modèle. Cette méthode divise les données en k sous-ensembles ou folds, avec chaque sous-ensemble prenant à tour de rôle le rôle de l'ensemble de validation pendant que les autres sont utilisés pour l'entraînement. Cela permet de limiter le biais lors de l'échantillonnage. La validation finale a été réalisée sur un ensemble indépendant pour évaluer de manière globale la performance du modèle.

4.2. Régularisation

Les réseaux de neurones ont tendance à trop se spécialiser. Pour contrôler l'overfitting, deux stratégies de régularisation principales ont été appliquées: le dropout et la régularisation L_2. Le dropout consiste à désactiver de manière aléatoire un certain pourcentage de neurones à chaque étape d'entraînement. Sinon le réseaux devient trop dépendant de certaines connexions spécifiques et favorise la généralisation. La régularisation L_2 pénalise les poids excessifs en ajoutant une contrainte à la fonction de perte, ce qui encourage le modèle à rechercher des solutions plus simples et moins sensibles aux variations des données.

4.3. Optimiseurs

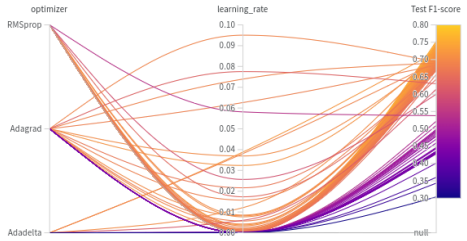

Figure 2. Comparaison des optimisateurs sur Wandb.

Parmi le choix des optimiseurs, j'ai pu exploré l'utilisation de plusieurs algorithmes pour ajuster les poids du modèle. L'optimiseur RMSprop a été majoritairement utilisé en raison de sa capacité à adapter les taux d'apprentissage. Cette caractéristique lui permet de converger rapidement et de manière stable. D'autres optimiseurs ont également été testés, tels que Adam qui converge rapidement mais peut aussi entraîner de fortes variation. Adelta, qui adapte des mises à jour dynamique pour les gradients de différentes échelles mais complexe à optimiser. J'ai également exploré Adagrad, qui ajuste le taux d'apprentissage en fonction de la fréquence d'apparition des gradients pour chaque paramètre. C'est une bonne chose dans de rares scénarios mais il peut diminuer de manière excessive le taux d'apprentissage à long terme. Enfin la descente de gradient stochastique (SGD) a également été utilisée comme base pour construire le réseau de neurone. Avec un ajustement plus fréquent du taux d'apprentissage mais une convergence parfois plus lente se trouvant trop facileement bloqué à un saddle point ou local.

Pour optimiser la performance du modèle, une descente par mini-lots a été adoptée, ce qui permet de mettre à jour les gradients par petits groupes d'échantillons. Un bon équilibre entre la précision des mises à jour et l'efficacité de l'exécution. De plus, un taux d'apprentissage adaptatif a également été mis en place à la fin de chaque epoch avec une réduction automatique lorsque la convergence ralentissait. Enfin la mise en place du early stopping afin de stabiliser les résultats et prévenir l'overfitting.

4.4. Laplace smoothing et logarithme

Le smoothing de Laplace était nécessaire pour le modèle Naïve Bayes. Lorsque nous travaillons avec un texte, il est courant que certains mots soient absents dans une classe donnée. Sans smoothing, si un mot n'apparaît jamais dans les exemples d'une classe, la probabilité conditionnelle associée à ce mot devient nulle, ce qui fausse complètement la prédiction.

En ajoutant une petite valeur $]0, 1]$ à chaque compte de mot, on contourne le problème. Cela garantit que chaque mot a une probabilité minimale non nulle, même s'il est absent dans les exemples de la classe. Ce simple ajustement évite que les prédictions soient dominées par l'absence de certains mots rares ou spécifiques.

D'autre part, lors de la prédiction le modèle utilise la formule suivante pour estimer la probabilité logarithmique d'appartenance de une input à chaque classe :

$$\log P(Y|X) = \log P(Y) + \sum \log P(X_i|Y) \times X_i$$

Cette technique évite les problèmes d'underflow lorsque les probabilités deviennent très petites. Le produit des probabilités est transformé en une somme de logarithmes, ce qui est numériquement plus stable.

4.5. Sélection des hyper-paramètres

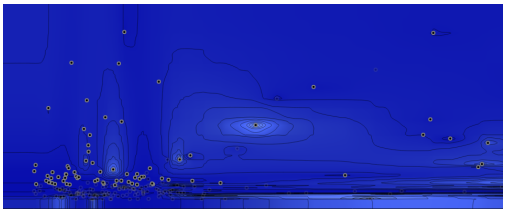

Figure 3. Contour plot - Affinage du terme de régularisation en fonction du taux d'apprentissage.

La plage de valeure est crucial pour le choix des hyper-paramètres. Des schémas comme la Figure 3 aide à comprendre dans quelle plage de valeurs je peux considérer mes hyper-paramètres.

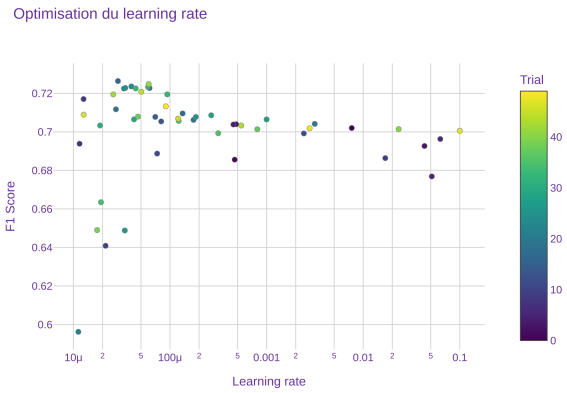

Figure 4. Scatter plot - Observation du taux d'apprentissage en fonction du F1-score

Le taux d'apprentissage a été sélectionné dans une échelle logarithmique car ela permet d'explorer plus facilement

les tendances (voir Figure 4). La régularisation des poids (*weight decay*) a été ajustée dans l'intervalle $[10^{-5}, 10^{-2}]$ pour réduire l'impact de la régularisation L_2. Enfin, le nombre d'époques a été déterminé entre 20 et 60 pour que le modèle atteigne la convergence sans overfit.

Pour gérer le déséquilibre des classes, un poids a été introduit, avec des valeurs entre 1 et 4 afin de jauger l'importance des classes minoritaires pendant l'entraînement. En complément, un seuil de classification (*infer_threshold*) a été ajusté entre 0.1 et 0.5 pour mieux traiter les prédictions de la classe minoritaire.

Le filtrage des fréquences des mots a également été pris en compte, avec des seuils de filtrage pour les fréquences basses (*low_frequency*) et hautes (*high_frequency*) définis entre 0 et 10. Les résultats de mes expériences, prouvaient systématiquement que ce filtre apportait de meilleurs résultats quelque soit le modèle.

Pour des modèles plus complexes comme le Perceptron Multi-couches, des paramètres spécifiques ont été optimisés, notamment la taille de(s) hidden layer(s). La première couche (variant entre 512 et 2048) permet de capturer les détails. La seconde (entre 128 et 320) réduit la complexité et garde les informations importantes. L'optimiseur choisit est RMSprop en raison de la discussion précédente.

4.6. Optimisation bayésienne

Pour affiner les hyper-paramètres du modèle de manière efficace, j'ai adopté une stratégie d'optimisation par recherche bayésienne. Contrairement à la recherche aléatoire ou à la recherche par grille, cette méthode s'appuie sur un modèle probabiliste qui ajuste les hyper-paramètres de manière itérative en utilisant les résultats des expérimentations précédentes.

L'optimisation des hyper-paramètres a été réalisée en utilisant la libraire *optuna*.

5. Résultats

Les résultats ont été obtenus en utilisant différentes méthodes de prétraitement, des configurations d'hyper-paramètres et des algorithmes de classification. L'analyse inclut une comparaison entre le Naïve Bayes, le Perceptron, et le MLP avec une ou plusieurs couches cachées sur 50 essaies par recherche bayésienne. Sera présenté en premier les performance mesurée par le F1-score sur les modèles finaux.

5.1. Comparaison des Modèles et des Scores F1

Résultats finaux : Les résultats finaux indiquent bien que les modèles ont réussi à généraliser sur l'ensemble de test (Kaggle). Cela démontre bien la robustesse de l'entraînement.

Table 1. Performance Finale des modèles de classification

Modèle	F1-score (Validation)	F1-score (Kaggle)
Naïve Bayes	0.72461	0.72097
Perceptron	0.7191	
MLP (1 couche cachée)	0.73273	0.73069
MLP (2 couches cachées)	0.74280	0.7385

Table 2. Performance des modèles de classification selon les méthodes de prétraitement sur 50 trials

Modèle	Prétraitement	F1-score (Validation)
Naïve Bayes		0.724193
Naïve Bayes	Lemmatise	0.721159
Naïve Bayes	Steeming	0.720138
Naïve Bayes	Stopwords	0.718657
Naïve Bayes	TF-IDF	0.718560
Perceptron		0.715531
Perceptron	Lemmatise	0.718222
Perceptron	Steeming	0.709050
Perceptron	Stopwords	0.699555
Perceptron	TF-IDF	0.713473
MLP (1 couche)		0.708952
MLP (1 couche)	Lemmatise	0.706886
MLP (1 couche)	Steeming	0.709570
MLP (1 couche)	Stopwords	0.713269
MLP (1 couche)	TF-IDF	0.716383
MLP (2 couches)		0.707226
MLP (2 couches)	Lemmatise	0.699637
MLP (2 couches)	Steeming	0.718749
MLP (2 couches)	Stopwords	0.709652
MLP (2 couches)	TF-IDF	0.728049

L'analyse des résultats obtenus pour les différents modèles de classification (Naïve Bayes, Perceptron, et MLP avec 1 ou 2 couches cachées) en fonction des méthodes de prétraitement appliquées révèle plusieurs tendances intéressantes. Nous analyserons ces résultats dans les prochaines sous-sections.

Naïve Bayes

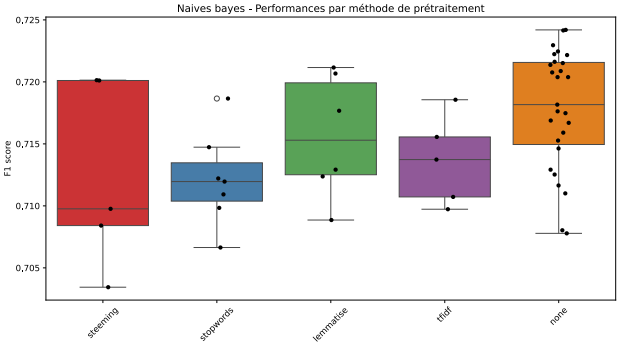

Figure 5. Effet du prétraitement sur les performances de Naïves Bayes

Le modèle Naïve Bayes présente des performances relativement stables, mais son efficacité semble être modérée par le choix du prétraitement, voir Figure 6. Les prétraitements semble impacter négativement le F1 score.

Perceptron

Le Perceptron, pour un modèle linéaire montre de bonnes performances et semble être sensible au prétraitement, en particulier à TF-IDF avec un F1 score de 0.713. Comme le Perceptron tire parti de la pondération des mots, cela semble lui permettre de mieux capturer les relations entre les mots dans les documents. En revanche, l'utilisation de la méthode stopwords diminue les performances du modèle, ce qui suggère que la suppression des mots fréquents ne soit pas bénéfique dans ce cas. Possiblement parce que ces mots contiennent encore des informations utiles pour la classification.

MLP

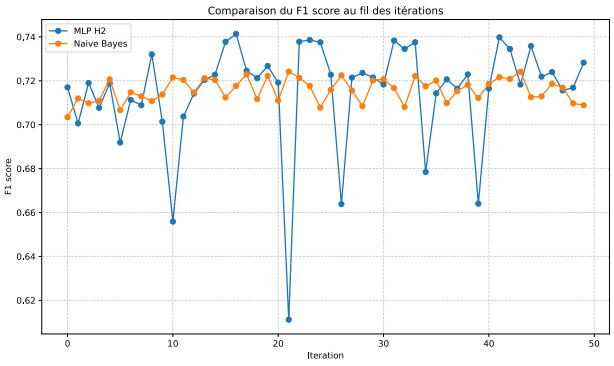

Figure 6. Comparaison des scores d'un MLP contre Naïves Bayes.

Les MLP (Multi-Layer Perceptron) avec 1 et 2 couches cachées présentent des comportements similaires. Avec l'inclusion d'une couche d'activation ReLU, on introduit de la non-linéarités, ce qui améliore sa capacité à apprendre des représentations plus complexes. Néanmoins le prétraitement par TF-IDF reste le plus performant. Le modèle avec 2 couches cachées atteint un F1-score de 0.728, légèrement supérieur à celui du Perceptron avec TF-IDF. Les autres méthodes de prétraitement, comme les stopwords et le stemming, semblent avoir un effet modéré, avec des scores proches de 0.71 à 0.72. Et donc que ces transformations n'apportent pas de gain majeur au-delà de la simple utilisation de TF-IDF.

Synthèses

En résumé, les résultats montrent clairement que TF-IDF est la méthode de prétraitement la plus bénéfique, en particulier pour des modèles comme le Perceptron et le MLP. Le Naïve Bayes, en revanche, semble moins influencé par la méthode de prétraitement. Le MLP atteint des meilleures

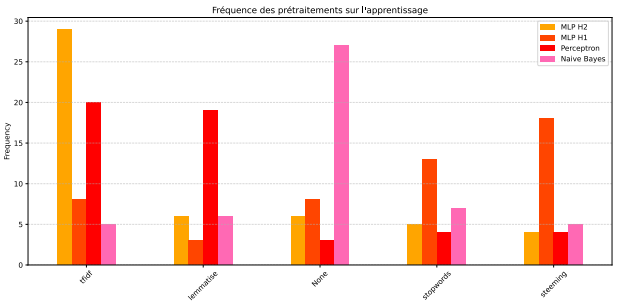

Figure 7. Comparaison des techniques de prétraitement appliquées aux modèles.

performances, grâce à une meilleure généralisation et l'ajout de couches dropout.

5.2. Équilibrage des Classes

Figure 8. Impact du déséquilibre des classes sur les performances du modèle.

L'un des défis majeurs pour cette classification de textes était le déséquilibre des classes. La Figure 8 montre l'impact de l'équilibrage des classes sur les performances des modèles. Pour faire face à ce problème, des poids spécifiques ont été attribués aux classes minoritaires, ce qui a permis d'améliorer significativement le F1-score.

5.3. Smoothing dans le Modèle Naïve Bayes

Pour le modèle Naïve Bayes, le smoothing a été utilisé pour traiter les données textuelles. La Figure 9 illustre l'impact du paramètre de lissage sur le F1-score. Ce lissage permet d'éviter les probabilités nulles en attribuant une faible probabilité aux mots absents dans les exemples d'une classe. Un lissage proche de 0 accorde souvent de meilleurs résultats.

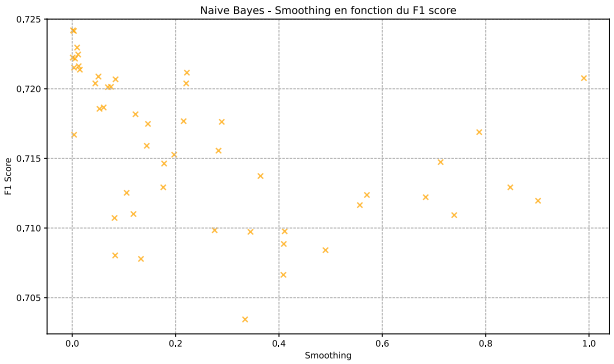

Figure 9. Impact du smoothing sur le F1-score du modèle Naïve Bayes.

5.4. Corrélation et Sélection des Hyper-paramètres

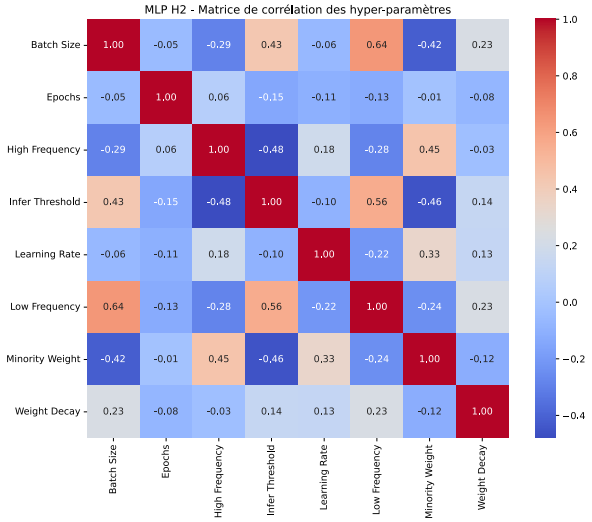

Figure 10. Corrélation des hyper-paramètres du MLP (2 couches cachées).

La Figure 10 est une matrice de corrélation entre les différents hyper-paramètres du MLP (2 couches cachées). Cette représentation a permis d'identifier les relations clés entre les hyper-paramètres.

Par exemple :

- La corrélation négative entre la taille des lots (*Batch Size*) et le poids de la classe minoritaire (*Minority Weight*) indique que lorsque la taille des lots augmente, cela atténue l'effet du poids accordé à la classe minoritaire.
- Par ailleurs, les interactions du seuil de classification (*infer threshold*) avec les fréquences élevées et basses montrent que la gestion des mots fréquents et rares influence directement les décisions du modèle.

6. Discussion et Perspectives

Au cours de ce projet, plusieurs approches ont été testé pour optimiser les performances des modèles. Parmi elles, l'utilisation d'AutoML et de NEAP a été envisagée pour trouver la meilleure structure de réseau de neurone. Cependant, ces solutions se sont avérées complexes à maîtriser à cause de la haute dimensionnalité des données, des hyper-paramètres à ajuster et surtout, de mes connaissances sur le deep learning...

En parallèle, j'ai pu tester BERT. Malgré sa popularité, cette méthode n'a pas permis d'améliorer les performances de manière significative par rapport aux autres modèles. Cela peut s'expliquer par la spécificité des données textuelles utilisées.

En revanche, l'approche retenue, centrée sur le MLP avec des stratégies de régularisation et une optimisation bayésienne est un bon compromis entre performance et simplicité. Cependant, elle reste sensible aux données déséquilibrées et une complexité accrue de l'entraînement en cas de couches supplémentaires.

Si je devais améliorer ce projet, je pense qu'il serait préférable de rechercher une architecture de réseau neuronal spécifiquement adaptée aux données uniques de ce projet. Une amélioration du traitement du texte comme la détection des émotions d'une classe à travers les mots utilisés, puis attribuer un score à chaque classe en conséquence. Par ailleurs, l'utilisation de données synthétiques pourrait être une piste intéressante à explorer et pourrait potentiellement améliorer la robustesse et la généralisation du modèle.

7. Références

https://www.analyticsvidhya.com/blog/2021/06/text-preprocessing-in-nlp-with-python-codes/

https://www.youtube.com/watch?v=NE88eqLngkgt=417sab$_{c}hannel = DeepBean$

https://www.kaggle.com/code/vbmokin/nlp-eda-bag-of-words-tf-idf-glove-bert

https://medium.com/analytics-vidhya/fundamentals-of-bag-of-words-and-tf-idf-9846d301ff22