Rapport d’analyse

Devoir #2

IFT3335

Introduction à l’intelligence artificielle

Prof. Jian-Yun Nie

Université de Montréal

Nahomy Durand (20201414)

Louis Forrer (20170329)

Remis le

19 avril 2024

Introduction

Dans le cadre du cours d'introduction à l'intelligence artificielle, nous avons créé un programme en python qui prends en entrée des textes de *tweets* et déclare si ceux-ci sont offensifs ou non grâce à l'utilisation de la bibliothèque *sklearn* et ses modèles d'apprentissages automatiques. Nous avons testé les algorithmes suivants avec les paramètres suivants :

1. Naive Bayes Multinomial: 'alpha': [0.5, 1.0, 2.0, 5.0, 10.0] ainsi que le Naive Bayes Gaussien.
2. Arbre de décision : 'max_depth': [None, 10, 20, 30, 40, 50], 'criterion': ['gini', 'entropy']
3. Forêt aléatoire : 'n_estimators': [10, 50, 100], 'max_depth': [None, 10, 20], 'criterion': ['gini', 'entropy']
4. SVM : 'kernel': ['linear', 'rbf'], 'C': [0.1, 1, 10, 100]
5. MultiLayerPerceptron : 'hidden_layer_sizes': [(50,), (100,)], 'activation': ['logistic', 'relu'], solver': ['adam', 'sgd', 'lbfgs']

Pour chaque modèle possédant plus qu'un type de paramètre à tester, notre programme teste toutes les combinaisons possibles de ces paramètres pour avoir le meilleur modèle donnant le meilleur résultat. Lorsque le programme est lancé, il nous est informé quelles combinaisons de paramètres est entraîné et testé. Pour chaque algorithme, les résultats finaux avec les meilleurs paramètres nous sont donnés pour la précision, le rappel, le score-F1 et le support pour chaque algorithme:

La **précision** mesure le nombre de tweets identifiés par le modèle comme positifs (OFF ou NOT) qui le sont réellement.

- Des tweets que le modèle a identifiés comme OFFENSIF, c'est le pourcentage de ceux qui le sont réellement (et même chose avec les non-offensifs).
- Une précision élevée signifie que le modèle renvoie plus de résultats pertinents que de résultats non pertinents.

Le **rappel** (*recall*), quant à lui, mesure le nombre d'éléments (OFF ou NOT) correctement identifiés par le modèle.

- Pourcentage des tweets OFFENSIFS qui ont été correctement identifié par le modèle (et même chose avec les non-offensifs).
- Un taux de rappel élevé signifie que le modèle récupère la plupart des éléments pertinents.

Le **score-F1** est un score représentant les résultats de la précision et du rappel en un. Cela nous donne une vue globale de l'exactitude générale du modèle.

Le **support** indique la taille de l'ensemble de données pour chaque classe et peut vous aider à comprendre la pertinence des autres mesures. Dans ce cas, puisque nous utilisons le même *dataset* d'entrainement ainsi que le même *dataset* de test, les nombres ne fluctuent pas.

** Nous avons deux *files* différents à lancer : le file main.py pour les algorithmes principaux ci-dessus et le file bert.py pour notre modèle BERT. Dans le file main.py, on y retrouve une section pour les paramètres des algorithmes. Celle-ci peut être modifié au besoin de l'utilisateur. **

Résultats

Naive Bayes

	Precision	Recall	F1-Score	Support
NOT	0.72	0.97	0.82	2639
OFF	0.79	0.25	0.38	1333
Accuracy			0.73	3972

Naive Bayes Best Params: {'alpha': 0.5}

Decision Tree

	Precision	Recall	F1-Score	Support
NOT	0.77	0.81	0.79	2639
OFF	0.57	0.51	0.54	1333
Accuracy			0.71	3972

Decision Tree Best Params: {'criterion': 'gini', 'max_depth': None}

Random Forest

	Precision	Recall	F1-Score	Support
NOT	0.75	0.94	0.84	2639
OFF	0.78	0.39	0.52	1333
Accuracy			0.76	3972

Random Forest Best Params: {'max_depth': None, 'n_estimators': 100}

SVM

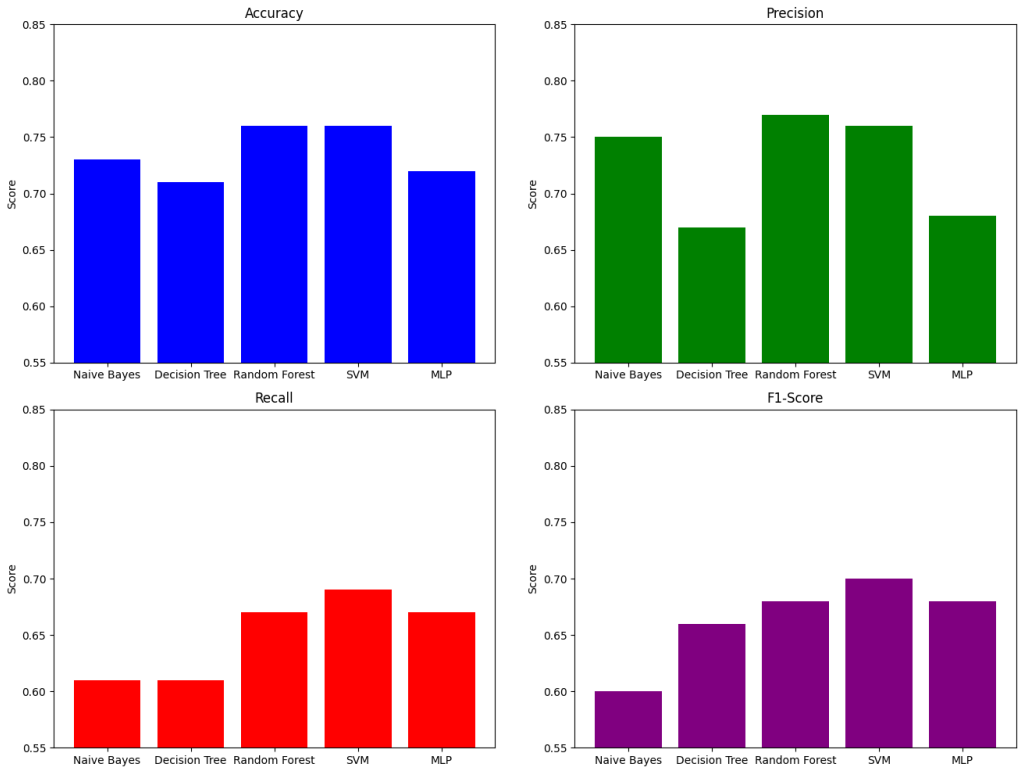
	Precision	Recall	F1-Score	Support
NOT	0.77	0.92	0.84	2639
OFF	0.75	0.45	0.56	1333
Accuracy			0.76	3972

SVM Best Params: {'C': 1, 'kernel': 'linear'}

Multilayer Perceptron (max iter = 200)

	Precision	Recall	F1-Score	Support
NOT	0.77	0.81	0.79	2639
OFF	0.59	0.53	0.56	1333
Accuracy			0.72	3972

Multilayer Perceptron Best Params: {'activation': 'logistic', 'hidden_layer_sizes': (100,), 'solver': 'lbfgs'}



Analyse des resultats :

Naive Bayes Multinomiale

Cet algorithme possède un score-F1 général de 0.73. Si nous regardons les résultats de plus près, nous pouvons voir que Naives Bayes a extrêmement de difficulté à bien identifié les *tweets* qui sont offensifs (25% pour son score de rappel). Ses résultats pour la classe non-offensifs sont donc plus haut puisqu'il note la plupart des données comme non-offensifs. Cependant, du peu de *tweets* que le modèle identifie comme offensif, il est correct 79% du temps. On peut aussi prendre en note que cet algorithme est aussi le plus rapide des 5 algorithmes d'apprentissages automatiques testés (techniquement, nous énumérons les algorithmes en ordre d'ascension sur leur temps).

Decision Tree

L'arbre de décision présente un score général de 0.71. Le modèle montre une performance plus équilibrée entre les classes NOT et OFF que le Naive Bayes, avec des scores de précision, de rappel et de F1-Score légèrement inférieurs pour la classe NOT mais nettement meilleurs pour la classe OFF. Cela suggère une meilleure capacité à différencier les tweets offensifs des non-offensifs sans pencher excessivement vers une classification conservatrice.

Random Forest

Le modèle de *Random Forest* affiche la meilleure performance générale avec une précision de 0.76. Il excelle particulièrement pour la classe NOT avec un excellent rappel de 0.94, indiquant sa capacité à identifier correctement la majorité des tweets non offensifs. Toutefois, bien que sa précision soit relativement élevée, le modèle montre toujours une présence notable de faux positifs comme le suggère sa précision de 0.75. Pour la classe OFF, il y a une nette amélioration par rapport au Naive Bayes, mais avec un rappel encore modeste de 0.39, de nombreux tweets offensifs restent non détectés.

SVM

Le SVM, avec un score général équivalent à celle de *Random Forest*, offre un meilleur équilibre entre précision et rappel pour la classe OFF. Cela indique une capacité améliorée à identifier correctement les *tweets* offensifs tout en minimisant les faux positifs et négatifs. Il montre donc des résultats plus justes que les algorithmes précédents. Le SVM prends, cependant, plus de temps que les 3 derniers algorithmes à rouler.

Multilayer Perceptron

Le Perceptron Multicouches montre un score général de 0.72 (à noter qu'il est plus faible que le SVM et le *Random Forest*). En examinant de plus près, on observe que ce modèle a plus de facilité à identifier un *tweet* comme offensif (rappel de 0.53). Cependant, seulement 59% de ces derniers le sont réellement, ce qui établit un faux-positif assez élevé pour la classe NOT. Il est aussi important de noter que cet algorithme a pris beaucoup plus longtemps à rouler. Pour notre programme, nous avons eu les résultats finaux après 45 mins d'attente– le tout pour des résultats assez médiocres.

Conclusions Générales

Le *Random Forest* et *SVM* présentent les meilleures performances générales, avec un léger avantage pour SVM en termes d'équilibre entre les classes. Nous pouvons clairement remarquer que la classe OFF est systématiquement plus difficile à prédire correctement pour tous les modèles. Le choix du meilleur modèle dépend de l'importance relative de la précision, du rappel, du F1-score et du temps d'entrainement du modèle pour la tâche spécifique.

Mise en place de notre méthode BERT :

Nous avons débuté notre projet en important les bibliothèques nécessaires, notamment *transformers* pour utiliser BERT et PyTorch pour le traitement des tensors et l'entraînement de notre modèle. Les données ont été chargé à partir d'un fichier .tsv, puis divisées en ensembles d'entraînement et de validation. Nous avons séparé les textes des étiquettes et converti ces dernières en valeurs numériques pour faciliter le traitement. Nous avons ensuite créé un objet *Dataset* personnalisé pour encapsuler les tweets et leurs étiquettes associées. Chaque tweet a été *tokenisé*, puis tronqué ou complété pour atteindre une longueur maximale avant d'être transformé en tensor adapté à l'utilisation par BERT.

Le modèle BertForSequenceClassification a été chargé avec la configuration bert-base-uncased, choisie spécifiquement pour la classification binaire. Dans notre cas, les classes sont définies comme

suit: "offensif" est marqué par 1 (classe 1) et "non-offensif" par 0 (classe 0). Cette distinction nous permet d'aborder le problème sous forme de classification binaire, où chaque tweet est classifié comme étant soit offensif soit non-offensif.

Nous avons développé une fonction **train_epoch** pour entraîner le modèle sur les données d'entraînement à travers plusieurs époques. Une époque correspond à une phase complète sur l'ensemble des données d'entraînement. Durant l'entraînement, **train_loss** (perte d'entraînement) est calculée pour évaluer à quel point la prédiction du modèle diverge de la réalité, aidant à ajuster les poids du modèle pour minimiser cette erreur.

Pour évaluer le modèle, nous avons utilisé la fonction **eval_model**. Cette évaluation nous permet de mesurer la **eval_loss** (perte de validation) et d'autres métriques importantes comme la précision sur l'ensemble de validation. Ces mesures aident à comprendre les performances du modèle et à vérifier s'il généralise bien sur des données non vues pendant l'entraînement.

L'optimiseur *AdamW* a été utilisé pour ajuster les poids du modèle. *AdamW* est une extension de l'algorithme Adam, optimisée pour mieux gérer les poids faiblement régularisés, améliorant ainsi la convergence de l'apprentissage. Un *scheduler*, ou *planificateur*, a été configuré pour ajuster le taux d'apprentissage au fil du temps, optimisant ainsi l'efficacité de l'entraînement. Le planificateur réduit le taux d'apprentissage selon un plan prédéfini, ce qui est souvent crucial pour améliorer les performances des modèles sur des tâches de classification complexes.

Durant tout le processus, nous avons géré divers avertissements et erreurs, notamment ceux liés à l'initialisation des poids et aux particularités de l'implémentation de *AdamW*. Ces étapes garantissent que notre modèle est correctement entraîné, efficace et prêt à être utilisé pour des prédictions précises.

Résultats bruts de notre méthode BERT :

Epoch 1/3

Train Loss	0.471900705083104	Accuracy	0.777945619335347
Val Loss	0.424646485821310	Accuracy	0.799093655589123

Epoch 2/3

Train Loss	0.345129449990571	Accuracy	0.857922121517287
Val Loss	0.446292483842516	Accuracy	0. 81797583081571

Epoch 3/3

Train Loss	0.254860597931398	Accuracy	0.907687143336690
Val Loss	0.573896911758256	Accuracy	0.807401812688821

	Precision	Recall	F1-Score	Support
Class 0	0.73	0.73	0.73	472
Class 1	0.85	0.85	0.85	852
Accuracy			0.71	1324

Analyse des résultats de notre méthode BERT :

Analyse des Résultats par *epoch*:

À chaque *epoch*, la perte diminue et la précision augmente pour l'ensemble d'entraînement, indiquant une amélioration du modèle. La perte de validation augmente entre l'*epoch 2* et 3, tandis que la perte d'entraînement continue de diminuer, suggérant un potentiel début de surajustement.

Conclusions:

Le modèle apprend efficacement de l'ensemble d'entraînement, comme en témoignent la réduction de la perte et l'augmentation de la précision au fil des *epochs*. Cependant, on peut observer certain signe de surajustements, se traduisant à accorder une attention particulière dans les itérations d'entraînement futures.

Les performances varient entre les classes, ce qui indique une possible nécessité d'équilibrer davantage l'entraînement ou d'appliquer des techniques spécifiques pour améliorer la classification des classes moins performantes. Des stratégies comme l'*early stopping*, l'ajustement des hyperparamètres ou l'introduction de méthodes de régularisation peuvent être envisagées pour optimiser davantage le modèle.