Rapport : Classification de commentaires Reddit

IFT-3395, Automne 2019

Présenté à :
Prof. Guillaume Rabusseau

Équipe : David Raby + Yifu Zhou
David Raby-Pepin, 918119
Yifu Zhou, 1163986 (cours abandonné récemment)

Introduction

L'objectif de cette compétition était de bâtir plusieurs modèles d'apprentissage machine prédictifs afin de classifier le mieux possible des messages textes entre 20 sujets différents, ainsi qu'identifier les modèles les plus performants. Voici un résumé de l'approche utilisée afin d'accomplir notre objectif :

Étape 1 : implémentation d'un classifieur de Bayes naïf avec traits caractéristiques "sacs de mots"

Nous avons premièrement fait un prétraitement des messages textes (documents) afin de les simplifier et de sous-ligner les redondances implicites entre les termes contenu dans le corpus. Ce prétraitement inclut l'élimination des ''stop words'' en anglais et des termes non-alphabétiques, ainsi que la lemmatization et le stemming de tous les mots.

À partir des messages textes (documents) prétraités, nous avons ensuite généré les traits caractéristiques utilisés en entrée dans notre modèle de Bayes naïf selon la méthode ''sacs de mots'' pour laquelle nous avons normalisé les fréquences des termes dans chaque document par la fréquence du terme le plus commun du document afin d'obtenir un ratio des fréquences plutôt qu'un simple compte.

Nous avons ensuite implémenté un classifieur Bayes naïf selon la méthode vue en classe, mais utilisant le lissage de Laplace plutôt qu'une distribution gaussienne afin d'estimer la vraisemblance. Nous avons testé l'algorithme de façon séquentielle avec différentes valeurs α (terme de lissage) afin de trouver la valeur qui générait les prédictions les plus exactes : α=0.1. Cette approche nous a permis de dépasser les 3 références grisées sur Kaggle avec un score final de 0.55977 sur l'ensemble de test.

Étape 2 : implémentation des méthodes de notre choix

Le prétraitement utilisé était idem à celui de l'étape 1. Après avoir séparé notre ensemble d'entrainement en ensembles d'entrainement et de validation, nous avons ensuite eu recours à 3 techniques différentes afin de générer des traits caractéristiques : les méthodes ''sacs de mots'', TF-IDF et TF-IDF avec des n-grams de 2 termes. Nous avons comparé les traits caractéristiques résultants des 3 méthodes sur chaque modèle considéré à l'aide de la validation croisée afin de déterminer la méthode la plus performante. TF-IDF a performé le mieux sur chaque modèle.

Le ''tuning'' des hyperparamètres des modèles considérés s'est effectué en même temps que les tests sur les méthodes de construction de traits caractéristiques par le biais de la validation croisée sur l'ensemble d'entrainement. Une fois les hyperparamètres et la méthode de construction de traits caractéristiques choisis, la performance prédictive de chaque modèle a été testée sur l'emble de validation. La mesure utilisée afin de comparer la performance des modèles et des méthodes de construction de traits caractéristiques est la macro-moyenne du score F1 afin de compenser pour une distribution légèrement non-uniforme des différentes classes (sujets), ainsi qu'une matrice de confusion afin de visualiser la performance pour chaque classe. Les 2 modèles les plus performants sont : Bayes naïf et le perceptron multi-couches à une couche cachée.

Finalement, une méthode d'ensemble avec ''soft voting'' a été utilisée sur 3 des modèles afin d'obtenir les meilleures prédictions possibles. Le score final obtenu sur l'ensemble de test avec la méthode d'ensemble est de 0.59100.

Construction de traits caractéristiques

Le prétraitement pour les deux phases de la compétition consistait de 4 étapes :

- Transformer toutes les lettres en minuscules afin que la casse des lettres n'influence pas les résultats.
- Éliminer les termes non-alphabétiques (tels que la ponctuation, les chiffres et les autres types de signes) afin de préserver seulement les lettres, étant les éléments les plus utiles dans notre cas.
- Éliminer les ''stop words'' en anglais afin de se débarrasser des termes sans signification. De plus, lors de la transformation des textes en traits caractéristiques, les termes qui apparaissent dans plus de 90% des documents sont également éliminés pour les mêmes raisons.
- Lemmatizer tous les termes afin d'expliciter les redondances implicites dans le corpus. Les mots ''ate'' et ''eating'', par exemple, représentent la même idée et devraient donc être réduits à la même racine commune ''eat''. Le stemming des mots a également été utilisé afin de complémenter la lemmatization.

Nous avons exploré 3 méthodes différentes de construction de traits caractéristiques à partir des documents texte prétraités:

- Sac de mots : il s'agit d'une méthode rudimentaire qui compte la fréquence des différents termes dans les documents et le corpus. L'utilisation de cette méthode était obligatoire pour la phase 1 de la compétition et nous l'avons utilisée comme point de référence pour la deuxième phase.
- TF-IDF : cette méthode bâtit sur la méthode ''sac de mots'' en divisant la fréquence de chaque mot par le nombre de documents dans lesquelles ce mot est présent. Ceci est fait dans l'objectif de réduire l'influence des mots moins significatifs

qui demeurent dans le corpus même après l'élimination des ''stop words'', soit les mots qui reviennent fréquemment dans plusieurs documents.

- TF-IDF avec n-grams de 2 mots : Dans l'intention d'enrichir la méthode TF-IDF avec une meilleur compréhension du contexte de chaque mot, nous prenons en compte toutes les séquences différentes de 2 mots dans notre corpus afin de bâtir un vocabulaire dans lequel chaque séquence est considérée et traitée comme un terme unique. Afin de limiter le nombre de traits caractéristiques construits, nous avons seulement considéré les 50 000 n-grams avec les meilleurs scores TF-IDF.

Ces 3 méthodes ont été comparées entre elles lors du ''tuning'' de nos différents modèles afin de trouver la plus performante. Les résultats de ces tests sont présentés à la section Résultats. La méthode la plus performante pour tous les modèles testés est TF-IDF.

Algorithmes

Bayes naïf avec lissage

Le modèle de Bayes naïf construit pour la phase 1 de la compétition, ainsi que celui provenant de la librairie sklearn utilisé pour la phase 2 implémentent tous deux une stratégie de lissage.

Régression logistique

Le modèle de régression logistique provenant de la librairie sklearn avec un ''tuning'' sur les hyperparamètres C (terme de régularisation inverse) et solver (méthode d'optimisation des paramètres).

KNN

Le modèle de classifieur plus proches voisins de la librairie sklearn avec une distance euclidienne et un ''tuning'' sur les hyperparamètres k (nombre de voisins) et weights (type de vote alloué à chaque voisin : uniforme vs pondéré selon la distance).

Forêt aléatoire

Le modèle de classifieur forêt aléatoire de la librairie sklearn avec l'utilisation d'échantillons ''out-of-bag'' afin d'estimer l'erreur de généralisation (oob_score = True) et un ''tuning'' effectué sur les hyperparamètres n_estimators (le nombres de modèles à faible capacité utilisés, soit des arbres de décision) et max_depth (la profondeur maximale de chaque arbre).

Perceptron multi-couches

Le modèle MLP de la librairie sklearn avec l'utilisation d'une seule couche cachée. Après des tests préliminaires, un ''learning rate'' de 0.001 a été sélectionné pour la phase de ''tuning' et 0.0001 pour l'entrainement, ainsi que l'utilisation de fonctions d'activation sigmoïde dans la couche cachée et le solver adam (méthode d'optimisation des paramètres). Un ''tuning'' a été effectué sur les hyperparamètres hidden_layer_sizes (le nombre de neurones dans la couche cachée) et alpha (le terme de régularisation avec une pénalité L2).

Méthode d'ensemble avec 3 modèles

Le modèle VotingClassifier de la librairie sklearn employé sur les 2 modèles précédents les plus performants et un troisième démontrant des prédictions complémentaires avec une technique de ''soft voting''. Ce modèle ne requiert pas de ''tuning'' étant donné que les modèles fournis en entrée sont déjà ''tunés'' avec les hyperparamètres optimaux.

Méthodologie

Division entrainement/validation

Nous avons sélectionné 20% de l'ensemble d'entrainement initial, soit 14 000 points de données, de manière aléatoire afin de former l'ensemble de validation. Après coup, l'ensemble d'entrainement comptait 56 000 points de données. Étant donné que la taille de notre ensemble d'entrainement initial n'était pas limitée, nous avons choisi un ratio standard de 20/80. Les deux ensembles résultants sont de taille assez grande afin de préserver une distribution des classes similaire à celle de l'ensemble d'entrainement initial.

Distribution pour Bayes naïf

L'ensemble d'entrainement est divisé par classe et l'algorithme est entrainé sur chacune des classes individuellement, soit 20 fois au total. Pour chaque classe c, la vraisemblance du i^{e} terme w_i, prend la forme du lissage de Laplace,

soit $\hat{p}_c(w_i|c) = \frac{count(w_i,c)+a}{[\sum_{w\epsilon V} count(w,c)]+a|V|}$

où a est la constante de lissage (le seul hyperparamètre), $[\sum_{w\epsilon V} count(w,c)]$ est le compte total de tous les mots appartenant au vocabulaire V dans c, et $|V|$ est le nombre de mots appartenant au vocabulaire V dans c.
Le prior pour chaque classe c est $\hat{P}_c = \frac{n_c}{\mathrm{n}}$ où n_c est le nombre d'exemples de l'ensemble d'entrainement appartenant à la classe c, et n est le nombre d'exemples total dans l'ensemble d'entrainement.

Choix d'hyperparamètres

Bayes naïf (étape 1) : Pour l'implémentation de Bayes naïf avec lissage à l'étape 1 de la compétition, l'unique hyperparamètre à optimiser était la valeur du terme de lissage α. Le ''tuning'' a été effectué en entrainant et testant le modèle sur les ensembles complets d'entrainement et de validation de manière séquentielle avec plusieurs valeurs différentes de α.

Pour l'étape 2 de la compétition, le ''tuning'' de chaque modèle considéré a été fait à l'aide de la validation croisée sur une partie ou l'entièreté de l'ensemble d'entrainement (selon le temps d'exécution requis). Le nombre de ''folds'' a également été ajusté selon le temps d'exécution de chaque modèle. La performance des modèles a été mesurée à l'aide de la macro-moyenne du score F1 afin de compenser pour une distribution légèrement non-uniforme des différentes classes (sujets) et de viser une performance de classification la plus similaire possible entre les classes.

Bayes naïf : Idem à l'étape 1, l'unique hyperparamètre que nous avons jugé important d'optimiser est la valeur du terme de lissage α.

Régression logistique : Les 2 hyperparamètres que nous avons jugés importants d'optimiser sont le terme de régularisation inverse C et le ''solver'' utilisé pour l'optimisation des paramètres

KNN : Les 2 hyperparamètres que nous avons jugés importants d'optimiser sont le nombre de voisins considérés k et le type de vote alloué à chaque voisin (vote uniforme vs vote pondéré selon la distance).

Forêt aléatoire : Les 3 hyperparamètres que nous avons jugés important d'optimiser sont l'activation du oob_score (l'utilisation d'échantillons ''out-of-bag'' afin d'estimer l'erreur de généralisation), le nombre d'arbres de décision (modèles à faible capacité) considérés et la profondeur maximale des arbres.
Perceptron multi-couches : Les 4 hyperparamètres que nous avons jugés important d'optimiser sont le ''learning rate'', la fonction d'activation utilisée pour la couche de neurones cachée, le terme de régularisation L2 α et le nombre de neurones dans la couche cachée

Stratégies de régularisation

Régression logistique : nous nous sommes limités à la régularisation L2 étant donné que la majorité des ''solvers'' considérés pour ce modèle ne supportent que ce type de régularisation.

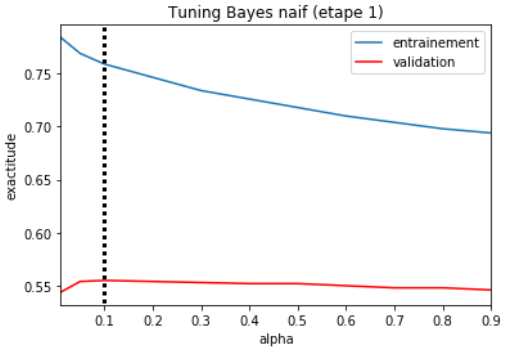
Perceptron multi-couches : nous avons utilisé la régularisation L2 étant donné qu'il s'agit du seul type de pénalité disponible pour l'implémentation sklearn de ce modèle.

Astuces d'optimisation

Régression logistique : différentes méthodes d'optimisation des paramètres (solvers) ont été testées lors du tuning des paramètres à l'aide de la validation croisée. La méthode qui a engendré les meilleurs résultats, soit ''lbfgs'' a été utilisée lors de l'entrainement du modèle sur l'ensemble d'entrainement complet.

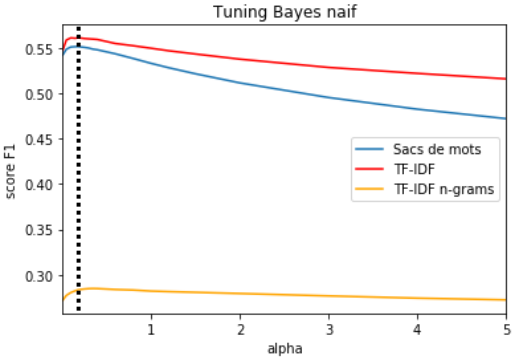
Perceptron multi-couches : différentes méthodes d'optimisation des paramètres (solvers) ont été testées lors de tests préliminaires et la méthode ''adam'' basée sur la méthode de gradient stochastique a engendré les meilleures performances. De plus, la technique de ''early stopping'' a été employée avec une tolérance de 0.0001 sur 10 époques afin d'arrêter l'entrainement au moment opportun pour minimiser l'erreur de validation.

Résultats



Bayes naïf

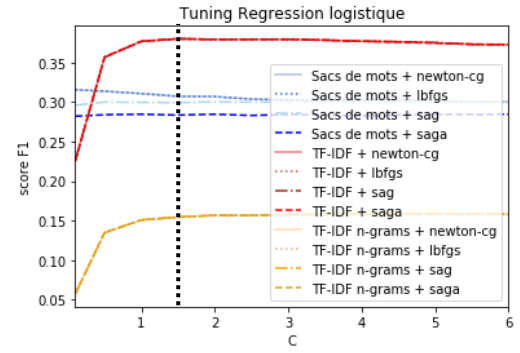
Pour l'étape 1 de la compétition, la valeur du terme de lissage α maximisant la performance du modèle a été trouvée en entrainant et testant le modèle avec les ensembles complets d'entrainement et de validation de manière séquentielle avec plusieurs valeurs différentes de α. La valeur optimale trouvée qui maximise l'exactitude des prédictions sur l'ensemble de validation est α = 0.1, avec une exactitude de 0.759 sur l'ensemble d'entrainement, 0.555 sur l'ensemble de validation et 0.55977 sur l'ensemble de test.



Pour l'étape 2, étant donné le temps d'exécution rapide de l'algorithme, nous avons effectué le ''tuning'' avec k = 10 et l'entièreté de l'ensemble d'entrainement. L'unique hyperparamètre qui nous avons jugé important d'optimiser est le terme de lissage α, pour une valeur optimale de α = 0.18 avec la méthode de construction de traits caractéristiques TF-IDF.

Les résultats obtenus sur l'ensemble de validation sont :

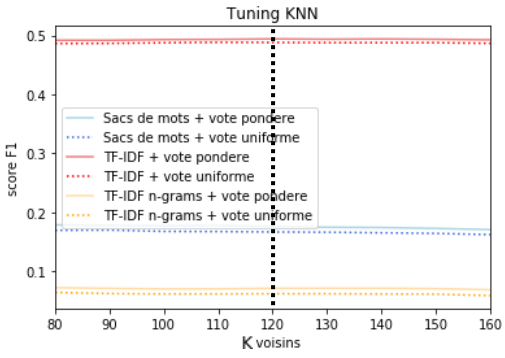
```
Precision (macro-avg):    0.5657026073639786
Recall (macro-avg):       0.5591891346060481
F1 Score (macro-avg):     0.5598018902262822
Accuracy:                 0.5578571428571428
```



Régression logistique : étant donné le temps d'exécution moyen de cet algorithme, nous avons effectué le ''tuning'' avec k = 5 et 10% des exemples de l'ensemble d'entrainement choisis de manière aléatoire sans remplacement. Les 2 hyperparamètres que nous avons jugés importants d'optimiser sont le terme de régularisation inverse C et le ''solver'' utilisé pour l'optimisation des paramètres. Les valeurs optimales trouvées sont C = 1.5 et solver = lbfgs, avec la méthode de construction de traits caractéristiques TF-IDF.

Les résultats obtenus sur l'ensemble de validation sont :

```
Precision (macro-avg):    0.5545710221694808
Recall (macro-avg):       0.5373103787491856
F1 Score (macro-avg):     0.5444044772198715
Accuracy:                 0.5363571428571429
```

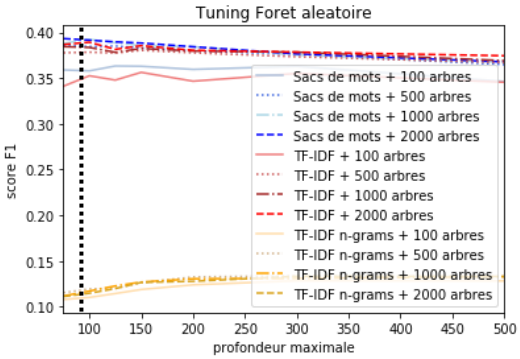


KNN : étant donné le temps d 'exécution rapide de cet algorithme, nous avons effectué le ''tuning'' avec k = 10 et l'entièreté de l'ensemble d'entrainement. Les 2 hyperparamètres que nous avons jugés importants d'optimiser sont le nombre de voisins considérés k et le type de vote alloué à chaque voisin (vote uniforme vs vote pondéré selon la distance). Les valeurs optimales trouvées sont k = 120 avec un vote pondéré selon la distance et la méthode de construction de traits caractéristiques TF-IDF.

Les résultats obtenus sur l'ensemble de validation sont :

```
Precision (macro-avg):     0.49057142062553416
Recall (macro-avg):        0.49100823044032904
F1 Score (macro-avg):      0.487126442594466
Accuracy:                  0.48957142857142855
```

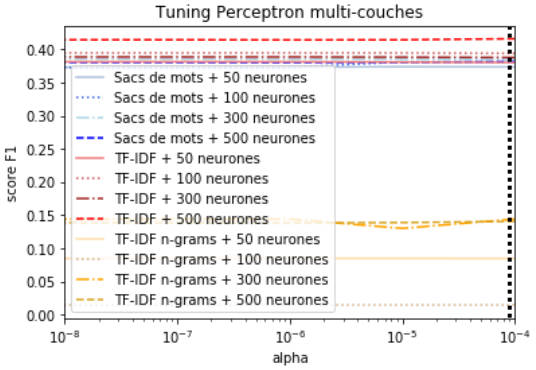
Forêt aléatoire : étant donné le temps d'exécution moyen de cet algorithme, nous avons effectué le ''tuning'' avec k = 3 et 10% des exemples de l'ensemble d'entrainement choisis de manière aléatoire sans remplacement. La validation croisée est moins importante pour ce modèle étant donné qu'il utilise déjà la méthode de ''bagging'' avec des arbres de décision afin de réduire la variance. Après quelques tests préliminaires sur les variations d'échelle du nombre d'arbres de décision et, sachant qu'en général pour ce modèle l'augmentation du nombre d'arbres est rattachée à une décroissance exponentielle de l'erreur d'entrainement et de validation, nous incluons seulement 4 valeurs différentes du nombre d'arbres. Ceci est dans le but de limiter le temps d'exécution du ''tuning''. Nous pouvons voir sur le graphique que les performances avec 1000 et 2000 arbres sont similaires. Le deuxième hyperparamètre considéré est la profondeur des arbres de décision. Les valeurs optimales trouvées sont 2 000 arbres et une profondeur maximale de 75 avec la méthode de construction de traits caractéristiques ''sacs de mots''.



Les résultats obtenus sur l'ensemble de validation sont :

```
Precision (macro-avg):      0.6104066619264767
Recall (macro-avg):         0.09153732210077034
F1 Score (macro-avg):       0.0843027013830697
Accuracy:                   0.093
```

Perceptron multi-couches : étant donné le temps d'exécution lent de cet algorithme, nous avons effectué le ''tuning'' avec k = 2 et 10% des exemples de l'ensemble d'entrainement choisis de manière aléatoire sans remplacement. Après l'exécution de tests préliminaires avec validation croisée, nous avons sélectionné un ''learning rate'' de 0.001 pour le ''tuning'' et de 0.0001 pour l'entrainement du modèle, une fonction d'activation sigmoïde pour la couche cachée et une méthode d'optimisation des paramètres ''adam''. Les 2 hyperparamètres que nous avons jugé important d'optimiser avec un ''tuning'' plus formel sont le terme de régularisation L2 α et le nombre de neurones dans la couche cachée. Les valeurs optimales trouvées sont α=0.0001 et une couche cachée contenant 500 neurones.



Les résultats obtenus sur l'ensemble de validation sont :

```
Precision (macro-avg):      0.588362507258043
Recall (macro-avg):         0.5710195178223443
F1 Score (macro-avg):       0.5775178605077267
Accuracy:                   0.5700714285714286
```

Méthode d'ensemble avec 3 modèles: une méthode d'ensemble avec ''soft voting'' a été implémentée, prenant en entrée les 2 modèles les plus performants et un troisième démontrant des résultats complémentaires, soit Bayes naïf, le perceptron multi-couches et la forêt aléatoire. Ces modèles sont jugés complémentaires, car les prédictions faites par la forêt aléatoire sont meilleures pour certaines classes, tandis que celles faites par Bayes naïf et le perceptron sont meilleures pour les autres[1]. Étant donné que les hyperparamètres des modèles pris en entrée étaient déjà choisis, aucun ''tuning'' n'a été requis pour ce modèle d'ensemble.

Les résultats obtenus sur l'ensemble de validation sont :

```
Precision (macro-avg):      0.584038472986594
Recall (macro-avg):         0.5755939466628395
F1 Score (macro-avg):       0.5789434711782967
Accuracy:                   0.5745
```

L'exactitude des prédictions su l'ensemble de test = 0.59100

[1]*Voir les matrices de confusion pour Bayes naïf, forêt aléatoire et le perceptron multi-couches en appendice.*

Discussion

Les méthodes de prétraitement de texte utilisées ont contribué substantiellement à l'amélioration de la performance de tous les modèles utilisés, en grande partie en explicitant les redondances implicites présentent dans les documents textes initiaux (p.ex. en réduisant les mots ''manger'', ''mangé'' et ''mangées'' à leur racine commune ''mange''). L'élimination des ''stop words'' à également allégé le vocabulaire considéré et nous a débarrassé de termes superflus. Des techniques additionnelles de correction du vocabulaire auraient pu être exploitées, mais celles-ci requièrent un temps d'exécution trop long avec l'équipement à notre disposition.

Afin d'assuré une sélection des hyperparamètres appropriés, la validation croisée a été utilisée sur chaque modèle au cours de la deuxième partie de la compétition. Dans le cas des modèles les plus longs à optimiser, soit la forêt aléatoire et le perceptron multi-couches, des valeurs de k (le nombre de ''folds'') assez petites ont été choisies. Refaire l'étape de ''tuning'' de ces modèles avec des valeurs de k plus grandes pourrait engendrer une meilleure sélection d'hyperparamètres.

L'utilisation d'une méthode d'ensemble avec ''soft voting'' intégrant les 2 modèles les plus performants et un troisième modèle avec des résultats complémentaires, mais de loin la pire performance, a engendré des prédictions avec une exactitude supérieure à la méthode d'ensemble appliquée seulement aux 2 modèles les plus performants. Bien que le modèle de forêt aléatoire ait obtenu un score d'exactitude bien plus bas que les autres modèles, elle a obtenu le meilleur score ''recall'' spécifiquement pour la classe ''funny'' (voir matrices de confusion en appendice). Donc, malgré la mauvaise performance de la forêt aléatoire, ce modèle était surprenamment une composante bénéfique à prendre en compte dans la méthode d'ensemble.

3 méthodes différentes de construction de traits caractéristiques ont été utilisées pour cette compétition. Une quatrième méthode, soit l'utilisation de ''word embeddings'' afin de créer des vecteurs de mots ou des vecteurs de documents, pourrait être explorée afin de voir si elle peut contribuer à des meilleures performances.

Dans le cadre de l'utilisation des modèles de Bayes naïf avec lissage et de régression logistique, les hyperparamètres les plus significatifs ont été considérés lors de la phase de ''tuning''. Il est donc improbable que ces modèles puissent livrer une meilleure performance. Par contre, les hyperparamètres des 3 autres modèles n'ont pas été complétement explorés, laissant la possibilité de pouvoir pousser ces modèles vers une meilleure performance. Dans le cas de KNN, l'unique mesure utilisée afin de calculer la distance entre les points est la distance euclidienne. Des tests additionnels avec différentes mesures de distance pourraient être effectués. Dans le cas de la forêt aléatoire, nous nous sommes limités à 2 000 arbres lors du ''tunning'' et 3 000 lors de l'entrainement sur l'ensemble d'entrainement complet afin de réduire le temps d'exécution. Il est possible de des valeurs plus grandes engendrent de meilleurs résultats. Pour des raisons similaires, nous nous sommes limités à l'implémentation d'une seule couche cachée dans le cas du perceptron multi-couches. Il est probable que l'utilisation de plusieurs couches cachées ou bien simplement une seule couche cachée comprenant davantage de neurones puissent engendrer de meilleurs résultats.

Liste des contributions

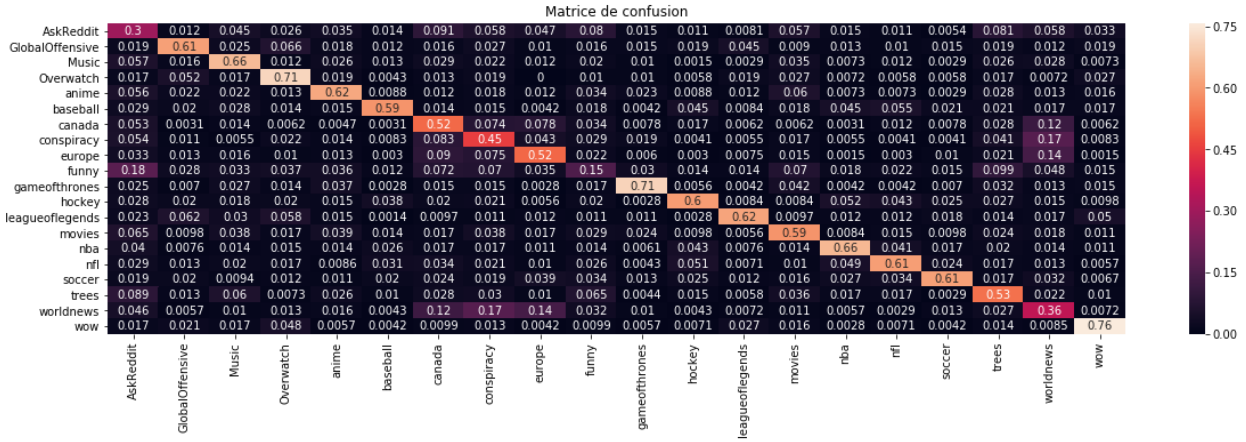
David Raby-Pepin : entièreté du projet
Yifu Zhou : abandon du cours
Nous certifions que nous sommes les auteurs des travaux présentés dans ce rapport.

Références

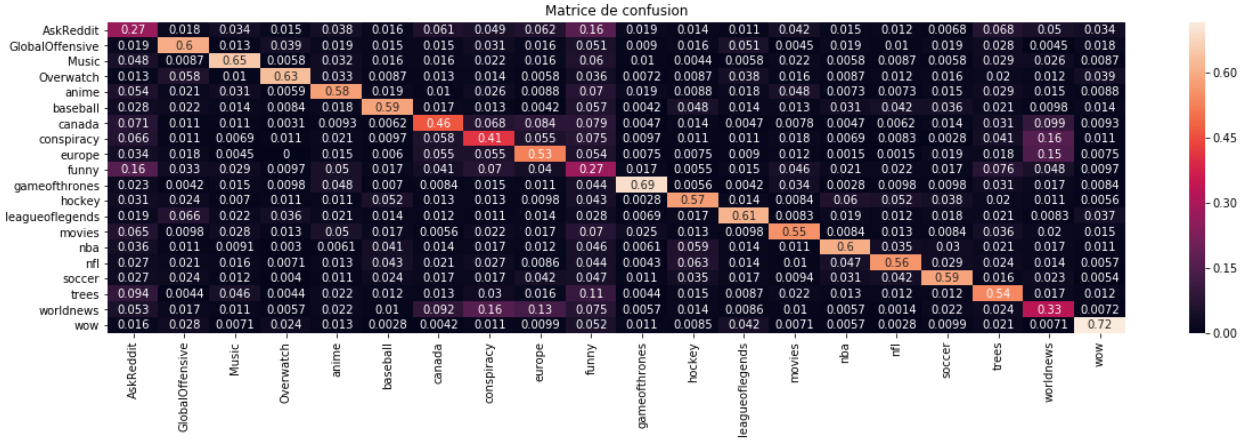
https://towardsdatascience.com/text-classification-in-python-dd95d264c802
https://medium.com/machine-learning-intuition/document-classification-part-2-text-processing-eaa26d16c719
https://towardsdatascience.com/multi-class-text-classification-model-comparison-and-selection-5eb066197568
https://towardsdatascience.com/multi-class-text-classification-with-scikit-learn-12f1e60e0a9f
https://towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a

Appendice

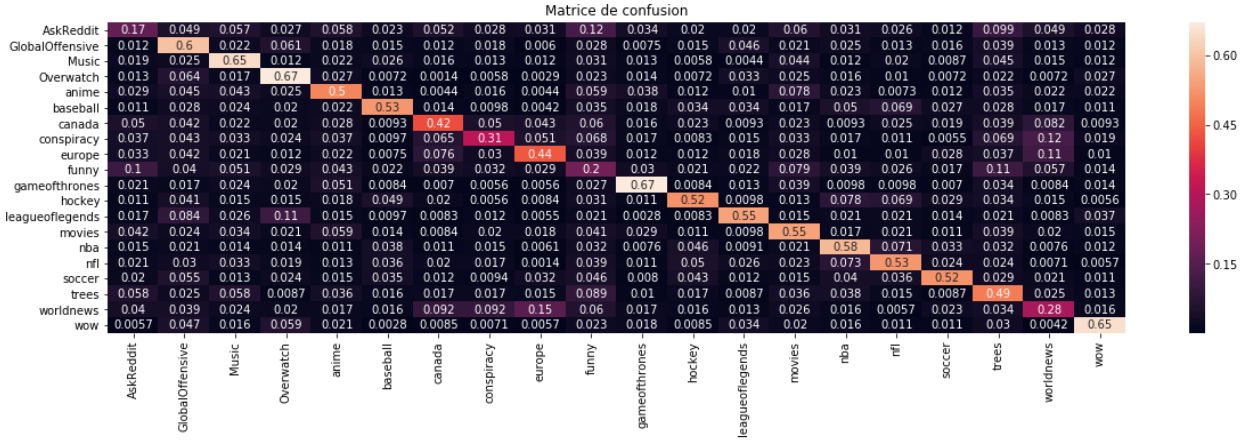
Bayes naïf : matrice de confusion sur l'ensemble de validation



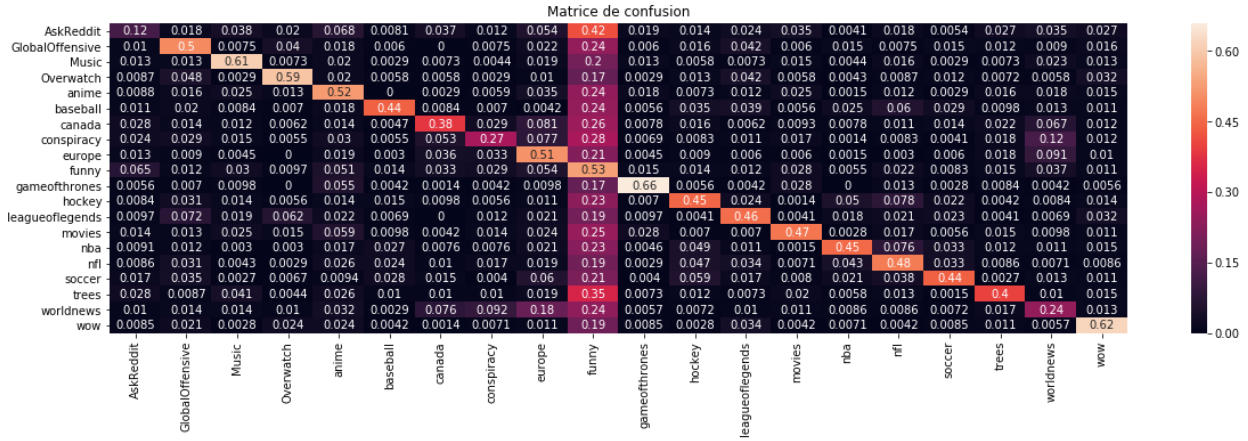
Régression logistique : matrice de confusion sur l'ensemble de validation



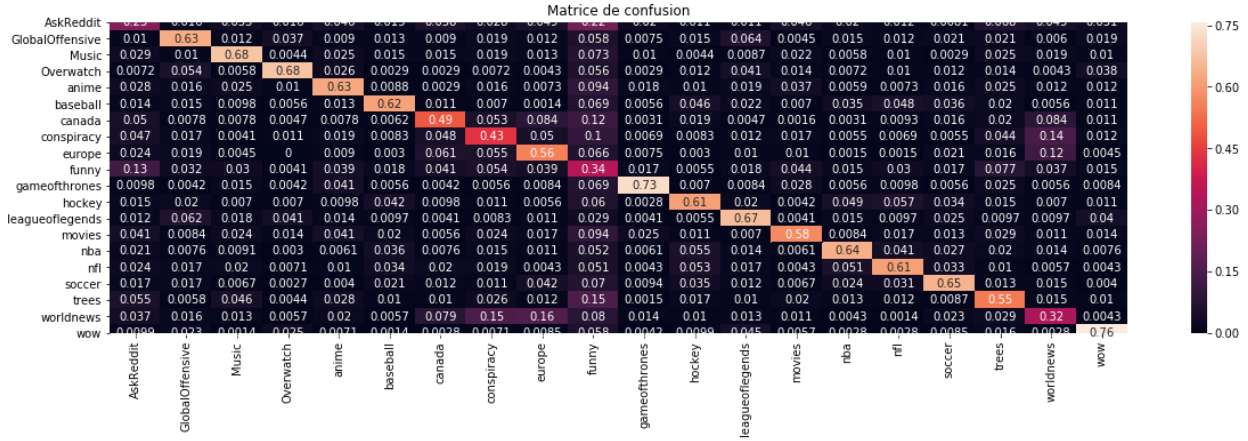
KNN : matrice de confusion sur l'ensemble de validation



Forêt aléatoire : matrice de confusion sur l'ensemble de validation



Perceptron multi-couches : matrice de confusion sur l'ensemble de validation



Méthode d'ensemble avec 3 modèles : matrice de confusion sur l'ensemble de validation