Rapport : Projet TAL – RI
NADLER Bastian

I) Code python :

1) Introduction

Il y a 6 fichiers python différent dans lesquels j'ai appliqué les différentes méthodes présentées par les TP. Ainsi dans le code présenté, il y a:

- projet_tdidf.ipynb correspond aux méthodes utilisées dans le 1er Tp
- projet_CNN_FP.ipynb et projet_CNN_FM.ipynb correspondent aux méthodes utilisées dans le 2er Tp
- projet_BILSTM_FM.ipynb et projet_BILSTM_FP.ipynb correspondent aux méthodes utilisées dans le 3er Tp (Les fichiers FP et FM se distinguent uniquement par le modèle utilisé, FM utilise [1] et FP [2])
- projet_tranfomer.ipynb correspond aux méthodes utilisées dans le 4er Tp

Un large éventail d'algorithme a été utilisé pour classifier automatiquement les films en fonction de leur genre avec un apprentissage sur leur synopsis et leur titre. Ces algorithmes de classification seront présentés par ordre croissant de précision sur la prédiction du genre. A noter que pour la plupart des apprentissages, différentes approches ont été envisagé et la validation croisée est toujours mise en place quand cela été envisageable *i.e.* quand cela était faisable en un temps raisonnable. Ainsi la validation croisée est notamment faite dans les 3 premiers codes ci-dessus. Dans projet_transformer elle n'est pas réalisée car elle prend beaucoup trop de temps et de puissance de calcul. De plus, je me suis concentré sur l'utilisation de tous les algorithmes présents dans les tps afin de comparer leur efficacité, en testant peu d'architectures différentes.

2) CNN

Cet algorithme est celui ayant eut les moins bon résultats en terme de précision. Pour les modèles de plongements de mots pré-entrainé, 2 modèles différents ont été utilisés trouvables sur les sites présents en bibliographie [1] et [2]. Le modèle de [1] donne des résultats moins bons que ceux de [2]. Notamment le modèle [2] donne des résultats similaires à un code n'utilisant pas de modèle (à 0.02 près).Nous obtenons les résultats suivant :

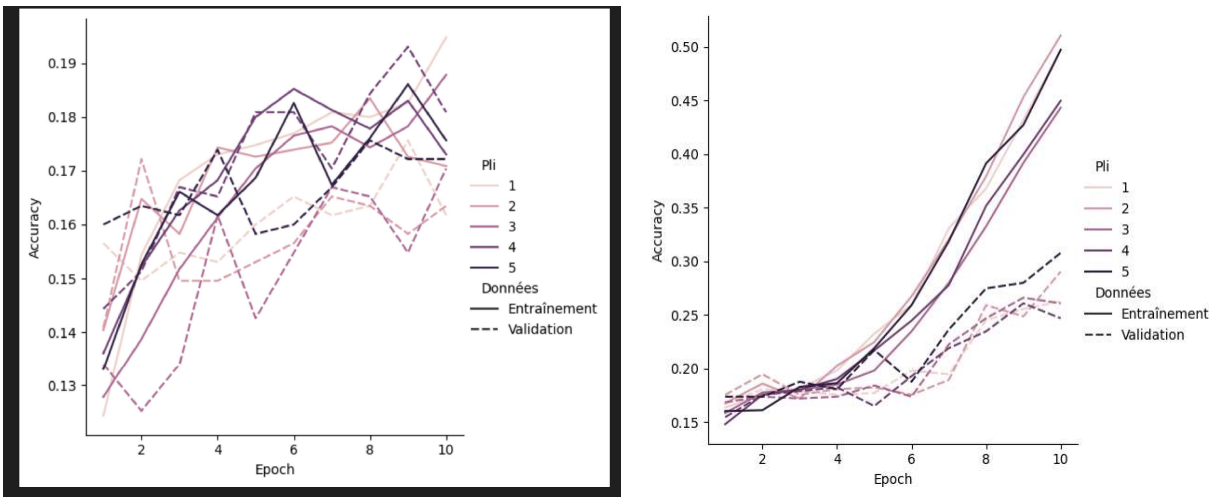


Figure 1 et 2 : Accuracy de l'agorithme CNN. A gauche avec l'utilisation du modèle [1] de plongement de mots pré-entrainé, et à droite sans.

Nous avons en moyenne 0,16 de précision avec l'utilisation du modèle [1] et 0,27 sans (ou en utilisant le modèle de [2]). Sachant que les 2 modèles de plongements de mots pré-entrainés fonctionnent pour l'algorithme BILSTM, il est probable que l'algorithme CNN soit mauvais pour résoudre notre problème. Pour améliorer les résultats, il faut très probablement un plus grand nombre de données. De plus, il faut probablement modifier son architecture *i.e.* changé le nombre de couches, la taille des filtres, les fonctions d'activation et les techniques de régularisation en fonction. A noté que j'ai tout de même essayé de changer l'architecture, mais cela n'a pas donné de bien meilleurs résultats.

3) BILSTM

De même que précédemment, les tests ont été effectués avec les 2 modèles de plongements de mots pré-entrainés présent en bibliographie. Néanmoins, les 2 modèles donnent sensiblement les mêmes résultats ici (à +-0.02 de précision). Ainsi en utilisant LSTM et GRU, on trouve comme résultats :

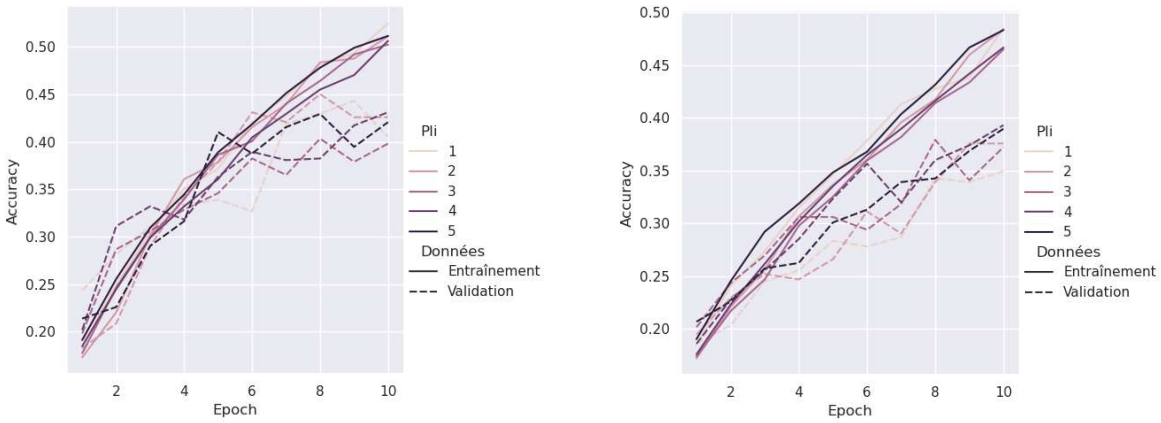


Figure 3 et 4 : Accuracy de l'algorithme BILSTM. A gauche avec LSTM et à droite avec GRU

En moyenne, on a une précision de 0,37 pour GRU et 0,42 pour BILSTM. On voit qu'avec cet algorithme on obtient une bien meilleure précision qu'avec l'algorithme CNN. Il y a cependant plusieurs pistes pour améliorer ce résultat. Tout d'abord, comme pour CNN, un plus grand nombre de données permettrait une plus grande précision. Un prétraitement des données (tels que la désuffixation) pourrait aussi améliorer les résultats .Il faudrait aussi chercher à modifier l'architecture du réseau afin d'en trouver une plus optimale. Je n'ai pas réalisé de comparaison avec d'autre architecture car je n'ai pas réussi à construire d'architecture plus optimale.

4) tdidf

Dans cette partie, plusieurs algorithmes différents ont été comparés, avec à chaque fois des paramètres différents. Comme algorithmes utilisé, nous avons : Random forest, KNN, LR, CART, Multinomial NB, Baseline. Des tests ont été réalisés avec une utilisation de la désuffixation, l'application de différents seuils de fréquence pour les tokens à conserver dans le vocabulaire, utilisation de différentes méthodes de pondération. Néanmoins Les pondération (sublinear_tf, use_idf, smooth_idf dans TfidfVectorizer) ne change pas sensiblement les résultats (l'accuracy est modifié de l'ordre de 0.01). Ainsi nous avons les résultats suivant :

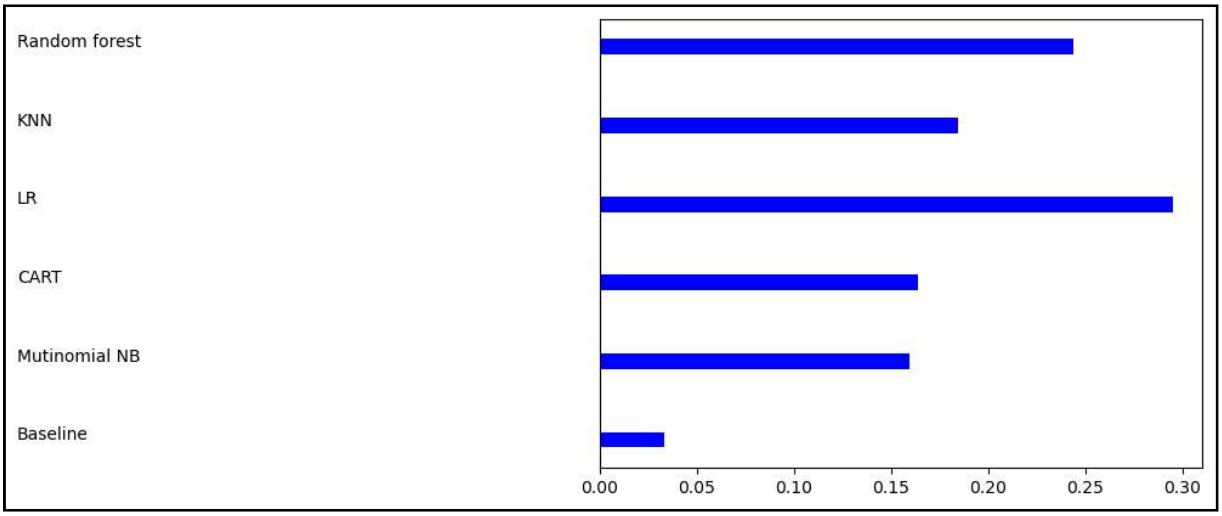


Figure 5 : Fréquence de 0,1 et sans désuffixation

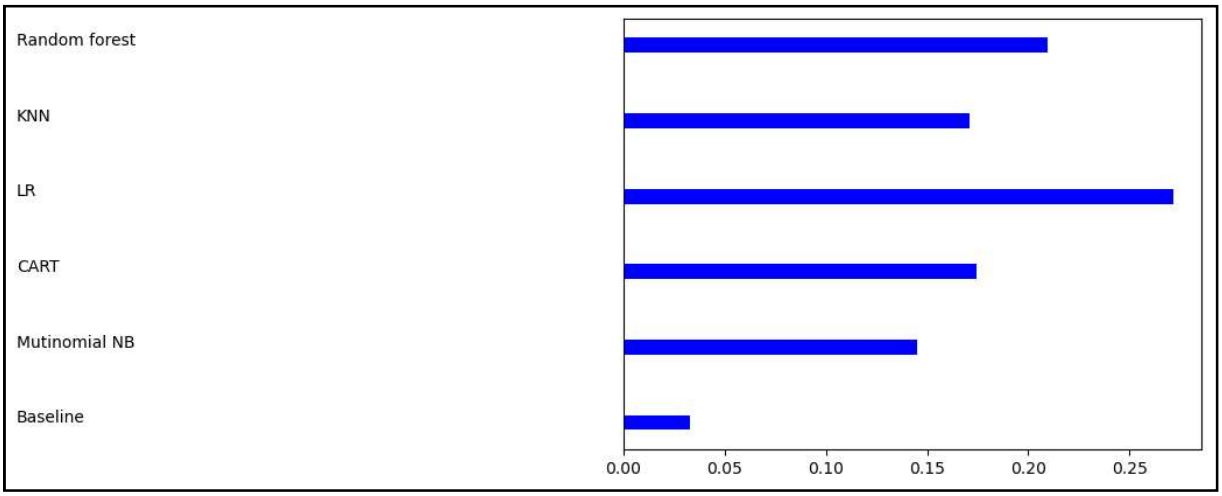


Figure 6 : Fréquence de 0,1 et avec désuffixation

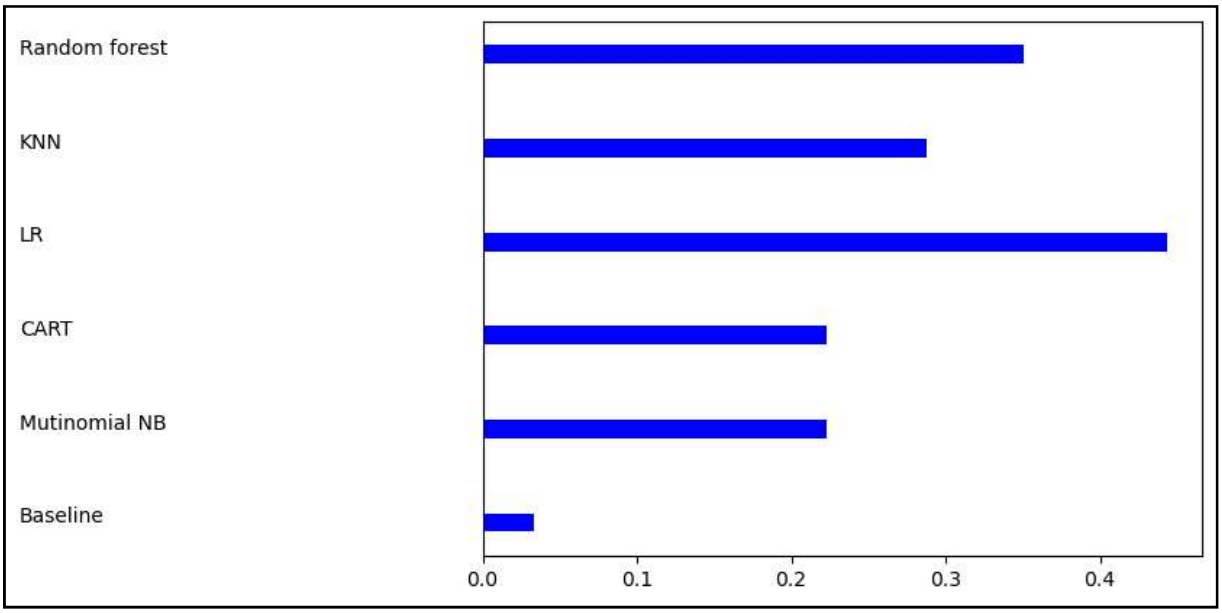


Figure 7 : Fréquence de 0,01 et sans désuffixation

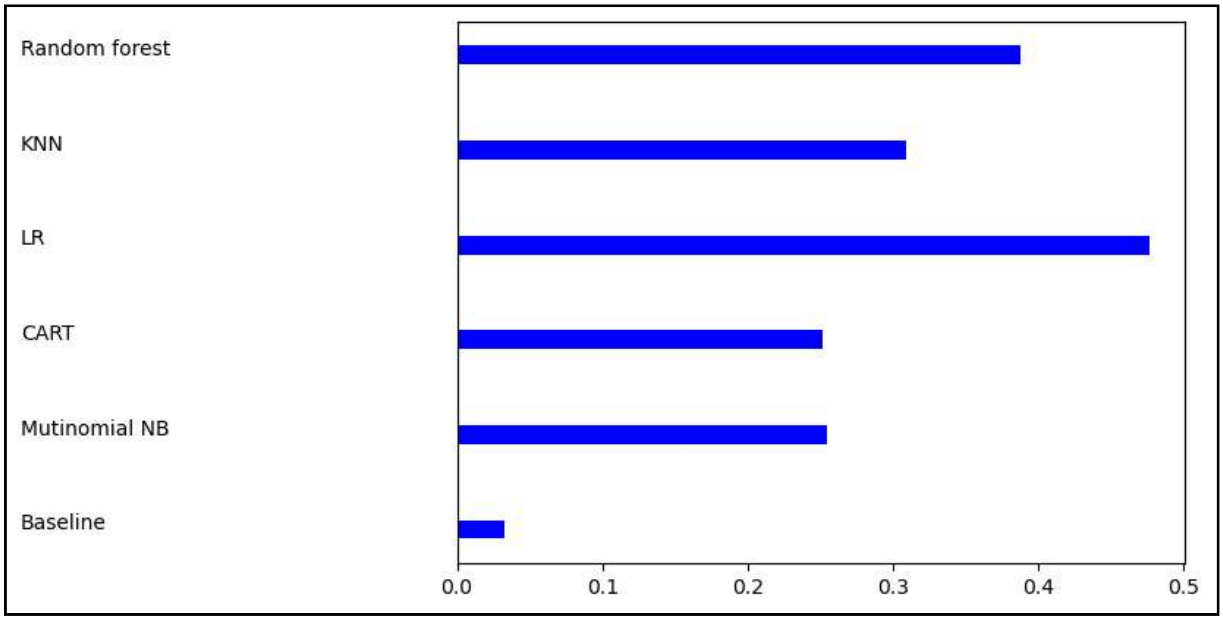


Figure 8 : Fréquence de 0,01 et avec désuffixation

Nous pouvons voir que l'algorithme le plus efficace est tout le temps LR. Il est le plus précis avec une fréquence de 0,01 et la désuffixation, atteignant une précision de 0,48 (0.43 sans désuffixation). Pour améliorer cette précision, il faudrait idéalement plus de données et veiller à un équilibrage des classes (ou gérer le déséquilibrage).

5) Transformer

La dernière méthode utilisée est le Transformer. C'est celle qui donne les meilleurs résultats avec une accuracy de 0,60. Pour celle-ci on utilise le modèle « xlm-roberta-base » pour la tokénisation des données, qui est le plus utilisé pour la langue française. On a ainsi les résultats suivant sur le fichier allocine_train.csv:

```
preds_output.metrics
```

```
{'test_loss': 1.1796503067016602,
 'test_accuracy': 0.6139130434782609,
 'test_runtime': 19.4273,
 'test_samples_per_second': 29.598,
 'test_steps_per_second': 3.706}
```

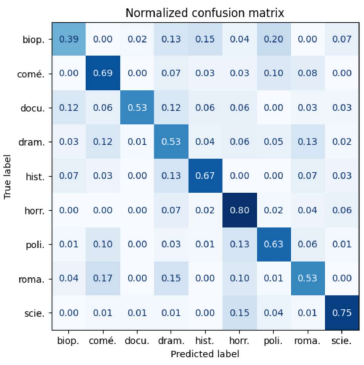


Figure 9 et 10 : Résultats du transformer lors du test et matrice de confusion

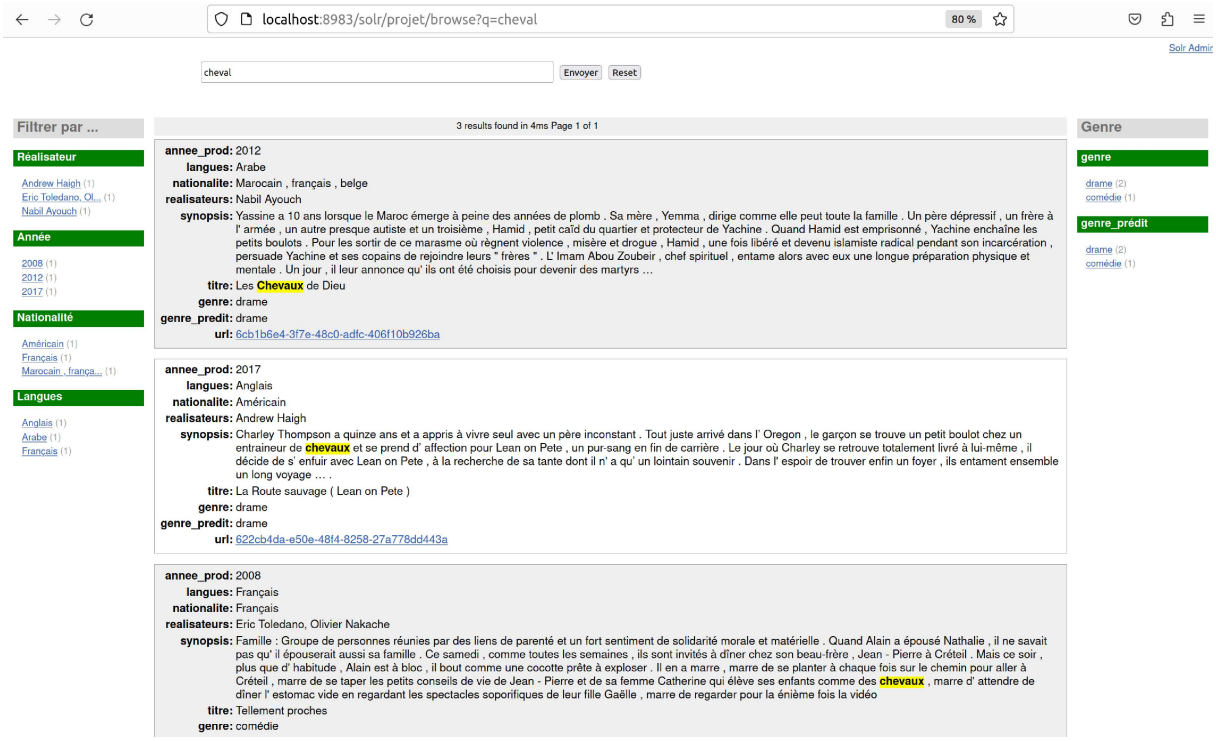
Ainsi comme c'est la méthode ayant les meilleurs résultats, le fichier allocine_test_enrichi.csv a été crée grâce à cette méthode. Pour améliorer la précision, il faudrait beaucoup plus de données, et modifier le learning rate, le nombre de batch, le nombre d'époque et le poids pourrait améliorer la précision. Néanmoins j'ai fait peu de test de modification de paramètre car les transformers prennent énormément de temps.

6) Conclusion :

De nombreuses méthodes ont été utilisées. La plus efficace utilise les transformer est atteint une précision de 0,60. En général, les pistes d'améliorations sont l'augmentation de données, la modification de la structure des réseaux et un prétraitement des données. Il faudrait aussi annuler l'impact négatif du déséquilibre des classes sur la précision des algorithmes. Pour les algorithmes utilisant des modèles provenant du net (xlm-roberta-base, les prolongements de mots pré-entrainés) on pourrait en essayer d'autre pour voir si cela peut améliorer les résultats. Néanmoins pour les prolongements de mots pré-entrainés, les 2 utilisés lors du projet sont assez fournis (entre 1.5Gb et 2.2Gb) et il y a peu de chance qu'un autre puisse améliorer drastiquement les résultats. Et aucun autre modèle que xlm-roberta-base n'a été testé pour les transformers car cela prend trop de temps et de puissance de calculs.

II) TAL

Le document TAL a été crée et ajusté pour correspondre à l'image donnée dans l'intitulé du projet. Le résultat final devrait être cela :



Par ailleurs, seulement les mots présent dans synopsis et le titre sont mit en surbrillance, (choix étant arbitraire). D'autres facettes que celles affichées sont calculées. Elles peuvent donc être affichées si on le souhaite en modifiant les facettes.

III) Bibliographie :

[1]http://vectors.nlpl.eu/repository/# : fichier ayant l'id 51 pour le modèle de plongement de mots en français.

[2]https://fasttext.cc/docs/en/crawl-vectors.html#models : pour le modèle de plongement de mots en français.