Projet TAL - RI

L'objectif du projet est de réaliser un système de recherche d’information dans une collection d’articles publiés sur Wikinews.

Partie 1 - Classification automatique

Prétraitement des données :

Dans cette partie, nous avons à disposition un fichier qui contient des articles publiés sur Wikinews étiquetés avec une catégorie. Nous pouvons observer 3 colonnes : `'title'`, `'text'`, `'categ'`. L’objectif est d’entraîner un outil de classification automatique des articles en fonction de leur catégorie. La classification doit se baser sur le texte et le titre des articles donc nous devons prédire la catégorie en fonction des deux éléments précédants. Nous allons donc travailler sur un Notebook Jupyter pour cette partie qui se trouvera en annexe du git.

Pour commencer, nous allons importer notre fichier et voir si il y a des modifications à faire. Comme indiquer dans l'énoncé, le text et le titre des articles ont déjà été tokénisés et tous les tokens sont séparés par une espace, ce qui nous simplifie grandement la tâche.

Nous allons effectuer une modification sur la colonne `'categ'` afin que son type ne soit pas un object mais une catégorie et donc de faire plus facillement de la classification. Je vous laisse regarder le Notebook afin de voir le code utilisé.

Après avoir effectué cela, nous allons faire une classification par catégorie où `'title'` et `'text'` sont les données sources utilisées pour l'apprentissage et `'categ'` est la donnée que l'on cherche à prédire. Nous allons découper automatiquement les données en jeu d'apprentissage et de test comme dans les TP's que nous avons vu en cours (données d'apprentissage = 80% et données d'entraînement = 20%).

Nous allons effectuer quelques modifications sur nos données d'entraînements afin que la classification soit meilleure.

La première modification se situe au niveau de la colonne `'title'` où nous allons donc définir une fonction de tokénisation qui découpe le titre en fonction de ce signe. Cette fonction de tokénisation sera ensuite utilisée par un objet de type CountVectorizer, afin de pouvoir passer à une représentation "sac de mots".

Notre seconde modification se situe au niveau de la colonne `'text'` où le texte sera découpé en mots à l'aide de la fonction split_into_tokens_nltk. Les mots seront également mis en minuscules et les mots vides seront supprimés, à l'aide des paramètres spécifiques à TfidfVectorizer. Je vous laisse regarder le Notebook pour voir le code.

Nous avons aussi ajouter une colonne afin d'avoir des données statistique sur la colonne `'text'` mais nous n'allons pas nous attarder dessus elle est juste là pour nous apporter des informations supplémentaires.

Il est maintenant possible de combiner toutes ces chaînes de pré-traitement afin d'obtenir une chaîne globale, qui produira l'union des traits générés indépendamment par chaque type de pré-traitement opéré sur les colonnes.

J'ai oublié de rajouter dans le Notebook le fait que j'ai aussi utiliser le stemming mais je n'obtiens pas vraiment de meilleure résultat cependant d'où le fait qu'il ne fasse pas partir de mon Notebook. J'ai utilisé les commandes suivantes :

```
from nltk.stem.snowball import SnowballStemmer
```

```
stemmer_france = SnowballStemmer("french")
```

Apprentissage :

Avec tout cela, on peut maintenant effectuer une chaîne complète et effectuer un apprentissage sur notre jeu de données d'entraînements. Pour notre premier apprentissage, nous allons utiliser l'algorithme de régression logistique qui est un des plus connus puis dans un second temps, nous allons évaluer notre modèle sur le jeu de test.

```
[25] y_pred = classifier_pipeline.predict(X_test)
     print("Classification report:\n\n{}".format(classification_report(y_test, y_pred)))

     Classification report:

                   precision    recall  f1-score   support

          economie       0.74      0.65      0.70       199
        enviro_cat       0.82      0.60      0.70        91
           justice       0.80      0.66      0.73       134
             polit       0.77      0.90      0.83       491
         sci_cult        0.75      0.67      0.71       128
           sports        0.99      0.99      0.99       957

          accuracy                           0.87      2000
         macro avg       0.81      0.75      0.77      2000
      weighted avg       0.87      0.87      0.87      2000
```

Nous allons à chaque fois regarder la colonne `'f1-score'` pour comparer nos résultat. Le f1-score permet de résumer les valeurs de la precision et du recall en une seule métrique. Nous pouvons voir que nous possédons une accuracy de 87% (87 des prédictions sont bonnes sur 100 [rapport]).

Nous avons réaliser une matrice de confusion afin de repérer facilement les classes pour lesquelles le modèle à le plus de difficultés (grandes valeurs qui ne se trouvent pas sur la diagonale) et avec quelle(s) autre(s) classe(s) ces classes sont le plus souvent confondues.

Nous allons essayer maintenant d'autres algorithme d'apprentissage afin de voir si on peut avoir un modèle plus performant.

Apprentissage par validation croisée :

Nous allons utilisé Kfold afin d'obtenir un découpage du jeu de données en plis consécutifs. La technique de la validation croisée à K plis (KFold) permet d'éviter de découper les données disponibles en jeu d'entraînement, de validation et de test : le jeu de validation n'est plus nécessaire.

Dans la validation croisée à k plis (k-fold cross-validation), les données d'entraînement sont découpées en parties (les "plis") :

- Un modèle est entraîné en utilisant k - 1 "plis" puis validé sur le "pli" restant.
- On fait ensuite la moyenne des mesures d'évaluation pour obtenir la performance finale du modèle et sélectionner les meilleurs paramètres.

Pour notre jeu d'apprentissage, nous allons utilisé 5 "plis" comme nous avons 80% de données d'apprentissage et 20% de données d'entraînement.

```
[ ] print(classification_report(y_train, y_kfold_pred))
```

	precision	recall	f1-score	support
economie	0.69	0.62	0.65	788
enviro_cat	0.77	0.57	0.65	378
justice	0.75	0.59	0.66	591
polit	0.74	0.89	0.81	1915
sci_cult	0.70	0.56	0.62	505
sports	0.99	0.98	0.99	3823
accuracy			0.85	8000
macro avg	0.77	0.71	0.73	8000
weighted avg	0.85	0.85	0.85	8000

On peut voir que l'accuracy du f1-score est de 85% donc un peu moins bonne que lors de notre régression logistique. Nous allons essayé maintenant avec une alternative de la méthode KFold, la méthode StratifiedKFold (prend en compte la répartition des classes, et le pourcentage d'exemples pour chaque classe est préservé dans chaque pli).

```
[33] print(classification_report(y_train, y_stratkfold_pred))
```

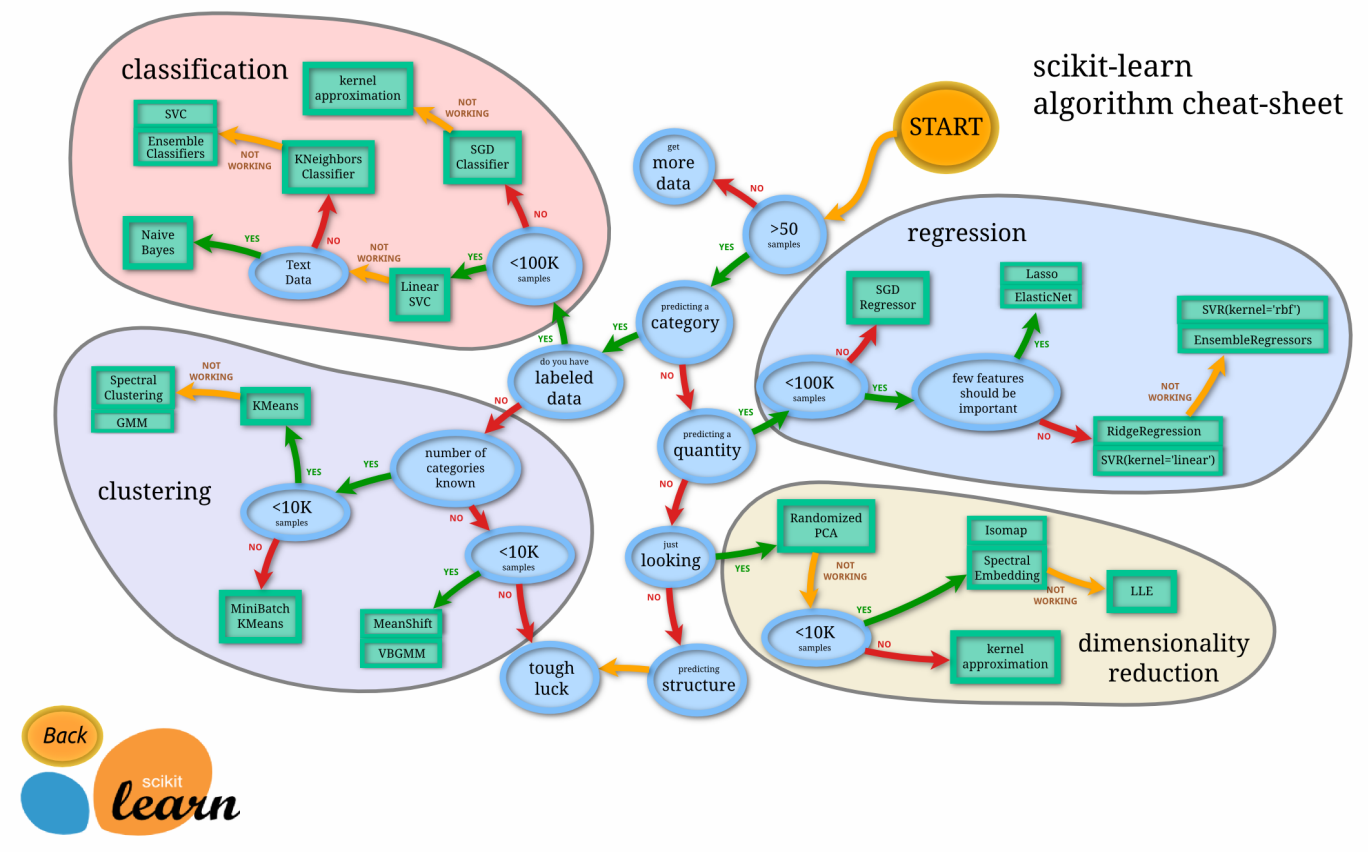
	precision	recall	f1-score	support
economie	0.68	0.63	0.65	788
enviro_cat	0.76	0.59	0.67	378
justice	0.75	0.60	0.66	591
polit	0.74	0.89	0.81	1915
sci_cult	0.74	0.59	0.65	505
sports	0.99	0.99	0.99	3823
accuracy			0.86	8000
macro avg	0.78	0.71	0.74	8000
weighted avg	0.86	0.86	0.85	8000

On peut voir que l'accuracy du f1-score est un peu meilleure que celui de l'algorithme KFold classique (85%) mais toujours moins bon que pour la régression logistique (87%).

En général, les modèles par validation croisée sont meilleurs mais vu que ici, on ne dispose pas de suffisament d'instances alors on obtient des moins bonnes valeurs. Sur un plus gros jeu de données, on obtient de bien meilleurs résultats avec la validation croisée.

Test d'évaluation avec différents modèles :

Dans la documentation scikit-learn, j'ai trouvé un chemin qui indique quel algorithme utilisé en fonction de notre jeu de données d'entrée.



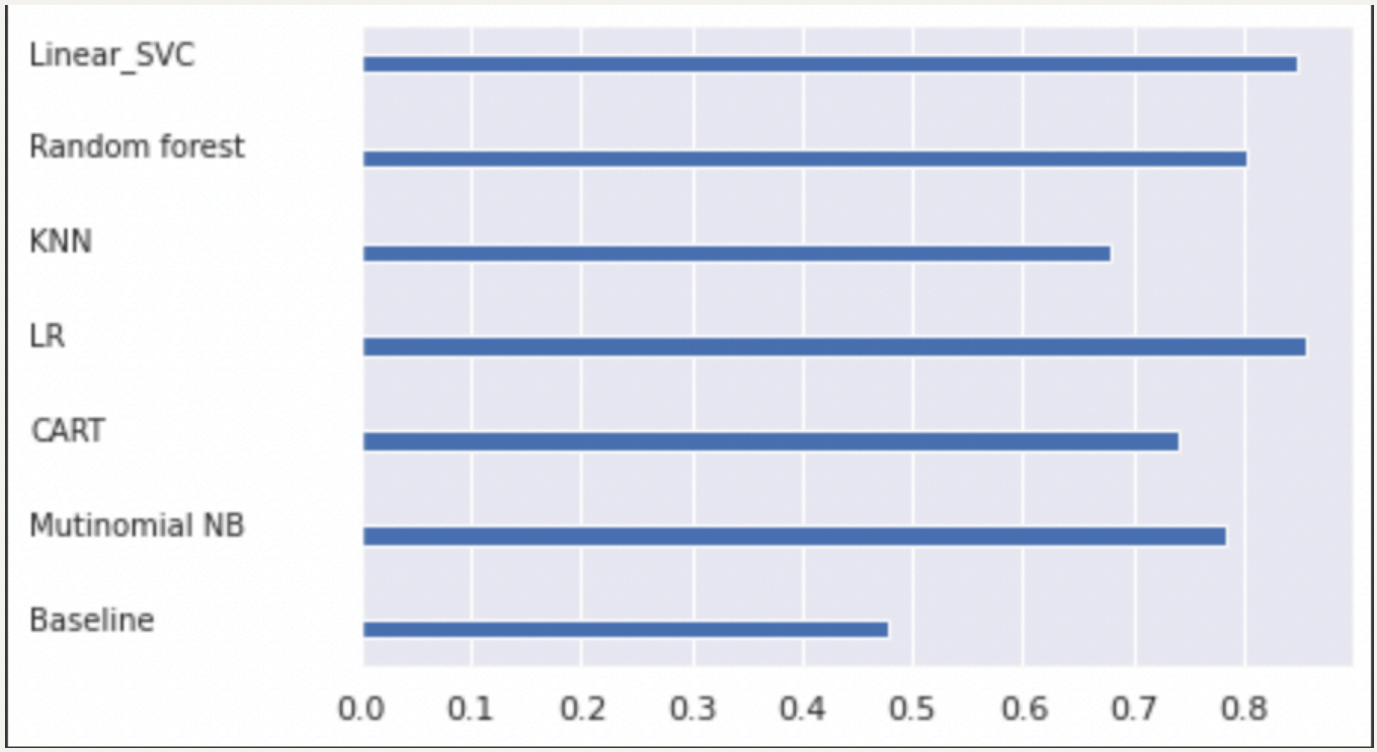
Nous pouvons donc en déduire que les modèles les plus performants que nous pouvons utilisé sont :

- Naive Bayes (il se base sur le théorème de Bayes).
- Linear SVC (Classification linéaire par vecteurs de support).

Nous allons donc effectuer des tests de performances des modèles ci-dessus ainsi que des algorithmes fréquement utilisées pour faire de la classification à l'aide d'une validation croisée à 5 plis.

Je n'ai pas mis l'algorithme GaussianNB() qui correspond à Naive Bayes car après quelques messages échangés avec le professeur, on utilise plutôt l'algorithme MultinomialNB() pour la classification de textes.

Voici un tableau qui montre le taux de performance de chaque algorithme sur le jeu de données de test :



On peut voir que la régression logistique est toujours le meilleur modèle. Nous pensons toutefois que cela est dû à la petite taille du jeu de données. Avec un plus gros jeu de données, il est probable qu'un autre algorithme aurait pris le dessus.

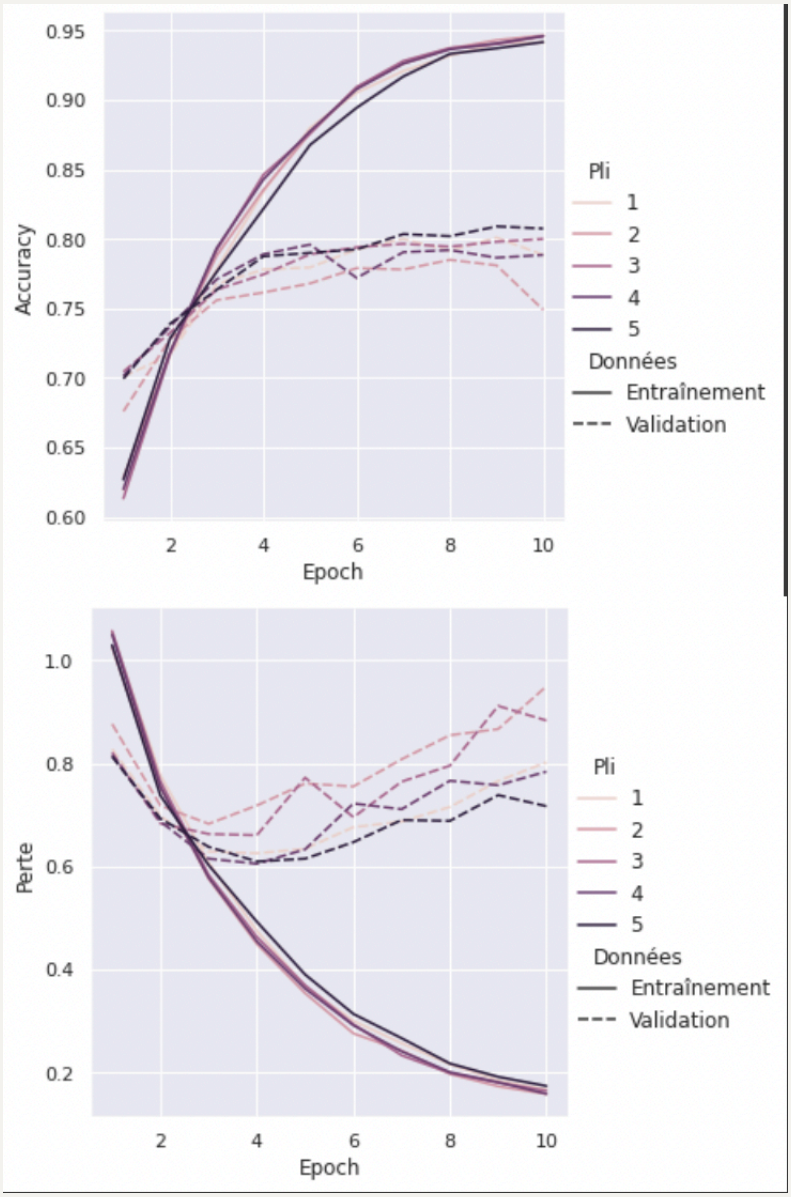
Entraînement uniquement avec une partie des traits :

Comme on pouvait se l'imaginer, lorsqu'on enlève une colonne du jeu de données d'apprentissage, l'acurracy du f1-score devient moins bon. On peut donc dire que chaque colonne est importante au bon déroulement de l'apprentissage. Les résultats sont largement moins bon que ce précedant. On peut aussi dire que la colonne la plus importante pour l'apprentissage est la colonne `test`. La colonne `title` obtient, en effet, de moins bon résultats.

CNN pour la classification de document :

Dans cette partie, nous avons entraîner le même CNN à l'aide à l'aide de deux plongements de mots pré-entrainés.

Dans le premier modèle CNN, nous avons entraîné notre modèle à l'aide d'un dictionnaire anglais. Ce qui n'est pas la meilleure solution pour de l'indexation avec du texte français mais c'est la manière que nous avons vu en TP. Nous avons donc chargé nos données, indexé le vocabulaire, chargé le plongement de mots pré-entrainés et enfin construit puis entraîné notre modèle à l'aide d'une validation croisé à 5 plis (voir le détails dans les commentaires sur le NoteBook en index pour plus d'informations). Pour ce modèle, nous obtenons une `Accuracy: 78.67 (+- 2.02)` et une `Loss: 0.83`, ce qui est déjà un bon résultat mais on peut mieux faire.

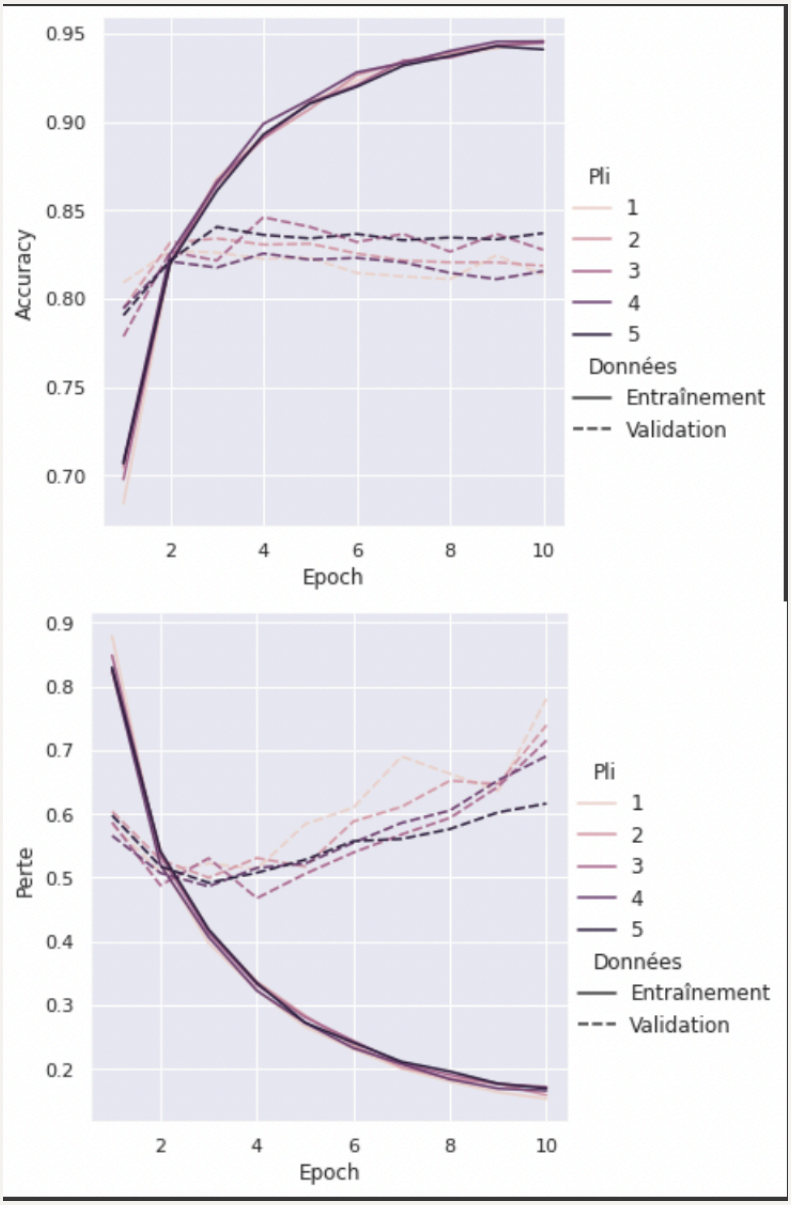


Dans le deuxième CNN, nous avons entraîné notre modèle à l'aide d'un dictionnaire français (fichier binaire que nous avons transformé en fichier text à l'aide d'un algorithme) que nous avons soigneusement modifié et placé sur un git afin de pouvoir l'utiliser dans notre Notebook. Voici le lien ci-dessous :

-> https://git.unistra.fr/asensenbrenner/algo_text_bank

Nous avons utilisé, dans le Notebook, le modèle `plage1.txt`. On a aussi effectuer des tests avec le modèle `plageComplet.txt` mais ils ne sont pas présents dans le Notebook sinon cela le surchargerai.

Nous avons effectué les mêmes étapes que pour le premier CNN (garder la même structure de fabrication du modèle aussi) et nous avons obtenus une `Accuracy: 82.24 (+- 0.87)` et une `Loss: 0.71`, ce qui est mieux que notre premier modèle. On pouvait s'y attendre.



Conclusion :

Le meilleure modèle que nous obtenons pour la classification de documents est le modèle qui utilise **la régression logistique** pour s'entraîner. Nous pensons avoir fait le tour de tout les modèles possibles pour la classification ainsi que les différents types de traits déduits du texte des articles.

BIGOT Aymeric & SENSENBRENNER Amaury