Projet de détection de fake news

1 Groupe

Notre groupe est composé de :

Arnaud COSSU : 21908322
Gatien HADDAD : 21903888
Gabrielle POINTEAU : 21917975
Adam SAID : 21905365

2 Mise en place

2.1 Organisation

Pour pouvoir travailler à plusieurs sur ce projet, nous avons utilisé Deepnote, un notebook collaboratif en ligne. Ce notebook nous permettait, contrairement à Google Colab, de pouvoir éditer simultanément le notebook et pouvoir effectuer des test sur différentes cellules.

2.2 Etude des datasets

Dans un premier temps, nous avons commencé par étudier le jeu de données, et plus précisément le nombre d'occurrences par classe :

Pour rappel, dans la classe "*our rating*", le label ***true*** correspond aux articles considérés comme vrais, et ***false*** comme des fake-news. De plus, le label ***mixture*** correspond à un mélange de vrai et de faux, et ***other*** à une catégorisation qui n'a pas été déterminée.

Classe	Occurences
false	578
true	211
mixture	358
other	117

Comme nous pouvons le remarquer, le jeu de données est assez déséquilibré. Nous avons opté pour une méthode de ***downsampling*** pour rééquilibrer les classes (suppression de données dans la classe majoritaire), car de l'*upsampling* (duplication de données dans la classe majoritaire) aurait pu entrainer un surapprentissage sur certaines données.
Malheureusement, le downsampling va drastiquement réduire la taille des données en entrée, ce qui, nous allons voir plus tard, va poser problème pour les classes minoritaires.

3 Pré-traitements

Nous avons ensuite réalisé une fonction pour automatiser les pré-traitements pour pouvoir les utiliser dans un pipeline. Nous avons donc une fonction *preTraitement*(...) qui regroupe la chaine des pré-traitements avec les différents paramètres en choisissant ceux que l'on veut utiliser ou non. Cette grande fonction sera utilisée dans le pipeline et pour la classification.
Pour plus de détails nous avons 7 prétraitements possibles :

* **Suppression** des textes qui se sont **pas en anglais :** *supp_not_english_words(textes)*

* Génération des ***wordclouds*** pour l'affichage : *generate_wordcloud(texts)*
* Mise sous **forme de token** (dans tous les pré-traitements) : *tokenize_titles(texts)*
* Donner la **catégorie** de tous les mots : *tokenize_titles(texts)*
* **Filtrer** les mots en fonction de leurs **types** pour chaque phrase : *filter_by_pos(text, selective_pos)*
* **Suppression des majuscules** sur les mots qui ne sont **pas des noms propres** ou **tout enlever** : *lowercase_filter(filter_cate_word_text, lower, is_all_min)*
* **Supprimer les stops words** : *remove_stopwords(text)*
* **Lemmatisation des mots** en remplacant par sa racine : *lemmatize_text(text)*

Ces fonctions seront utilisées dans la fonction de pré-traitements en commençant par mettre sous forme de token pour pouvoir traiter chaque string. Puis la fonction applique les pré-traitements dont on a spécifié *true* lors de l'appel

Par exemple : Si on veut un prétraitement qui réalise une sélection de catégorie en prenant les noms, les verbes et les adjectifs de chaque phrase. Puis, on veut supprimer les majuscules de tous les mots, supprimer les *stopWords* ainsi que de mettre sous forme de lemme ; on utilisera la fonction de la façon suivante :

$$\begin{aligned} preTraitement(\ & dfPret[\text{"}title\text{"}] \\ & selective\_pos = [\text{'}NOUN\text{'},\text{'}VERB\text{'},\text{'}ADJ\text{'}], \\ & filter\_category = False, \\ & supp\_maj = True, \\ & supp\_all\_maj = True, \\ & supp\_stopwords = True, \\ & lemmatisation = True\ ) \end{aligned}$$

4 Traitements

4.1 Traitement des données manquantes

Une fois le pré-traitement effectué, nous avons dans un premier temps vérifié s'il y avait des valeurs manquantes dans le *dataset*. Ainsi, comme nous avons constaté qu'il n'en manquait pas, nous avons pu directement transformer les données.

4.2 Transformation en matrice de poids

Nous avons transformé les données pré-traitées en matrices de poids, avec une approche utilisant Tf-IDF.

5 Classification

5.1 Structuration des données

En vue d'entrainer notre classifieur, nous nous sommes dans un premier temps demandé si nous devions concaténer les *titres* avec leurs *textes* respectifs (articles), ou si il était préférable d'entrainer le modèle avec uniquement les *titres* ou uniquement les *textes*.

Pour répondre à cette question, nous avons dans un premier temps testé la classification sur les titres seuls. Comme cette méthode ne produisait pas de résultat satisfaisant dans les partiels suivants, nous avons ensuite entrainé le modèle avec seulement les textes, puis avons concaténé les titres et les textex pour de meilleurs résultats.

5.2 Tests des différents classifieurs

Une fois la transformation en matrice faite, nous avons défini une fonction qui permet de comparer les performances de différents modèles de classification en utilisant un KFold à 10 plis. La fonction applique une classification avec chaque modèle fourni, et calcule les scores de performance avec *classification_report* (accuracy, précision, recall et f1-score, figure 1). Les résultats comparant l'accuracy de chaque classifieur sont également affichés sous forme graphique (figure 2).

```
================================
Evaluation de RandomForest
              precision    recall  f1-score   support

       false       0.81      0.79      0.80       210
        true       0.79      0.81      0.80       210

    accuracy                           0.80       420
   macro avg       0.80      0.80      0.80       420
weighted avg       0.80      0.80      0.80       420

================================
```

Figure 1: Rapport de la classification pour RF

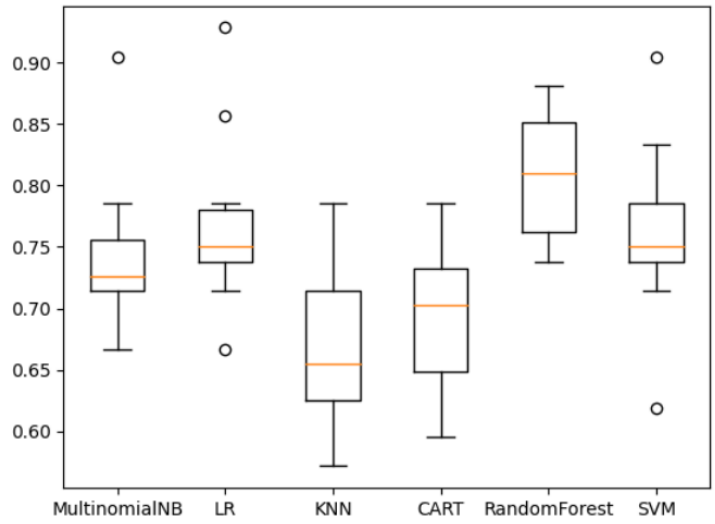

Figure 2: Comparaison des classifieurs en fonction de l'accuracy

6 Résultats finaux

6.1 Création d'un pipeline

Pour pouvoir obtenir des résultats sur les différentes tâches de classification, nous avons créé un pipeline pour effectuer à la chaine les pré-traitements, la transformation du dataset, l'entrainement et les tests sur le modèle.

Nous avons entrainé notre modèle sur la totalité du jeu de données d'entrainement *HAI817_Projet_train.csv*, et ensuite testé ce même modèle à l'aide du jeu de test *HAI817_Projet_test.csv*.

6.2 Résultats selon la tâche de classification

Voici, pour les trois tâches de classification, un tableau regroupant les résultats obtenus avec des valeurs moyennes pour chaque score, suivi de la matrice de confusion de chaque tâche :

	accuracy	**precision**	**recall**	**f1-score**
VRAI vs. FAUX	74,7%	73%	74%	73%
VRAI ou FAUX vs. AUTRE	61.3%	62%	61%	61%
VRAI vs. FAUX vs. AUTRE vs. MIXTE	37.1%	37%	38%	37%

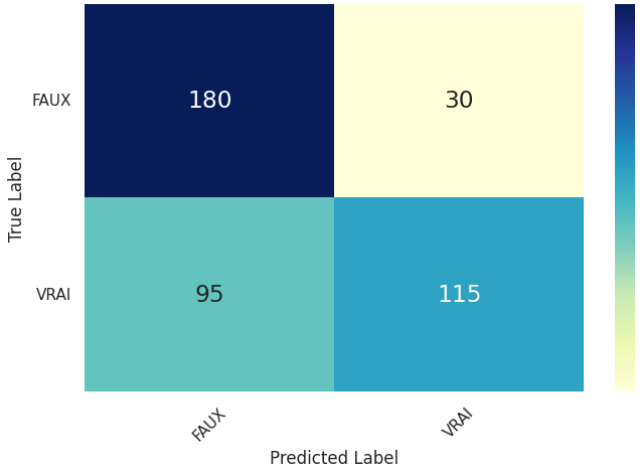

Figure 3: {VRAI} vs {FAUX}

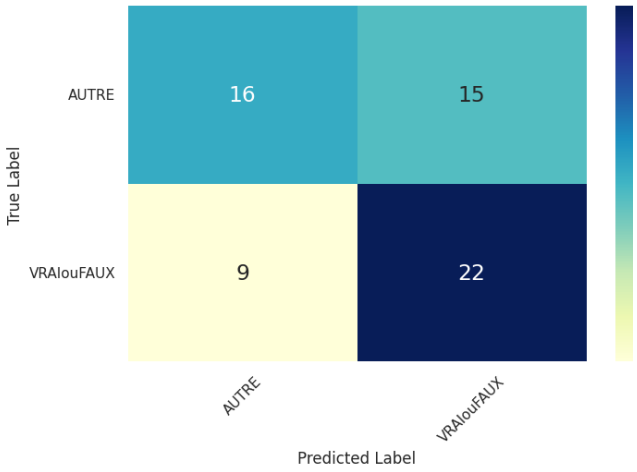

Figure 4: {VRAIouFAUX} vs {AUTRE}

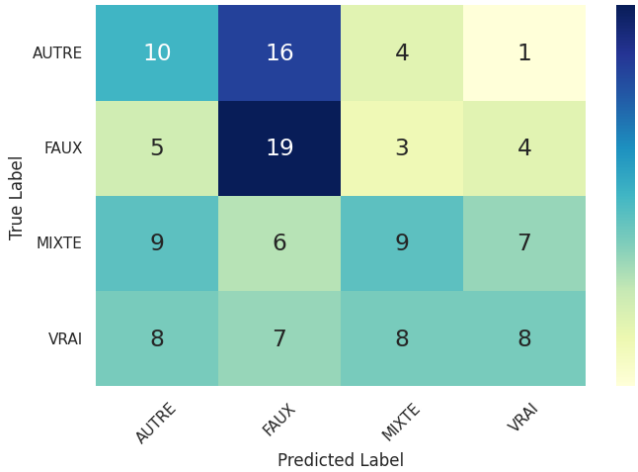

Figure 5: {VRAI} vs {FAUX} vs {AUTRE} vs {MIXTE}

6.3 Analyse des résultats

La première chose à remarquer est la différence des résultats entre la tâche "{VRAI} vs {FAUX}", et les deux autres tâches "{VRAIouFAUX} vs {AUTRE}", et "{VRAI} vs {FAUX} vs {AUTRE} vs {MIXTE}". Ces deux dernières fournissent des résultats très en dessous de ceux de la première classe. Cette différence est en grande partie expliquée par la faible quantité de données disponibles pour les classes "AUTRES" et "MIXTE". Pour essayer de pallier le problème, nous avons essayé de ré-adapter nos pré-traitements, mais la taille des données restait un problème bloquant. Nous avons également essayé d'augmenter artificiellement le jeu de données avec l'*upscaling*, mais comme nous l'avions évoqué dans la première partie de ce rapport, cela a favorisé le sur-apprentissage ; car même si lors des tests avec les k-flod l'accuracy atteignait les 0.95, lors des tests sur le fichier *HAI817_Projet_test.csv* les résultats étaient de nouveau très décevant (voir figure 7 en annexe).

Cependant, comme le montre la matrice de confusion de la figure 3, nous avons obtenu des bons résultats pour la tâche "{VRAI} vs {FAUX}". Les différents tests sur les pré-traitements et le fait que les jeux de données soient un peu plus gros pour ces deux classes nous ont permis d'atteindre ces résultats encourageants, ce qui nous a permis de pouvoir tester le modèle en situation réél en nous attendant à des résultats positifs.

7 Mise en production

En vue de mettre notre modèle en production, nous avons utilisé le pipeline afin d'entrainer le modèle sur la totalité du dataset. Une fois le modèle entrainé, nous le sauvegardons dans un fichier .pkl.

Lors de l'utilisation, nous chargons le modèle enregistré, et l'utilisons pour prédire 6 article non étiquetés. Sur la figure 6, on peut y voir 3 vrais article tiré du New York Times, et 3 fake issus d'un dataset de fake-news (capture d'écran raccourcie pour un gain de place). Ici, notre modèle prédit correctement 5 articles sur 6 comme étant ou non des fake-news.

```
# Articles vrais (New York Time)
title1 = u"""Specter of Trump Loosens Tongues, if Not Purse Strings, in Silic
text1 = u"""After years of scorning the political process, Silicon Valley has
 ...

# Articles faux (WELFake Dataset)
title4 = u"""Schumer calls on Trump to appoint official to oversee Puerto Ric
text4 = u"""WASHINGTON (Reuters) - Charles Schumer, the top Democrat in the U
 ...

predict_fake_news(articles_a_tester, filename)
```

```
L'article 1 est : VRAI
L'article 2 est : VRAI
L'article 3 est : VRAI
L'article 4 est : VRAI
L'article 5 est : FAUX
L'article 6 est : FAUX
```

Figure 6: Prédictions sur des articles non étiquetés

8 Déploiement Web

Pour pouvoir utiliser notre modèle en condition réele et de manière plus pratique en proposant une interface utilisateur ergonomique, nous avons déployé le modèle sur un serveur web.
Nous avions dans un premier temps essayé de le déployer sur un serveur **Flask** à cette adresse : fake-news-detection, mais le serveur n'était pas assez puissant pour rendre un résultat assez rapide, nous avons donc du changer de methode.
Nous nous sommes alors orientés vers un editeur de notebook collaboratif : **Deepnote**. Le site est disponible à l'adresse suivante : deepnote.com/HAI817I-Machine-Learning. Pour run le notebook, il faut simplement avoir un compte créé sur deepnote.

Sur la figure 9 en annexe, nous pouvons voir qu'il suffit d'entrer le titre et le texte d'un article, et le modèle nous prédit si l'article est vrai ou faux. Le résultat est cependant relativement long ($\tilde{1}$ minute) car il faut recharger tout le notebook lors du démarage (imports, chargement du modèle, etc).

9 Conclusion du projet

Pour conclure, la détection de la véracité d'un article de presse est une tâche complexe même avec des modèles de classification. Notre modèle {VRAI} vs {FAUX} a cependant montré de bon résultats lors des tests avec les K-folds, avec une accuracy allant jusqu'à 0,80 grâce à nos pré-traitements et des tests sur les différents classifieurs et leurs hyperparametres.

Une piste d'amélioration serait dans un premier temps d'avoir d'avoir un set de données plus grand, car c'est ce qui a majoritairement posé problème lors de ce projet. Il serait également possible d'explorer des modèles plus sophistiqués telles que les réseaux de neurones, ou encore d'utiliser des techniques de sélection de caractéristiques (feature selection) pour réduire la dimensionnalité de l'espace de caractéristiques et améliorer les performances du modèle.

10 Annexe

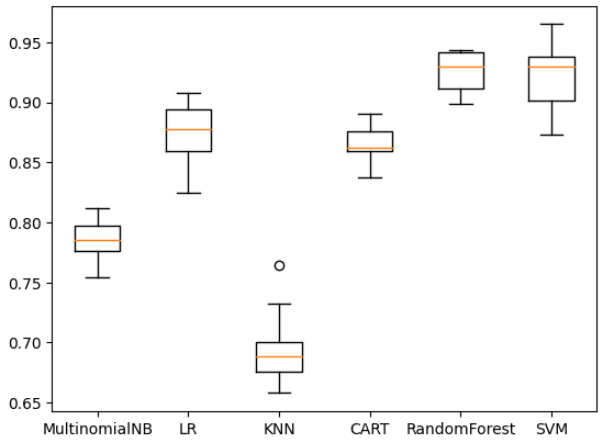

Figure 7: Classifieurs entrainés un Kfold

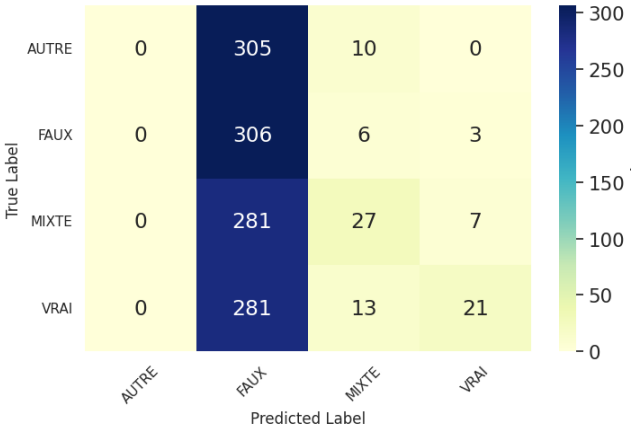

Figure 8: Modèle testé avec le jeu de test

Comparaison de {VRAI} vs {FAUX} vs {AUTRE} vs {MIXTE} avec l'upscaling

Figure 9: Interface Web pour tester l'outil de détection de fake news