Rapport Défi 3

Nicolas Melaerts
253882

Janvier 2024

1 Introduction

Les fake news, représentent un enjeu majeur dans notre société actuelle, car elles peuvent influencer l'opinion publique, altérer les processus démocratiques, et amplifier la désinformation à grande échelle. Détecter ces fausses informations de manière automatisée est devenu essentiel pour lutter contre leur propagation rapide, en particulier sur les réseaux sociaux.

Ce projet vise à mettre en œuvre deux pipelines distincts pour la détection de fake news. Le premier repose sur des classificateurs traditionnels utilisant des algorithmes d'apprentissage supervisé, tandis que le second exploite des modèles pré-entraînés de type transformers.

L'objectif est de comparer leurs performances sur un ensemble de données que j'ai enrichi par un processus de data augmentation.

2 Processus de data augmentations

L'ensemble de code que j'ai réalisé pour la data augmentation se trouve dans le dossier **data augmentation**.
J'ai effectué de la data augmentation sur le fichier **fake_train.csv** pour augmenter la quantité de données disponibles pour l'entraînement de mon modèle de classification de true news et fake news. L'objectif de cette augmentation est d'avoir un dataset composé d'un nombre suffisant de news variés pour améliorer la performance des modèles mis en place ainsi qu'améliorer leur capacité à généraliser sur des données inédites. Étant donné le nombre limité de news (1457), je voulais limiter les risques d'overfitting.
J'ai utilisé un ensemble de données trouvé sur Internet, le **True-Fake News v2**, disponible sur Hugging Face. Ce dataset contient des articles d'actualités, chacun étant étiqueté comme soit vrai, soit faux. Mais ce dataset était uniquement en anglais, ce qui allait limiter sa capacité à bien fonctionner pour les tests qui se font sur des news en français. Donc, pour remédier à cette limitation et augmenter mes données, j'ai mis en place un processus de traduction en ajoutant au fichier fake_train.csv des news de True-Fake News v2 traduites en français.
J'ai utilisé une bibliothèque de traduction, en intégrant une gestion d'erreurs pour garantir que seules les traductions réussies soient ajoutées à l'ensemble de données. Pour implémenter cela de manière efficace, j'ai créé un script qui charge l'ensemble de données, puis traduit chaque news une à une. Une fois la traduction terminée, chaque news est sauvegardée dans un fichier csv. Et puis j'ajoute ce csv à fake_train.csv.
Ce processus permet de diversifier l'ensemble des données sans collecte manuelle qui pourrait prendre énormément de temps.
J'ai donc travaillé avec 3 datasets :

- Le dataset train fourni
- Le dataset train + 3000 news
- Le dataset train + 10000 news

3 Pipeline 1 : classificateurs traditionnels basés sur des algorithmes d'apprentissage supervisé

L'ensemble du code se trouve dans le dossier **pipeline 1**

3.1 Entraînement :

J'ai choisi de tester plusieurs modèles d'apprentissage supervisé sur l'ensemble de données pour évaluer leur performance en fonction de différents paramètres. L'objectif était de comparer ces modèles et sélectionner celui qui offre les meilleures performances.
Petite explication des modèles utilisés :

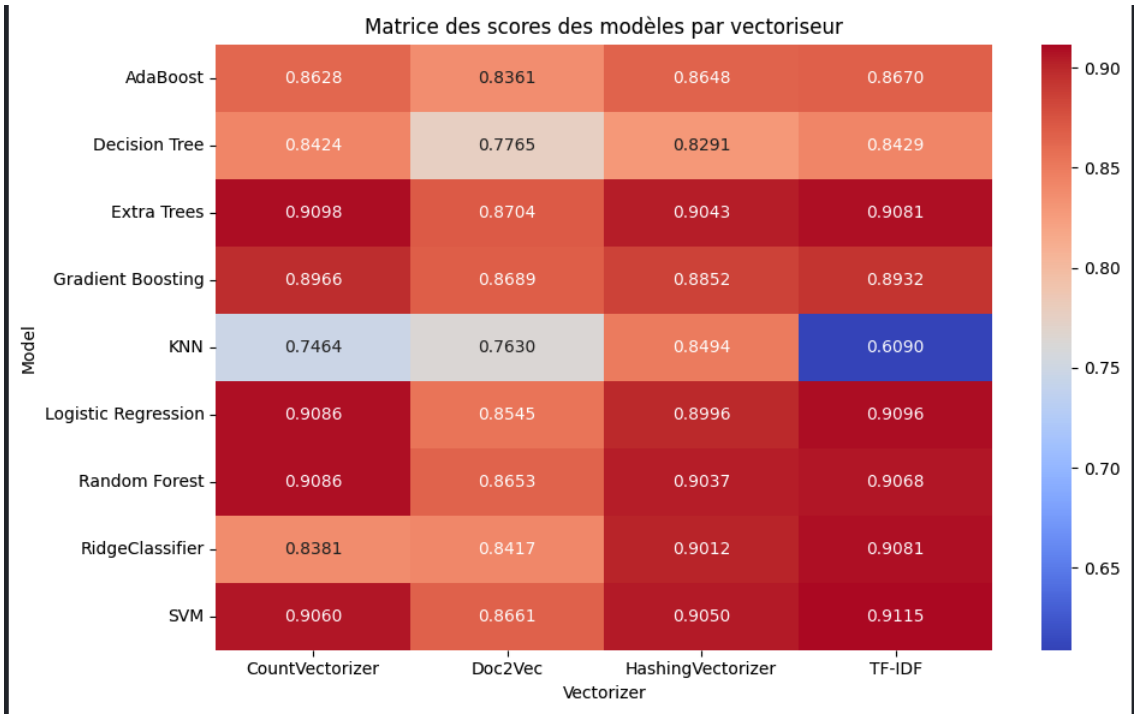
- **SVM** (Support Vector Machine) utilise un hyperplan pour séparer les deux classes. Un noyau linéaire est utilisé ici.
- **Random Forest** utilise un ensemble d'arbres de décision entraînés sur des sous-ensembles de données. Chaque arbre vote pour une classe, et la classe la plus votée est élue.
- **Logistic Regression** utilise un modèle statistique qui utilise une fonction logistique pour estimer la probabilité d'appartenance à une classe.
- **KNN** (K-Nearest Neighbors) classifie les données en fonction de la majorité des classes parmi les k voisins les plus proche
- **Gradient Boosting** est une méthode d'assemblage qui combine plusieurs modèles faibles (arbres de décision) de manière séquentielle, chaque modèle corrige les erreurs du précédents
- **AdaBoost** est un algorithme de boosting qui combine plusieurs classificateurs faibles en un classificateur fort, en mettant davantage l'accent sur les erreurs des modèles précédents
- **Extra Trees** est un ensemble d'arbres de décision qui utilise un sous-échantillon aléatoire pour chaque arbre et fait une sélection aléatoire parmi les caractéristiques.
- **Decision Tree** est un arbre de décision classique qui divise les données en fonction de règles de décision basées sur les caractéristiques.
- **RidgeClassifier** est une variante de la régression logistique qui utilise une régularisation L2 pour éviter l'overfitting

J'utilise un processus de cross-validation (CV) avec **GridSearchCV** qui permet de tester différentes combinaisons d'hyperparamètres pour chaque modèle afin de maximiser ses performances. Le modèle ayant la meilleure combinaison de paramètres, basé sur un score de précision moyen est sélectionné. Je teste également chaque modèle sur différent vectoriseur.

- **CountVectorizer** : Transforme les textes en vecteurs en comptant la fréquence d'apparition de chaque mot dans le corpus.
- **TF-IDF** : Mesure l'importance de chaque mot dans un document en tenant compte de sa fréquence dans le document et dans l'ensemble du corpus.
- **HashingVectorizer** : Applique une fonction de hachage pour transformer le texte en un espace de caractéristiques fixe, permettant de traiter de grands volumes de texte de manière efficace.
- **Doc2Vec** : Génère des vecteurs denses représentant des documents entiers, prenant en compte le contexte global des mots dans le document.

3.2 Résultat de l'entraînement :

Je lance donc un entraînement complet pour trouver quel est le modèle et le vectoriseur qui me donne les meilleurs résultats sur les données d'entraînement.



On constate que le vectoriseur **Doc2Vec** donne des résultats assez faible par rapport aux autres vectoriseur. On constate également que **KNN** donne aussi des résultats assez faible. Le vectoriseur dominant est clairement **TF-IDF** qui obtient la meilleure accuracy pratiquement partout et qui à la meilleure accuracy avec le modèle **SVM**. J'exporte donc le modèle **SVM** et le vectoriseur **TF-IDF** pour l'utiliser dans le script de test.

L'entraînement à été effectué avec le dataset train + 10000 news.

3.3 Test :

Maintenant, dans le fichier de test, avec le modèle **SVM** et le vectoriseur **TF-IDF** exporté on peut trouver l'accuracy avec notre dataset de test. On obtient un résultat de **0.9383**.
Pour changer le dataset de test il suffit de le changer changer ici :

```
test_file_path = 'fake_test.csv'
```

4 Pipeline 2 : basé sur des modèles pré-entrainer de type transformers

L'ensemble du code se trouve dans le dossier **pipeline 2**

4.1 Entraînement :

Pour le pipeline 2, j'ai choisi d'utiliser différents modèles trouvés sur Hugging Face pour cette tâche de détection de fake news :

- **CamemBERT** (camembert-base) qui est une variante de BERT optimisée pour le français, entraînée sur un corpus de texte français très vaste. Sa spécialisation en français permet de mieux capturer les subtilités linguistiques propres au français.

- **BERT** (dbmdz/bert-base-french-europeana-cased) est une version de BERT conçue pour le français, entraînée sur le corpus Europeana, ce qui en fait un modèle pertinent pour les textes formels ou issus de médias.
- **RoBERTa** (hamzab/roberta-fake-news-classification) est modèle spécifiquement pré-entraîné pour la classification de fake news. Bien que entraîné sur un dataset en anglais je voulais tester ses performances.
- **XLM-RoBERTa** (xlm-roberta-base) Un modèle multilingue capable de gérer plusieurs langues, y compris le français. Il est deux fois plus grand que les trois autres en taille.

J'ai entraîné ces quatre modèles avec 10 epochs, un batch size de 16 et un learning rate de 0.00002 sur le dataset train. Puis, à cause de contraintes techniques et de temps sur Google Colab, j'ai dû me limiter à un entraînement de 3 epochs pour mes deux autres datasets qui sont bien plus conséquents.
L'ensemble de mes modèles entraînés se trouve dans les dossiers **train_with_basic_dataset** qui sont les entraînements avec le fichier train fourni et dans **train_with_3000_dataset** avec le dataset train + 3000 news en plus. Dans le dossier courant, est placé **bert-base-french-europeana-cased.zip** qui est mon meilleur modèle et qui peut être testé sur le dataset de test.
Pour changer le dataset de test il suffit de le changer ici :

```
test_file_path = 'fake_test.csv'
```

4.2 Tests :

Voici les résultats de mes différents entraînements que j'obtiens :

Modèle	Dataset Normal	Dataset Augmenté (3000)	Dataset Augmenté (10000)
camembert-base	0.9794	0.9609	-
bert-base-french-europeana-cased	0.8477	0.9650	0.9712
roberta-fake-news-classification	0.5700	0.9342	-
xlm-roberta-base	0.7078	0.9547	-

Table 1: Résultats des différents modèles sur des datasets normaux et augmentés (3000 et 10000). Les valeurs représentent l'accuracy.

Les résultats obtenus montrent que le modèle CamemBERT (camembert-base) atteint la meilleure accuracy (0.9794) sur le dataset normal. Cependant, cette performance élevée sur le dataset normal s'explique par la similarité entre les données d'entraînement et de test, ce qui pourrait biaiser les performances si le modèle était confronté à des données complètement nouvelles.
Pour des raisons de contraintes de temps liées à l'entraînement, un seul modèle (BERT) a été testé avec le dataset de 10000 news (car c'était le meilleur avec 3000 news). Ce modèle a obtenu une accuracy de 0.9712, confirmant que l'ajout de données supplémentaires peut renforcer les capacités de généralisation du modèle.

5 Conclusion

Ce projet a permis de mettre en œuvre et de comparer deux approches distinctes pour la détection de fake news : un pipeline basé sur des classificateurs traditionnels et un pipeline exploitant des modèles pré-entraînés de type transformers.

Les résultats obtenus montrent que les modèles de type transformers, en particulier BERT et CamemBERT, surpassent les classificateurs traditionnels en termes de précision, notamment sur les datasets augmentés. Cela souligne la capacité des modèles pré-entraînés à capturer les subtilités linguistiques et contextuelles des textes, surtout dans le cadre de langues spécifiques comme le

français. Cependant, les classificateurs traditionnels, lorsqu'ils sont associés à un vectoriseur performant comme TF-IDF, restent des solutions intéressantes pour des environnements à ressources limitées ou des données de taille réduite. Ils ont également l'avantage de pouvoir être utiliser en local sur mon ordinateur, tandis que pour les modèles de types transformers je dois utiliser Google Colab pour utiliser un GPU. Il existe évidemment encore des pistes d'amélioration comme encore augmenter le dataset avec des news pour augmenté la généralisation. Il peut aussi être envisager d'utiliser la librairie **Optuna** pour chercher les meilleurs hyperpamètres pour les modèles transformers. Et puis on pourrait étendre ce projet à d'autres langues. En conclusion, ce projet met en évidence l'efficacité des modèles transformers pour la détection de fake news, tout en soulignant le rôle complémentaire des classificateurs traditionnels.

6 Annexe

- **True-Fake News v2** : `https://huggingface.co/datasets/AlexanderHolmes0/true-fake-news-v2`
- **camembert-base** : `https://huggingface.co/docs/transformers/model_doc/camembert`
- **dbmdz/bert-base-french-europeana-cased** : `https://huggingface.co/dbmdz/bert-base-french-euro`
- **hamzab/roberta-fake-news-classification** : `https://huggingface.co/hamzab/roberta-fake-news-cla`
- **xlm-roberta-base** : `https://huggingface.co/docs/transformers/model_doc/xlm-roberta`