Université de Montpellier
Faculté des Sciences

Projet Machine Learning HAI817

Détection automatique des fake news à partir de données textuelles (Fake News Detection)

Master 1 Informatique parcours Génie Logiciel

Réalisé par :
Louis BALANDRAS 20132305
Lydia BEDRI 22316619
Sara ZIDOUNE 22317124
Alexandre SAR 21900476
Mohamad EL-KAAKOUR 22309292

Enseignants :
Pascal PONCLET
Ensiyeh RAOUFI

Année universitaire : 2023-2024

Table des matières

1 Introduction

Dans le cadre de ce projet, nous avons entrepris plusieurs étapes pour préparer et analyser un dataset de fake news provenant de sources fiables de fact-checking. Le prétraitement des données, essentiel pour obtenir un dataset de qualité, a été réalisé dans un notebook spécifique intitulé `Pretraitement.ipynb`. Ce notebook détaille toutes les étapes de nettoyage, de normalisation et de transformation des textes.

Pour chaque tâche de classification, nous avons utilisé des notebooks séparés afin de structurer notre travail de manière organisée et claire. Ces tâches de classification sont les suivantes :

— La classification binaire entre "True" et "False", documentée dans `True_vs_False.ipynb`.

— La classification binaire "True or False" vs. "Other", documentée dans `True_or_False_vs_Other.ipynb`.

— La classification multi-classes entre "True", "False", "Mixte" et "Other", documentée dans `True_vs_False_vs_Mixte_vs_Other.ipynb`.

2 Prétraitement

Pour préparer les données textuelles avant la phase de vectorisation, plusieurs étapes de prétraitement ont été appliquées. Ces étapes visent à nettoyer, normaliser et structurer les textes des articles pour les rendre adaptés à l'analyse et à la modélisation.

1. **Suppression des colonnes inutiles :** Nous avons identifié les colonnes `public_id` et `ID` comme non pertinentes pour notre analyse et les avons supprimées du dataframe.
2. **Gestion des valeurs manquantes :** Nous avons décidé d'éliminer les lignes contenant des valeurs nulles dans notre dataframe puisqu'elles représentent moins de **5%** afin d'assurer la qualité des données pour l'analyse ultérieure.
3. Fusion de `title` et `text` : Fusionner les colonnes `title` et `text` en une nouvelle colonne `combined`.
4. **Conversion en minuscules :** Tous les textes dans les colonnes `title` et `text` ont été convertis en minuscules pour normaliser la casse des mots.
5. **Suppression des stopwords (anglais) :** Les stopwords en anglais (mots courants tels que "the", "of", "and") ont été supprimés des textes pour réduire le bruit et améliorer la pertinence des mots dans le contexte de l'analyse en anglais.
6. **Lemmatisation :** Les mots ont été ramenés à leur forme canonique (lemmes) pour normaliser le vocabulaire et regrouper les mots apparentés.

Remarque :

Les prétraitements mentionnés ci-dessus représentent les étapes de nettoyage et de préparation initiales des données textuelles. Cependant, lors de la phase de classification, nous avons testé différentes combinaisons de prétraitements pour évaluer leur impact sur les performances des modèles. Voici la liste des ces combinaisons :

— **Combined :** Textes de `title` et `text` après fusion sans aucune modification additionnelle.

— **Combined avec conversion en minuscules :** Tous les textes ont été convertis en minuscules.

— **Combined sans stopwords :** Suppression des stopwords (mots courants tels que "the", "of", "and").

— **Combined avec lemmatisation :** Application de la lemmatisation pour ramener les mots à leur forme canonique.

— **Combined avec toutes les transformations :** Application combinée de toutes les transformations ci-dessus (minuscules, suppression des stopwords, lemmatisation).

3 Classification

3.1 Classification 1 : True vs. False

Pour la première classification entre "true" et "false", nous avons dû éliminer les lignes avec un rating égal à "mixture" ou "other" pour ne conserver que celles avec les valeurs "true" ou "false". Ensuite, nous avons catégorisé les ratings en attribuant 1 à "true" et 0 à "false". Dans cette classification nous utilsons la colone rating true vs false.

3.2 Classification 2 : True ou False vs. Other

Dans cette classification, nous avons regroupé les catégories "True" et "False" sous une seule classe "True or False", que nous avons comparée à la catégorie "Other". Les ratings avec le label "Mixture" ont été supprimés pour simplifier le problème en une classification binaire, facilitant ainsi l'analyse des performances des différents modèles.

Les étapes suivies pour cette classification sont résumées ci-dessous. Chaque étape a été effectuée en variant les prétraitements appliqués aux textes. Les différentes configurations de prétraitement sont celles décrites dans la section 2 Prétraitements.

Étapes de la classification

1. **Vectorisation :** Les textes ont été vectorisés à l'aide de différentes techniques, notamment `CountVectorizer` et `TfidfVectorizer`, avec et sans l'utilisation de `n-grammes`.
2. **Équilibrage des Données :** Étant donné le déséquilibre initial des classes (1302 occurrences de la classe 0 contre 146 occurrences de la classe 1), nous avons appliqué la technique de suréchantillonnage `SMOTE` pour équilibrer les classes pour obtenir 264 occurrences de la classe 0 contre 257 de la classe 1.
3. **Séparation des Données en train set et de test set :** Les données équilibrées ont ensuite été divisées en ensembles d'entraînement et de test pour les différentes configurations de vectorisation.
4. **Modélisation et évaluation :** Nous avons utilisé les algorithmes de classification cités ci-dessus pour évaluer les performances de notre modèle.

3.3 Classification 3 : True vs. False vs. Mixture vs. Other

Dans cette section, nous abordons la tâche de classification multi-classes visant à distinguer entre quatre catégories : "True", "False", "Mixture" et "Other".

Pour cette classification, nous avons utilisé la colonne "rating_all" qui contient les étiquettes pour les quatre classes. Nous avons veillé à ce que chaque étiquette soit correctement attribuée et que les données soient équilibrées autant que possible.

Nous avons exploré différentes approches pour résoudre ce problème, en utilisant des techniques avancées de vectorisation et de modélisation. La vectorisation des textes a été effectuée en utilisant à la fois des techniques de Count Vectorizer et de TF-IDF Vectorizer, permettant de transformer les données textuelles en caractéristiques numériques exploitables par les modèles de machine learning.

4 Comparaison des résultats

4.1 Cas Classification 1 : True vs. False

Dans un premier temps nous avons evaluer nos données avec le model Logistic regression. Le meilleur pipeline que nous avons trouvé pour obtenir les meilleurs résultats est le suivant : nous avons utilisé le vectoriseur TF-IDF, auquel nous avons appliqué des n-grams de taille 2. Ensuite, nous avons équilibré les données et recherché les meilleurs paramètres pour le modèle de régression logistique avec le vectoriseur TF-IDF. Nous avons obtenu les meilleurs résultats avec les prétraitements combinant la lemmatisation et la suppression des stopwords.

Voici les résultats obtenus pour les 5 différents prétraitements en utilisant ce pipeline sur chaque vectoriseur :

```
Dataset BoW: combined, Test Accuracy: 0.8816901408450705, Cross-Validation Mean Accuracy: 0.85182152988603, Std: 0.01967746473407157
Dataset BoW: without_stopwords, Test Accuracy: 0.8845070422535212, Cross-Validation Mean Accuracy: 0.85393918279998, Std: 0.022250330940948474
Dataset BoW: with_lowercase, Test Accuracy: 0.8816901408450705, Cross-Validation Mean Accuracy: 0.85182152988603, Std: 0.01967746473407157
Dataset BoW: with_lemmatization, Test Accuracy: 0.8845070422535212, Cross-Validation Mean Accuracy: 0.8574503558453193, Std: 0.024800416063605644
Dataset BoW: with_all, Test Accuracy: 0.8929577464788733, Cross-Validation Mean Accuracy: 0.8496765042552132, Std: 0.005878258784941342
```

FIGURE 1 – Resultats du pipe line avec le vectoriseur BOW

```
Dataset: combined, Test Accuracy: 0.9126760563380282, Cross-Validation Mean Accuracy: 0.8828522370974966, Std: 0.013640025998917754
Dataset: without_stopwords, Test Accuracy: 0.9211267605633803, Cross-Validation Mean Accuracy: 0.8779077290598716, Std: 0.011983386439371344
Dataset: with_lowercase, Test Accuracy: 0.9126760563380282, Cross-Validation Mean Accuracy: 0.8828522370974966, Std: 0.013640025998917754
Dataset: with_lemmatization, Test Accuracy: 0.8929577464788733, Cross-Validation Mean Accuracy: 0.8849798437266709, Std: 0.02020812049436113
Dataset: with_all, Test Accuracy: 0.9352112676056338, Cross-Validation Mean Accuracy: 0.8870800776389787, Std: 0.014843173498271429
```

FIGURE 2 – Resultats du pipe line avec le vectoriseur TF-IDF

Avec ces résultats nous pouvons constater que la conversion des textes en minuscules (with_lowercase) n'a eu aucun impact significatif sur l'accuracy des modèles, les résultats étant identiques à ceux du dataset combined, ce qui suggère que cette technique de prétraitement n'apporte pas de valeur ajoutée significative.

En revanche, l'élimination des stopwords (without_stopwords) a légèrement amélioré les performances des modèles.

L'application de la lemmatisation (with_lemmatization) a parfois réduit l'efficacité du modèle. Cela peut s'expliquer par la perte d'information contextuelle que certaines formes des mots peuvent apporter.

Toutefois, lorsque la lemmatisation était associée à l'élimination des stopwords (with_all), une amélioration notable des performances a été observée. Cette combinaison a permis d'obtenir la meilleure accuracy parmi les différentes configurations testées, montrant que l'intégration de plusieurs techniques de prétraitement peut être particulièrement bénéfique.

Globalement, les modèles utilisant TF-IDF ont surpassé ceux utilisant BoW. Par exemple, le dataset with_all a obtenu une test accuracy de 0.9352 avec TF-IDF, contre 0.8929 avec BoW.

Impact de la recherche d'hyperparamètres

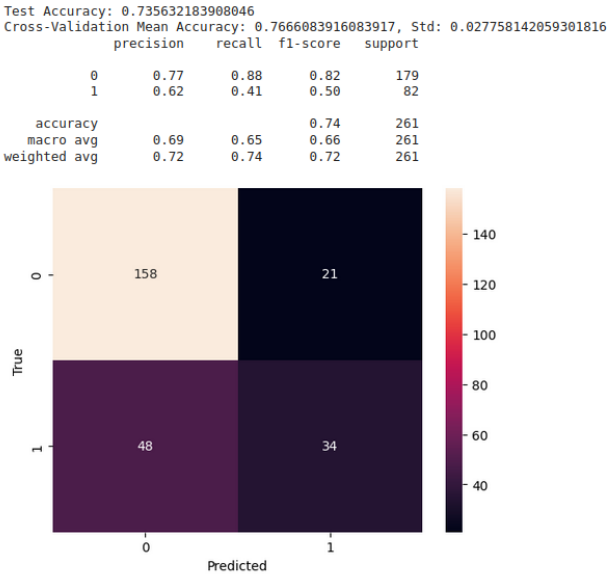

FIGURE 3 – Impact de la recherche des hyperparametre du modèle

L'image ci-dessus illustre clairement la diminution des performances du modèle sans l'utilisation d'une recherche d'hyperparamètres. L'accuracy est plus basse et les scores de précision, rappel et F1 sont déséquilibrés, indiquant une classification moins efficace.

Impact de la reduction de dimension des vecteurs

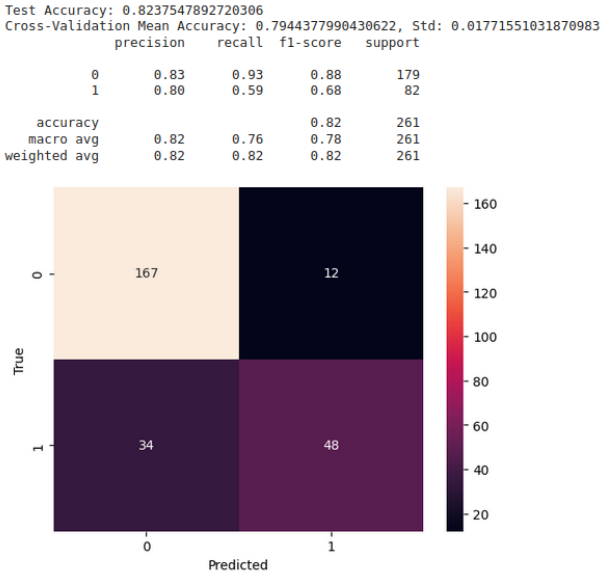

FIGURE 4 – Impact des reductions de dimension des vecteurs

La réduction de dimension des vecteurs peut parfois aider à améliorer les performances des modèles en éliminant le bruit et en réduisant la complexité des données. Cependant, dans certains cas, elle peut également entraîner une perte d'information cruciale, affectant négativement les performances du modèle. C'est le cas ici puisque lorsqu'on enlève la reduction de dimension on observe clairement une amélioration des performences du modèles. Par conséquent, nous avons décidé de ne pas utiliser cette méthode dans notre pipeline final afin de maximiser les performances de notre modèle de classification de texte.

Impact de l'égalisation des données

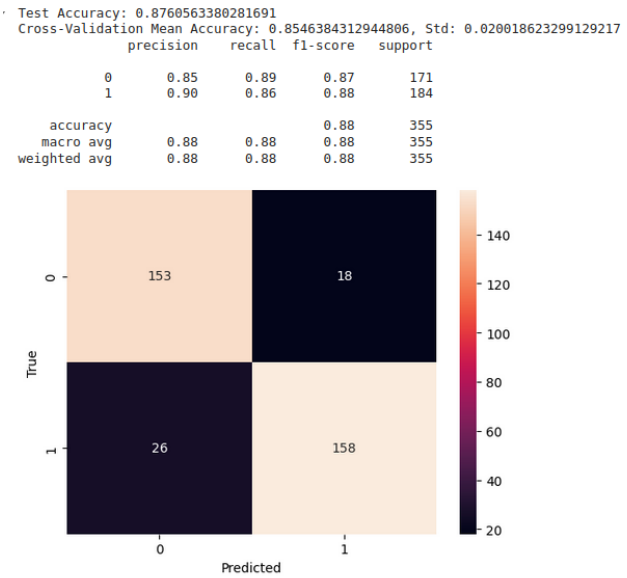

FIGURE 5 – Impact de l'égalisation des données

L'image ci-dessus illustre clairement l'amélioration des performances du modèle avec l'utilisation de l'égalisation des données. L'accuracy est un peu plus élevée et les scores de précision, rappel et F1 sont plus équilibrés, indiquant une classification plus efficace et plus juste entre les classes. Par conséquent, cette étape est essentielle pour obtenir des performances optimales et éviter les biais dans les modèles.

Impact des n-grams

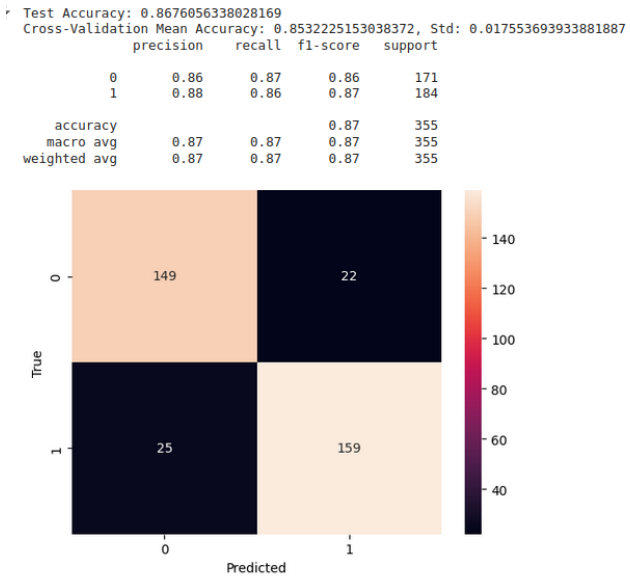

FIGURE 6 – Impact des n-grmas

L'image ci-dessus illustre l'amélioration des performances du modèle avec l'utilisation de bigrams. Bien que l'utilisation de bigrams ait montré une amélioration des performances, nous aurions pu potentiellement augmenter encore plus l'efficacité du modèle en utilisant des trigrams (n-grams de taille 3). Cependant, le temps de traitement pour les trigrams est considérablement plus long en raison de la complexité accrue et de la taille plus importante des vecteurs de caractéristiques générés. Par conséquent, nous avons décidé de nous limiter aux bigrams pour maintenir un équilibre entre la performance du modèle et le temps de traitement.

Test du modèle RandomForest avec notre pipeline

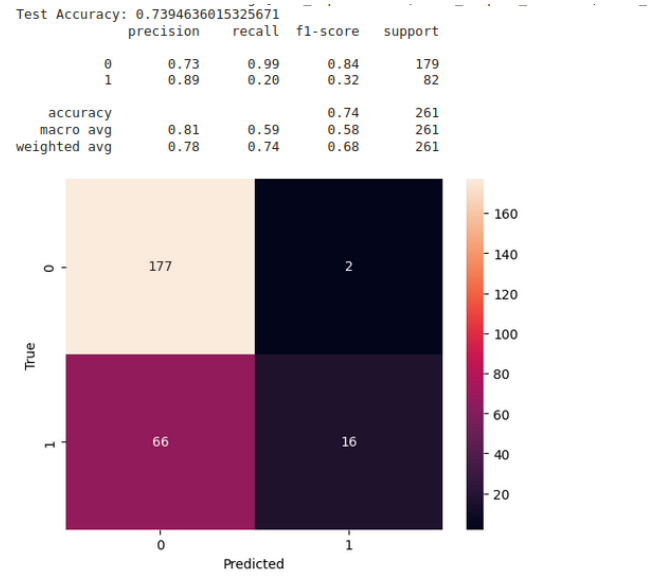

FIGURE 7 – Resultat avec RandomForest

Afin d'explorer l'efficacité de différents modèles de classification, nous avons également essayé d'utiliser le modèle Random Forest avec le même pipeline que précédemment utilisé pour la régression logistique. Cependant, les résultats obtenus étaient nettement moins bons.

Les résultats montrent que le modèle Random Forest, lorsqu'il est utilisé avec le même pipeline que la régression logistique, n'est pas aussi performant. Il y a plusieurs raisons possibles pour expliquer ces performances inférieures :

— **Inadéquation du Pipeline :** Le pipeline que nous avons utilisé, bien qu'optimal pour la régression logistique, peut ne pas être adapté aux spécificités du modèle Random Forest.

— **Moins Efficace pour cette Classification :** Il est également possible que le modèle Random Forest soit tout simplement moins efficace pour ce type de classification textuelle par rapport à la régression logistique.

Test du modele RandomForest avec notre pipeline

4.2 Cas Classification 2 : True ou False vs. Other

Les résultats d'évaluation selon l'accuracy de nos 3 classifiers avec différents techniques de prétraitements et vectorizers utilisés se résument dans le tableau 1.

Les lignes représentent les différents modèles de classification (LR : Linear Regression ; RF : Random Forest et SVM : Support Vector Machines) avec les differents victorizers (sans et avec ngrams = (1,2)) et les colonnes représentent les différents prétraitements effectués.

Classifier/Victorizer	combined	without_stopwords	with_lowercase	with_lemmatization	with_all
LR_TFIDF	0.9865	0.9884	0.9865	0.9827	0.9865
LR_TFIDF-(1,2)	0.9808	0.9827	0.9808	0.9846	**0.9884**
LR_BOW	0.9251	0.9270	0.9270	0.9309	0.9270
LR_BOW-(1,2)	0.9366	0.9347	0.9366	0.9328	0.9251
RF_TFIDF	0.9769	0.9827	0.9788	0.9750	0.9827
RF_TFIDF-(1,2)	0.9673	**0.9846**	0.9654	0.9712	0.9808
RF_BOW	0.9635	0.9500	0.9635	0.9635	0.9577
RF_BOW-(1,2)	0.9404	0.9309	0.9462	0.9404	0.9309
SVM_TFIDF	**0.9942**	**0.9942**	**0.9942**	**0.9942**	**0.9942**
SVM_TFIDF-(1,2)	**0.9942**	**0.9942**	**0.9942**	0.9923	0.9923
SVM_BOW	0.9500	0.9481	0.9481	0.9404	0.9328
SVM_BOW-(1,2)	0.9424	0.9328	0.9424	0.9443	0.9328

TABLE 1 – Résultats des accuracies de la Classification 2 avec les differents classifers, vectorizers et prétraitements

Les résultats montrent que les modèles basés sur la **vectorisation TF-IDF** avec des **n-grammes (1,2)** ont généralement de meilleures performances que ceux basés sur BoW. Parmi les classifiers, les **SVM** et la régression logistique se sont distingués comme les plus performants pour la classification binaire "True or False vs Other". Le modèle **SVM** avec **TF-IDF** et **n-grammes (1,2)** a donné les meilleurs résultats globaux, avec une **précision** de **99,42%**.

4.3 Cas Classification 3 : True vs. False vs. Mixture vs. Other

Résultat de l'accuracy pour le modèle LogesticRegression avec la vectorisation tf-idf. Nous retrouvons les différents prétraitement effecutués :

Jeu de Données	Test Accuracy
Combined	0.8463
Without Stopwords	0.8604
With Lowercase	0.8463
With Lemmatization	0.8350
With All Preprocessings	0.8829

Table 2 – Résultats des accuracies de la Classification 3, modèle LogesticRegression et vectorisation tf-idf

Résultat de l'accuracy pour le modèle LogesticRegression avec la vectorisation bow :

Jeu de Données	Test Accuracy
Combined	0.8082
Without Stopwords	0.8011
With Lowercase	0.8082
With Lemmatization	0.7969
With All Preprocessings	0.7955

Table 3 – Résultats des accuracies de la Classification 3, modèle LogesticRegression et vectorisation bow

L'utilisation de BoW avec LogisticRegression sans paramètres avancés a donné les résultats suivant pour la colonne "combined" :

Mesure	Valeur
Test Accuracy	0.5903

Table 4 – Performances de BoW sans paramètres avancés.

L'équilibrage des données avec SMOTE a amélioré les performances globales avec LogisticRegression pour la colonne "combined" :

Mesure	Valeur
Test Accuracy	0.623

Table 5 – Performances de BoW sans paramètres avancés.

LogisticRegression en utilisant Bow sans inclure de 2-grams.

Mesure	Valeur
Test Accuracy	0.588

Table 6 – Accuracy du modèle

Nous comparons les performances d'accuracy de deux autres classifieurs : Random Forest et Support Vector Machine (SVM). Utilisation de tf-idf et n gramme.

Classifieur	Test Accuracy
Random Forest	0.8477
Support Vector Machine (SVM)	0.8181

TABLE 7 – Accuracy des classifieurs alternatifs

En comparant les performances de LogisticRegression une meilleure performance est souvent observée avec la vectorisation tf-idf. De plus, l'équilibrage des données a conduit à une amélioration notable des performances,. Il convient également de noter que l'utilisation de n-grammes a également contribué à des résultats plus fiables.

5 Conclusion

En conclusion, cette analyse a démontré l'importance du choix des techniques de prétraitement, de vectorisation et de modèle de classification pour obtenir des performances optimales dans la classification de texte pour la detection de fake news. Le pipeline optimal que nous avons trouvé inclut l'utilisation de TF-IDF avec l'application de bigrams, l'équilibrage des données avec SMOTE, et l'utilisation d'un modèle de régression logistique optimisé. Cette approche a permis d'obtenir les meilleures performances globales. Il est essentiel de continuer à explorer et tester différentes configurations pour s'assurer que les solutions choisies sont les plus adaptées aux données et aux tâches spécifiques.