Université de Montpellier
Master 1 Informatique

Projet de machine learning

Classification d'assertions selon leur valeur de véracité

Rapport du projet

Etudiants :
Desbos Marie-Lou
Garcia Léa
Romero Gaétan
Sanchez Martin
Trinquart Matthieu

Table des matières

1 Introduction

Aujourd'hui les fausses informations se répandent de plus en plus vite sur internet. La crise du covid a accentué ce problème avec des fausses informations qui se propagent très vite parfois plus vite que de vraies informations sourcées. Les fausses informations peuvent avoir des conséquences très graves dans une société en créant de la désinformation pouvant inciter la population à faire des choix contre-productifs.

Les réseaux sociaux sont un terrain fertile pour la propagation des fausses informations (Twitter, Facebook, Youtube, etc..). Il est donc important que ces grands groupes prennent conscience du danger de ces fausses informations et appliquent des mesures pour ralentir le plus possible leur propagation. Le plus gros problème est qu'aujourd'hui beaucoup de monde s'exprime sur les réseaux sociaux. Rien que sur Twitter, il y a environ 504 millions de tweets chaque jour ce qui rend impossible la vérification manuelle de chaque information. Il est donc impératif de rendre ces vérifications automatiques et donc pouvoir limiter la propagation de ces informations.

La difficulté de cette automatisation est qu'il est difficile de juger ce qui est une fausse information ou pas. Par exemple la phrase "Trump est un mauvais président" est une phrase subjective et ne peut être catégorisée comme une vraie ou une fausse information. Il est aussi important que cette automatisation soit juste, afin d'éviter que le modèle laisse se propager de fausses informations ou inversement qu'elle bloque de vraies informations au public.

L'objectif du projet est donc de créer un modèle qui puisse distinguer une bonne information d'une fausse information. Notre projet est d'actualité et donc pourrait avoir une importance dans la lutte contre la propagation des fausses informations. Pour justement éviter les conséquences d'une mauvaise automatisation il est important d'avoir un bon jeu de données à donner à nos classifieurs afin d'avoir un résultat le plus juste possible.

2 Pré-traitement des données

Au cours de l'introduction précédente, nous avons parlé de l'objectif de notre sujet qui consiste à classer des assertions selon leur véracité. C'est dans ce contexte que nous avons commencé par effectuer le pré-traitement des données. Cette étape est obligatoire dans le sens où elle va nous permettre d'obtenir, à la fin, des classifieurs les plus performants possible, en vérifiant la qualité des données. Pour réaliser cela, nous devons réaliser les étapes suivantes :

— Formatage des données
— Normalisation

2.1 Formatage des données

Nos données sont sous la forme d'un fichier csv constitué de colonnes chacune symbolisant un attribut de celles-ci. On a, par exemple, une colonne text pour le nom de nos données mais aussi ratingName pour savoir si la donnée est vrai ou fausse. C'est pourquoi, nous avons défini des variables d'apprentissage et de prédiction dans notre formatage des données. Ces variables d'apprentissage vont chacune récupérer les informations utiles de notre jeu de données puis nous les concaténons pour avoir une seule variable à utiliser dans l'étape suivante. Pour la variable de prédiction, elle récupère la valeur de la colonne ratingName.

2.2 Normalisation

Tokenisation

Lors de cette deuxième étape du pré-traitement, nous allons nous intéresser aux tokens de notre document. Pour cela, nous allons découper nos phrases pour récupérer chaque mot différent, pour par la suite, pouvoir les traiter séparément. Cette étape s'appelle la tokenisation. Elle est indispensable pour pouvoir faire le pré-traitement des données. Pour réaliser cette étape, nous utilisons la fonction *word_ tokenize()* qui va découper nos données (donc nos phrases) en token.

Suppression de ponctuation

Parmi les tokens créés, certains contiennent de la ponctuation. Or, elle ne nous intéresse absolument pas, étant donné qu'elle ne permet pas de distinguer une fausse information d'une vraie. Nous avons choisi d'utiliser la fonction *str.maketrans(", ", string.punctuation)*. Elle nous permet de récupérer les signes de ponctuation, puis de les appliquer aux tokens pour les enlever par la suite.

Stop-words

Une autre étape de la normalisation est la suppression des stop-words. Pour cela, on utilise le dictionnaire NLTK et on retire les tokens récupérés qui sont contenus dans la liste des mots stop-words

Lemmatisation et Racinisation

Les deux dernières étapes de normalisation sont la lemmatisation et la racinisation. Pour la première, nous avons utilisé la fonction *WordNetLemmatizer()*, qui permet de relier les mots qui ont des significations similaires, à un seul autre mot, tout en tenant compte du contexte. Pour la deuxième, nous avons utilisé *PorterStemmer()*, qui permet de transformer les mots en leur racine.

3 Classifieurs utilisés et Hyper-Paramètres

3.1 Les classifieurs utilisés

3.1.1 Multinomial Naive Bayes

Hyper-Paramètres

Listing 3.1 – Hyper-Paramètres de Multinomial Naive Bayes

```
'MultinomialNB' : [{
    'alpha': np.linspace(0.5, 1.5, 6),
    'fit_prior': [True, False]}]
```

3.1.2 Logistic Regression

Hyper-Paramètres

Listing 3.2 – Hyper-Paramètres de Logistic Regression

```
'LR' : [{
    'C' : [0.001,0.01,0.1,1,10,100]}],
```

3.1.3 KNeighbors Classifier

Hyper-Paramètres

Listing 3.3 – Hyper-Paramètres de KNeighbors Classifier

```
'KNN' : [{
    'n_neighbors' : [5,7,9,11,13,15],
    'weights' : ['uniform','distance'],
    'metric' : ['minkowski','euclidean','manhattan']}],
```

3.1.4 DecisionTree Classifier

Hyper-Paramètres

Listing 3.4 – Hyper-Paramètres de DecisionTree Classifier

```
'CART' : [{
    'max_depth': [2, 3, 5, 10, 21],
    'min_samples_leaf': [5, 10,16 ,20, 50, 100],
    'criterion': ["gini", "entropy"]}],
```

3.1.5 RandomForest Classifier

Hyper-Paramètres

Listing 3.5 – Hyper-Paramètres de RandomForest Classifier

```
'RF':[{
    'n_estimators': [4, 6, 9],
    'max_features': ['log2', 'sqrt','auto'],
    'criterion': ['entropy', 'gini'],
    'max_depth': [2, 3, 5, 10],
    'min_samples_split': [2, 3, 5],
    'min_samples_leaf': [1,5,8]
    }],
```

3.1.6 Support Vector Classification

Hyper-Paramètres

Listing 3.6 – Hyper-Paramètres de Support Vector Classification

```
'SVM' : [{
    'C': [0.001, 0.01, 0.1, 1, 10],
    'gamma' : [0.001, 0.01, 0.1, 1],
    'kernel': ['linear','rbf']}],
```

4 Analyse de nos résultats

4.1 Résultats en fonction de la taille des données en entrée

Afin de tester l'influence de la taille des données en entrée des classifieurs, nous avons effectué plusieurs tests, avec tous les classifieurs et leur hyper-paramètres par défaut, en changeant le nombre de tuples dans les fichiers csv.

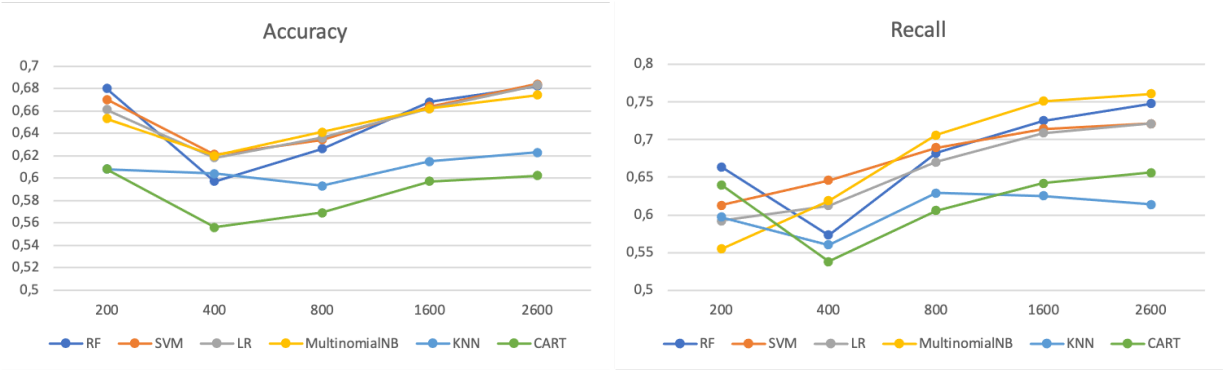


FIGURE 4.1 – Accuracy et Recall des classifieurs

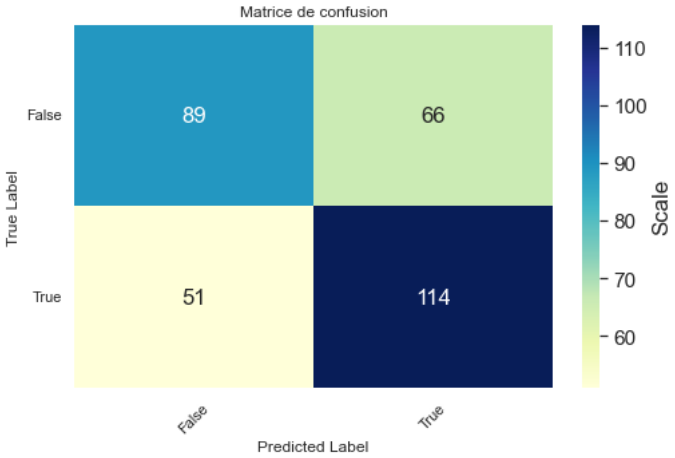
Après avoir effectué ces tests, nous pouvons en conclure que les classifieurs ont des résultats assez homogènes, cependant la taille des entrées ne fait pas varier l'accuracy de manière significative. Concernant le recall, on voit que sa valeur augmente avec la taille des données. On prendra donc des données plutôt grandes (1000 tuples au minimum) pour effectuer nos tests pour chacune des trois tâches de classifications.

4.2 Résultats pour TRUE vs FALSE

Pour ces données, le meilleur classifieur est MultinomialNB.

Il atteint une accuracy de 0.6234375
Avec les paramètres suivants :

- 'alpha' : 1.1.
- 'fit_prio' : False.

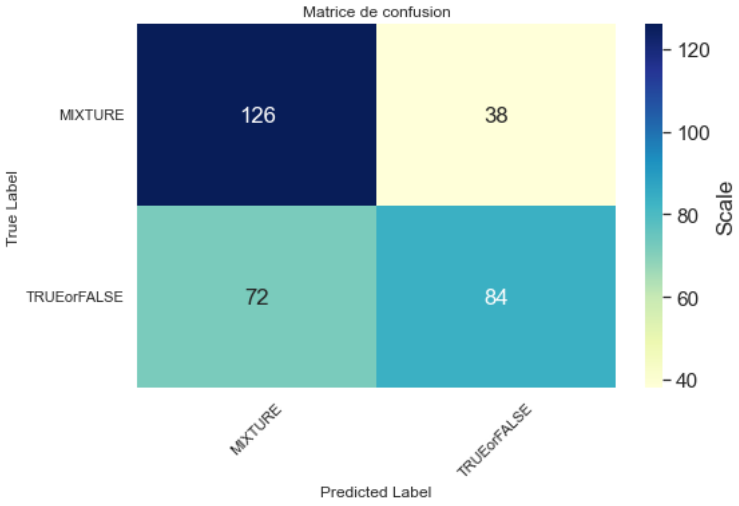


4.3 Résultats pour TRUE ou FALSE vs MIXTURE

Pour ces données, le meilleur classifieur est RandomForest.

Il atteint une accuracy de 0.65625.
Avec les paramètres suivants :

- 'criterion' : 'entropy'.
- 'max_depth' : 10.
- 'max_features' : 'sqrt'.
- 'min_samples_leaf' : 1.
- 'min_samples_split' : 2
- 'n_estimators' : 9

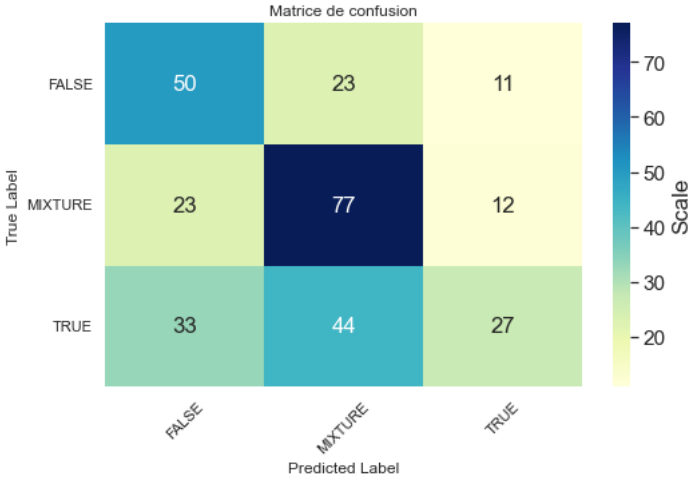


4.4 Résultats pour TRUE vs FALSE vs MIXTURE

Pour ces données, le meilleur classifieur est RandomForest.

Il atteint une accuracy de 0.6268292682926829.
Avec les paramètres suivants :

- 'criterion' : 'entropy'.
- 'max_depth' : 10.
- 'max_features' : 'auto'.
- 'min_samples_leaf' : 1.
- 'min_samples_split' : 3
- 'n_estimators' : 9



5 Conclusion du projet

Pour conclure, il est très difficile de détecter si des assertions sont vraies ou fausses avec des modèles de classification. Ces modèles doivent pouvoir comprendre le lien entre les mots d'une phrase, utiliser des données telles que la source, l'auteur, headline, les mots-clés, le texte, afin de déterminer la véracité ou non d'une assertion... Nos résultats ne dépassent pas les 0,65 malgré l'utilisation de plusieurs classifieurs différents qui peuvent s'adapter au mieux à nos données. Pour pouvoir obtenir de meilleurs résultats, il faudrait d'abord améliorer les pré-traitements de nos données, peut-être augmenter ou diminuer le nombre de variables associées à l'assertion. On pourrait également effectuer des recherches d'hyper-paramètres plus poussées en augmentant les possibilités pour chacun et éventuellement tester avec d'autres classifieurs qui pourraient correspondre encore mieux à nos données.