UNIVERSITÉ
DE MONTPELLIER

Classifications de documents de fact-checking

Groupe H :
Nguyen Hoai Nam - 21512583
Nguyen Huu Khang - 21506865
Nguyen Tran Tuan Nam - 21914400
Tran Thi Tra My - 21511002

15 mai 2020

Nous souhaitons remercier nos enseignants qui nous ont encadrés et guidés tout au long de l'élaboration de ce projet. Tous les conseils qu'ils nous ont données sont d'une grande grande importante et impactent positivement nos connaissances et compétences.

Table des matières

Chapitre 1

Introduction

1.1 Présentation

Ce projet a pour but de proposer des modèles de classification supervisée d'assertions faites par des figures politiques selon leur valeur de véracité, ou autrement dit, de proposer une approche de fact-checking automatique. Nous avons à disposition un jeu de données d'environ 40 000 lignes de données et 23 colonnes. Chaque colonne contient un label différemment. Les données sont stockées dans un fichier CSV.

Nous devons alors effectuer de divers pré-traitements et de tester différents modèles de classifications afin de trouver le meilleur d'entre eux.

1.2 Méthode

Pour déterminer votre modèle de prédiction, nous avons procédé de cette façon :

— Déterminer et effectuer les pré-traitements nécessaires sur les données permettant leur exploitation optimale.
— Lancer plusieurs classifieurs avec les paramètres par défaut en utilisant la **kFold Cross Validation**. Cette méthode consiste à diviser les données en plusieurs échantillons, dont un sert à la validation et les autres sert à l'apprentissage. Chaque échantillon est utilisé pour la validation tour à tour, permettant ainsi notamment d'éviter le sur-apprentissage.
— Prendre les meilleurs et trouver les hypers paramètres optimisant la précision.
— Sauvegarder le modèle et tester sur de nouvelles données.

1.3 Outils et langages

Nous avons utilisé la bibliothèque **NLTK** et ses différents modules pour effectuer les pré-traitements. Afin de visualiser les données et lire le fichier CSV, nous avons aussi utilisé la bibliothèque **Pandas**. Enfin, nous avons utilisé **Scikit-learn**. C'est une bibliothèque pour la machine learning en Python que nous avons l'occasion d'apprendre à utiliser dans les TP avec les notebooks afin de faciliter le partage du travail et l'ergonomie du code.

Chapitre 2

Création du modèle

2.1 Pré-traitements et choix

Nous avons mis en place plusieurs prés-traitements permettant de polir les textes par exemple la suppression des contractions, la mise en minuscule des mots, etc. De plus, nous avons aussi effectuait des choix en supprimant plusieurs colonnes qui nous semblent inutiles où manquent beaucoup de données pour les classifications.

2.1.1 Pré-traitement des données textuelles

Concernant les colonnes qui ne contiennent que des données textuelles, nous avons effectué les pré-traitements comme suivant :

Tokenisation

- Pour commencer, nous avons tout d'abord supprimé les contractions dans le texte car en anglais, on utilise beaucoup de formes composées. Il est donc important que l'on remplace ces contractions afin de pouvoir travailler sur un texte propre. Nous retrouverons des exemples typiques dans le texte comme "wasn't" qui vient de deux mots "was not". Cette étape permet de remplacer la forme composée en deux mots complets.
- Nous avons supprimé les URLs références vers les photos ou les profils car ils nous semblent inutiles pour l'analyse textuelle.
- Les caractères non-ASCII sont aussi supprimés car ils pourraient affecter le résultat de la classification.
- Nous avons décidé de mettre en minuscule tous les caractères afin d'éviter l'augmentation du nombre de mots différents à l'analyse textuelle. Du même raisonnement, nous avons aussi supprimé les ponctuations et remplacé les chiffres numériques en lettres. Nous avons constaté que le temps d'exécution a baissé et la précision a diminué légèrement.

Stop words

- Nous avons utilisé la liste de Stopwords fourni par la bibliothèque **NLTK** avec l'option Anglaise qui correspond au langage de nos données textuelles. Cette bibliothèque permet d'enlever les mots qui seront ignorés pendant l'exécution de l'analyse textuelle car ils n'ont pas de signification propre. Cela nous économisera du temps d'exécution sans perdre la précision sur le classifieur.

— De plus, afin d'avoir le meilleur résultat, nous avons aussi ajouté des mots qui nous semblaient inutiles à l'analyse et supprimé certains mots de la liste de Stopwords

Lemmatisation et POS-tagging

Nous avons utilisé lemmatisation afin de ramener tous les mots vers un mot de base commune. Cela permet d'éviter d'avoir des mots similaires qui seront inutiles pour l'analyse textuelle comme par exemple "blessing" et "bless". En complément de la lemmatisation, nous avons aussi appliqué la notion POS-tagging afin de trouver la nature de chaque mot pour application de lemmatisation.

2.1.2 Pré-traitement des méta-données

Concernant les colonnes qui contiennent les méta-data, nous avons effectué les pré-traitements comme suivant :

Suppression des colonnes

Nous avons effectué la suppression sur les colonnes qui contiennent la majorité des valeurs nulles. Voici ces colonnes claimReview_author, creativeWork_author_sameAs.

Ensuite, nous avons effectué la suppression sur les autres colonnes pour plusieurs raisons :

— Concernant les colonnes claimReview_author_name, claimReview_source celles-ci ne donne pas beaucoup d'informations mais en revanche, il provoque une diminution de précision fortement.
— Suivi des colonnes qui diminuent la précision, ce sont les colonnes qui ont rendu la classification très lourde. Le temps d'exécution et la ressource nécessaire ont été très importants. Nous n'avons pas pu réussir à l'obtenir la précision avec ces colonnes. Voici la liste des colonnes concernées claimReview_author_url, extra_entities_author, extra_entities_body, extra_entities_claimReview_claimReviewed, extra_entities_keywords, extra_refered_links.

Pré-traitement sur les valeurs manquantes

Concernant les colonnes qui contiennent beaucoup de valeurs manquantes mais le reste de ses valeurs qui ont l'air important. Nous avons procédé de les remplir par différentes façons.

— Tout d'abord, concernant la colonne "extra_body" qui contient les données textuelles, nous avons décidé de les enlever car il nous semble que la suppression de ces deux lignes ne pose pas un problème. De plus, nous n'avons pas pu estimer la valeur pour cette ligne-ci.
— Ensuite, nous avons choisi de remplir les valeurs nulles des colonnes suivants "creativeWork_author_name", "extra_tags", "claimReview_datePublished", "creativeWorkdatePublished" par une chaîne caractère "Inconnu". Nous n'avons pas pu trouver un moyen pour remplir ces valeurs correctementes.
— Enfin, concernant les colonnes "rating_bestRating", "rating_worstRating", "rating_ratingValue", nous avons pris la moyenne des valeurs disponibles pour remplir les lignes contenant les valeurs nulles.

2.1.3 Sélection de variables

Nous avons choisi de classer la colonne "rating_alternateName" en deux groupe. Nous avons changé tous les valeurs "true" et "almost true" en 1 qui correspond à "vrai" et le reste en 0 qui correspond à "faux". Cela nous permettra d'avoir une colonne qui contient que les valeurs binaires. Ensuite, Nous avons traité cette même colonne en convertissant les valeurs "true" et "false" en 1 et le reste en 0. Nous avons obtenu un résultat très déséquilibre soit 96 valeurs 0 qui correspond aux valeurs "mixture".

2.1.4 Downsampling

Lors de la classification, nous avons trouvé que les données d'apprentissages ont été déséquilibrées soit d'environ 4500 valeurs fausses et d'environ 32000 vraies. Nous avons donc procédé à l'utilisation de la notion **Downsampling**. Elle consiste à équilibrer les données afin d'avoir un résultat plus réel. En effet, sur nos données, nous avons eu beaucoup plus de résultats négatifs que de positifs. Cela a probablement rendu notre résultat moins réel.

2.2 Choix du classifieur

Après avoir effectué les pré-traitements sur notre jeu de données, nous avons utilisé l'outil **kFold Cross Validation** sur plusieurs classifieurs différents afin de pouvoir décider lesquelles seront utiles pour notre analyse. Notre choix étaient basés sur la précision et l'écart-type. Les classifieurs utilisés pour la classification étaient RandomForestClassifier, LogisticRegression, KNeighborsClassifier, DecisionTreeClassifier, GaussianNB. Nous les avons choisi car ce sont les classifieurs que nous avons l'occasion d'apprendre pendant les séances de TP ou de cours. Il est donc plus simple pour nous de les appliquer et les améliorer leurs performances. En complément de ça, nous avons utilisé aussi les diagrammes pour visualiser le résultat obtenu. Ensuite, nous avons utilisé le **GridSearchCV** afin de trouver les classifieurs qui ont les meilleures notes. Cet outil nous permettra d'avoir un choix sur les classifieurs les plus efficaces. Après les analyses, nous avons décidé de garder "RandomForestClassifier" et "DecisionTreeClassifier". Ce sont les deux meilleurs classifieurs en terme de la précision et du temps.

2.3 Sauvegarde du modèle

Pour sauvegarder, nous avons utilisé la module Pipeline. Celle-ci nous permettra de sauvegarder un modèle afin de pouvoir faire des prédictions sur des données différentes.

Chapitre 3

Résultats

Nous avons pu obtenu un résultat de 100% de précision utilisant RandomForestClassifier et DecisionTreeClassifier utilisant le GridSearchCV et kFold mais avec le Pipeline, nous n'avons pas eu le même résultat. La précision a été respectivement 79% et 76% pour les 2 méthodes nommés précédent sur les données appliquant "downsampling" et 93% ou 90% sur les données initiales pour les 2 méthodes précédemment. L'écart type des différences classifieurs a été minimal. Nous n'avons pas pu voir la différence.

3.1 Analyse

Nous avons vu dans le paragraphe précédemment qu'il y a une différence de 14% avant et après l'utilisation de "downsampling". Cela vient probablement de la dés-équilibration du nombre de valeurs binaires. En utilisant les valeurs initiales, nous avons obtenu une précision de 100% alors qu'en utilisant les valeurs apprentissages, nous n'avons eu que 76%. La différence peut venir à cause de l'initialisation de pipeline après l'entraînement des données pré traitées. Nous avons dû réaliser l'opération avant le pré-traitement.

En ce qui concerne la différence entre la méthode kFold et la méthode Pipeline, nous avons pensé aux paramètres pour chaque méthode. Avec Pipeline, nous avons appliqué le PCA et le "standard scaling". Ce n'était pas le cas pour kFold ou GridSearchCV.

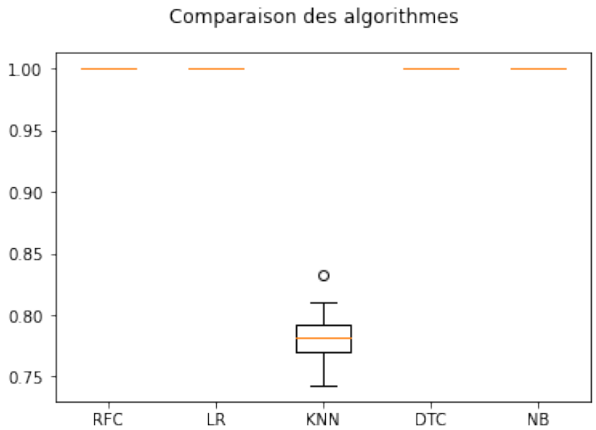

FIGURE 3.1 – Comparaison des classifieurs utilisant kFold

Chapitre 4

Conclusion

Nous n'avions pas pu tester sur tous les hyper-paramètres dus au temps manquant. Nous avons rencontré un problème concernant la quantité de mémoire utilisable par jupyter-notebook. Cela nous a pris 1 mois pour résoudre car il n'y avait pas des solutions sur internet. En effet, cela est due à l'installation de Python 32 bits qui limitera la quantité de mémoire de Python à inférieur de 1 Gb de mémoire. Nous aurions pu tester sur les autres hyper-paramètres afin de voir la différence.

De plus, nous aurions pu trouver un moyen le plus pratique afin de remplacer les valeurs manquantes à la place de les remplacer par la valeur "Inconnu" ce qui peut provoquer à la précision de résultat. Le résultat peut être affecté à cause des valeurs remplaçantes.

En conclusion , nous avons pu apprendre un nouveau domaine et les nouveaux outils grâce à ce projet ainsi d'acquérir des nouvelles compétences.