HMIN232
Méthode de la science des données

Fact-Checking

Auteur :

Canta Thomas
Desgeneztez Charles
Fontaine Quentin
Reiter Maxime

Master 1 - AIGLE
Faculté des sciences de Montpellier
Année universitaire 2020/2021

Table des matières

Chapitre 1

Introduction

Ce projet a pour but de proposer des modèles de classification supervisée d'assertions faites par des figures politique selon leurs véracités. Autrement dit, nous devons proposer une approche automatique de fact-checking. Nous avons choisi de faire notre propre extraction en téléchargeant séparément les différentes évaluations nécessaires au projet, à savoir True, False et mixture, à l'aide de l'interface data.gesis.org.

Au cours du projet, la principale problématique lors de l'établissement des modèles était le déséquilibre entre les évaluations. Cela menait à des résultats de classification satisfaisants, mais inspecté de plus près cela invalide notre précision. Une fois le rééquilibrage effectué, à l'aide de différentes méthodes comme le suréchantillonnage (*Upsampling*) et/ou le sous-échantillonnage (*Downsampling*, nous avons pu nous consacrer au choix du meilleur classifieur à adopter pour la suite de l'étude.

Pour ce faire nous avons utilisé une GridSearchCV, permettant de tester plusieurs classifieurs avec différents paramètres et de récupérer le plus précis par rapport à un certain score, ici la précision. Une fois choisis-nous pouvions encore optimiser notre modèle en sélectionnant les meilleurs features pour la prédiction.

Chapitre 2

Pré-traitement des données

Le prétraitement des données est une étape importante dans l'analyse de texte et le traitement du langage naturel. Il permet de fournir à nos algorithmes d'apprentissage un texte simplifié et analysable, amplifiant ainsi leur performance. Nous n'allons pas nous attarder sur le sujet car c'est assez trivial à la compréhension.

- Radicalisation (ou Stemming) → Réduit les mots en leur radical ou un dérivé.
- Gestion du texte non ASCII → Normalisation du texte en supprimant tous les caractères non ASCII.
- Mise en minuscule (Lowercase) → Conversion du texte en minuscule.
- Retrait des caractères uniques → On supprime les mots de taille 1.
- Retrait de la ponctuation.
- Retrait des nombres.
- Retrait des mots d'arrêts → Retrait des mots courant tel que "the", "a", "an", "in"

NB : Au lieu de supprimer les nombres nous aurions pu les remplacer (1000 → mille) mais nous avons considéré les données inutiles et potentiellement redondantes pour notre modèle.

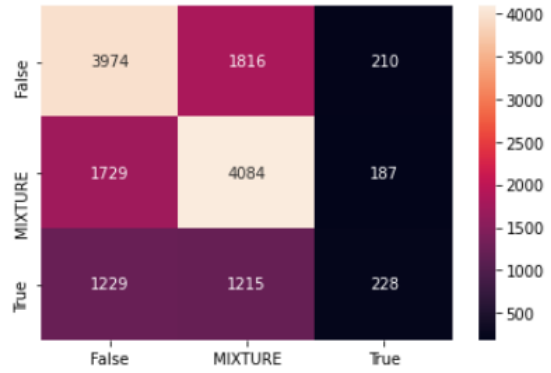
Chapitre 3

Classification

3.1 Un premier classifieur

Une fois les données pré-traitées nous pouvons utiliser des classifieurs. Pour se faire nous utiliserons la technique du sac de mot qui consiste à transformer les différentes occurrences d'un mots en un entier puis de les pondérées sur leur fréquences à l'aide de TF-IDF. De ce fait nous avons pu faire un test avec un classifieur ici LogisticRegression (LR).

```
[+] Accuracy: 56.475%

[~] Matrice de confusion:
```

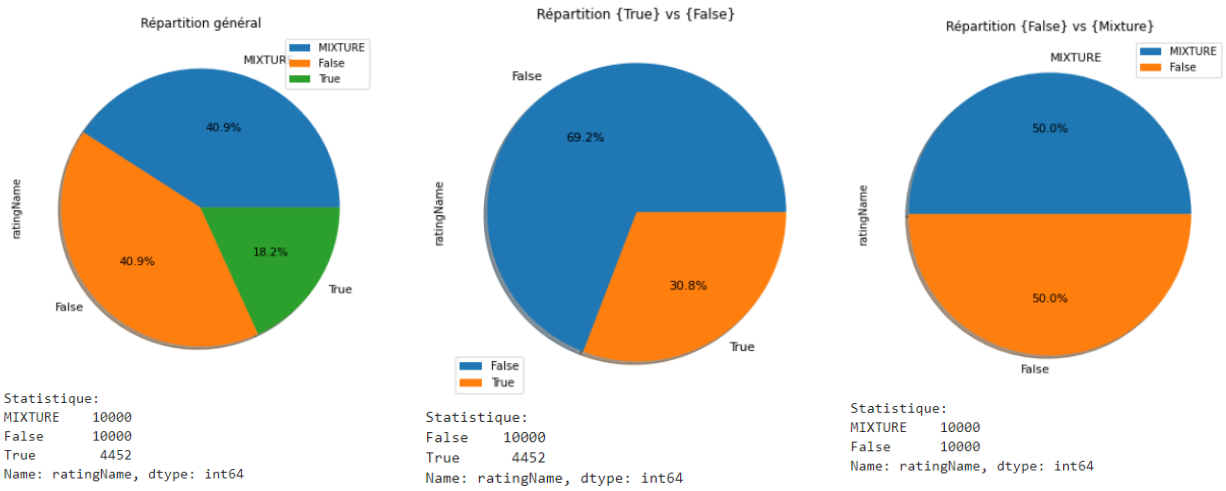



```
[~] Rapport de classification:
              precision    recall  f1-score   support

       False       0.57      0.66      0.61      6000
     MIXTURE       0.57      0.68      0.62      6000
        True       0.36      0.09      0.14      2672

    accuracy                           0.56     14672
   macro avg       0.50      0.48      0.46     14672
weighted avg       0.54      0.56      0.53     14672
```

Ce premier test a soulevé un problème critique visible via la matrice de confusion. En effet, elle laisse paraître un problème sur notre classe *True* avec 228 True positif contre 4084 Mixture positives et 3974 False positifs. La cause est facilement identifiable via le *support* du rapport de classification, celui-ci affiche utiliser presque 3 fois moins d'informations que les deux autres (2672 contre 6000). Ceci est le signe d'un déséquilibre dans notre modèle.

3.2 Equilibrage des données

Comme nous l'avons vu nos données sont biaisées, nous avons utilisé deux méthodes très connues pour ce type de problème, le sur et sous-échantillonnage, mieux connus sous le nom d'Up et Down Sampling. Tout d'abord visualisons la répartition des données via les différentes tâches de classifications demandées pour ce projet :



NB : Nous avons choisis False vs Mixture, nous expliquons notre choix plus tard dans le rapport.

Comme nous l'avions soupçonnée, notre jeu de données et déséquilibrée de par notre classe *True* qui représente seulement 18.2% de données dans tout notre modèle. En revanche les classes *Mixture* et *False* possèdent le même nombre de données, nous n'aurons donc qu'une classe à modifier en conséquences. Pour ce faire nous avons essayé deux méthodes très connues pour pallier ce problème, le sous et sur-échantillonage, respectivement Down et Up sampling.

3.2.1 sous-échantillonage (Down slamping)

Ce processus a le désavantage de faire perdre potentiellement un nombre considérable de données suivant la quantité de données à retirer. Prenons {True} vs {False} par exemple, plus de la moitié des informations à False vont être retirés en passant de 10.000 à 4452 données.

3.2.2 sur-échantillonage (up slamping)

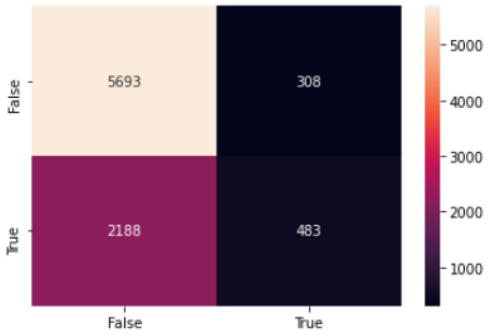
À l'inverse de la perte de données, les désavantages de cette méthode sont les données aléatoires créent pendant l'opération. Celles-ci peuvent fausser notre prédiction par l'apparition de nouvelles features qui ajouterait du bruit dans notre modèle.

Maintenant que nous connaissons les désavantages de chaque classe on comprend l'avantage d'avoir un modèle à donner égale. C'est pourquoi nous avons choisi {False} vs {Mixture}, car ils possèdent le même nombre de données.

Pour faciliter la poursuite de l'étude nous continuerons à traiter en premier la classification {True} vs {False}.

[+] Accuracy: 71.218%

[~] Matrice de confusion:



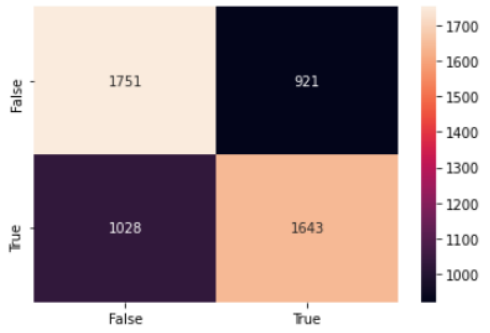
[~] Rapport de classification:

	precision	recall	f1-score	support
False	0.72	0.95	0.82	6001
True	0.61	0.18	0.28	2671
accuracy			0.71	8672
macro avg	0.67	0.56	0.55	8672
weighted avg	0.69	0.71	0.65	8672

FIGURE 3.1 – Précision obtenus sans échantillonnage

[+] Accuracy: 63.522%

[~] Matrice de confusion:



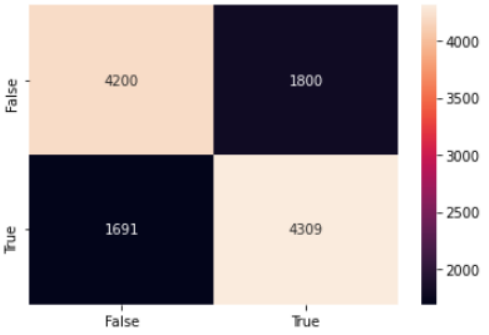
[~] Rapport de classification:

	precision	recall	f1-score	support
False	0.63	0.66	0.64	2672
True	0.64	0.62	0.63	2671
accuracy			0.64	5343
macro avg	0.64	0.64	0.64	5343
weighted avg	0.64	0.64	0.64	5343

FIGURE 3.2 – Précision obtenus avec sous-échantillonnage (DSamp)

[+] Accuracy: 70.908%

[~] Matrice de confusion:



[~] Rapport de classification:

	precision	recall	f1-score	support
False	0.71	0.70	0.71	6000
True	0.71	0.72	0.71	6000
accuracy			0.71	12000
macro avg	0.71	0.71	0.71	12000
weighted avg	0.71	0.71	0.71	12000

FIGURE 3.3 – Précision obtenus avec sur-échantillonnage (USamp)

La figure 3.1 est le résultat du classifieur sans échantillonage, la figure 3.2 celui avec sous-échantillonage (DSamp) et la dernière figure 3.3 avec sur-échantillonage (USamp).

Bien que USamp promet une précision plus élevée que DSamp elle reste très légèrement plus faible que notre précision de départ. Néanmoins le rapport de classification démontre bien que cette méthode est la plus performante. Ses valeurs sont aussi stable que DSamp avec des valeurs supérieurs. L'Upsampling sera donc la méthode adopter pour la suite de l'étude.

Chapitre 4

Optimisation

4.1 Sélection de features

Nous avons décidé d'optimiser notre sac de mot en gérant les groupes de mots (*Gram*). Nous n'avons pas eu le temps de créer un graphe démontrant l'importance de ceux-ci par rapport à la portée (*range*) utilisée, mais par nous-mêmes nous avons pu n'identifier que le meilleur choix était entre 1 et 3 mots. De plus nous avons ajouté un attribut permettant d'ignorer les termes dont la fréquence de document est strictement inférieure au seuil donné, ici nous avons choisi $1e^{-2}\%$. En d'autre termes sur 10.000 features si son occurrence est inférieure à 10 elle est ignoré.

Nous pouvons ensuite passer un coup de balais en supprimant les autres features constantes à l'aide de VarianceThreshold. Nous supprimons toutes les features dont la variance est inférieure à $2e^{-3}$. Nous voilà avec $\pm$1300 features au lieu des $\pm$24000 de départ.

Choisissons les meilleures features parmi celles restantes. Nous utilisons SelectFromModel qui permet de sélectionner les meilleurs features en fonction de leur poids d'importance à partir d'un modèle donné. De plus nous pouvons fournir un seuil permettant de conserver les features dont l'importance est supérieure ou égale sont conservées, de manière empirique nous avons utilisé un seuil de $5e^{-2}$.

NB : Nous avions essayé plusieurs sélecteurs, comme RFE, SequentialFeatureSelector (Forward et Backward Selection) ou encore SelectKBest mais ces méthodes étaient trop longues ou n'amélioraient pas notre modèle.

Avec ceci nous obtenons une précision de $\approx$**67.2%**, toujours avec le classifiait LR, nous avons donc perdu de la précision. Nous nous sommes demandé si cela pourrait provenir du classifieur qui ne serait pas adapté. Nous avons donc essayé avec de le classifier Random Forest (RF) et nous obtenons $\approx$**74.4%**, ce qui est une bonne nouvelle car cela signifie que notre sélection de features est performante mais qu'elle était inadaptée à notre premier classifieur.

4.2 Ajout de données additionnelles

Jusqu'à présent seul les textes principals de nos articles étaient traités, pour améliorer notre précision nous pouvons ajouter de nouvelles données mais il faut les choisir avec soin. En effet, de nouvelles données inutiles ajouteraient du bruit à l'estimation, de même si l'on ajoute des informations redondantes. Nous pouvons ainsi économiser du temps et/ou de l'argent en ne les mesurant pas. Listons les différentes données disponibles (cf : tableau page 8 Tableau 4.1)

Premier constat que nous avons pu omettre c'est que certaines données sont incluses dans d'autres, par exemple ($author \subset named\_entities\_claim$). Donc il n'est pas nécessaire d'ajouter les deux, seuls le meilleur des deux sera ajoutés.

Pour déterminer leurs importances dans l'amélioration de notre précision nous avons fait un benchmark en ajoutant une unique colonne en plus de notre texte et nous avons obtenu les résultats suivants (cf : Schéma page 8 Figure 4.1)

nom	analyse	conservé
date	Composée uniquement de chiffre elle n'apportera aucune information et ne sera que source de bruit	✘
author named_entities_claim named_entities_article	Peuvent regorger de données absentes dans notre texte, elles peuvent donc être utiles.	✔
headline keywords	Peut regorger de données absentes dans notre texte, elles peuvent donc être utiles.	✔
sourceURL link	Les liens relatifs et absolus sont sources de bruits et sont supprimés lors du prétraitement, les ajouter n'est donc pas correcte	✘
source	Retient uniquement la partie intéressante de l'URL (https ://politifact/ → politifact) ceci est donc intéressant.	✔
language	Inutile car est une unique valeur, ici "English". Les données trop redondantes sont inutiles pour améliorer les prédictions.	✘

TABLE 4.1 – Analyse des données additionnels

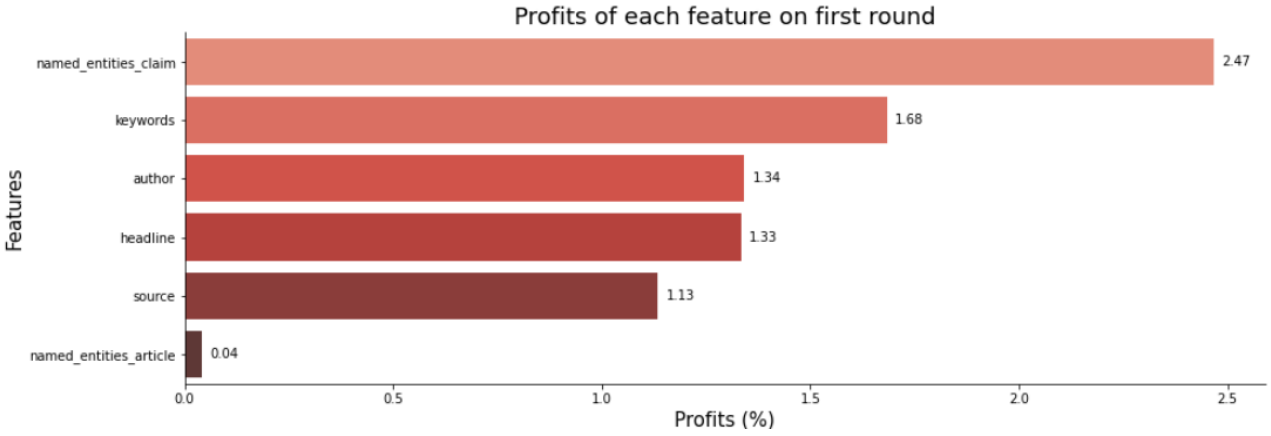


FIGURE 4.1 – Profits par unique donnée ajoutée

Le graphe ci-dessous montre, par exemple, que l'ajout de named_entities_claim ($texte + named\_entities\_claim$) augmenterait notre précision de 2.47%, ou encore l'ajout de named_entities_article ($texte + named\_entities\_article$) l'augmenterai de 0.04%. Toutes les données sembles utiles à ajouter. Sauf named_entities_article qui est très faible et author car on le rappelle celui-ci appartient déjà à named_entities_claim

Après avoir ajouté nos nouvelles données et appliquer notre classifieur RF nous obtenons $\approx$**80.4%** de précision, soit $\approx$ 5% de plus que notre score précédent. En revanche le temps de fit est 3 fois plus long ($\approx 10 secondes$), cela est dû aux nouvelles features traitées, créent par les données ajoutées.

La seule chose dont nous n'avons pas encore optimisée est notre classifier, en testant de multiples paramètres ou même en essayer d'autre, nous pourrions trouver de meilleures performances, en matière de vitesse et de précision. Pour ce faire nous utiliserons une GridSearchCV.

4.3 GridSearch

Nous avons mené nos tests avec 4 classifieurs, RandomForestClassifier (RF), LogisticRegression (LR), KNeighborsClassifier (KN) et SGDClassifier (SGD). Les deux plus performants furent RF et KN avec respectivement **80.43%** et **79.5%** de précision.

4.4 Evolution

Voici une évolution de notre précision part les différentes étapes maître de notre étude. Nous aurons réussi à améliorer notre précision $\approx 9\%$ par différentes optimisations et améliorations, néanmoins ce ne sont pas les meilleurs car certains paramètres comme les ngram, ou encore la sélection de feature, on était choisis de manière empirique par nous-mêmes. Il conviendrait de faire de multiples tests pour optimiser notre modèle.

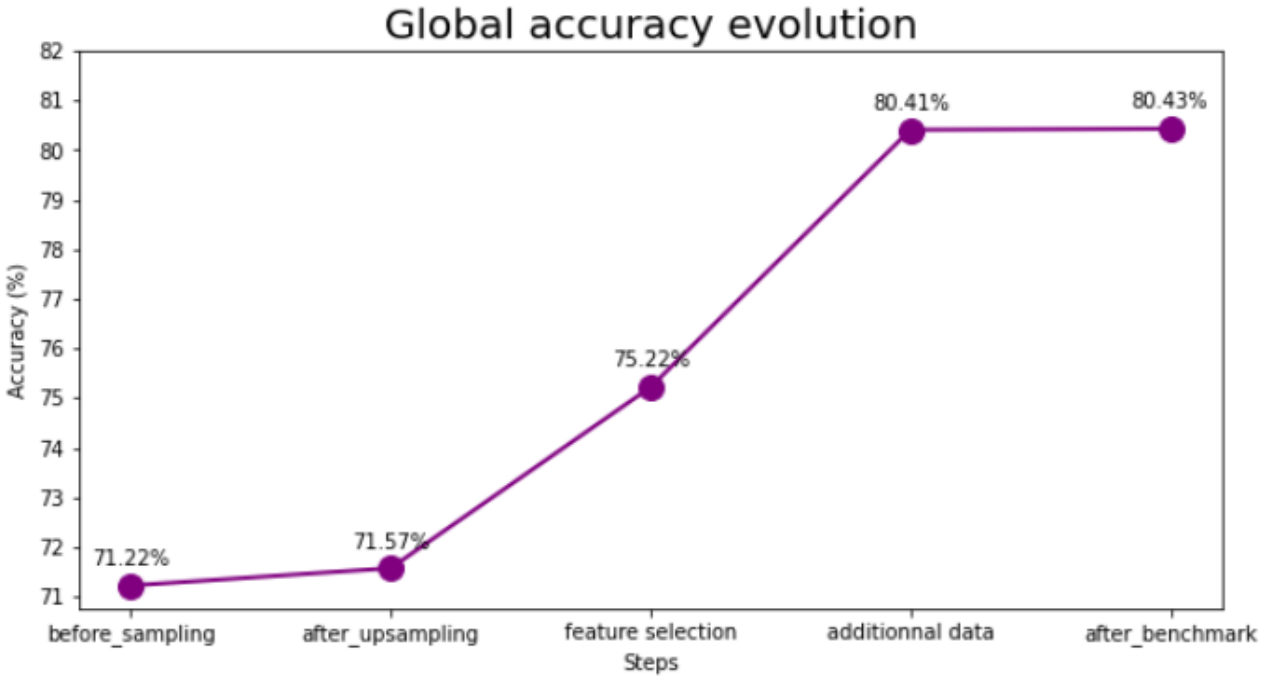


FIGURE 4.2 – Evolution de notre précision depuis le début

4.5 Test des autres classifications

N'oublions pas que nous devions aussi tester ({False} vs {Mixture}) et ({True} vs {False} vs {Mixture}). En réutilisant notre méthode nous obtenons respectivement $\approx$**75.5%** et $\approx$**69.9%** de précision. Ce sont de moins bonne précision et elles peuvent s'expliquer par plusieurs moyens, tout d'abord l'upsampling joue un rôle par la recopie de données déjà existantes. De ce fait il augmente l'occurrence des mots et favorise une meilleure précision. Pour ({False} vs {Mixture}) aucune donnée n'est upsampler, ceci implique potentiellement une plus grande diversité dans les mots utilisés, donc un nombre de features potentiellement plusieurs amoindris dû à notre nettoyage assez sélectif sur les valeurs à faible occurrence.

Le test sur les 3 classes reprend le même raisonnement vu précédemment, malgré l'UP sampling de notre classe *True*, nous avons les données des deux autres qui donnent un texte plus varié. De plus, la tâche de prédiction de notre classifier devient plus compliquée car il a dorénavant trois classes à prendre en compte.

Chapitre 5

Conclusion et observation

Le prétraitement étant assez conséquent il pourrait malgré tout être plus optimisé. Par exemple en utilisant un correcteur orthographique tel que TextBlob afin de minimiser au mieux le nombre de mots. Dans notre cas celui-ci n'est pas nécessaire dans le sens où nous manipulons des articles, donc vouer à être lus par de nombreux lecteurs, les fautes se font donc très rares, de plus l'exécution est très longue.

Au début du projet nous avions opté pour la lemmatisation plutôt que le stemming. Nous avons révisé notre choix car elle son temps d'exécution est plus élevée et n'était pas optimale pour notre utilisation. Bien que la lemmatisation permette de corriger les mots pour les ajuster à la phrase ceci implique donc une plus grande variété de features contrairement au stemming qui radicalise tout.

En concurrence au DownSampling, l'UpSampling a su montrer qu'il était le meilleur. Au cours de nos recherches, nous avions découvert l'Over-sampling (voir SMOTE) qui reprend le système de l'UpSampling mais en minimisant l'impact des sources d'erreurs qu'il peut provoquer en choisissant plus intelligemment la création des nouvelles données.

Notre sélection de features, bien que performantes pour le cas {True} vs {False} n'a pas forcément été la meilleure pour les deux autres classifications. Nous supposons notre sélection trop restrictive mais nous n'avons pas eu le temps de le vérifier cela avant la date de rendus. Des optimisations et visualisations seraient encore à fournir comme le choix des groupes de mots et les seuils que nous avons choisis jusqu'à maintenant de manière empirique.