Faculté des Sciences de Montpellier

Master 1 Imagine
Année 2021-2022

Rapport

Projet Machine Learning

Élèves :
Gauthier Germain (21501688)
Maxime Ginda (22114440)
Pierre Hennecart (22107715)
Albena Stefanova (22109542)
Benjamin Villa (22110753)

Professeurs :
Pascal Poncelet
Konstantin Todorov

1. Introduction

Notre projet s'inscrit dans le contexte de l'apprentissage supervisé afin de mettre en application les connaissances acquises lors des cours de Machine Learning. L'objectif de ce projet est de trouver le modèle le plus performant et le plus adapté pour classer les informations en fonction de leur véracité (fake news, real news ou mixte). Les principales étapes par lesquelles nous sommes passés lors de ce projet sont celles du pré-traitement des données, de classification des données et d'analyse des résultats obtenus. Le jeu de données utilisé pour ce projet a été généré par ClaimsKG.

2. Ingénierie de données

Le traitement des données textuelles dans l'ensemble est difficile car il dépend fortement des types de données disponibles telles que le texte brut, la liste de mots-clés et autres. Nous avons surtout utilisé la bibliothèque Python NLTK. Nous avons décidé d'utiliser trois classes *Transformer* différentes, spécifiques à chaque type de données, en fonction des différentes colonnes. Nous présenterons d'abord les différentes étapes de prétraitement séparément, puis la manière dont nous les avons utilisées dans un contexte spécifique à chaque colonne.

a. Prétraitement

Voici tous les paramètres que nous avons mis en œuvre pour l'étape de prétraitement :

- **Lowercase**
- **Suppression des caractères spéciaux** (ponctuation, chiffres...)
- **Substitution des espaces multiples par un seul espace**
- **Découpage en tokens** (tokenisation).
- **Étiquetage grammatical** (Part of Speech Tagging).

  L'étiquetage morpho-syntaxique (ou étiquetage grammatical) permet d'associer à chaque mot d'un texte l'information grammaticale correspondante.

  Nous avons apporté une modification par rapport à l'étiquetage dans les TPs. En raison de la multiplicité des types de noms, verbes, adjectifs, etc., nous avons simplifié l'utilisation du filtre Étiquetage en ajoutant simplement la forme de base.

  Par exemple, si nous voulons garder uniquement les verbes et les noms, il suffit d'ajouter ['NN', 'VB'] et le programme générera automatiquement toutes les sous-catégories de noms et de verbes nécessaires.

  Remarque : Ce filtrage est effectué avant la racinisation et la lemmatisation.
- **Stop words**.

  Comme toutes les données sont en anglais, nous avons utilisé la version anglaise pour les Stop Words.

- **Stemming** et **Lemmatisation**.

  Nous avons ajouté une vérification supplémentaire pour le processus de lemmatisation. Si un mot n'est pas spécifiquement lemmatisé comme un verbe, le résultat ne sera pas satisfaisant, car il dépendra de la forme du verbe et non pas de l'infinitif.

  Nous avons résolu ce problème en vérifiant le type morphologique du mot, en utilisant l'algorithme d'étiquetage grammatical décrit précédemment.

- **Spellcheck**

  La bibliothèque autocorrect propose une vérification et une correction orthographique. Même si cela ne fait généralement pas une grande différence, nous l'avons conservé comme option.

b. Les classes Transformer

Nous allons maintenant expliquer comment nous avons utilisé les différents prétraitements pour des colonnes spécifiques. Nous avons créé trois classes *Transformer* : *TextNormalizer*, *KeywordsNormalizer* et *DateNormalizer* qui héritent de *BaseEstimator* et *TransformerMixin*.

- **TextNormalizer**

  Pour la classe *TextNormalizer*, nous avons ajouté la possibilité d'appliquer tous les prétraitements proposés ci-dessus. Nous utilisons ce *Transformer*, pour les deux colonnes de texte, organisées comme des phrases (« headline » et « text »).

- **KeywordsNormalizer**

  Nous avons décidé qu'il serait préférable de combiner chaque groupe de mots-clés. Par exemple, initialement le champ *"Health Care,Climate Change"* est ensuite traité comme *"Health Care"* et *"Climate Change"*. De plus, pour éviter les résultats indésirables causés par la vectorisation, nous combinons les mots en minuscules sous la forme *"healthcare climatechange"*, et les séparons par un espace vide.

  En outre, dans cette colonne il n'y a pas de variations comme le temps du verbe ou la forme de l'adjectif comme dans le texte. À cause de cela nous avons décidé de ne pas donner autant de possibilités dans le choix de prétraitement que dans le texte.

  La logique derrière le prétraitement des colonnes « author » et « keywords » est aussi exactement la même et c'est pourquoi nous utilisons *KeywordsNormalizer* pour ces deux colonnes.

- **DateNormalizer**

  Toutes les dates ont le même format : Année-Mois-Jour. Nous les transformons en leur valeur ordinale qui représente le nombre de jours à partir de la date 01-01-01.

3. Classification

a. Processus de classification

La mise en place de la classification suit 3 étapes :

1. Définition des pipelines, contenant pour chacun la référence d'une des colonnes de la base de données grâce à la fonction ColumnSelector, qui subira un prétraitement, avant d'être encodé dans des vecteurs. On définit un Pipeline pour les colonnes text, author, date, keywords et headline.
2. On va pouvoir ensuite transformer les sorties de ces pipelines en utilisant le Feature Union, qui nous permettra de concaténer leurs données en une liste. De plus, on pourra également apposer un poids sur chaque pipeline, pour définir leur importance lors de la phase de test des résultats.
3. Enfin, nous pouvons passer à la mise en place des différents classifieurs que nous avons choisi dans de nouvelles pipelines, qui utilisent cette liste de donnée issue du FeatureUnion :

- SVC
- Logistic Regression (LR)
- Multinomial NB (MNB)
- Bernouilli NB (BNB)
- Complement NB (CNB)
- Gaussian NB (GNB)
- Random Forest (RFC)
- Decision Tree (DTC)
- K-Neighbors (KNN)

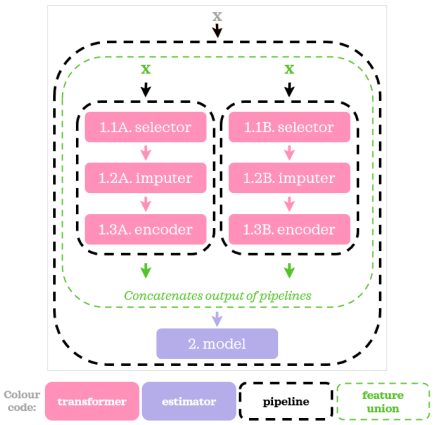


FIGURE 1 – Fonctionnement des Pipelines et du Feature Union

b. Évaluation du meilleur modèle par Cross Validation avec GridSearchCV

On va ensuite effectuer une recherche des meilleurs hyperparamètres. Pour cela, pour chaque classifieur, on a dressé une liste de paramètres, et de leurs valeurs possibles, car chacun d'entre-eux possède des paramètres différents.

Avec nos pipelines composés du FeatureUnion et de nos classifieurs, nous avons automatisé la recherche de leurs meilleurs hyperparamètres par une fonction que nous avons créé, qui effectue une passe de cross-validation via GridSearchCV, une fonction de scikit-learn effectuant une cross validation par K-Fold. Cette fonction sera appelée pour chaque modèle, et prendra en entrée leur pipeline et renvoie en sortie une liste ordonnée des modèles en fonction de leur score (accuracy) moyen.

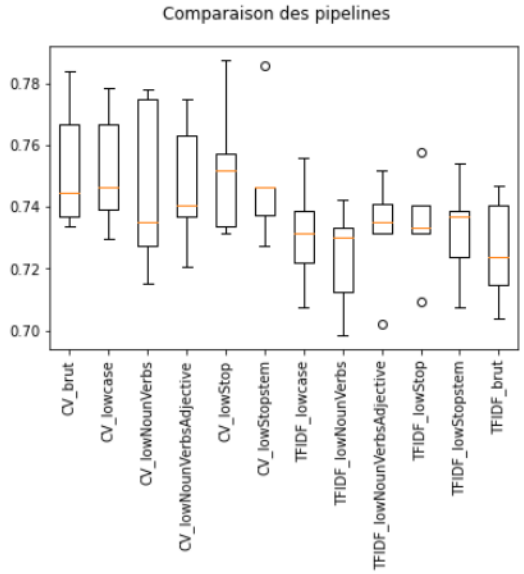
4. Analyse des Résultats

a. Paramètres de prétraitement sur le texte

Nous allons vous montrer l'impact de quelques paramètres de prétraitement sur la variation de la plus haute précision moyenne obtenue après cross validation pour une tâche de classification bien précise : TRUE vs FALSE

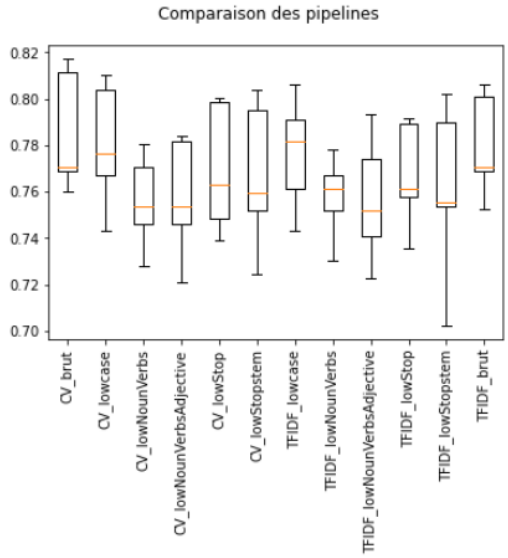
La première chose que l'on a remarqué, c'est que les prétraitements utilisés dans le *TextNomalizer* pour les colonnes avec des phrases sont loin de donner une amélioration significative de la précision du modèle. Au contraire le texte brut non traité propose de meilleur résultats.

Le choix du type de vectorisation est un peu plus clair et il semble avoir un certain avantage à utiliser un *CountVectorizer* au lieu d'un *TfidfVectorizer* en particulier sur la colonne texte (amélioration de la précision de plus de 2%)



Pipeline	Score
CV_brut	0.753
CV_lowStop	0.752
CV_lowcase	0.752
CV_lowStopstem	0.749
CV_lowNounVerbsAdjective	0.747
CV_lowNounVerbs	0.746
TFIDF_lowStop	0.735
TFIDF_lowNounVerbsAdjective	0.732
TFIDF_lowStopstem	0.732
TFIDF_lowcase	0.731
TFIDF_brut	0.726
TFIDF_lowNounVerbs	0.723

(a) application sur la colonne *text*



Pipeline	Score
CV_brut	0.786
CV_lowcase	0.780
TFIDF_brut	0.780
TFIDF_lowcase	0.777
CV_lowStop	0.770
CV_lowStopstem	0.767
TFIDF_lowStop	0.767
TFIDF_lowStopstem	0.761
TFIDF_lowNounVerbs	0.758
CV_lowNounVerbsAdjective	0.757
TFIDF_lowNounVerbsAdjective	0.757
CV_lowNounVerbs	0.756

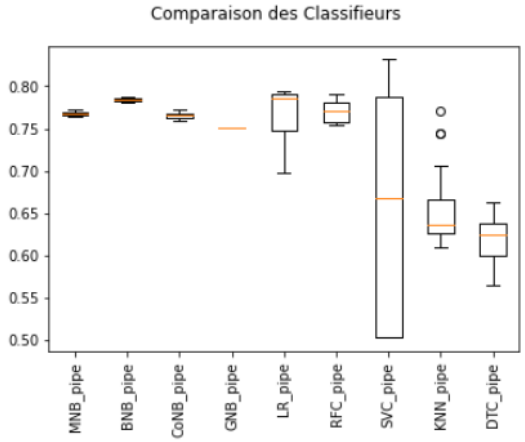
(b) application sur la colonne *headline*

FIGURE 2 – Mesure de l'efficacité des Prétraitements sur la précision du modèle

b. Stabilité du classement des classifieurs

On observe aussi peu de différences des prétraitements sur le classement des différents classifieurs sur notre base de donnée. Pour illustrer cela on a réalisé deux cas extrêmes avec deux chaînes de prétraitement qui diffèrent en tout : choix du type de vectorizer, choix du ColumnTransformer et choix des poids appliqué aux colonnes totalement différents (cf. notebook pour les détails précis de ces changements).

Pipeline	Score
BNB_pipe	0.784
RFC_pipe	0.771
MNB_pipe	0.768
CoNB_pipe	0.765
LR_pipe	0.763
GNB_pipe	0.751
KNN_pipe	0.653
SVC_pipe	0.652
DTC_pipe	0.618



Pipeline	Score
BNB_pipe	0.775
GNB_pipe	0.773
MNB_pipe	0.759
CoNB_pipe	0.758
RFC_pipe	0.746
LR_pipe	0.739
KNN_pipe	0.681
SVC_pipe	0.613
DTC_pipe	0.591

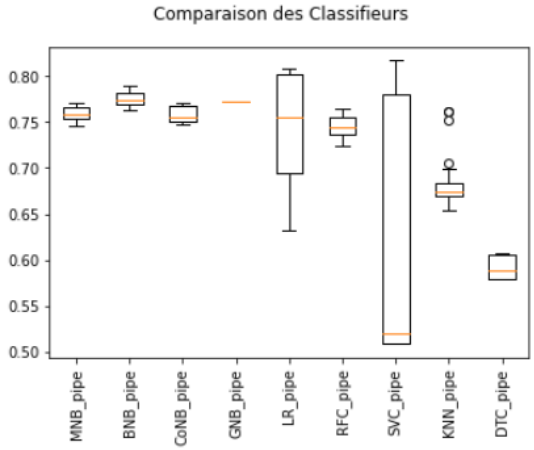


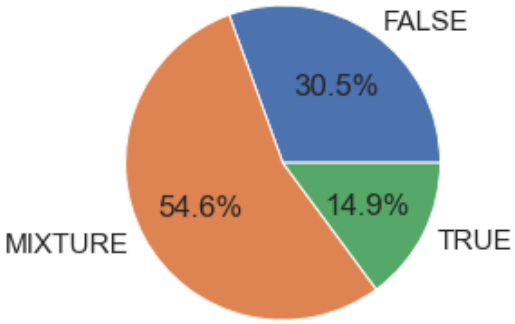
FIGURE 3 – Classement des classifieurs après cross validation sur deux chaînes de prétraitement totalement différentes

Le **DecisionTreeClassifier** fait systématiquement les plus mauvais scores. Le **SVM** quant à lui peut avoir des scores très bons lors de la phase de cross validation mais possède une trop grande variance pour être retenu De manière empirique on trouve donc que les meilleurs classifieurs dans notre projet sont ceux du type Naive Bayes (en particulier le **BernoulliNB** et le **MultinomialNB**) ainsi que le **RandomForestClassifier**.

c. Gestion du déséquilibre de la base de donnée

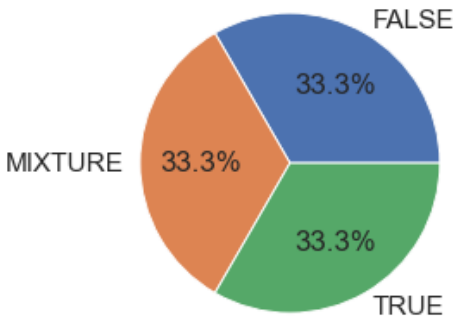
On a vite réalisé que les classes dans la base de donnée étaient déséquilibrées. La première idée était de supprimer les lignes de la classe majoritaire pour avoir des classes de même taille. Mais on s'est rendu compte que l'on perdait trop de données suite à cette opération. On s'est donc servi de la fonction *resample()* de *Scikit Learn* pour faire l'opération inverse et suréchantilloner les classes minoritaires.

```
MIXTURE    3196
FALSE      1788
TRUE        871
Name: ratingName, dtype: int64
```



(a) Avant

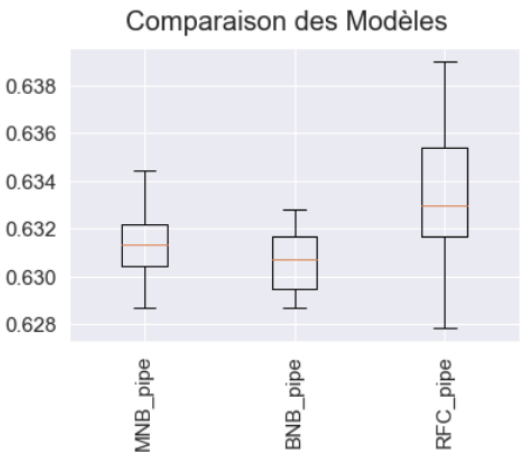
```
FALSE      3196
MIXTURE    3196
TRUE       3196
Name: ratingName, dtype: int64
```



(b) Après Upsampling des classes minoritaires (TRUE et FALSE)

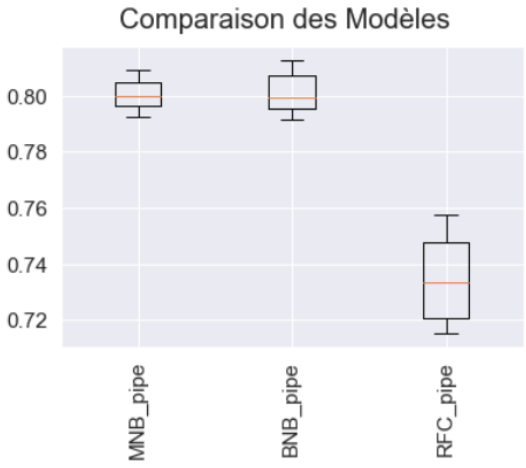
FIGURE 4 – Rééquilibrage des classes par Upsampling

```
Pipeline      Score
----------  -------
RFC_pipe      0.633
MNB_pipe      0.631
BNB_pipe      0.631
```



(a) Base de donnée rééquilibrée par downsampling de la classe majoritaire

```
Pipeline      Score
----------  -------
BNB_pipe      0.801
MNB_pipe      0.801
RFC_pipe      0.735
```



(b) Base de donnée rééquilibrée par upsampling de la classe minoritaire

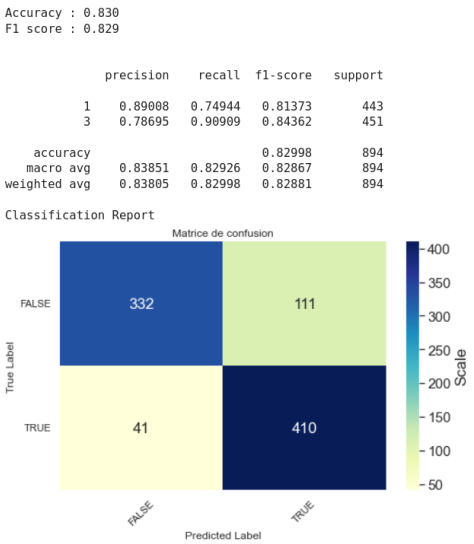
On a pas remarqué de phénomène d'overfitting et on a toujours eu des meilleurs résultats en effectuant de l'upsampling plutôt qu'en effectuant un downsampling des classes majoritaires (amélioration significative de la précision de plus de 15%).

d. Résultats finaux

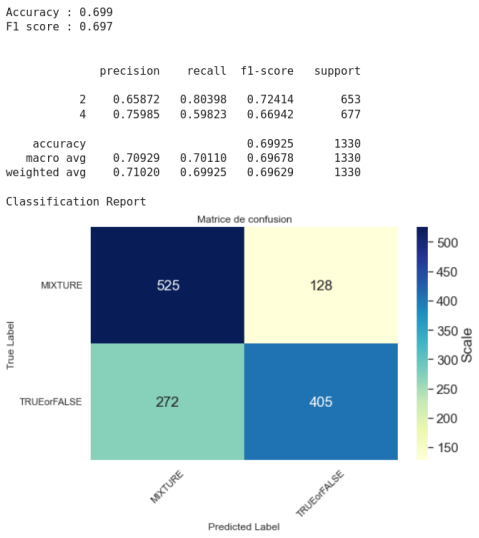
Une chaîne de classification satisfaisante est la suivante :

- Texte en Lowercase et removeStopWords en CountVectorizer pondéré à 1.
- Auteur en Lowercase CountVectorizer pondéré à 0.5.
- Date en CountVectorizer pondéré à 0.3.
- Headlines en Lowercase TfidfVectorizer pondéré à 1
- meilleurs classifieurs BernouilliNB et MultinomialNB

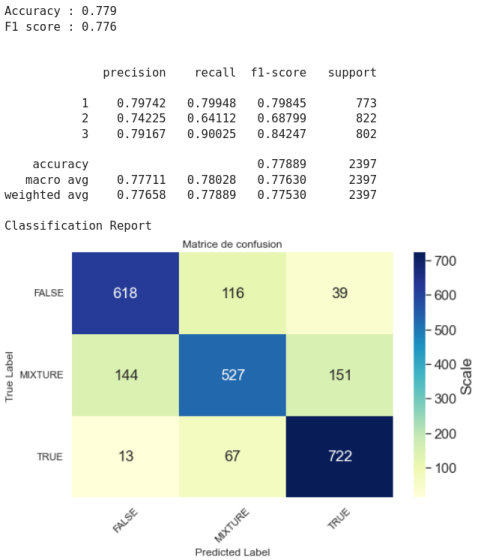
Cette chaîne de classification est mieux détaillée dans le notebook et nous donne les résultats suivants :



(a) True vs False



(b) (True or False) vs Mixture



(c) True vs False vs Mixture

FIGURE 6 – Scores et Matrices de confusion résultantes sur les 3 tâches de classification

5. Bilan

a. Conclusion

Après de nombreux tests empiriques afin d'acquérir une affinité avec les principales bibliothèques de Machine Learning, nous avons pu initier nos premières tentatives de création et entraînement d'un modèle. Au départ, notre accuracy était basse et chaque gain demeurait peu significatif malgré tous nos essais. À ce moment, nous ne considérions qu'une colonne par modèle, c'était là notre erreur.

Heureusement, notre découverte des FeatureUnion nous a permis d'étendre nos traitements à plusieurs colonnes et enfin obtenir des résultats intéressants. Conjoint à l'intégration de la détermination automatique des hyper-paramètres, notre modèle était désormais solide. Le dernier gap fut d'ailleurs franchi suite à l'ajout d'un système d'up-sampling.

b. Pistes d'amélioration

- Transformer notre modèle de Machine Learning en modèle de Deep Learning
- Optimiser la paramétrisation des transformers lors de l'usage de FeatureUnion
- Calculer en amont un paramètre de confiance pour chaque auteur basé sur sa proportion d'articles véridiques
- Approfondir le traitement des named_entities