M1 INFORMATIQUE AIGLE

HMIN232M

MÉTHODES DE LA SCIENCE DES DONNÉES

Classification de documents d'opinions

Anis KALOUN
Corentin KIRSCHER
Vincent MAZEL

Sommaire

Table des matières

I – Introduction

De nos jours, avec le développement d'internet et l'explosion de la quantité de données que nous pouvons trouver en ligne, il est parfois difficile de juger de la véracité de ces-dernières, et un travail de recherche est assez souvent nécessaire afin de savoir si oui ou non, nous sommes en présence d'une fake news ou de quelque chose de véridique.

Cependant, l'avancée de nouvelles technologies nous donne de nouveaux outils afin de répondre à ce problème et de pouvoir affirmer avec plus de certitudes – sans toutefois être totalement sûr – que l'article présent sous nos yeux reflète bien un fait réel.

En effet, cette multiplication des données a permis l'apparition d'une nouvelle branche du doux nom de « Deep Learning », ou « Apprentissage Profond » en français. Comme son nom l'indique, nous allons faire apprendre à une machine à différencier un vrai article d'un fake new, à évaluer la véracité de données récupérées en ligne.

Et pour en arriver là, nous allons justement nous servir de ces très nombreuses données déjà présentes en ligne qui seront justement utilisées dans cette phase d'apprentissage, avec pour objectif de mettre au point un système ayant le meilleur pourcentage de précision dans la reconnaissance d'article fake ou non possible.

II – Traitements des données

Nous arrivons ici à une partie très importante du projet – si ce n'est la plus importante – qu'est le traitement des données. En effet, il est primordial de passer par cette étape afin de présenter à notre algorithme de Deep Learning des données propres et le plus représentatives de chaque article, sans quoi le pourcentage de notre machine se limiterait à des scores très faibles.

A – Sélection des données à conserver

Il est important de préciser en premier lieu la structure initiale des données avec lesquelles nous allons travailler tout au long de ce projet. Ces-dernières sont classées dans de multiples colonnes, que nous pouvons voir en annexe[1]. Et il était important avant toute chose d'analyser ces différentes colonnes afin de comprendre leur utilité et de savoir si oui ou non il est possible de les exploiter.

En effet, comme on peut l'observer sur l'affichage des informations de notre dataframe, de nombreuses colonnes sont très difficilement exploitables – pour ne pas dire pas exploitables pour certaines – à cause par exemple de la présence de très peu de valeurs « non-null ». Certaines sont mêmes totalement vides comme « claimReview_author » ou « claimReview_author_url ».

Pour ce qui est des colonnes n'ayant que très peu de valeurs[2], comme par exemple « rating_bestRating » avec 2704 lignes ou « rating_worstRating » avec 2480 lignes, il nous était très complexe de les utiliser dans notre apprentissage. En effet, le dataframe étant composé d'un peu moins de 38000 lignes, il est difficile de garder une colonne dans laquelle plus de 90% des données seront « null ».

De plus, il est également important de considérer un deuxième problème que nous ne voyons pas sur cet affichage des informations de notre dataframe : de nombreuses colonnes étant en apparence remplies car affichant quelques 38000 lignes considérées comme « non-null » ne le sont en fait pas, car étant composée d'un très grand nombre de « NaN », ou « Not A Number », qui est évidemment une donnée qui ne nous intéresse pas, ou encore d'arrays vides « [] » ce qui est le cas pour la colonne « extra_entities_keywords » qui n'est finalement pas très remplie.

[1] Voir Annexes : A – Structure du Dataframe initial
[2] Voir Annexes : B – Représentation graphique des valeurs nulles

Une fois ces colonnes difficilement exploitables supprimées, nous avons également décidé de nous passer de données que nous considérions comme peu ou pas utiles :

- **claimReview_author_name :** cette colonne ne représente pas l'auteur de l'article mais le site sur lequel la vérification de la véracité de l'article a été réalisé, ce qui selon nous nous importe peu

- **claimReview_url :** le lien du site ayant vérifié la véracité de l'article

- **extra_title :** pour ce qui est de l'extra title, il est vrai que nous aurions pu le conserver mais nous trouvions le contenu de ce-dernier très similaire avec la colonne « claimReview_claimReviewed », avec de nombreux mots se répétant dans les deux colonnes

Cela nous laisse donc après toutes ces suppressions les colonnes suivantes :

- **claimReview_claimReviewed :** contient le texte

- **cleanUrl :** lien d'une personne ayant partagé l'information

- **entity :** les entités (mots-clés) liées à l'information

- **author-name :** le nom de la personne à l'origine de l'information

- **day, month year :** la date de publication

- **truth_rating :** un chiffre représentant la véracité de l'information (FALSE = 1, MIXTURE =2, TRUE = 3, OTHER = -1)

B – Traitements appliqués sur les données conservées

Si le traitement des données ne consistait qu'en la simple suppression de colonnes, le projet aurait été bien trop simple. En effet, maintenant que nous connaissons les données sur lesquelles nous allons travailler, il faut faire en sorte que ces-dernières soient exploitables : tandis que le texte doit subir de nombreuses modifications afin d'augmenter la précision de notre algorithme, certaines données telles que des urls doivent également subir un traitement afin de sortir la partie du lien nous intéressant vraiment, tandis qu'une colonne comme « extra_entities_claimReview_claimReviewed » ne sera pas utilisable en l'état car étant une array d'objets Json comportant des erreurs de syntaxe.

1) Traitement des URLS

Contrairement aux URLS que nous avons supprimés représentants le lien du site ayant prouvé ou non la véracité d'un article, nous disposons des liens nous donnant le site sur lequel chaque article à traiter a été initialement posté, ce qui est évidemment une information intéressante.

Cela étant, et si nous avions décidé de laisser en l'état les liens à notre disposition, il aurait été impossible de les exploiter. En effet, nous pouvons voir ci-dessous un exemple de lien présent dans notre dataframe :

« https://t.co/Oo5Q56ALAu,https://twitter.com/ianbremmer/status/1180501727960866816?ref_src=twsrc%5Etfw,https://www.instagram.com/p/B3H7aF7hHAa/,https://www.cbsnews.com/news/sauli-niinisto-and-trump-press-conference-president-finland-receives-praise-and-memes-after-white-house-meeting/,https://twitter.com/hashtag/TrumpMeltdown?src=hash&ref_src=twsrc%5Etfw,https://t.co/Ko4o4mxVLN,https://twitter.com/DogsHateBoots/status/1179476215327010816?ref_src=twsrc%5Etfw »

Un des problèmes que nous avons dans ce qui a été mis en place pour la récupération de liens et que nous nous sommes rendu compte trop tard que nous ne sommes pas en présence d'un seul mais de plusieurs liens séparés par des virgules, et que par conséquent le lien que nous récupérons à chaque fois n'est autre que le premier dans chacune des lignes. Par exemple, après traitement, le lien récupéré ici est « t.co ».

Par conséquent, une des améliorations que nous aurions pu appliquer sur notre projet aurait été la récupération de l'intégralité des liens au lieu de ne se servir que du premier.

Pour aller plus loin, nous aurions même pu imaginer un système récupérant dans l'url non pas seulement le site sur lequel l'info a été publié mais également la personne ou le compte ayant relayé l'information, sachant que de nombreux comptes ou personnes auront plus tendances à partager des informations fausses.

Par exemple, dans ce lien : « https://twitter.com/ianbremmer/status ... » , nous pourrions potentiellement récupérer la partie « ianbremmer » qui est le nom du compte ayant relayé l'information.

2) Extraction des extra entities

Une des colonnes que nous avons conservée contient des « extra entities » dans des objets JSON qui sont des mots que nous pourrions considérer comme des mots clés et donc des mots importants. Nous avons dû en premier lieu « réparer » les JSON qui contenaient une erreur avant de pouvoir les exploiter. En effet, comme nous pouvons le voir ci-dessous :
« {"id" : 1042690","""begin": 18,"end": 32,"enti... » , il y a une double quote en trop à côté de begin qui empêche notre algorithme de convertir la chaine de caractères en objet JSON.

Une fois cette opération effectuée, il ne nous reste plus qu'à aller récupérer la partie « entity » de chacun des objets.

3) Extraction du nom de l'auteur

Le nom de l'auteur est forcément un élément déterminant dans ce projet, mais il doit passer par une phase de traitement afin d'être pleinement exploitable. A la manière de l'extraction des extra entities, l'auteur doit être récupéré dans un objet JSON.

Mais au-delà de la récupération de cette information, il est également important de parler du nombre de lignes de notre dataframe contenant cette-dernière. En effet, plus des deux tiers de notre ensemble de données ne dispose pas de ce renseignement pourtant crucial.

Cela étant, nous avons quand même décidé de conserver cette colonne cette-dernière étant trop importante à nos yeux pour la supprimer.

4) Généralisation du « Truth Rating »

Lors de la phase d'apprentissage de notre algorithme, il est essentiel que les données que nous lui passions contiennent une information précisant si l'article en question est vrai, faux ou une mixture. Cependant, les données de base récupérées ne se limitent clairement pas qu'à ces trois catégories, et un traitement a donc été nécessaire : en effet, nous pouvions retrouver des « incorrect », « mostly correct », « legend » ou même des « pants on fire ». Une normalisation a donc été réalisé afin que chacune des lignes de notre dataframe possède une colonne « Truth Rating » dont les valeurs soient contenues dans les suivantes : 1 pour FALSE, 2 pour MIXTURE, 3 pour TRUE et -1 pour OTHER.

4) Traitement du texte des articles

Cette partie est probablement une des plus importantes du projet. En effet, il est ici nécessaire de mettre en forme le texte le mieux possible en retirant les caractères considérés comme « inutiles » et réduisant la précision de notre algorithme final : nous transformons les chiffres en lettres, retirons la ponctuation des textes, retirons les « stopwords » ou « mots vides » qui sont des mots tellement communs qu'il est inutile de les utiliser (ex : « le », « la », « de » …), retirons les mots fréquents[3] que nous considérons comme inutiles et qui apparaissent le plus dans nos textes, retirons également les mots considérés comme « rares[4] » et qui n'apparaissent que très peu de fois.

Enfin, nous appliquons également la méthode de lemmatisation qui consiste en la conversion d'un mot dans sa forme canonique : « continua », « continuait » et « continuant » renvoient tous à « continuer », tandis que « continuation » ou « continuations » renvoient à « continuation ». Cette méthode est intéressante car elle conserve le sens original du mot, à l'inverse d'une autre méthode appelée « Stemming » ou « Racinisation » qui ne fait pas cette distinction, et qui dans notre exemple aurait renvoyé pour chacun des cas le mot « continu ». C'est pour cette raison que nous avons choisi de nous orienter vers la lemmatisation.

5) Transformation des données en nombres

Une fois les pré-traitements réalisés sur l'ensemble de nos données, il nous faut transformer ces-dernières dans un format reconnaissable par les algorithmes d'apprentissage, ce qui veut dire qu'il est nécessaire de transformer les valeurs textuelles en chiffres.

Nous utilisons par conséquent les méthodes « LabelEncoder() » et « TfidVectorizer() » afin de respectivement transformer les mots colonnes composées de très peu de mots et le texte prétraité auparavant.

[3] Voir Annexes : C – 10 mots les plus fréquents
[4] Vois Annexes : D – 10 mots les plus rares

III – Classification, modèles et apprentissage

Nous arrivons désormais dans la dernière partie du projet. Avant de passer à l'apprentissage et au choix d'un modèle le plus performant possible, il est nécessaire dans notre cas de tester différentes classifications de données que sont les suivantes :

- **TRUE vs FALSE**
- **TRUE et FALSE vs MIXTURE**

Nous nous heurterons également à un problème de taille (littéralement) : en effet, il est nécessaire que nous renseignions à notre modèle autant de données d'une classe que d'une autre, autant de TRUE et de FALSE. Or, nos classes sont déséquilibrées, et nous allons devoir par conséquent essayer d'y remédier.

A - Downsampling et upsampling des classes

Comme énoncé ci-dessus, il était nécessaire de passer par cette étape. En effet, nous disposons d'à peu près de 14000 FALSE, 13000 MIXTURE et 4000 TRUE, ce qui est un problème car nous nous devons de donner à l'algorithme autant de données d'une classe que de l'autre.

Nous avons par conséquent décidé de downsample quelque peu le nombre de FALSE et de MXITURE et de parallèlement upsample le nombre de TRUE afin d'arriver à quelque chose d'équilibré. Nous avons choisi cette solution car nous ne voulions pas trop réduire notre nombre de données en mettant seulement en place un downsample des FALSE et MIXTURE au même niveau que les TRUE, mais en même temps nous ne voulions également pas upsample uniquement les valeurs TRUE pour les remonter au même niveau que les autres ce qui les aurait quelques peu faussées selon nous.

B – Utilisation de SelectKBest

Afin de savoir quelles sont les features les plus impactantes, nous avons décidé d'utiliser la fonction « SelectKBest » qui nous renseigne sur les scores[5] de chacune des features.

Bien qu'intéressante, nous avons eu du mal à la rendre pleinement fonctionnelle : en effet, nous avons utilisé la fonction « TfidVectorizer » sur notre colonne dédiée au texte. Cette fonction nous renvoie alors un vecteur qui nous donne pour chaque mot du texte en question une valeur. Mais le problème ici est que la fonction « SelectKBest » considère chacun des mots de ce vector comme une feature et leurs donne un score, ce qui fait que les résultats sont un peu faussés. Malheureusement, nous n'avons jamais réussi à régler ce problème.

Cela étant, nous avons également pu expérimenter manuellement l'efficacité des features utilisées en en enlevant une différente avant chaque test, et n'avons pu apercevoir de variations vraiment intéressantes sur la précision globale, sauf pour la colonne correspondant au texte. En effet, si nous enlevons cette-dernière, la précision chute drastiquement pour arriver quasiment à 50% de précision, ce qui reviendrait globalement à lancer une pièce et faire un pile ou face.

Nous pouvons par conséquent conclure que même sans avoir réussi à avoir le score de la feature du texte dans la fonction « SelectKBest », nous pouvons aisément supposer que ce-dernier était de loin le meilleur de tous.

C - Résultats obtenus

Globalement, nous pouvons affirmer que les résultats[6] sont moyens. En effet, le pourcentage de précision varie selon les algorithmes, allant de 62% avec le « Gaussian NB » jusqu'à 75% avec le « RandomForestClassifier ».

Cela étant, il est important de préciser que nous n'avons pas vraiment pu tester avec l'intgralité des données de notre dataframe. En effet, l'apprentissage des différents algorithmes était bien trop long, et nous pensons que la précision aurait pu augmenter dans des conditions optimales d'apprentissage.

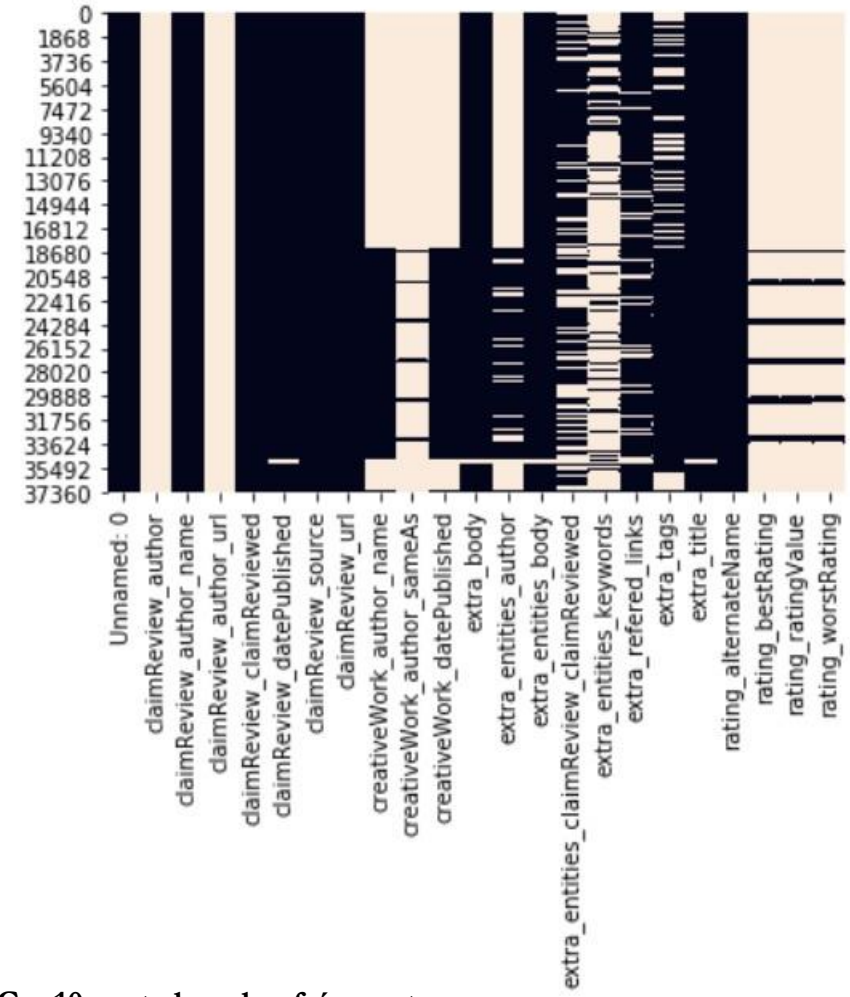
[5] Voir Annexes : E – SelectKBest scores
[6] Voir Annexes : F – Résultats algorithmes

V - Annexes

A – Structure du Dataframe initial :

```
Data columns (total 23 columns):
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   Unnamed: 0                              37687 non-null  int64
 1   claimReview_author                      0 non-null      float64
 2   claimReview_author_name                 37687 non-null  object
 3   claimReview_author_url                  0 non-null      float64
 4   claimReview_claimReviewed               37687 non-null  object
 5   claimReview_datePublished               37687 non-null  object
 6   claimReview_source                      37687 non-null  object
 7   claimReview_url                         37687 non-null  object
 8   creativeWork_author_name                16831 non-null  object
 9   creativeWork_author_sameAs              1310 non-null   object
 10  creativeWork_datePublished              16819 non-null  object
 11  extra_body                              37685 non-null  object
 12  extra_entities_author                   37687 non-null  object
 13  extra_entities_body                     37687 non-null  object
 14  extra_entities_claimReview_claimReviewed 37687 non-null  object
 15  extra_entities_keywords                 37687 non-null  object
 16  extra_refered_links                     34113 non-null  object
 17  extra_tags                              29423 non-null  object
 18  extra_title                             37687 non-null  object
 19  rating_alternateName                    37687 non-null  object
 20  rating_bestRating                       2704 non-null   float64
 21  rating_ratingValue                      2704 non-null   float64
 22  rating_worstRating                      2480 non-null   float64
dtypes: float64(5), int64(1), object(17)
memory usage: 6.9+ MB
```

B – Représentation graphique des valeurs nulles



La partie foncée représente les valeurs non nulles

C – 10 mots les plus fréquents

```
[('Says', 3900),
 ('Trump', 3147),
 ('President', 3012),
 ('U', 2756),
 ('climate', 2357),
 ('global', 2330),
 ('Donald', 2194),
 ('one', 2141),
 ('Obama', 2126),
 ('percent', 1823),
 ('shows', 1618),
 ('get', 1587),
 ('illegal', 1500),
 ('world', 1491),
 ('Bill', 1474)]
```

D – 10 mots les plus rares

```
print(RAREWORDS)
```

```
{'Nasir', 'ACTU', 'Greens', 'Critic', 'talent', 'Zamfara', 'JESUS', 'COMING', 'Rufai', 'SOON'}
```

E – SelectKBest scores

	Feature	Scores
38974	entity	27614.493344
38975	author_name	25411.276301
38973	cleanUrl	4038.992421
38976	day	158.855615
14373	claimReview_claimReviewed_global	46.094089
...	...	...
38966	claimReview_claimReviewed_zones allow	NaN
38967	claimReview_claimReviewed_zookeeper	NaN
38968	claimReview_claimReviewed_zookeeper paderborn	NaN
38971	claimReview_claimReviewed_zuma	NaN
38972	claimReview_claimReviewed_zuma claim	NaN

F – Résultats algorithmes