UNIVERSITÉ
DE MONTPELLIER

HAI817I - Machine Learning

Rapport de Projet

Détection de *fake news*

Étudiants :
Abla Amharref (22015772)
Roland Bertin-Johannet (22009809)
Tom BROS (22013249)
Rihab HAOULANI (21914085)
Manon Ramaroson (22102547)

Enseignants :
Pascal Poncelet
Todorov Konstantin

Mai 2022

1 Introduction

Ce projet a pour but de trouver les plus performants modèles de prédictions afin de pouvoir classer des assertions comme vraies ou fausses.

Nous allons donc expliquer à travers ce rapport comment nous avons procédé pour atteindre ce but. Dans un premier temps nous expliquerons le contexte général du projet ainsi les données utilisées. Ensuite, nous analyserons les données exportées afin d'effectuer les prétraitements appropriés et de rendre nos données prêtes à la classification. Puis nous présenterons les différentes méthodes de classification traditionnelles et nous les analyserons. Enfin, nous présenterons une approche de classification par le *Deep Learning*.

2 Données

Les données que nous avions à étudier étaient des informations extraites de déclarations[1] qui ont été vérifiées (*fact-checking*), et que nous pourrons nommer également articles. Elles ont ensuite été traitées par une machine d'exploration utilisant ClaimsKG pour les rentrer dans un fichier csv dont chaque ligne décrit une déclaration. Chacune d'entre elle est définit par 14 caractéristiques : *id*, *text*, *date*, *truthRating*, *ratingName*, *author*, *headline*, *named_entities_claim*, *named_entities_article*, *keywords*, *source*, *sourceURL*, *link*, *language*.

Revenons sur une description de certaines d'entre elles qui ont été utilisées pour nos fouilles de données. Le *truthRating* est l'attribut cible, par rapport auquel on souhaite classer nos données. Le vrai est représenté par un 3, et le faux par 1. Le *text* représente le contenu de la déclaration en chaînes de caractères et les champs *headline* et *keywords* représentent l'idée principale de la déclaration, avec *headline* semblable à un titre et *keywords* un ensemble de mots caractérisant l'assertion. Nous avons également pu étudier le champs *source* qui représente le site qui a vérifié l'assertion. Enfin, l'attribut *date* peut représenter la date de la déclaration ou celle de sa vérification. Dans le cadre de notre projet, nous avons uniquement utilisé des données dont le *truthRating* était vrai ou faux, mais pas les assertions mixtes. Notre jeu de données était composé de 7366 déclarations fausses et 2634 déclarations vraies. On peut observer un certain déséquilibre dans ces données, ce qui a dû être pris en compte pour le traitement de nos données.

3 Ingénierie des données

Après avoir examiné les données que nous souhaitons analyser, nous pouvons commencer à effectuer des traitements. Pour les analyser, nous avons fait le choix de comparer les différents modèles par rapport à l'exactitude (*accuracy*), car c'est dans notre cas la mesure la plus évidente du succès de notre modèle. Cependant nous avons régulièrement vérifié le f-score et les matrices de confusion pour être sûr que le modèle se comportait correctement. Nous avons tenté d'analyser le contenu des données mal classées, mais nous n'en avons pas tiré de conclusion. Dans cette section, nous allons lister les différents prétraitements que nous avons pu mettre en place :

- Dans un premier temps, inspirés par l'article[2] dans lequel les auteurs utilisent les antécédents des sources pour les articles (principalement des personnes) pour informer la détection de *fake news*, nous avons tenté d'établir des corrélations entre certains mots et les classes. Par exemple, dans les données ré-équilibrées, il y a deux fois plus de *fake news* contenant le mot 'Cruz' que d'articles considérés comme véraces contenant ce mot. Obtenir cette information pour tous les noms propres contenus dans les données d'entrainement pour éviter un biais, pour aider le classificateur au moment du test, pourrait augmenter l'exactitude . Cependant, de nombreux articles ne contenant pas de nom propre, nous avons décidé de ne pas poursuivre cette piste.

- Dans un second temps, nous avons essayé une variété de prétraitements :

    1. L'inclusion de n-grammes dans les *features* de nos *vectorizers*,

[1] https://data.gesis.org/claimskg/explorer/home (consulté le 20/05/2022)
[2] https://drive.google.com/file/d/1zEbxEfMfZn-2frsU2bVScvZM8dsDS3hZ/view (consulté le 20/05/2022)

2. La suppression des *stop-words*,
3. Le *Stemming* et la *Lemmatization* des mots des articles,
4. L'approche *Bag Of Words* avec la vectorisation TFIDF ou *CountVectorizer*, avec éventuellement une limite au nombre de *features*,
5. La suppression des chiffres et nombres,
6. La suppression de la ponctuation,
7. La prise en compte des colonnes "auteurs", "date", et "sources", en *one-hot-encoding*,
8. La prise en compte uniquement des noms, verbes, adjectifs, entités nommées, ou des combinaisons de ces éléments.

- Ensuite, nous avons tenté d'utiliser un *K-means* pour segmenter nos documents texte en plusieurs *clusters*, en espérant trouver des déséquilibres entre les deux classes au sein des *clusters*. Dans ce cas, un classificateur pourrait utiliser l'information du *cluster* où se trouve un article. En expérimentant sur le nombre de *clusters*, nous avons déterminé qu'avec 35 *clusters*, les classes semblaient plus inégalement réparties.
  Cependant, nous avons abandonné cette piste car seuls les *clusters* les plus petits présentaient un déséquilibre de représentation des classes (voir figure 1). Ainsi, sur la majorité des articles nous n'obtenions aucune information.

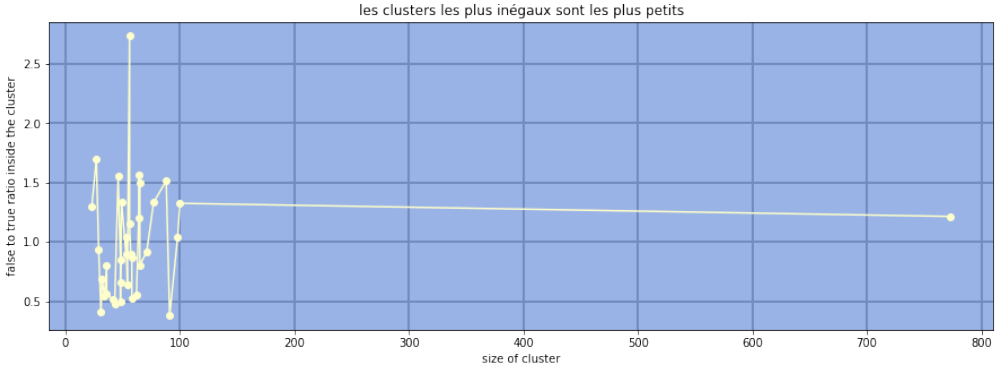
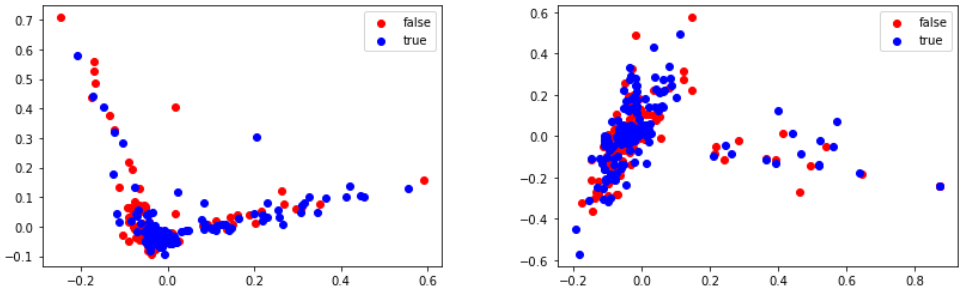


Figure 1: Ratio entre le nombre d'articles faux et vrais dans les *clusters*, en fonction de leur taille

- Nous avons utilisé des ACP (Analyse en Composantes Principales) avec deux objectifs :
  1. Visualiser les documents sur un graphique en deux dimensions, pour voir si nos données étaient séparables (sans que ce soit un test définitif, car l'ACP détruit de l'information, surtout si on projette sur deux dimensions). Voir figure 2 pour un exemple.
  2. Réduire le nombre de features des phrases vectorisées (passer de 15000 à 200 dimensions par exemple) pour faciliter la classification.



(a) Une centaine de documents dont on n'a gardé que les noms au sens grammatical, projetés sur deux dimensions avec une ACP.

(b) Cette fois-ci, on a gardé les noms, les verbes et les adjectifs.

Figure 2: Exemple de résultats de l'ACP

- Pour régler le déséquilibre entre les classes, nous avons essayé l'*undersampling*, ou de l'*oversampling* avec la bibliothèque SMOTE [3]. Nous avons fait l'*oversampling* séparément sur les données de test et d'entraînement, pour éviter que des textes générés à partir des données de test se retrouvent dans les données d'entraînement, ce qui fausserait nos résultats.

- Nous avons essayé de numériser le texte avec la matrice d'*Embedding word2vec-google-news-300* [4]. Pour transformer nos séquences de *Word Embeddings* en un seul vecteur, nous avons d'abord essayé de garder la moyenne des *Embeddings* de la séquence, puis de prendre l'*Embedding* avec la plus grande norme.

- Pour préparer les données pour un réseau de neurones récurrent, nous avons transformé les articles en séquences d'index d'un vocabulaire, que nous avons rembourrées avec des 0 pour qu'elles aient toutes la même longueur, et considéré les possibilités suivantes :

  1. Normaliser les index pour qu'ils soient tous entre 0 et 1, et les passer en entrée du modèle,
  2. Apprendre une matrice d'*Embedding* au sein du modèle,
  3. Utiliser du *one-hot-encoding* pour avoir en entrée du modèle, des vecteurs catégoriques faisant la taille du vocabulaire.

4 Méthodes Traditionnelles

Après avoir énuméré nos méthodes de prétraitement, intéressons-nous aux modèles traditionnels de classification.

4.1 Classifications

Tout d'abord, nous avons testé les classificateurs suivants sans prétraitement afin de trouver un modèle classifiant au mieux les données. Nous devions trouver une combinaison entre nos prétraitements et nos classificateurs satisfaisante. Voyons alors les classificateurs étudiés :

1. Logistic Regression
2. Decision Tree
3. SVM ou séparateurs à vaste marge
4. KNearestNeighborsClassifier
5. MultinomialNB
6. RandomForestClassifier
7. PassiveAggressive

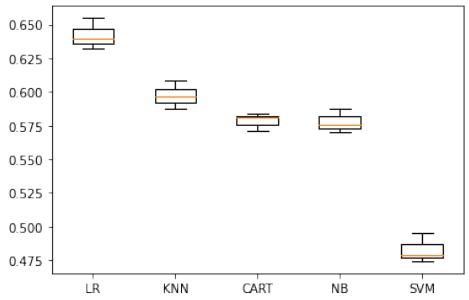
4.1.1 Classifications sans prétraitements

L'effet que nous avons pu observer est que l'exactitude était aux alentours des 50%. Dans nos matrices de confusion, cela s'exprimait souvent par une quantité relativement importante de *False Negative* et *False Positive*. Sur quelques tests, nous avons tenté d'observer des différences entre l'étude sur les champs *text* et *headline*. Nous n'avons pas tiré de conclusion sur ces derniers.

4.1.2 Classifications avec prétraitements

Une première approche des classifications a été de s'inspirer de l'article[5] sur la détection de *fake news* : en gardant les majuscules, en retirant la ponctuation et les stopwords, on obtient des résultats en terme d'exactitude, observables en figure 3. Par la suite, nous avons expérimenté un plus large ensemble de combinaisons de nos prétraitements et classificateurs. Remarquons que si SVM a été peu performante sur ce test, avec les bons paramètres elle se comportait mieux. Nous reviendrons sur l'analyse des résultats par la suite.

[3] SMOTE : https://jmlr.org/papers/v18/16-365.html (consulté le 20/05/2022)
[4] Téléchargée avec l'API Gensim https://radimrehurek.com/gensim/intro.html (consulté le 20/05/2022)
[5] Lien de l'article sur les méthodes de classifications en machine learning sur les *fake news* : ici (consulté le 20/05/2022)



(a) Logistic Regression, KNearest Neighborsp,Classifier And Regression Tree, Native Bayes, Support Vector Machines

Figure 3: Boîtes à moustache de nos classificateurs traditionnels

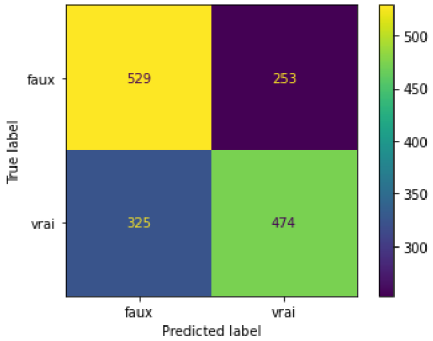


Figure 4: Matrice de Confusion obtenue en réalisant une classification avec *Logistic Regression*, dont le prétraitement consistait à enlever les *stop-words* et la ponctuation

4.2 Analyse

Dans cette section nous allons expliquer les différents choix faits par rapport à leur impact sur la classification.

Choix sur les caractéristiques (*features*) :

Le champ *text* a été notre principale caractéristique a étudier car il regroupait la déclaration dans son entièreté. Quant à la caractéristique *headline*, elle donne des résultats équivalents au champ *text* (ils sont mêmes parfois meilleurs). Nous ne l'avons pas expérimentée sur tous nos modèles car cette piste ne paraissait pas être la bonne, spécialement dû à sa taille.

Prétraitements :

Nous allons étudier les prétraitements énoncés dans la section 3 afin de pouvoir expliquer leurs conséquences sur la classification. On démarque donc 3 cas de prétraitements, ceux n'ayant pas d'effet, certains diminuant l'exactitude et d'autres l'augmentant :

1. Concernant les prétraitements ayant peu d'effet, nous avons pu recenser la *Lemmatization* et le retrait des nombres. Il semble que la simplification du texte apportée par la *Lemmatization* ne permette pas de meilleurs résultats.

2. Nous avons pu observer un effet négatif de certains prétraitements. Un premier prétraitement ayant cet impact est le *Stemming*. Selon notre interprétation, cela vient du fait que le *Stemming* enlève de l'information au texte.

   On note cette baisse d'exactitude aussi sur les *Word Embeddings* que ce soit avec une moyenne des mots ou avec le mot de plus grande norme. Nous attribuons cela à l'étape de transformation des séquences d'*Embeddings* en un seul vecteur.

   En effectuant un traitement sur notre texte qui ne garde que les noms, verbes, adjectifs ou des entités nommées ou une composition d'entre eux, nous avons constaté une chute de l'exactitude. Peut-être que des informations sur le style de l'article étaient contenues dans des mots filtrés par ce procédé.

3. Nous avons privilégié les prétraitements suivants dû à leur augmentation de l'exactitude :

   - Vectoriser les phrases avec TFIDF ou *CountVectorizer*. Nous remarquerons que les meilleurs tests ont été obtenus avec *CountVectorizer* même si la différence entre les deux méthodes n'est ni considérable ni systématique.
   - Conserver les majuscules.
   - Mettre les n-grammes à 1. Nous avons pu tester les n-grammes sur tous les classificateurs, en augmentant les n-grammes les changements n'étaient pas notable alors qu'en enlevant les 1-grammes on pouvait observer une diminution des scores d'exactitude.
   - Retirer les *stop-words* et la ponctuation.

Modèles :

Expliquons à présent nos choix quand aux classificateurs séléctionnés. Dans nos expérimentations, selon les prétraitements (vus précédemment), l'ordre de succès des classificateurs variait grandement. Ainsi, nous avons fait le choix de ne pas nous concentrer sur un seul modèle. Néanmoins certains classificateurs traditionnels se sont légèrement démarqués des autres avec les prétraitements, comme SVM et LR.

En effet, malgré le résultat de SVM sur la figure 3, nous avons obtenu des résultats de l'ordre de 65% d'exactitude en utilisant *Grid Search*. Le classificateur Régression Logistique a obtenu des scores du même ordre. Une matrice de confusion de ce dernier est représentée en figure 4.

Échantillonnage :

Au début de nos tests nous nous étions limités à 500 échantillons d'assertions de chaque classe, nous avions réussi à obtenir entre 54 et 59% d'exactitude. C'est en les augmentant pour atteindre un nombre de 2000 échantillons de chaque, que l'exactitude a elle aussi augmenté en passant à 65% (sans avoir à changer les prétraitements). Il parait logique qu'avec plus d'exemples pour s'entraîner, des modèles comme LR et SVM se comporteraient mieux.

Augmenter le nombre d'échantillons était donc une bonne amélioration mais nous réalisions de l'*undersampling* sur les fausses déclarations. C'est pourquoi nous avons décidé de faire de l'*oversampling* avec SMOTE (voir section 3) avec un modèle LR. Nous n'avons pas noté de différence entre l'*oversampling* avec SMOTE et l'*undersampling* (suppression des échantillons de certaines classes pour atteindre le même nombre dans chaque classe). Peut-être que l'algorithme de SMOTE n'a pas généré des données utiles dans notre cas, ou bien que 2000 données par classe suffisaient à l'entraînement de nos modèles.

Expérimentation finale sur les méthodes traditionnelles :

Pour assurer autant que possible nos résultats, nous cherchons à lancer une expérimentation plus complète. Pour ce faire, nous avons défini et utilisé une fonction se nommant *launchExperimentations* prenant en argument une liste de modèles, qui réalise une 10-validation sur chacun de ces modèles avec toutes les combinaisons possibles de prétraitements parmi ceux-ci :

1. Retirer les nombres
2. Retirer les *stop-words*
3. Faire du *Stemming*
4. Retirer la ponctuation
5. Ajouter auteurs
6. Ajouter dates
7. Ajouter sources

Cette fonction nous retourne le score d'exactitude et le f-score.

Nous utilisons *launchExperimentations* avec la liste de modèles suivante : SVM, KNN et LR. Nous remarquons que les mêmes prétraitements n'ont pas le même effet sur les différents classicateurs. On obtient alors 12 exactitudes pour SVM et LR supérieures à 65% contre 4 pour KNN.

Les meilleurs taux d'exactitude sont à 65,8%, ils ont été obtenu avec deux modèles SVM (voir Table 1).

Modèle \ Prétraitements	Nombres	*Stop-words*	*Stemming*	Ponctuation	Auteurs	Dates	Sources
SVM	False	True	True	True	True	True	False
SVM	False	True	True	False	True	True	False

Table 1: Tableau des différents prétraitements pour les meilleurs taux d'exactitude obtenus

5 *Deep Learning*

Après avoir passé la majorité du projet sur une approche par classificateurs "traditionnels", nous voulons à présent essayer une approche *Deep Learning*. L'approche *Bag of Words* employée précédemment a l'inconvénient assumé d'ignorer la nature séquentielle des informations textuelles. Nous pouvons confronter ce problème grâce aux Réseaux de Neurones Récurrents ; parmi eux, les LSTM (Long-Short-Term-Memory) ont l'utilisation la plus répandue. Après avoir trouvé des antécédents de classification

de *fake news* à l'aide d'un LSTM[6], nous avons utilisé les LSTMs. Pour expliquer notre démarche, nous présentons une tentative naïve, puis les améliorations successives que nous lui avons apportée.

5.1 Tentative naïve

Tout d'abord, nous essayons d'utiliser un réseau LSTM avec des vecteurs de taille 256 passant d'une unité à l'autre [7]. À la sortie de cette couche LSTM, nous ajoutons deux unités entièrement connectées avec un softmax comme activation. Pour préparer les données, nous procédons comme suit :

1. Tokeniser les champs *text* de nos données
2. Limiter le nombre de *tokens* à 500, nous verrons sur la Figure 5 qu'un plus grand vocabulaire n'aide pas.
3. Ajouter des 0 dans les séquences de *tokens* pour qu'elles fassent toutes la taille de la plus grande.
4. Coder chaque *token* en *one-hot-encoding*. Les séquences de vecteurs serviront d'entrée au réseau.
5. Pour l'instant nous équilibrons les classes en supprimant le surplus de faux.

Après un entraînement de 4 *epochs* par paquets (*batch*) de 10 (on obtient le même résultat avec 50), nous obtenons 74% d'exactitude sur les données d'entraînement, contre seulement 61% sur les données de test. Nous avons fait de l'*overfitting*. Pour y remédier nous essayons d'ajouter du *Dropout* après la couche LSTM. L'exactitude sur les données de test monte alors à 65%.

Nous avons essayé les deux choses mentionnées à la fin de la section 3 (apprendre un *Embedding* dans le modèle ou représenter les phrases comme séquences d'index normalisés), sans progrès des scores. Arbitrairement, nous choisissons maintenant de rester sur des séquences de vecteurs en *one-hot-encoding*.

5.2 Première amélioration : *Oversampling*

La plus criante amélioration est de ne pas supprimer 5000 données pour équilibrer les classes, nous optons pour SMOTE pour les ré-équilibrer. En utilisant le même modèle pour l'entraînement, nous avons alors une exactitude de 68% sur les données de test.

5.3 Deuxième amélioration : Prétraitements

Pour aider notre réseau à classifier les articles, nous cherchons à tester certains prétraitements que nous avions essayés pour nos modèles traditionnels de classification. Nous les appliquons avant la tokenisation, comme nous les appliquions avant la vectorisation pour l'approche *Bag Of Words*. Voici les prétraitements testés et leurs effets :

1. Suppression des *stop-words* et Stemming : ensemble ou séparément, ils font systématiquement baisser le score d'exactitude sur les données de test.
2. Suppression des nombres et de la ponctuation : ensemble ou séparément, ils font systématiquement augmenter le score d'exactitude vers environ 70%.

Notre interprétation de ces résultats est qu'en supprimant le bruit de la ponctuation et des chiffres, nous rendons le signal du texte plus lisible pour le classificateur, là où nous perdons de l'information quand nous appliquons le Stemming ou la suppression des stop-words.
L'idée est que, le LSTM prenant en compte l'ordre dans les séquences, les stop-words et la forme des mots peuvent toujours ajouter de l'information. Cependant, nous pouvons nuancer cette remarque en disant qu'avec notre choix de n'autoriser un vocabulaire que de 500 tokens, nous avons déjà dénaturé les phrases. Par exemple, voici un article obtenu en retrouvant les mots à partir des index de tokens après tokenisation :

states are with for unknown media department of in states

[6] https://aclanthology.org/I17-2043.pdf
[7] Référence sur les LSTM : https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (consulté le 20/05/2022)

Il est donc possible que le *Stemming* et la suppression des *stop-words* casse la continuité des phrases, mais il est possible qu'elle soit déjà cassée par la tokenisation et que la perte d'exactitude soit due à autre chose.

Il est intéressant alors de se demander l'impact de la taille du vocabulaire du *Tokenizer* sur nos scores. Dans la Figure 5, nous prenons le modèle obtenu à la fin de cette partie, et calculons son exactitude sur les données de test en ayant été entraînées sur différentes tailles de vocabulaire (équivalent au nombre de *tokens*). Nous voyons que ce paramètre n'a aucun effet sur nos résultats.

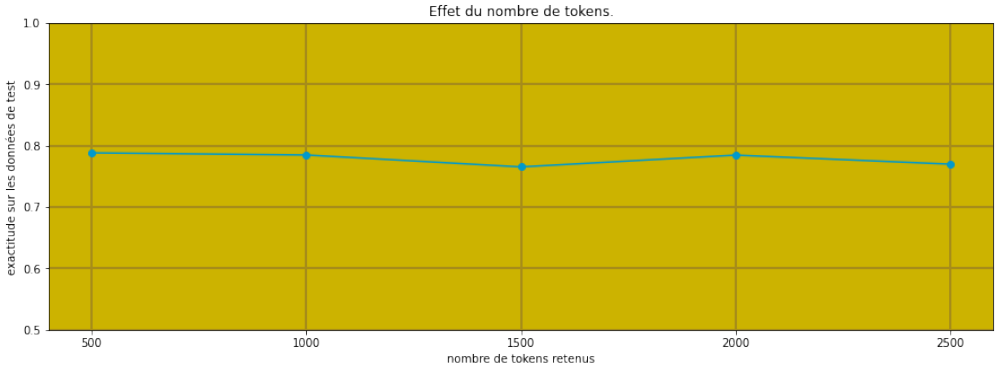


Figure 5: Effet de la taille du dictionnaire du Tokeniser sur le score d'exactitude obtenu.

5.4 Troisième amélioration : Ajouter des données

Nous disposons d'autres données que juste les textes des articles. Comme pour les classificateurs traditionnels, nous pouvons essayer de les mettre à notre service. Pour ce faire, avant la tokenisation, nous concaténons le texte d'une colonne que nous voulons prendre en compte, à celui du champ *text*. Voici les colonnes que nous avons essayé de prendre en compte, et leur effet sur nos scores :

1. Auteurs : L'ajouter ne change pas les scores.
2. Headlines : L'ajouter monte les scores d'exactitude à une moyenne de 72.6% après 3-validation.
3. Keywords : L'ajouter monte les scores d'exactitude à une moyenne de 73.5% après 3-validation.
4. Source : L'ajouter monte les scores d'exactitude à une moyenne de 73.0% après 3-validation.
5. Headlines, Keywords, Source, Auteurs : En ajoutant toutes ces colonnes à notre texte, nous obtenons un score d'exactitude de 77.8% après 3-validation.

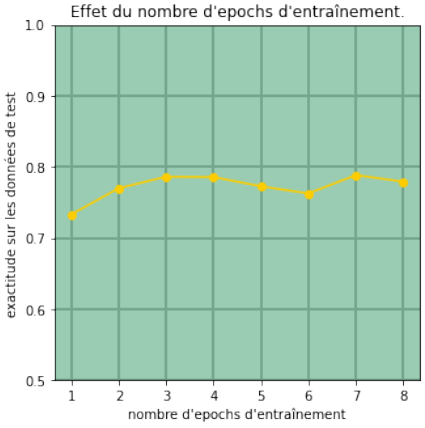


Figure 6: Effet du nombre d'*epochs* d'entraînement sur le score d'exactitude obtenu.

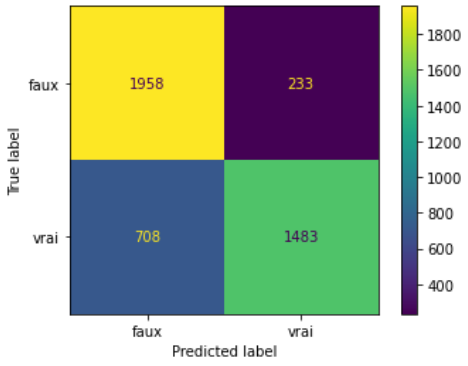


Figure 7: Matrice de Confusion obtenue en faisant une classification avec notre modèle

En mettant en relation les nouvelles informations ajoutées, le LSTM a pu faire des déductions qu'il ne pouvait pas faire quand on ne les lui donnait qu'une à la fois. Cela explique qu'avec toutes les colonnes, on ait un meilleur score qu'avec chacune d'entre elles séparément.
Maintenant que nous disposons d'une méthode satisfaisante, nous pouvons chercher à savoir quand arrêter l'entraînement de notre LSTM. En effet, si nous l'entraînons excessivement longtemps, nous risquons l'*overfitting*. Sur la Figure 6, nous étudions ce problème.

À partir de 2-3 *epochs*, l'exactitude ne monte plus. Nous pensons qu'avec l'architecture utilisée et les prétraitements réalisés, nous avons atteint un bon minimum local de la fonction de coût. Nous choisissons donc d'arrêter l'apprentissage à ce moment.
Par ailleurs, nous avons tenté d'ajouter une seconde couche LSTM au réseau, sans remarquer d'effet. Sûrement l'architecture actuelle est-elle suffisamment complexe pour le problème tel qu'il est posé.
Nous avons essayé un certain nombre d'autres choses, comme retirer les majuscules, sans remarquer d'effet. Nous omettons ces tentatives ici.

Bilan sur l'approche *Deep Learning* :

Nous avons pu obtenir environ 78% d'exactitude avec un réseau LSTM, en sur-échantillonnant la classe des vrais avec SMOTE, en supprimant les nombres et la ponctuation, et en ajoutant les colonnes Keywords, Headlines, Authors, et Source au texte. En Figure 7, on peut trouver une Matrice de Confusion obtenue en faisant une classification avec notre modèle.

6 Conclusion

Au cours de ce projet, nous avons étudié le problème de la classification de *fake news*. Ne disposant pas d'un contexte général sur les *news*, nous avons développé un ensemble de méthodes basées sur le style de l'article plutôt que sur des vérifications de faits ; celles-ci de deux catégories : modèles traditionnels de classification, et modèles de *Deep Learning*. Le succès de chacun de nos modèles reposait sur la qualité des prétraitements réalisés sur les données.
Parmi les méthodes traditionnelles , à l'aide de *gridsearch*, nous avons établi un certain nombre de combinaisons entre prétraitements et modèles permettant une exactitude de 65%. Les classificateurs SVM et LR se sont notamment démarqués sur ce point. Dans le *Deep Learning*, nous avons conçu un modèle qui, avec des prétraitements donnés, augmentait l'exactitude jusqu'à 78%.

Un travail n'étant jamais fini, il est possible d'améliorer nos résultats. La piste la plus évidente serait de tester un plus grand nombre de prétraitements différents. Si nous pensons avoir obtenu une vision d'ensemble sur l'efficacité des classificateurs traditionnels sur ce problème, nous n'excluons pas qu'une combinaison de paramètres ou un prétraitement particulier pourraient dépasser les scores que nous avons obtenus. Du côté du *Deep Learning*, au delà des prétraitements, des meilleurs résultats pourraient être obtenus en utilisant d'autres architectures : par exemple, des Réseaux Temporels Convolutionnels[8]. Il serait aussi possible d'appliquer la méthode proposée dans l'article mentionné dans la partie 5, où des modules d'attention plus poussés sont utilisés.

[8] http://sersc.org/journals/index.php/IJAST/article/view/19574