Fake News Classification

Hands On AI 2

CORDUANT Valentin (191383) & VANSNICK Tanguy (191451)

« Dans le cadre de notre projet Hands On AI 2 en traitement du langage naturel, nous avons mis en place un système de classification de *Fake News* françaises. Ce projet vise à aider les utilisateurs à mieux comprendre l'origine et la fiabilité de l'information qui leur est présentée, notamment dans le contexte actuel où la désinformation est devenue un enjeu majeur sur les réseaux sociaux et dans les médias en général.

Pour atteindre cet objectif, nous avons utilisé des techniques avancées d'apprentissage automatique, en particulier des modèles de traitement du langage naturel basés sur du machine learning : LSTM et Transformers. Nous avons entraîné ces modèles sur un grand corpus de données.

Le résultat de notre projet est un ensemble de modèles de classification performant capables de prédire avec une grande précision si une News est vraie ou fausse. Dans les sections suivantes, nous décrirons en détail les différentes étapes de notre projet, ainsi que les résultats obtenus et les enseignements tirés de cette expérience. » *made with chatGPT.*

GitHub du projet : https://github.com/ValentinCord/HandsOnAI_2
One Drive du projet avec codes, données et modèles : https://shorturl.at/zDT49

1. Guide d'utilisation

Durant ce projet, nous avons créé plusieurs notebooks qui répondent chacun à une tâche précise :

- « **NLP_DataAnalysis** » : analyse et visualisation les données.

- « **NLP_Adding_data** » : récupération et sauvegarde de nos nouvelles données d'entraînement. Pour que celui-ci soit utilisable, il faut avoir un compte Kaggle ainsi que le token de l'API publique. Les données finales sont déjà disponible dans un fichier csv.

- Méthode 1 – LSTM :
  - « **LSTM_Training** » : preprocessing et entraînement du modèle de type LSTM.
  - « **LSTM_Testing** » : évaluation de notre modèle de type LSTM. En fonction d'où le modèle, le tokenizer et les données sont enregistrés, le notebook doit être adapté.

- Méthode 2 – Transformer :
  - « **Transformer_Training** » : preprocessing et entraînement du modèle de type Transformer.
  - « **Transformer_Testing** » : évaluation de notre modèle de type Transformer. En fonction d'où le modèle et les données sont enregistrés, le notebook doit être adapté.

2. Analyse des données

Le jeu de données contient 1458 éléments. Parmi ces éléments, on retrouve 816 *Real News* et 642 *Fake News*. En fonction du type de News, nous pouvons comparer leurs différences en termes de tailles et de champs lexicaux.

Longueur des News : en analysant la longueur des News, on constate que les *Real News* ont tendance à être plus longue que les *Fake News*.

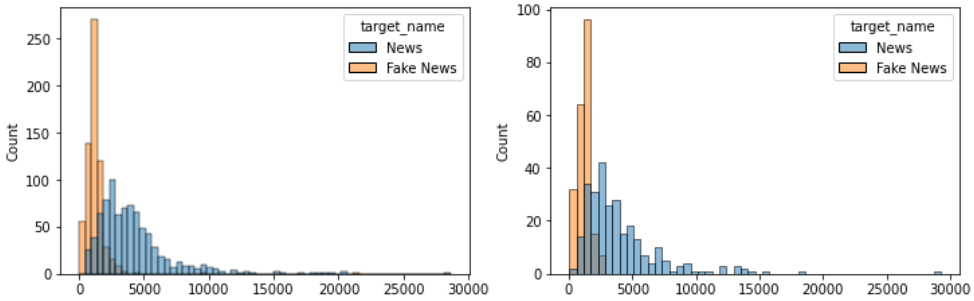

Figure 1 : Longueur des données d'entraînement (gauche) et de test (droite)

Si on applique un logarithme sur ces résultats, ceux-ci suivent une loi normale. Ces résultats sont intéressants à garder en tête lors de l'entraînement des modèles.

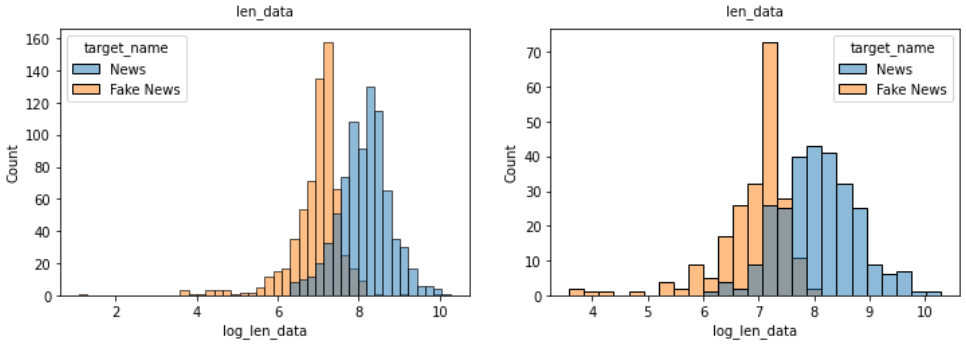

Figure 2 : Logarithme des longueurs des données d'entraînement (gauche) et de test (droite)

Taille des mots : nous avons également analysé la taille moyenne des mots dans chaque News. On constate qu'il y a des cas critiques parmi les *Fake News*. En effet, on retrouve une *Fake News* contenant en moyenne des mots de 100 lettres. Il existe aussi une *Fake News* avec un autre alphabet. Une fois ces caractères spéciaux enlevés, la News possède une longueur moyenne de 3 lettres.

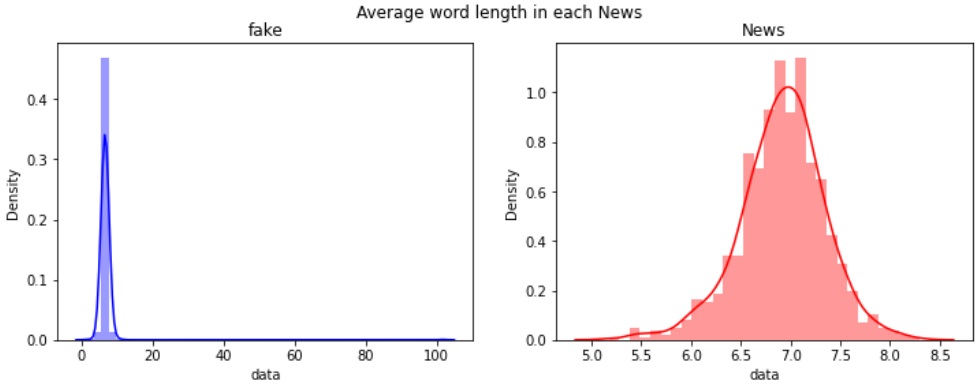

Figure 3 : Taille des mots dans les fake news (gauche) et les real news (droite)

À la suite de la suppression de ces deux éléments, la courbe de distribution des *Fake News* se rapproche de celle des *Real News*. Bien que l'on constate toujours des points extrêmes chez les *Fake News*, il est tout à fait possible d'avoir de telles distributions.

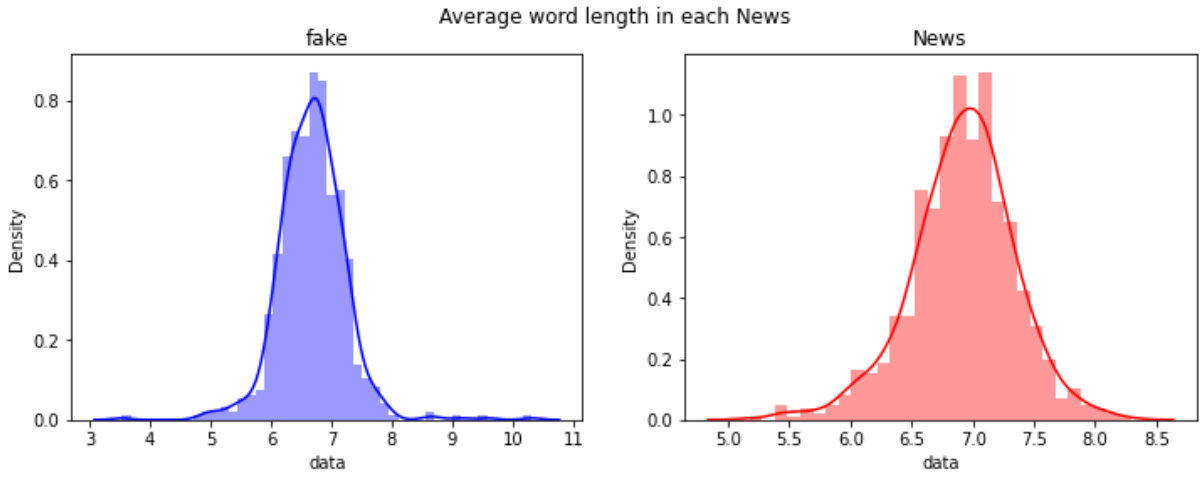

Figure 4 : Taille des mots dans les fake news (gauche) et les real news (droite) après preprocessing

Mots les plus fréquents : nous avons comparé quels étaient les mots les plus fréquemment utilisés dans les *Real News* et les *Fake News*. Dans un premier temps, nous avons effectué cette comparaison à partir des données déjà nettoyées. Constatant qu'il y avait une grande quantité de « stop words », nous les avons supprimés.

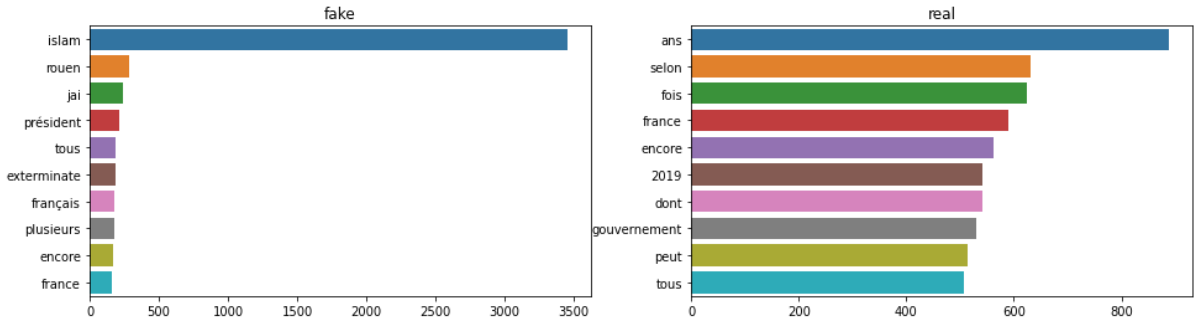

Figure 5 : Mots les plus fréquemment utilisés dans les fake news (gauche) et les real news (droite)

La plupart des mots semblent normaux. Cependant, nous avons vérifié que ceux-ci apparaissent bien dans plusieurs News et non pas dans une seule. Par la suite, nous avons rencontré deux *Fake News* ayant un seul mot écrit à répétition. Cela était notamment le cas pour le mot « islam » et le mots « exterminate » qui faisait référence à un personnage de la série Doctor Who. Nous avons décidé de supprimer ces données du dataset.

3. Ajout de données

Pour enrichir le jeu de données, nous avons trouvé des données sur Kaggle. Celle-ci étant en anglais, nous avons dû les traduire en français. Pour ce faire, nous avons utilisé la libraire « GoogleTrans » . Le nouveau jeu de données étant d'origine américaine, il se peut qu'il diverge de notre jeu de données d'origine. Il faut donc faire attention à ne pas ajouter trop de nouvelles données afin de ne pas trop perturber le modèle.

Source de nouvelles données : https://www.kaggle.com/competitions/fake-news

4. Modèle : LSTM

Comme premier modèle, nous avons décidé de faire une architecture de type LSTM. Dans le notebook, on retrouve toutes les étapes menant à la création du premier modèle. Comme paramètre, il est possible de définir :

- Le nombre de données à ajouter ;
- La taille des séquences ;
- Le batch size ;
- Le nombre d'epochs ;

Preprocessing : une fois les paramètres définis, nous pouvons passer à l'étape de preprocessing. Dans cette étape, nous effectuons un nettoyage de données et une tokenisation. Les données aberrantes rencontrées lors de l'analyse de données ont été supprimées. Les caractères spéciaux ainsi que les « stop words » issue de la libraire « nltk » ont également été enlevés.

Une fois les données nettoyées, on applique la tokenisation. On applique la fonction « fit » du tokenizer sur les données d'entraînement. Pour les données de validation et de tests, c'est la fonction « fit_transform » qui est appliquée. Lorsque les données sont converties sous le format de token, un padding est appliqués sur toutes les données.

Création du modèle : le modèle créé est une architecture de type LSTM. En amont, nous avons appliqué un CNN afin d'extraire des caractéristiques. Après la couche CNN, nous avons ajouté deux couches LSTM bidirectionnelles. Avant la classification, une couche « Dense » a également été ajoutée.

```
model = Sequential()
model.add(Embedding(input_dim = (len(tokenizer.word_counts) + 1),
output_dim = 128, input_length = MAX_SEQ_LEN))
model.add(Conv1D(32,3, padding='same'))
model.add(MaxPooling1D())
model.add(Bidirectional(LSTM(100, return_sequences=True)))
model.add(Bidirectional(LSTM(100)))
model.add(Dropout(0.4))
model.add(Dense(128, activation='relu'))
model.add(Dense(2, activation='softmax'))
```

Entraînement et sauvegarde : pour terminer, le modèle construit est entraîné en fonction du nombre d'epochs qui a été défini précédemment. On constate que ce modèle overfit très rapidement. Avec trois epochs, on atteint déjà de très bons résultats.

Une fois le modèle entraîné, nous pouvons le sauvegarder ainsi que le tokenizer. Le tokenizer est enregistré sous format pickle alors que le modèle est dans un fichier « .h5 ».

5. Modèle : Transformer

Comme second modèle, nous avons décidé de créer une architecture de type Transformer. C'est le modèle « cmarkea/distilcamembert-base » issue de la librairie HuggingFace qui a été utilisé. Le modèle étant de type « distil » et « base », celui-ci est de très petite taille par rapport à la taille d'un Transformer standard.

Nous avons le droit à plus de paramètres que pour le modèle de type LSTM. Ces nouveaux paramètres sont propre à la structure des données dont nous avons choisi ainsi qu'à l'architecture du modèle.

Preprocessing : comme pour le premier modèle, nous avons commencé notre étape de preprocessing par un nettoyage des données. Les News aberrantes ont été supprimées. Les « stop words » ainsi que les caractères spéciaux ont également été éliminées.

Contrairement aux LSTM, les Transformers ont un input d'une longueur bien définie. Dans notre cas, cette longueur est de 512 tokens. Généralement, si le texte est trop long ou trop court, un padding ou une troncation est appliqué.

Comme nous ne voulions pas appliquer de troncation, nous avons décidé de couper certaines News en plusieurs morceaux. Cette approche permet de garder un maximum d'informations. Nous avons donc créé un paramètre permettant de définir une taille maximale pour chaque morceau de News.

Pour garder un certain contexte entre chaque morceau ceux-ci ne sont pas collés les uns à la suite des autres. Les morceaux adjacents ont un léger overlap qui est aussi paramétrable.

Une fois que le DataFrame a été correctement reformulé, nous l'avons mis sous la forme de Dataset. Le tokenizer a été appliqué sur les données. Un padding a été appliqué. Des tokens spéciaux ont également été ajoutés.

Un Dataloader a été créé afin de pouvoir charger les données durant l'entraînement. À partir de ce Dataloader, il est possible de mélanger les données d'entraînement.

Création du modèle : pour la création du modèle, nous avons créé la classe « BertClass ». De cette manière, il est possible d'ajouter de nouvelles couches au modèle. Concernant la Loss, nous avons opté pour la « CrossEntropy » avec l'optimizer Adam.

Entraînement et sauvegarde du modèle : suite à la création du modèle, celui-ci est entraîné en fonction du nombre d'epochs demandé. Ensuite, le modèle entraîné est évalué deux fois.

La première fois, nous évaluons les prédictions par rapport à chaque morceau de News que nous avons créé. Cette première évaluation permet d'avoir une idée globale de la qualité du modèle. Cependant, nous voudrions avoir une prédiction globale pour chaque News et non sur des morceaux.

Pour l'évaluation finale, nous devons regrouper toutes les prédictions de chaque News. Si tous les morceaux de la News ont fait une même prédiction, alors la News aura la même prédiction. Cependant, lorsque tous les morceaux n'ont pas une prédiction identique, on opte pour le label ayant été prédit le plus. Si on tombe dans le cas où le nombre de prédictions de *Fake News* et de *Real News* est identique, on prédit que c'est une *Real News*.

6. Conclusion

Nos deux modèles montrent des résultats assez encourageants, ils se complètent très bien. Le premier est plus simple et a uniquement besoin d'un entraînement rapide. Quant au second modèle, celui-ci est un peu plus performant mais, requiert un plus grand stockage et meilleure puissance de calcul.

Modèle	**Précision**
LSTM	0.9568
TRANSFORMER par morceau	0.96
TRANSFORMER complet	0.9485

On voit que les résultats obtenus avec le jeu de données de test sont au-dessus des 95%. La loss est relativement faible. L'ajout de nouvelles données a permis au modèle de mieux généraliser. En espérant que les modèles puissent bien classifier les données de test cachées.