Projet de NLP : Classification de questions inappropriées sur Quora

Joachim Dublineau, Samuel Montini

Février 2020

1 Introduction

Quora est une entreprise gérant un forum en ligne du même nom. Sur ce forum, les utilisateurs sont libres de poster des questions auxquels d'autres utilisateurs pourront répondre. Comme pour tous les forums, il arrive que certaines questions ou commentaires soient inappropriés. Tout l'enjeu est alors d'avoir un outil de détection efficace pour pouvoir faire le tri sans avoir à attendre les éventuels retours des utilisateurs.

Ainsi Quora a publié sur Kaggle il y a un peu plus d'un an un challenge qui vise à développer un outil permettant de dire si une question est inappropriée ou non. Ce challenge s'accompagne d'un jeu de données important mis à disposition des compétiteurs.

1.1 Jeu de données

Les données mises à disposition se présentent sous la forme d'un ensemble d'entraînement (Figure 1) et d'un ensemble de test (Figure 2). On remarquera que les données de test, en pratique, ne servent qu'à Quora pour la compétition puisque les questions ne sont pas "labelisées". Il est donc impossible pour nous de l'utiliser en ensemble de test. Nous nous limiterons donc à l'ensemble d'entraînement.

Pour le projet, nous avons donc uniquement utilisé le fichier *train* que nous avons divisé en deux parties : un nouvel ensemble d'entraînement (80%) et un ensemble de test (20%). Ensuite, à partir de cet ensemble d'entraînement, nous avons crée un ensemble d'entraînement bis dans lequel nous avons rééqulibré la proportion de mauvaises et de bonnes questions en faisant de l'*undersampling*. Nous avons également fait une version sans *undersampling* pour comparer.

```
Train data:
                         qid  ... target
0        00002165364db923c7e6  ...      0
1        000032939017120e6e44  ...      0
2        0000412ca6e4628ce2cf  ...      0
3        000042bf85aa498cd78e  ...      0
4        0000455dfa3e01eae3af  ...      0
...                       ...  ...    ...
1306117  ffffcc4e2331aaf1e41e  ...      0
1306118  ffffd431801e5a2f4861  ...      0
1306119  ffffd48fb36b63db010c  ...      0
1306120  ffffec519fa37cf60c78  ...      0
1306121  ffffed09fedb5088744a  ...      0

[1306122 rows x 3 columns]
```

FIGURE 1 – Ensemble d'entrainement, 3 colonnes (qid, text, classe)

```
Test data:
                       qid                                      question_text
0       0000163e3ea7c7a74cd7  Why do so many women become so rude and arroga...
1       00002bd4fb5d505b9161  When should I apply for RV college of engineer...
2       00007756b4a147d2b0b3  What is it really like to be a nurse practitio...
3       000086e4b7e1c7146103                          Who are entrepreneurs?
4       0000c4c3fbe8785a3090   Is education really making good people nowadays?
...                      ...                                              ...
375801  ffff7fa746bd6d6197a9  How many countries listed in gold import in in...
375802  ffffa1be31c43046ab6b  Is there an alternative to dresses on formal p...
375803  ffffae173b6ca6bfa563  Where I can find best friendship quotes in Tel...
375804  fffbb1f7f1a008620287        What are the causes of refraction of light?
375805  fffff85473f4699474b0  Climate change is a worrying topic. How much t...

[375806 rows x 2 columns]
```

FIGURE 2 – Ensemble de test, 2 colonnes (qid, text)

1.2 Evaluation des performances du modèle

Quora imposait pour ce projet, que les performances des différents modèles soient évaluées par le F1-score. Le F1-score est défini de la manière suivante :

$$f1 = \frac{2}{recall^{-1} + precision^{-1}}$$

Pour cette compétition, les scores varient entre 0.71 pour les meilleurs groupes et autour de 0.30 pour les moins bons. Nous essaierons lors de la présentation des résultats de présenter le classement que l'on aurait obtenu avec le modèle étant donné son f1 score. Cependant, si besoin, le classement est consultable ici : `https://www.kaggle.com/c/quora-insincere-questions-classification/`

`leaderboard`.

2 Traitement des données

Avant toute chose, il est important de définir la méthode de pré-traitement de données afin de pouvoir rendre les questions exploitables par un réseau de neurones. Pour cela, nous allons d'abord utiliser une fonction visant à transformer le texte, puis nous allons transformer la phrase en séquence d'*embeddings*.

2.1 Normalisation de texte

En jetant un premier regard à nos données, nous pouvons observer plusieurs problèmes. Tout d'abord, les phrases que nous analysons peuvent être mal écrites en contenant des fautes d'orthographes (assez difficiles à contrer). Les caractères spéciaux sont souvent collés aux mots qui les précèdent. De plus, l'anglais contient de nombreuses contractions de mots du type *aren't* qu'il peut être intéressant de décomposer. Enfin, on peut également avoir des problèmes avec les majuscules. Nous avons ainsi développé une fonction *normalize_text(text, unusual_characters, dict, no_figure = True)* qui va normaliser le texte placé en argument. Voici la description de la fonction :

"Transforms the text string by removing every unusual character and by lowering the characters that are located inside a word.
INPUT :
- text : string corresponding to the text that we want to normalize.
- unusual_characters : is a list of characters that we want to remove from the text.
- dict_ : dictionary containing the string that needs to be replaced by another string.
- no_figure : boolean saying if we want numbers to appear in the resulting text or not.
OUTPUT :
- normalized_text : string of the normalized word.
RUNNING TIME :
Immediate "

Voici les résultats que l'on peut obtenir avec cette fonction :
Origin text : ' The sentence Charles said is cumplex, with mistakes and unUsual characters like 24/7. Sometimes, it can even be 2 sentences mentioning money or €. But normalizing it, it's doable. '
Normalized text : ' the sentence Charles said is cumplex , with mistakes and unusual characters like number / number . sometimes , it can even be number sentences mentioning money or euros . but normalizing it , it is doable . '

Maintenant que nous avons vu la normalisation de texte, intéressons-nous aux *embeddings*.

2.2 Les *embeddings*

Pour ce projet, Quora met en lien plusieurs dictionnaires d'*embeddings* déjà pré-entrainés tel que Word-2-vec, Wiki-news... Pourquoi utiliser ces dictionnaires plutôt que de mettre une couche d'*embeddings* dans notre réseau et de l'entraîner ? La raison est que nous avons ici à faire à des données très larges et donc avec un *vocabulary_size* très grand. L'entraînement risque d'être long et fastidueux et avec un résultat potentiellement médiocre. Le fait d'utiliser des *embeddings* pré-entraînés est plus sûr et potentiellement plus efficace. Cependant, il pourrait être intéressant également de faire du *fine-tuning*.

En revanche, nous avons fait le choix de ne pas utiliser de *stemming* ni de *stopword* parce que nous pensions que cela nous pourrait nous faire perdre de précieuses informations. En effet, ces méthodes sont principalement utiles pour alléger le contenu du texte en vue de faire un entrainement des *embeddings*.

Parmis les différents *embeddings* proposés, il y avait Wikinews, GoogleNews, Paragram et GloVe. Nous avons testé les performances de ces différents *embeddings* avec le premier modèle décrit plus bas. Nous avons obtenus les résultats suivants, I.V. signifiant "[words] in vocabulary" :

Noms *Embeddings*	Pourcentage de I.V. (%)	F1 score max
GoogleNews	80	0.5375
WikiNews	98.47	0.5000
GloVe	99	0.5375

À partir de ces résultats, nous avons décidé d'utiliser *GloVe*. Nous avons donc tout d'abord créé une fonction qui ouvre le fichier et en extrait, ligne par ligne, les différents vecteurs qu'il contient pour les mettre dans un dictionnaire python. Afin d'éviter d'avoir à faire cette opération très coûteuse à chaque fois, on sauvegarde le dictionnaire avec numpy. Nous nous sommes rendus compte plus tard que des méthodes pour faire cela existaient déjà...

Avec ce dictionnaire, nous pouvons étudier d'un peu plus près nos données. Ainsi nous sommes en mesure de dire qu'il y a à peu près 99% des mots du data set qui sont dans le dictionnaire (comme montré dans le tableau plus haut), que la taille moyenne des phrases est de 15 mots. On a également regardé quels sont les mots les plus fréquents et quelle est leur fréquence d'apparition. Le résultat est en bonne qualité au lien suivant (`https://drive.google.com/open?id=1_73230RZT6eYZElfwyloucoNs9wRaUyC`), et le graphique est en Figure 3

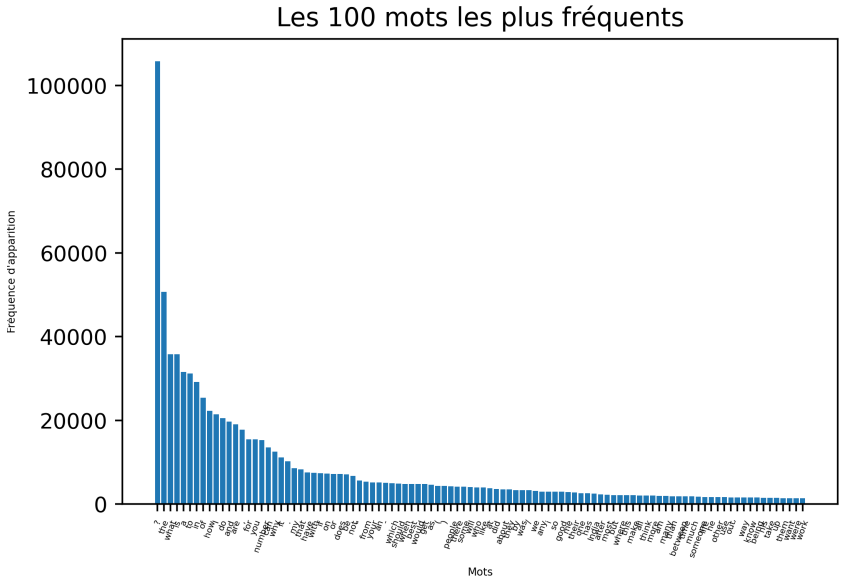

FIGURE 3 – Apparition des 100 mots les plus fréquents

2.3 Les questions comme séquences d'*embeddings*

Nous avons une fonction *process_data(data, dictionary, nb_words_per_sentence)* qui va prendre en entrée un ensemble de chaînes de caractères représentant les questions que nous voulons classifier et va en sortir un tableau de sequences de vecteurs tirés du dictionnaire évoqué précédemment (pour les mots qui y seraient présents), ou bien calculés comme moyenne des vecteurs des mots "voisins" (pour ceux qui n'y serait pas).

Cette fonction fait donc appel d'abord à la fonction *normalize_text* puis elle va rechercher les vecteurs correspondant aux mots qui sont dans le dictionnaire et utiliser une autre fonction *create_contextual_embedding* pour ceux qui n'y seraient pas. Cette fonction va à son tour chercher parmi les mots voisins du mot inconnu, quels sont les vecteurs correspondants, en faire la moyenne et attribuer le vecteur résultant au mot inconnu. Le fait de moyenner évite d'introduire des valeurs irrégulières dans notre liste d'*embeddings* comme le vecteur nul.

Voici le descriptif de la fonction *create_contextual_embedding* :

" Computes the embedding vector for a word depending on the context. The context is defined by the number of neighbors in the sentence. If one of the words in the work list is not a part of dictionary.keys, we ignore this word.

INPUTS :

- word_index : is the index of the word in the sentence.

- sentence : is a list of string, each string is a word.
- dictionary : is the dictionary associating embedding to each word.
- common_words : list of usual words that we will not take into account because they might pollute our final embedding
- nb_neighbors : is an int that defines the number of neighbors that we will take into account to compute the output.

OUTPUTS :

- embedding_vector : is an array of size 300 corresponding to the estimated embedding vector.

RUNNING TIME :

Very short "

2.4 La fonction d'entrainement et la fonction de test

L'ensemble d'entraînement et l'ensemble de test, une fois vectorisés, ne rentrent plus en mémoire. On est donc obligés de charger les données progressivement. Nous avons donc réunis en une seule fonction, le préprocessing des données, la vectorisation des mots et l'entraînement.

Nous effectuons un test complet à la fin de chaque *epoch* et nous sauvegardons les poids correspondant à l'*epoch* afin de nous assurer de bien avoir le meilleur modèle à la fin de l'entrainement.

Ce genre de fonctions existe déjà en PyTorch, mais comme cela n'est pas très difficile à coder, nous avons choisi de faire cela nous-mêmes. De plus, nous nous sommes rendus compte *a posteriori*, que refaire les étapes de *preprocessing* pour le nouvel échantilon, prenait moins de temps que de le charger en mémoire avec un *numpy.load*. Sans plus attendre, présentons les réseaux que nous avons étudiés et leur performance.

3 Les réseaux

En résumé, nous avons donc maintenant transformé notre ensemble de questions en ensemble de vecteurs exploitables par un réseau avec deux hyperparamètres à régler : la taille maximale des questions, la taille du voisinage pour la création des *embeddings* contextuels pour les OOV. Maintenant, présentons les réseaux que nous avons étudiés.

3.1 Tests sur quelques réseaux récurrents

Nous avons d'abord essayé de comparer l'efficacité des RNN et des LSTM sur ce problème. Nous avons pris 2 modèles similaires avec pour seule différence le type de cellule récurrente utilisée. Nous avons donc comparé un modèle RNN monodirectionel suivi par deux couches denses, récupérant la sortie de la dernière entrée et le même modèle mais avec un RNN à la place. Pour ces deux modèles

nous avons obtenu au bout de 4 époques un meilleur F1 score pour le modèle avec RNN ainsi qu'un meilleur temps de calcul. Nous avons donc décidé de nous appuyer sur des cellules RNN pour la suite.

À partir de là, nous avons designé un premier modèle (cf. Figure 4) alliant des cellules RNN et des cellules convolutives avec plusieurs couches denses. Ce réseau donne les résultats suivants :

— au bout de 5 epochs :
"acc avg : 0.871, acc 0 : 0.867, acc 1 : 0.933, recall : 0.933, precision : 0.315,
F1 Score : 0.4707"
— au bout de 10 epochs :
"acc avg : 0.901, acc 0 : 0.901, acc 1 : 0.895, recall : 0.895, precision : 0.397, F1 Score : 0.5499"

Avec données réequilibrées A la fin de ces 10 epochs, un test sur 50 000 phrases a donné les résultats suivants : acc avg : 0.901, acc 0 : 0.902, acc 1 : 0.883, recall : 0.883, precision : 0.374, F1 Score : 0.52515

Sans réequilibrage des données En faisant le même entraînement mais sans effectuer de rééquilibrage des données, nous obtenons les résultats de test suivants : acc avg : 0.956, acc 0 : 0.981, acc 1 : 0.567, recall : 0.567, precision : 0.661 F1 Score : 0.610. Conclusion : ces résultats sont meilleurs qu'avec rééquilibrage des données mais l'entraînement est plus long car il nous faut plus de données en tout pour avoir suffisament de 1.

3.2 *Finetuning* d'un modèle type BERT

Il nous semblait également intéressant de tester les performances d'un modèle comme BERT, qui prend en compte des informations syntaxiques intéressantes en s'intéressant aux racines des mots, appelées *token*. Nous avons donc *finetuné* un modèle BERT importé grâce à la librairie d'*Hugging Face*.

Avec réequilibrage des données Dès la première epoch, les résultats sont satifaisants. L'entraînement est cependant très long car le modèle est très lourd. Au bout de 2 *epochs*, la *loss* de validation commence à remonter et nous obtenons les résultats suivants : acc : 0.9383, F1 Score : 0.6416

Le F1 Score est bien meilleur que pour le modèle RNN tout simple, mais c'est au prix d'un modèle beaucoup plus coûteux en temps de calcul. Avec de tels résultats, notre modèle se classe 1200 dans la compétition.

Sans réequilibrage des données Tout d'abord, sans rééquilibrage des données, l'entraînement est plus long car nous devons prendre plus de données pour conserver un nombre important d'exemple de classe 1 : ainsi le modèle prendre

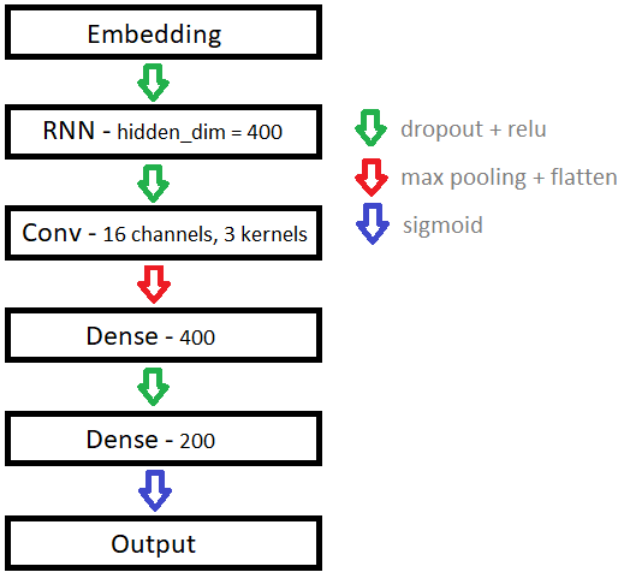

FIGURE 4 – Premier réseau

environ 1h30 par *epoch* contre 30 min précédemment. Les performances sont bonnes puisque dès la première *epoch*, nous dépassons les performances du modèle entraîné sur les données non-équilibrées : acc : 0.960, F1 score : 0.659.

Au bout de l'*epoch* 2, le test donne les résultats suivants : loss : 0.0971, acc : 0.962, 'f1' : 0.677

4 Conclusion

En somme, les modèles que nous avons essayés fonctionnent assez bien. Le BERT donne de bien meilleurs résultats mais au prix d'une complexité plus importante. Le classement indique globalement que la plupart des candidats ont proposé des modèle avec un F1 score entre 0.6 et 0.7. Les meilleurs disent avoir utilisé des techniques plus sophistiquées comme des combinaisons d'*embeddings* différents, des *tokenizers* spéciaux... Nous avons choisi, pour nos premiers modèles, de développer nous-mêmes nos fonctions de préprocessing et de faire un code plus *low-level* afin de bien comprendre ce qui peut se cacher derrière les *tokenizers* et fonctions standards de préprocessing. Cela ne donne pas forcément des performances équivalent à des fonctions de préprocessing toute faite comme on peut en trouver dans des librairies comme SpaCy, mais est assez intéressant à développer.

Toujours est-il que les scores obtenus avec les modèles type RNN, où avec BERT sont encourageants, d'autant que nous n'avons pu les entraîner sur tout le jeu de données pour cause de manque de capacité de calculs.

Concernant le problème du déséquilibre entre nos deux classes dans le jeu de données de départ, nous nous sommes rendu compte que l'*undersampling* n'était pas une bonne solution pour ce problème et dégradait les performances des modèles. Pour résoudre ce problème de déséquilibre, mieux vaut sûrement privilégier le *cost-sensitive learning* !