Classifieur de textes à partir d'un RNN

Arthur Desbiaux et Valentin Cuzin-Rambaud

Abstract Dans ce rapport, nous verrons différentes manières de faire de la classification de texte. Le jeu de données contient des phrases avec une émotion associée. Le but est de retrouver cette émotion.

Introduction

Préparation des données

Le dataset est divisé en trois fichiers : train (16 000 lignes), validation (2 000 lignes) et test (2 000 lignes). Chaque ligne est représentée de la manière suivante : texte;sentiment. Dans un premier temps, nous chercherons à récupérer chaque phrase et chaque sentiment venant de notre dataset. L'idée est la suivante, dans un premier temps, ferons correspondre à chaque mot un id, cela nous permettra d'avoir un vocabulaire qui contiendra donc tous les mots (on aura donc un vocabulaire pour les mots des phrases et un vocabulaire de sentiments). Dans notre implémentation, notre vocabulaire de mots contient donc des mots venant des trois fichiers du dataset pour une taille totale de 17097 mots et 6 émotions. Par la suite, selon le dataset que nous voulons charger, nous allons one hot nos phrases. Pour éviter d'avoir des phrases trop longues, nous avons décidé dans notre fonction de one hot de mettre seulement 10 mots par phrase. Également, pour que chaque phrase possède le même nombre de mots, si une phrase est trop petite, nous la complétons avec un token (<unk>). À la fin, nous possèdons donc une classe "CustomTextDataset" qui nous permet, avec un fichier et les vocabulaires de one hot les phrases et émotions d'un fichier, de récupérer ces phrases et émotions.

Traitement des données

Pour essayer d'améliorer nos résultats, nous avons pu tester différentes optimisations. Tout d'abord, nous avons mis une fonction de loss adaptée au problème, qui est donc

la CrossEntropy.

Une autre amélioration était de retirer les stops words (ex: i, we, theirs, ...), ces mots ne nous permettent pas de prédire une émotion, il est donc plus utile de les retirer. Pour ce faire, nous avons utilisé 'feature_extraction' de scikit-learn.

La dernière amélioration consiste à retirer tous les mots rares (TF-IDF). On remarque que ce sont souvent des mots avec des fautes d'orthographe (ou qui ne veulent rien dire), de toute façon, ils sont trop peu présents pour en tirer des réelles conclusions sur l'émotion de la phrase. Dans notre implémentation, nous retirons les mots qui ont des valeurs de rareté qui sont supérieures au troisième quartile de la distribution des mots.

RNN

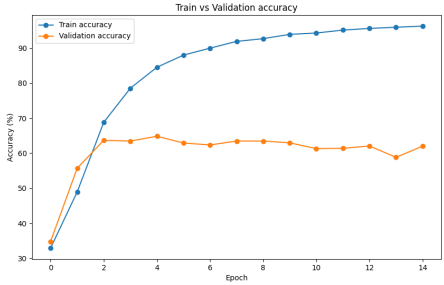
Architecture

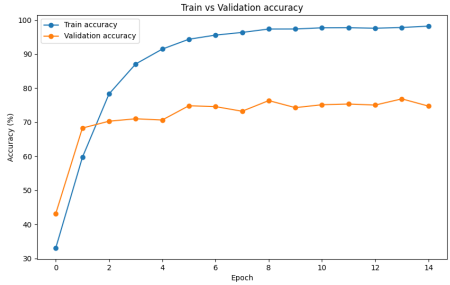
Nous avons quatre couches linéaires dans notre RNN, i2e est une couche d'embedding qui va réduire les dimensions du mot. Nous combinons l'embedding et l'hidden de t-1. On met à jour hidden au temps t, en appliquant tanh à la somme de i2h et h2h. Nous avons i2h qui est une couche linéaire qui prend en entrée la combinaison, h2h qui est une couche linéaire qui prend en entrée hidden t-1. Enfin,h2o est une couche linéaire qui prend hidden à t pour en faire la sortie. Nous avons remarqué que la fonction logsoftmax n'apportait pas de plus-value.

Résultat

Notre meilleur RNN est le suivant :
- batch size: 64
- hidden size: 140
- emb size: 36
- learning rate: 0.001

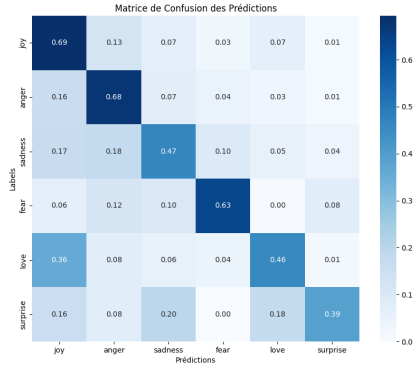
Avec une précision de 62.29% sur le test

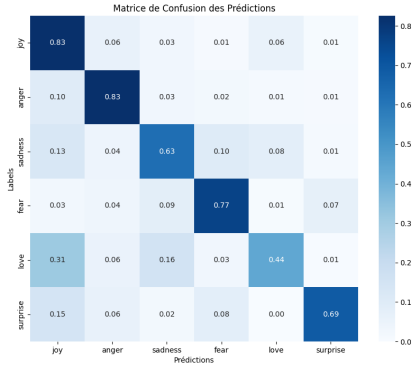




avec stop word sans stop word

FIGURE 1. *courbe d'apprentissage pour RNN*





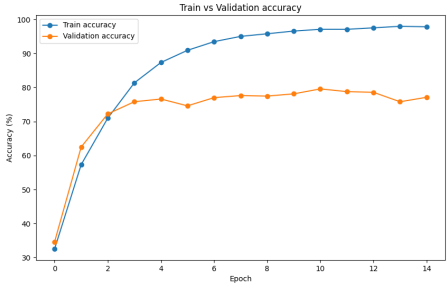
avec stop word sans stop word

FIGURE 2. *Matrice de confusion pour RNN*

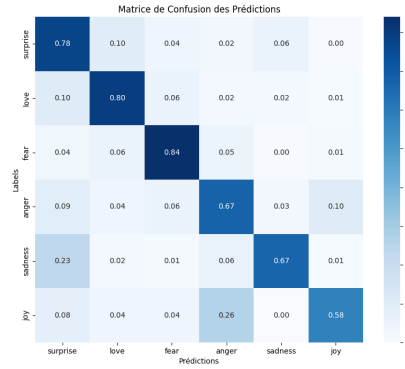
En comparaison, en retirant les stops words, nous avons une précision de 74.79% sur le test.

Comme on peut le voir, après avoir retiré les stops words, l'accuracy est bien meilleur lors du train et notamment de la validation (passage de 65.0% à 73.0%). Ces résultats se remarquent également dans la matrice de confusion.

Maintenant regardons en retirant les mots rares (TF-IDF): Précision de 76.46% sur le test



courbe d'apprentissage sans stop word et sans mot rare



Matrice de confusion sans stop word et sans mot rare

FIGURE 3. *Résultat du meilleur RNN avec optimisation complète*

En retirant les mots rares, on observe une augmentation de 2-3% sur la validation. Pour ce qui est de la matrice de confusion, on prédit mieux la surprise, love et joy mais moins bien pour fear. Cependant, les résultats globaux sont meilleurs, on peut donc affirmer que le traitement a amélioré les résultats.

LSTM

Architecture

Pour implémenter cette architecture, nous nous sommes basés sur https://colah.github.io/posts/2015-08-Understanding-LSTMs/, l'idée est donc la suivante, nous possédons cinq couches, trois couches correspondant à la porte d'oubli, d'input et d'output, nous avons également une couche pour la cell et une couche de sortie. La cell fonctionne comme une mémoire globale, nous la mettons à jour grâce aux différentes portes et nous l'utilisons donc pour calculer la sortie. Entre chaque itération, on récupère la cell et la prédiction précédente. Pour la porte de sortie, on passe dans une couche linéaire avec la concaténation de notre input et dernière prédiction, puis on passe le tout dans une sigmoid. Pour la porte d'input, on répète l'idée précédente, mais en plus, on calcule un C_hat en prenant la concaténation

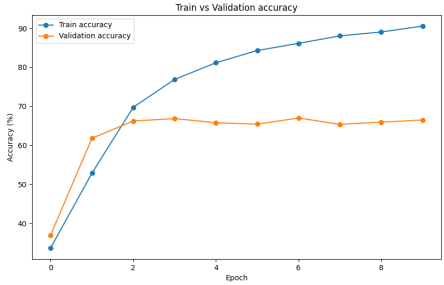
dans une couche linéaire, mais avec une tanh cette fois-ci. Par la suite, nous mettons à jour la cell en multipliant le résultat de la porte d'oubli * la dernière cell + la porte d'input * C_hat. Finalement, pour la porte d'output, nous passons encore une fois la concaténation dans une couche linéaire avec sigmoid puis nous multiplions le résultat par tanh de la cellule.

Résultat

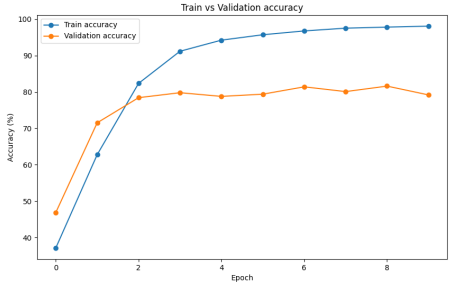
Notre meilleur LSTM est le suivant :

- batch size: 64
- hidden size: 128
- learning rate: 0.001

Avec une précision de 66.58% sur le test. En comparaison, en retirant les stops word, 78.72% sur le test

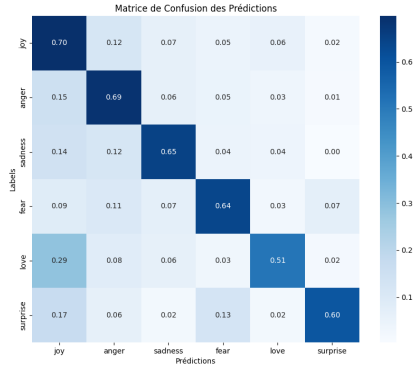


avec stop word

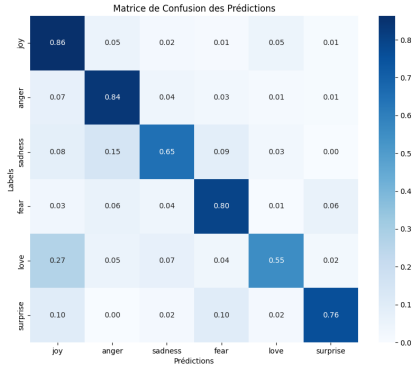


sans stop word

FIGURE 4. *courbe d'apprentissage pour LSTM*



avec stop word

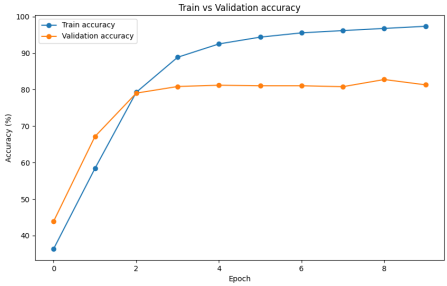


sans stop word

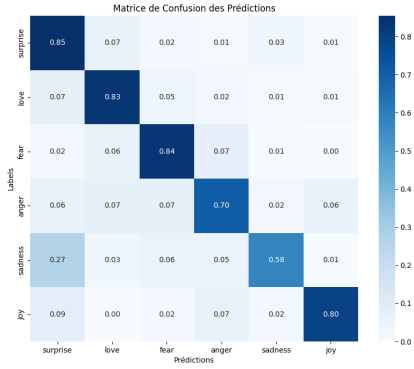
FIGURE 5. *Matrice de confusion pour LSTM*

Comme on peut le voir, après avoir retiré les stops words, l'accuracy est bien meilleur lors du train et notamment de la validation (passage de 66.0% à 78.0%). Ces résultats se remarquent également dans la matrice de confusion.

Maintenant regardons en retirant les mots rares (TF-IDF):



courbe d'apprentissage sans stop word et sans mot rare



Matrice de confusion sans stop word et sans mot rare

FIGURE 6. *Résultat du meilleur LSTM avec optimisation complète*

Nous obtenons une précision de 80.2% sur le test

On peut observer une légère augmentation de l'accuracy sur la validation qui va dépasser les 80% en moyennes. Pour ce qui est de la matrice de confusion, on remarque que l'on prédit bien mieux surprise, love, anger mais petite perte sur sadness. Cependant, les résultats globaux sont meilleurs, on peut donc affirmer que le traitement a amélioré les résultats.

En récapitulatif voici nos accuracy sur le test:

TABLE 1. *Récapitulatif des résultats*

Models	*Accuracy*
RNN basique	62.29%
LSTM basique	66.58%
RNN stop word	74.79%
LSTM stop word	78.72%
RNN stop word + tf-idf	76.46%
LSTM stop word + tf-idf	80.2%

Conclusion

En conclusion, ce projet de classification d'émotions à partir de textes a permis d'implémenter et d'évaluer des architectures RNN et LSTM sur un dataset structuré en phrases associées à des émotions. Les étapes de préparation et de traitement des données, notamment la vectorisation par one-hot encoding, l'élimination des stops words, et le filtrage des mots rares via TF-IDF, ont montré leur importance dans l'amélioration des performances.

Les résultats obtenus démontrent que l'optimisation des modèles en retirant les stops words et les mots rares contribue de manière significative à la précision, avec un bon gain pour le modèle LSTM optimisé, atteignant une précision de 80,2% contre 66,58% pour le modèle LSTM de base. La comparaison entre RNN et LSTM met en évidence la supériorité des LSTM pour ce type de tâches de classification d'émotions, grâce à leur capacité de gestion de contextes plus longs dans les séquences. Des pistes d'amélioration pourraient inclure des essais de modèles encore plus avancés, tels que les architectures de type GRU ou des approches basées sur les Transformers.