Desitter Jimmy

Natural Language Processing :

Projet étude des sentiments

1 SOMMAIRE

I] CONTEXTE DU PROJET

Pour une entreprise, il est important de bien comprendre ses clients. Il n'est pas toujours évident de savoir ce que vos clients pensent de votre entreprise. Aujourd'hui, grâce aux réseaux sociaux, les utilisateurs donnent leur ressenti de façon libre.

Une bonne application est d'utiliser ces données pour prendre la température des réseaux sociaux à propos de votre entreprise. Le but est de savoir si vos clients parlent en bien ou en mal de vos entreprises et de vos produits. Cette température peut ensuite être vue comme un indicateur à améliorer via des campagnes de communication ou une amélioration des offres ou des produits.

Il faut donc entraîner un modèle d'analyse de sentiments. Pour chaque tweet, le modèle doit être capable de déterminer le sentiment de la personne l'ayant écrit.

II] DONNEES ET APPROCHE UTILISEES

Les données utilisées sont des tweets de personnes s'exprimant. Parmi ces données, on y retrouve 800 000 tweets positifs et 800 000 négatifs. On y retrouve de tout, autant des tweets complétement ironiques que des tweets clairs.

Les données utilisées peuvent être retrouvées ici : https://www.kaggle.com/kazanova/sentiment140.

L'approche utilisée était de traiter sous forme de bag of words la data pour ensuite entrainer un modèle de régression logistique et tester le modèle avec une pipeline de traitement et des données test récupérées grâce à l'API Yelp.

Pour commencer, j'ai choisi de prendre 5000 tweets positifs et 5000 négatifs pour faciliter les temps de traitements.

Une fois le modèle de régression logistique réalisé, entrainé et tester le but était de passer sur un modèle de RNN (**réseau de neurones récurrents**).

J’avais pour but d’entrainer plusieurs modèles en changeant par exemple le nombre de couches de layers et essayer de trouver le modèle le plus performant.

III] MES SOLUTIONS

III.1] MODELE DE REGRESSION LOGISTIQUE

Au niveau de la technique d’embedding, j’ai utilisé la technique des bags of words. Avant de transformer mes tweets en vecteur, j’ai dû passer par plusieurs étapes de traitement :

- Mise en minuscule de ma data
- Suppression de la ponctuation et des chiffres
- Tokenization des tweets
- Stemmatization des tweets
- Lemmatization des tweets
- Création d’un set contenant les mots uniques de mon dataset

Enfin j’ai créé des vecteurs pour chaque tweet de la taille du set des mots uniques contenant le nombre de fois qu’un des mots est présent.

Grâce à ces vecteurs et à une map des sentiments pour chaque tweet (1 pour positif et 0 pour négatif), j’ai pu entrainer un modèle de régression logistique. Pour 10 000 tweets positifs et 10 000 tweets négatifs, j’obtiens une précision de 72%.

L’API Yelp ma permis de récupérer des reviews positives, négatives et ambiguës pour tester mon modèle. Finalement les 3 reviews tests ont étaient correctement devinées, mais j’ai remarqué que sur certaines phrases un peu ambiguës ou ironiques, le modèle pouvait se tromper. La précision n’était que de 72% aussi ce qui jouait sur le résultat.

Pour améliorer le modèle on aurait pu par exemple augmenter le nombre de tweets utilisé.

Une fois le modèle de régression logistique terminé, je suis passé sur un modèle de RNN pour essayer de trouver un modèle plus efficace.

III.2] MODELE DE RNN

J’ai décider d’utiliser les fonctions de keras pour Tokenizer mon dataset après avoir enlever la ponctuation et les chiffres et avoir mis en minuscules les tweets.

La classe Tokenizer de Keras est utilisée pour vectoriser un corpus de texte.

Pour cela, chaque entrée de texte est convertie en séquence d'entiers ou en un vecteur qui a un coefficient pour chaque token sous la forme de valeurs binaires.

tokenizer = Tokenizer(num_words=20000, split=" ")

La méthode fit_on_texts fait partie de la classe de tokenizer Keras qui est utilisée pour mettre à jour le vocabulaire interne de la liste de texte.

Nous devons l'appeler avant d'utiliser d'autres méthodes de text_to_sequences ou de text_to_matrix.

tokenizer.fit_on_texts(dataset_lower["text"].values)

La méthode text_to_sequences aide à convertir les tokens du corpus de texte en une séquence d'entiers.

X= tokenizer.texts_to_sequences(dataset_lower["text"].values)

pad_sequence prend une LISTE de séquences comme entrée (liste de liste) et retournera une liste de séquences remplies (toutes de même longueur).

X = pad_sequences(X)

Une fois nos inputs prêts, il faut maintenant faire la structure des modèles.

Pour trouver un modèle efficace, j’ai décidé d’en entrainer plusieurs avec des paramètres et des couches de layers différentes et de voir le résultat.

Mon premier modèle (Modèle_1) était le plus simple des 4, il contenait :

- Une couche d'embedding de 256 layers avec un dropout de 0.3
- Une couche de LTSM de 256 layers avec un dropout de 0.3
- Une couche de Dense de 2 layers avec la fonction d'activation Sigmoid

L'embedding layer transforme les entiers positifs (index) en vecteurs de taille fixe.

La première étape de l'utilisation d'une couche d'incorporation consiste à encoder cette phrase par des indices qui ont des retards très longs.

Les réseaux de longue mémoire à court terme (LSTM) sont une extension pour les réseaux neuronaux récurrents, qui étend leur mémoire. Le dropout permet d'éviter l'overfitting.

Par conséquent, il est bien adapté pour apprendre des expériences importantes qui ont des retards très longs. C'est donc pour ça que j'ai choisis une couche de LTSM avec un dropout de 0.3.

J'ai choisi 256 layers pour la couche d'embedding et de LTSM par pure empirisme (je me suis inspiré des modèles déjà existant).

J'ai choisi une couche de 2 layers en sortie car je souhaitait avoir un résultat du type [x y] avec x la proba que l'input soit positif et y la proba qu'il soit négatif.

En séparant le dataset en 2, 70% pour l'entrainement et le reste pour la validation du modèle, j'obtiens une précision de presque 98% à l'entrainement et de 72% à la validation sur toujours 20 000 tweets.

Cela dit, mon premier modèle s'est finalement montré le plus performant des 4.

Pour ce qui est des 3 autres modèles, j'ai décider de rajouter à mon modèle une couche de layer avec une taille qui varie (64, 128,256) et avec une fonction d'activation relu. Cette fonction converge plus rapidement, optimise et produit la valeur souhaitée plus rapidement. C'est de loin la fonction d'activation la plus populaire utilisée dans les couches cachées.

Voici les résultats de chaque modèle :

- Avec 64 layers : 64,93% de précision sur la validation du modèle.
- Avec 128 layers : 62,91% de précision sur la validation du modèle.
- Avec 256 layers : 57,95% de précision sur la validation du modèle.

Grâce à l'outil Tensorboard et aux différents indices du modèle (loss, accuracy , etc…), on remarque que la précision diminue quand on augmente le nombre de layer de cette couche. Un modèle plus simple serait donc (comme le modèle 1) le plus performant dans notre cas. De plus, dans le modèle 1 qui était plus simple, la loss diminue plus rapidement.

Malheureusement pour ce qui est des logs de Tensorboard, celle-ci ne veulent pas s'enregistrer et je ne sais pas pourquoi, je dois donc entrainer de nouveau mes modèles pour avoir les données du tensorboard.

IV] AMELIORATIONS POSSIBLES ET CONCLUSION

Les améliorations possibles pour mon modèle 1, celui avec 72% de précision sur la partie validation pourraient être par exemple :

- Augmenter le nombre de tweets pour le traitement
- Diminuer le nombre de layers des couches de LTSM et Embedding
- Mettre ma couche de layer d'activation relu entre celle d'embedding et celle de LTSM

Ce projet m'a permis d'approfondir la partie technique du cours de NLP, pour ainsi mieux comprendre le fonctionnement d'un réseau RNN et de me servir de différents outils du machine learning comme le Tensorboard ou la librairie keras.