Natural Language Processing :

Analyse de sentiment

Table des matières

I-Contexte et présentation du sujet

Le projet consiste à réaliser un modèle d'analyse de sentiments, qui prend en compte un commentaire, et qui indique le sentiment de la personne qui l'a écrit. Il indique donc si ce commentaire est plutôt positif ou négatif.

Cela peut s'avérer très utile pour une entreprise qui souhaite avoir un retour sur l'avis de ses clients, de par les réseaux sociaux et des avis en général sur les produits qu'elle vend et sur l'entreprise en elle-même.

Utiliser ce modèle permettrait forcément un gain de temps considérable pour l'entreprise, la lecture des commentaires un à un étant redondant. De plus, il serait possible d'effectuer une vue d'ensemble, un bilan suite à l'analyse de toutes ces données : un outil visuel pour l'entreprise.

A l'aide de ceci, l'entreprise peut donc réorganiser sa stratégie, en adaptant ses offres et produits selon l'avis des clients.

II-Données et approche utilisées

Les consignes étaient tout d'abord d'utiliser une architecture de deep learning qui utilise les réseaux de neurones récurrents, ou Recurrent Neural Networks en anglais (RNN).

Ensuite, l'utilisation d'une technique de word embedding pour transformer les mots en vecteurs était également demandée.

Enfin, créer une pipeline qui prend en entrée une phrase, qui effectue tous les traitements nécessaires, et qui fait la prédiction selon si la phrase est plutôt positive ou négative.

Nous avons à notre disposition un jeu de données mis à disposition sur Kaggle pour servir d'entraînement, et l'API Yelp pour tester notre pipeline et nos modèles.

L'utilisation d'une baseline était nécessaire, dans le but d'avoir plusieurs modèles et de les tester pour déterminer lequel est le plus performant, le plus adapté à notre problématique.

Pour parvenir au but final, nous n'avons pas créé la pipeline immédiatement. Nous avons effectué les différentes étapes indépendamment, pour teste si chacune d'entre elle était fonctionnelle, et pour avoir un premier résultat.

Ainsi, nous avons d'abord récupéré les données du jeu de données : extraire uniquement les commentaires car chacune des données possède différents attributs (le sentiment du commentaire, la date, le commentaire en lui-même…).

Nous avons stocké un petit volume de données en prenant le même nombre de commentaires positifs que de commentaires négatifs. Travailler d'abord avec un petit volume permet de réduire le temps d'exécution et d'avoir rapidement un résultat. Une fois que la solution est prête et fonctionnel avec un petit volume, on peut voir plus grand.

Pour ce qui est du traitement, nous avons effectué le même que celui utilisée habituellement. Il se fait en plusieurs étapes : mettre en minuscules la donnée, effectuer la tokenization, retirer les stopwords, effectuer le stemming et la lemmatization.

Ensuite nous avons déterminé le vocabulaire du jeu de données, il nous permettra d'avoir un ensemble de mots (avec un indice correspondant) sur lequel se baser. On transforme donc les mots en indice grâce au vocabulaire, car celui-ci se présente sous la forme d'un dictionnaire avec comme clés les mots et comme valeurs les indices correspondants.

Puis, nous avons créé un 1er modèle de RNN, plus précisément de LSTM (Long Short-Term Memory).

Nous nous sommes servis du sentiment de chacun des commentaires donnés en entraînement pour définir la sortie à prédire.

III-Solution

Nous avons finalement créé 2 modèles de RNN. Le 1er étant un modèle basé sur du RNN simple, constitué d'une couche d'embedding en tant que couche d'entrée, une couche Simple RNN en tant que couche cachée avec la fonction relu comme fonction d'activation, et une couche de sortie avec un neurone et la fonction sigmoid comme fonction d'activation.

Le 2ème étant un modèle basé sur du LSTM, qui suit la même architecture que la précédente, sauf pour la couche cachée qui est une couche LSTM.

Pour entraîner le modèle, on a créé un tableau de prédictions (0 ou 1 selon si le commentaire est positif ou négatif), qui correspond aux réponses des données d'entraînement. On utilise comme fonction de perte la binary crossentropy, avec l'optimizer adam.

Dans un premier temps, on a testé notre jeu de données avec 2000 données (1000 positifs et 1000 négatifs). On entraîne le modèle avec 50 epochs.

Nous n'avons créé que 2 modèles pour notre baseline, car les 2 sont basés sur du RNN : l'un est un simple RNN, l'autre un LSTM.

Pour créer notre pipeline, nous nous sommes resservis des fonctions créées précédemment lorsque chaque étape se faisait indépendamment.

Nous les avons adaptées selon les besoins, de sorte à ce que notre pipeline sert à la fois pour notre entraînement lorsque le modèle n'existe pas et qu'il a besoin d'être créé, mais aussi pour charger le modèle et effectuer notre prédiction.

Ainsi, elle a été adaptée pour prendre en entrée une liste de phrases ou une phrase simple, mais aussi pour faire l'entraînement sur notre jeu de données.

Pour éviter que le modèle soit recréé à chaque fois et que l'on doit attendre un nouvel entraînement, on sauvegarde notre modèle et on le recharge lors de chaque utilisation.

Comme celui-ci a besoin du vocabulaire que l'on crée lors de l'initialisation du modèle, on exporte également ce vocabulaire au format CSV pour le garder en mémoire.

On l'importe de nouveau à chaque fois que l'on va charger le modèle, et on le retransforme en dictionnaire pour garder la correspondance entre les mots et leurs indices.

Une fois le modèle créé ou chargé, on procède au traitement sur notre donnée d'entrée (la donnée dont on veut prédire le sentiment), et on effectue la prédiction.

Comme nous utilisons la fonction sigmoid en sortie de notre réseau neuronal et que celle-ci nous renvoie une valeur comprise entre 0 et 1, nous devons définir un seuil. Ce seuil permet de savoir si, selon la valeur renvoyée par la fonction sigmoid, la prédiction tend plus vers le positif ou le négatif.

Pour déterminer ce seuil, nous avons créé une fonction qui nous calcule le F1 score pour une liste de seuils déterminés, et nous retourne le maximum d'entre eux. Le F1 score est une valeur qui indique, selon les prédictions et les résultats attendus, la précision d'un modèle par rapport au seuil indiqué. Cette valeur est comprise entre 0 et 1, plus elle est proche de 1, plus le modèle est précis pour ce seuil.

On a donc déterminé une liste de seuils (de 0,1 à 0,9), et on a fait le test pour les données dont on s'est servies pour l'entraînement, avec chacun des modèles.

On a donc ensuite fixé le seuil sur la valeur que nous renvoie cette fonction. Dans un premier temps, lorsque nous n'avions que très peu de données, les F1 scores pour chaque seuil étaient tous très proches de 1 et étaient quasi identiques. Mais plus nous donnons de données, plus le seuil augmente et se rapproche de 0,5.

Il est donc pertinent de le fixer à 0,5 avec un plus grand jeu de données, car en généralisant, il y a des données plus difficiles à analyser.

Cependant, pour la quantité de données que nous avons prises en entrée (2000), les seuils sont légèrement plus faibles (0,4 pour le LSTM et 0,2 pour le simple RNN). Nous gardons donc ces dernières valeurs car après avoir effectué des tests, elles sont plus fiables.

Ceci montre l'intérêt de créer une fonction qui calcule le F1 score pour différents seuils.

Enfin, en ce qui concerne le vocabulaire, comme expliqué précédemment, il répertorie l'ensemble des mots utilisés lors de l'entraînement, mais pour que les données que l'on donne en entrée du modèle soit de la même forme, nous avons dû faire du padding.

En d'autres termes, pour chacune des phrases de l'entraînement, on a récupéré la taille de la phrase la plus longue, et on a fait en sorte que chacune des phrases soit de la même longueur.

Pour cela, on a ajouté une valeur qui indique que nous faisons du padding dans le vocabulaire, avec comme clé '<PAD>' et comme valeur '1'. Pour chacune des phrases nécessitant le padding, on ajoute des 1 (après transformation des mots en indices) jusqu'à obtenir la même longueur que la plus longue des phrases.

Enfin, lors de l'utilisation du modèle pour prédire le sentiment de notre donnée d'entrée, si le mot n'apparaît pas dans le vocabulaire, il est transformé en indice '0'.

Comme indiqué précédemment, nous avons utilisé l'API Yelp pour récupérer des commentaires pour tester notre pipeline, mais celle-ci ne nous permet d'en récupérer que 3. Ainsi, nous avons testé notre pipeline avec 3 phrases transmises par l'API Yelp.

En termes de résultats, on peut dire qu'ils sont déjà satisfaisants par rapport à la complexité du modèle et le nombre de données qu'on a transmis en entrée.

En effet, pour les 3 commentaires récupérés grâce à l'API, les 2 modèles ont prédis justement le sentiment. Ils remplissent donc correctement leur rôle.

Après avoir testé avec d'autres données (phrases personnelles ou quelques commentaires repris directement sur le site AirBnb), on peut estimer que le modèle basé sur du LSTM est plus performant que celui basé sur du simple RNN.

On peut quand même noter quelques différences, notamment le fait que le seuil n'est pas le même pour les 2 modèles (pour la quantité de données que nous lui avons données). De plus, l'entraînement du simple RNN était bien plus rapide que celui du LSTM.

IV-Améliorations possibles

Il est évident que nous pouvons toujours penser à des améliorations possibles, quel que soit le projet. Dans notre cas, pour améliorer notre rendu, il y a plusieurs possibilités.

La 1ère consiste à entraîner nos modèles avec plus de données. De manière évidente, plus le modèle possède de données sur lesquelles se baser, plus il est enclin à analyser correctement de nouveaux commentaires. De même, il pourra enrichir son vocabulaire.

La 2ème possibilité consiste à modifier nos modèles actuels, modifier le nombre de couches, le nombre de neurones par couches, modifier les fonctions d'activation, ou encore modifier des hyperparamètres comme la fonction de perte lors de l'entraînement ou l'optimizer.

La 3ème et dernière possibilité notable consiste à compléter notre baseline en ajoutant d'autres modèles qui différent des précédents.

Il existe donc diverses options qui s'offrent à nous pour améliorer nos résultats, il faut effectuer des tests pour déterminer quel modèle répond le mieux à nos besoins.