FERET

Nathan

Projet IA “Mood For Tweet”

Le but de ce projet est d’estimer l’humeur d’un tweet, ou de tout autre court texte écrit en anglais. Il s’articule en deux parties principales, la première consiste en la construction d’un modèle permettant d’en réaliser une estimation. La seconde partie est une tentative de montrer l’imprédictibilité des tweets.

Vous trouverez tous les codes du projet (données exclues) sur ce dépôt github : github.com/NathanFortyTwo/mood-4-tweet

Partie 1, Construction du modèle de language processing

L’approche générale à un problème de machine learning consiste en les étapes suivantes :

- collection des données

- pré-traitement des données

- construction du modèle

- entrainement du modèle

- évaluation des performances

La première chose que j’ai faite en me lançant dans ce projet a donc été de chercher un dataset sur le site kaggle, j’ai trouvé le dataset « sentiment140 » de l’utilisateur kazanova : kaggle.com/datasets/kazanova/sentiment140

Ce dataset contient 1.6 millions de tweets, avec la polarité de chaque tweet (0 pour négatif et 4 pour positif), les données sont collectées. Il faut alors les traiter. J’ai commencé par ne garder que les colonnes de « mood » (la polarité du tweet) et « text » contenant le tweet en question. J’ai ensuite mélangé les lignes du jeu de données et séparé en deux parties : données d’entrainement et données de test avec un ratio de 0.9

Comme nous travaillons avec du texte, une étape supplémentaire est nécessaire, c’est celle de la vectorisation du texte. Cette étape nécessaire transforme un texte en vecteur, dont chaque composante est un entier unique associé au mot. On applique cette couche de vectorisation de texte sur le jeu d’entrainement afin de ne pas faire fuiter de données du jeu de test.

Bien sur les labels sont eux encodé en un « one-hot vector ». La présence de 2 classes uniquement implique que les vecteurs seront [1,0] et [0,1]

Une fois les données traitées, vient la construction du modèle. Plutôt que de me tourner vers des modèles de type « Transformative Language Models » ou des modèles Seq2Seq, je choisis d'utiliser un type de RNN (Recurrent Neural Network) qu'est le réseau LSTM (Long-Short Term Memory) qui est souvent utilisé dans la classification de texte.

Lors d'un premier essai, je donne au layer de vectorisation un vocabulaire limité a 1000mots, et on construit le modèle avec une architecture simple. Elle est constituée d'un layer « embedding » de dimension 16, d'un LSTM de 8 neurones et de quelques couches denses, les résultats obtenus après un entrainement de 20 minutes sur 2 epochs sont les suivants :

```
acc,loss = model.evaluate(Xtest_vectorized,Ytest_vect)
```

```
5000/5000 [==============================] - 30s 6ms/step - loss: 0.4743 - accuracy: 0.7712
```

On obtient une précision de 77%, ce qui est une performance acceptable, mais qui peut encore être améliorée.

Je décide alors d'augmenter la plupart des hyperparamètres du modèle, en multipliant la taille du vocabulaire par 10, la dimension d'embedding par 4 et le nombre de neurones du LSTM par 2. On obtient alors un modèle à 650 000 paramètres dont l'architecture est la suivante.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 64)          640000
_________________________________________________________________
bidirectional (Bidirectional (None, 32)                10368
_________________________________________________________________
flatten (Flatten)            (None, 32)                0
_________________________________________________________________
dense (Dense)                (None, 50)                1650
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 102
=================================================================
Total params: 652,120
Trainable params: 652,120
Non-trainable params: 0
_________________________________________________________________
```

Nous évaluons alors les performances du nouveau modèle.

[10]:
```
h=model.fit(Xtrain_vectorized,Ytrain_vect,batch_size=512,epochs=2)
```

```
Epoch 1/2
2813/2813 [==============================] - 211s 74ms/step - loss: 0.4326 - accuracy: 0.7986
Epoch 2/2
2813/2813 [==============================] - 206s 73ms/step - loss: 0.3940 - accuracy: 0.8199
```

[11]:
```
acc,loss = model.evaluate(Xtest_vectorized,Ytest_vect)
```

```
5000/5000 [==============================] - 44s 9ms/step - loss: 0.3954 - accuracy: 0.8195
```

La précision du modèle est augmentée de 5%, pour atteindre les 82% de précision sur le jeu de données de test. Cependant, ces augmentations ont un prix à payer en calcul supplémentaires, qui correspond augmentation de 50% du temps de calcul (augmentation de 6 à 9 millisecondes par étape) On pourrait encore tenter d'améliorer la précision en augmentant la dimension du vocabulaire, ou en ajoutant des neurones au LSTM par exemple. On décide que les résultats obtenus sont satisfaisants étant donné l'objectif inital.

Partie 2, Tentative de prédiction de séries temporelles imprévisibles.

Dans cette seconde partie, nous verrons

Les tweets sont connus pour être de courts messages très imprévisibles sur des sujets très variés. Cette seconde partie vise à montrer la difficulté de prédire l'humeur d'un tweet à partir de données disponibles sur twitter uniquement. Le problème est donc une prédiction de séries temporelles.

Pour cela, nous devons retourner à l'étape 1, c'est-à-dire la récupération de données. Plutôt que de se connecter et de copier les tweets manuellement, j'opte pour l'utilisation de l'API twitter qui permet de récupérer des données. Cependant pour la récupération massive de tweets il est nécessaire faire une demande d'accès expliquant les besoin et raisons de l'utilisation de cette API.

J'obtiens une réponse positive 48h plus tard et commence donc à télécharger les tweets et à les stocker dans un fichier csv. J'ai ciblé les tweets du président des états unis Joe Biden (@JoeBiden sur Twitter)

Les données principales récupérées sont les suivantes :

- Texte nettoyé* du tweet
- Date et heure du tweet
- Nombre de likes
- Le « mood » du tweet, évalué par notre premier modèle

(*) Les liens et images ne sont pas prises en compte

La base de données ainsi constitué comporte les quelques derniers milliers de tweets de Joe Biden. La phase de pré-processing contient la vectorisation du texte, et le formatage des dates et heures. Le texte doit être vectorisé, avec le même layer qu'on a utilisé, sauvegardé en un pickle (utilisé pour sérialiser et désérialiser des objets Python). Le nouveau modèle utilisé pour la prédiction de série temporelle est un LSTM simple, comportant 5 neurones.

Au fur et a mesure des entrainements on remarque que la meilleure manière de minimiser la fonction de perte MSE sur ce cas, n'est pas d'essayer de prédire l'humeur de chacun des tweets, mais de se rapprocher de la moyenne de l'humeur de l'ensemble des tweets du dataset.

Ce comportement est clairement visible sur l'animation accuracy.gif. Les variations initiales présentes sur la courbe prédit initiale se dissipent peu à peu pour donner une prédiction constante, à la moyenne des « mood » estimées du jeu de test.

Le modèle est, comme prévu, incapable de prédire l'humeur, à cause de leur nature justement imprévisible.

```
model.evaluate(shaper(x_test_scaled),y_test,batch_size=1)

32/32 [==============================] - 0s 1ms/step - loss: 0.1456
```

On arrive à une erreur MSE de 0.14, donc en moyenne une différence de $\sqrt{0.1456} = 0.38$

C'est une performance au mieux médiocre quand on note que les valeurs que l'on tente de prédire sont comprises entre 0 et 1 (et donc que la MSE est majorée par 0.25)

Conclusion

La première partie permet d'obtenir des résultats convenables, à partir d'un modèle relativement léger (650k paramètres) et d'un jeu de données d'un million lignes.

Les enjeux techniques que présentait la seconde partie ont pu être pour la plupart surmontés : accès développeur a l'API de twitter, sauvegarde du modèle, sérialisation et désérialisation du layer de vectorisation, ainsi que la construction et l'évaluation du LSTM ont fonctionné. Seule la mauvaise performance du modèle est à déplorer, bien qu'elle ait été très largement prévisible de part la nature aléatoire des tweets.