documentation pour ensemble_v1

November 28, 2020

0.1 28 Nov

Ceci est un petit résumé de la façon d'obtenir ce que nous avons appelé *ensemble_v1*, qui est en réalité seulement un ensemble de plusieurs $\text{RoBERTa}_{\text{large}}$.

Le fait d'utiliser plusieurs fois le même "modèle", ou tout du moins la même architecture avec le même pré-entrainement peut paraître bizarre, car normalement, on essaie de faire des ensemble de modèles de performance similaire, mais les plus différents possible…

Ici l'idée vient principalement de mes lectures des solutions des dernieres compétitions Kaggle sur de la classification de phrases, nottament celle-ci. Cette discussion amène directement vers cet article qui présente l'idée de chercher des *bonnes* combinaison de graines aléatoire pour les transformers d'architectures similaire à BERT.

0.2 Méthode d'évaluation et premiers tests

Nous avions déjà remarqué que un simple hold-out semblait ok pour estimer l'erreur de généralisation du modèle, je reprend donc la même méthode ici (`test_size`=0.2 et `random_state`=42069).

Tout d'abord, il est vrai que j'avais déjà remarqué quand je faisais des tests avec DistilBERT ou $\text{RoBERTa}_{\text{large}}$ que les résultats sur l'échantillon test étaient relativement instable (de l'ordre des 3% pour notre jeu de données). Mais je pensais que cela était du à un mauvais nombre d'epochs, j'avais bien compris que l'initialisation des poids de la dernière couche de classification était un problème, mais je pensais le résoudre en *freezant* les premiers layers.

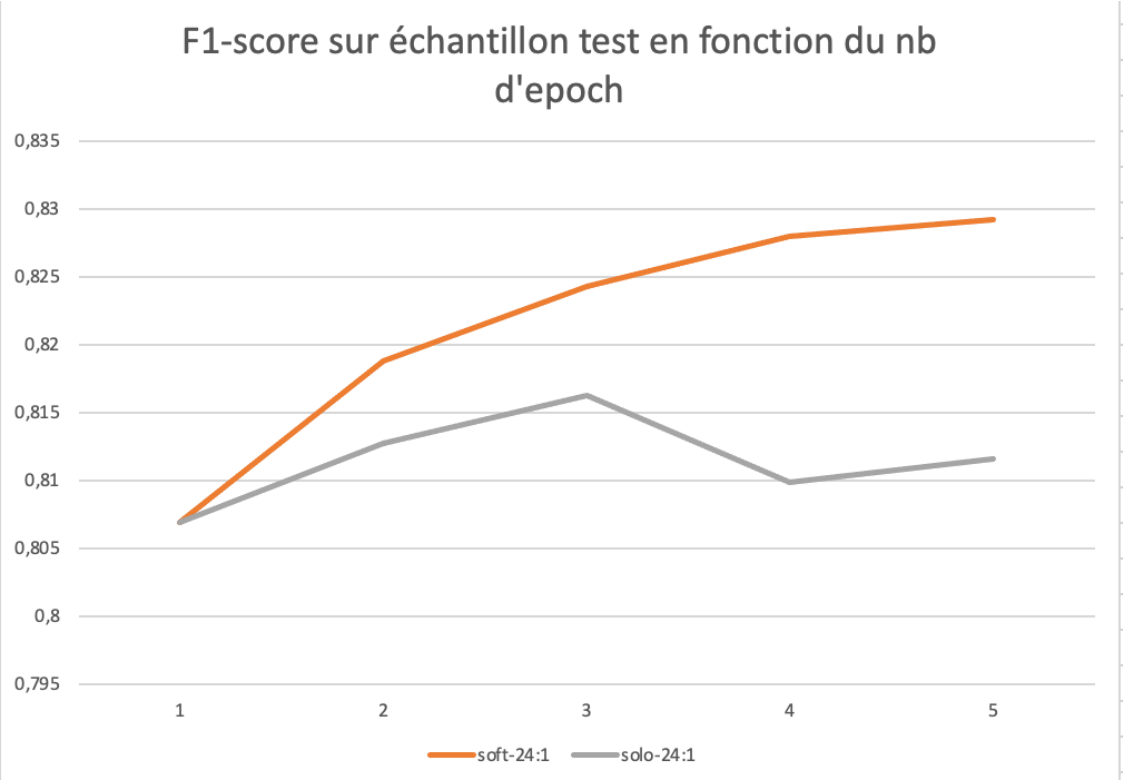
Ici, l'idée est donc plutôt de ne freeze aucun layers, mais plutôt d'essayer plusieurs combinaisons différentes de graines aléatoire. Cependant un problème apparait : **comment choisir les BONNES combinaison sans avoir de jeu témoin à disposition**, quand on prototype, on voit bien comment faire puisque il est facile de prendre une petite partie des données. Mais quand à la fin on est dans une optique de soumission pour la compet, on a envie de prendre TOUTES les données à notre disposition pour l'entrainement. Ce que j'ai choisit de faire (c'est un choix qu'il va falloir justifier si on le conserve…) c'est de prendre quelques combinaisons de graines (j'ai fait avec 5 pour l'instant, mais pour la fin une quinzaine serait peut-être mieux ?), et de faire un *soft-voting*. l'idée est donc de faire :

$$\hat{y} = \arg\max_j \sum_{i=1}^{n} p_{i,j}$$

où $p_{i,j}$ est la probabilité pour le label j prédite par le i-ème classifier, sachant que les i correspondent à différentes combinaison de graines aléatoire pour l'initialisation des poids de la dernière couche, ainsi que pour l'ordre d'arrivé des données.

Mais ATTENTION C'EST PAS FINI !!!

Cette méthode sous-entend tout de même de choisir un nombre d'epoch, de fine tune sur ce nombre d'epoch, puis de faire la prédiction. Si l'on revient à la discussion Kaggle que j'ai link ci dessus, et qu'on lit les réponses aux commentaires, le gagnant dit à un moment que ce qu'ils ont fait, plutôt que de choisir un nombre d'epoch et de faire 1 seule prédiction après entrainement, est plutôt de faire une prédiction après CHAQUE epoch, puis de faire un soft voting de ces quelques prédictions. J'ai essayé de faire cela sur 2 modèles $\text{RoBERTa}_{\text{large}}$ qui ne diffère que par leurs combinaisons de graines aléatoires. Voici un peu les différence en capacité de généralisation pour l'un de ces 2 modèles, l'autre suit la même tendance :



Donc, même quand on pourrait considérer que après la 3ème ou 4ème époch de fine tuning, on commence à overfit, le soft-voting des prédiction après chaque epoch continue à devenir encore meilleur ! En plus, faire ainsi retire le besoin de cherche précisement une bonne epoch pour la prédiction, on va plutôt en faire 5 ou 6, faire une prédiction à la fin de chaque epoch, et faire un *soft-voting* des 5.

Ainsi, l'ensemble final est plutôt de cette forme :

$$\hat{y} = \arg\max_j \sum_{i=1}^{n} \sum_{e=1}^{m} p_{i,j,e}$$

où $p_{i,j,e}$ est la probabilité pour le label j prédite par le i-eme classifier juste après la e-ième epoch.

Comme suggéré par Loic, on fait ça avec un callback plutot qu'une boucle `for` (ça semble pas changer grand chose en terme de rapidité, mais c'est plus beau) :

[ ]:
```
from tensorflow.keras.callbacks import Callback

class prediction_history(Callback):
    def __init__(self):
        self.predhis = []
    def on_epoch_end(self, epoch, logs={}):
        self.predhis.append(model.predict(test_dataset, verbose=1))
```

Puis, après entrainement, je regroupe tout ça dans un fichier `.npy` de cette manière :

[ ]:
```
y_prob_somme = sum(callbacks[0].predhis)

with open('roberta-large-sub-125.npy', 'wb') as f:
    np.save(f, y_prob_somme)
```

Les nombre à la fin du fichier correspondent aux `seed`. Les 2 premiers sont ceux de la graine globale (`transformers.seed(12)`, qui change les seed de `random`, `tensorflow`…) et le chiffre à la fin correspond à la seed dans le `shuffle` à la création du `tf.data.Dataset`

Une fois cela fait avec quelques combinaison différentes, je regroupe tout en local, le code est dans le notebook sans titre…

0.3 Quelles perspectives pour la suite ?

- essayer de reprendre ce code pour pouvoir entrainer sur TPU XLNet, AlBERT et autre ?
- essayer de faire du bagging de BERT, cela semble fonctionner d'après cet article