Projet 8:
CommonLit Readability Prize

Edward Levavasseur

Ingénieur Machine Learning

La compétition Kaggle de CommonLit Readability Prize proposait aux participants de développer des algorithmes de machine learning pour classer - de manière supervisée - des textes utilisés dans les systèmes scolaires américains. Les participants sont invités à effectuer des prédiction de difficulté d'une sélection de textes. L'équipe de CommonLit évalue ensuite les performances de ces prédictions en utilisant un RMSE. Pour ma part j'ai eu recours à plusieurs méthodes de word embedding (CountVectorizer, TF-IDF) pour représenter les textes dans un même espace vectoriel. Ensuite j'ai construit un réseau de neurones pour prédire la difficulté des textes, qui fait fitté sur le word embedding. J'ai obtenu un RMSE de 0.750, ce qui correspond à un R2 d'environ 0.89 sur les données test.

Keywords: NLP, Word Embedding, Supervised Learning, Readability

1. Introduction

Dans l'objectif d'optimizer la classification de textes étudiés dans le programme scolaire américain, CommonLit a organizé une compétition Kaggle dont l'algorithme gagnant pourra être utilisé pour facilier la classification des textes par niveau de difficulté. Ils expliquent que traditionnellement, la mesure de la difficulté des textes a reposé sur l'utilisation de formules telles que le *Flesch-Kincaid Grade Level*, qui compte le nombre de mots moyen par phrase, et le nombre de syllabes moyen par mots. Comme précisé dans l'ennoncé, ces méthodes ne sont pas solidement étayées par la théorie, et ne génèrent pas des prédictions

qui soient très en accord avec l'opinion des lecteurs.

Par conséquent, afin de développer des méthodes plus fines de classification de textes, CommonLit a fait lire des extraits de textes à un nombre d'individus qui on ensuite été mandaté de noter la difficulté de ces textes. CommonLit fournit donc une base de données contenant les extraits de plus de 2.000 textes, avec une note de difficulté moyenne telle que renseignée par les lecteurs : variable nommée *'target'*. C'est précisément cette variable qu'il s'agit de prédire, en utilisant les extraits de textes.

Dans un premier temps, en m'inspirant de l'indice de *Flesch-Kincaid Grade Level*, j'ai essayé de developper mes propres indices de difficulté sémantique. Pour cela, j'ai commencé par compté la fréquence de chaque mot utilisé dans l'ensemble des corpus de textes. Grâce à cela j'ai pu déterminer quels mots étaient rares, et quels mots étaient plus courrants. J'ai pu ensuite déterminer 2 features des textes: la fréquence des mots rares dans chaque texte, et la fréquence moyenne des mots du textes dans le corpus complet de textes. En appliquant une régression linéaire, un régression random forest, et une régression XGBoost, je n'ai pas pu dépasser un R2 de 0.4 sur les données Training.

N'étant pas une méthode très utile, j'ai approché le problème d'une manière différente, en représentant les textes dans un word embeddings, puis en "fittant" un réseau de neurones sur le word embedding. Dans la partie suivante je décris le modèle que j'ai utilisé. Dans la 3e partie je décrit les résultats obtenus, et que j'ai soumis à Kaggle. Dans la 4e partie je décris comment j'ai également utilisé le semantic embedding avec la méthode RoBERTa, en m'inspirant des kernels des autres candidats. Dans la partie 5 je conclus.

2. Le modèle

Etant donné que l'évaluation des performances de l'algorithme sont mesurées sur un échantillon de 7 textes prédéfinies, et afin de mieux anticiper comment mon algorithme est susceptible de performer sur cet échantillon, et extrait 3 jeux de données du *training set* avec 7 observations chacune, et que j'ai appelé respectivement *Test 1*, *Test 2* et *Test 3*. Après cela j'ai appliqué une cross-validation en divisant les données train en 10 sous-ensembles. L'intégralité du modèle est donc lancé 10 fois avec 1 sous-ensemble différent pour les don-

nées de validation à chaque itération, et les 9 restantes étant utilisées comme données de training.

Les données se présentent comme suit:

- Training: 89.3%
- Validation: 9.92%
- Test 1: 7 observations
- Test 2: 7 observations
- Test 3: 7 observations

Où *Training* et *Validation* sont différent à chacune des 10 itérations, et où *Test 1*, *Test 1* et *Test 3* restent identiques du début à la fin.

Ensuite j'ai essayé deux méthodes de Word Embedding - *CountVectorizer* et *TF-IDF* - que j'ai "fitté" sur les données de *Training* et que j'ai utilisé pour "transfromer" les données *Trainning*, *Validation*, *Test 1*, *Test 2*, *Test 3*. D'abord, le CountVectorizer compte simplement le nombre de fois que chaque mot du corpus a été utilisé dans chaque texte. Il en résulte que les textes sont représentés dans un espace vectoriel, où chaque dimension correspond à un mot du corpus. J'ai ensuite appliqué le réseau de neuronnes sur cet espace. Dans un second temps, j'ai appliqué un TF-IDF qui représente chaque texte dans un espace vectoriel, similairement au CountVectorizer, mais compte la fréquence de chaque mot dans le texte, divisé par le logarithme de l'inverse du nombre de documents contenant ce mot. A nouveau j'ai appliqué le même réseau de neurones sur l'espace vectoriel du TF-IDF à la place du CountVectorizer.

Le réseau de neurones que j'ai utilisé repose sur la structure suivante:

- Dense: 2.000 (activation: ReLu)
- Dense: 1.000 (activation: ReLu)

- Dense: 500 (activation: ReLu)
- Dense: 250 (activation: ReLu)
- Dense: 125 (activation: ReLu)
- Dense: 60 (activation: ReLu)
- Dense: 30 (activation: ReLu)
- Dense: 15 (activation: ReLu)
- Dense: 1 (activation: ReLu)

Par ailleurs j'ai utilisé un Dropout Rate de 0.1, qui désactive 10% des neurones à chaque couche afin de réduire le sur-aprentissage sur les données Train. Enfin j'ai utilisé un optimizateur *sgd* et une fonction de loss *mse*, le tout avec 150 epochs.

3. Résultats

J'ai d'abord utilisé un CountVectorizer, sur lequel j'ai appliqué le réseau de neurones plus haut. En revanche dans cette version préliminaire, je n'avais pas effectué une cross-validation. A la place, j'avais utilisé 2 validations sets l'un après l'autre, et j'avais lancé 10 fois l'algorithme.

```
R2 - CountVectorizer :

Train :  0.974
Validation 1 :  0.859
Validation 2 :  0.894
Test 1 :  0.929
Test 2 :  0.826
Test 3 :  0.958


RMSE - CountVectorizer :

Train :  0.358
Validation 1 :  0.838
Validation 2 :  0.722
Test 1 :  0.603
Test 2 :  0.982
Test 3 :  0.405
```

Ensuite en remplaçant le CountVectorizer par un TF-IDF:

```
R2 - TF-IDF :

Train :  0.981
Validation 1 :  0.862
Validation 2 :  0.896
Test 1 :  0.919
Test 2 :  0.824
Test 3 :  0.947


RMSE - TF-IDF :

Train :  0.308
Validation 1 :  0.826
Validation 2 :  0.715
Test 1 :  0.648
Test 2 :  0.987
Test 3 :  0.455
```

J'ai également remplacé le word embedding par une méthode de semantic embedding: Doc2Vec. Le problème rencontré avec cette méthode est que la transformation d'un texte en vecteur n'est pas constante. Ainsi le même texte sera transformé en 2 vecteurs légèrement différents, lorsqu'on lui applique Doc2Vec 2 fois. Cela a pour conséquence de réduire la capacité du réseau de neurones à prédire la variable *'target'*.

```
R2 - Doc2Vec :

Train :  0.957
Validation 1 :  0.82
Validation 2 :  0.819
Test 1 :  0.952
Test 2 :  0.744
Test 3 :  0.827


RMSE - Doc2Vec :

Train :  0.46
Validation 1 :  0.945
Validation 2 :  0.941
Test 1 :  0.499
Test 2 :  1.188
Test 3 :  0.819
```

On remarque ici que les prédictions sur les Validation sets 1 et 2 (R2 respectivement égal à 0.820 et 0.819) sont inférieures aux performances obtenues avec CountVectorizer (respectivement 0.859 et 0.894) ou avec TF-IDF (respectivement 0.862 et 0.896). Pour le modèle final, j'ai donc décidé d'utiliser le TF-IDF comme word embedding.

Dans le dernier modèle (détaillé dans la partie 2. de ce papier), puisque j'ai utilisé le cross-validation avec 10 sous-ensembles de données, j'ai fait 10 prédictions pour chaque

base de données Test. Ayant effectué ces 10 prédictions, je les ai aggrégées en calculé la médiane et la moyenne des prédictions. Les résultats de ce algorithme - qui sont soumis à la compétition Kaggle - sont les suivants:

```
Median Predictions - R2 :

Test 1 :  0.9354198436081085
Test 2 :  0.8615050006348941
Test 3 :  0.8991503191379402

Mean Predictions - R2 :

Test 1 :  0.9309244590857284
Test 2 :  0.8658349111150256
Test 3 :  0.9098448163144289
```

```
Median Predictions - RMSE :

Test 1 :  0.4957983810644573
Test 2 :  0.8714743898206182
Test 3 :  0.7260139364634548

Mean Predictions - RMSE :

Test 1 :  0.5127642108805324
Test 2 :  0.8577433189621063
Test 3 :  0.6864407301763708
```

Par ailleurs, le score finale de la soumission kaggle était de RMSE=0.750.

4. Modification : Roberta

Après avoir effectué cette Soumission à Kaggle, j'ai cherché à optimizer d'avantage les prédictions, en m'inspirant des kernels des autres participants au concours. J'ai pu remarquer que beaucoup d'autres candidats reposaient sur la méthode d'embedding de RoBERTa, qui est une version optimizée de BERT.

Par conséquent, avant d'implémenter le word embedding CountVectorizer, j'ai appliqué un Tokenizer RoBERTa qui est pré-entrainé sur un grand nombre de textes. Cette fonction de Tokenization remplace un texte ou une phrase par un vecteur de chiffres.

Enfin, j'ai appliqué le même réseau de neurones, sur le CountVectorizer et j'ai lancé 10 prédictions, en utilisant toujours la même cross-validation. En calculant le score médian et moyen des 10 prédictions, j'ai obtenu les résultats suivants:

```
Median Predictions - RMSE :

Test 1 :  0.6501217693178905
Test 2 :  0.6286083598152495
Test 3 :  0.41139320098665966

Mean Predictions - RMSE :

Test 1 :  0.6446965056403581
Test 2 :  0.6354785420312691
Test 3 :  0.4141882248484389

Final Predictions - RMSE :

Test 1 :  0.64727407107668
Test 2 :  0.6320054406835749
Test 3 :  0.4125495987113205
```


```
Median Predictions - RMSE :

Test 1 :  0.6501217693178905
Test 2 :  0.6286083598152495
Test 3 :  0.41139320098665966

Mean Predictions - RMSE :

Test 1 :  0.6446965056403581
Test 2 :  0.6354785420312691
Test 3 :  0.4141882248484389

Final Predictions - RMSE :

Test 1 :  0.64727407107668
Test 2 :  0.6320054406835749
Test 3 :  0.4125495987113205
```


Bien que les résultats soient meilleurs dans l'ensemble avec cette tokenization RoBERTa, c'est une méthode qui est pré-entraînée et qui nécessite d'être téléchargée d'internet. Or la compétition n'accepte pas l'accès à internet. Bien qu'il soit possible de télécharger les dossiers nécessaires, et de les "uploader" sur Kaggle, je n'ai personnelleemnt pas réussi à le faire.

5. Conclusion

Pour conclure, la compétition *CommonLit Readability Prize* évaluait les candidats sur leur capacité à prédire le niveau de difficulté de 7 textes. Les candidats avaient accès à des données Train contenant les extraits de plus de 2000 textes, accompagné d'une mesure de difficulté (variable 'target') telle que notée par un panel d'individus.

J'ai d'abord essayé d'identifier des features des textes qui pourraient révéler leur difficulté. J'ai notamment compté le nombre de mot rares, et la fréquence moyenne des mots dans le corpus général de textes. En fittant des différentes régressions sur ces 2 variable (*linéaire*, *Random Forest* et *XGBoost*) dans les données Train, je n'ai pas obtenu des performances remarquables ($R2 \leq 0.4$). J'ai donc du procéder autrement.

La méthode que j'ai retenu consiste à représenter les textes dans un word embedding *TF-IDF*, puis à "fitter" un réseau de neurones sur le TF-IDF des données Train avec 10

cross-validations, et à effectuer 10 prédictions sur données Test. Enfin, j'ai retenu la médiane des 10 prédictions comme métrique de difficulté des textes. J'ai obtenu un $RMSE$ de 0.755 dans la soumission finale (ce qui correspond à un $R2$ d'environ 0.85).

Enfin, j'ai appliqué une tokenization RoBERTa avant le word embedding TF-IDF, et j'ai obtenu des $RMSE$ de 0.64, 0.63 et 0.41. Sachant que les meilleures performances actuellement obtenues par un candidat dans la compétition sont de 0.462, ce sont des performances acceptables.

Il est probable que cette méthode génèrerait de bonnes performances sur l'échantillon de textes à soumettre, mais cette méthode nécessite d'avoir recours à internet, car la méthode RoBERTa est pré-entraînée, et elle nécessite donc de télécharger les paramêtre du pré-entrainement. Or Kaggle internet l'utilisation d'internet dans le kernel à soumettre.