ES203 Projet
Compétition Kaggle : Toxic Comments

Kaori Watanabe, Zhi Zhou

Avril 2018

Table des matières

1 Introduction

Discuter de nos centres d'intérêt peut être quelque chose de difficile. La menace d'abus et de harcèlement en ligne entraîne le fait que beaucoup de gens cessent de s'exprimer et abandonnent la recherche d'opinions différentes. Les plates-formes de communication se débattent pour faciliter les conversations, amenant de nombreuses communautés à limiter ou à fermer complètement les commentaires des utilisateurs.

Nous cherchons à construire un modèle à plusieurs têtes capable de détecter différents types de toxicité comme les menaces, l'obscénité, les insultes et la haine reposant sur l'identité mieux que les modèles actuels de Perspective. Nous utilisons un jeu de données de commentaires provenant des modifications de la page de discussion de Wikipédia. Les améliorations apportées au modèle actuel devraient aider les discussions en ligne à devenir plus productives et plus respectueuses.

2 Pré-traitement des données

Considérant que nous n'avons pas de GPU dans nos ordinateurs, et le fait que l'algorithme d'apprentissage nécessite une capacité de calcul puissante, nous utilisons le GPU en ligne **Google Colabratory** pour exécuter du code dans ce projet.

Google Colaboratory est un projet de recherche Google créé pour aider à diffuser l'apprentissage et la recherche en apprentissage automatique. Il s'agit d'un environnement de Jupyter (notebook Python) qui ne nécessite aucune configuration et qui s'exécute entièrement dans le cloud.

Les Colaboratory-notebooks sont stockés dans Google Drive et peuvent être partagés comme vous le feriez avec Google Docs ou Sheets. L'utilisation de Colaboratory est gratuite pendant 12 heures consécutives.

2.1 Préparer les données

Premièrement, observons les données fournies.

```
import pandas as pd
TRAIN_FILE = "train.csv"
TEST_FILE = "test.csv"
train = pd.read_csv(TRAIN_FILE)
test = pd.read_csv(TEST_FILE)
train.head()
```

Listing 1 – Affichage des données

Nous pouvons voir que les données sont constituées d'un identifiant pour chaque commentaire (id), du commentaire (comment_text), et six classifications des commentaires, qui sont toxic, severe toxic, obscene, threat, insult et identity hate.

Nous vérifions les données pour voir s'il y a une valeur nulle.

```
# check if there is null value
print(train.isnull().any())
print(test.isnull().any())
```

Listing 2 – Code pour vérifier les données

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
45506	79b6a5bd45e6daa8	"\n\nFirstly, please kindly specifically say w...	0	0	0	0	0	0
68774	b7fc9dc12eb8c7c8	Note to self \n\nI'm putting this here so that...	1	0	0	0	0	0
9216	1888ac35c8979187	:DenTek Oral Care\nA tag has been placed on De...	0	0	0	0	0	0
120229	830afe753aef4f61	Note that someone called Bidge appears to be v...	0	0	0	0	0	0
42645	71d065cf82dfd118	It'd be nice to have youtube or audio link... ...	0	0	0	0	0	0

FIGURE 1 – Afficher les données

Nous pouvons voir qu'il n'y a pas de valeur nulle.

```
id               False
comment_text     False
toxic            False
severe_toxic     False
obscene          False
threat           False
insult           False
identity_hate    False
dtype: bool
id              False
comment_text    False
dtype: bool
Processing text dataset
```

FIGURE 2 – vérifier les données nulles

Avant le traitement des données avec GloVe, nous avons préparé les données pour améliorer la qualité de la vectorisation. Nous faisons la même chose avant de former le modèle d'apprentissage. Voici les étapes de cette préparation.

1. Transformer les mots en lettres minuscules.
2. Supprimer les "mots d'arrêt". Il y a beaucoup de mots d'arrêt en anglais, tels que *the*, *a*, *an*, *that*, etc. En effet ces mots n'ont pas beaucoup d'effet sur la classification des commentaires, on peut donc les supprimer directement.
3. Supprimer les caractères spéciaux, comme '*', '/', '@', etc.
4. Remplacer les numéros avec 'n'.

```
# Regex to remove all Non-Alpha Numeric and space
special_character_removal=re.compile(r'[^a-z\d ]',re.IGNORECASE)
# regex to replace all numerics
replace_numbers=re.compile(r'\d+',re.IGNORECASE)
def text_to_wordlist(text, remove_stopwords=False, stem_words=False):
    # Convert words to lower case and split them
    text = text.lower().split()
    # Remove stop words
    if remove_stopwords:
        stops = set(stopwords.words("english"))
        text = [w for w in text if not w in stops]
    text = " ".join(text)
    # Remove Special Characters
    text=special_character_removal.sub('',text)
    # Replace Numbers
    text=replace_numbers.sub('n',text)
    # Return a list of words
    return(text)
```

Listing 3 – Code pour nettoyer les données

2.2 Words Embedding

On utilise la méthode Global Vectors for Word Representation (**GloVe**) pour générer "vectoriser" mots.

GloVe est un algorithme d'apprentissage non supervisé permettant d'obtenir des représentations vectorielles de mots. L'apprentissage est effectué sur des statistiques agrégées de co-occurrence mot-à-mot global à partir d'un corpus, et les représentations résultantes présentent des sous-structures linéaires intéressantes de l'espace vectoriel mot. Il s'agit d'examiner chaque mot, considéré comme mot central et comparer avec les mots voisins de chaque côté. Cela permet de repérer des caractéristiques similaires ou différentes entre les différents mots, et attribuer un taux à chaque caractéristique, pour chaque mot.

Quelques paramètres importants dans Glove :

- Window size : le nombre de mots considérés avant et après le mot central.
- Vector size : dimension du vecteur du mot.

Ces paramètres influencent la qualité de la présentation des mots. Définir la window size comme 10 permet d'obtenir une bonne précision lors de la représentation d'un grand nombre de mots[2]. Dans notre cas, nous la définissons à 10. Vector size est généralement compris entre 100 et 300 ; une grande dimension de vecteur pourrait mieux représenter les mots, mais compte tenu le temps de l'exécution, nous le définissons à 100.

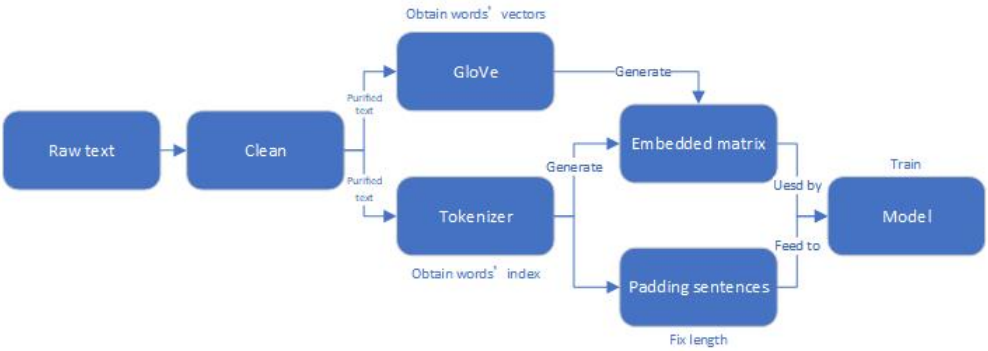

FIGURE 3 – Étapes du processus

On doit ensuite fixer un nombre de mots constant pour chaque échantillon : il s'agit du padding. En effet, certains commentaires étant plus longs que d'autres, on a besoin de fixer un nombre de mots pour uniformiser les échantillons (couper les phrases qui contiennent plus de mots que ce nombre, et remplir avec des zéros les phrases contenant moins de mots). Nous avons tracé un diagramme pour montrer la distribution du nombre de mots dans les phrases, et nous pouvons voir que la plupart des phrases contiennent moins de 150 mots. Nous avons donc décidé de fixer les phrases à 150 mots.

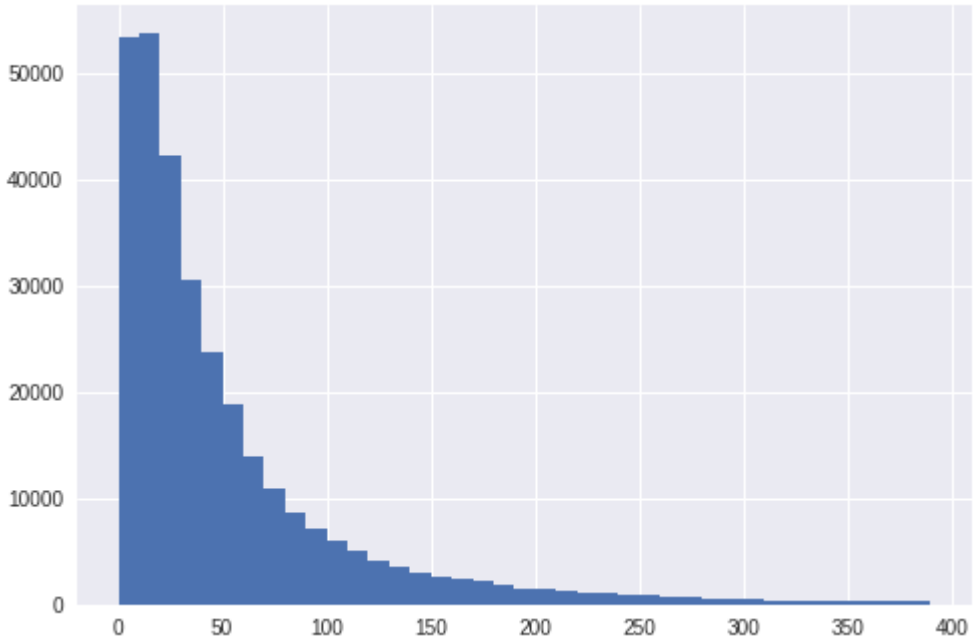

FIGURE 4 – Nombre de mots dans les phrases

```
1841 x 785

# Makes programs, downloads sample data, trains a GloVe model, and then evaluates it.
# One optional argument can specify the language used for eval script: matlab, octave or [default] python

make
# data file to be embedded
CORPUS=comments_all.csv
# vocabulary file, generated automatically
VOCAB_FILE=vocab.txt
COOCCURRENCE_FILE=cooccurrence.bin
COOCCURRENCE_SHUF_FILE=cooccurrence.shuf.bin
BUILDDIR=build
SAVE_FILE=pretrain.glove.100d
VERBOSE=2
MEMORY=4.0
VOCAB_MIN_COUNT=5
VECTOR_SIZE=100
MAX_ITER=15
WINDOW_SIZE=10
BINARY=2
NUM_THREADS=8
X_MAX=10

echo "$ $BUILDDIR/vocab_count -min-count $VOCAB_MIN_COUNT -verbose $VERBOSE < $CORPUS > $VOCAB_FILE"
$BUILDDIR/vocab_count -min-count $VOCAB_MIN_COUNT -verbose $VERBOSE < $CORPUS > $VOCAB_FILE
echo "$ $BUILDDIR/cooccur -memory $MEMORY -vocab-file $VOCAB_FILE -verbose $VERBOSE -window-size $WINDOW_SIZE < $CORPUS > $COOCCURRENCE_FILE"
$BUILDDIR/cooccur -memory $MEMORY -vocab-file $VOCAB_FILE -verbose $VERBOSE -window-size $WINDOW_SIZE < $CORPUS > $COOCCURRENCE_FILE
echo "$ $BUILDDIR/shuffle -memory $MEMORY -verbose $VERBOSE < $COOCCURRENCE_FILE > $COOCCURRENCE_SHUF_FILE"
$BUILDDIR/shuffle -memory $MEMORY -verbose $VERBOSE < $COOCCURRENCE_FILE > $COOCCURRENCE_SHUF_FILE
echo "$ $BUILDDIR/glove -save-file $SAVE_FILE -threads $NUM_THREADS -input-file $COOCCURRENCE_SHUF_FILE -x-max $X_MAX -iter $MAX_ITER -vector-size $VECTOR_SIZE -binary $BINARY -vocab-file $VOCAB_FILE -ve
rbose $VERBOSE"
$BUILDDIR/glove -save-file $SAVE_FILE -threads $NUM_THREADS -input-file $COOCCURRENCE_SHUF_FILE -x-max $X_MAX -iter $MAX_ITER -vector-size $VECTOR_SIZE -binary $BINARY -vocab-file $VOCAB_FILE -verbose $V
ERBOSE
if [ "$CORPUS" = 'text8' ]; then
   if [ "$1" = 'matlab' ]; then
       matlab -nodisplay -nodesktop -nojvm -nosplash < ./eval/matlab/read_and_evaluate.m 1>&2
   elif [ "$1" = 'octave' ]; then
       octave < ./eval/octave/read_and_evaluate_octave.m 1>&2
   else
       echo "$ python eval/python/evaluate.py"
       python eval/python/evaluate.py
   fi
fi
```

FIGURE 5 – Configuration pour GloVe

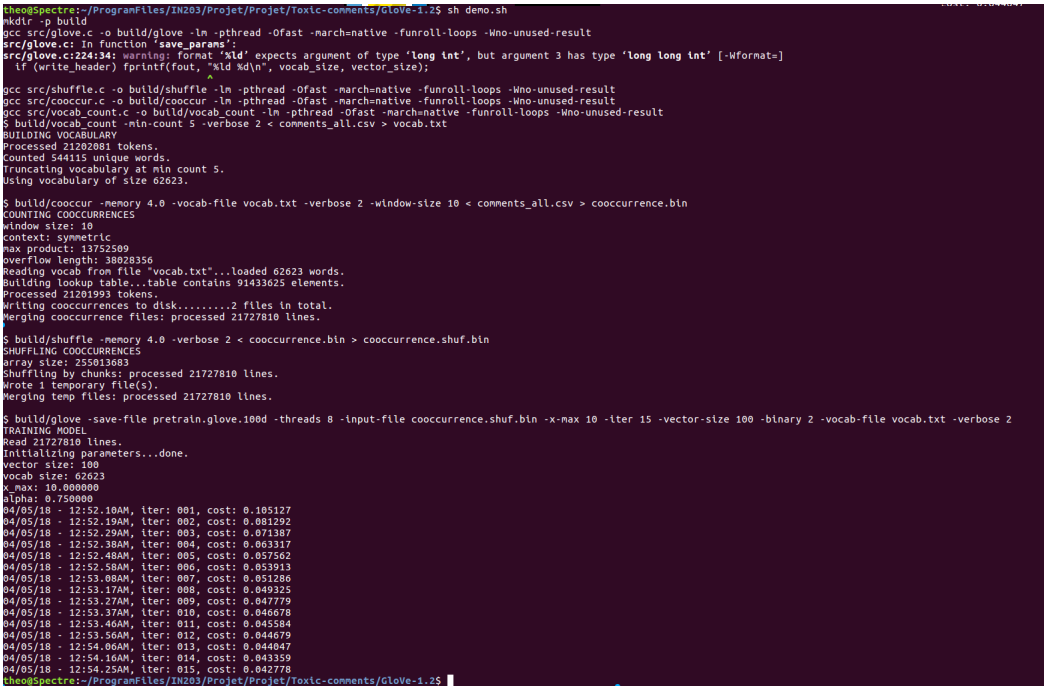

FIGURE 6 – L'exécution de GloVe

3 Training

3.1 Modèle

Long Short-Term Memory (LSTM), le modèle que nous avons choisi, est un type de réseau neuronal récurrent qui peut apprendre la dépendance d'ordre entre les éléments d'une séquence.

Les LSTMs ont la promesse d'être en mesure d'apprendre le contexte requis pour faire des prédictions dans les problèmes de prévision de séries chronologiques, plutôt que d'avoir ce contexte prédéfini et corrigé. Ils peuvent mémoriser des informations pendant de longues périodes. En effet, ils permettent de choisir de garder en mémoire ou oublier certaines informations, même "lointaines", et ainsi éviter le problème des dépendances à long terme (long-term dependencies).

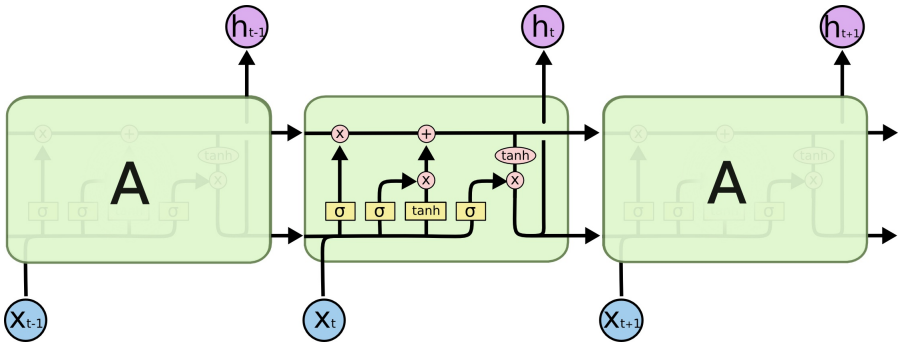

FIGURE 7 – The repeating module in an LSTM[1]

Compte tenu de ces avantages, nous utilisons un modèle LSTM, avec principalement 7 couches : *Embedding, LSTM, Maxpooling, Dropout, Fully connection, Dropout* et *Fully connection*. En outre, nous utilisons le *k-fold cross validation* dans notre cas (cf. notre code pour plus de détails).

```
model = Net(max_features,embed_size,hidden_dim,pool_size,lin_dim)
print(model)
```

Listing 4 – Code pour imprimer le modèle

```
Net(
  (embeddings): Embedding(100000, 100)
  (lstm): LSTM(100, 50, num_layers=2, batch_first=True, bidirectional=True)
  (max_pool): MaxPool1d(kernel_size=50, stride=50, padding=0, dilation=1, ceil_mode=False)
  (dropout): Dropout(p=0.25)
  (lin_1): Linear(in_features=300, out_features=50)
  (relu): ReLU()
  (dropout_2): Dropout(p=0.25)
  (lin_2): Linear(in_features=50, out_features=6)
  (bn): BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True)
  (sig): Sigmoid()
)
```

FIGURE 8 – L'architecture du modèle

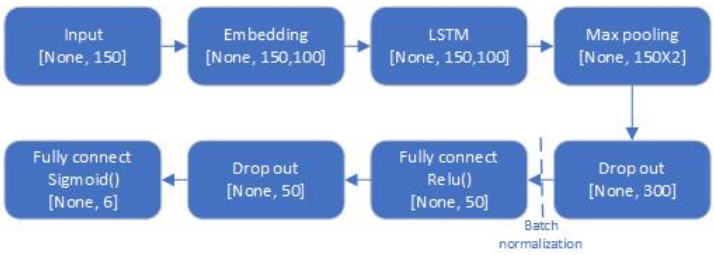

FIGURE 9 – Process steps

Nous utilisons la méthode de *batch normalization* dans notre exemple. L'entraînement des réseaux neuronaux profonds est compliqué par le fait que la distribution des entrées de chaque couche change au cours de l'entraînement, à mesure que les paramètres des couches précédentes changent. Cela ralentit le processus en exigeant des taux d'apprentissage plus faibles et une initialisation prudente des paramètres, et il est difficile de former des modèles avec des non-linéarités saturantes. Nous adressons le problème en normalisant les entrées de couche. Notre méthode tire sa force de faire de la normalisation une partie de l'architecture du modèle et d'effectuer la normalisation pour chaque entraînement mini-batch. Batch normalization nous permet de utiliser des taux d'apprentissage beaucoup plus élevés.

La couche *Dropout* consiste à ignorer les unités (c'est-à-dire les neurones) pendant la phase d'entraînement d'un certain ensemble de neurones choisis au hasard, ce qui oblige un réseau de neurones à apprendre des caractéristiques plus robustes.

Nous utilisons la couche *pooling* pour réduire les dimensions spatiales. En ayant moins d'informations spatiales, nous pouvons accélérer le calcul. Moins d'informations spatiales signifie également moins de paramètres, donc moins de risque de surapprentissage.

Dans la *K-fold cross validation*, l'échantillon original est partitionné au hasard en k sous-échantillons de taille égale. Dans les k sous-échantillons, un seul sous-échantillon est retenu comme données de validation pour tester le modèle et les k-1 sous-échantillons restants. Le processus de validation croisée est ensuite répété k fois. L'avantage de cette méthode par rapport au sous-échantillonnage aléatoire répété est que toutes les observations sont utilisées à la fois pour la formation et la validation, et chaque observation est utilisée pour la validation une seule fois. Le résultat est plus convaincant.

3.2 Résultat

Voici les résultats obtenus. Nous avons divisé les données en 10 sous-données, après des heures de training, nous avons obtenu un meilleur score de 0.980, et le score moyen est de 0.968.

Nous pouvons voir à partir des données obtenues, la prédiction est très correspondre aux commentaires dans le fichier test.

```
epoch: 1 | loss: 0.24226313829421997 | score: 0.9457322498118996
epoch: 2 | loss: 0.11190773546695709 | score: 0.955419264610652
epoch: 3 | loss: 0.07162892818450928 | score: 0.9666263541356948
epoch: 4 | loss: 0.06313909590244293 | score: 0.9661902232355523
epoch: 5 | loss: 0.04353374242782593 | score: 0.9686514568265266
epoch: 6 | loss: 0.044561441987752914 | score: 0.9741479369260243
epoch: 7 | loss: 0.0416451171040535 | score: 0.9743568309253252
epoch: 8 | loss: 0.044528160244226456 | score: 0.9738868851686101
epoch: 9 | loss: 0.038383953273296356 | score: 0.9756414374956327
epoch: 10 | loss: 0.029537569731473923 | score: 0.9801490998557965
---------------------------------------------------------------
Best score is: 0.9801490998557965 , Average score is: 0.9680801738991714
train complete
total train time:  7.16752675751845 h
```

FIGURE 10 – Résultat

	id	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	00001cee341fdb12	0.977194	0.246823	0.874671	0.036069	0.756161	0.151352
1	0000247867823ef7	0.002163	0.000036	0.000333	0.000090	0.000308	0.000165
2	00013b17ad220c46	0.046709	0.000592	0.005277	0.001469	0.006918	0.002441
3	00017563c3f7919a	0.001579	0.000028	0.000197	0.000059	0.000236	0.000106
4	00017695ad8997eb	0.005452	0.000098	0.000817	0.000241	0.000833	0.000416

FIGURE 11 – Contenu des données dans le résultat

	id	comment_text
0	00001cee341fdb12	Yo bitch Ja Rule is more succesful then you'll...
1	0000247867823ef7	== From RfC == \n\n The title is fine as it is...
2	00013b17ad220c46	" \n\n == Sources == \n\n * Zawe Ashton on Lap...
3	00017563c3f7919a	:If you have a look back at the source, the in...
4	00017695ad8997eb	I don't anonymously edit articles at all.

FIGURE 12 – Contenu des données dans test

4 Conclusion

En comparant avec les résultats sur Kaggle, le score est plutôt bon. Mais nous avons encore quelques aspects à améliorer.

En raison des limites du matériel, nous avons testé seulement certains des paramètres. Même si nous avons utilisé Google Colaboratory, le temps maximal d'utilisation est de 12 heures, donc nous ne pouvons pas définir une dimension des données ou du réseau de neurones trop grande, sinon nous n'aurions pas pu terminer le training dans les 12 heures. Voici quelques aspects que nous pourrions améliorer.

1. Augmenter le nombre de couches cachées, rendant le modèle plus performant et puissant.

2. Augmenter la dimension des représentations vectorielles de mots (GloVe), par exemple, définir la dimension à 200 ou même 300. Cela peut améliorer la précision du résultat.

3. Pour aller plus loin, nous pouvons utiliser la méthode de Transfer Learning pour accélérer le processus d'apprentissage.

Références

[1] *Understanding LSTM Networks.*

[2] Jeffrey Pennington, Richard Socher, and Christopher Manning. *Glove : Global vectors for word representation.* 2014.