BLOUIN Maxime
PETIT Clément

Toxic Comment classification

Contexte du Projet :

La détection et la labélisation des commentaires toxiques sont devenues essentielles dans le monde des réseaux sociaux pour filtrer les commentaires des utilisateurs et protéger l'intégrité des entreprises ou des utilisateurs.

Par exemple, des plateformes comme YouTube empêchent la publication de vidéos contenant des propos contraires à leur charte. Les commentaires sous les vidéos sont également filtrés dans ce but. Cela vise à garantir un environnement sûr pour les utilisateurs, notamment les enfants, tout en préservant l'image de l'entreprise vis-à-vis des annonceurs.

Pour ce projet, nous nous sommes basés sur les ensembles de données Toxic Comment classification challenge disponibles sur Kaggle.

Etude du jeu de donnée :

Le jeu de donnée sur lequel nous allons travailler est composé des colonnes :

Id: Cette colonne nous donne l'identifiant de chaque commentaire, il est clair que cette colonne n'est pas en révélateur de la vélocité d'un commentaire. Nous avons donc décidé de la supprimer.

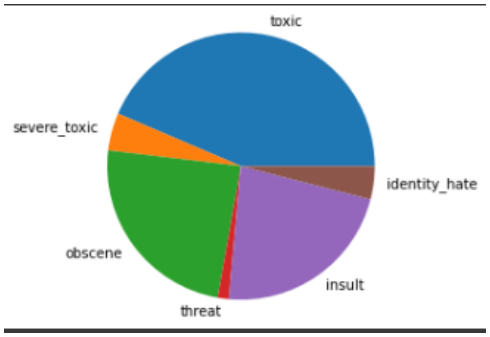
- Comment_text : contient les commentaires soumis à l'analyse.
- Toxic, Severe_toxic, Obscene, Insult, Identity_hate : ces colonnes indiquent si un commentaire est toxique, extrêmement toxique, obscène, une insulte, ou haineux envers une identité, avec des valeurs de 1 pour oui et 0 pour non.
- Id : identifiant de chaque commentaire, non pertinent pour l'analyse et donc supprimé.

Nous avons constaté que la catégorie "threat" ne contenait pas suffisamment de données pour être significative. Par conséquent, nous avons supprimé cette colonne et regroupé les commentaires menaçants sous la catégorie "Insult". La majorité des commentaires sont considérés comme sains (environ 80%) dans notre jeu de données, qui contient environ 160 000 entrées.

Comment avons-nous procédé :

Nous avons d'abord analysé la répartition de nos données pour identifier les problèmes, en particulier concernant la catégorie "threat".

BLOUIN Maxime
PETIT Clément



Grâce à ce plot nous avons très vite compris que la catégorie threat allait nous poser un problème. Surtout dans la partie tri de donnés pour exécuter notre modèle sur une partie moins importante de notre modèle.

Une fois les colonnes « threat » et « id » supprimées, nous avons dû traiter nos commentaires en modifiant les abréviations, en supprimant les stopwords, en passant les lettres majuscules en minuscules, etc.

Cette étape permet d'améliorer la qualité de nos données et donc de rendre notre futur modèle plus performant. Pour cela nous utilisons les fonctions clean et convert_abbrev_in_text.

.

```
[14] def clean(tweet):

        # Contractions
        tweet = re.sub(r"he's", "he is", tweet)
        tweet = re.sub(r"there's", "there is", tweet)
        tweet = re.sub(r"We're", "We are", tweet)
        tweet = re.sub(r"That's", "That is", tweet)
        tweet = re.sub(r"won't", "will not", tweet)
        tweet = re.sub(r"they're", "they are", tweet)
        tweet = re.sub(r"Can't", "Cannot", tweet)
        tweet = re.sub(r"wasn't", "was not", tweet)
        tweet = re.sub(r"don\x89Ût", "do not", tweet)
        tweet = re.sub(r"aren't", "are not", tweet)
        tweet = re.sub(r"isn't", "is not", tweet)
        tweet = re.sub(r"What's", "What is", tweet)
        tweet = re.sub(r"haven't", "have not", tweet)
        tweet = re.sub(r"hasn't", "has not", tweet)
        tweet = re.sub(r"There's", "There is", tweet)
        tweet = re.sub(r"He's", "He is", tweet)
        tweet = re.sub(r"It's", "It is", tweet)
        tweet = re.sub(r"You're", "You are", tweet)
        tweet = re.sub(r"I'M", "I am", tweet)
        tweet = re.sub(r"shouldn't", "should not", tweet)
        tweet = re.sub(r"wouldn't", "would not", tweet)
        tweet = re.sub(r"i'm", "I am", tweet)
        tweet = re.sub(r"I\x89Ûm", "I am", tweet)
        tweet = re.sub(r"I'm", "I am", tweet)
        tweet = re.sub(r"isn't", "is not", tweet)
        tweet = re.sub(r"Here's", "Here is", tweet)
        tweet = re.sub(r"you've", "you have", tweet)
        tweet = re.sub(r"you\x89Ûve", "you have", tweet)
        tweet = re.sub(r"we're", "we are", tweet)
        tweet = re.sub(r"what's", "what is", tweet)
        tweet = re.sub(r"couldn't", "could not", tweet)
        tweet = re.sub(r"we've", "we have", tweet)
        tweet = re.sub(r"it\x89Ûs", "it is", tweet)
        tweet = re.sub(r"doesn\x89Ût", "does not", tweet)
```

Pour entrainer notre modèle nous créons en premier lieu un bag of word car nous avons supposé dans un premier temps que la fréquence des mots était plus importante que leur sens dans une phrase. Cela était peut-être une erreur, il serait donc intéressant d'utiliser d'autres techniques qui permettraient de garder le sens de la phrase.

```
] #tokenization des variables
  from tensorflow.keras.preprocessing.text import Tokenizer
  tok = Tokenizer(num_words=1000, oov_token='UNK')
  tok.fit_on_texts(text+text_test)
```

Nous avons ensuite préparé les données en les convertissant en séquences numériques et en les normalisant pour une utilisation efficace dans notre modèle.

La première ligne utilise la fonction texts_to_sequences de l'objet "tok" pour convertir les textes en séquences de nombres. Les séquences sont générées en attribuant un entier

BLOUIN Maxime
PETIT Clément

```
#on normalise notre jeu de donnée
x_train = tok.texts_to_sequences(text)
x_test = tok.texts_to_sequences(text_test)
# print(x_test)
training_padded = pad_sequences(x_train,
                                maxlen=50,
                                truncating='post',
                                padding='post'
                               )
# #tst_texts
test_padded = pad_sequences(x_test,
                            maxlen=50,
                            truncating='post',
                            padding='post'
                               )


tr_X, val_X, tr_y, val_y = train_test_split(training_padded, y_train, train_size=0.90, random_state=30)
```

unique à chaque mot unique dans les textes, puis en convertissant chaque texte en une séquence d'entiers correspondants aux mots.

Les lignes suivantes implémentent la fonction pad sequences, qui ajoute des zéros à la fin des séquences pour les normaliser. Cela est utile pour les algorithmes de traitement de séquences qui nécessitent des entrées de même longueur. Les paramètres "maxlen=50" spécifient que la longueur maximale d'une séquence sera de 50. Par la suite nous divisons nos données pour l'entrainement et l'évaluation.

```
#implementation de reseaux de neuronne
model_lstm = models.Sequential()
model_lstm.add(layers.Embedding(1000,128, input_length=50))
model_lstm.add(Bidirectional(layers.LSTM(32,activation='tanh')))
model_lstm.add(layers.Dense(256,activation='relu'))
model_lstm.add(layers.Dense(128,activation='relu'))
model_lstm.add(layers.Dense(5, activation='sigmoid'))
```

Notre modèle est basé sur un LSTM bidirectionnel avec une fonction d'activation tanh. La couche LSTM bidirectionnelle permet d'explorer la séquence de texte à la fois de manière avant et arrière, en prenant en compte le contexte des deux directions. Cela est très important car certaines phrases peuvent paraître insultante si l'on ne fait pas attention à certains mots qui peuvent ajouter de la nuance à la phrase.

Une fois notre modèle produit nous évaluons sa précision grâce au f1_score, nous avons décidé d'utiliser cette méthode d'évaluation car nous avons un grand nombre de données qui ne sont pas toxiques. Nous pouvons alors avoir une accuracy de 80% si notre modèle décide que tous les commentaires sont non toxiques. Le f1_score vérifie les faux positifs ce qui nous permettra d'avoir plus d'informations sur notre modèle.

Nous obtenons un f1_score moyen de 56% ce qui est loin d'être parfait.

Nous avons également mis en place une fonction pipeline pour prédire si une phrase est toxique, ce qui a révélé que notre modèle est efficace pour détecter les commentaires négatifs mais moins précis dans leur catégorisation, surtout en ce qui concerne le sarcasme ou les phrases ambiguës.

```
def pipeline(entry):
  text=[]
  text.append(entry)
  text[0]=clean(text[0])
  x=tok.texts_to_sequences(text)
  x_padded=pad_sequences(x,maxlen=50,truncating='post',padding='post')
  return model_lstm.predict(x_padded)
```

BLOUIN Maxime
PETIT Clément

Nos analyses

Suite à nos résultats, l'utilisation de transformers aurait pu être envisagée pour améliorer la capacité de notre modèle à saisir les relations entre les mots et à mieux comprendre les nuances dans les phrases, notamment le sarcasme ou les sens cachés.