Classification de tweets basée sur le transfer learning

ELBIDI Louai
Berkani Yacine

April 19, 2024

Introduction

Ce projet s'inscrit dans le cadre de l'apprentissage profond appliqué au traitement automatique du langage naturel, plus précisément à la classification de tweets selon leur intensité sentimentale. L'objectif est de développer un système capable de classer un tweet dans l'une des sept catégories de sentiments, utilisant les techniques de transfer learning à partir d'un modèle pré-entraîné.

1 Ressources du projet

Pour faciliter la reproduction de ce projet et l'accès aux ressources utilisées, les liens suivants sont fournis :

1.1 Données

Les données utilisées pour l'entraînement et l'évaluation de notre modèle peuvent être téléchargées à partir de Kaggle. Ces données incluent un ensemble de tweets avec annotations de sentiments qui ont été utilisés tout au long de nos expériences.

Lien vers les données :

- Kaggle Dataset: Twitter Entity Sentiment Analysis

1.2 Code source

Le code source de ce projet est disponible sur GitHub. Il comprend les scripts de prétraitement, d'entraînement du modèle, et de prédiction, ainsi que des notebooks Jupyter démontrant le flux de travail complet.

Lien vers le code source :

- GitHub Repository: projetNLP

2 Pré-traitements

Les données textuelles, particulièrement celles issues des réseaux sociaux comme Twitter, nécessitent un nettoyage minutieux et une préparation avant d'être utilisées pour l'entraînement des modèles de machine learning. Cette section détaille les étapes clés du pré-traitement des tweets dans notre projet.

2.1 Configuration initiale des outils de traitement

Pour commencer, nous utilisons le `TweetTokenizer` de NLTK, qui est spécialement conçu pour les tweets, permettant de mieux gérer les particularités de ce type de texte, comme les mentions et les hashtags. Nous employons également le `WordNetLemmatizer` pour ramener les mots à leur forme canonique, ce qui aide à réduire la dimensionnalité des données et à améliorer la performance des modèles. Les stopwords, ou mots à faible contenu informatif, sont filtrés, à l'exception de certains négatifs qui sont cruciaux pour l'analyse de sentiments.

```
tknzr = TweetTokenizer()
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english')) - set(('not', 'no', 'never', 'nor'))
```

2.2 Fonction de standardisation des tweets

La fonction `standardization` est au cœur de notre pipeline de pré-traitement. Elle applique plusieurs transformations pour nettoyer et standardiser le texte : - Suppression des URL et des symboles spéciaux. - Remplacement des abréviations courantes par leur forme complète. - Retrait des mentions et des hashtags. - Conversion de toutes les numéros en espaces pour éviter les confusions numériques. - Utilisation de lemmatisation conditionnelle basée sur les tags POS pour assurer que les mots sont correctement ramenés à leur base.

```
def standardization(tweet):
    tweet = re.sub(r"\u2019", "'", tweet)
    tweet = re.sub(r"\u002C", ",", tweet)
    tweet = re.sub(r"(http|https)?:\/\/[a-zA-Z0-9\-.]+\.[a-zA-Z]{2,4}(/\S*)?", " ", tweet)
    tweet = re.sub(r"\bve\b", "have", tweet)
    tweet = re.sub(r"\bcan't\b", "cannot", tweet)
    tweet = re.sub(r"\b'n't\b", " not", tweet)
    tweet = re.sub(r"\bre\b", "are", tweet)
    tweet = re.sub(r"\bd\b", "would", tweet)
    tweet = re.sub(r"\b'll\b", "will", tweet)
    tweet = re.sub(r"\b's\b", " ", tweet)
    tweet = re.sub(r"\bn\b", " ", tweet)
    tweet = re.sub(r"\bm\b", "am", tweet)
    tweet = re.sub(r"@\w+", " ", tweet)
    tweet = re.sub(r"#\w+", " ", tweet)
    tweet = re.sub(r"[0-9]+", " ", tweet)
    tweet = tknzr.tokenize(tweet)
    tweet = [lemmatizer.lemmatize(i, j[0].lower()) if j[0].lower() in ['a','n','v']
             else lemmatizer.lemmatize(i) for i, j in pos_tag(tweet)]
    tweet = [i for i in tweet if (i.lower() not in stop_words) and (i.lower() not in punctuation)]
    tweet = ' '.join(tweet)
    return tweet
```

2.3 Nettoyage et préparation des datasets

Après avoir configuré nos outils et fonctions de nettoyage, nous procédons à la préparation des datasets. Nous chargeons les données, supprimons les colonnes et les lignes inutiles, et mappons les labels à des valeurs numériques pour une meilleure manipulation par les algorithmes d'apprentissage automatique.

```
df_3 = pd.read_csv("twitter_training.csv", delimiter=',', names=['ID', 'Game', 'label', 'tweet'])
df_3.drop(columns=['ID', 'Game'], inplace=True)
df_3.dropna(inplace=True)
df_3 = df_3[df_3['label'] != 'Irrelevant']

label_mapping = {'Positive': 1, 'Neutral': 0, 'Negative': -1}
df_3.loc[:, 'label'] = df_3['label'].replace(label_mapping)

tweets_train_3 = df_3['tweet'].apply(lambda x: standardization(x))
sentiments_train_3 = df_3['label']
```

Ces pré-traitements assurent que les données sont propres, normalisées et prêtes pour l'étape suivante, qui est la représentation vectorielle des mots et l'entraînement des modèles.

3 Représentation vectorielle des mots

Pour former nos modèles de réseau de neurones, il est crucial de convertir le texte en une forme numérique que les modèles peuvent interpréter. Cette section détaille la méthode utilisée pour générer des représentations vectorielles des tweets à partir des données textuelles prétraitées.

3.1 Indexation des mots

Avant de pouvoir utiliser les embeddings de mots, nous devons d'abord indexer notre vocabulaire, c'est-à-dire transformer chaque mot unique en un numéro d'index spécifique. Cela est réalisé avec le tokenizer de Keras, qui scanne tous les tweets et attribue un index unique à chaque mot.

```
word_index = tokenizer.word_index
print("Word Index:", word_index)
nb_words = len(word_index) + 1
```

3.2 Création de la matrice d'embeddings

Après l'indexation des mots, nous préparons une matrice d'embeddings où chaque ligne correspond à un mot et chaque colonne à une dimension de l'embedding. Les embeddings que nous utilisons proviennent du modèle Word2Vec de Google News, qui offre des vecteurs de 300 dimensions pour des millions de mots anglais.

```
EMBEDDING_DIM = 300
embedding_matrix = np.zeros((nb_words, EMBEDDING_DIM))

# Chargement du modèle Word2Vec
EMBEDDING_FILE = "word2vec-google-news-300.bin"
word2vec = KeyedVectors.load_word2vec_format(EMBEDDING_FILE, binary=True)
```

3.3 Intégration des embeddings

Chaque mot dans notre index reçoit son vecteur correspondant du modèle Word2Vec, si disponible. Si un mot n'est pas trouvé dans Word2Vec, nous lui attribuons un vecteur généré aléatoirement (représentant les mots hors vocabulaire, ou OOV).

```
oov = (np.random.rand(EMBEDDING_DIM) * 2.0) - 1.0
oov = oov / np.linalg.norm(oov)  # Normalisation du vecteur OOV

for word, i in word_index.items():
    if word in word2vec:
        embedding_matrix[i] = word2vec[word]
    else:
        embedding_matrix[i] = oov
```

3.4 Utilisation des embeddings

Ces embeddings sont ensuite utilisés comme la première couche dans nos modèles de réseaux de neurones, où chaque tweet est transformé en une séquence d'indices, qui sont à leur tour convertis en séquences de vecteurs via la matrice d'embeddings.

```
print("Exemple d'embedding pour le mot 'game':", word2vec['game'])
```

Cette méthode de représentation vectorielle nous permet non seulement de réduire la dimensionnalité des données textuelles mais aussi de capturer des informations sémantiques profondes sur les mots, ce qui est essentiel pour les tâches de traitement automatique du langage naturel telles que la classification de sentiment.

4 Entraînement des modèles

L'entraînement de modèles efficaces pour la classification de tweets repose sur la correcte configuration et optimisation des réseaux de neurones. Nous avons utilisé Keras, une bibliothèque de deep learning hautement modulaire, pour construire et entraîner nos modèles.

4.1 Modèle pour la classification à trois classes

Le modèle initial est conçu pour classifier les tweets en trois catégories de sentiment: positif, négatif et neutre. Il est composé de plusieurs couches qui transforment les entrées en prédictions.

4.1.1 Configuration du modèle

Nous commençons par définir une couche d'embedding qui transforme les indices des mots en vecteurs denses. Cette couche utilise des poids non entraînables pré-apprentis pour mieux capturer les relations sémantiques entre les mots.

```
embedding_layer = Embedding(input_dim=nb_words, output_dim=EMBEDDING_DIM,
                            trainable=False, name='embedding_layer')
model = Sequential()
model.add(embedding_layer)
```

Nous ajoutons ensuite une couche LSTM bidirectionnelle, qui permet au modèle de capturer des dépendances à long terme dans les deux directions du texte. Cela est suivi par une couche de Dropout pour réduire le surajustement.

```
model.add(Bidirectional(LSTM(64)))
model.add(Dropout(0.2))
```

La dernière couche est une couche Dense avec une activation softmax, qui distribue la probabilité à travers les trois classes de sortie.

```
model.add(Dense(3, activation='softmax'))
```

4.1.2 Compilation et entraînement du modèle

Le modèle est compilé avec l'optimiseur Adam et une fonction de perte de cross-entropy catégorique, idéale pour les tâches de classification multiclasse.

```
model.compile(loss='categorical_crossentropy',
              optimizer='Adam', metrics=['accuracy', Precision(), Recall()])
```

L'entraînement est effectué sur des données divisées en ensembles d'entraînement et de validation, permettant de surveiller et de prévenir le surajustement.

```
history = model.fit(x_train_3, y_train_3, validation_data=(x_val_3, y_val_3),
                    batch_size=64, epochs=20, verbose=1)
```

4.1.3 Résultats de l'entraînement

Le modèle atteint une précision satisfaisante sur l'ensemble de validation après plusieurs époques, montrant une amélioration continue de la capacité du modèle à classer correctement les sentiments des tweets.

4.2 Adaptation à la classification à sept classes

Pour étendre notre modèle à une classification plus fine en sept classes, nous réutilisons la structure et certains poids du modèle précédemment entraîné.

4.2.1 Modification du modèle

Le modèle pour sept classes est construit en reprenant la couche d'embedding et les couches LSTM du modèle à trois classes. Nous ajoutons des couches Dense supplémentaires pour augmenter la capacité du modèle à différencier entre un plus grand nombre de classes.

```
model_7 = Sequential()
embedding_layer = model_3.get_layer('embedding_layer')
model_7.add(embedding_layer)
model_7.add(Bidirectional(LSTM(64)))
model_7.add(Dropout(0.2))
model_7.add(Dense(150, activation='relu', name='dense1'))
model_7.add(Dense(64, activation='relu', name='dense2'))
model_7.add(Dense(7, activation='softmax', name='dense3'))
```

4.2.2 Compilation et entraînement

Ce nouveau modèle est également compilé avec une fonction de perte de cross-entropy catégorique et entraîné avec des paramètres similaires au modèle initial, mais ajustés pour la tâche plus complexe.

```
model_7.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=["accuracy"])
history = model_7.fit(x_train_7, y_train_7, validation_data=(x_val_7, y_val_7),
                      epochs=40, batch_size=25)
```

4.2.3 Observations finales

L'adaptation du modèle à sept classes montre des défis inhérents à la classification fine, avec une nécessité accrue de régularisation et d'ajustement des hyperparamètres pour maintenir une performance acceptable sans surajustement significatif.

5 Prédiction et évaluation

Une fois les modèles entraînés, la phase suivante consiste à utiliser ces modèles pour prédire la classe sentimentale des nouveaux tweets. Cette évaluation nous permet de mesurer l'efficacité de notre approche dans des conditions réalistes.

5.1 Chargement du modèle et préparation des données de test

Après avoir entraîné et sauvegardé notre modèle, nous le rechargeons pour l'utiliser dans l'évaluation. Les données de test sont préparées de manière similaire aux données d'entraînement, en utilisant le même pipeline de pré-traitement.

```
model_7 = load_model('model_7.h5')

df_test1 = "2018-EI-oc-En-anger-test-gold.txt"
df_test2 = "2018-EI-oc-En-fear-test-gold.txt"
df_test3 = "2018-EI-oc-En-joy-test-gold.txt"
df_test4 = "2018-EI-oc-En-sadness-test-gold.txt"

tweets_train_7_test1, sentiments_train_7_test1 = data_preprocessing(df_test1)
tweets_train_7_test2, sentiments_train_7_test2 = data_preprocessing(df_test2)
tweets_train_7_test3, sentiments_train_7_test3 = data_preprocessing(df_test3)
tweets_train_7_test4, sentiments_train_7_test4 = data_preprocessing(df_test4)
```

5.2 Fonction de prédiction

La fonction `predict_class` prépare et prédit le sentiment des tweets. Elle convertit le texte en séquences numériques, ajuste leur longueur à celle maximale observée, et utilise le modèle pour obtenir les prédictions. Voici un exemple de cette fonction en action :

```
def predict_class(text):
    max_len = 50
    xt = tokenizer.texts_to_sequences([text])
    xt = pad_sequences(xt, padding='post', maxlen=max_len)
    yt = model_7.predict(xt).argmax(axis=1)
    return yt
```

5.3 Évaluation et résultats

Pour évaluer notre modèle, nous avons utilisé des tweets de différentes catégories émotionnelles. Chaque catégorie a été testée pour évaluer l'efficacité du modèle à classifier correctement les sentiments.

Exemples de prédiction:

- Evaluation de la prédiction pour un tweet de la catégorie 'colère':

```
print("Tweet:", tweets_train_7_test1[0])
print("Label réel:", sentiments_train_7_test1[0], "Prédiction:", predict_class(tweets_train_7_test
```

- Les résultats des tests pour les autres émotions montrent également une bonne capacité du modèle à différencier les sentiments subtils comme la joie et la tristesse.

Ces résultats montrent que le modèle peut effectivement distinguer entre les différentes nuances émotionnelles exprimées dans les tweets, confirmant son utilité pratique pour des applications telles que l'analyse des médias sociaux et le ciblage marketing.