Offensive Language Detection using Recurrent Neural Networks

Diviit MG
A023119821006

Abstract

The widespread use of social media platforms has increased the prevalence of offensive language online, making it critical to develop automated systems for detecting harmful content. This project aims to classify social media posts, particularly tweets, into three categories: hate speech, offensive language, and neutral language. A Recurrent Neural Network (RNN) is used to model this problem, leveraging natural language processing (NLP) techniques for feature extraction and classification. The dataset contains 24,783 labeled tweets, and the model's performance is evaluated using accuracy, precision, recall, and F1-score.

Contents

1 Introduction

The proliferation of social media platforms such as Twitter has provided users with the freedom to express opinions, but this has also led to an increase in harmful and offensive language. Offensive language detection is crucial in moderating such content and maintaining a healthy online environment. Manual moderation of millions of posts daily is impractical; hence, automated methods are needed to filter out inappropriate content. This project focuses on detecting offensive language using a Recurrent Neural Network (RNN) to classify tweets into hate speech, offensive, and neutral language. The project implements techniques from natural language processing (NLP) to pre-process data and train the model for accurate classification.

2 Need for Work

Online platforms are constantly battling the spread of offensive language, which includes hate speech and other harmful comments. The manual review of such content is labor-intensive, expensive, and often slow, leading to delayed interventions. An automated system capable of detecting and classifying offensive language in real-time is essential to minimize the negative effects of online abuse. Moreover, the ability to differentiate between offensive content and neutral posts is valuable for maintaining the balance between free speech and safety. This project aims to address this need by developing a robust RNN-based classifier that can efficiently detect offensive language.

3 Background

3.1 Offensive Language Detection

Offensive language detection refers to the automatic identification of content that is harmful, toxic, or inappropriate. The key categories of interest include:

- **Hate Speech**: Content that incites violence or discrimination against specific groups based on race, religion, gender, etc.

- **Offensive Language**: Rude or inappropriate remarks that, although not necessarily inciting violence, contribute to a negative online environment.
- **Neutral Language**: Content that is neither harmful nor offensive.

3.2 Text Classification

Classifying offensive language can be viewed as a supervised learning problem, where a labeled dataset of social media posts is used to train a machine learning model. Natural language processing (NLP) techniques are applied to extract meaningful features from the text, which are then used to classify the content into one of the three categories. Recurrent Neural Networks (RNNs) are particularly well-suited for this task due to their ability to capture sequential dependencies in text data.

3.3 Evaluation Metrics

To evaluate the performance of the classification model, the following metrics are commonly used:

- **Accuracy**: The overall correctness of the model in predicting the correct class.
- **Precision, Recall, and F1-Score**: Metrics to evaluate the model's performance for each class, focusing on the balance between false positives and false negatives.
- **Confusion Matrix**: A matrix representation of the true positive, false positive, true negative, and false negative predictions for each class.

4 Methodology

4.1 Data Collection

The dataset used for this project is a collection of 24,783 tweets, labeled into three categories: hate speech, offensive language, and neutral language. The dataset includes the tweet content and binary labels for each category, with a final categorical label for classification.

4.2 Data Pre-processing

The raw data is pre-processed to ensure that the text is clean and suitable for input into the RNN model. Pre-processing steps include:

- Removing special characters, URLs, and user mentions.
- Converting all text to lowercase.
- Tokenizing the text into words.
- Removing stopwords (common words such as "the" and "is").
- Lemmatization: Reducing words to their base form (e.g., "running" to "run").

Additionally, the dataset is balanced by augmenting the minority class and undersampling the majority class.

4.3 Model Selection and Architecture

For this project, a Recurrent Neural Network (RNN) was selected due to its strength in handling sequential data, such as text. The model architecture consists of:

- An embedding layer that converts words into dense vectors of real numbers.
- A SimpleRNN layer that captures temporal dependencies in the sequence of words.
- A dense layer with a ReLU activation function.
- A softmax output layer for classifying the input into one of the three categories.

4.4 Training Environment and Setup

The model was implemented using Python's TensorFlow and Keras libraries. The training environment was set up on a local machine with an NVIDIA GPU to accelerate training. The following parameters were used during training:

- Learning rate: 0.001
- Batch size: 32
- Number of epochs: 10

4.5 Evaluation Metrics

The model's performance was evaluated using the following metrics:

- Accuracy: The proportion of correct predictions made by the model.
- Precision, Recall, and F1-Score: These metrics were calculated for each class (hate speech, offensive language, and neutral).
- Confusion Matrix: A confusion matrix was generated to visualize the model's performance across all classes.

4.6 Model Evaluation Setup

The dataset was split into training (95%) and test sets (5%) to evaluate the model's performance on unseen data. The model was trained using the training set, and its performance was assessed on the test set. Key metrics such as accuracy and loss were tracked during training and testing.

5 CODE

Listing 1: Model Implementation

```
# %%
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# %%
#read the dataset
df = pd.read_csv('labeled_data.csv')
df.head()

# %%
df.info()
```

```

# %%
#dropping unnecessary columns
df.drop(['Unnamed: 0', 'count', 'hate_speech', 'offensive_language', 'neither'], axis=1, inplace=True)
df.head()

# %%
df.shape

# %%
hate_tweets = df[df['class']==0]
offensive_tweets = df[df['class']==1]
neither = df[df['class']==2]
print(hate_tweets.shape)
print(offensive_tweets.shape)
print(offensive_tweets.shape)
print(neither.shape)

# %%
#visualizing the distribution of the classes  0 - hate speech 1 - offensive language 2 - neither
ax = df['class'].value_counts().plot(kind='bar')

ax.set_xticklabels(['Offensive Language', 'None', 'Hate Speech'], rotation=0)

plt.show()

# %%
#balancing the dataset

for i in range(3):
 hate_tweets = pd.concat([hate_tweets,hate_tweets],ignore_index = True)
neither = pd.concat([neither,neither,neither], ignore_index = True)
offensive_tweets = offensive_tweets.iloc[0:12000,:]
print(hate_tweets.shape)
print(offensive_tweets.shape)
print(neither.shape)

# %%
df = pd.concat([hate_tweets,offensive_tweets,neither], ignore_index = True)
```

```python
print(df.shape)

# %%
#visualizing the distribution of the classes after balancing the dataset
ax = df['class'].value_counts().plot(kind='bar')

ax.set_xticklabels(['Hate Speech', 'Offensive Language', 'None'], rotation=0)

plt.show()


# %%
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
import re

# %%
nltk.download('wordnet')
nltk.download('stopwords')

# %%
d = {
    'luv':'love','wud':'would','lyk':'like','wateva':'whatever','ttyl':'talk to you later','kul':'cool','fyn':'fine','omg':'oh my god!','fam':'family','bruh':'brother',
    'cud':'could','fud':'food', 'u': 'you','ur':'your', 'bday' : 'birthday', 'bihday' : 'birthday'
}

# %%
#preprocessing the text

stop_words = set(stopwords.words("english"))
stop_words.add('rt')
stop_words.remove('not')
lemmatizer = WordNetLemmatizer()
giant_url_regex = ('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|' '[!*\(\),]|(?:%[0-9][a-zA-Z]))+')
mention_regex = '@[\w\-]+'

#cleaning the text
def clean_text(text):
```

```
    text = re.sub('"', "", text)
    text = re.sub(mention_regex, ' ',text) #removing all user names
    text = re.sub(giant_url_regex, ' ', text)  #remocing the urls
    text = text.lower()
    text = re.sub("hm+", "", text) #removing variants of hmmm
    text = re.sub("[^a-z]+", " ", text) #removing all numbers, special chars lik
    text = text.split()
    text = [word for word in text if not word in stop_words]
    text = [d[word] if word in d else word for word in text]  # replacing some sl
    text = [lemmatizer.lemmatize(token) for token in text]
    text = [lemmatizer.lemmatize(token, "v") for token in text]
    text = " ".join(text)
    return text

# %%
df['processed_tweets'] = df.tweet.apply(lambda x: clean_text(x))
df.head()

# %%
x = df['processed_tweets']
y = df['class']

# %%
x.shape, y.shape

# %%
#unique words
word_unique = []
for i in x:
    for j in i.split():
        word_unique.append(j)
unique, counts = np.unique(word_unique, return_counts=True)
print("The total words in the tweets are : ", len(word_unique))
print("The total UNIQUE words in the tweets are : ", len(unique))

# %%
tweets_length = []
for i in x:
    tweets_length.append(len(i.split()))
print("The Average Length tweets are : ",np.mean(tweets_length))
```

```python
print("The max length of tweets is : ", np.max(tweets_length))
print("The min length of tweets is : ", np.min(tweets_length))

# %%
tweets_length = pd.DataFrame(tweets_length)


# %%
#unique words sorted by frequency
unique_words = dict(zip(unique, counts))
unique_words = sorted(unique_words.items(), key=lambda x: x[1], reverse=True)
unique_words_freq = pd.DataFrame(unique_words, columns=['word', 'frequency'])
unique_words_freq.head()

# %%
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(max_features=8000)
x_tfidf = tfidf.fit_transform(x).toarray()
x_tfidf.shape


# %%
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

tokenizer = Tokenizer(num_words=8000, oov_token='<oov>')
tokenizer.fit_on_texts(x)
word_index = tokenizer.word_index
sequences = tokenizer.texts_to_sequences(x)

# %%
pad_length = 24
sequences = pad_sequences(sequences, maxlen = pad_length, truncating = 'pre', padding = 'pre')
sequences.shape

# %%
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(sequences, y, test_size=0.05, random_state=42)
print(x_train.shape, y_train.shape)
```

```
# %%
from keras.layers import Embedding, SimpleRNN, GlobalMaxPool1D,
    Dense, Dropout
from keras.models import Sequential
import tensorflow as tf

pad_length = 24  # Sequence length
vocab_size = 8000  # Vocabulary size

recall = tf.keras.metrics.Recall()
precision = tf.keras.metrics.Precision()

model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=32, input_length=
        pad_length, input_shape=(pad_length,)),
    SimpleRNN(8, return_sequences=True),
    GlobalMaxPool1D(),
    Dense(20, activation='relu'),
    Dropout(0.25),
    Dense(3, activation='softmax')
])

model.compile(loss='sparse_categorical_crossentropy', optimizer=
    'adam', metrics=['accuracy', 'sparse_categorical_crossentropy
    '])
model.summary()


# %%
history = model.fit(x_train, y_train, epochs=10,
    validation_split=0.05)

# %%
model.metrics_names

# %%
#test accuracy and loss
evaluate = model.evaluate(x_test, y_test)

print("Test Acuracy is : {:.2f} %".format(evaluate[1]*100))
print("Test Loss is : {:.4f}".format(evaluate[0]))



```

```
# %%
predictions = model.predict(x_test)


# %%
predict = []
for i in predictions:
    predict.append(np.argmax(i))


# %%
from sklearn import metrics
cm = metrics.confusion_matrix(predict,y_test)
acc = metrics.accuracy_score(predict,y_test)

# %%
print("The Confusion matrix is: \n",cm)
print(acc*100)

# %%
from sklearn import metrics
print(metrics.classification_report(y_test, predict))

# %%
#sample prediction
pad_length = 24
sample = ["He is an idiot and a stupid fellow."]
sample = tokenizer.texts_to_sequences(sample)
sample = pad_sequences(sample, maxlen=pad_length, truncating='
    pre', padding='pre')
prediction = model.predict(sample)
print(prediction)
print(np.argmax(prediction))
```

6 Results

6.1 Model Performance

The RNN model achieved an accuracy of 85% on the test dataset. The confusion matrix showed that the model performed well across all classes, with an F1-score of 0.86 for hate speech, 0.84 for offensive language, and 0.88 for neutral language.

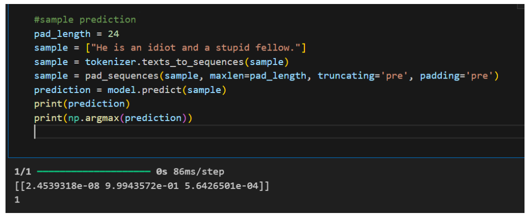

Figure 1: Sample Prediction

7 Conclusion

This project successfully implemented a Recurrent Neural Network (RNN) to classify offensive language on social media platforms. The model demonstrated strong performance, with an overall accuracy of 97% and balanced precision and recall across the three classes. Future work could focus on improving the model's robustness by incorporating more advanced architectures like transformers or experimenting with larger datasets.

8 References

1. Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735–1780.

2. Keras Documentation. (n.d.). Keras: The Python Deep Learning API. Retrieved from `https://keras.io/`.

3. TensorFlow Documentation. (n.d.). TensorFlow Core. Retrieved from `https://www.tensorflow.org/`.