Multi-class Sentiment Analysis using Deep Learning

Prerak Jayeshkumar Patel
Department of Computer Science
Lakehead University
Thunder Bay, Canada
Student Id: 1117675

***Abstract*—The dataset of movie reviews from Rotten Tomatoes is a collection of phrases and their corresponding sentiments in five different categories. We have used the Convolutional Neural Network for task of this sentiment classification. But for improving the efficiency of the model we have made use of preprocessing methods.**

I. Introduction

In the problem of sentiment analysis from text data, preprocessing of the text plays an important role in getting most efficient performance. There are multiple methods available for the preprocessing of the text data. Methods include lemmatization, tokenization, removal of stop words and punctuations. All of these methods have a sole motive of making the text more informative. Another method like TF-IDF (Term Frequency – Inverse Document Frequency) is used to decrease the sparsity of the data. It is used to remove the unwanted words which are not important for the particular domain. For example, in this dataset we have movie review's so the word "movie" has more frequency and has less importance. Whereas, the word like "good" or "bad" may have less frequency but more importance. This kind of understanding is helpful for the machine to learn and get more meaning from the data given as input. The model used is Convolutional Neural Network (CNN) for text classification and sentiment analysis.

II. Literature Review

The dataset of movie reviews from Rotten Tomatoes contains totally 156000 phrases in total. There are 5 sentiments labels used for all of them. They are numbered from 0 to 4 with negative, somewhat negative, neutral, somewhat positive and positive as their sentiments. The preprocessing methods used include, TF-IDF (Term Frequency – Inverse Document Frequency). This method is used to differentiate the words which appears frequently and has great importance with the words which appears frequently in all documents and hence has less importance in the particular domain. So, it is a simple mining technique to categorize the document and have a frequency count for each word. For getting an accurate computation of TF-IDF there has been some preprocessing done on the input text data like removal of most frequent words and rare words. Also, bi-grams made more sense in this dataset. The model used for text classification is Convolutional Neural Network (CNN) [1].

The preprocessing part of text plays a significant role in any Natural Language Processing system. Some of the most well-known text preprocessing methods include tokenization, lowercasing, multiword grouping and Lemmatization. There is a technique named Word2Vec to generate the word embedding for grouping a particular set of words. This technique is nothing but a pre-trained neural network with two layers which generates the linguistic context among words. Tokenization is a simple division of word units with a white space. The motive of tokenization is to separate every unit of the input text. Lowercasing is just converting the input text to l¬¬owercase. Lemmatization is a technique of converting the words into their respective lemma. This helps in reducing the sparsity of the input text but with a cost of sometimes ignoring the syntactic nuances. Having a great choice of selection among the preprocessing methods has always been a matter of conflict. Experimental results show that sometimes only a simple tokenization can generate great performance accuracy but, in any domain specific dataset like medical dataset used in the experiment, selecting only tokenization generated very poor results. So, sometime we need to combine multiple methods to get a good accuracy. The experimental results are generated for the best performing model for text classification. There has been, on an average, high variance of ±2.4%-symbol on the results depending on the selection of preprocessing methods. [2].

III. Proposed Model

The proposed model has a single-layered one-dimensional convolutional neural network. The preprocessing task is shown below.

```
import pandas as pd
from sklearn.model_selection import train_test_split

data = pd.read_csv("https://raw.githubusercontent.com/cacoderquan/Sentiment-Analysis-on-the-Rotten-Tomatoes-movie-review-dataset/master/train.tsv", sep = '\t')


# Import the numpy library to work with and manipulate the data
import numpy as np

# Check the head of the dataframe
data.head(20)

# check the shape of df
data.shape

# Get number of unique sentences
numSentences = data['SentenceId'].max()
```

```python
# extract full sentences only from the dataset
fullSentences = []
curSentence = 0
for i in range(data.shape[0]):
  if data['SentenceId'][i]> curSentence:
    fullSentences.append((data['Phrase'][i], data['Sentiment'][i]))
    curSentence = curSentence +1

len(fullSentences)

# put data into a df
fullSentDf = pd.DataFrame(fullSentences,
                                columns=['Phrase', 'Sentiment'])

# Check class imbalance in tokenized sentences
data['Sentiment'].value_counts()

# Check class imbalance in full sentences
fullSentDf['Sentiment'].value_counts()

import nltk
import random
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
from nltk.tokenize import word_tokenize

documents = []
# Use only complete sentences
for i in range(fullSentDf.shape[0]):
  tmpWords = word_tokenize(fullSentDf['Phrase'][i])
  documents.append((tmpWords, fullSentDf['Sentiment'][i]))

random.seed(9001)
random.shuffle(documents)
print(documents[1][0])

len(documents)

from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer, PorterStemmer, LancasterStemmer
porter = PorterStemmer()
lancaster=LancasterStemmer()
wordnet_lemmatizer = WordNetLemmatizer()
stopwords_en = stopwords.words("english")
punctuations="?:!.,;'\"-()"

#parameters to adjust to see the impact on outcome
remove_stopwords = True
useStemming = False
useLemma = True
removePuncs = True

for l in range(len(documents)):
  label = documents[l][1]
  tmpReview = []
  for w in documents[l][0]:
    newWord = w
    if remove_stopwords and (w in stopwords_en):
      continue
    if removePuncs and (w in punctuations):
      continue
    if useStemming:
      #newWord = porter.stem(newWord)
      newWord = lancaster.stem(newWord)
    if useLemma:
      newWord = wordnet_lemmatizer.lemmatize(newWord)
    tmpReview.append(newWord)
  documents[l] = (' '.join(tmpReview), label)
print(documents[2])

all_data = pd.DataFrame(documents,
                                columns=['text', 'sentiment'])
# Splits the dataset so 70% is used for training and 30% for testing
x_train_raw, x_test_raw, y_train_raw, y_test_raw = train_test_split(all_data['text'], all_data['sentiment'], test_size=0.3, random_state=2003)

len(x_train_raw)

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = TfidfVectorizer(stop_words="english", ngram_range=(1, 1))

x_train = vectorizer.fit_transform(x_train_raw)
y_train = y_train_raw
x_test = vectorizer.transform(x_test_raw)
y_test = y_test_raw

# Converts the datasets to numpy arrays to work with our PyTorch model
x_train_np = x_train.toarray()
y_train_np = np.array(y_train)

# Convert the testing data
x_test_np = x_test.toarray()
y_test_np = np.array(y_test)

x_train_np.shape
x_train_np = np.expand_dims(x_train_np, -1)

x_test_np = np.expand_dims(x_test_np, -1)
x_train_np.shape
```

Listing 1. Python code

The model and evaluation part is given below.

```python


from keras import backend as K

def recall_m(y_true,y_pred):
  true_positives = K.sum(K.round(K.clip(y_true*y_pred,0,1)))
  possible_positives = K.sum(K.round(K.clip(y_true,0,1)))
  recall = true_positives / (possible_positives + K.epsilon())
  return recall

def precision_m(y_true,y_pred):
  true_positives = K.sum(K.round(K.clip(y_true*y_pred,0,1)))
  predicted_positives = K.sum(K.round(K.clip(y_pred,0,1)))
  precision = true_positives / (predicted_positives + K.epsilon())
  return precision

def f1_m(y_true, y_pred):
  precision = precision_m(y_true, y_pred)
  recall = recall_m(y_true, y_pred)
  return 2*((precision*recall)/(precision+recall+K.epsilon()))

from keras.models import Sequential
```

```python
from keras.layers import Conv1D, MaxPooling1D, Dense
    , Flatten, Activation
from keras import optimizers
from keras import layers
from keras.utils import to_categorical

def cnn_model(fea_matrix, compiler):
  model = Sequential()
  model.add(Conv1D(filters = 16, kernel_size=2,
    activation='relu', input_shape = (fea_matrix.
    shape[1],fea_matrix.shape[2])))
  model.add(MaxPooling1D(pool_size=2))
  model.add(Conv1D(filters = 32, kernel_size=2,
    activation='relu'))
  model.add(MaxPooling1D(pool_size=2))
  model.add(Conv1D(filters = 64, kernel_size=2,
    activation='relu'))
  model.add(MaxPooling1D(pool_size=2))
  model.add(Flatten())
  model.add(Activation('relu'))
  model.add(Dense(5))
  model.add(Activation('softmax'))
  model.compile(optimizer = compiler, loss = '
    categorical_crossentropy', metrics = ['acc',
    f1_m, precision_m, recall_m])
  return model

model = cnn_model(x_train_np, optimizers.Nadam(lr =
    1e-3))

y_train_np = to_categorical(y_train_np)
y_test_np = to_categorical(y_test_np)
model.fit(x_train_np, y_train_np, batch_size=64,
    epochs = 30, verbose = 1, validation_split =
    0.3)
model.save("1117675_1dconv_reg")

loss, accuracy, f1_score, precision, recall = model.
    evaluate(x_test_np, y_test_np, verbose = False)

#get_metrics(accuracy,f1_score, precision, recall)
print('Accuracy:',np.round(accuracy,4))
print('Precision:',np.round(precision,4))
print('Recall:',np.round(recall,4))
print('F1 Score:',np.round(f1_score,4))
```

Listing 2. Python code

IV. Experimental Analysis

Selecting a perfect model for text classification always require some trial and error. The number of convolutional layers, the batch size, learning rate, number of epoch and the number of filters decide the efficiency of the model. After multiple combinations of these deciding parameters, the final combination of model is three convolutional layers with a filter size of 16,32,64 respectively. The activation function used everywhere is relu and softmax in the end before output layer because our problem is multiple class. The table below indicates the results for different number of epochs.

Epochs	Accuracy	Precision	F1 Score	Recall
100	0.3077	0.3092	0.3054	0.3019
30	0.3147	0.3176	0.3145	0.3116
10	0.3186	0.3167	0.3089	0.3127

Fig. 1. Block diagram of proposed model

V. Conclusion

The final accuracy, precision, recall and f1-score for our Convolutional Neural Network (CNN)model are 0.3077,0.3092,0.3019 and 0.3054 respectively. SO preprocessing helped in improving the performance.

References

[1] Sorostinean, Mihaela, Katia Sana, Mohamed Mohamed, and Amal Targhi. ”Sentiment analysis on movie reviews.” In Journal Agroparistech. 2017.

[2] Camacho-Collados, Jose, and Mohammad Taher Pilehvar. ”On the role of text preprocessing in neural network architectures: An evaluation study on text categorization and sentiment analysis.” arXiv preprint arXiv:1707.01780 (2017).