Multi-class Sentiment Analysis using Deep Learning

Nidhi Patel
Department of Computer Science
Lakehead University
Thunder Bay, Canada
Student Id: 1095526
Email: pateln@lakeheadu.ca

***Abstract*— There are several deep learning algorithms for solving the problem of text-based sentiment analysis such as Gated recurrent units (GRU), Recurrent neural network (RNN), Convolutional neural network (CNN), Long short-term memory (LSTM) and so on. In this paper, I have analyzed the data of Rotten Tomatoes Movie Reviews and proposed scalable and robust model based on Convolution Neural Network (CNN) to perform multi-class sentiment analysis. First, the data of the movie reviews is split into 70:30 ratio for training-testing data and then after data preprocessing and vectorization, convolutional neural network model performed on training data. The performance evaluation of the proposed model is done on testing data using Figure-of-Merit score, accuracy, recall and precision. The proposed model in this paper is able to achieve accuracy of 0.6229 on training set and 0.6215 testing set.**

Keywords—Multi-class sentiment analysis, Deep learning, Rotten Tomatoes, movie review, ConvNet, BoW, Word2Vec, TF-IDF, CNN

I. Introduction

Sentiment Analysis is one of the applications of the text classification. Text classification categorize the text according to subject categories, topics or genres which can be one or multiple. Most of the problems are based on binary classification but this paper provides the model for multi-class classification. The sentiment analysis is the process of identifying and categorizing the given text into the writer's contextual meaning or attitude towards the thing or topic which helps a business to understand the social sentiment of their brand, product or service [1].

The sentiment analysis is done through four steps. The first step of sentiment analysis is preprocessing of data in which text is converted to word sequences through tokenization, removal of stop-words, stemming and lemmatization. The second step is vectorization that transforms word sequences into the numerical features. Here, vectorization can be done by using one or combination of bag of words (BoW), Term Frequency Inverse Document Frequency (TF-IDF) and Word2Vec. The next step is to build the model and apply it on the training data. The final step is to apply the model on testing data and evaluate the model by getting the sentiment of given text.

In this paper, the proposed model is trained and tested on the dataset of Rotten Tomatoes Movie Reviews which is available on GitHub and the link for the dataset is https://raw.githubusercontent.com/cacoderquan/Sentiment-Analysis-on-the-Rotten-Tomatoes-movie-review-dataset/master/train.tsv. The dataset contains four rows: PhraseId, SentenceId, Phrase and Sentiment. Form these data, I have taken row 'Phrase' as input text and 'Sentiment' as target class. Here, sentiment is of five different classes from 0 to 4 and most of the text from the data is having sentiment class 2 as shown in figure 1:

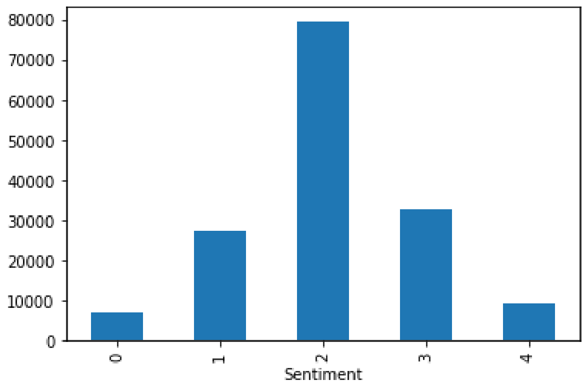

Fig. 1 Plots for each sentiment

II. Literature Review

The paper, Predicting Sentiment from Rotten Tomatoes Movie Reviews published by authors Jean Y. Wu and Yuanyuan Pao from Stanford University contains three different approaches for the problem of sentiment analysis. They used improved version of rotten tomatoes movie review dataset in year 2012 and applied Multinomial Naïve Bayes, Support vector machine and deep learning approach. In the deep learning approach, for the input layer, the authors fed word vector representation for various n-grams. The hidden layer of the model converts vector representation to the vector of dimension H and for the output layer they used logistic regression classifier. The authors achieved the best validation accuracy of 0.6167 and on the test set the accuracy achieved is 0.6130 [2].

Paper written by Suhariyanto, Ari Firmanto and Riyanarto Sarno on topic Prediction of Movie Sentiment based on Reviews and Score on Rotten Tomatoes using SentiWordnet contains the model for sentiment analysis by combining the sentiment score from SentiWordnet and expert original score. The authors used the rotten tomatoes dataset, but they did binary classification for sentiment. Their model included with preprocessing steps like remove punctuation and number, stop-words removal, lemmatization and tokenization, the feature extraction in which they used senti score from SentiWordnet and expert original score. For the sentiment classification, logistic regression was used. The result of 0.97 achieved by applying this method on for binary sentiment analysis [3].

Another paper I referred was Deep Neural Networks for multi-class sentiment classification published in IEEE 20th International Conference on High Performance Computing and Communications. In this paper, the authors model based on Convolutional Neural Network and Long short-term memory recurrent neural network. Their model used the two layers of CNNs to capture partial features and then for more accurate it was fed to the LSTM that can capture contextual information. Then, they combined the improved deep learning model with a one-versus-rest training mechanism and applied it for multi-class sentiment analysis. The model proposed by the authors was tested on the dataset D1 which was captured through Web crawler and has three categories. The accuracy of the model achieved on the dataset is 0.7842 [4].

III. PROPOSED MODEL

For the proposed model, the required libraries are imported. In the first step the dataset is loaded using "pandas" library form the GitHub link mentioned as above. The preprocessing on the dataset is performed. In the next step, vectorization is performed and then the convolutional neural network model is built and performed on training set. In the last step, model evaluation is done on testing data. Here, the evaluation is done in terms of accuracy, precision, recall and Figure-of-Merit (F-1) score.

A. *Preprocessing of data*

First, tokenization is done for which the "word_tokenize" library form nltk. tokenize is used. After that, removal of stop-words, stemming and lemmatization is done on the text in the dataset. For stemming, experimental analysis done on PorterStemmer, SnowballStemmer and LancasterStemmer. For lemmatization, WordNetLemmatizer is used. After preprocessing data is split into 70:30 ratio with random seed of 2003. The code for preprocessing on data is provided in Appendix A.

B. *Feature Vectorization*

Vectorization is done with purpose of converting word sequence to numerical feature for which used two different methods have been used and results are also compared. These two methods are Bag of Words and Tf-Idf. The code for vectorization of data is provided in Appendix B.

C. *Model Building*

For model building, convolutional neural network is used. The CNN model has been developed using "keras" library of python. For that, the defined model has been set as sequential and then convolutional layers with activation function as well as max pooling layers are added. Here, three convolutional layers using Conv1D are defined as well as three max pooling are used in the model. Here, different activation functions are used, and results are compared. Also, different kernel sizes are tested for prediction of class. After all convolutional layers flatten layer has been added [5].

Here, to prevent the neural network from the problem of overfitting, "Dropout" technique is used which randomly drop some hidden or visible units of neural network. Different values of dropout rates have been tested.

The softmax is applied on the last layer of the model as the dataset has five different classes for sentiment and softmax can be used for multi-class classification. The function for the convolutional neural network model defined as cnn_model is provided in Appendix C.

D. *Model Training*

The model is first fitted on training dataset. Class balance has been done on the training dataset. Here, different optimizers are used for experimental analysis. Also, if accuracy does not improve on training set then earlystopping is also used during training time to stop the model from training. The code for training model is in Appendix D.

E. *Performance Evaluation*

The evaluation of built model is done using library "backend" of python keras. The code includes three functions to calculate recall, precision and f-1 score as recall_value, precision_value and f1_score consecutively for multi-class sentiment which is provided in Appendix E-a, Appendix E-b and Appendix E-c.

IV. EXPERIMENTAL ANALYSIS

The proposed model for multi-class sentiment analysis is tested on different parameters during preprocessing, vectorization, model building and training. During the time of preprocessing of data two stemmers are compared for results. For vector transformation, BoW and Tf-Idf are used for comparison as well as different values of n-gram range are tried. Number of convolutional layers as well as dropout rates are also analyzed by increasing and decreasing it. Comparative analysis between activation functions is also performed for ReLU, tanh, softmax and sigmoid. Finally, during training time numbers of epochs, optimizers such as Adam, Nadam and SGD, size of validation set and batch size are used to differentiate results.

A. *Comparative analysis between stemmers*

The table 1 given as below describes the comparison between the stemmers without changing other parameters. From the table, it can be inferred that LancasterStemmer is having more accuracy.

Stemmer	**Accuracy**	**Precision**	**Recall**	**F-1 Score**
PorterStemmer	0.5151	0.5151	0.5151	0.5151
SnowballStemmer	0.6128	0.653	0.5403	0.5905
LancasterStemmer	0.6175	0.6553	0.5535	0.5992

Table 1: comparison between stemmers

B. *Comparative analysis between vectorizers*

The table 2 describes the comparison between the vectorization methods and n-gram range. From the table, we can conclude that Tf-Idf works better than Bag-of-Words.

Vectorizer	**ngram range**	**Accuracy**	**Precision**	**Recall**	**F-1 Score**
BoW	(1,2)	0.6091	0.6528	0.5316	0.5849
Tf-Idf	(1,1)	0.6174	0.6639	0.5375	0.5929
Tf-Idf	(1,2)	0.6192	0.66	0.5473	0.5974

Table 2: comparison between vectorizers

C. *Comparative analysis between Activation Functions*

The table 3 describes the comparison between various activation function applied on convolutional layers. Here, three one dimensional convolutional layers have been used.

Activation Function	**Accuracy**	**Precision**	**Recall**	**F-1 Score**
ReLU	0.6128	0.6775	0.5032	0.576
tanh	0.6167	0.6563	0.5488	0.5968
Sigmoid	0.5151	0.5151	0.5151	0.5151

Table 3: comparison between Activation Functions

D. *Comparative analysis for values of dropout rate*

The table 4 describes the comparison between values of dropout rate between layers to overcome overfitting problem.

Dropout Rate	Accuracy	Precision	Recall	F-1 Score
0.50	0.6167	0.6563	0.5488	0.5968
0.25	0.6181	0.6566	0.5473	0.596

Table 4: comparison between Dropout Rate

E. *Comparative analysis between optimizers*

The table 5 provides the comparison between various optimizers and learning rates used for the optimizers. The number of epochs is set to 100. Although, the model contains earlystopping which will stop the training the model early if performance does not improve. Also, the batch size is set to 256. However, values such as 128 and 64 are also tried for the batch size which resulted same as with batch size 256.

Optimizer	Learning Rate	Accuracy	Precision	Recall	F-1 Score
SGD	1e-3	0.5151	0.5151	0.5151	0.5151
Adam	0.0007	0.6111	0.6695	0.4889	0.5636
Adam	0.0001	0.6073	0.6753	0.474	0.5552
Nadam	1e-7	0.6054	0.645	0.5333	0.5829
Nadam	1e-3	0. 6215	0. 6577	0.5558	0.6016

Table 5: comparison between Dropout Rate

V. Conclusion

From the experimental analysis, we can conclude the all the stemmers performed well and provided almost same results. In addition, the Tf-Idf vectorization provided comparatively better result than BoW vectorization. Similarly, the tanh activation function came out as best among other activation functions and the Nadam optimizer proved to be perform best on the model. Finally, the proposed model performed best by using LancasterStemmer, Tf-Idf vectorizer, tanh activation function, dropout rate of 0.25 and Nadam optimizer with 1e-3 learning rate. The model built using convolutional neural network for multi-class sentiment analysis is able to give maximum accuracy of 0.6215.

VI. Appendix

A. *Data Preprocessing*

```
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer, Porter
Stemmer, LancasterStemmer

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

documents = []
for i in range(dataset.shape[0]):
  tmpWords = word_tokenize(dataset['Phrase'][i]
)
  documents.append((tmpWords, dataset['Sentimen
t'][i]))

#parameters to adjust to see the impact on outc
ome
remove_stopwords = True
useStemming = True
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(ne
wWord)
    tmpReview.append(newWord)
  documents[l] = (' '.join(tmpReview), label)
print(documents[0])
```

Listing 1: Tokenization, Stop-words removal, Lemmatization and Stemming

B. *Vectorization*

```
from sklearn.feature_extraction.text import Tfi
dfVectorizer

vector_size = 2500
vectorizer = TfidfVectorizer(max_features = vec
tor_size, stop_words = "english", ngram_range =
 (1, 1))
x_train = vectorizer.fit_transform(x_train_raw)
.toarray()
y_train = y_train_raw
x_test = vectorizer.transform(x_test_raw).toarr
ay()
y_test = y_test_raw

y_test = np.array(y_test)
y_test = y_test.reshape(y_test.shape[0],1)
```

Listing 2: Vectorization using Tf-Idf

C. *cnn_model*

```
def cnn_model(fea_matrix, n_class, compiler):#d
efining the cnn model
  #create model
  model = Sequential()# setting the model as se
quential one
  model.add(Conv1D(filters=128,
                   kernel_size=3,
                   activation='tanh',
                   input_shape=(fea_matrix.shap
e[1], fea_matrix.shape[2])))
  model.add(MaxPooling1D(pool_size=1))
  model.add(Dropout(rate = 0.5))
  model.add(Conv1D(filters=64,
                   kernel_size=5,
                   activation='tanh'))
  model.add(MaxPooling1D(pool_size=1))
  model.add(Dropout(rate = 0.5))
  model.add(Conv1D(filters=32,
                   kernel_size=5,
                   activation='tanh'))
  model.add(MaxPooling1D(pool_size=1))
  model.add(Dropout(rate = 0.5))
  model.add(Flatten())
  model.add(Dense(16, activation='tanh'))
  model.add(Dense(n_class, activation='softmax'
))
  model.compile(optimizer=compiler,
                loss='categorical_crossentropy”,
                 metrics=['acc',
                                     f1_score,
                                     precision_value,
                                     recall_value])

  return model
```

Listing 3: method cnn_model

D. *train cnn_model*

```
from keras import optimizers
from keras.callbacks import EarlyStopping
from sklearn.utils import class_weight

num_epochs = 100
batch_size = 256
decay = 1e-4
n_class = 5

#defining the different optimizers
adam = optimizers.Adam(lr = 0.0007)
sgd = optimizers.SGD(lr = 1e-3,
                     nesterov=True,
                     momentum=0.7)
Nadam = optimizers.Nadam(lr = 1e-3,
                         beta_1 = 0.9,
                         beta_2 = 0.999,
                         epsilon = 1e-08)

model = cnn_model(x_train, n_class, Nadam)
es = EarlyStopping(monitor='val_acc',
                   mode='max', patience=3)

model.fit(x_train, y_train,
          batch_size = batch_size,
          epochs = num_epochs,
          validation_split = 0.3,
          class_weight='auto',
          callbacks=[es])
```

Listing 4: training set on cnn_model

E. *Model Evaluation*

1) recall_value

```
#defining the recall method
def recall_value(y_true, y_pred):
  true_positives = K.sum(K.round(K.clip(y_t
rue*y_pred, 0, 1)))
  possible_positives = K.sum(K.round(K.clip
(y_true, 0 ,1)))
  recall = true_positives / (possible_posit
ives + K.epsilon())
  return recall
```

Listing 5: method recall_value

2) precision_value

```
#defining the precision method
def precision_value(y_true, y_pred):
  true_positives = K.sum(K.round(K.clip(y_t
rue*y_pred, 0, 1)))
  predicted_positives = K.sum(K.round(K.cli
p(y_pred, 0 ,1)))
  precision = true_positives / (predicted_p
ositives + K.epsilon())
  return precision
```

Listing 6: method precision_value

3) f1_score

```
#defining the f-1 score method
def f1_score(y_true, y_pred):
  precision = precision_value(y_true,
                              y_pred)
  recall = recall_value(y_true,
                        y_pred)
  return 2*((precision*recall)/(precision+r
ecall+K.epsilon()))
```

Listing 5: method f1_score

References

[1] https://towardsdatascience.com/sentiment-analysis-concept-analysis-and-applications-6c94d6f58c17

[2] Jean Y. Wu, Yuanyuan Pao, “Predicting Sentiment from Rotten Tomatoes Movie Reviews”, Stanford University, 2012.

[3] Suhariyanto, Ari Firmanto and Riyanarto Sarno on, “Prediction of Movie Sentiment based on Reviews and Score on Rotten Tomatoes using SentiWordnet”, International Seminar on Application for Technology of Information and Communication, 2018.

[4] Bohang Chen, Qiongxia Huang, Yi-Ping Phoebe Chen, Li Cheng and Riqing Chen, “Deep Neural Networks for multi-class sentiment classification”, IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th Intl. Conference on Data Science and Systems, 2018.

[5] https://adventuresinmachinelearning.com/keras-tutorial-cnn-11-lines/