Multi-class Sentiment Analysis using Convolutional Neural Network (CNN)

Darshak Mota (1101352)
Department of Computer Science
Lakehead University
Thunder Bay, Ontario
Email: {dmota}@lakeheadu.ca

I. Abstract

The paper demonstrates the use of Convolutional Neural Network (CNN) for multi-class classification for sentiment classification of movie reviews. Sentiment analysis simply means the interpretation of emotions, for example, positive, negative or neutral, in various domain using text analysis techniques. The Rotten Tomatoes sentiment dataset is used for analysing the sentiment of each review or comment provided by users. The model used for proposing this analysis is a Convolutional Neural Network (CNN) model and PyTorch framework is used to develop this model. A simple CNN model is used to experiment the sentiment analysis of users reviews or comments. In the end, the experiment provides moderately successful results which in turn will be helpful for classifying the future reviews or comments in correct sentiment.

II. Introduction

Multi-class classification is the task of classifying the instances into more than two classes or categories i.e. it outputs three or more features. Sentimental analysis is categorising the instances into different emotions (positive, negative or neutral) [1]. Basically, whenever a person needs to purchase any product online, it is obvious that he/she wants to get the reviews of this product before buying it. This not only helps the user to purchase a quality product but also helps the online vendor to get good range of customers. Similarly, such sentimental analysis is needed in many different domains or platforms. Problems like predicting the rating value of restaurants or movies, can be a tedious task if done manually by human calculations, hence to evaluate this thing a Machine Learning based sentiment analysis could help to get an appropriate result based on human components like emotions, opinions or feelings.

Sentiment analysis is done in three different levels, aspect level, document level and sentence level [1]. Document aspect can be defined whether a document's or a record's emotion is positive, negative or neutral. And as the name suggests, sentence level sentiment analysis is based on a sentence or phrase, and comprehends if the emotion of a sentence is positive, negative or neutral. This paper is based on sentiment analysis on movie reviews, hence it is a sentence level analysis. One of the main goal for applying sentimental analysis is to classify the Rotten Tomatoes phrases through machine learning of Natural Language Processing (NLP) approach which saves human from evaluating the sentiment manually [2].

The Rotten Tomatoes dataset is used to perform the sentimental analysis on the reviews or comments made by user, I haven't normalized the labels but categorising each review in its appropriate sentiment can be interpreted as shown in following Table. I.

Sentiment	Label
Negative	0
Somewhat Negative	1
Neutral	2
Somewhat Positive	3
Positive	4

TABLE I: Sentiment Label Codes

The dataset comprises of four columns namely, PhraseId, SentenceId, Phrase and Sentiment. The dataset is quite complex as each phrase consists of a separate sentiment label. Moreover, some phrases may include sarcasm, language ambiguity and many other arguments which makes the task problematic for Natural Language Processing (NLP). Presently, there are 156060 rows, means 156060 number of phrases in this dataset which can be split into separate training and testing sets. Below Table. II shows an example of the dataset.

PhraseId	SentenceId	Phrase	Sentiment
1	1	A series of escapades...	1
2	1	A series of escapades...	2
3	1	A series	2
4	1	A	2
5	1	series	2
6	1	of escapades demonstrating...	2
7	1	of	2
8	1	escapades demonstrating...	2
9	1	escapades	2
10	1	demonstrating the adage...	2

TABLE II: Dataset Preview

III. Background

Many authors have successfully developed a sentiment analysis system for the Rotten Tomatoes movie review dataset. In this [1] paper, the authors have proposed implementation of n-gram method for Rotten Tomatoes movie review dataset. Three different machine learning approaches are used namely, Support Vector Machine (SVM), Naive Bayes (NB) and Maximum Entropy (ME). Pre-processing steps such as removal of stop words and word vectorizer is done. They have blended both Counter Vectorizer and TF-IDF Vectorizer using n-gram model. Maximum Entropy (ME) resulted to give the best result among the three supervised learning methods using combined unigram + bigram model. Overall, from their research, it is noticed that the accuracy of classification reduces by increasing the value of 'n' in n-gram.The results for unigram, bigram and tri-gram are better but accuracy decreases if used four-gram or further on. For all the three supervised algorithms, it is observed that using combination of unigram and bigram results the best accuracy individually.

The next paper [2] simply proposes the use of different machine learning techniques on the Rotten Tomatoes movie review dataset. They compare four different algorithms like Binned Multinomial Naïve Bayes, Multinomial Naïve Bayes, Baseline and Linear Regression. The main goal of the author is to prove that Multinomial Naïve Bayes works better than the Linear Regression model.

In this article [3], the author has successfully implemented the sentiment analysis using logistic regression with Onevs-Rest classifier. According to the author, this strategy is used for multi-label classification. The author uses various deep learning techniques for classifying the Rotten Tomatoes movie review dataset. Experiment is done using Recurrent Neural Network (RNN), Convolutional Neural Network (CNN) and CNN with Gated Recurrent Unit (GRU). For RNN, Long-Short-Term-Memory (LSTM) is used which produced accuracy of 71.94%.Overall, among all three of the above mentioned learning techniques, CNN gave the highest accuracy of 80.17%.

IV. Proposed Model

This report is based on experimenting the sentiment analysis on the Rotten Tomatoes dataset for movie reviews. Each phrase is associated to any one of the five mentioned sentiments in Table. I. It is a multi-class classification problem as there are more than two labels provided in this dataset. Presently, the dataset consists of 4 columns and 156060 rows. A sample preview can be seen in Table. II. This dataset is a complex and ambiguous dataset where phrases have different complexities. Some reviews might be sarcastic, negating or ambiguous in nature.

Sentiment analysis require some data pre-processing before training our model. Data pre-processing is nothing but cleaning the data which includes removing stop words, punctuation, lower casing all words, and stemming or lemmatization. Firstly, the dataset needs to be split into training and testing set. As required, I split the dataset using 'train_test_split', the training set consists of 70% and testing set contains 30% of the dataset. First, I randomly shuffle the complete dataset with 'random_state' as 2003 and then the training and testing sets are created separately. The actual code snippet is given in Appendix A. The number of rows for training set is 109242 and for testing set is 46818. Now, before training the our model, we need to first clean the dataset using various pre-processing techniques. The phrases are first lower cased and then we remove the stop words and punctuation. Next, we can either use stemming or lemmatizing, I have trained and testing the model using both of these methods and I decided to use stemming as it give me better results, usually using lemmatization is a good decision but sometimes stemming might come in handy. The actual code snippet is given in Appendix B. Next, I drop all the rows which consists of black or NaN values, these blank phrases won't affect our model's performance. After pre-processing, the training set consists of 108578 rows, hence it is noticed that there were many blank rows which were dropped. I haven't done any changes to my testing dataset yet, we keep it as it is.

Next step is to resample our training dataset because the dataset might be imbalanced and to make sure that our model is not biased. The following Table. III shows the phrases across sentiment labels.

Labels	Phrases
2	54977
3	23021
1	19191
4	6468
0	4921

TABLE III: Phrases associated to Label

It is necessary to resample the training set, otherwise our model might classify most of the phrases to the sentiment label '2', because that sentiment is associated most of the times. Hence, it is a better approach to down sample our training set and not to up sample it. I use the sklearns library to resample (down sample) the training set and I use the least label associated value. Hence, our model won't be biased to any label and a balanced model will be formed. The actual code snippet for resampling is given in Appendix C. The final step before creating our model is to vectorize our training set. Here, each words should be transformed into vector of word counts. Methods like CountVectorizer, TF-IDF Vectorizer and word2vec can be used to perform this task. In this experiment, I have used Count Vectorizer and TF-IDF Vectorizer, and chose the Count Vectorizer as it provides me better results. Usually, TF-IDF Vectorizer works better than Count Vectorizer. I tried out different n-grams and after observing the results, I decided to use unigram + bigram vectorization with max features set to 20000.

Now it's time to create our model for sentiment analysis. I have used Convolutional Neural Network (CNN) model using PyTorch framework for training this cleaned data. Initially,

there is one input layer, following a maxpooling layer. I experimented by adding multiple convolutional and linear layers but didn't increase my performance, hence decided to put two convolutional layers, then added a flatten layer, following a linear layer. I observed a lot of overfitting in the model, adding a dropout layer after linear layer to reduce overfitting and then lastly the output layer which throws out five features. The below Fig. 1 represents the CNN model used for training.

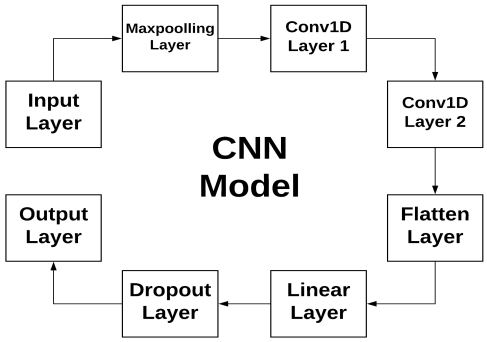

Fig. 1: CNN Model

The batchsize is set to 128 and the dropout rate is set to 0.2, the optimizer used is Adam with learning rate set to 1e-3 and 50 epochs are used for training the model. To evaluate the performance, various metrics are used namely, accuracy, recall, precision, F1 score and loss. All these metrics are imported from sklearns library except F1 score, I have manually calculated the f1 score. Cross-entropy loss is used to calculate the loss because it can be used for multi-class classification problem. Now that the model is trained, it's time to test our trained model. Similar to our training dataset, first we will pre-process the data in our testing dataset which includes tasks like removing stop words, lower casing all words, removing punctuation, and normalizing using stemming or lemmatization.

Below Fig. 2 shows the top accuracy and F1 score the model acquired with respect to epoch.

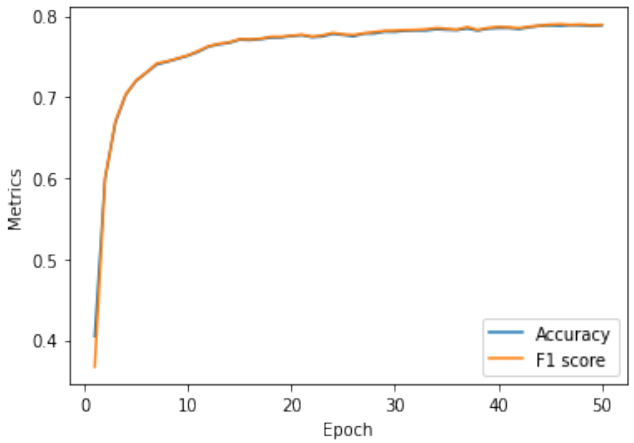

Fig. 2: Accuracy

Below Fig. 3 shows cross-entropy loss the model acquired with respect to epochs.

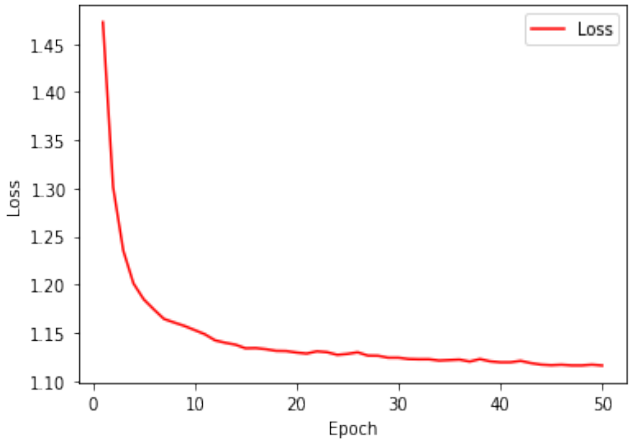

Fig. 3: Loss

Below Table. IV shows the results of all the various methods tried for sentiment analysis.

Normalization	Optimizer	Accuracy	F1 score
Porter Stemmer	Adam	0.52	0.46
Porter Stemmer	Adamax	0.51	0.46
Lancaster Stemmer	Adam	0.50	0.45
Lancaster Stemmer	Adamax	0.50	0.46
Lemmatization	Adam	0.53	0.41

TABLE IV: Phrases associated to Label

The top accuracy and other metrics for this proposed model is as shown in Table. V. The model's result is based on pre-processing with removing stop words and punctuation, Porter Stemmer, Count Vectorization with unigram + bigram, and Adam optimizer.

Metrics	Score
Accuracy	0.52
Precision	0.42
Recall	0.50
F1 Score	0.46
Loss	1.38

TABLE V: Top Metrics Score

V. Conclusion

As you can see from the above Table. IV, the best accuracy was provided by stemming and using the CountVector vectorizer combining the unigram + bigram. Though, the accuracy ranges between 0.51 and 0.53. I tried to tweak different methods of vectorizing, stemming or lemmatizing, changed the optimizers and its learning rate as well, also tried to change the batchsize and added convolutional layers and linear layers as well, but could get the best accuracy as 0.51 only. One thing

is observed that adding more number of convolutional layers or linear layers does not increase the performance at all. So it seems like the data pre-processing must be done properly to get an appropriate result. After reading few articles, it is noticed that using Recurrent Neural Network (RNN) would preferably increase the accuracy and give better sentiment analysis result. Also, using Convolutional Neural Network (CNN) with Gated Recurrent Unit (GRU) can be used to increase the metrics scores and accuracy. Another method can be used is word2vec vectorization, which might give a dramatic difference the our model and word2vec based model.

REFERENCES

[1] Tiwari, Prayag & Mishra, Brojo & Kumar, Vivek, "Implementation of n-gram Methodology for Rotten Tomatoes Review Dataset Sentiment Analysis", International Journal of Knowledge Discovery in Bioinformatics (IJKDB), 7. 12. 10.4018/IJKDB, March 2017.

[2] Kevin Hung, "Sentiment Analysis Classification for Rotten Tomatoes Phrases on Kaggle", Sept 2018.

[3] https://medium.com/@josephroy/multi-label-text-classification-rotten-tomatoes-f998bf54f81

APPENDIX

A. Split code

```
x_train, x_test, y_train, y_test =
    train_test_split(ds['Phrase'],
    ds['Sentiment'], test_size=0.3,
    shuffle='True', random_state=2003)
```

B. Pre-processing

```
X, Y = [], []

for y in range(len(train_ds)):
  label = train_ds['Sentiment'][y]
  temp_phrase = []

  space = str(train_ds['Phrase'][y])
              .lower().split(' ')

  for word in space:
    new_word = word

    if(word in stopwords_en):
      continue
    if(word in punctuations):
      continue

    if(use_stemming):
      new_word = porter_stemmer
                  .stem(new_word)
      # new_word = lancaster_stemmer
                  .stem(new_word)

    if(use_lemmatization):
      new_word = wordnet_lemmatizer
                  .lemmatize(new_word)
    temp_phrase.append(new_word)

  # if(not temp_phrase == []):
  X.append(temp_phrase)
  Y.append(label)
```

C. Resampling

```
y0 = ds_train[ds_train['Sentiment'] == 0]
y1 = ds_train[ds_train['Sentiment'] == 1]
y2 = ds_train[ds_train['Sentiment'] == 2]
y3 = ds_train[ds_train['Sentiment'] == 3]
y4 = ds_train[ds_train['Sentiment'] == 4]

y00 = resample(y0, replace = True,
n_samples = count, random_state = 2003)
y01 = resample(y1, replace = True,
n_samples = count, random_state = 2003)
y02 = resample(y2, replace = True,
n_samples = count, random_state = 2003)
y03 = resample(y3, replace = True,
n_samples = count, random_state = 2003)
y04 = resample(y4, replace = True,
n_samples = count, random_state = 2003)
```