Multi-class Sentiment Analysis using Deep Learning

Instructor: Dr. T Akilan
Dept. of MSc Computer Science
Professor at Lakehead University

Bhavik Subhashchandra Ray
Student ID: 111419
Lakehead University
Email: bsubhash@lakeheadu.ca

***Abstract*—This paper is presenting the powerful and expandable Convolutional Neural Network(CNN) to solve the issue of multi-class sentiment analysis using deep learning for the raw trained text based data of Rotten Tomatoes movie reviews. The model is supposed to predict the sentiment based on the trainable parameters and the accuracy which is based on key performance indicators(KPIs) such as figure-of-metrics(F1) score, accuracy, precision, recall and model loss. The highest accuracy for the proposed model is 61.3% on basis of precision, recall, F1 score, model loss and the train/test split ratio for the dataset was 70/30 with fulfilling the defined conditions.**

***Index Terms*—Multi-class sentiment analysis, 1D CNN, Pytorch, Deep learning.**

I. Introduction

The aim of Sentiment Analysis is to obtain the feelings of the writer by positive or negative reviews, questions and demands through analysed a large volume of documents. Sentiment analysis for deep learning can be defined as a way of feeding the algorithm with a massive amount of data so that it can adjust itself and continually improve. Sentiment analysis is contextual text mining that identifies and extracts subjective data in source material and helps a company know its brand, product or social service sentiment while tracking internet discussions. However, analyzing streams of social media is generally limited to analyzing fundamental feelings and counting based metrics.

Sentiment analysis can be modeled as classification problem where two sub-problems must be resolved:

- Classifying a sentence as subjective or objective, known as the definition of subjectivity.
- Classifying a sentence as expressing a positive, negative or neutral opinion, known as the description of polarity.

II. Literature Review

Sentiment analysis methods seek to derive from a text positive or negative emotion with words and identify the text as being optimistic, negative or otherwise neutral where no meaning can be identified with words. It can be considered a function of categorization of text in this regard. There are several groups of different topics of text grouping, although we only have 3 broad groups of sentimental analysis.

Challenges found in recent times for sentiment analysis:

- Implicit sentiment and sarcasm: Even without words of feeling, a sentence can have an Implicit sentiment.
- Domain dependency: There are many words whose polarity changes from domain to domain.
- Thwarted expectations: Often the author intentionally creates a meaning just to eventually refute.
- Pragmatics: The pragmatics of consumer sentiment are necessary to recognise, which can deeply modify the sentiments.
- Word knowledge: World information will regularly be used in the emotion detection method.
- Entity identification: There might be many individuals within a document or paragraph. The person to which the opinion is guided is extremely significant

III. Background

A. Process of sentiment analysis

In the sentiment analysis, first collection of the raw data from the web and cleaning the data by removing incomplete and error-ed data. After cleaning the data, pre-processing steps such as stemming, lemmatizing, labeling the data are done and it will generate test set from this processed data. Data training phase, feature extraction and data training is done. After that, deep learning model is created to classify the data. In the last phase, testing of the training dataset is done and from this phase accuracy of model is evaluated. These all phases can be easily understood from figure 1.

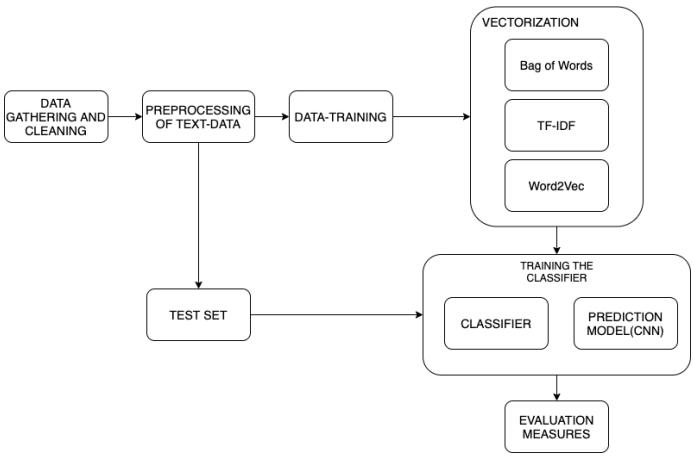

Fig. 1. Architecture of sentiment analysis.

B. Methods for vectorization

1) Bag of words(BOW): The simplest text vectorization technique. Bow converts text into the matrix of occurrence

of words within a document. Steps involved in BOW are as follows:

- Collect data and pre-process
- Design vocabulary
- Create document vectors

Limitations of BOW:

- Vocabulary: The vocabulary requires careful design, most specifically in order to manage the size, which impacts the sparsity of the document representations.
- Sparsity: Sparse representations are harder to model both for computational reasons (space and time complexity) and also for information reasons, where the challenge is for the models to harness so little information in such a large representational space.
- Meaning: Discarding word order ignores the context, and in turn meaning of words in the document (semantics).

2) Term frequency-inverse document frequency (TF-IDF): TF-IDF is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. The outcome is a combination of two metrics: how frequently a term occurs in a document, and the inverse variable frequency of the term across a list of documents.

For record search and material retrieval, TF-IDF has been invented. It operates by growing proportionally with the amount of times a term occurs in a document, however the number of documents including the term is compensated. So, words that are common in every document, such as this, what, and if, rank low even though they may appear many times, since they don't mean much to that document in particular.

Expression to find TF-IDF: To find the TF-IDF weights, there is a formula which is explained as below.

$$W_{i,j} = tf_{i,j} \times log(\frac{N}{df_i}) \tag{1}$$

Where,

$W_{i,j}$ = TF-IDF weight for token i in document j
$tf_{i,j}$ = Number of occurrences of token i in document j
df_i = Number of documents that has token i
N = Total number of documents in the training corpus

3) Word2Vec: Word embedding is one of the most popular representation of document vocabulary. It is capable of capturing context of a word in a document, semantic and syntactic similarity, relation with other words, etc. Word2Vec is one of the most popular technique to learn word embeddings using shallow neural network. It was developed by Tomas Mikolov in 2013 at Google. Word2Vec is a method to construct such an embedding. It can be obtained using two methods (both involving Neural Networks): Skip Gram and Common Bag of Words (CBOW) CBOW Model: This method takes the context of each word as the input and tries to predict the word corresponding to the context. Skip gram MODEL: Does the inverse and predicts the source context-words from the target words.

4) Word Embedding: : It is the vector, which reflects the structure of the word in terms of morphology (Enriching Word Vectors with Sub word Information). The concept behind the word embeddings is for the textual, morphological, meaning, hierarchical and some other elements to be caught by them, as far as feasible details. They are vector representations that capture the context of the underlying words in relation to other words in the sentence. This transformation results in words having similar meaning being clustered closer together in the hyper-plane and distinct words positioned further away in the hyper-plane.

IV. Proposed CNN Model

A. Dataset

In this work Rotten Tomatoes movie reviews dataset is used to train and test the model. This dataset is a trained dataset and it is a corpus of movie reviews used for sentiment analysis, originally collected by Pang and Lee. It contains total of 156060 data entries and those entries are having 4 feature columns like phrase id, sentence id, phrase and sentiment. The dimensionality of dataset is 4. Also, dataset parameters can be easily understood from the below table1 I. In addition to this, I have used given train/test split ratio of 0.3 and gave the random state value 2003 which fulfil the need of defined classification model.

TABLE I
Detail about attribute

Attribute Name	Data type	Description
Phrase Id	float64	It is the id for given phrase in dataset.
Sentence Id	int64	It is the id of given sentence.
Phrase	object	It is the collection of phrases for a sentence.
Sentiment	int64	Sentiment for a given sentence in range of 0 to 4.

B. Libraries

In order to do the multi class- sentiment analysis using 1D-CNN, I have used the pandas library to load the data and do some pre-processing part. In order to do mathematical calculation and converting the pandas data-frame to array I have used numpy. Finally, in order to develop the CNN model I have used the keras for this work. So, below is the list of library used:

- numpy
- sklearn
- pandas
- random
- nltk
- keras

C. Pre-processing steps for the dataset

1) Creating documents of only complete sentences: As this is a huge dataset, there are number of incomplete sentences are available which is not useful to predict the sentiment. Thus, I

generate a new document which have only 8544 sentences to perform further processing steps.

2) Removing stop words: In the next pre-processing step, I removed all the stop words from document of complete sentences using Natural Language Tool-kit(NLTK) library.

3) Stemming: Stemming is used to identify the root of the word. Here, I used lancaster stemmer to perform the operation of stemming.

4) Removing punctuations: In this step, I have remove all the punctuations from the documents of complete sentences. By doing this, dataset should be more understandable for our classification.

5) Tokenization: After cleaning and pre-processing the data, I have generated tokens of only complete sentences . Tokenizer is used to generate tokens from the raw text data.

6) Vectorization: As this data needs to be converted into vector form to perform mathematical operations and calculations. Here, I used the TF-IDF vectorization method as I believe it will perform better than other techniques.

D. Defining the 1D CNN model for classifier

In this portion, CNN classifier was created using Keras libraries and I also included dropout layer as it will overcome the over-fitting issues in code. Moreover, init() method was used to initialize the different layers and feed() method will create a neural network for our dataset. Furthermore, with the batch size of 64, kernel size=5, optimizer as ADAM and epochs=50 the model was trained. . The epochs is defined to train the model and a performance measure and optimizer is defined and once the data loader is defined the training loop is started.

The hyper parameters such as learning rate, epochs, batch size, optimizer, dropout rate, number of layers and used activation function for CNN model are described in the table II. Moreover, the actual code of CNN model is given in the appendix section VI.A.

TABLE II
HYPER PARAMETERS OF PROPOSED MODEL

Hyper-parameter	Value
Learning rate	0.01
Epochs	50
Batch size	64
Optimizer	ADAM
Dropout rate	0.5
Kernel size	5
Number of layers	2
Activation function	ReLu

E. Model saving procedure

In order to save the model, I mount my google drive with the Colab and then used the model.save() method to save my model with extension of h5 in keras. The code for this can seen in appendix VI.B.

F. Experimental results and comparison

We used the classifier in our experiments, and applied 4 (KPIs) for evaluating the classification: Accuracy, Precision, Recall, and F-measure:

- Accuracy: It is refers to the overall correctness of classification, measuring the ratio of correctly classified instances over the total number of instances.
- Precision: It is refers to the fraction of the tweets correctly classified, for a given sentiment, over the total number of tweets classified as belonging to that sentiment.
- Recall: It refers to the fraction of tweets correctly classified, for a given sentiment, over the total number of tweets actually belonging to that sentiment. In other words, for a single sentiment, this KPI is equivalent to its accuracy.
- F1-score: It is defined as following equation 2.

$$F1 = 2 \times \frac{(Precision \times Recall)}{(Precision + Recall)} \tag{2}$$

For this implementation ADAM optimizer is used to get the best accuracy for the proposed classification model and the comparison of this optimizer with other optimizer are discussed in comparison section. Moreover, best accuracy I got from proposed model was 0.613 and the results can be shown in appendix VI.C where batch size of 64, kernel size of 5, with dropout layer rate is 0.5 and learning rate of 0.01 is set to perform the model accuracy. Also, other metrics measure can seen as follows:

- Model's loss: 1.19
- Model's F1 measure: 0.60
- Model's precision: 0.64
- Model's recall: 0.57

G. Comparison between two approaches of optimizer

In the result analysis, I found that when Dropout Layer is used to handle the over-fitting issues it will reduce the accuracy of testing dataset. Also, ADADELTA, ADAGRAD and ADAM optimizer were used to measure which will give the best outcomes so from the below table we can see that ADAM optimizer outperforms the ADADELTA and ADAGRAD in both condition with different learning rates. From the table III, difference between different optimizers and batch size can be easily understood.

TABLE III
DIFFERENT RESULTS ANALYSIS

Optimizers	Accuracy without Dropout Layer	Accuracy with Dropout Layer(rate=0.5)
AdaDelta	0.60	0.59
AdaGrad	0.609	0.601
Adam	0.613	0.61

V. CONCLUSION

In the conclusion, with the different optimizer, changing the values of epochs,batch size and kernel size following the consideration of inference time, over-fitting issues, maximum number of trained dataset and train/test split ratio of 70/30 with

random state=2003 ADAM optimizer with batch size of 64, 50 epochs and kernel size of 5 have produced optimal result for this 1D CNN classifier. The results are 0.613 accuracy and loss of 1.19. Hence, goal for the aim is achieved in respect of all the fulfillment conditions of classifier model.

VI. APPENDIX

A. Proposed CNN model

```
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=5,
                 activation='relu',
                 input_shape=(2500,1)))
model.add(Conv1D(128, kernel_size=5, activation='relu'))
model.add(MaxPooling1D(pool_size=1))
model.add(Dropout(rate = 0.50))
model.add(Flatten())
# model.add(Dense(64, activation='relu'))
# model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

Fig. 2. Proposed model attributes with their values

B. Saving of model

```
from google.colab import drive
drive.mount('/content/drive')
```

```
model.save('/content/drive/My Drive/1111419_1dconv_reg.h5')
```

Fig. 3. Saving the model to drive

C. Result of testing model

```
Test loss: 1.192158264414942
Test accuracy: 0.613439275492332
Test F1 measure: 0.6014003699707544
Test precision: 0.6395768366343157
Test recall: 0.5686701695928916
```

Fig. 4. Best results among different approaches for defined model

REFERENCES

[1] https://raw.githubusercontent.com/cacoderquan/Sentiment-Analysis-on-the-Rotten-Tomatoes-movie-review-dataset/master/train.tsv
[2] https://www.researchgate.net/publication/236203597_Sentiment_Analysis_A_Literature_Survey
[3] https://medium.com/@hussein.sajid7/deep-learning-for-sentiment-analysis-7da8006bf6c1
[4] https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa