Multi-class Sentiment Analysis using Deep Learning

BIRVA PATEL
Department of Computer Science
Lakehead University 1111092
Thunder Bay, Ontario
bpatel24@lakeheadu.ca

***Abstract*—The aim of the project is to experiment with different Deep learning algorithms(mainly by CNN) to find the solution for the problem of text-based movie review multi-class sentiment analysis And returns the accuracy, recall, Precision and figure of merits(f-1) score.**

***Index Terms*—Convolutional neural network, Tf-Idf, tokenization, vectorisation,n-gram,Deep Learning, Neural Network**

I. Introduction

The human language is complex therefore teaching a machine to analyze the various grammatical nuances, cultural variations, slang and misspellings that occur in reviews provided by users is a difficult process. Teaching a machine to understand how context can affect tone is even more difficult. This method of sentiment analysis is useful in a wide range of domains, such as business or politics. [1] The purpose is to experiment different machine learning models for the task of sentiment analysis, using the Rotten tomatoes movie review corpus. More specifically, phrases from the dataset have to be classified in 5 categories, according to the intended tone of the user: negative, somewhat negative, neutral, somewhat positive, positive.

In this paper i will mainly focus on how you can predict the sentiment of movie reviews.in Python using the Keras deep learning library. This is a multi-class classification problem, which simply means the data set have more than 2 classes(binary classifier) that is 5.I am using raw train data of Rotten Tomatoes movie reviews.which is showed in fig-1, The dataset used is Pang and Lee's movie review sentiment polarity dataset, which consists of 5,331 positive and 5,331 negative sentences, formed from a vocabulary size of 20,000. For all the models, the data is shuffled and 30% of the dataset is used as the test set [2]. For that I can use several sequence handling

	PhraseId	SentenceId	Phrase	Sentiment
0	1	1	A series of escapades demonstrating the adage ...	1
1	2	1	A series of escapades demonstrating the adage ...	2
2	3	1	A series	2
3	4	1	A	2
4	5	1	series	2

Fig. 1. First five row of dataset.

components like GRU, RNN, LSTM but here i implemented it using convolution neural network.

TF-IDF (term frequency-inverse document frequency) is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. This is done by multiplying two metrics: how many times a word appears in a document, and the inverse document frequency of the word across a set of documents.

II. Literature Review

Early works of sentiment classification mainly focus on polarity detection of English product reviews or movie reviews.

Kennedy and Inkpen (2006) [3] present two methods for determining the sentiment expressed by a movie review. The first method uses GI to classify customers' reviews based on the number of positive and negative terms they contain, as well as negations, overstatements and understatements. Negations are used to reverse the semantic polarity of a particular term, while intensifiers and diminishers are used to increase and decrease, respectively, the degree to which a term is positive or negative. The second method uses a ML algorithm, SVM. Authors start with unigram features and then add bigrams that consist of a valence shifter and another word. The accuracy of classification is very high, and the valence shifter bigrams slightly improve it. They also demonstrate that combining the two methods achieves better results than either method alone.

The work in Prabowo and Thelwall (2009) [4] combines rule-based classification, supervised learning and machine learning into a new method. This method is tested on movie reviews, product reviews and MySpace comments. The procedure is that if one classifier fails to classify a document, the classifier will pass the document onto the next classifier, until the document is classified or no other classifier exists. A number of approaches that focus on acquiring and defining a set of rules are used along with SVM learning: General Inquirer Based Classifier (GIBC), Rule-Based Classifier (RBC), Statistics Based Classifier (SBC) and Induction Rule Based Classifier (IRBC). Experiments were carried out by applying different sequences of previous classifiers. Results showed that the use of multiple classifiers in a hybrid manner can improve the effectiveness of sentiment analysis. The sequence RBC then SBC then GIBC then SVM of classifiers yielded the highest accuracy in comparison to other classifiers' sequences.

As you can notice that most of the researchers tries to use the time sequence algorithms which are easy to handle the large data where in this project I am experimenting to use the deep learning algorithms like CNN and trying to get as much

high accuracy as I can. The data set is quite challenging as the dataset is split into separate phrases and each one of them have a sentiment label. Moreover, obstacles like sentence negation, sarcasm, terseness, language ambiguity, and many others make this task very interesting for Natural Language Processing.

here I have mentioned the complete path to my model that i have used fig-2 is the flow chart that exactly shows my workflow.

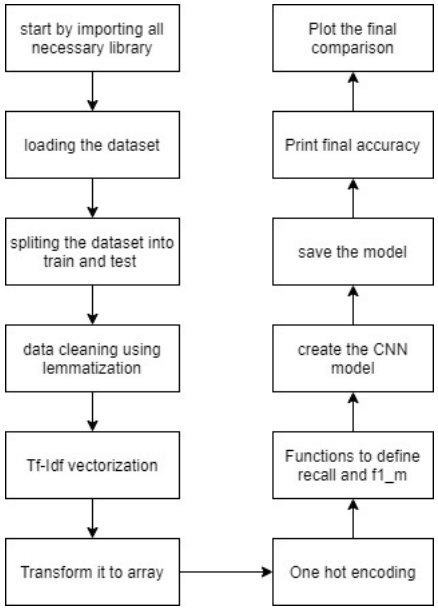

Fig. 2. Flow of the Proposed Model.

III. Proposed Model

- Import necessary libraries: As per appendix-1, we can call numpy for array operations, pandas for data visualization, sklearn for data analysis.
- Dataset: As per appendix-2, I load the rotten tomato's movie review analysis data from the GitHub link which is provided in the lab.Then I used pandas's 'read csv' function file to read the data from the given link into tabular format which is seprated by tab and store it in to fullSent variable as shown in appendix3. Now, I divide all the phrases column as an inputs and its sentiment column as an outputs.only full sentences are considered and stored in dataframe.
- Data preprocessing: As per appendix5,I removed all punctuations and stope words using Lemmatizer and stemmer
- Splitting dataset: As per appendix-6, I have split the dataset in training and testing data. training dataset is used,while we train the model and testing dataset is used while predicting the data. I use 70% for the training and 30% for the testing of whole dataset. train test split() function from sklearn used for that and as a parameter I set test size= 0.3.
- Vectorization: As per appendix 7, I used TfIdf vectorizer which convert phrases into bigrams. It removes the stop words in english langueages as well and make a list of words.Also I Convert train and test data into vectorize array and I transform the x train and x test into train idf and test idf respectively using vectorizer and then convert it into the array and reshape it into three dimension.
- One-hot encoding: As per appendix8, I convert all the labels into binary array using to categorical from the keras. for example, if an output is 6, it will convert it to the [0 0 0 0 0 0 1 0 0 0].
- Define necessary functions: As per appendix-9, define the function for recall which means true positive out of possible positives value, precision is true positives out of predicted positives and f1-score can be interpreted as a weighted average of the precision and recall.
- Convolution neural network(CNN) for multi-class classification: As per appendix-10, CNN class is created for the convolution operations. I initialize all the variables. I used convolution layers like conv1d and relu activation function for converting all the negative values into zero.After that,max-pooling is used for sub-sampling and then used flatten() to convert the output from CNN to feed forward network.then add dense layer for the five classes as the dataset has five classes.
- Compile the model:As per appendix-10, we compile our cnn model using adam optimizer and using 'reg' mode. we set the learning rate 0.001. number of classes are five.
- Train model: As per appendix-11, we train and test model using 10 epochs and batch size is 64. If number of epoch is more network is deep and accuracy will increase for certain number of epochs.
- Save the model: As per appendix-12, I saved the model as '1116576 1dconv reg.h5' using model.save() function.
- Test model: As per appendix-13, I test the model on the training data and get accuracy, precision,recall and f1-score and print these values up to two decimal points.
- Plot the accuracy graph: As per appendix14,define a plothist() function which plot the graph for training and testing accuracy per epochs.fig-3 denotes the accuracy graph for both training and testing dataset.

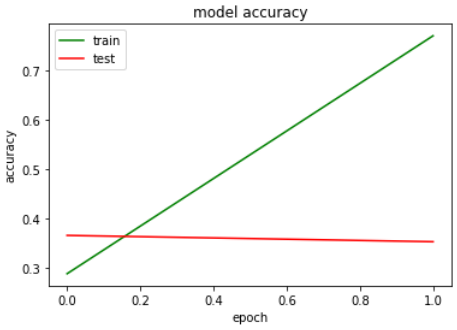

Fig. 3. Accuracy graph.

- plot the metrics graph: to show the training and testing value difference between the precision, recall and f1-score. I ploted this graph using same function.fig-4

indicates the difference of traning and testing values of the precision,recall and f1-score

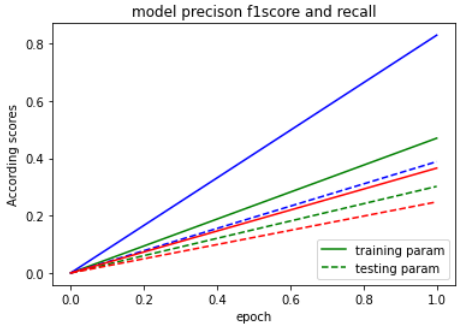

Fig. 4. precision,recall,f1-score.

By that way, The model's accuracy is 34% , precision is 35% , recall is 32% and f1-score is also 33%. My traing accuracy is quite high but testing accuracy is quite low because of the small dataset that I have used. The reason behind that is when i convert the data into corpus of words it will go out of memory utilization.

IV. Experimental Analysis

I have experimented on different models listed below in the table and it indicates the experimental results of the different models.

Layers	Accuracy	Precision	Recall	F1-score
CNN	34	35	32	33
Embedding	59	62	59	58

V. Appendix

```
# Import the pandas library to read our dataset
import pandas as pd

# Get the train/test split package from sklearn for preparing our dataset to
# train and test the model with
from sklearn.model_selection import train_test_split

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

Listing 1. importing library

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

train_data.head(5)
```

Listing 2. load dataset

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

Listing 3. get the full sentences

```
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])
```

Listing 4. separate the dataset

```
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 = True
useLemma = False
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])
```

Listing 5. Data Preprocessing

```
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)

len(x_train_raw)
```

Listing 6. Splitting the data

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

# Transform each text into a vector of word counts

vectorizer = TfidfVectorizer(stop_words="english",
                             ngram_range=(1, 2))
train_x = vectorizer.fit_transform(x_train_raw)


train_vectorized = vectorizer.transform(x_train_raw)
train_vectorized= np.array(train_vectorized.toarray
    ())
train_vectorized = train_vectorized.reshape(
    train_vectorized.shape[0],train_vectorized.shape
    [1],1)
print(train_vectorized.shape)

test_vectorized = vectorizer.transform(x_test_raw)
test_vectorized= np.array(test_vectorized.toarray())
test_vectorized = test_vectorized.reshape(
    test_vectorized.shape[0],test_vectorized.shape
    [1],1)
print(test_vectorized.shape)
```

Listing 7. Vectorization

```
from keras.utils import to_categorical
Y_train = to_categorical(y_train_raw)
Y_test = to_categorical(y_test_raw)
```

Listing 8. One Hot Vectorization

```
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()))
```

Listing 9. Define necessary functions

```
import keras
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout,
     Flatten, Conv1D, MaxPooling1D
from keras.layers.normalization import
    BatchNormalization
import numpy as np
from keras import optimizers
np.random.seed(1000)
def CNN_model(fea_matrix, n_class, mode, compiler):
  model = Sequential()
  model.add(Conv1D(filters=64, kernel_size=1,
    activation='relu',
                   input_shape=(fea_matrix.shape[1],
    fea_matrix.shape[2])))
  model.add(MaxPooling1D(pool_size=2))
  model.add(Conv1D(filters=128, kernel_size=1,
    activation='relu'))
  model.add(MaxPooling1D(pool_size=2))
  model.add(Flatten())
  model.add(Activation('relu'))
  model.add(Dense(n_class))

  model.add(Activation('softmax'))
  model.compile(optimizer=compiler,loss='
    categorical_crossentropy',
                metrics=['acc',f1_m,precision_m,
    recall_m])
  return model


lr = 1e-3
decay=1e-4
mode="reg"
n_class = 5

adm = optimizers.Adam(lr=lr,decay=decay)
model = CNN_model(train_vectorized,n_class,mode,adm)
```

Listing 10. Model Layout

```
#fit the model
x = model.fit(train_vectorized,Y_train,batch_size
    =64, epochs=10, verbose=1, validation_split=0.3)
```

Listing 11. Fit the Model

```
#save the model
model.save('1111092_1dconv_reg.h5')
```

Listing 12. Save the Model

```
#function call for accuracy f1-score, precision and
    recall
def metrics(accuracy,f1_score,precision,recall):
  print('CNN model performance')
  print('Accuracy:', np.round(accuracy,2))
  print('Precision: ', np.round(precision,2))
  print("recall:", np.round(recall,2))
  print("f1score:", np.round(f1_score,2))
loss, accuracy, f1_score, precision, recall = model.
    evaluate(test_vectorized, Y_test, verbose=False)

# get the accuracy,f1_score,presion,recall
metrics(accuracy,f1_score, precision, recall)
```

Listing 13. Test model

```
import matplotlib.pyplot as plt
def plothist(hist):
    plt.plot(hist.history['acc'],color='green')
    plt.plot(hist.history['val_acc'],color='red')
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()
plothist(x)
```

Listing 14. plot the accuracy graph

```
import matplotlib.pyplot as plt
def plothist(hist):
    plt.plot(hist.history['f1_m'],color='green')
    plt.plot(hist.history['val_f1_m'],color='green',
    linestyle='dashed')
    plt.plot(hist.history['precision_m'],color='blue
    ')
    plt.plot(hist.history['val_precision_m'],color='
    blue',linestyle='dashed')
    plt.plot(hist.history['recall_m'],color='red')
    plt.plot(hist.history['val_recall_m'],color='red
    ',linestyle='dashed')
    plt.title('model precison f1score and recall')
    plt.legend(['training param','testing param'],
    loc='lower right')
    plt.ylabel('According scores')
    plt.xlabel('epoch')
    plt.show()
plothist(x)

```

Listing 15. plot the functional graph

VI. CONCLUSION

To conclude,1D CNN works well with the text datasets and in proposed method you can identify that it was not much complex problem where as sequencing models works much better.To find the more accurate results dataset should have to be in large amount.still accuracy, Precision, f1-score, recall are measured from the available data. Also I explained the important method used for text summarization which is tf-idf.

REFERENCES

[1] M. Sorostinean, K. Sana, M. Mohamed, and A. Targhi, "Sentiment analysis on movie reviews," in *Journal Agroparistech*, 2017.

[2] J. Hong, A. Nam, and A. Cai, "Multi-class text sentiment analysis," 2019.

[3] A. Kennedy and D. Inkpen, "Sentiment classification of movie reviews using contextual valence shifters," *Computational intelligence*, vol. 22, no. 2, pp. 110–125, 2006.

[4] R. Prabowo and M. Thelwall, "Sentiment analysis: A combined approach," *Journal of Informetrics*, vol. 3, no. 2, pp. 143–157, 2009.