Assignment 2: Multi-Class Sentiment Analysis Using Deep Learning

Priya Patel (1093483)
Department of Computer Science
Lakehead University
Email: patelp@lakeheadu.ca

Abstract – Sentiment analysis is process of determining whether the data is positive, negative or neutral. It is also known as assigning the polarity to the data. It is the process of analysing the data using various techniques into the different emotions. In this research the main objective is to implement the Convolutional Neural Network for the text based movie reviews of the rotten tomatoes dataset.

Keywords – Convolutional Neural Network, Sentiment Analysis, Neural Networks, Sentiment, Phrase.

1 INTRODUCTION

Sentiment Analysis is the way to classify the data collected online into the different emotion classes. It can help in making the data structured using the NLP. Here the data is first pre-processed and then with the use of Conv and dense layers the model is created using which the prediction is made on the test data. The sentiment analysis is an interpretation and classification of emotions.

1.1 PROBLEM DEFINITION

Implement a scalable and robust Convolutional Neural Network-based solution for the problem of text-based movie review multi-class sentiment analysis.

Data: Use the raw train data of Rotten Tomatoes movie reviews available at https://raw.githubusercontent.com/cacoderquan/Sentiment-Analysis-on-the-Rotten-Tomatoes-movie-review-dataset/master/train.tsv into a Colab notebook.

Constraints:

- Model: The model cannot take advantage of advanced sequence handling components, viz. GRU, RNN, LSTM, and so forth. This assignment stays within the scope of Convolutional (Conv) and dense layers and the associated operations, like pooling (Ave, Max, etc.) and non-linear activation functions (ReLU, Sigmoid, tanh, Softmax, Leaky ReLU, etc.).
- Training: You are free to train your model for any n number of epochs. However, you must archived the training results you obtained in a period epoch, let's say for every 100 epochs. It is to verify the results one-to-one.
  - Features: This assignment lays on one or combination of BoW, TF-IDF, and Word2Vec only.
  - Data set: Split the loaded train.tsv Rotten Tomatoes Movie Reviews into this assignments' train and test sets with a ratio of 70:30 using sklearn.model selection library with random state is set to 2003.
  - Performance evaluation: It is based on the following metrics - Accuracy, Recall, Precision, and Figure-of-Merit (f-1) score.

2 LITERTURE REVIEW

2.1 What is sentiment analysis?

Sentiment analysis is the method used to retrieve the meaningful data from the raw data. The sentiment analysis can be carried out on various levels, including on an individual sentence level, paragraph level, or the entire document as well. It is the interpretation and classification of emotions i.e. positive or negative within the raw data. The rotten tomatoes movie review is a collection of movie reviews collected by Pang and Lee. Here each sentence is parsed into the tree structure and each node is assigned a fine-grained sentiment label ranging from 1 – 5 where the numbers represent very negative, negative, neutral positive and very positive.

3 FLOWCHART

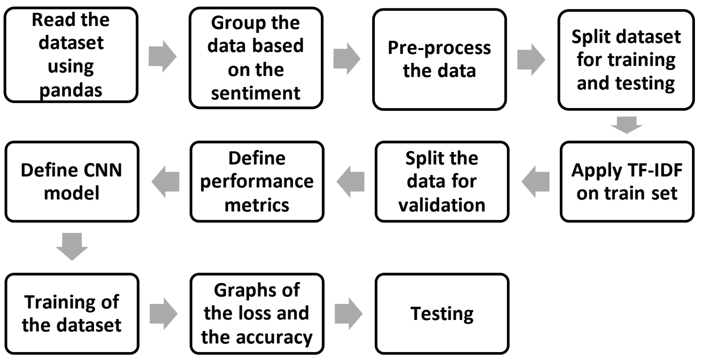
Read the dataset using pandas
Group the data based on the sentiment
Pre-process the data
Split dataset for training and testing
Apply TF-IDF on train set
Split the data for validation
Define performance metrics
Define CNN model
Training of the dataset
Graphs of the loss and the accuracy
Testing

Figure 1 Flowchart

4 RELATED WORK AND EXPERIMENT

Here the task is to perform the sentiment analysis on the movie reviews using the scalable and robust CNN.

1. Import libraries

Firstly, the important libraries required in the program are imported.

```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
import nltk
from nltk.tokenize import word_tokenize
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
import random
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer, PorterStemmer, LancasterStemmer
from keras.utils import to_categorical
from sklearn.feature_extraction.text import CountVectorizer , TfidfVectorizer
from keras import backend as K
from keras.models import Sequential
from keras.layers import Conv1D, MaxPooling1D, Dense, Dropout, Flatten,Embedding
from keras.layers import Activation, Conv1D, GlobalMaxPooling1D
from keras import optimizers
from keras import layers
import time
```

2. Import dataset

The dataset is imported from the given url using the read_csv function of the pandas.

```
URL_Train= "https://raw.githubusercontent.com/cacoderquan/Sentiment-Analysis-on-the-Rotten-Tomatoes-movie-review-dataset/master/train.tsv"
```

```
#load data
train=pd.read_csv(URL_Train,sep='\t')
# print(len(train))
train.head()
```

The first 5 samples of the data are as follows.

	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

3. Visualizing the data

The train data contains the 4 columns i.e. PhraseID, SentimentID, Phrase and Sentiment. The data used for the training and testing will be Phrase and Sentiment. The bar graph below shows the count of the Sentiment in the data.

It can be seen that the data contains maximum the neutral reviews.

```
phrase = train['Phrase']
sentiment = train['Sentiment']
sentiment.value_counts().plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc95b32a1
```

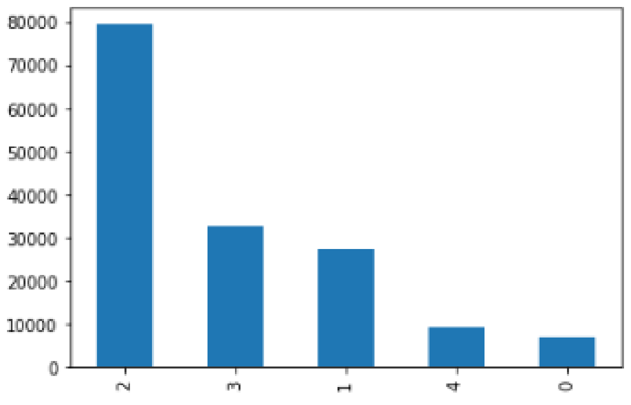

Figure 2 Bar graph of Sentiments

4. Group the data based on the sentiment

```
fullSentences = []
curSentence = 0
for i in range(train.shape[0]):
  if train['SentenceId'][i]> curSentence:
    fullSentences.append((train['Phrase'][i], train['Sentiment'][i]))
    curSentence = curSentence +1

# put data into a df
fullSentDf = pd.DataFrame(fullSentences,
                            columns=['Phrase', 'Sentiment'])
print(fullSentDf['Sentiment'].value_counts())
print(fullSentDf['Phrase'][1])
```

5. Split the dataset into the train and test data using the sklearn.model_selection train_test_spilt

```
X_train, X_test, y_train, y_test = train_test_split(train['Phrase'],
                                                    train['Sentiment'],
                                                    test_size=0.3,
                                                    random_state=2003)
```

To know the size of the train and test data

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(109242,)
(46818,)
(109242,)
(46818,)
```

6. Convert the text data into the numbers

The vector length is set to 5000 to avoid the running out of memory issue.

```
# Transform each text into a vector of word counts
vec_len = 5000
tfidf_vectorizer = TfidfVectorizer(stop_words="english",
                                   ngram_range=(1, 1),
                                   max_features = vec_len)

bow_vectorizer = CountVectorizer(stop_words="english",
                                 ngram_range=(1,1),
                                 max_features = vec_len)
```

7. Pre-processing the data

```
porter = PorterStemmer()
lancaster=LancasterStemmer()
wordnet_lemmatizer = WordNetLemmatizer()
StopWords = stopwords.words("english")
punctuations="?:!.,;'\"-()"
#parameters to adjust to see the impact on outcome
remove_stopwords = True
useStemming = True
useLemma = False
removePuncs = True
# print(StopWords)

for l in range(len(documents)): #For each review docu-ment
  # print(l)
  label = documents[l][1] #Save review label
  tmpReview = [] #Placeholder list for new review
  # train_new=[]
  for w in documents[l][0]: #For each word this is review
    newWord = w #Set newWork to be the updated word
    if remove_stopwords and (w in StopWords):#if the word is a stopword
      continue #skip the word and don’t had it to the normalized review
    if removePuncs and (w in punctuations):#if the word is a punc
      continue #skip the word and don’t had it to the normalized review
    if useStemming: #if useStemming is set to True
      #Keep one stemmer commented out
      #newWord = porter.stem(newWord) #User porter stemmer
      newWord = lancaster.stem(newWord) #Use Lancaster stemmer
    if useLemma:
      newWord = wordnet_lemmatizer.lemmatize(newWord)
    tmpReview.append(newWord) #Add normalized word to the tmp review
  documents[l] = (' '.join(tmpReview), label)
# print(documents[2]) #Update the reviews list with clean review
```

8. Fit transform the train and test data using TF-IDF

```
tfidf_vectorizer = TfidfVectorizer(stop_words="english",
                                   ngram_range=(1, 1),
                                   max_features = 5000)

train_tfid = tfidf_vectorizer.fit_transform(X_train)
train_tfid = np.array(train_tfid.toarray())

test_tfid = tfidf_vectorizer.fit_transform(X_test)
test_tfid = np.array(test_tfid.toarray())
```

9. Split the validation data

```
X_train_emb, X_valid_emb, y_train_emb, y_valid_emb = train_test_split(X_train_seq_trunc,
                                                                      y_train_oh,
                                                                      test_size=0.1,
                                                                      random_state=2003)

assert X_valid_emb.shape[0] == y_valid_emb.shape[0]
assert X_train_emb.shape[0] == y_train_emb.shape[0]

print('Shape of validation set:',X_valid_emb.shape)
```

10. Performance metrics

```
# Functions used to get the performnce of the model
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()))
```

11. Building the Model

```
from keras import models
from keras import layers
from keras import regularizers
##### Model 1 ######
Model = models.Sequential()
Model.add(layers.Embedding(NB_WORDS, 5, input_length=MAX_LEN))
Model.add(Conv1D(filters=100, kernel_size=2, activation= 'relu'))
Model.add(MaxPooling1D(pool_size=2))
Model.add(Conv1D(filters=100, kernel_size=2, activation= 'relu'))
Model.add(MaxPooling1D(pool_size=2))
Model.add(layers.Flatten())
Model.add(layers.Dense(5, activation='softmax'))
Model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_9 (Embedding)      (None, 51, 5)             25000
_________________________________________________________________
conv1d_17 (Conv1D)           (None, 50, 100)           1100
_________________________________________________________________
max_pooling1d_17 (MaxPooling (None, 25, 100)           0
_________________________________________________________________
conv1d_18 (Conv1D)           (None, 24, 100)           20100
_________________________________________________________________
max_pooling1d_18 (MaxPooling (None, 12, 100)           0
_________________________________________________________________
flatten_9 (Flatten)          (None, 1200)              0
_________________________________________________________________
dense_9 (Dense)              (None, 5)                 6005
=================================================================
Total params: 52,205
Trainable params: 52,205
Non-trainable params: 0
_________________________________________________________________
```

12. Initializing the hyper-parameters

```
lr=1e-4
decay=1e-4
num_epochs=100
adm = optimizers.Adam(lr=lr, decay=decay)
```

13. Compile and fit the model

The time() function is used to measure the time to run the program.

```
Model.compile(optimizer=adm,
              loss='categorical_crossentropy',
              metrics=['accuracy',f1_m,precision_m,recall_m])

t1=time.time()
history_train = Model.fit(X_train_emb,
                    y_train_emb,
                    epochs=num_epochs,
                    batch_size=128,
                    validation_data=(X_valid_emb,y_valid_emb))
t2=time.time()
t3=t2-t1
print('Training Time:',np.round(t3,2),'seconds')
```

14. Testing the model

```
history_test= Model.fit(X_train_seq_trunc,
              y_train_oh,
              epochs=num_epochs,
              batch_size=128,
              verbose=1)
loss, accuracy, f1_score, precision, recall = Model.evaluate(X_test_seq_trunc, y_test_oh)
# print('\nTest accuracy of word embeddings model: {0:.2f}%'.format(emb_results[1]*100))
# print(emb_results)
print('\nTest accuracy of word embeddings model: {0:.2f}%'.format(accuracy*100))
print('\nf1 of word embeddings model: {0:.2f}%'.format(f1_score*100))
print('\nprecision of word embeddings model: {0:.2f}%'.format(precision*100))
print('\nrecall of word embeddings model: {0:.2f}%'.format(recall*100))
```

15. Saving the trained model

```
#Save the trained model

!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

Model.save('1093483_SentimentAnalysis.h5')
Model_file = drive.CreateFile({'title' : '1093483_SentimentAnalysis.h5'})
Model_file.SetContentFile('1093483_SentimentAnalysis.h5')
Model_file.Upload()

# download to google drive
drive.CreateFile({'id': Model_file.get('id')})
```

5 RESULTS

1. Training time

```
Epoch 98/100
98317/98317 [==============================] -
Epoch 99/100
98317/98317 [==============================] -
Epoch 100/100
98317/98317 [==============================] -
Training Time: 359.94 seconds
```

```
Epoch 99/100
109242/109242 [==============================]
Epoch 100/100
109242/109242 [==============================]
46818/46818 [==============================] -

Test accuracy : 63.39%

f1 : 61.47%

precision : 66.58%

recall : 57.23%
```

2. Graph of the training accuracy

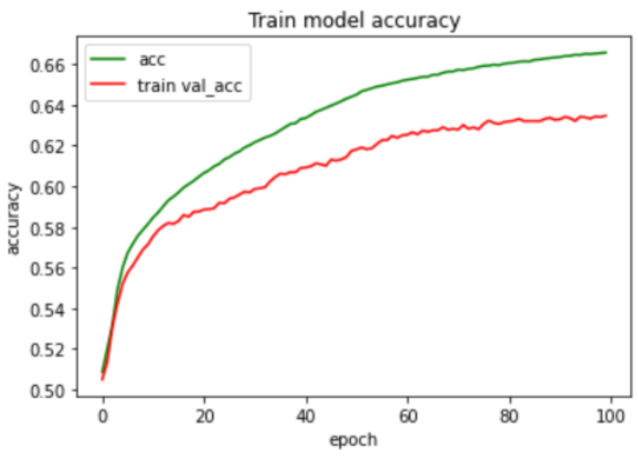

Figure 3 Model Accuracy

3. Loss

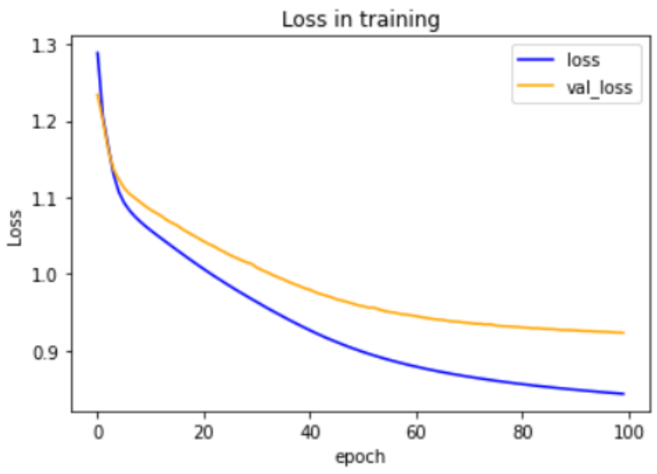

Figure 4 Model Loss

6 CONCLUSION

After performing the pre-processing on the data the accuracy obtained for the 100 epochs is 63.39%. The time required for training the model is around 359 seconds.

7 REFERENCES

[1] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
[2] https://github.com/cacoderquan/Sentiment-Analysis-on-the-Rotten-Tomatoes-movie-review-dataset
[3] https://monkeylearn.com/sentiment-analysis/
[4] https://expertsystem.com/natural-language-processing-sentiment-analysis/
[5] https://www.pythonforengineers.com/natural-language-processing-and-sentiment-analysis-with-python/