Multi-class Sentiment Analysis using Deep Learning

Parth Valani
Computer Science
Lakehead University 1116576
Thunder Bay, Canada
Valaniparth13@gmail.com

***Abstract*—This paper gives idea about how to use simple convolution neural network on the multi-class dataset and doing the multi-class sentiment classification get accuracy, recall, Precision and figure of merits using keras library.**

***Index Terms*—Convolution Neural Network, Overfitting, Kernal, pytorch, pooling, activation, epoch, batch size, learning rate.**

I. Introduction

In this report, I will going to discuss how we can use 1D convolution neural network to train the model with multi-class values. I use rotton tomatoes dataset [1] which has 5 classes like negetive,somewhat negetive,nuaral,somewhat positive,positive.

The goal of this report is to explore and apply the most recent technique's used in this task involving Natural language processing and to see and quantify the benefit of using those techniques to this kind of problem. [2]

II. Literature Review

With the rise of social media, Sentiment Analysis, which is one of the most well-known NLP tasks, gained a lot of importance over the years. The recent advances made in Machine Learning and Deep Learning made it an even more active task where a lot of work and research is still done.

CNN, which has been widely used on image datasets, extracts the significant features of the image, as the "convolutional" filter (i.e., kernel) moves through the image. If the input data are given as one-dimensional, the same function of CNN could be used in the text as well. In the text area, while the filter moves, local information of texts is stored, and important features are extracted. Therefore, using CNN for text classification is effective. [3]

There are many other model that have been used in the past like RNN, Embedding and LSTM and other model can also give far more accuracy than CNN network.

III. Proposed Model

Here is the step by step process:

- First, import all the necceesary libraries in the python. for example pandas for data visulization, numpy for array operartions, sklearn for machine learning operations.
- Now, load the rotton tomatos dataset from the given github link.
- Then, store it into a tabular format using pandas and get the full sentence from the tabular data.

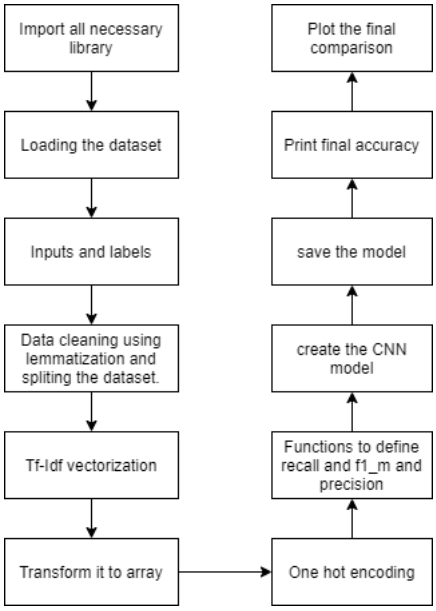

Fig. 1. Block diagram of process

- Divide the training data into inputs(phrase) and labels(sentiment) for further processing.
- remove stop words and use lemmatazation.
- Now, split the data in to training and testing data in the ratio of 70 to 30 percentage.
- In the next step, initiate the Tfidf vectorizer and fit the input phrase into into vectorizer.
- Converting train data into the 3 dimensional vector from 2 dimension. so that, we can feed it to the network.
- Then, convert the value to binary format using one hot encoding..
- Now, define all the necessary functions like recall, precision, figure-of metric.
- Now, define the convolution network of 4 layer and at the end connect fully connected network and compile the model.
- Finally, In the training step,feed the training data into 3-D shape and classify it.
- Now, save our model and get metrics including accuracy,recall,precision,f1_score.
- plot the training and validation accuracy vs epoch.

(A) **Import necessary libraries:** As per appendix-1, we

can call numpy for array operations, pandas for data visualization, sklearn for data analysis.

(B) **Load 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. I used pandas's 'read_csv' function file to read the data from the given link into tabular format which is separated by tab. [4]

(C) **Data-preprocessing:** As per appendix-2, get full sentence and convert it to the tokens using word_tokenize function.

(D) **Removing stop words and use lammetization:** As per appendix-3, I have removed all stop words and punctuation and use wordnet_lemmatizer and store it in the document list. and then take 'text' column of the all_data as an input and 'sentiment' column as an output.

(E) **Splitting dataset:** As per appendix-4, 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.

(F) **Vectorization:** As per appendix-5, I used TfIdf vectorizer which convert text into the vector form. It removes the stop words in English languages as well and make a list of words.

(G) **Convert train and test data into vectorize array:** As per appendix-6, I convert all the training and testing data into the array.

(H) **Convert 2-D to 3-D:** As per appendix-7, I convert inputs from two dimension to three dimensional space.

(I) **One-hot encoding** As per appendix-8, I convert all the labels into binary array using to_catagorical 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].

(J) **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.

(K) **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.

(L) **Compile the model:** As per appendix-11, we compile our cnn model using Nadam optimizer. we set the learning rate 0.001. number of classes are five.

(M) **Train model:** As per appendix-12, 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.

(N) **Save the model:** As per appendix-13, I saved the model as '1116576_1dconv_reg.h5' using model.save() function.

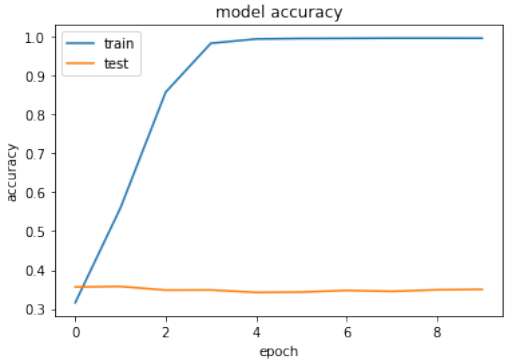

Fig. 2. epoch vs accuracy

(O) **Test model:** As per appendix-14, 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.

(P) **Plot the epoch vs accuracy graph:** As per appendix-15,define a Plothist() function which plot the graph for training and testing accuracy per epochs.

By that way, The model's accuracy is 35% , precision is 36% , recall is 34% and f1-score is also 35%. My training accuracy is quite high but testing accuracy is quite low because i used less number of data. The reason behind that is when I convert the data into corpus of words it will go out of memory utilization.

IV. Conclusion

To conclude, I have received 35% accuracy using just 3 convolution layer and 1 dense layer. I have also tried using embedding layer and I got little high accurracy.

From the above results,it has been clear that proper preprocessing of data based on natural language processing approaches as well as incorporating already existing models in the domain of sentiment analysis altogether with appropriate classification process can improve the performance of the model for multi-class classification.

V. Appendix

```
import pandas as pd
from sklearn.model_selection import train_test_split
import numpy as np
```

Listing 1. Import all necessary libraries

```
data = pd.read_csv('https://raw.githubusercontent.
    com/cacoderquan/Sentiment-Analysis-on-the-Rotten
    -Tomatoes-movie-review-dataset/master/train.tsv'
    , sep='\t')
data.head(20)
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
# put data into a df
fullSentDf = pd.DataFrame(fullSentences,
                                columns=['Phrase', '
    Sentiment'])
documents = []

# Use only complete sentences
for i in range(fullSentDf.shape[0]):
  tmpWords = word_tokenize(fullSentDf['Phrase'][i])
  documents.append((tmpWords, fullSentDf['Sentiment'
    ][i]))
```

Listing 2. Data-prepossessing

```
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])
all_data = pd.DataFrame(documents,
                                columns=['text', '
    sentiment'])
```

Listing 3. Removing stopwords and use lammetizer

```
# 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)
```

Listing 4. Spliting data into 70:30 ratio

```
from sklearn.feature_extraction.text import
    TfidfVectorizer
vectorizer = TfidfVectorizer(stop_words="english",
                             ngram_range=(1, 1))
x_train = vectorizer.fit_transform(x_train_raw)
y_train = y_train_raw
x_test = vectorizer.transform(x_test_raw)
y_test = y_test_raw
```

Listing 5. TfIDF vectorizer

```
# Converts the datasets to numpy arrays to work with
    our keras model
x_train_np = x_train.toarray()
y_train_np = np.array(y_train)

# Convert the testing data
x_test_np = x_test.toarray()
y_test_np = np.array(y_test)
```

Listing 6. Converts the datasets to numpy arrays

```
# Converting 2-D to 3-D
x_train_np = np.expand_dims(x_train_np, 2)
x_test_np = np.expand_dims(x_test_np, 2)
print(x_train_np.shape)
```

Listing 7. Convert 2-dimension data to 3-dimension for convolution

```
from keras.utils import to_categorical
y_train_np = to_categorical(y_train_np)
y_test_np = to_categorical(y_test_np)
print(y_train_np.shape)
```

Listing 8. One-hot encoding

```
# define the function for recall,precision,figure-of
    metric score
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 the neccesary libraries for the training
    the model
from keras.models import Sequential
from keras.layers import Conv1D, MaxPooling1D, Dense
    , Dropout, Flatten
from keras.layers import Activation, Conv1D,
    GlobalMaxPooling1D
from keras import optimizers
from keras import layers
# define convolution neural network
def cnn_model(data, num_class, opt):
  model= Sequential()
```

```
  model.add(Conv1D(filters=32, kernel_size=2,
    activation= 'relu', input_shape= (data.shape[1],
    data.shape[2])))
  model.add(MaxPooling1D(pool_size=2))
  model.add(Conv1D(filters=64, kernel_size=2,
    activation= 'relu'))
  model.add(MaxPooling1D(pool_size=2))
  model.add(Conv1D(filters=128, kernel_size=2,
    activation= 'relu'))
  model.add(MaxPooling1D(pool_size=2))
  model.add(Flatten())
  model.add(Activation('relu'))
  model.add(Dense(num_class))

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

  return model
```

Listing 10. 1D-CNN

```
# assign values to variables
batch_size= 64
num_epochs= 10
n_class = 5
Nadam = optimizers.Nadam(lr = 1e-3 , beta_1=0.9,
    beta_2=0.999, epsilon=1e-08)
# compile the model
model=cnn_model(x_train_np, n_class, Nadam)
```

Listing 11. Compile the model

```
# run the model
x = model.fit(x_train_np, y_train_np, batch_size=
    batch_size, epochs=num_epochs, verbose=1,
    validation_split=0.3)
```

Listing 12. Train the model

```
# save model with 50 epoch
model.save('1116576_1dconv_reg.h5')
```

Listing 13. Save the model

```
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(x_test_np, y_test_np, verbose=False)
metrics(accuracy,f1_score,precision,recall)
```

Listing 14. Get accuracy&f1-score&precision&recall and print the values

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

Listing 15. Plot the epoch vs accuracy

REFERENCES

[1] https://raw.githubusercontent.com/cacoderquan/Sentiment-Analysis-on-the-Rotten-Tomatoes-movie-review-dataset/master/train.tsv.

[2] https://www.kaggle.com/ynouri/rotten-tomatoes-sentiment-analysis.

[3] H. Kim and Y.-S. Jeong, "Sentiment classification using convolutional neural networks," *Applied Sciences*, vol. 9, no. 11, p. 2347, 2019.

[4] https://github.com/pratikmjoshi/sentiment-analysis-moviereviews/blob/master/SentimentAnalysisofMovieReviews.ipynb.