Department of Computer Science and Engineering,

Anna University Chennai - 600025

CS6301 – Machine Learning Laboratory

SARCASM DETECTION ON TWITTER DATA

TEAM NO. : 13

NAME	REG.NO	AADHAR NO	E-MAIL	MOBILE NO
N.A.Nisha	2018103047	508089229029	na.nisha2018@gmail.com	6379887669
Tanguturi Naga Lakshmi Sucharitha	2018103613	357571842068	sucharithatanguturi99@gmail.com	6300061998

TABLE OF CONTENTS

1. ABSTRACT

Sarcasm is the use of language where people used positive words in order to convey a negative message. Sarcasm is an example of what some researchers call uncertain speaking, where what is said by the people is actually differ from what is meant (especially so as to insult or wit somebody, to indicate irritation, or to be funny). Sarcasm detection in twitter is a very important task as it had helped in the analysis of tweets. With the help of sarcasm detection, companies could analyze the feelings of user about their products. This is helpful for companies, as the companies could improve their quality of product.

Sarcasm detection is a very narrow research field in NLP. Therefore the task of this field is to detect if a given text is sarcastic or not. For the classification we have used models of deep learning like CNN (Convolutional Neural Networks) and RNN(Recurrent Neural Networks). Further, On e-commerce websites such as amazon, many times customers make the use of sarcasm in their reviews in an attempt to criticize the product. With the help of sarcasm detection, products can be classified into the relevant categories with more accuracy.

2. INTRODUCTION

Sarcasm is the use of positive words to annoy someone, or for humorous purposes. For instance, When you get an “Fail” on an exam and your friend says, “Nice job, Einstein.” Here, actually the friend annoyed Einstein for getting low grade but with the positive word like “Nice”.

Users of Twitter may write a negative tweet if they are not feeling good with some news when they came to know it through social media. The tweet can be regular one if it is written in the straightforward way. The user can also make use of sarcasm in the post to mock or criticize any announcement/news. Detection of sarcasm in language in the form of text has been a challenging problem. This is because a sarcastic text might appear as a regular text. Using traditional opinion mining techniques can lead to incorrect classification .

A sarcastic sentence written with the view of expressing contempt might contain abundant positive words. Humans can figure out sarcasm easily when they know the tone of the sentence and the context in which it is used. We can thus improve the performance of a sarcasm detection system if we use contextual features along with traditional features to train our model . And we are going to recommend best classifying algorithm or model based on the model performance measures.

The overview of our project is structured as follows. Chapter 1 describes abstract of our project. Chapter 2 focuses on introduction. Chapter 3 focuses on the work related from the relevant literatures in the field carried out by various researchers with limitations and advantages. It also describes our improvement compared to other work. Chapter 4 describes problem statement . Chapter 5 gives a summary of the proposed model and description of the dataset. Chapter 6 gives the novelty of the project . In chapter 7 system architecture diagram and explanation of the architecture diagram is provided. Chapter 8 gives the description of modules defined in our project. Chapter 9 describes implementation of our work. In Chapter 10 results are discussed and Chapter 11 gives the conclusion and future enhancement. Chapter 12 contains references of our work.

3. LITERATURE SURVEY

1) Patrick Adolf Telnoni, Reza Budiawan, Mutia Qana'a." Comparison of Machine Learning Classification Method on Text-based Case in Twitter". In 978-1-7281-4880-9/19/$31.00 2019 IEEE.

   METHODOLOGY :
   In this paper they have done comparison of Machine Learning Method on Text-based Case in Twitter they have done classification using classic ml algorithms.

   ADVANTAGES :

   This paper will discuss the performance of classification methods for text-based data. It gives the best choices of classification method in terms of accuracy and training time. So that will help ML project that does not require high computational cost. It also gives recommendation to practitioner and academic about which classifier best for text classification.

   LIMITATION :

   In this paper they have compared the performance of classic machine learning algorithms which has lower accuracy range.

2) Neha Pawar and Sukhada Bhingarkar." Machine Learning based Sarcasm Detection on Twitter Data".In the Fifth International Conference on Communication and Electronics Systems (ICCES 2020) IEEE Conference Record # 48766; IEEE Xplore ISBN: 978-1-7281-5371-1.

   METHODOLOGY :
   In this paper they had done Machine Learning based sarcasm Detection on Twitter Data they have done classification using classic ml algorithms.

ADVANTAGES :

In this paper they have done mood analysis and opinion mining rely on emotional words to detect their polarity in a text(that is whether it relates to "positivity" or " negativity" in its thread).They have taken four sets of features that include a lot of specific sarcasm , is proposed and classify tweets as sarcastic and non sarcastic.

LIMITATION :

All patterns for sarcastic detection are not covered in the extracted patterns here. classic ml algorithm has lesser accuracy. The accuracy cannot be at the desired level.

3) Azilawati Azizan,Nurul Najwa SK Abdul Jamal,Mohammad Nasir Abdullah,Masurah Mohamad,Nurkhairizan Khairudin."Lexicon-Based Sentiment Analysis for Movie Review Tweets".In 978-1-7281-3041-5/19/$31.00 2019 IEEE

METHODOLOGY :
In this paper they had done lexicon -based sentiment analysis for movie review tweets and they have also mentioned that lexicon-based model is able to classify sentiment with very minimal computational cost compared to machine learning which is very data-dependent.

ADVANTAGES :

In this paper they have used sentiment analysis to identify and classify subjective information such as positive, negative and neutral from the source material .They have used R as the language and development framework which is a lexicon based method. This result is comparable with other existing experiments, Additionally the result of this project also proves that lexicon based model is able to classify sentiment with very minimal computational cost compared to machine learning which is very data-dependent.

LIMITATION :

In comparison either other research work on lexicon based approach, it seems that many other researchers for higher accuracy value because they incorporate some additional features to the basic lexicon. The accuracy score for this project is not impressive may be due to its straight forward and plain lexicon used in the algorithm. Besides that, movie review data is also said to be difficult to analyse compared to other domain review.

4) Soujanya Poria ,Erik Cambria ,Devamanyu Hazarika, Prateek Vij" A Deeper Look into Sarcastic Tweets Using Deep Convolutional Neural Networks" Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers, pages 1601–1612, Osaka, Japan, December 11-17 2016.

METHODOLOGY :

In this paper they exploited sentiment and emotion features for sarcasm detection. As user profiling is also an important factor for detecting sarcastic content, moreover, they used personality based features for the first time in the literature.

ADVANTAGES :

Pre trained models are commonly used in computer vision. In the context of natural language processing(NLP),however they are barely used. Hence the use of pre - trained models for feature extraction is also a major contribution of this work.

LIMITATIONS :

word2vec models does not give much performance than GLOVE.

OUR IMPROVEMENT

We detect the sarcasm by deep neural networks(solving the cons of paper [2] using deep neuro network as explained in paper[4].We used the supervised learning approach explained in paper[2] but we have done it as transfer learning approach. To do sarcasm detection ,we should do some pre-processing to the dataset that are explained in paper[3].we also using the tweets taken in general(solving the cons of paper[3]) and we have used GLOVE model (solving cons of paper[4])for better performance.

All the papers we took have only discussed about identifying sarcasm using classic machine learning classifications algorithms . And they didn't use neural networks which has higher performance. Also that they have analysed through Movie Review Tweets which is not very efficient. These are the issues we have identified.

4. PROBLEM STATEMENT

Customers of e-commerce website like amazon may write a negative review of a product if they are not happy with the product's price, quality or service of the dealer. The review can be regular one if it is written in the straightforward way. The customer can also make use of sarcasm in the review to criticize the product. Suppose a customer buys a very costly headphone and it turns out the quality of sound is not very good. He then writes a review as "the speaker is so good that I can hear sounds from other galaxies! Totally worth the money" The user is obviously not satisfied with the product and is mocking the product's quality and price using sarcasm. So, in these kind of situations it seems very difficult to detect the negative comment with the given positive words. So, the user personalization of the website becomes in affective. Also, finding the classifier model with better accuracy matters a lot. Since it signifies the efficiency of the system.

5. PROBLEM SOLUTION

5.1. DATA SET

We have collected tweets, using Twitter’s streaming API. To collect sarcastic tweets, we queried the API for tweets containing the hashtag “#sarcasm”. Although this hashtag is not the best way to collect sarcastic tweets, the fact that this hashtag can be used for this purpose. However, the hashtag cannot be reliable and is used mainly for 3 purposes:

- to serve as a search anchor,
- to serve as a sarcasm marker in case of a very subtle sarcasm where it is very hard to get the sarcasm without an explicit marker, as in “Today was fun. The first time since weeks! #Sarcasm”,
- to clarify the presence of sarcasm in a previous tweet, as in “I forgot to add #sarcasm so people like you get it!”.

In total, we have collected 1500 tweets with the hashtag “#sarcasm”, which we clean up by removing the noisy and irrelevant ones, as well as ones where the use of the hashtag does fall into one of the two first uses of the three described above. As for non-sarcastic tweets, we are going to collect tweets dealing with different topics and made sure they have some emotional content.

5.2. APPROACH

We collect tweets using twitters streaming API , which we clean up by removing the noisy and irrelevant ones, then we break each sentence into words. If a tweet contains unwanted information then that tweet is removed from the dataset. If a tweet contains any stop words or punctuation they too removed using nltk toolkit. We split our data in to test and train.

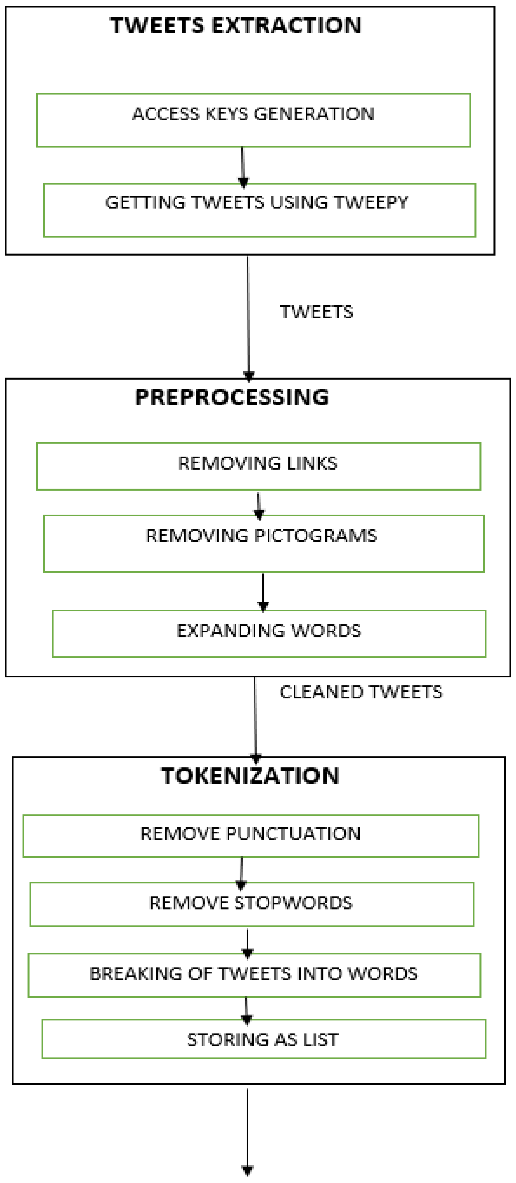
Pretrained glove model is loaded in to our project. By using glove model we create embedding layer .We use sequential model to build neural networks and we will pass embedding layer as first layer to the neural networks. We select best neural network among RNN and CNN based on performance metrics such as accuracy, precision, recall and f1-score.

6. NOVELTY

By analysing all the papers we found that all these papers have done model training using normal classifiers. They have not done using neural networks which has higher performance than normal classifiers. we are going to do sarcasm detection using neural networks like RNN and CNN. We are going to use glove model for doing this. And we are going to recommend model which has high performance. In module 5 we have built our Innovative CNN and RNN models using keras().

7. ARCHITECTURE

7.1. DETAILED ARCHITECTURE DIAGRAM



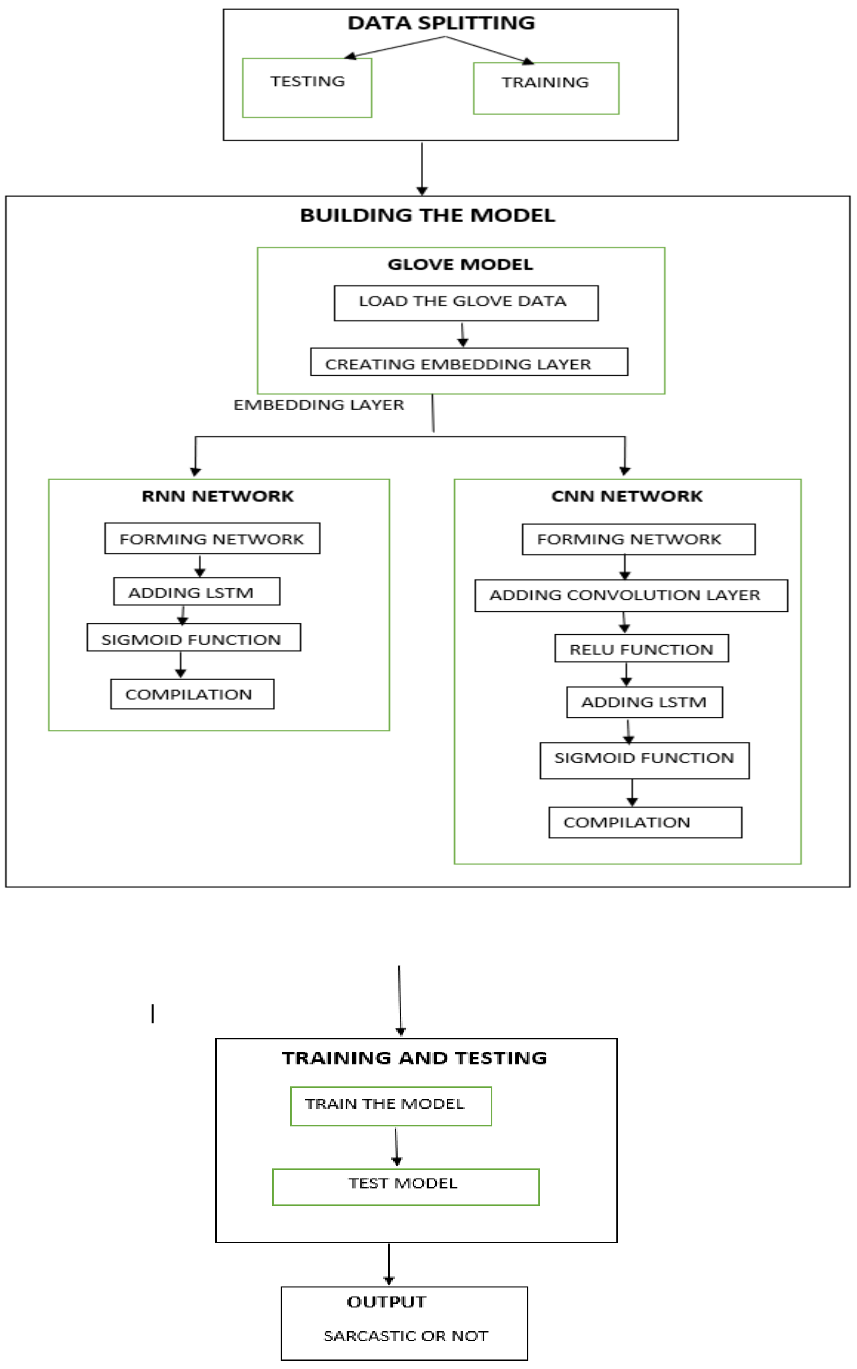


FIGURE-7.1.1 : System Architecture diagram

7.2. EXPLANATION OF DETAILED ARCHITECTURE DIAGRAM

Given with twitter dataset, The first thing to do is get the consumer key, consumer secret, access key and access secret from twitter developer available easily for each user. These keys will help the API for authentication. Collected tweets, using Twitter's streaming API. To collect sarcastic tweets, queried the API for tweets containing the hashtag "#sarcasm". After that all the available tweets are first cleaned by removing links, pictograms, emojis if any, expanding shortened form to full form. Converts the tweets to its lowercase form, removes float/numeric values if it contains any in between the text and removes special characters like [,.\"\'!@#$%^&*(){}?/;`~:<>+=-].

Then we break each sentence into words in tokenization. If a tweet contains unwanted information then that tweet is removed from the dataset. If a tweet contains any stop words or punctuation they too removed using nltk toolkit. For tokenization , we had make use of nltk toolkit and it is the Natural Language Toolkit (NLTK) is a platform used for building Python programs that work with human language data for applying in statistical natural language processing (NLP). In tokenization , it read each tweets line and separate the words in the each tweet and store it in a list.

Split the data into test and train data. After data split , visualize a word cloud by taking the words from train data and depicted a picture which contains words of different frequencies. Pre-trained Glove model developed as an open-source project at Stanford loaded into our project. It makes a co-relation matrix among words.

First create a embedding index .After that we will create a embedding layer from the glove model which is to be passed as first layer to form a recurrent neural network.. Pass embedding layer as the first layer to Neural Network. Then we have added the LSTM layer(Long Short Term Memory). Since ,it's a binary classification we have used activation function as SIGMOID ,and the last layer would contain only one node.

Add a convolutional layer (which are used mainly for image processing, classification, segmentation and also for other auto correlated data. A convolution is essentially sliding a filter over the input)and then an LSTM layer on top of CNN layer. Further we have called ReLu activation function for CNN and sigmoid function for LSTM layer. We have trained the models separately by providing train data, validation data. For detecting Sarcasm, the sentence given by the user undergo necessary pre-processing, tokenization etc., and finally it gets predicted by the trained RNN and CNN models. Then the models provide a probabilistic value for that sentence based on the prediction. By that value we come to an conclusion that the given input sentence is sarcastic or not .

8. DETAILED MODULE DESIGN

8.1. EXTRACTION OF TWEETS USING TWEEPY

Twitter is a popular social network where users share messages called tweets. Twitter allows us to mine the data of any user using Twitter API or Tweepy. The data will be tweets extracted from the user. The first thing to do is get the consumer key, consumer secret, access key and access secret from twitter developer available easily for each user. These keys will help the API for authentication.

Types of tweets to be extracted

We have collected tweets, using Twitter's streaming API. To collect sarcastic tweets, we queried the API for tweets containing the hashtag "#sarcasm". Although this hashtag is not the best way to collect sarcastic tweets, the fact that this hashtag can be used for this purpose. However, the hashtag cannot be reliable and is used mainly for 3 purposes:

- to serve as a search anchor,
- to serve as a sarcasm marker in case of a very subtle sarcasm where it is very hard to get the sarcasm without an explicit marker, as in "Today was fun. The first time since weeks! #Sarcasm",
- to clarify the presence of sarcasm in a previous tweet, as in "I forgot to add #sarcasm so people like you get it!".

In total, we have collected 1500 tweets with the hashtag "#sarcasm", which we clean up by removing the noisy and irrelevant ones, as well as ones where the use of the hashtag does fall into one of the two first uses of the three described above. As for non-sarcastic tweets, we are going to collect tweets dealing with different topics and made sure they have some emotional content.

INPUT : Access keys with csv file to store tweets.

OUTPUT : Regular and Sarcastic tweets

PSEUDO CODE:

```
Module TweetsExtraction():
    import python libraries
    declare access keys
    set the authorization variable
    call Tweepy.API()
    Begin loop
    for tweets in tweepy.Cursor
    do
    Extract Sarcastic tweets
    write it into a csv file
    repeat
    Endloop
    repeat for regular tweets
EndModule
```

8.2. PREPROCESSING

We retrieve tweets with Twitter API by using the manually produced query. Final queries for the retrieval are a combination of sarcastic and non-sarcastic tweets. However, there is a problem with the final queries. Tweets often contain noisy/unwanted data. To remove such noisy data, we perform morphological analysis.

Here, we have defined a function called CleanText() in which tweet from tweets column of the dataframe undergo several refinery process which includes converting the tweets to its lowercase form, removing float/numeric values if it contains any in between the text, removing special characters like [,.\"\'!@#$%^&*(){}?/;`~:<>+=-] and by using regular expressions we have expanded the shortened form of words to its full form, removed pictograms, emojis if any. So, from this function we can get a cleaned form of tweets which can be further tokenized by the next module.

INPUT : Raw tweets from twitter API

OUTPUT : Pre-processed tweets

PSEUDOCODE :

```
Module DataPreprocessing()
    Load data files
    combine them into a single file
    Begin loop
    for tweet in tweetscolumn
    do
    CleanText(tweet):
    tweet.toLowerCase
    remove emoticons,links,symbols & pictographs,transport & map symbols,
    flags (iOS),special characters[,.\"\'!@#$%^&*(){}?/;`~:<>+=-]
    perform expansion of shortened words[I'm - I am]
    repeat
    Endloop
    return cleanedtweet
EndModule
```

8.3. TOKENIZATION

We break each sentence into words in tokenization. If a tweet contains unwanted information then that tweet is removed from the dataset. If a tweet contains any stop words or punctuation they too removed using nltk toolkit. For tokenization , we had make use of nltk toolkit and it is the Natural Language Toolkit (NLTK) is a platform used for building Python programs that work with human language data for applying in statistical natural language processing (NLP).

It contains text processing libraries for tokenization, parsing, classification, stemming, tagging and semantic reasoning. In tokenization , it will read each tweets line and separate the words in the each tweet and store it in a list.

INPUT : Cleaned tweets

OUTPUT : Tokenized tweets

PSEUDOCODE:

```
Module Tokenization()
        import word_tokenize,stopwords from nltk
        call CleanTokenize(df):
        CleanTokenize(df):
                Separating the tweets column from file
                convert it into list
Begin loop
        for tweet in tweetslist
          do
            CleanText(tweet)
        tokenize the text
            remove puntuations
            remove non alphabetic characters
            remove stop words
          repeat
        Endloop
        return Tokenizedlist
EndModule
```

8.4. DATA SPLIT

In this module, firstly we have converted all the words into numbers most particularly vectors. Then for data split we have used softmaxclass. This module just splits the tweets into test and train data set in the ratio 1:9 which means splits the 100% dataset into 10%test data and 90% train data.Then for accuracy,we have set max_length parameter as 25 which tells that the maximum number of words to be taken from a tweet line is 25.After that, shuffling happens so that no partiality occurs. After data split , we have visualized a word cloud by taking the words from train data and depicted a picture which contains words of different frequencies.

INPUT : Tokenized tweets

OUTPUT : Splitting data set as test and train .

PSEUDOCODE :

```
Module DataSplit()
    Declaring validation_split and max_length
    Creating an object of class Tokenizer()
    fitting our data by fit_on_textsconverting into sequences using text_to_sequences()
    finding unique tokens and vocabulary size
    call pad_sequences
    declaring sentiment
    call np.random.shuffle()
    print dimensions of train,test data.
    Visualize wordcloud
EndModule
```

8.5. BUILDING THE MODEL

GLOVE MODEL

Pre-trained Glove model developed as an open-source project at Stanford will be loaded into our project. It makes a co-relation matrix among words. In that, first we create a embedding_index dictionary where we have word as key and it's index as values. Here , the index simply refers to co-efficients of vectors . After that we will create a embedding layer from the glove model which is to be passed as first layer to form a recurrent neural network.

RNN (Recurrent Neural Network)

We have used Sequential module from keras to build a neural network. After that, the passed embedding layer and added as the first layer to our Neural Network.Then we have added the LSTM layer(Long Short Term Memory) with 54 neurons and RNN. RNNs are designed to recognize a data's sequential characteristics and use patterns to predict the next likely scenario. Since ,it’s a binary classification we have used activation function as SIGMOID ,and the last layer would contain only one node.

PSEUDOCODE :

```
Module Build_RNN()
          call model.sequential()
          add embedding layer as first layer
          add LSTM ,RNN
          call model.compile()
          print(Model Summary)

          EndModule
```

CNN (Convolution Neural Network)

We have used Sequential module from keras to build a neural network. After that, the passed embedding layer from the previous module is added as the first layer to our Neural Network.We have added a convolutional layer (which are used mainly for image processing, classification, segmentation and also for other auto correlated data. A convolution is essentially sliding a filter over the input)and then an LSTM layer on top of CNN layer. Further we have called ReLu activation function for CNN and sigmoid function for LSTM layer.

PSEUDOCODE :

```
Module Build_CNN()
    call model.sequential()
     add embedding layer as first layer
     add CNN
     add LSTM
     call model.compile()
     print(Model Summary)
EndModule
```

INPUT : Embedding layer .

OUTPUT : Summary of the models.

8.6. TRAINING AND TESTING

TRAINING AND VISUALIZATION

We have trained the models separately by providing train data, validation data. Also specified the running parameters such as epochs, batch size then we have fitted the model and had run it. As a result of this we can view the incremental accuracy values and decremental loss values which are changing from epochs to epochs.

We have calculated the precision,F1score,Recall,Confusion matrix for each of the training models and on training we will get performance measures like

- Training accuracy
- Validation accuracy
- Training loss
- Validation loss

by comparing these values in this performance model we can find out which model serves best for sarcasm detection with higher performance.

INPUT : Train and validation data.

OUTPUT : Accuracy and loss values.

PSEUDOCODE :

```
Module Training_and_Visualization()
        Declaring epochs,batch_size,validation_data
        call model.fit()
        declare label,color
        call plt.legend()
        call plt.figure()
        call plt.show()
EndModule
```

TESTING

For detecting Sarcasm, we have a testing function predict_sarcasm() that accepts user input sentence from an external file .The sentence then undergo necessary pre-processing, tokenization etc., and finally it gets predicted by the trained RNN and CNN models. Then the models provide a probabilistic value for that sentence based on the prediction. By that value we come to an conclusion that the given input sentence is sarcastic or not .

INPUT : User sentence

OUTPUT : Sarcastic or not

PSEUDOCODE :

```
Module Testing()
        Declaring function predict_sarcasm(s)
        getting the user sentence
        call CleanTokenize(s)
        call text_to_sequence()
        call pad_sequence()
        call model.predict()
if (pred[0][0] > 50)
        return "It's a sarcasm"
        else "It's not a sarcasm"
        # to test
        call predict_sarcasm(user Input Sentence)
EndModule
```

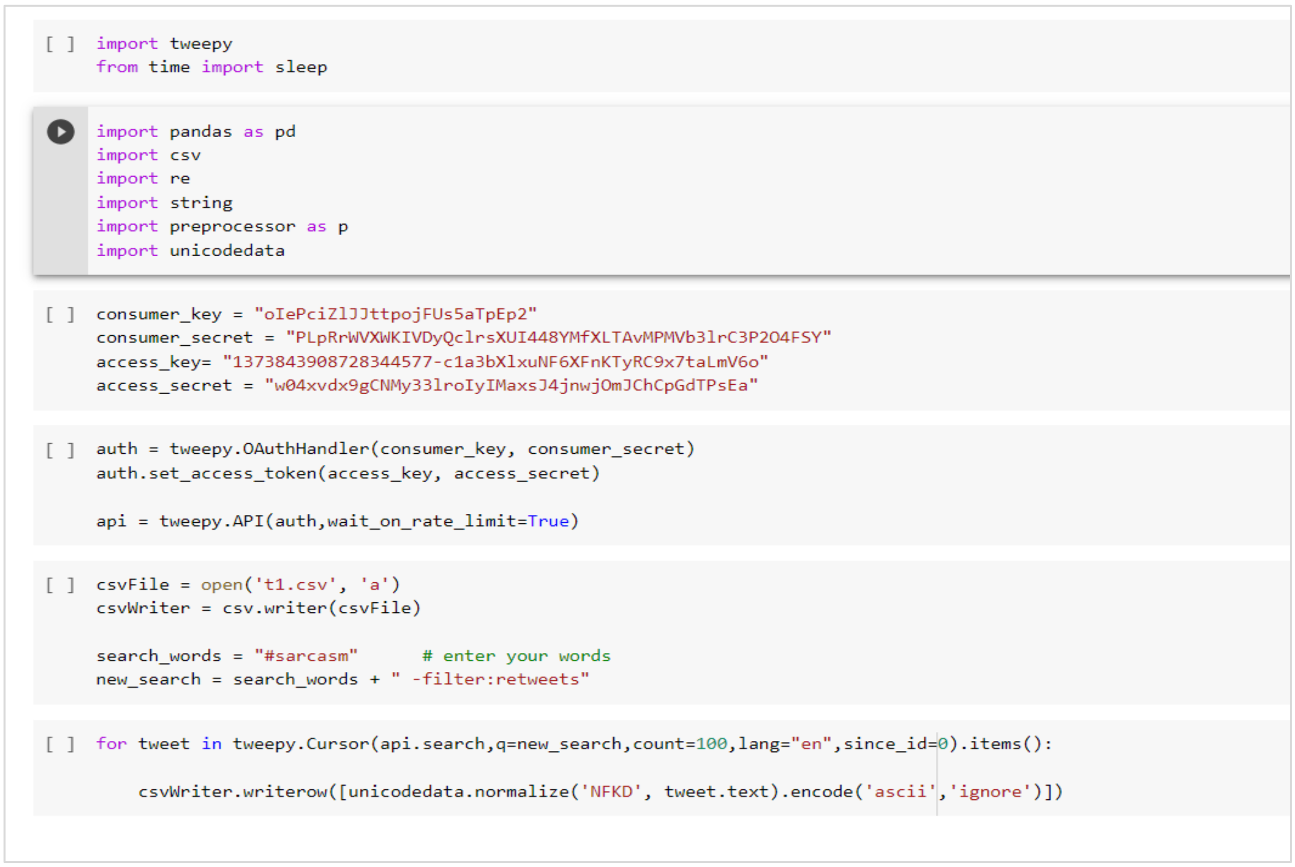
9. IMPLEMENTATION

9.1. INITIAL SET UP

- Download tweepy,preprocesssor in your system.
- The first thing to do is get the consumer key, consumer secret, access key and access secret from twitter developer available easily for each user.
- These keys will help the API for authentication.
- Do some preprocessing for the dataset.
- Download nltk tool kit to perform tokenization and to remove stop words.
- Pretrained glove model should be loaded for co-occurrence matrix.

9.2. CODE SNIPPET

9.2.1. EXTRACTION OF TWEETS



```
[ ]  import tweepy
     from time import sleep
```

```
     import pandas as pd
     import csv
     import re
     import string
     import preprocessor as p
     import unicodedata
```

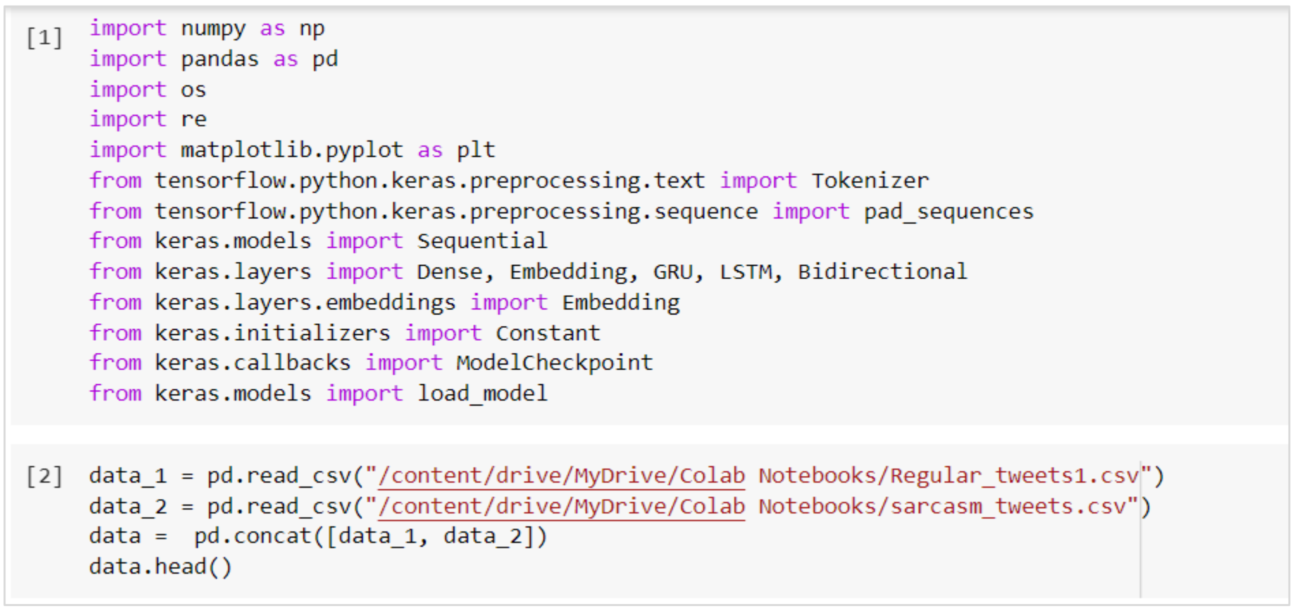
```
[ ]  consumer_key = "oIePciZlJJttpojFUs5aTpEp2"
     consumer_secret = "PLpRrWVXWKIVDyQclrsXUI448YMfXLTAvMPMVb3lrC3P2O4FSY"
     access_key= "1373843908728344577-c1a3bXlxuNF6XFnKTyRC9x7taLmV6o"
     access_secret = "w04xvdx9gCNMy33lroIyIMaxsJ4jnwjOmJChCpGdTPsEa"
```

```
[ ]  auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
     auth.set_access_token(access_key, access_secret)

     api = tweepy.API(auth,wait_on_rate_limit=True)
```

```
[ ]  csvFile = open('t1.csv', 'a')
     csvWriter = csv.writer(csvFile)

     search_words = "#sarcasm"      # enter your words
     new_search = search_words + " -filter:retweets"
```

```
[ ]  for tweet in tweepy.Cursor(api.search,q=new_search,count=100,lang="en",since_id=0).items():

         csvWriter.writerow([unicodedata.normalize('NFKD', tweet.text).encode('ascii','ignore')])
```

Concatenating regular and sarcasm files

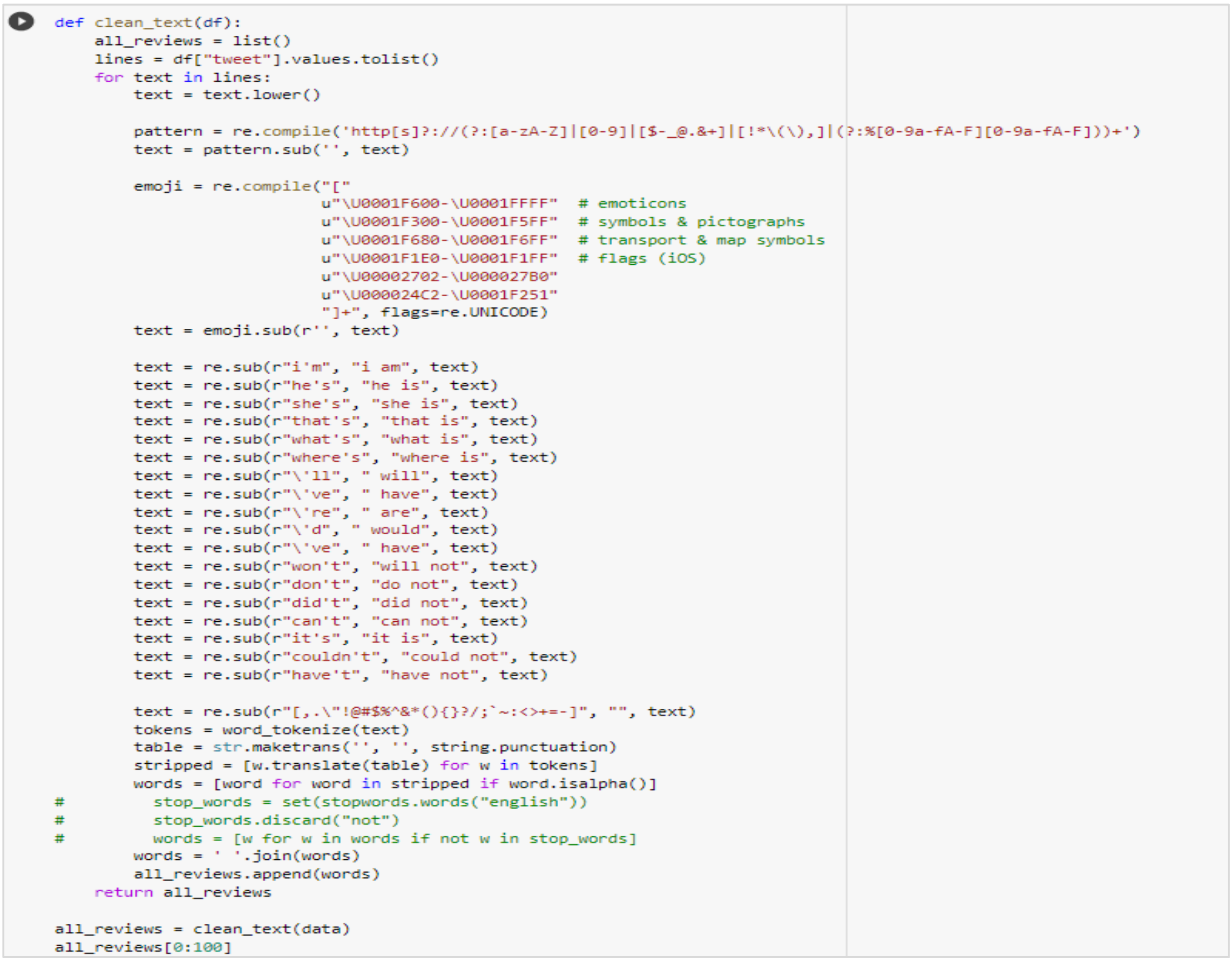


```
[1]  import numpy as np
     import pandas as pd
     import os
     import re
     import matplotlib.pyplot as plt
     from tensorflow.python.keras.preprocessing.text import Tokenizer
     from tensorflow.python.keras.preprocessing.sequence import pad_sequences
     from keras.models import Sequential
     from keras.layers import Dense, Embedding, GRU, LSTM, Bidirectional
     from keras.layers.embeddings import Embedding
     from keras.initializers import Constant
     from keras.callbacks import ModelCheckpoint
     from keras.models import load_model
```

```
[2]  data_1 = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Regular_tweets1.csv")
     data_2 = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/sarcasm_tweets.csv")
     data =  pd.concat([data_1, data_2])
     data.head()
```

9.2.2. PREPROCESSING

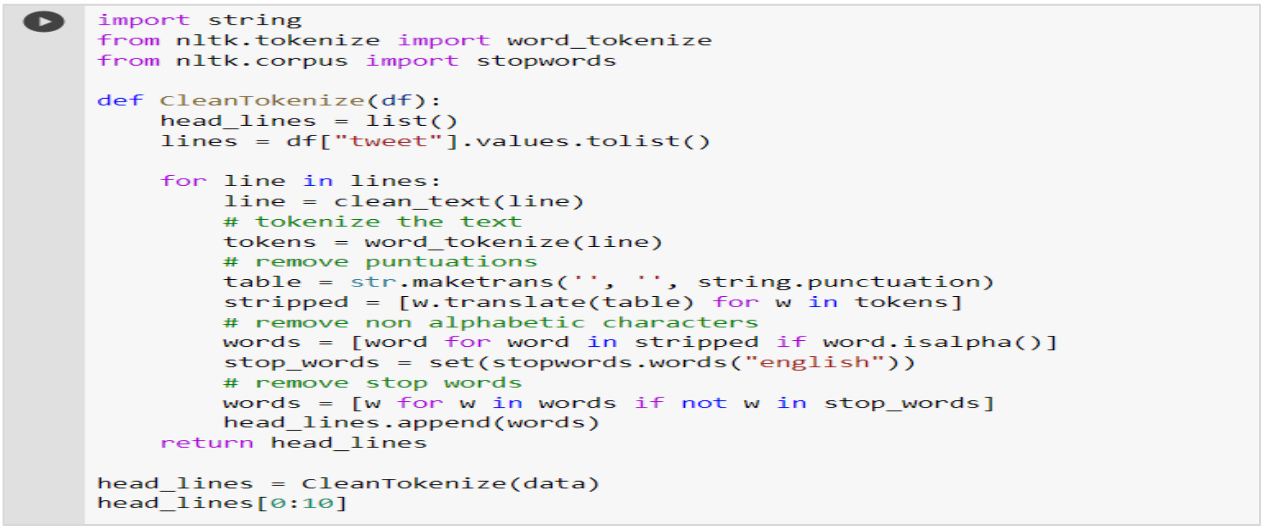
Cleaning tweets



```
def clean_text(df):
    all_reviews = list()
    lines = df["tweet"].values.tolist()
    for text in lines:
        text = text.lower()

        pattern = re.compile('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+')
        text = pattern.sub('', text)

        emoji = re.compile("["
                           u"\U0001F600-\U0001FFFF"  # emoticons
                           u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                           u"\U0001F680-\U0001F6FF"  # transport & map symbols
                           u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           u"\U00002702-\U000027B0"
                           u"\U000024C2-\U0001F251"
                           "]+", flags=re.UNICODE)
        text = emoji.sub(r'', text)

        text = re.sub(r"i'm", "i am", text)
        text = re.sub(r"he's", "he is", text)
        text = re.sub(r"she's", "she is", text)
        text = re.sub(r"that's", "that is", text)
        text = re.sub(r"what's", "what is", text)
        text = re.sub(r"where's", "where is", text)
        text = re.sub(r"\'ll", " will", text)
        text = re.sub(r"\'ve", " have", text)
        text = re.sub(r"\'re", " are", text)
        text = re.sub(r"\'d", " would", text)
        text = re.sub(r"\'ve", " have", text)
        text = re.sub(r"won't", "will not", text)
        text = re.sub(r"don't", "do not", text)
        text = re.sub(r"did't", "did not", text)
        text = re.sub(r"can't", "can not", text)
        text = re.sub(r"it's", "it is", text)
        text = re.sub(r"couldn't", "could not", text)
        text = re.sub(r"have't", "have not", text)

        text = re.sub(r"[,.\"!@#$%^&*(){}?/;`~:<>+=-]", "", text)
        tokens = word_tokenize(text)
        table = str.maketrans('', '', string.punctuation)
        stripped = [w.translate(table) for w in tokens]
        words = [word for word in stripped if word.isalpha()]
#         stop_words = set(stopwords.words("english"))
#         stop_words.discard("not")
#         words = [w for w in words if not w in stop_words]
        words = ' '.join(words)
        all_reviews.append(words)
    return all_reviews

all_reviews = clean_text(data)
all_reviews[0:100]
```

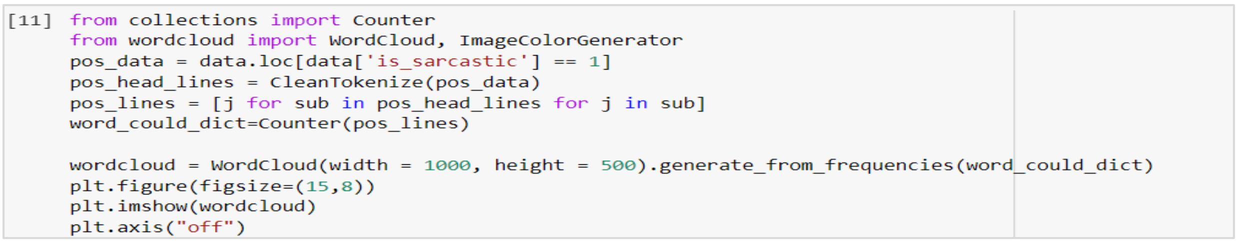
9.2.3. TOKENIZATION

Tokenization of tweets



```
import string
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

def CleanTokenize(df):
    head_lines = list()
    lines = df["tweet"].values.tolist()

    for line in lines:
        line = clean_text(line)
        # tokenize the text
        tokens = word_tokenize(line)
        # remove puntuations
        table = str.maketrans('', '', string.punctuation)
        stripped = [w.translate(table) for w in tokens]
        # remove non alphabetic characters
        words = [word for word in stripped if word.isalpha()]
        stop_words = set(stopwords.words("english"))
        # remove stop words
        words = [w for w in words if not w in stop_words]
        head_lines.append(words)
    return head_lines

head_lines = CleanTokenize(data)
head_lines[0:10]
```

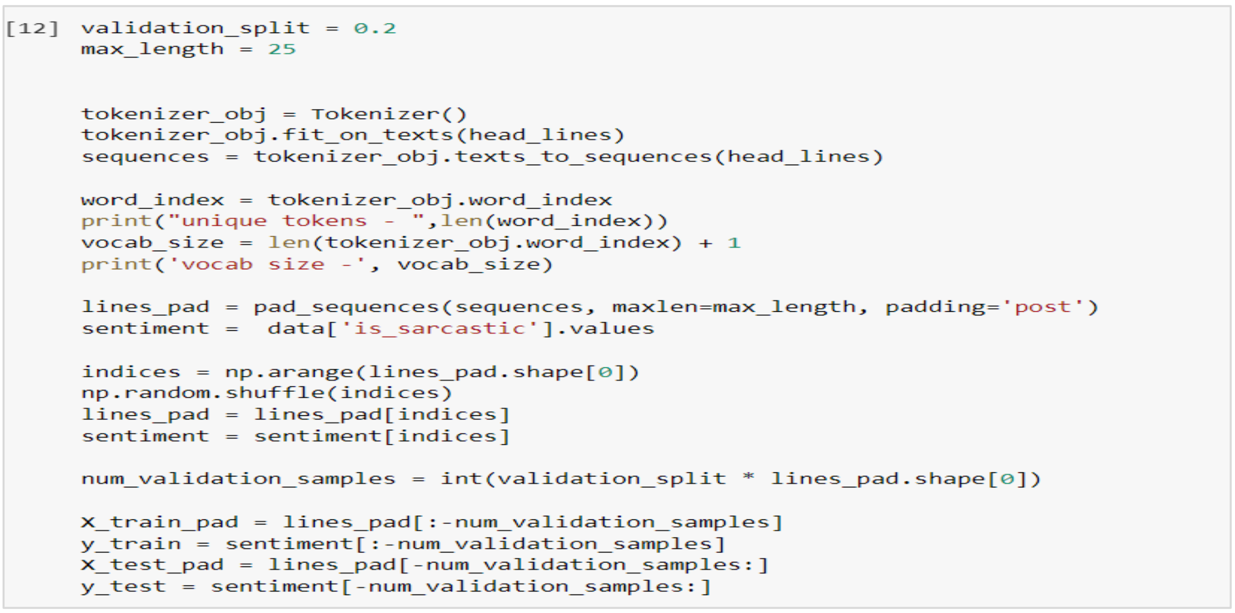
Word cloud for sarcastic tweets



```
[11] from collections import Counter
     from wordcloud import WordCloud, ImageColorGenerator
     pos_data = data.loc[data['is_sarcastic'] == 1]
     pos_head_lines = CleanTokenize(pos_data)
     pos_lines = [j for sub in pos_head_lines for j in sub]
     word_could_dict=Counter(pos_lines)

     wordcloud = WordCloud(width = 1000, height = 500).generate_from_frequencies(word_could_dict)
     plt.figure(figsize=(15,8))
     plt.imshow(wordcloud)
     plt.axis("off")
```

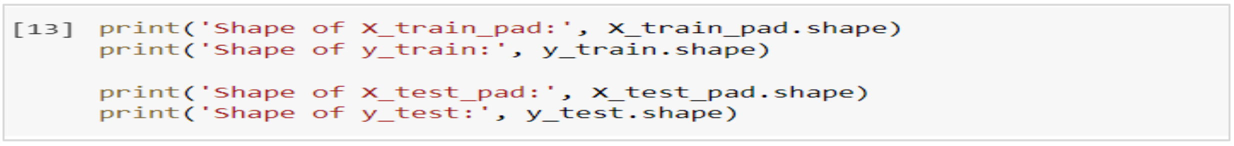
9.2.4. DATA SPLITTING

Vocabulary of word size



```
[12] validation_split = 0.2
     max_length = 25


     tokenizer_obj = Tokenizer()
     tokenizer_obj.fit_on_texts(head_lines)
     sequences = tokenizer_obj.texts_to_sequences(head_lines)

     word_index = tokenizer_obj.word_index
     print("unique tokens - ",len(word_index))
     vocab_size = len(tokenizer_obj.word_index) + 1
     print('vocab size -', vocab_size)

     lines_pad = pad_sequences(sequences, maxlen=max_length, padding='post')
     sentiment =  data['is_sarcastic'].values

     indices = np.arange(lines_pad.shape[0])
     np.random.shuffle(indices)
     lines_pad = lines_pad[indices]
     sentiment = sentiment[indices]

     num_validation_samples = int(validation_split * lines_pad.shape[0])

     X_train_pad = lines_pad[:-num_validation_samples]
     y_train = sentiment[:-num_validation_samples]
     X_test_pad = lines_pad[-num_validation_samples:]
     y_test = sentiment[-num_validation_samples:]
```

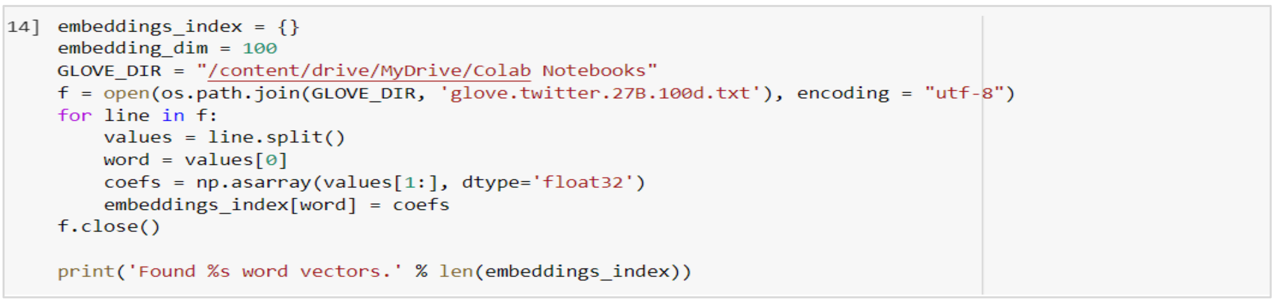
Shape of testing and training data



```
[13] print('Shape of X_train_pad:', X_train_pad.shape)
     print('Shape of y_train:', y_train.shape)

     print('Shape of X_test_pad:', X_test_pad.shape)
     print('Shape of y_test:', y_test.shape)
```

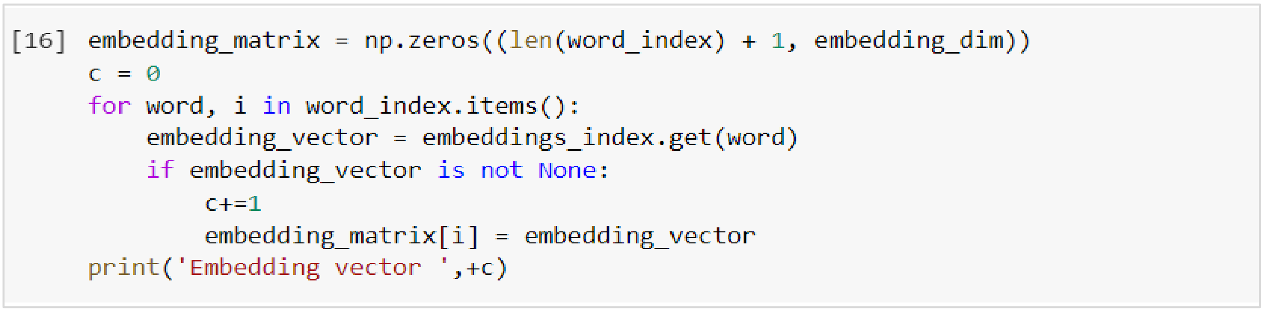
9.2.5. BUILDING THE MODELS

Loading glove model



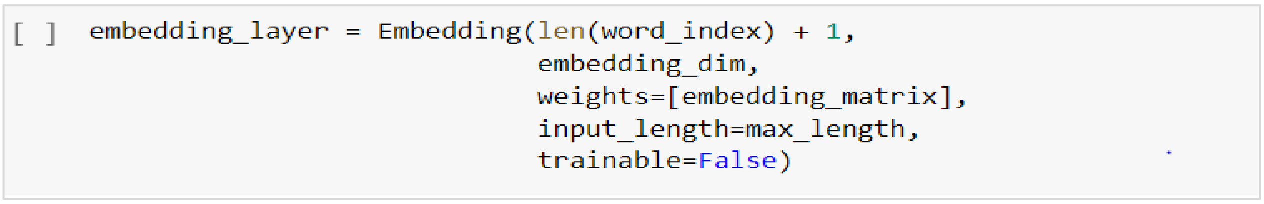
```
14] embeddings_index = {}
    embedding_dim = 100
    GLOVE_DIR = "/content/drive/MyDrive/Colab Notebooks"
    f = open(os.path.join(GLOVE_DIR, 'glove.twitter.27B.100d.txt'), encoding = "utf-8")
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs
    f.close()

    print('Found %s word vectors.' % len(embeddings_index))
```

Creating embedding matrix



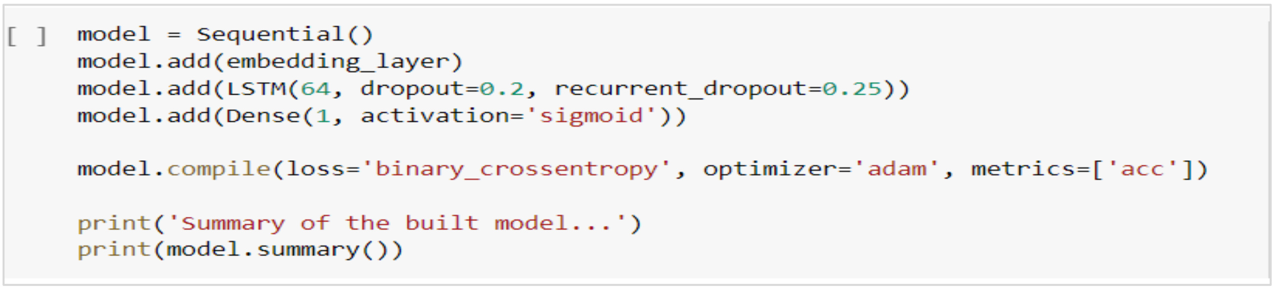
```
[16] embedding_matrix = np.zeros((len(word_index) + 1, embedding_dim))
     c = 0
     for word, i in word_index.items():
         embedding_vector = embeddings_index.get(word)
         if embedding_vector is not None:
             c+=1
             embedding_matrix[i] = embedding_vector
     print('Embedding vector ',+c)
```

Creating embedding layer



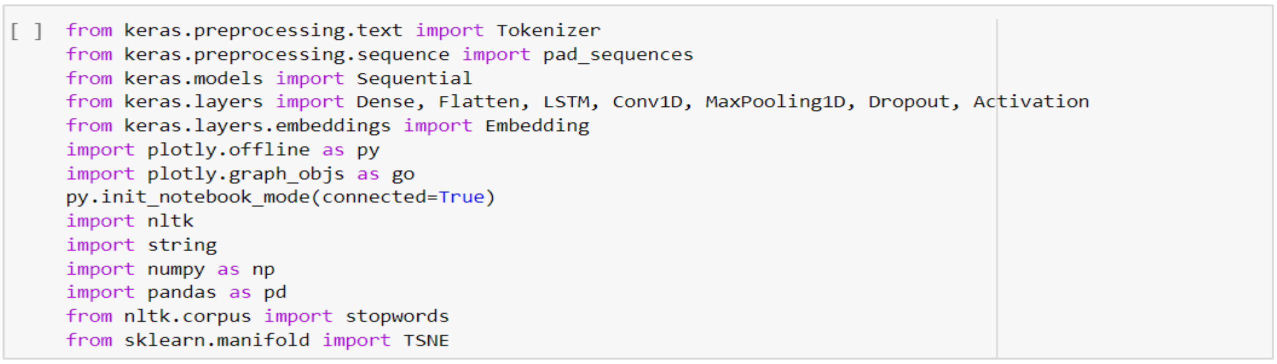
```
[ ]  embedding_layer = Embedding(len(word_index) + 1,
                                 embedding_dim,
                                 weights=[embedding_matrix],
                                 input_length=max_length,
                                 trainable=False)
```

Build the RNN model



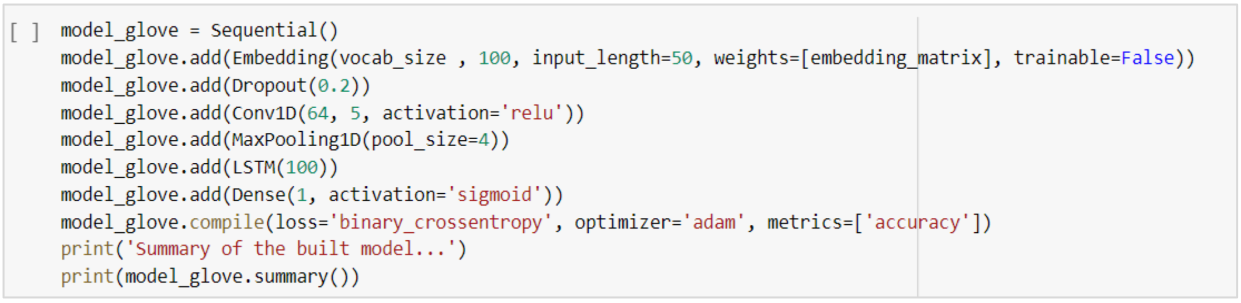
```
[ ]  model = Sequential()
     model.add(embedding_layer)
     model.add(LSTM(64, dropout=0.2, recurrent_dropout=0.25))
     model.add(Dense(1, activation='sigmoid'))

     model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])

     print('Summary of the built model...')
     print(model.summary())
```

Importing required packages for CNN model



```
[ ]  from keras.preprocessing.text import Tokenizer
     from keras.preprocessing.sequence import pad_sequences
     from keras.models import Sequential
     from keras.layers import Dense, Flatten, LSTM, Conv1D, MaxPooling1D, Dropout, Activation
     from keras.layers.embeddings import Embedding
     import plotly.offline as py
     import plotly.graph_objs as go
     py.init_notebook_mode(connected=True)
     import nltk
     import string
     import numpy as np
     import pandas as pd
     from nltk.corpus import stopwords
     from sklearn.manifold import TSNE
```

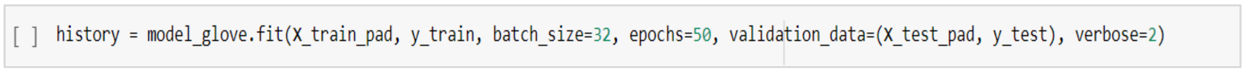
Build the CNN model



```
[ ]  model_glove = Sequential()
     model_glove.add(Embedding(vocab_size , 100, input_length=50, weights=[embedding_matrix], trainable=False))
     model_glove.add(Dropout(0.2))
     model_glove.add(Conv1D(64, 5, activation='relu'))
     model_glove.add(MaxPooling1D(pool_size=4))
     model_glove.add(LSTM(100))
     model_glove.add(Dense(1, activation='sigmoid'))
     model_glove.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
     print('Summary of the built model...')
     print(model_glove.summary())
```

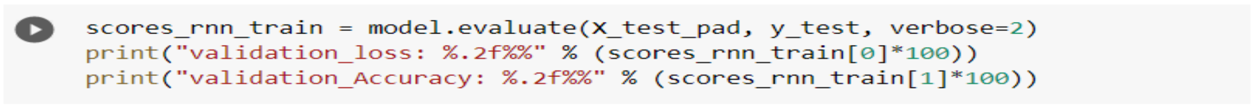
9.2.6. TRAINING THE MODELS

Train the RNN model



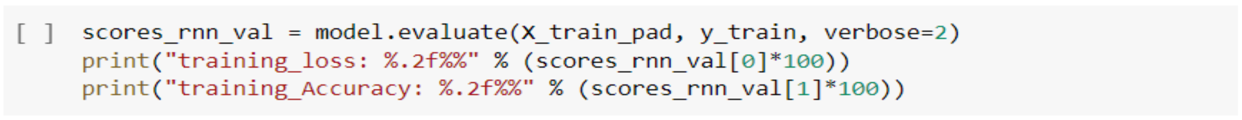
```
[ ] history = model_glove.fit(X_train_pad, y_train, batch_size=32, epochs=50, validation_data=(X_test_pad, y_test), verbose=2)
```

Validation loss and accuracy



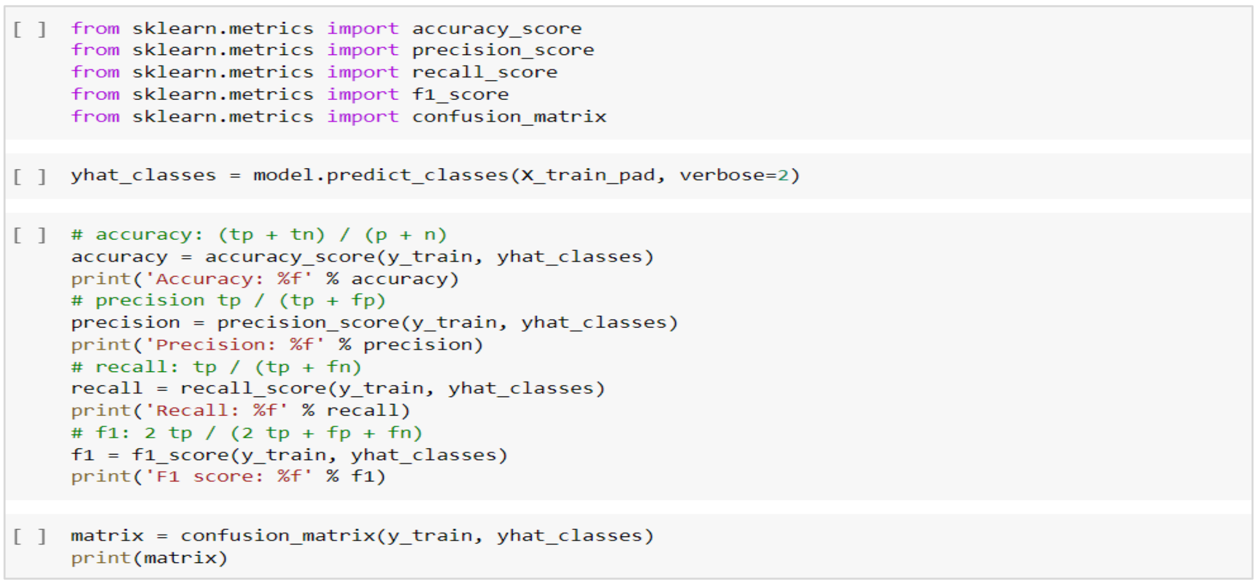
```
scores_rnn_train = model.evaluate(X_test_pad, y_test, verbose=2)
print("validation_loss: %.2f%%" % (scores_rnn_train[0]*100))
print("validation_Accuracy: %.2f%%" % (scores_rnn_train[1]*100))
```

Training loss and accuracy



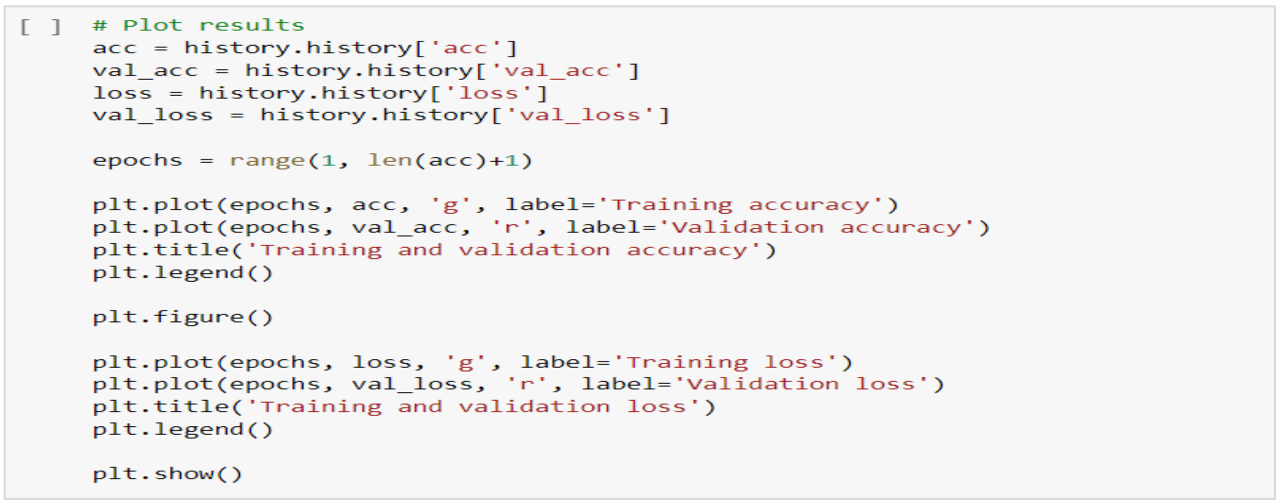
```
[ ]  scores_rnn_val = model.evaluate(X_train_pad, y_train, verbose=2)
     print("training_loss: %.2f%%" % (scores_rnn_val[0]*100))
     print("training_Accuracy: %.2f%%" % (scores_rnn_val[1]*100))
```

Performance metrics of RNN model



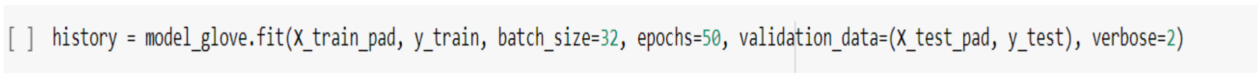
```
[ ]  from sklearn.metrics import accuracy_score
     from sklearn.metrics import precision_score
     from sklearn.metrics import recall_score
     from sklearn.metrics import f1_score
     from sklearn.metrics import confusion_matrix

[ ]  yhat_classes = model.predict_classes(X_train_pad, verbose=2)

[ ]  # accuracy: (tp + tn) / (p + n)
     accuracy = accuracy_score(y_train, yhat_classes)
     print('Accuracy: %f' % accuracy)
     # precision tp / (tp + fp)
     precision = precision_score(y_train, yhat_classes)
     print('Precision: %f' % precision)
     # recall: tp / (tp + fn)
     recall = recall_score(y_train, yhat_classes)
     print('Recall: %f' % recall)
     # f1: 2 tp / (2 tp + fp + fn)
     f1 = f1_score(y_train, yhat_classes)
     print('F1 score: %f' % f1)

[ ]  matrix = confusion_matrix(y_train, yhat_classes)
     print(matrix)
```

Plotting graph



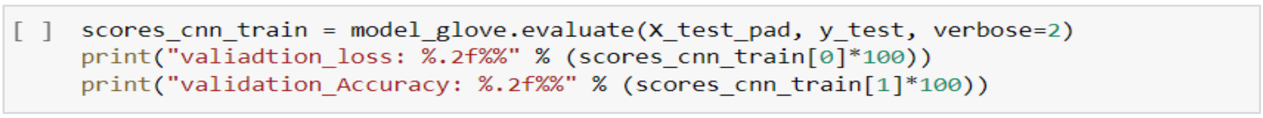
```
[ ]  # Plot results
     acc = history.history['acc']
     val_acc = history.history['val_acc']
     loss = history.history['loss']
     val_loss = history.history['val_loss']

     epochs = range(1, len(acc)+1)

     plt.plot(epochs, acc, 'g', label='Training accuracy')
     plt.plot(epochs, val_acc, 'r', label='Validation accuracy')
     plt.title('Training and validation accuracy')
     plt.legend()

     plt.figure()

     plt.plot(epochs, loss, 'g', label='Training loss')
     plt.plot(epochs, val_loss, 'r', label='Validation loss')
     plt.title('Training and validation loss')
     plt.legend()

     plt.show()
```

Training the CNN model



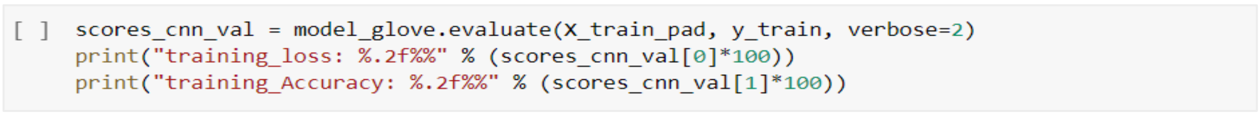
```
history = model_glove.fit(X_train_pad, y_train, batch_size=32, epochs=50, validation_data=(X_test_pad, y_test), verbose=2)
```

Validation loss and accuracy



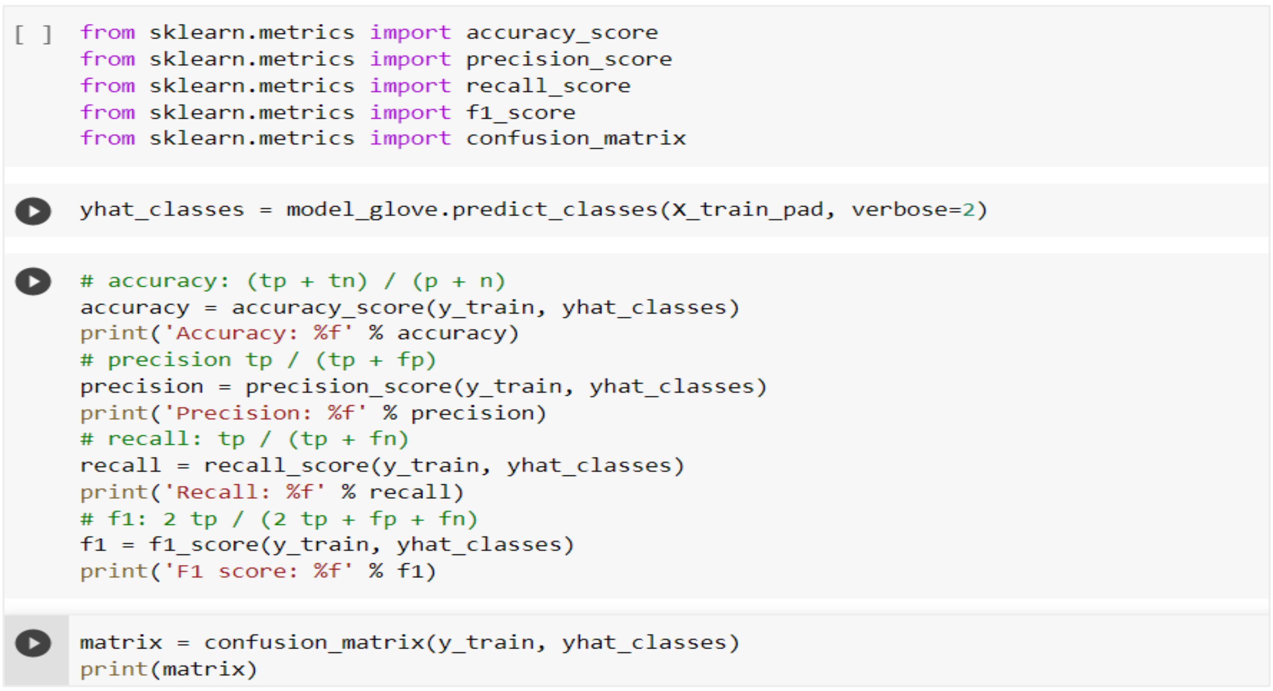
```
scores_cnn_train = model_glove.evaluate(X_test_pad, y_test, verbose=2)
print("valiadtion_loss: %.2f%%" % (scores_cnn_train[0]*100))
print("validation_Accuracy: %.2f%%" % (scores_cnn_train[1]*100))
```

Training loss and accuracy



```
scores_cnn_val = model_glove.evaluate(X_train_pad, y_train, verbose=2)
print("training_loss: %.2f%%" % (scores_cnn_val[0]*100))
print("training_Accuracy: %.2f%%" % (scores_cnn_val[1]*100))
```

Performance metrics of CNN



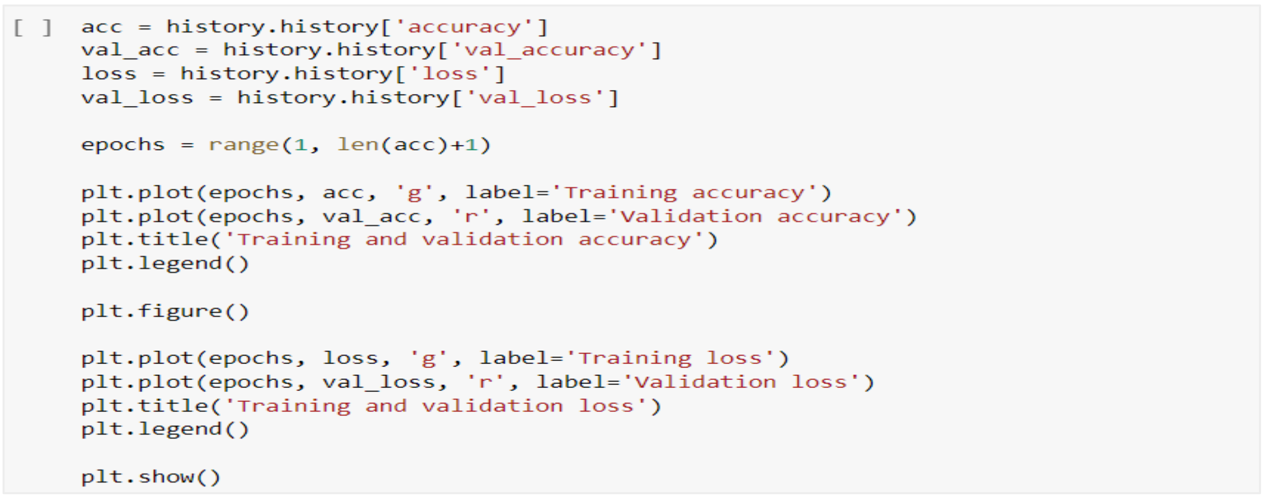
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
```

```
yhat_classes = model_glove.predict_classes(X_train_pad, verbose=2)
```

```
# accuracy: (tp + tn) / (p + n)
accuracy = accuracy_score(y_train, yhat_classes)
print('Accuracy: %f' % accuracy)
# precision tp / (tp + fp)
precision = precision_score(y_train, yhat_classes)
print('Precision: %f' % precision)
# recall: tp / (tp + fn)
recall = recall_score(y_train, yhat_classes)
print('Recall: %f' % recall)
# f1: 2 tp / (2 tp + fp + fn)
f1 = f1_score(y_train, yhat_classes)
print('F1 score: %f' % f1)
```

```
matrix = confusion_matrix(y_train, yhat_classes)
print(matrix)
```

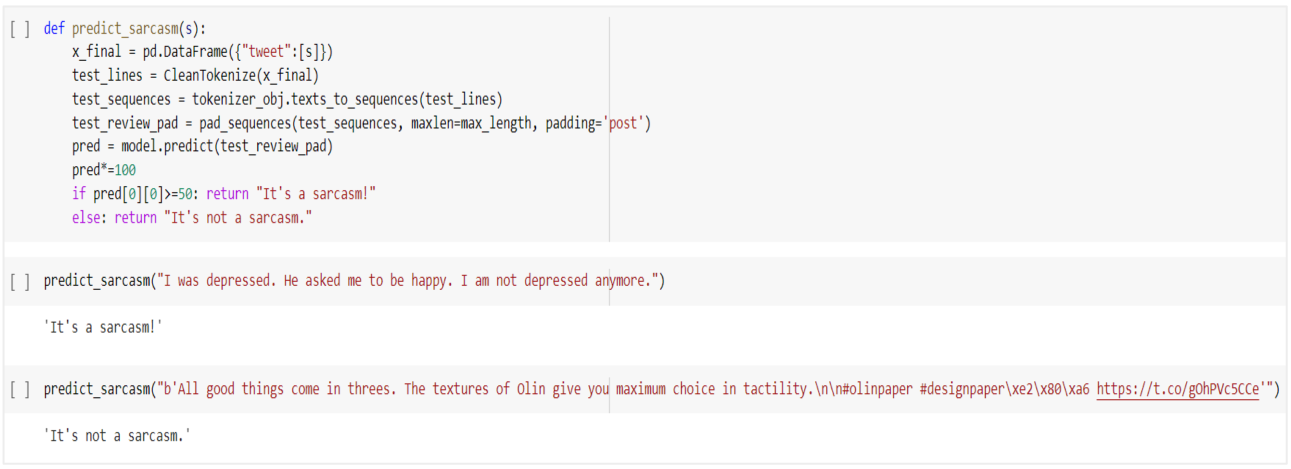
Plotting graph



```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc)+1)

plt.plot(epochs, acc, 'g', label='Training accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'g', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

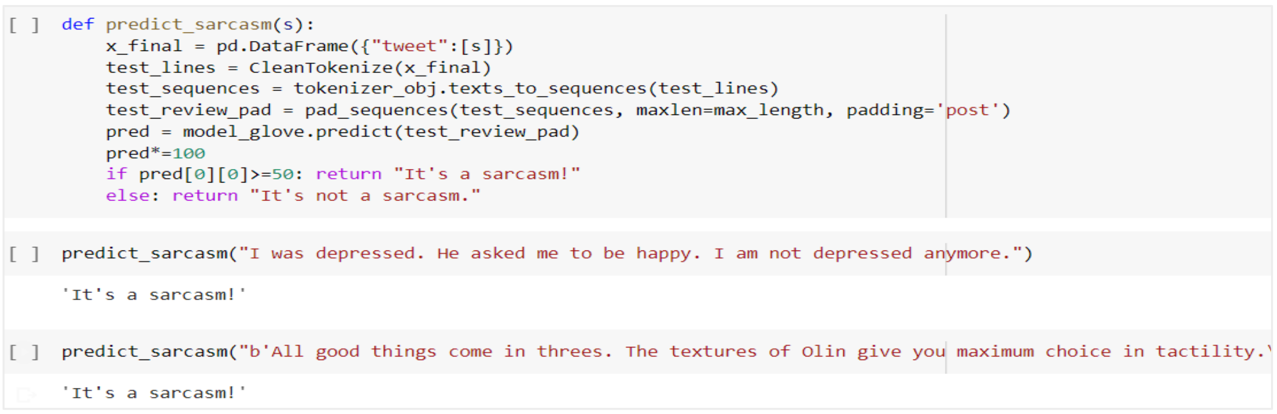
9.2.7. TESTING THE MODELS

Testing RNN model



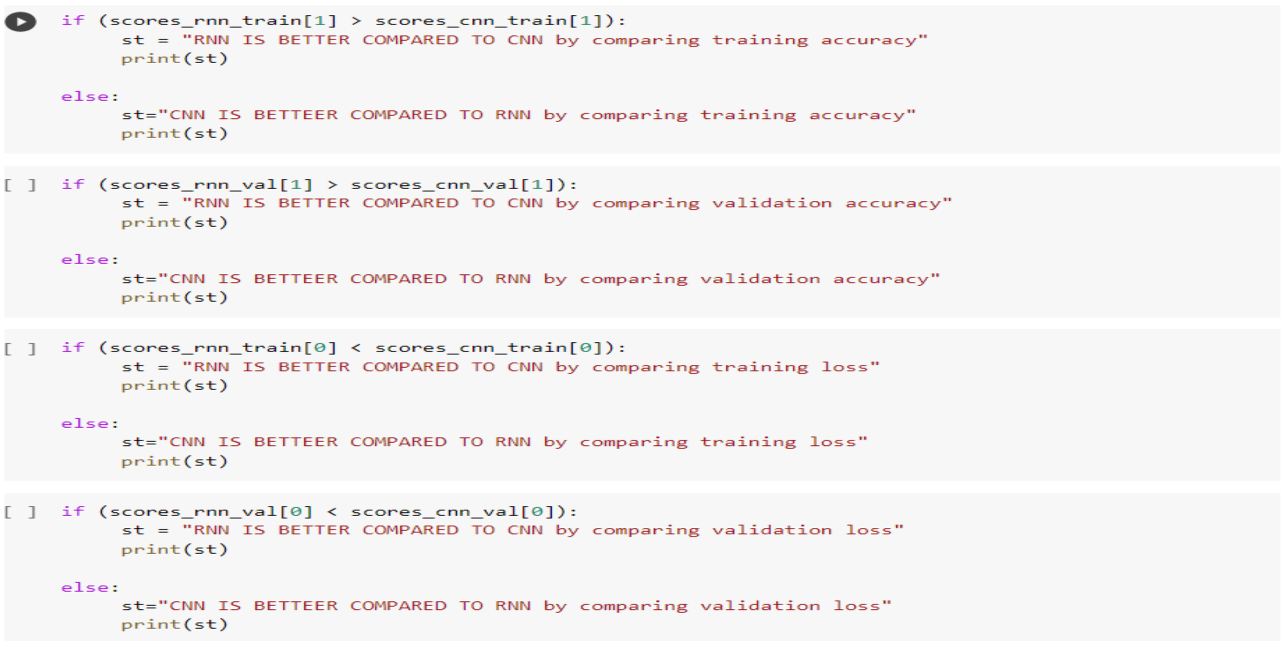
```
[ ] def predict_sarcasm(s):
        x_final = pd.DataFrame({"tweet":[s]})
        test_lines = CleanTokenize(x_final)
        test_sequences = tokenizer_obj.texts_to_sequences(test_lines)
        test_review_pad = pad_sequences(test_sequences, maxlen=max_length, padding='post')
        pred = model.predict(test_review_pad)
        pred*=100
        if pred[0][0]>=50: return "It's a sarcasm!"
        else: return "It's not a sarcasm."

[ ] predict_sarcasm("I was depressed. He asked me to be happy. I am not depressed anymore.")

    'It's a sarcasm!'

[ ] predict_sarcasm("b'All good things come in threes. The textures of Olin give you maximum choice in tactility.\n\n#olinpaper #designpaper\xe2\x80\xa6 https://t.co/gOhPVc5CCe'")

    'It's not a sarcasm.'
```

Testing CNN model



```
[ ] def predict_sarcasm(s):
        x_final = pd.DataFrame({"tweet":[s]})
        test_lines = CleanTokenize(x_final)
        test_sequences = tokenizer_obj.texts_to_sequences(test_lines)
        test_review_pad = pad_sequences(test_sequences, maxlen=max_length, padding='post')
        pred = model_glove.predict(test_review_pad)
        pred*=100
        if pred[0][0]>=50: return "It's a sarcasm!"
        else: return "It's not a sarcasm."

[ ] predict_sarcasm("I was depressed. He asked me to be happy. I am not depressed anymore.")

    'It's a sarcasm!'

[ ] predict_sarcasm("b'All good things come in threes. The textures of Olin give you maximum choice in tactility.\

    'It's a sarcasm!'
```

Comparing RNN and CNN



```
    if (scores_rnn_train[1] > scores_cnn_train[1]):
        st = "RNN IS BETTER COMPARED TO CNN by comparing training accuracy"
        print(st)

    else:
        st="CNN IS BETTEER COMPARED TO RNN by comparing training accuracy"
        print(st)

[ ] if (scores_rnn_val[1] > scores_cnn_val[1]):
        st = "RNN IS BETTER COMPARED TO CNN by comparing validation accuracy"
        print(st)

    else:
        st="CNN IS BETTEER COMPARED TO RNN by comparing validation accuracy"
        print(st)

[ ] if (scores_rnn_train[0] < scores_cnn_train[0]):
        st = "RNN IS BETTER COMPARED TO CNN by comparing training loss"
        print(st)

    else:
        st="CNN IS BETTEER COMPARED TO RNN by comparing training loss"
        print(st)

[ ] if (scores_rnn_val[0] < scores_cnn_val[0]):
        st = "RNN IS BETTER COMPARED TO CNN by comparing validation loss"
        print(st)

    else:
        st="CNN IS BETTEER COMPARED TO RNN by comparing validation loss"
        print(st)
```

10. RESULT AND COMPARISION

☆ Accuracy

Accuracy is one metric for evaluating classification models. Informally, accuracyis the fraction of predictions our model got right. It shows the overall accuracy of the instances which are correctly classified to the total number of the instances. It is calculated by the following formula:

$$\boldsymbol{Accuracy = \frac{TP + TN}{TP + TN + FP + FN}}$$

Where, TP = true positive, TN = true negative, FP = false positive, FN = false negative.

☆ Confusion Matrix

A confusion matrix is a performance measurement technique for Machine learning classification. It is a kind of table which helps you to the know the performanceof the classification model on a set of test data for that the true values are known.

☆ Precision

It represents the percentage of relevant searched sarcastic tweets. That is, it measures the amount of tweets categorized as sarcasm against the total amount of tweets classified as sarcasm. It is calculated by the following formula:

$$\boldsymbol{Precision = \frac{TP}{TP + FP}}$$

☆ Recall

It represents the percentage of relevant sarcastic tweets that have been searched. That is, against the total amount of sarcastic tweets, measured the amount of tweets that are normally classified as sarcastic. It is calculated by the following formula:

$$\boldsymbol{Recall = \frac{TP}{TP+FN}}$$

☆ F1-score

F1 score is a measure of accuracy that can be interpreted as a weighted average of accuracy and recall:

$$\boldsymbol{F1 - score = \frac{2 * precision * recall}{precision + recall}}$$

10.1. GRAPHS

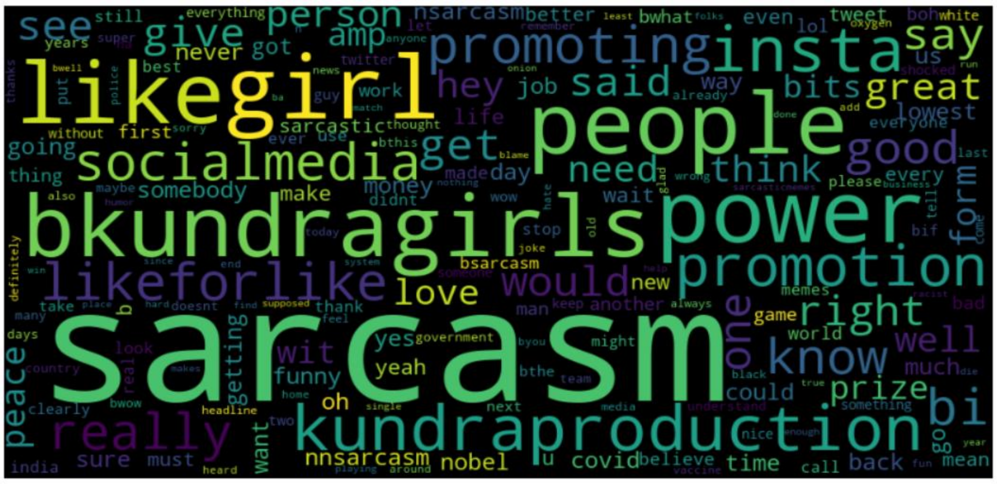


FIGURE – 10.1.1 : Word cloud graph for sarcastic tweets

Figure 10.1.1 shows a word cloud graph for sarcastic tweets. The graph is plotted for highly repeated words. It represents no of unique words present in it.

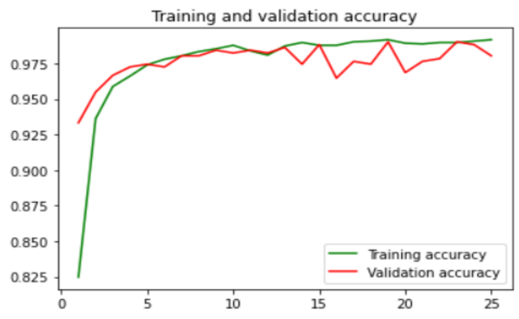


FIGURE – 10.1.2 : Accuracy vs epoch for the model

The figure 10.1.2 shows the training and validation accuracy of the RNN model. It takes X-axis as epochs and Y-axis as accuracy values. From the graph , it is observed that as epochs values increases accuracy value increase. The accuracy of the model during training and validating comes out to be 97% and 98%.

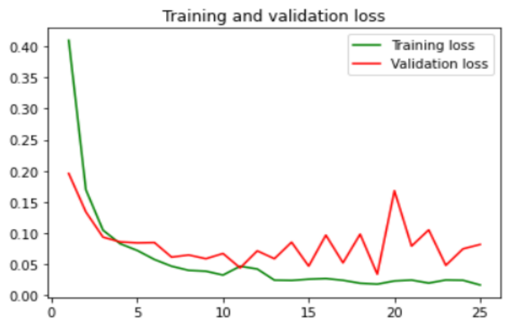


FIGURE -10.1.3 : Training and validation loss vs epochs of the model

The figure 10.1.3 shows the training and validation loss of the RNN model. It takes X-axis as epochs and Y-axis as loss values. From the graph , it is observed that as epochs values increases loss value decrease. The loss of the model during training and validating comes out to be 0.03 and 0.07.

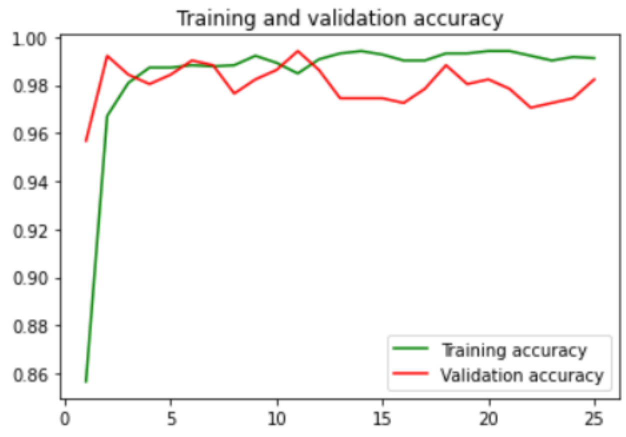


FIGURE – 10.1.4 : Accuracy vs epoch for the model

The figure 10.1.4 shows the training and validation accuracy of the CNN model. It takes X-axis as epochs and Y-axis as accuracy values. From the graph , it is observed that as epochs values increases accuracy value increase. The accuracy of the model during training and validating comes out to be 99.9% and 98%.

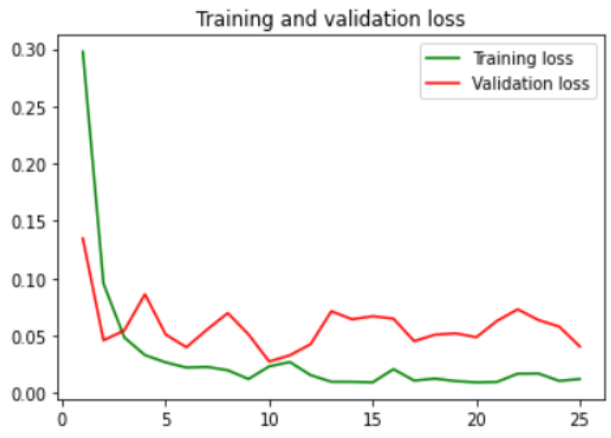


FIGURE -10.1.5 : Training and validation loss vs epochs of the model

The figure 10.1.5 shows the training and validation loss of the CNN model. It takes X-axis as epochs and Y-axis as loss values. From the graph , it is observed that as epochs values increases loss value decrease. The loss of the model during training and validating comes out to be 0.01 and 0.04.

10.2. TABLE OF INFERENCE

RNN MODEL

	Data set split up as 80 % of train and 20% of test			*Train data (10%) and test data (90%)*
	EPOCH=15	**Epoch=25**	**EPOCH=50**	**EPOCH = 25**
Training Accuracy	99.36%	99.36%	99.36%	100%
Taining Loss	1.26%	1.29%	1.10%	0.46%
Validation Accuracy	98.04%	98.04%	96.66%	90.96%
Validation loss	5.46%	8.17%	17.2%	37.36%
Precision	100%	99.9%	99%	100%
Recall	99.07%	99.1%	100%	100%
F1 score	99.5%	99.5%	99.5%	100%

CNN MODEL

	Data set split up as 80 % of train and 20% of test			*Train data (10%) and test data (90%)*
	EPOCH-15	**EPOCH-25**	**EPOCH-50**	**Epoch-25**
Training Accuracy	99.41%	99.41%	99.36%	100%
Taining Loss	1.04%	0.90%	0.99%	0.13%
Validation Accuracy	98.82%	98.23%	97.45%	92.05%
Validation loss	2.44%	4.04%	17.7%	44.41%
Precision	99.1%	100%	99%	100%
Recall	99.9%	99.1%	100%	100%
F1 score	99.5%	99.5%	99.7%	100%

10.3. COMPARING MODELS

EPOCH-15

RNN **CNN**

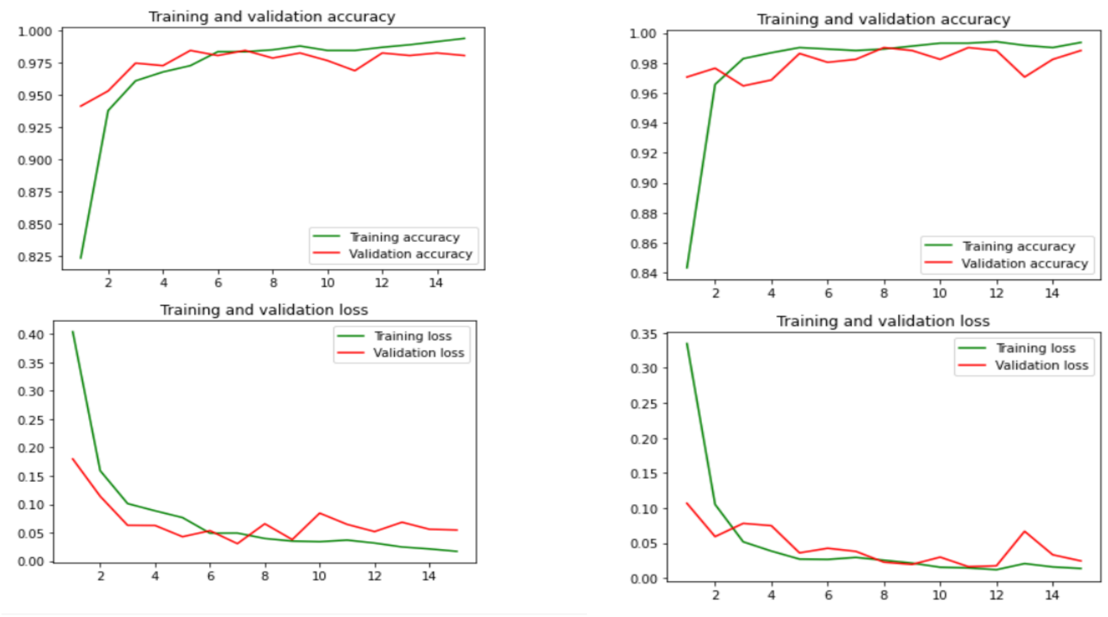


FIGURE-10.3.1 : Accuracy,loss vs epochs for RNN and CNN (Epoch-15)

Figure-10.3.1 shows the training and validation accuracy and loss for both RNN and CNN when epoch is 15 . X-axis is taken as epoch and Y-axis is taken as accuracy and loss.As epoch value increases accuracy increases and loss value decreases .When comparing both RNN and CNN neural networks CNN shows high accuracy.

EPOCH-25

RNN **CNN**

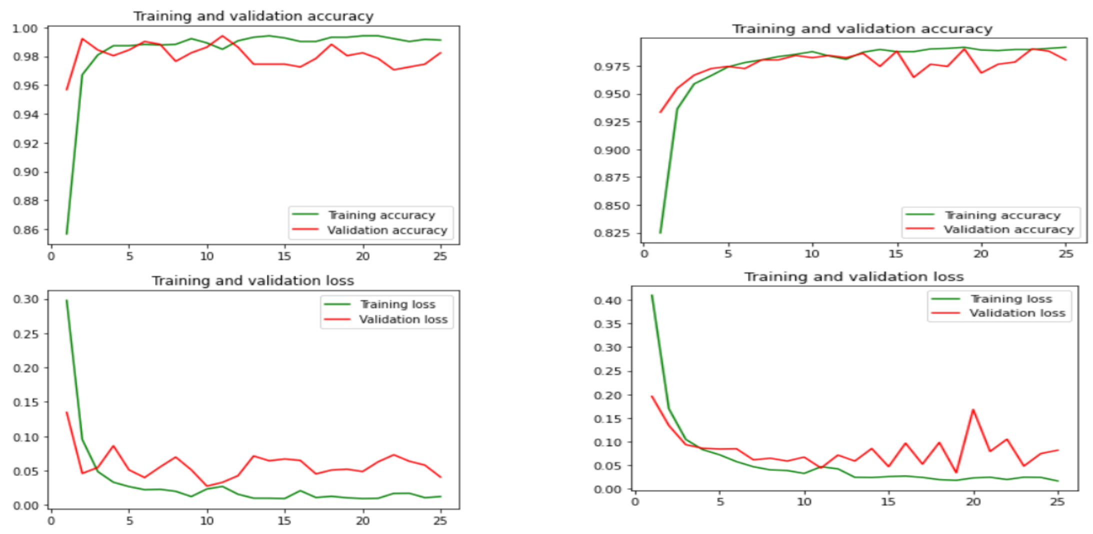
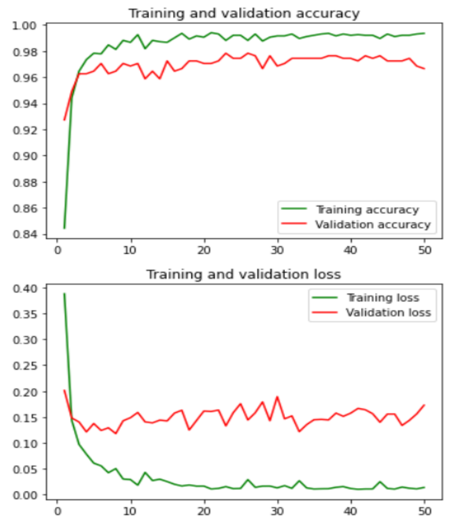


FIGURE-10.3.2 : Accuracy,loss vs epochs for RNN and CNN (Epoch-25)

Figure-10.3.2 shows the training and validation accuracy and loss for both RNN and CNN when epoch is 25 . X-axis is taken as epoch and Y-axis is taken as accuracy and loss. As epoch value increases accuracy increases and loss value decreases .When comparing both RNN and CNN neural networks CNN shows high accuracy.

EPOCH-50

RNN



CNN

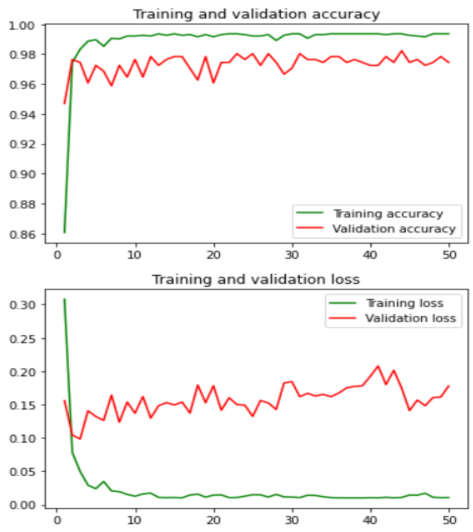
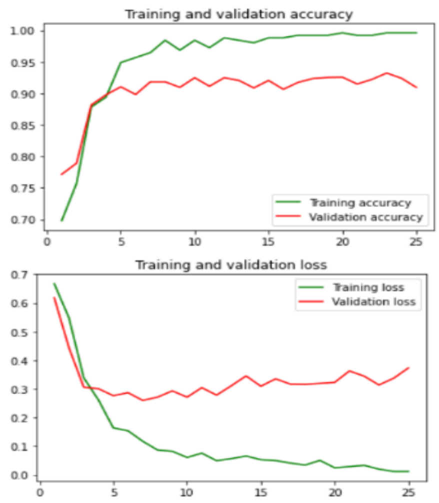


FIGURE-10.3.3 : Accuracy,loss vs epochs for RNN and CNN (Epoch-50)

Figure-10.3.3 shows the training and validation accuracy and loss for both RNN and CNN when epoch is 50 . X-axis is taken as epoch and Y-axis is taken as accuracy and loss.As epoch value increases accuracy increases and loss value decreases .When comparing both RNN and CNN neural networks CNN shows high accuracy.

Train data as 10%

RNN



CNN

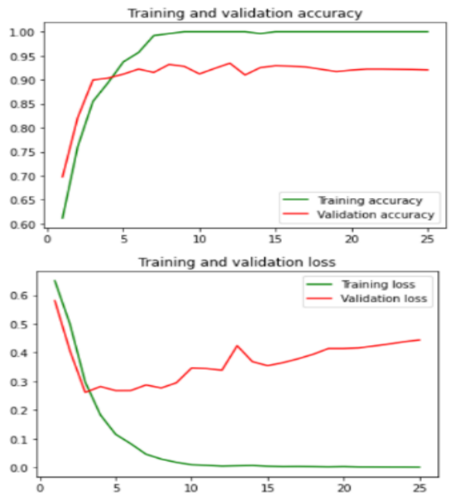
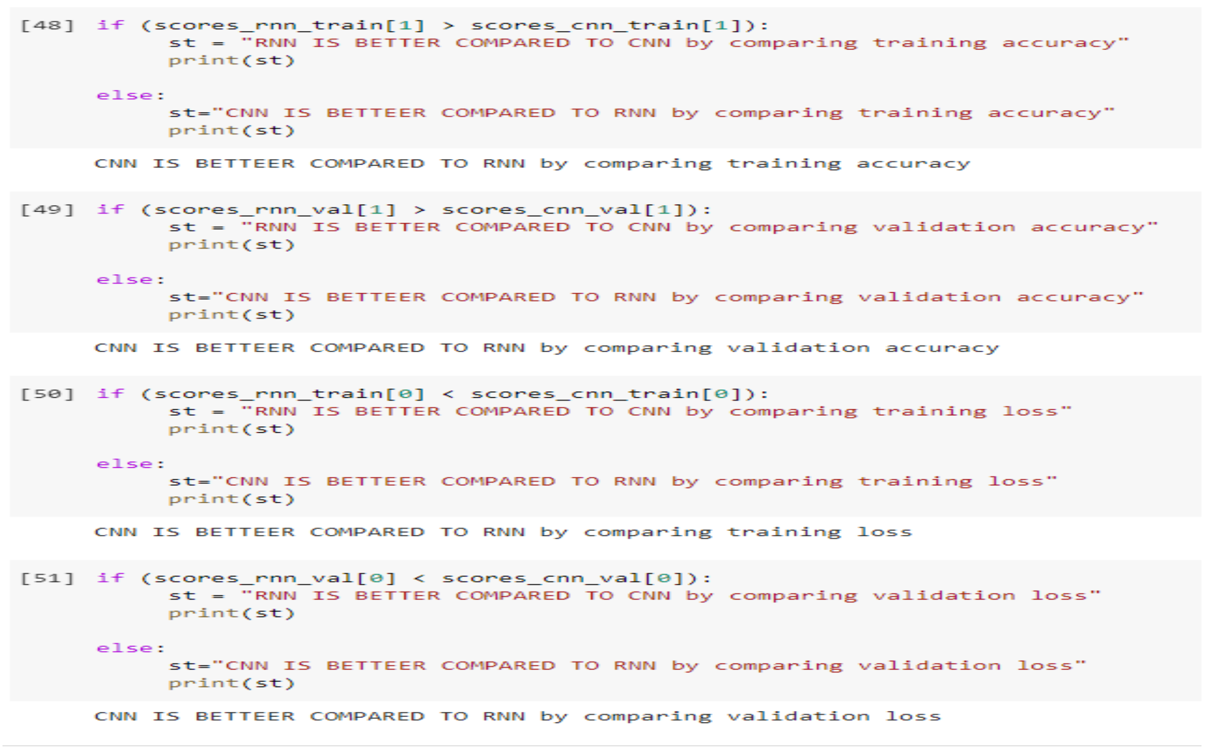


FIGURE-10.3.4 : Accuracy,loss vs epochs for RNN and CNN (Train data as 10%)

Figure-10.3.4 shows the training and validation accuracy and loss for both RNN and CNN taking train data as 10% and epoch is 25 . X-axis is taken as epoch and Y-axis is taken as accuracy and loss.As epoch value increases accuracy increases and loss value decreases. When comparing both RNN and CNN neural networks CNN shows high accuracy.



```
[48] if (scores_rnn_train[1] > scores_cnn_train[1]):
         st = "RNN IS BETTER COMPARED TO CNN by comparing training accuracy"
         print(st)

     else:
         st="CNN IS BETTEER COMPARED TO RNN by comparing training accuracy"
         print(st)

     CNN IS BETTEER COMPARED TO RNN by comparing training accuracy

[49] if (scores_rnn_val[1] > scores_cnn_val[1]):
         st = "RNN IS BETTER COMPARED TO CNN by comparing validation accuracy"
         print(st)

     else:
         st="CNN IS BETTEER COMPARED TO RNN by comparing validation accuracy"
         print(st)

     CNN IS BETTEER COMPARED TO RNN by comparing validation accuracy

[50] if (scores_rnn_train[0] < scores_cnn_train[0]):
         st = "RNN IS BETTER COMPARED TO CNN by comparing training loss"
         print(st)

     else:
         st="CNN IS BETTEER COMPARED TO RNN by comparing training loss"
         print(st)

     CNN IS BETTEER COMPARED TO RNN by comparing training loss

[51] if (scores_rnn_val[0] < scores_cnn_val[0]):
         st = "RNN IS BETTER COMPARED TO CNN by comparing validation loss"
         print(st)

     else:
         st="CNN IS BETTEER COMPARED TO RNN by comparing validation loss"
         print(st)

     CNN IS BETTEER COMPARED TO RNN by comparing validation loss
```

FIGURE-10.3.5 : Comparision of CNN and RNN

From the figure-10.3.5 , models is trained by using normal classifiers,but we are using neural networks which has higher performance than normal classifiers.In existing system we have used RNN neural network which is the first algorithm that remembers its input, due to an internal memory, which makes it perfectly suited for machine learning problems that involve sequential data.For proposed system we have used CNN(Convolutional Neural Network).By comparing the both networks(RNN and CNN) training accuracy,training loss,validation accuracy and validation loss. We discovered CNN neural network has high performance metrics.

11. CONCLUSION AND FUTURE ENHANCEMENTS

Sarcasm detection on twitter data provides opinion about public opinion on the recent trend and events. The system uses sarcastic tweets, 3,000 tweets containing #sarcasm, and #regular dataset. The system uses RNN and CNN Neural Networks. Various graphs have been shown of training accuracy,training loss,validation accuracy and validation loss . A total of 3000 tweets were used for calculating the accuracy of RNN and CNN networks. The accuracy of CNN is 1.00 and of RNN is 0.99. The approach has shown good results and it is observed that Convolutional Neural Network has more accuracy than Recurrent.

This model detects the sarcasm in the user's tweets . The future direction includes detecting the user's features since we are using user's data as training set. Also the system can be implemented for multi-label data. .So, in future Neural Network, Genetic Algorithm and Pattern-based approach can be combined for more accuracy. Using sarcasm detection, the e-commerce websites can make their system more user-personalized.

12. REFERENCES

1 . Patrick Adolf Telnoni, Reza Budiawan, Mutia Qana'a." Comparison of Machine Learning Classification Method on Text-based Case in Twitter".In 978-1-7281-4880-9/19/$31.00 2019 IEEE

2 . Neha Pawar and Sukhada Bhingarkar." Machine Learning based Sarcasm Detection on Twitter Data".In the Fifth International Conference on Communication and Electronics Systems (ICCES 2020) IEEE Conference Record # 48766; IEEE Xplore ISBN: 978-1-7281-5371-1.

3 . Azilawati Azizan,Nurul Najwa SK Abdul Jamal,Mohammad Nasir Abdullah,Masurah Mohamad,Nurkhairizan Khairudin."Lexicon-Based Sentiment Analysis for Movie Review Tweets".In 978-1-7281-3041-5/19/$31.00 2019 IEEE

4 . Soujanya Poria ,Erik Cambria ,Devamanyu Hazarika, Prateek Vij" A Deeper Look into Sarcastic Tweets Using Deep Convolutional Neural Networks" Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers, pages 1601–1612, Osaka, Japan, December 11-17 2016.

13.APPENDIX A

The undersigned acknowledge they have completed implementing the project “Sarcasm detection on twitter data ” and agree with the approach it presents.

Signature:		Date:	24/05/2021
Name:	N.A.Nisha		
Signature:		Date:	24/05/2021
Name:	Tanguturi Naga Lakshmi Sucharitha		

14.APPENDIX B : References

The following table summarizes the research papers referenced in this document.

Document Name and Version	Description	Location
Comparison of Machine Learning Classification Method on Text-based Case in Twitter	The performance of classification methods for text-based data. It finds the best choices of classification method in terms of accuracy and training time.	https://www.researchgate.net/publication/338943842_Comparison_of_Machine_Learning_Classification_Method_on_Text-based_Case_in_Twitter
Machine learning based Sarcasm detection on Twitter Data	Mood analysis and opinion mining rely on emotional words to detect their polarity in a text(that is whether it relates to "positivity" or " negativity" in its thread).It takes four sets of features that include a lot of specific sarcasm and classify tweets as sarcastic and non sarcastic.	https://www.researchgate.net/publication/342853963_Machine_Learning_based_Sarcasm_Detection_Twitter_Data
A Deeper Look into Sarcastic Tweets Using Deep Convolutional Neural Networks	Pre-trained sentiment, emotion and personality models for identifying sarcastic text using CNN, which are found to be very effective for sarcasm detection.	https://sentic.net/sarcasm-detection-with-deep-convolutional-neural-networks.pdf

15.APPENDIX C : Key terms

The following table provides definitions for terms relevant to this document.

Terms	Definition
Sarcasm	**Sarcasm** is a form of communication that is intended to mock or harass someone by using words with the **opposite of their literal meaning .**
RNN	**Recurrent Neural Network(RNN)** are a type of Neural Network where the **output from previous step are fed as input to the current step**. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words.
CNN	**CNNs** are regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean **fully connected networks**, that is, each neuron in one layer is connected to all neurons in the next layer. The "full connectivity" of these networks make them prone to overfitting data.