International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

TEXT CLASSIFICATION USING DEEP LEARNING

A. Santhoshini[1], A. Shirisha[2], T. Sai meghana[3]
B. Department of Computer Science and Engineering, Stanley College of Engineering and Technology for Women, Telangana, India

Abstract We introduce a novel approach for automatically classifying the sentiment of Twitter messages using NLP and deep learning. These messages are classified as either positive or negative with respect to a query term using sentiment analysis. This is useful for consumers who want to research the sentiment of products before purchase, or companies that want to monitor the public sentiment of their brands. There is no previous research on classifying sentiment of messages on microblogging services like Twitter. We present the results of deep learning algorithm for classifying the sentiment of Twitter messages using distant supervision. Our training data consists of Twitter messages. This type of training data is abundantly available and can be obtained through automated means. We show that deep learning algorithm LSTM which is a variant of recurrent neural network (RNN) have accuracy approximately 80% when trained with this data. This paper also describes the preprocessing steps needed in order to achieve high accuracy. The main contribution of this paper is the idea of using tweets for distant supervised learning.

Keywords: Twitter, NLP, Sentiment Analysis, Deep Learning, RNN, LSTM

1. Introduction

1.1 About Project

Text classification is the process of assigning tags or categories to text according to its content. It's one of the fundamental tasks in natural language processing (NLP) with broad applications such as sentiment analysis, topic labelling, spam detection, and intent detection. Unstructured data in the form of text is everywhere: emails, chats, web pages, social media, support tickets, survey responses, and more. Text can be an extremely rich source of information but, extracting insights from it can be hard and time-consuming due to its unstructured nature. Businesses are turning to text classification for structuring text in a fast and cost-efficient way to enhance decision-making and automate processes.

1.2 Objectives of the Project

The experiment will evaluate the performance of popular deep learning models, such as LSTM (Long Short-Term Memory) on sentiment140 dataset, which contains 1,600,000 tweets extracted using the twitter API classification dataset. The result shows the robustness of word embedding as a feature extractor to the model in making a better final prediction and the effectiveness of the LSTM (Long Short-Term Memory) in achieving good performance and accuracy.

1.3 Scope of the project

Scope of our project is to achieve beautiful outcomes like accuracy by classifying the text using deep learning model. Annotate the tweets automatically which are used to detect the sentiment (NLP-Sentiment Analysis). Later, training the model using deep learning model LSTM (Long Short-term Memory).

1.4 Twitter Introduction

Twitter is a popular microblogging service where users create status messages (called "tweets"). These tweets sometimes express opinions about different topics. We propose a method to automatically extract sentiment

(positive or negative) from a tweet. This is very useful because it allows feedback to be aggregated without manual intervention.
There has been a large amount of research in the area of sentiment classification. Traditionally most of it has focused on classifying larger pieces of text, like reviews.

Tweets (and microblogs in general) are different from reviews primarily because of their purpose: while reviews represent summarized thoughts of authors, tweets are more casual and limited to 140 characters of text. Generally, tweets are not as thoughtfully composed as reviews. Yet, they still offer companies an additional avenue to gather feedback. There has been some work by researchers in the area of phrase level and sentence level sentiment classification recently.

In order to train a classifier, supervised learning usually requires hand-labeled training data. With the large range of topics discussed on Twitter, it would be very difficult to manually collect enough data to train a sentiment classifier for tweets. Our solution is to use distant supervision, in which our training data consists of tweets with emoticons. The emoticons serve as noisy labels. For example, :) in a tweet indicates that the tweet contains positive sentiment and :(indicates that the tweet contains negative sentiment. With the help of the Twitter API, it is easy to extract large amounts of tweets with emoticons in them. This is a significant improvement over the many hours it may otherwise take to hand-label training data.

1.5 Characteristics of tweets

Twitter messages have many unique attributes, which differentiates our research from previous research:
Length: The maximum length of a Twitter message is 140 characters. From our training set, we calculate that the average length of a tweet is 14 words or 78 characters. This is very different from the previous sentiment classification research that focused on classifying longer bodies of work, such as movie reviews.
Data availability: Another difference is the magnitude of data available. With the Twitter API, it is very easy to collect millions of tweets for training. In past research, tests only consisted of thousands of training items.
Language model: Twitter users post messages from many different media, including their cell phones. The frequency of misspellings and slang in tweets is much higher than in other domains.
Domain: Twitter users post short messages about a variety of topics unlike other sites which are tailored to a specific topic. This differs from a large percentage of past research, which focused on specific domains such as movie reviews.

1.5 Defining Sentiment

For the purpose of our research, we define sentiment to be "a personal positive or negative feeling." Many times, it is unclear if a tweet contains a sentiment. For these cases, we use the following litmus test: If the tweet could ever appear as a frontpage newspaper headline or as a sentence in Wikipedia, then it belongs in the neutral class. For example, the following tweet is considered neutral because it could have appeared as a newspaper headline, even though it projects an overall negative feeling about General Motors: RT @Finance Info Bankruptcy filing could put GM on road to profits (AP) http://cli.gs/9ua6Sb #Finance.

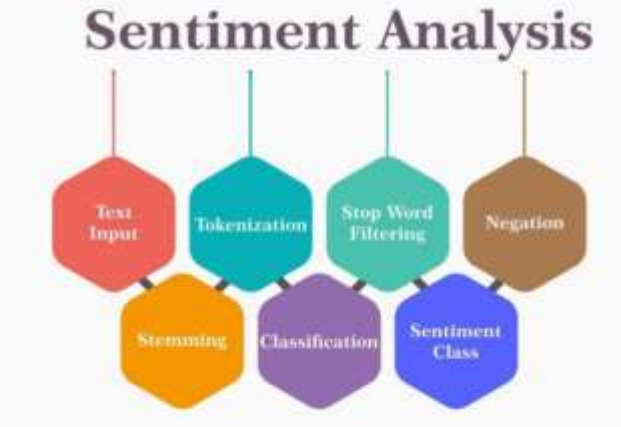
In this research, we do not consider neutral tweets in our training or testing data. We only use positive or negative tweets. Many tweets do not have sentiment, so it is a current limitation of our research to not include the neutral class.

1.6 NLP and Sentiment Analysis

Natural Language Processing or NLP is a branch of Artificial Intelligence which deal with bridging the machines understanding humans in their Natural Language. Natural Language can be in form of text or sound, which are used for humans to communicate each other. NLP can enable humans to communicate to machines in a natural way. NLP has been very successful in healthcare, media, finance, and human resource.

The most common form of unstructured data is texts and speeches. It's plenty but hard to extract useful information. If not, it would take a long time to mine the information. Written text and speech contain rich information. It's because we, as intelligent beings, use writing and speaking as the primary form of communication. NLP can analyze these data for us and do the task like sentiment analysis, cognitive assistant, span filtering, identifying fake news, and real-time language translation.

Text Classification is a process involved in Sentiment Analysis. Sentiment analysis (or opinion mining) natural language processing (NLP) technique. It is a classification of people's opinions or expressions into different sentiments. Sentiments include Positive, Neutral, and Negative, Review Ratings and Happy, Sad. Sentiment Analysis can be done on different consumer centered industries to analyze people's opinion on a particular product or subject.



Sentiment Analysis of Python

Scikit-learn is the go-to library for machine learning and has useful tools for text vectorization. Training a classifier on top of vectorizations, like frequency or tf-idf text vectorizers is quite straightforward. Scikit-learn has implementations for Support Vector Machines, Naïve Bayes, and Logistic Regression, among others.
NLTK has been the traditional NLP library for Python. It has an active community and offers the possibility to train machine learning classifiers.
SpaCy is an NLP library with a growing community. Like NLTK, it provides a strong set of low-level functions for NLP and support for training text classifiers. TensorFlow, developed by Google, provides a low-level set of tools to build and train neural networks. There's also support for text vectorization, both on traditional word frequency and on more advanced through-word embeddings.
Keras provides useful abstractions to work with multiple neural network types, like recurrent neural networks (RNNs) and convolutional neural networks (CNNs) and easily stack layers of neurons. Keras can be run on top of Tensorflow or Theano. It also provides useful tools for text classification.
PyTorch is a recent deep learning framework backed by some prestigious organizations like Facebook, Twitter, Nvidia, Salesforce, Stanford University, University of Oxford, and Uber. It has quickly developed a strong community.

1.7 Deep Learning Model – LSTM

Deep learning is based on the branch of machine learning, which is a subset of artificial intelligence. Since neural networks imitate the human brain and so deep learning will do. In deep learning, nothing is programmed explicitly. Basically, it is a machine learning class that makes use of numerous nonlinear processing units so as to perform feature extraction as well as transformation. The output from each preceding layer is taken as input by each one of the successive layers.

Deep learning models are capable enough to focus on the accurate features themselves by requiring a little guidance from the programmer and are very helpful in solving out the problem of dimensionality. Deep learning algorithms are used, especially when we have a huge no of inputs and outputs.

Since deep learning has been evolved by the machine learning, which itself is a subset of artificial intelligence and as the idea behind the artificial intelligence is to mimic the human behavior, so same is "the idea of deep learning to build such algorithm that can mimic the brain".

Deep learning is implemented with the help of Neural Networks, and the idea behind the motivation of Neural Network is the biological neurons, which is nothing but a brain cell.

LSTM

LSTM stands for Long-Short Term Memory. LSTM is a type of recurrent neural network but is better than traditional recurrent neural networks in terms of memory. Having a good hold over memorizing certain patterns LSTMs perform fairly better. As with every other NN, LSTM can have multiple hidden layers and as it passes through every layer, the relevant information is kept and all the irrelevant information gets discarded in every single cell.

One good reason to use LSTM is that it is effective in memorizing important information.
If we look and other non-neural network classification techniques they are trained on multiple word as separate inputs that are just word having no actual meaning as a sentence, and while predicting the class it will give the output according to statistics and not according to meaning. That means, every single word is classified into one of the categories.

This is not the same in LSTM. In LSTM we can use a multiple word string to find out the class to which it belongs. This is very helpful while working with Natural language processing. If we use appropriate layers of embedding and encoding in LSTM, the model will be able to find out the actual meaning in input string and will give the most accurate output class. The following code will elaborate the idea on how text classification is done using LSTM.

2. Literature survey

2.1 Existing System

Text can be an extremely rich source of information but, extracting insights from it can be hard and time-consuming due to its unstructured nature. Lot of existing systems of text classification are based on various machine learning algorithms like CNN, GCN SVM, Naïve bayes etc. who classifies the text and predicts the accuracy.

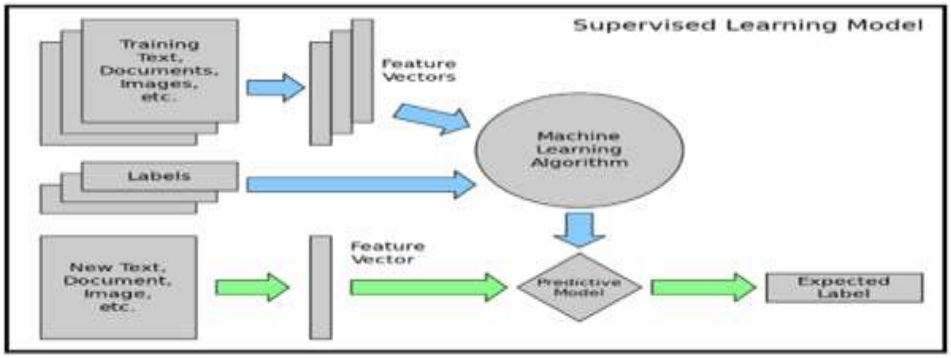
2.2 Proposed System

In proposed system, we introduce a novel approach for automatically classifying the sentiment of Twitter messages using NLP and deep learning. These messages are classified as either positive or negative with respect to a query term using sentiment analysis. We present the results of deep learning algorithm for classifying the sentiment of Twitter messages using distant supervision. This type of training data is abundantly available and can be obtained through automated means. We show that deep learning algorithm LSTM which is a variant of recurrent neural network (RNN) have accuracy approximately 80% when trained with this data.

3 Architecture

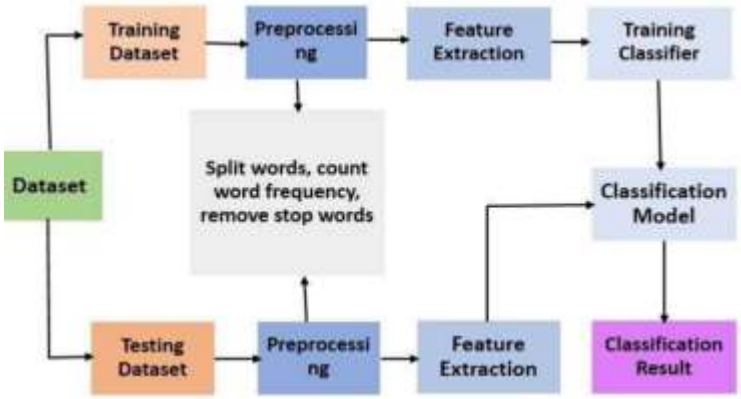
The following is the architecture of this project.

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org



Training text: It is the input text through which our supervised learning model is able to learn and predict the required class. In our case it is the dataset of Twitter messages.
Feature Vector: A feature vector is a vector that contains information describing the characteristics of the input data.
Labels: These are the predefined categories/classes that our model will predict.
ML Algorithm: It is the algorithm through which our model is able to deal with text classification (In our case: RNN-LSTM).
Predictive Model: A model which is trained on the historical dataset which can perform label.

3.1 Block Diagram



4. Flow of Code Implementation

4.1 Importing Dependencies

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

```
In [1]: import tensorflow as tf
        import matplotlib.pyplot as plt
        import pandas as pd
        import numpy as np


        import nltk
        nltk.download('stopwords')
        from nltk.corpus import stopwords
        from nltk.stem import SnowballStemmer



        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import LabelEncoder


        import re

        print("Tensorflow Version",tf.__version__)

        Tensorflow Version 2.3.0

        [nltk_data] Downloading package stopwords to
        [nltk_data]     C:\Users\Santhoshini\AppData\Roaming\nltk_data...
        [nltk_data]   Package stopwords is already up-to-date!
```

Here we are importing different libraries with different versions. Such as

- Numpy-1.16.4
- Tensor flow-2.3.0
- Matplotlib-3.3.2
- Pandas-1.1.5
- Scikitlearn-0.23.2

4.2 Data Pre-processing

In this project, we are using Sentiment-140 dataset, we named it as a LSTM.csv. It contains a labelled data of 1.6Milli the columns are without any proper names.

- Since the columns are without any proper names. we renamed them for our reference.

```
In [2]: df = pd.read_csv("LSTMSET.csv.csv",
                        encoding = 'latin',header=None)
        df.head()

Out[2]:
   0  1                            2         3         4                5
0  0  1467810369  Mon Apr 06 22:19:45 PDT 2009  NO_QUERY  _TheSpecialOne_  @switchfoot http://twitpic.com/2y1zl - Awww, t...
1  0  1467810672  Mon Apr 06 22:19:49 PDT 2009  NO_QUERY   scotthamilton  is upset that he can't update his Facebook by ...
2  0  1467810917  Mon Apr 06 22:19:53 PDT 2009  NO_QUERY        mattycus  @Kenichan I dived many times for the ball. Man...
3  0  1467811184  Mon Apr 06 22:19:57 PDT 2009  NO_QUERY         ElleCTF    my whole body feels itchy and like its on fire
4  0  1467811193  Mon Apr 06 22:19:57 PDT 2009  NO_QUERY          Karoli  @nationwideclass no, it's not behaving at all...

In [3]: df.columns = ['sentiment', 'id', 'date', 'query', 'user_id', 'text']
        df.head()

Out[3]:
   sentiment          id                          date     query     user_id                                               text
0          0  1467810369  Mon Apr 06 22:19:45 PDT 2009  NO_QUERY  _TheSpecialOne_  @switchfoot http://twitpic.com/2y1zl - Awww, t...
1          0  1467810672  Mon Apr 06 22:19:49 PDT 2009  NO_QUERY   scotthamilton  is upset that he can't update his Facebook by ...
2          0  1467810917  Mon Apr 06 22:19:53 PDT 2009  NO_QUERY        mattycus  @Kenichan I dived many times for the ball. Man...
3          0  1467811184  Mon Apr 06 22:19:57 PDT 2009  NO_QUERY         ElleCTF    my whole body feels itchy and like its on fire
4          0  1467811193  Mon Apr 06 22:19:57 PDT 2009  NO_QUERY          Karoli  @nationwideclass no, it's not behaving at all...
```

Fig-4.2 Reading the dataset and renaming the columns

- we are going to train only on text to classify its sentiment. So we can remove the rest of the useless columns. And apply sentiments to the tweets.

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

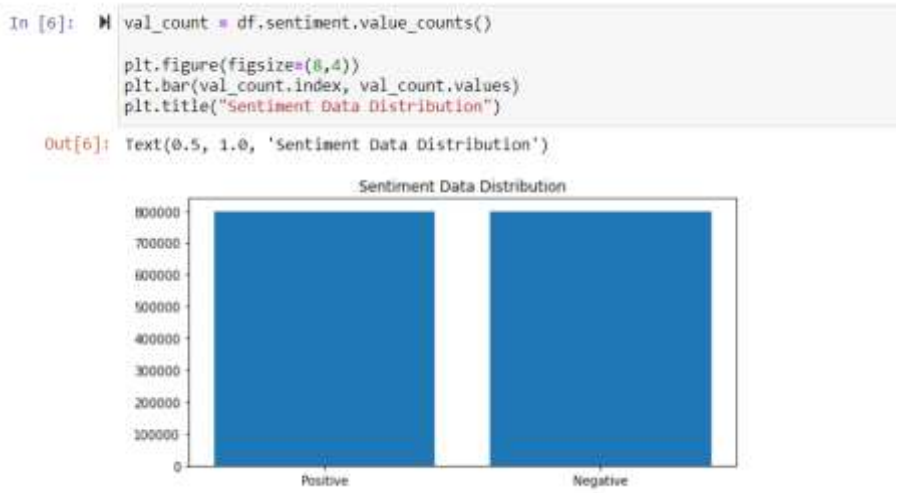
```
In [4]: df = df.drop(['id', 'date', 'query', 'user_id'], axis=1)

In [5]: lab_to_sentiment = {0:"Negative", 4:"Positive"}
        def label_decoder(label):
          return lab_to_sentiment[label]
        df.sentiment = df.sentiment.apply(lambda x: label_decoder(x))
        df.head()
```

Out[5]:

	sentiment	text
0	Negative	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	Negative	is upset that he can't update his Facebook by ...
2	Negative	@Kenichan I dived many times for the ball. Man...
3	Negative	my whole body feels itchy and like its on fire
4	Negative	@nationwideclass no, it's not behaving at all ...

Fig-4.3 Applying sentiments to the tweets

- Here we are decoding the labels. We map 0 -> Negative and 1 -> Positive as directed by the dataset description. Now that we decoded, we shall now analyze the dataset by its distribution. Because it's important that we have almost small amounts of examples for given classes.

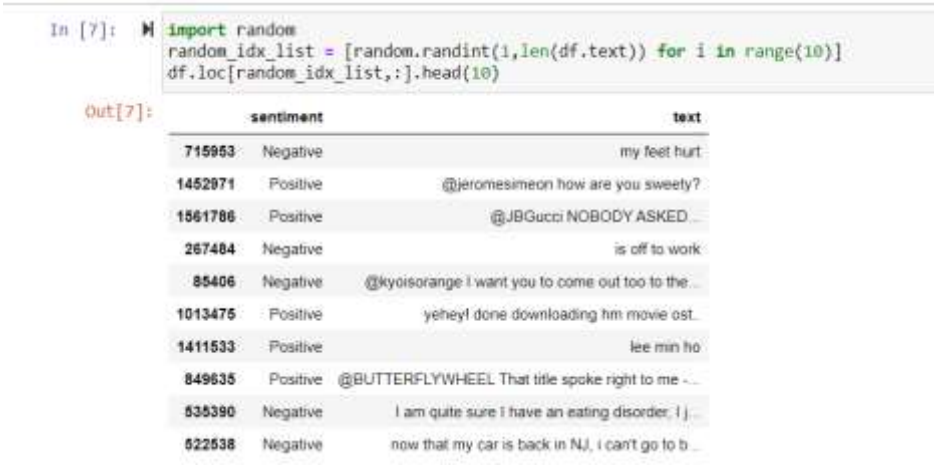


```
In [6]: val_count = df.sentiment.value_counts()

        plt.figure(figsize=(8,4))
        plt.bar(val_count.index, val_count.values)
        plt.title("Sentiment Data Distribution")
```

```
Out[6]: Text(0.5, 1.0, 'Sentiment Data Distribution')
```


Fig-4.4 Sentiment data distribution

- Following code creates a random indixes which are chosen from the data frame.
- And returns the rows with the index



```
In [7]: import random
        random_idx_list = [random.randint(1,len(df.text)) for i in range(10)]
        df.loc[random_idx_list,:].head(10)
```

Out[7]:

	sentiment	text
715953	Negative	my feet hurt
1452971	Positive	@jeromesimeon how are you sweety?
1561786	Positive	@JBGucci NOBODY ASKED...
267484	Negative	is off to work
85406	Negative	@kyoisorange I want you to come out too to the...
1013475	Positive	yeheyl done downloading hm movie ost..
1411533	Positive	lee min ho
849635	Positive	@BUTTERFLYWHEEL That title spoke right to me -...
535390	Negative	I am quite sure I have an eating disorder. I j...
522538	Negative	now that my car is back in NJ, i can't go to b...

Fig 4.5 display of random indexes

4.3 Text Pre-processing

Looks like we have a nasty data in text. Because in general we use lot of punctuations and other words without any contextual meaning. It has no value as feature to the model we are training. So, we need to get rid of them.

Tweet texts often consists of other user mentions, hyperlink texts, emoticons and punctuations. In order to use them for learning using a Language Model. We cannot permit those texts for training a model. So, we have to clean the text data using various pre-processing and cleansing methods.

stemming/Lemmatisation

For grammatical reasons, documents are going to use different forms of a word, such as write, writing and writes. Additionally, there are families of derivationally related words with similar meanings. The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.

Stemming usually refers to a process that chops off the ends of words in the hope of achieving goal correctly most of the time and often includes the removal of derivational affixes.

Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base and dictionary form of a word

Hyperlinks and Mentions

Twitter is a social media platform where people can tag and mentions other people's ID and share videos and blogs from internet. So the tweets often contain lots of Hyperlinks and twitter mentions.

Twitter is a social media platform where people can tag and mentions other people's ID and share videos and blogs from internet. So the tweets often contain lots of Hyperlinks and twitter mentions.

- Twitter User Mentions – example- @arunrk7, @andrewng
- Hyperlinks – example- https://keras.io

Stopwords

Stop-words are commonly used words in English which have no contextual meaning in a sentence. So there-fore we remove them before classification. Some stop-words are...

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

```
> stopwords("english")
  [1] "i"          "me"         "my"         "myself"     "we"
  [6] "our"        "ours"       "ourselves"  "you"        "your"
 [11] "yours"      "yourself"   "yourselves" "he"         "him"
 [16] "his"        "himself"    "she"        "her"        "hers"
 [21] "herself"    "it"         "its"        "itself"     "they"
 [26] "them"       "their"      "theirs"     "themselves" "what"
 [31] "which"      "who"        "whom"       "this"       "that"
 [36] "these"      "those"      "am"         "is"         "are"
 [41] "was"        "were"       "be"         "been"       "being"
 [46] "have"       "has"        "had"        "having"     "do"
 [51] "does"       "did"        "doing"      "would"      "should"
 [56] "could"      "ought"      "i'm"        "you're"     "he's"
 [61] "she's"      "it's"       "we're"      "they're"    "i've"
 [66] "you've"     "we've"      "they've"    "i'd"        "you'd"
 [71] "he'd"       "she'd"      "we'd"       "they'd"     "i'll"
 [76] "you'll"     "he'll"      "she'll"     "we'll"      "they'll"
 [81] "isn't"      "aren't"     "wasn't"     "weren't"    "hasn't"
 [86] "haven't"    "hadn't"     "doesn't"    "don't"      "didn't"
 [91] "won't"      "wouldn't"   "shan't"     "shouldn't"  "can't"
 [96] "cannot"     "couldn't"   "mustn't"    "let's"      "that's"
[101] "who's"      "what's"     "here's"     "there's"    "when's"
[106] "where's"    "why's"      "how's"      "a"          "an"
```

Fig 4.6 Stop-words

- There is a library of python which helps in text pre-processing, NLTK.

```
In [8]:
stop_words = stopwords.words('english')
stemmer = SnowballStemmer('english')

text_cleaning_re = "@\S+|https?:\S+|http?:\S|[^A-Za-z0-9]+"

In [9]:
def preprocess(text, stem=False):
  text = re.sub(text_cleaning_re, ' ', str(text).lower()).strip()
  tokens = []
  for token in text.split():
    if token not in stop_words:
      if stem:
        tokens.append(stemmer.stem(token))
      else:
        tokens.append(token)
  return " ".join(tokens)
```

Fig 4.7 Text pre-processing

- It is clean and tidy now. Now let's see some word cloud visualizations of it.

Positive and Negative words

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

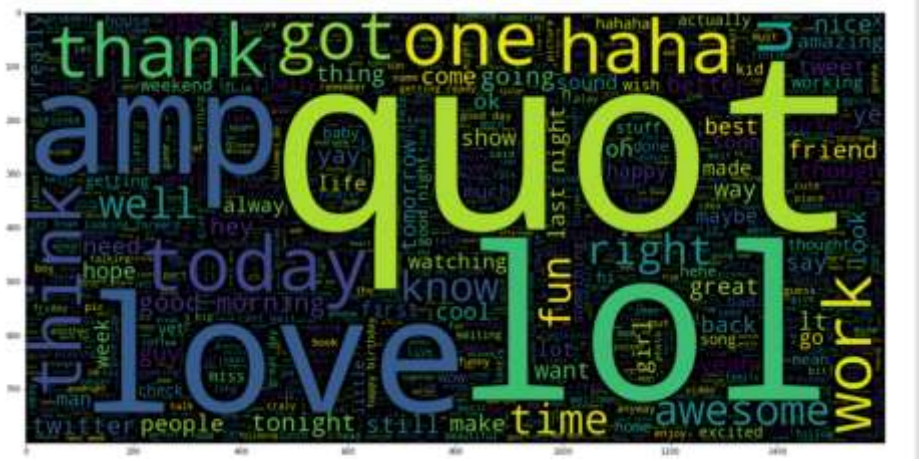


Fig 4.8 Positive words

```
In [12]: plt.figure(figsize = (20,20))
         wc = WordCloud(max_words = 2000 , width = 1600 , height = 800).generate(" ".join(df[df.sentiment == 'Negative'].text))
         plt.imshow(wc , interpolation = 'bilinear')
Out[12]: <matplotlib.image.AxesImage at 0x2ba0ef5c130>
```

Fig 4.9 Negative words

Train and Split Data

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

Fig 4.10 Training and testing data splitting

4.4 Tokenization

Given a character sequence and a defined document unit, tokenization is the task of chopping it up into pieces, called tokens, perhaps at the same time throwing away certain characters, such as punctuation. The process is called Tokenization.

- tokenizer create tokens for every word in the data corpus and map them to a index using dictionary.
- word_index contains the index for each word
- vocab_size represents the total number of words in the data corpus

```
In [16]: from keras.preprocessing.text import Tokenizer

         tokenizer = Tokenizer()
         tokenizer.fit_on_texts(train_data.text)

         word_index = tokenizer.word_index
         vocab_size = len(tokenizer.word_index) + 1
         print("Vocabulary Size :", vocab_size)

         Vocabulary Size : 290575
```

Fig 4.11 Tokenizing

- Now that we got a tokenizer object, which can be used to convert any word into a Key in dictionary (number).
- Since we are going to build a sequence model. We should feed in a sequence of numbers to it. And also, we should ensure there is no variance in input shapes of sequences. It all should be of same length. But texts in tweets have different count of words in it. To avoid this, we seek a little help from pad_sequence to do our job. It will make all the sequence in one constant length MAX_SEQUENCE_LENGTH.

```
In [17]: from keras.preprocessing.sequence import pad_sequences

         x_train = pad_sequences(tokenizer.texts_to_sequences(train_data.text),
                                 maxlen = MAX_SEQUENCE_LENGTH)
         x_test = pad_sequences(tokenizer.texts_to_sequences(test_data.text),
                                maxlen = MAX_SEQUENCE_LENGTH)

         print("Training X Shape:",x_train.shape)
         print("Testing X Shape:",x_test.shape)

         Training X Shape: (1280000, 30)
         Testing X Shape: (320000, 30)

In [18]: labels = train_data.sentiment.unique().tolist()
```

Fig 4.12 Making constant sequence length

- Label Encoding: We are building the model to predict class in encoded form (0 or 1 as this is a binary classification). We should encode our training labels to encodings.

```
In [19]: encoder = LabelEncoder()
         encoder.fit(train_data.sentiment.to_list())

         y_train = encoder.transform(train_data.sentiment.to_list())
         y_test = encoder.transform(test_data.sentiment.to_list())

         y_train = y_train.reshape(-1,1)
         y_test = y_test.reshape(-1,1)

         print("y_train shape:", y_train.shape)
         print("y_test shape:", y_test.shape)

         y_train shape: (1280000, 1)
         y_test shape: (320000, 1)
```

Fig 4.13 Label encoding

4.5 Word Embedding

In Language Model, words are represented in a way to intend more meaning and for learning the patterns and contextual meaning behind it.

Word Embedding is one of the popular representations of document vocabulary. It's capable of capturing context of a word in a document, semantic and syntactic similarity, relation with other words, etc.

Basically, it's a feature vector representation of words which are used for other natural language processing applications.

We could train the embedding ourselves but that would take a while to train and it wouldn't be effective. So going in the path of Computer Vision, here we use Transfer Learning. We download the pre-trained embedding and use it in our model.

The pretrained Word Embedding like GloVe & Word2Vec gives more insights for a word which can be used for classification. If you want to learn more about the Word Embedding, please refer some links that I left at the end of this notebook.

In this project, we have used GloVe Embedding from Stanford AI.

```
In [20]: GLOVE_EMB = "glove.6B.300d.txt"
         EMBEDDING_DIM = 300
         LR = 1e-3
         BATCH_SIZE = 1024
         EPOCHS = 10
         MODEL_PATH = "D:\best_model.hdf5"

In [21]: embeddings_index = {}

         f = open(GLOVE_EMB, encoding="utf8")
         for line in f:
           values = line.split()
           word = value = values[0]
           coefs = np.asarray(values[1:], dtype='float32')
           embeddings_index[word] = coefs
         f.close()

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

         Found 400000 word vectors.

In [22]: embedding_matrix = np.zeros((vocab_size, EMBEDDING_DIM))
         for word, i in word_index.items():
           embedding_vector = embeddings_index.get(word)
           if embedding_vector is not None:
             embedding_matrix[i] = embedding_vector

In [23]: embedding_layer = tf.keras.layers.Embedding(vocab_size,
                                                     EMBEDDING_DIM,
                                                     weights=[embedding_matrix],
                                                     input_length=MAX_SEQUENCE_LENGTH,
                                                     trainable=False)
```

Fig 4.14 word Embedding

4.6 Model Training - LSTM

We are clear to build our Deep Learning model. While developing a DL model, we should keep in mind of key things like Model Architecture, Hyper parameter Tuning and Performance of the model.

As you can see in the word cloud, some words are predominantly feature in both Positive and Negative tweets. This could be a problem if we are using a Machine Learning model like Naive Bayes, SVD, etc. That's why we use Sequence Models. Recurrent Neural Networks can handle a sequence of data and learn a pattern of input sequence to give either sequence or scalar value as output. In our case, the Neural Network outputs a scalar value prediction.

For model architecture, we use

1) Embedding Layer - Generates Embedding Vector for each input sequence.

2) Conv1D Layer – It’s using to convolve data into smaller feature vectors.

3) LSTM - Long Short-Term Memory, it’s a variant of RNN which has memory state cell to learn the context of words which are at further along the text to carry contextual meaning rather than just neighbouring words as in case of RNN.

4) Dense - Fully Connected Layers for classification

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

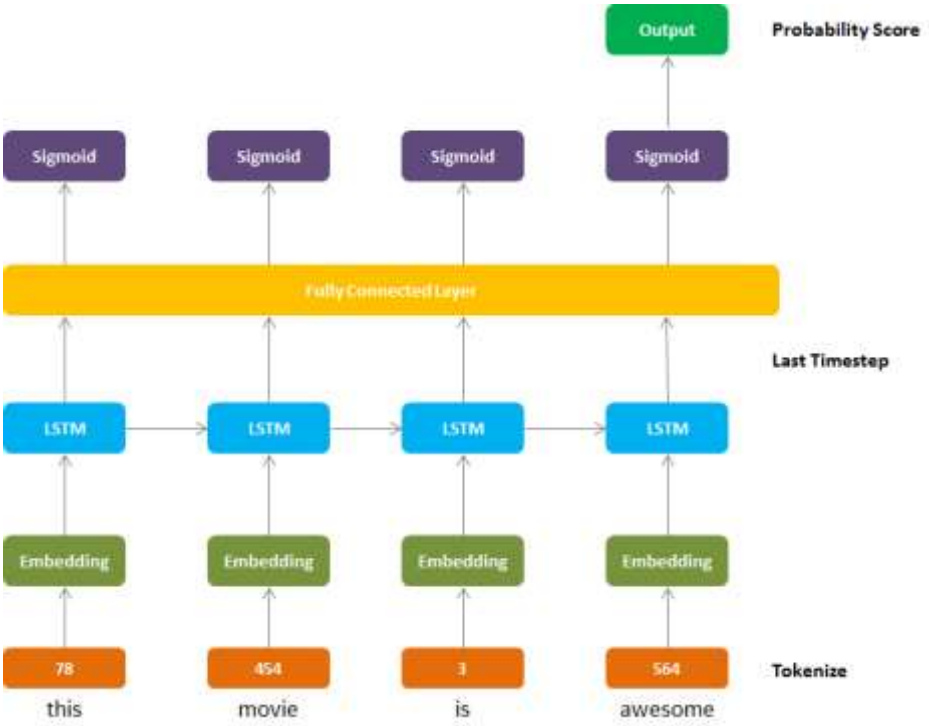


Fig 4.15 Sequence Model

```
In [24]: from tensorflow.keras.layers import Conv1D, Bidirectional, LSTM, Dense, Input, Dropout
         from tensorflow.keras.layers import SpatialDropout1D
         from tensorflow.keras.callbacks import ModelCheckpoint

In [25]: sequence_input = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32')
         embedding_sequences = embedding_layer(sequence_input)
         x = SpatialDropout1D(0.2)(embedding_sequences)
         x = Conv1D(64, 5, activation='relu')(x)
         x = Bidirectional(LSTM(64, dropout=0.2, recurrent_dropout=0))(x)
         x = Dense(512, activation='relu')(x)
         x = Dropout(0.5)(x)
         x = Dense(512, activation='relu')(x)
         outputs = Dense(1, activation='sigmoid')(x)
         model = tf.keras.Model(sequence_input, outputs)
```

Fig 4.16 Model training LSTM

4.7 Optimization Algorithm

We are using Adam optimization algorithm for Gradient Descent.

Call backs

Call backs are special functions which are called at the end of an epoch. We can use any functions to perform specific operation after each epoch. we used two call backs here,

LR Scheduler - It changes a Learning Rate at specific epoch to achieve more improved result. In this notebook, the learning rate exponentially decreases after remaining same for first 10 Epoch.

Model Check Point - It saves best model while training based on some metrics. Here, it saves the model with minimum Validity Loss.

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

```
In [26]: from tensorflow.keras.optimizers import Adam
         from tensorflow.keras.callbacks import ReduceLROnPlateau

         model.compile(optimizer=Adam(learning_rate=LR), loss='binary_crossentropy',
                       metrics=['accuracy'])
         ReduceLROnPlateau = ReduceLROnPlateau(factor=0.1,
                                               min_lr = 0.01,
                                               monitor = 'val_loss',
                                               verbose = 1)

In [27]: print("Training on GPU...") if tf.config.list_physical_devices('GPU') else print("Training on CPU...")

         Training on GPU...
```

Fig 4.17 Adam optimization algorithm

```
In [28]: history = model.fit(x_train, y_train, batch_size=BATCH_SIZE, epochs=EPOCHS,
                            validation_data=(x_test, y_test), callbacks=[ReduceLROnPlateau])

         Epoch 1/10
         1250/1250 [==============================] - 295s 236ms/step - loss: 0.5196 - accuracy: 0.7392 - val_loss: 0.4824 - val_accu
         racy: 0.7661
         Epoch 2/10
         1250/1250 [==============================] - 294s 235ms/step - loss: 0.4890 - accuracy: 0.7632 - val_loss: 0.4721 - val_accu
         racy: 0.7725
         Epoch 3/10
         1250/1250 [==============================] - 305s 244ms/step - loss: 0.4795 - accuracy: 0.7678 - val_loss: 0.4670 - val_accu
         racy: 0.7759
         Epoch 4/10
         1250/1250 [==============================] - 292s 234ms/step - loss: 0.4754 - accuracy: 0.7723 - val_loss: 0.4659 - val_accu
         racy: 0.7774
         Epoch 5/10
         1250/1250 [==============================] - 291s 233ms/step - loss: 0.4697 - accuracy: 0.7736 - val_loss: 0.4648 - val_accu
         racy: 0.7788
         Epoch 6/10
         1250/1250 [==============================] - 290s 232ms/step - loss: 0.4666 - accuracy: 0.7761 - val_loss: 0.4601 - val_accu
         racy: 0.7802
         Epoch 7/10
         1250/1250 [==============================] - 290s 232ms/step - loss: 0.4634 - accuracy: 0.7777 - val_loss: 0.4594 - val_accu
         racy: 0.7802
         Epoch 8/10
         1250/1250 [==============================] - 291s 232ms/step - loss: 0.4617 - accuracy: 0.7784 - val_loss: 0.4569 - val_accu
         racy: 0.7833
         Epoch 9/10
         1250/1250 [==============================] - 290s 232ms/step - loss: 0.4596 - accuracy: 0.7805 - val_loss: 0.4561 - val_accu
         racy: 0.7829
         Epoch 10/10
         1250/1250 [==============================] - 291s 233ms/step - loss: 0.4580 - accuracy: 0.7808 - val_loss: 0.4550 - val_accu
         racy: 0.7831
```

Fig 4.18 Training in GPU

5. Results

5.1 Model Evaluation

Now that we have trained the model, we can evaluate its performance. We will some evaluation metrics and techniques to test the model.

Let's start with the Learning Curve of loss and accuracy of the model on each epoch.

```
<matplotlib.legend.Legend at 0x7f3739c12490>
```

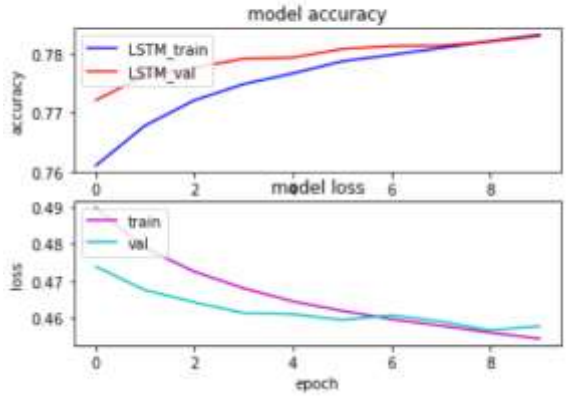


Fig 5.1 Model evaluation

• The model will output a prediction score between 0 and 1. We can classify two classes by defining a threshold value for it. In our case, I have set 0.5 as THRESHOLD value, if the score above it. Then it will be classified as POSITIVE sentiment.

```
In [33]: def decode_sentiment(score):
             return "Positive" if score>0.5 else "Negative"


         scores = model.predict(x_test, verbose=1, batch_size=5000)
         y_pred_1d = [decode_sentiment(score) for score in scores]

         64/64 [==============================] - 12s 182ms/step
```

5.2 Confusion Matrix

Confusion Matrix provides a nice overlook at the model's performance in classification task.

```
In [30]: import itertools
         from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
         def plot_confusion_matrix(cm, classes,
                                   title='Confusion matrix',
                                   cmap=plt.cm.Blues):
             """
             This function prints and plots the confusion matrix.
             Normalization can be applied by setting `normalize=True`.
             """

             cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

             plt.imshow(cm, interpolation='nearest', cmap=cmap)
             plt.title(title, fontsize=20)
             plt.colorbar()
             tick_marks = np.arange(len(classes))
             plt.xticks(tick_marks, classes, fontsize=13)
             plt.yticks(tick_marks, classes, fontsize=13)

             fmt = '.2f'
             thresh = cm.max() / 2.
             for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
                 plt.text(j, i, format(cm[i, j], fmt),
                          horizontalalignment="center",
                          color="white" if cm[i, j] > thresh else "black")

             plt.ylabel('True label', fontsize=17)
             plt.xlabel('Predicted label', fontsize=17)
```

```
In [35]: cnf_matrix = confusion_matrix(test_data.sentiment.to_list(), y_pred_1d)
         plt.figure(figsize=(6,6))
         plot_confusion_matrix(cnf_matrix, classes=test_data.sentiment.unique(), title="Confusion matrix")
         plt.show()
```

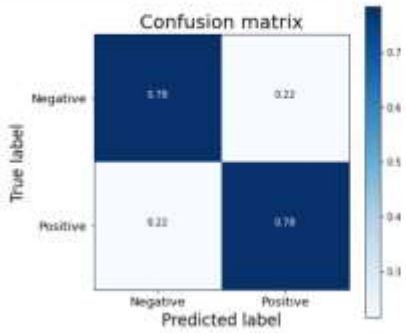


Fig 5.2 Confusion matrix

5.3 Classification Scores

```
In [36]:
print(classification_report(list(test_data.sentiment), y_pred_1d))
```

	precision	recall	f1-score	support
Negative	0.78	0.78	0.78	160542
Positive	0.78	0.78	0.78	159458
accuracy			0.78	320000
macro avg	0.78	0.78	0.78	320000
weighted avg	0.78	0.78	0.78	320000

6. CONCLUSION

We have successfully implemented and tested deep learning model-LSTM. The system has the capabilities to classify given text which results in high accuracy. We are able to achieve satisfactory results based on our training data.

We have collected 1600000 tweets extracted using the twitter API. We have applied sentiment analysis to the given data after pre-processing of it. Later, the text which is obtained is pre-processed. Then the cleansed data is split into training and testing data. Then deep learning model LSTM is applied to achieve high accuracy around 80%.

7. FUTURE SCOPE

Our future work includes exploring other classification algorithms on a much more diverse dataset with different deep Learning techniques.

Semantics Our algorithms classify the overall sentiment of a tweet. The polarity of a tweet may depend on the perspective you are interpreting the tweet from. For example, in the tweet Federer beats Nadal :), the sentiment is positive for Federer and negative for Nadal. In this case, semantics may help. Using a semantic role labeller may indicate which noun is mainly associated with the verb and the classification would take place accordingly. This may allow Nadal beats Federer :) to be classified differently from Federer beats Nadal :)

Handling neutral tweets In real world applications, neutral tweets cannot be ignored. Proper attention needs to be paid to neutral sentiment.

Internationalization We focus only on English sentences, but Twitter has many international users. It should be possible to use our approach to classify sentiment in other languages.

8. References

1. [1] Bao Y. and Ishii N., “Combining Multiple kNN Classifiers for Text Categorization by Reducts”, LNCS 2534, 2002, pp. 340-347.
2. [2] Bi Y., Bell D., Wang H., Guo G., Greer K., Combining Multiple Classifiers Using Dempster's Rule of Combination for Text Categorization”, MDAI, 2004, 127-138.
3. [3] Brank J., Grobelnik M., Milic-Frayling N., Mladenic D., “Interaction of Feature Selection Methods and Linear Classification Models”, Proc. of the 19th International Conference on Machine Learning, Australia, 2002.
4. [4] Chawla, N. V., Bowyer, K. W., Hall, L. O., Kegelmeyer, W. P., “SMOTE: Synthetic Minority Over-sampling Technique,” Journal of AI Research, 16 2002, pp. 321-357.

5. [5] Forman, G., An Experimental Study of Feature Selection Metrics for Text Categorization. Journal of Machine Learning Research, 3 2003, pp. 1289-1305.
6. [6] Fragoudis D., Meretakis D., Likothanassis S., "Integrating Feature and Instance Selection for Text Classification", SIGKDD '02, July 23-26, 2002, Edmonton, Alberta, Canada.
7. [7] Guan J., Zhou S., "Pruning Training Corpus to Speedup Text Classification", DEXA 2002, pp. 831-840.
8. [8] D. E. Johnson, F. J. Oles, T. Zhang, T. Goetz, "A decision-tree-based symbolic rule induction system for text categorization", IBM Systems Journal, September 2002.
9. [9] Han X., Zu G., Ohyama W., Wakabayashi T., Kimura F., Accuracy Improvement of Automatic Text Classification Based on Feature Transformation and Multi-classifier Combination, LNCS, Volume 3309, Jan 2004, pp. 463-468.
10. [10] Ke H., Shaoping M., "Text categorization based on Concept indexing and principal component analysis", Proc. TENCON 2002 Conference on Computers, Communications, Control and Power Engineering, 2002, pp. 51-56.