ANGEL MACWAN
MSC DATA SCIENCE

Sentiment Analysis

ABSTRACT

Twitter allows people to express public opinion and this might be a concern to companies and the government.
As there is so much data on Twitter that it becomes very hard for brands to prioritize which tweets to respond to. To solve this problem, sentiment analysis is used as a tool to monitor the emotion of these tweets.
Sentiment Analysis is the automated process of identifying and classifying subjective information in text data. The project addressed here studies the Twitter feed of various users and classifies their tweets into one of three sentiments (Positive, Neutral, Negative).
This task is done using a Simple Neural Network resulting in 80% accuracy and efficient deployment.

PROJECT AVAILABLE AT
https://github.com/angelmacwan/Sentiment_Analysis

CONTENTS

EXPLORATORY DATA ANALYSIS

The Data Set

The dataset used in this project is the "sentiment140" dataset from Kaggle. It contains 1,600,000 tweets extracted using the Twitter API. https://www.kaggle.com/kazanova/sentiment140

The data contains the following 6 fields:

1. target: the polarity of the tweet (0 = negative, 4 = positive)
2. ids: The id of the tweet (2087)
3. date: the date of the tweet (Sat May 16 23:58:44 UTC 2009)
4. flag: The query (lyx). If there is no query, then this value is NO_QUERY.
5. user: the user that tweeted (robotickilldozr)
6. text: the text of the tweet (Lyx is cool)

Taking a look at the data

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1600000 entries, 0 to 1599999
Data columns (total 6 columns):
 #   Column  Non-Null Count    Dtype
---  ------  --------------    -----
 0   0       1600000 non-null  int64
 1   1       1600000 non-null  int64
 2   2       1600000 non-null  object
 3   3       1600000 non-null  object
 4   4       1600000 non-null  object
 5   5       1600000 non-null  object
dtypes: int64(2), object(4)
```

As seen above, there are **1600000 records** and their data types are 64bit Integer and Object.

For training and implementing the model we only need the tweet text and its label. for this reason, other columns can be dropped.

There are no Null Values present in the data so imputation is not necessary.

The data contains two (2) labels, Positive and Negative in the dataset, Positive is labeled with the number 4, and Negative is labeled with the number 0.

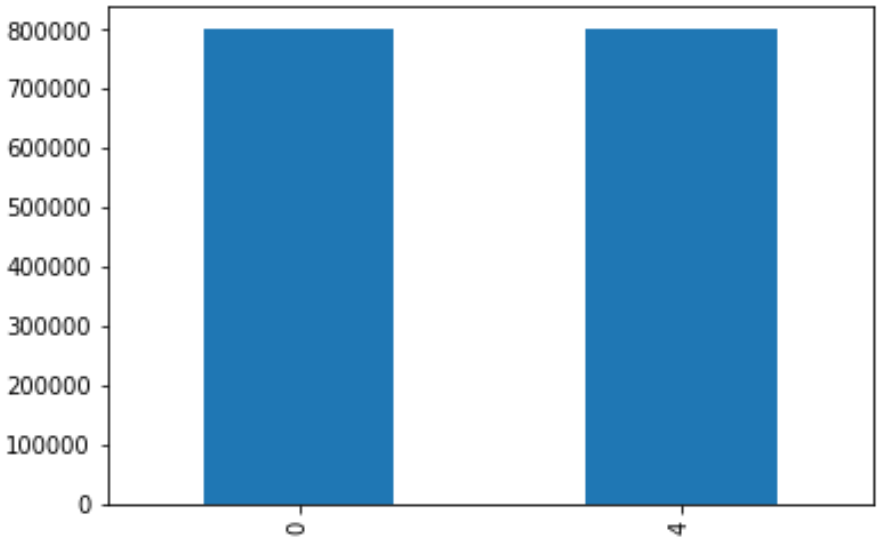

Figure 1.1

Figure 1.1 shows the distribution of both labels in the dataset. 0 (Zero) represents Negative sentiment and 4 (four) represents Positive Sentiment

Data Cleaning

In order to use the data for training the model, it has to be cleaned and structured.

First all stop words such as "is", "and", "the", "then", etc are removed from all tweets.

Then other artifacts such as mentions (@UserName) and HashTags (#HashTag) are removed by using regex.

Word Cloud

A word cloud is a novelty visual representation of text data, typically used to depict keyword metadata on websites or to visualize free-form text.
Tags are usually single words, and the importance of each tag is shown with font size or color.
Here font size is used to hilight impact of that word.

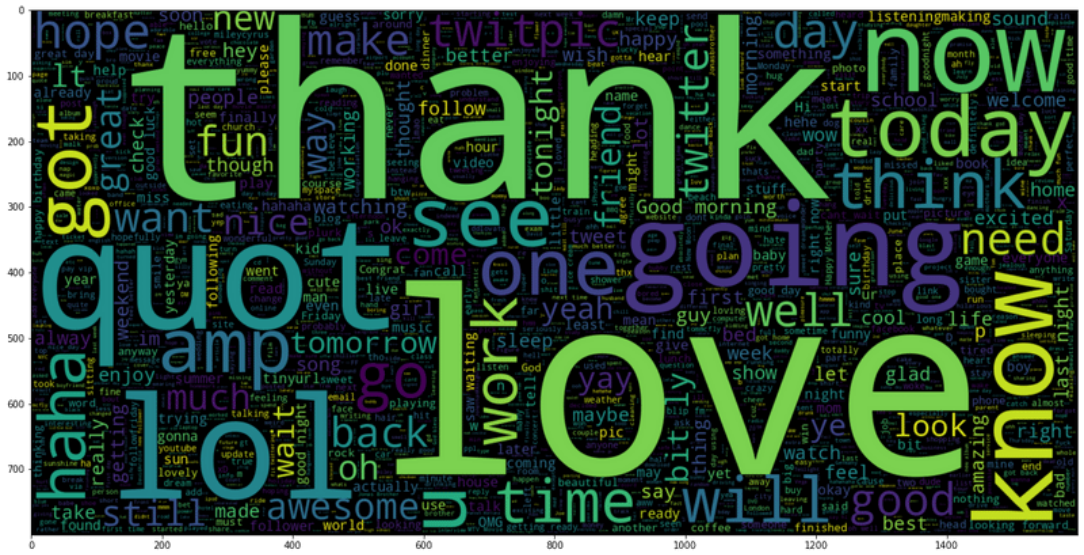

Figure 2.1

Figure 2.1 shows the word cloud of Positive Tweets.

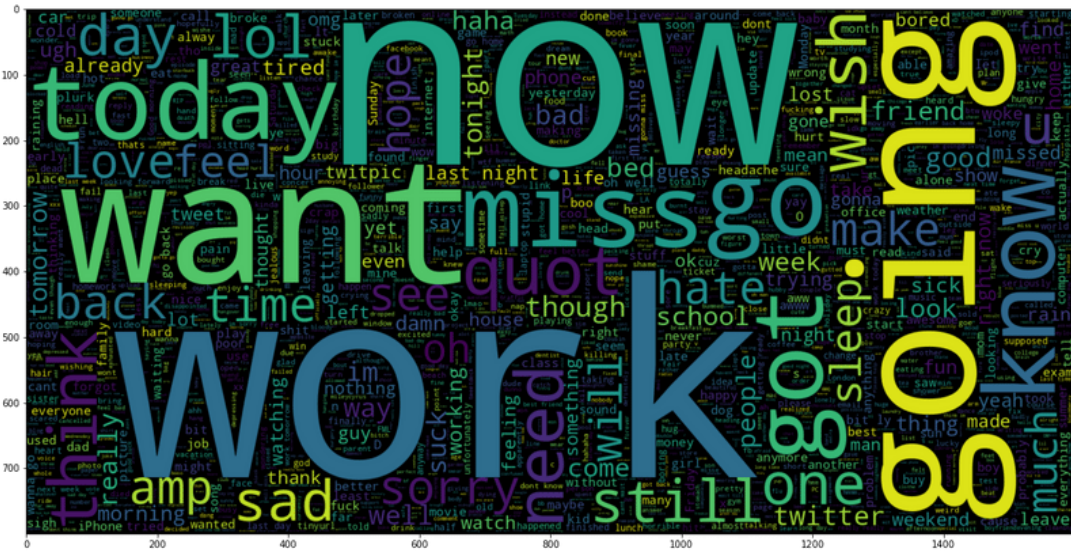

Figure 2.2

Figure 2.2 shows the word cloud of NegativeTweets.

THE MACHINE LEARNING MODEL

About the Model

For creating and training the ML Model, TensorFlow 2.5.0 is used which is an abstraction layer on top of Keras API.

TensorFlow is used for both the Tokenizer Model and NeutalNet.

Tokenizing, Sequencing, and Padding

Tokenizing is the mapping of words to numbers.
Tokenizer() in TensorFlow generates a dictionary of keys as words and its values are unique numbers.
tokenizer.word_index returns the generated dictionary

Sequencing is the conversion of a text string into a corresponding number sequence.
Tokenizer() object can be used to convert strings to numbers once it is fit the corpus of text. tokenizer.texts_to_sequences() method will map words to its corresponding int value from word_index.

Padding is used to convert all tokenized strings to the same length so that ML Models can use them as input

Parameters for the Tokenizer Model

```
# number of maximum elements in the tokenizer model
vocab_size = 30000

# maximum length if a sentence
max_length = 200

# sentences longer than max_length will be
# truncated from its end
trunc_type='post'

# 0 (zero) will be added at the end of sentences shorter than
# max_length to match the dimensions of other sentences
padding_type='post'

# if a word from a sentence is not found in the tokenizer
# model, it will be replaced with "<OOV>"
# and its index will be 0
oov_tok = "<OOV>"
```

These parameters help to standardize the data and use it as input to an ML Model.
This also ensures that the dimension of each input to the model remains constant.

Once the Tokenizer model is trained it is exported to a JSON file to be used in various applications without retraining it again.

The NeutalNet

```
Model: "sequential"
Layer (type)                 Output Shape              Param

embedding (Embedding)        (None, 200, 16)           480000
global_average_pooling1d (Gl (None, 16)                0
dense (Dense)                (None, 24)                408
dense_1 (Dense)              (None, 1)                 25
```

To create a NeuralNet, Keras Sequential API is used. This ensures that data will pass from the top layer to the bottom layers.

LAYER 01 (Embedding)

The 1st layer is the Embedding layer that receives input of length 200 (max_length) and returns a vector of length 16.

The Embedding layer is used to map discrete or categorical values to a continuous vector of reduced dimension. In the context of neural networks, embeddings are low-dimensional.

Neural network embeddings are useful because they can reduce the dimensionality of categorical variables and meaningfully represent categories in the transformed space.

LAYER 02 (Global Average Pooling)

The 2nd layer is a GlobalAveragePooling1D layer.

A GlobalAveragePooling1D layer returns a fixed-length output vector for each input by averaging over the sequence dimension. This allows the model to handle input of variable length, in the simplest way possible.

If n inputs are passed, this layer makes sure that all dimensions are the same

LAYER 03 (Dense)
The 3rd layer is a Dence Layer that returns an output of length 24 and has an activation function of ReLU (Rectified Linear).

The dense layer is a neural network layer that is connected deeply, which means each neuron in the dense layer receives input from all neurons of its previous layer.

The activation parameter is helpful in applying the element-wise activation function in a dense layer.

Here the ReLU activation function is used. This functions returns a boolean value ie 0 (false) or 1 (true).

If a ReLU for a neuron returns 1, that neuron is considered active. The ReLU function returns 0 or 1 based on a threshold that is calculated internally. If the output is higher than the threshold then 1 is returned else 0 is returned.

This layer will return the output of 24 such neurons.

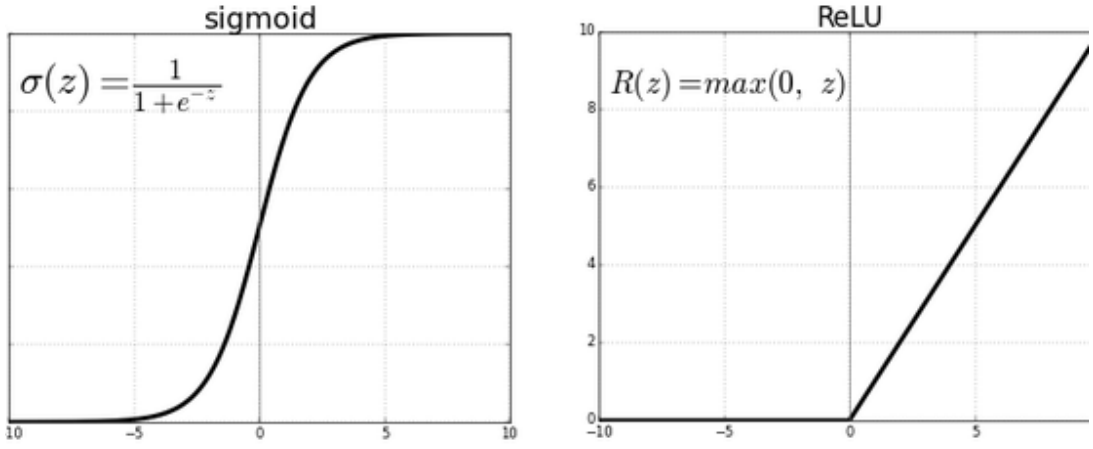

Figure 3.1

As seen in figure 3.1, the ReLU function returns either 0 or 1 depending on the input value and threshold. While a Sigmoid function returns a probability score between 0 and 1.

LAYER 04 (Dense)
The 4th layer is a Dence Layer that returns a single output and has an activation function of Sigmoid.

This layer receives input from the previous dense layer and returns a probability score.

The previous layer used ReLU which returns a boolean value. This can be used for classification (0 is negative, 1 is positive). But this does not give an idea about how sure the model is about this prediction.

For this reason, a sigmoid function is used. This function returns a value between 0 and 1 instead of a boolean value.

For example, ReLU can output 1 for a sentence indicating that the sentence is positive, while for the same sentence sigmoid might return 0.6 indicating the sentence is only slightly positive or 0.8 indicating the sentence is very positive.

Compiling the Neural Net

For compiling the model, the following parameters are used

```
loss = 'binary_crossentropy'
optimizer='adam'
metrics=['accuracy']
```

The **loss function** is responsible for calculating the loss or error rate of an algorithm. Here Binary Crossentropy method is used to calculate the loss for the model.

Binary cross entropy compares each of the predicted probabilities to actual class output which can be either 0 or 1. It then calculates the score that penalizes the probabilities based on the distance from the expected value. That means how close or far from the actual value.

Binary Cross Entropy is the negative average of the log of corrected predicted probabilities.

The **optimizer** is an algorithm that is used to change attributes of the neural network such as weights, biases, and learning rate in order to reduce loss or error rate.

Adaptive Moment Estimation or **ADAM** is an algorithm for optimization technique for gradient descent. The method is really efficient when working with a large problem involving a lot of data or parameters. It requires less memory and is efficient. Intuitively, it is a combination of the 'gradient descent with momentum' algorithm

The goal of the optimizer is to reach the global minimum of a given function.

The metrics is set to Accuracy. ie the loss will be calculated with accuracy as input.

The Model is created with the above-mentioned attributes and trained for 30 epochs.

The number of epochs defines the number of times data is passed through the model. Each time data is randomized, this helps the model learn to train more effectively. With each epoch, loss decreases, and the accuracy of the model increases.

Model Evaluation

Once the model is trained on data, it has to be tested and eveluated

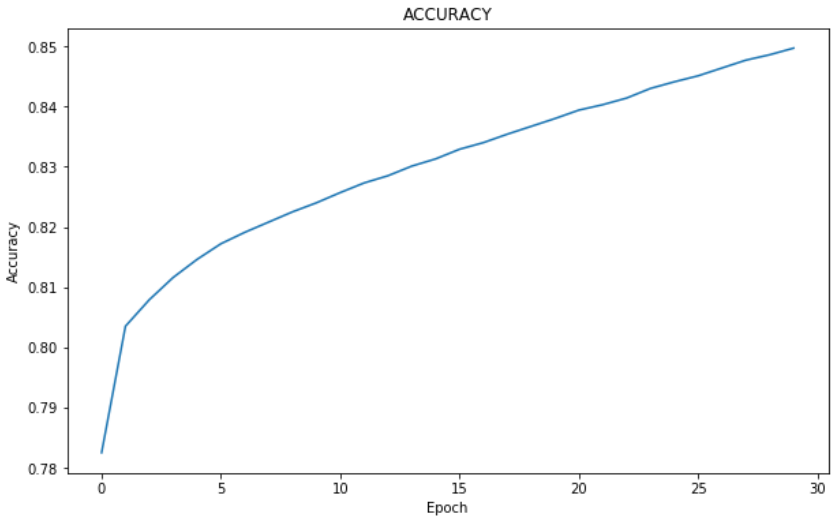

Figure 4.1

Figure 4.1 shows Accuracy of the model rising for each Epoch

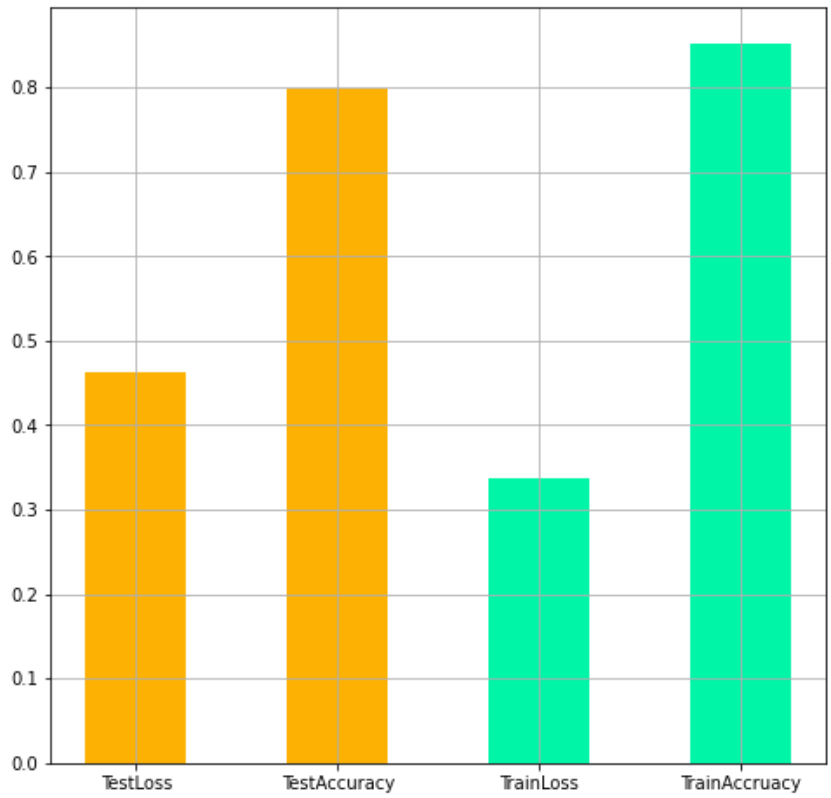

Figure 4.2

Figure 42 shows Accuracy and Loss during both training and testing phase.

test accuracy 79.83 %

test loss 46.27 %

train accuracy 85.24 %

train loss 33.66 %

Confusion Matrix & Classification Report

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known

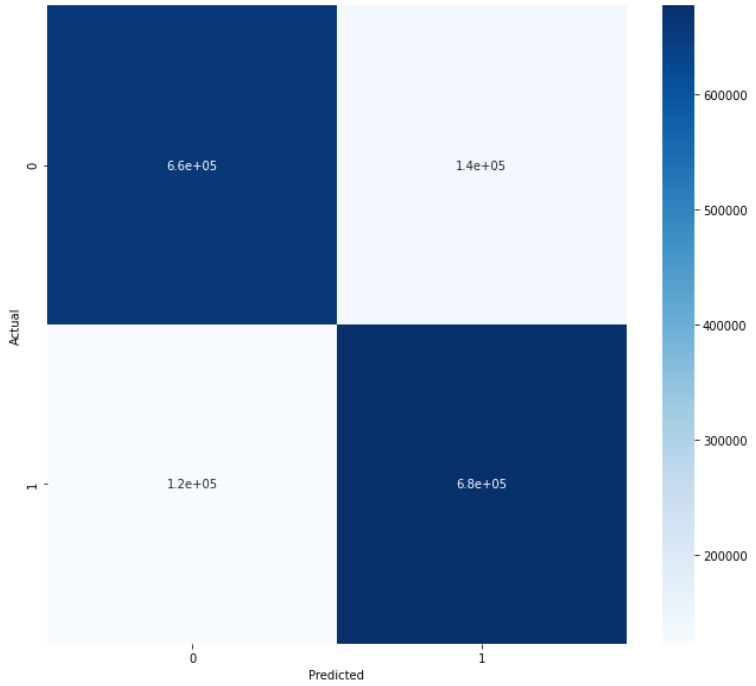

Figure 4.3

Figure 4.3 shows the confusion matrix of the ML model

```
Values of the matrix are
shown bellow
array([[658648, 141352],
       [123234, 676766]])
```

By using the confusion matrix, a classification report can be generated. This report includes precisson, recall, f1-score and support for each class.

	precision	recall	f1-score	support
NEGATIVE	0.84	0.82	0.83	800000
POSITIVE	0.83	0.85	0.84	800000
accuracy			0.83	1600000
macro avg	0.83	0.83	0.83	1600000
weighted avg	0.83	0.83	0.83	1600000

The final model has an accuracy score of about 80% which is a good fit for our purposes.
However, this can be improved by using other types of neural networks such as CNN or LSTM.

TWITTER API

The Twitter API is a set of programmatic endpoints that can be used to learn from and engage with the conversation on Twitter. This API allows you to find and retrieve, engage with, or create a variety of different resources.

The Twitter API Documentation is available at https://developer.twitter.com/en/docs/getting-started

In order to use the Twitter API, one must have a Twitter account and register for a Twitter developer API after which you will be given a BEARER TOKEN that acts as a password through which you can interact with the API using scripts.

This project requires Tweets of various users that can be fed into the Model and be classified into its sentiment.

I have created a class that allows the program to fetch tweets for a used. Code available at https://github.com/angelmacwan/Sentiment_Analysis/blob/main/TwitterConnectionAPI.ipynb

The class contains methods such as getUserInfo() and getTweets().

The getUserInfo() method takes in one argument that is the Twitter Handle of a user.

The getTweets() method takes in one argument that is the user ID which can be obtained from getUserInfo() method.

BACKEND

The backend of the application is made using **FLASK**.

Flask us a python Micro Framework for creating web applications

In the backend, the TwitterAPI Class is implemented to get tweets from the user and the Tokenizer Model and NeuralNet Model are loaded into memory for classification.

Endpoints for the frontend are also implemented as mentioned below:

/gettweets/<name>
this endpoint fetches the tweets of user <name> and classifies them into their sentiment. This is converted to a JSON object and sent to the frontend where this data is visualized.

/textanalysis
this endpoint receives text from the user by POST method and splits the text into sentences. These sentences are used to perform sentiment classification and a JSON object of the result is sent back to the frontend.

RUN THE APPLICATION

In order to run this application, a few python3 libraries must be installed:

- TensorFlow 2.x
- nltk
- json
- requests
- flask

Once all this is installed, download the project from github using this link https://github.com/angelmacwan/Sentiment_Analysis

Inside the folder WebApp Flask, run main.py file using
python main.py
or
python3 main.py

This will start a Flask server and expose the application to port 5000 and the application will be accessible on any installed browser at **http://127.0.0.1:5000/**

REFERENCES

Other such datasets

https://www.kaggle.com/columbine/imdb-dataset-sentiment-analysis-in-csv-format?select=Train.csv

Pre-trained model (Transformers model)

https://pypi.org/project/transformers/

Transformers provides thousands of pre-trained models to perform tasks on texts such as classification, information extraction, question answering, summarization, translation, text generation, and more in over 100 languages. Its aim is to make cutting-edge NLP easier to use for everyone.

Reference study

https://www.kaggle.com/isanbel/depression-on-twitter

https://www.kaggle.com/mayank954/twitter-sentiment-analysis-naive-bayes

Here Machine learning methods such as RandomForestClassifier, XGBClassifier and SVM are used.

More information on Sentiment Analysis

https://callminer.com/blog/sentiment-analysis-examples-best-practices