Mood detection in Tweets

Thomas Emment

N0789683

Supervised by: Dr. Archie Giannakidis

A dissertation submitted in partial fulfilment of the requirements for BSc (Hons) Data Science at Nottingham Trent University.

April 2021

Abstract

The problem was to label text with the emotion it is showing. The mathematical approach used was machine learning and the main research was done using the books "Hands of machine learning" and "Deep learning with python"[1][3].

A variety of machine learning techniques were researched and used. From support vector machines (SVM) and neural networks to 1D convolution and long short term memory (LSTM) machines.

A multitude of emotions where investigated, including Happy Vs Sad, Angry Vs None and Sarcastic Vs None

SVM showed good recognition of certain words that belonged to different emotions categories allowing them to get high accuracy on my easily classifiable data. They struggled with words that influenced the following word as no sequencing was considered. They also struggled when none of the key obvious words (happy, good, sad, bad) were not present.

Neural Networks showed no real improvements over SVMs and often performed rather poorly in comparison. 1D Convolution and LSTM both showed promise at recognising the effect of previous words but where not as confident at classifying obvious statements like SVM. In the end they both had

slightly less accuracies than the SVM but generalized better to less obvious data.

Happy Vs Sad went well, and good accuracies were found as well as a high degree of generalization when faced with new data. Angry Vs None also performed well, however, not as well as happy vs sad and struggled being confident on obvious data (such as data that included the word angry). Sarcasm while seeing some good accuracies did not generalize well to any new data.

The best models where successfully deployed using a raspberry pi to monitor messages to a twitter account and reply to the messages with the emotion showed. Full deployment of this is shown however it will not be running at the time of reading.

Note: Not all models could be shown/talked about in this document. Extra training examples and demos can be found in appendix B and a dropbox folder where the vast majority of models are stored (see [5](can be found in references .

Contents

Chapter 1

Introduction

1.1 Overview

The problem:

These days everyone is online on their phones/computers and its social media where people share their opinions. Twitter is one of these social media platforms that takes a large portion of the market [6]. Thousands of tweets are sent every second and the information these tweets bring is invaluable, from reactions to products, advertisements and campaigns to knowing in the moment what a potential customer is thinking or feeling. However, sorting through all these tweets would be impossible as well as very costly to employ a person to take note of these trends. The solution is a computer running a model to label the emotion of the text. The computer can look through thousands of tweets a minute with minimal cost and hopefully a good deal of accuracy.

Applications:

- Chat bots can seem more intelligent by commenting on or reacting to how the user is feeling. Additionally, customer service bots could detect if a user is angry to pass them onto a real person.
- Targeted advertisement: Certain products sell better when a person is happy or sad, knowing when a user is one of these would help advertisers decide when to advertise certain products.
- Assisting autistic people in their day to day conversations with people. Helping them communicate by informing them how the other person is feeling as this is near impossible for people with autism to detect.
- If a brand wanted to gauge the reaction to a new product or promotion, they could download related tweets through the twitter API and apply a detection bot, such as the one this project will develop, to all the tweets to get statistics on the general reaction from the public.

Current solutions:

As mentioned previously, humans can be employed to manually analyse data but this is incredibly slow and costly. The only upside this brings is that humans will inherently have a high degree of accuracy when analysing the emotion of a text although still not near perfect accuracy [7]. To justify the use of the bot over these, the models will have to obtain a reasonable degree of accuracy.

Microsoft Azure, Amazon AWS and other companies offer inbuilt sentiment analysis bots as well as many other similar services [8]. While these are excellent products, they only focus on positive and negative rather than more broad emotions which this project aims to investigate.

1.2 My Solution

My solution:

This project aims to create a model that can classify the emotion in a tweet/small piece of text. This will be achieved by using machine learning techniques with the program that will identify and learn which words or sequence of words correspond to certain emotion. Data will be gathered using the twitter API and a variety of online existing databases. Data will then be processed for understanding by the computer and used to train the models to identify the patterns and words. Extra care will be exercised to make sure the model generalizes to everyday tweets.

Areas of investigation:

This project will investigate 3 main areas of emotion:

- Happy vs Sad and Happy vs Sad vs None

Happy and sad will be a focus of the investigation. These emotions are opposite so should be easier to distinguish between and are very telling emotions about what a person might be interested in or reacting to something. I will start with happy vs sad to get an understanding of how the models are working and then introduce the none class for better distinguishing between emotions. This will change the problem from binary classification task to multiclass classification task.

- Angry vs None

Anger is quite a strong emotion and very easy to detect in-person but often this is down to visual clues. It is anticipated that the vast amount of trigger words for anger will aid the model and I believe this to be a very useful as well as underrepresented area of investigation.

- Sarcastic vs None

This is an extension to the project. Even humans find it hard to detect sarcasm and it will be incredibly difficult for a bot to pick up on, especially with no visual clues. This has great applications in making chat bots more life like and pushes the model to human level intelligence.

Final Product:

To demonstrate my work, I will create a twitter bot that when tweeted will reply with the emotion it thinks is in the main tweet. This will be done using the twitter API and the models that will be built during the project. A stream will be set up on a raspberry pi or laptop to look for incoming tweets, analyse them, and tweet them back.

1.3 Machine Learning

Machine Learning:

Machine learning is the understanding that computers can learn without being explicitly programmed. Models work by learning off historic data looking for common patterns or values that lead to similar outputs. The models are trained by being scrutinised by a performance measure and the parameters of the model are adjusted according to the performance[2].

Supervised vs Unsupervised:

There are 2 main types of learning when it comes to machine learning: supervised and unsupervised learning. The idea behind these is that in supervised we teach a computer to do a task, such as predicting the price of a house in California based of previous data. In unsupervised we let the computer teach itself, for example teaching a bot to play chess by making it play itself millions of times and change its strategy based of what works well. In this project I am dealing with labeled tweets therefore we have a supervised learning task.

Classification vs regression:

A machine learning model can perform two types of task: classification or regression. Regression is where data is used to predict a value from a continuous range, for example the price of a house. Classification is where data is used to choose between discrete classes for example happy and sad. While we could use regression for predicting probability of single emotions this would require the labelling of each tweet with probabilities and so I will stick with classification for this task.

Chapter 2

Design

2.1 Methods

I will be using the following machine learning techniques to analyse my data and come up with the best model for predicting the emotion of future text.

Some of the following terms apply to all machine learning techniques:

- Hyper Parameter – These are variables of the function/inputs into the function that can be changed before training. These allow the tuning and adjusting of the model to better perform on our data and task.
- Vectorised – Data must be transformed into a format that machines can easily process. Vectorised data is text data transformed into a number list format.
- Training – Refers to feeding the model or some function pre labelled

data for it to analyse.

- Regularisation – This is the adding of additional variables or adjustments to a function to address overfitting.

2.1.1 Support Vector Machine

Support vector machines work by looking at all the data at once and creating a separating hyperplane between 2 or more classes also known as a decision boundary. The decision boundary is chosen by maximising the distance between the hyperplane and the closest point of each class, this is shown in the figure 2.1. The distances are calculated using the dot product and points that fall on one side of the plane belong to one class e.g., happy and on the other side a second class e.g., sad. The decision boundary also has support vectors. Support vectors correspond to the closest point on each side of the decision boundary but do allow for misclassification within the margin. The margin is between the two support vectors and allows points to fall on the wrong side of the decision boundary with this margin[13]. This is shown in figure 2.6 and is explained in the soft vs hard margin section.

While the example in figure 2.1 shows a 2D representation of a decision boundary (two variables), the power of a SVM is that large amounts of variables can be take into consideration and the basic formula does not change nor does training time increase by much. This is because the SVM problem is convex and easily solvable, allowing us to construct boundaries in dimensions not perceivable by humans through hyper planes.

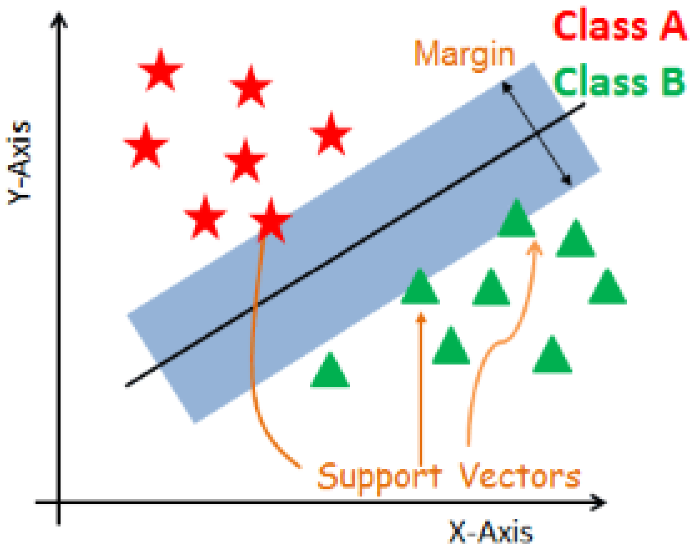


Figure 2.1: Shows classes split by a decision boundary

The calculation solved by the model is shown in figure 2.2.

Features:

Kernel – The kernel lets us transform data into different spaces allowing for more complex patterns and separations to be picked up on by the support vector machine. Data may not be linearly separable in its current state but when transformed linear separation may be possible. The main kernels I will be exploring, along with the transformation they make on the data are shown in figure 2.3.

Visual representations of the patterns these kernels are good at picking up on, once transformed back into the original space, are shown in figure 2.4.

An example of how a complex separation that can be found from transforma-

$$\min_{\omega,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i$$

$$y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i$$

where ζ_i denotes the distance to the correct margin with $\zeta_i \geq 0,\ i = 1, \ldots, n$

where C denotes a regularization parameter

were $\mathrm{w}^\mathrm{T}\mathrm{w} = \ \|\mathrm{w}^2\|$ denotes the normal vector

where $\phi(\mathrm{x}_i)$ denotes the transformed input space vector

where b denotes a bias parameter

where y_i denotes the i-th target value

Figure 2.2: Relevant Equation

– linear	$k(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1 \cdot \mathbf{x}_2$
– polynomial	$k(\mathbf{x}_1, \mathbf{x}_2) = (\gamma\, \mathbf{x}_1 \cdot \mathbf{x}_2 + c)^d$
– Gaussian or radial basis	$k(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(-\gamma \|\mathbf{x}_1 - \mathbf{x}_2\|^2\right)$
– sigmoid	$k(\mathbf{x}_1, \mathbf{x}_2) = \tanh(\gamma\, \mathbf{x}_1 \cdot \mathbf{x}_2 + c)$

Figure 2.3: Kernel Equations

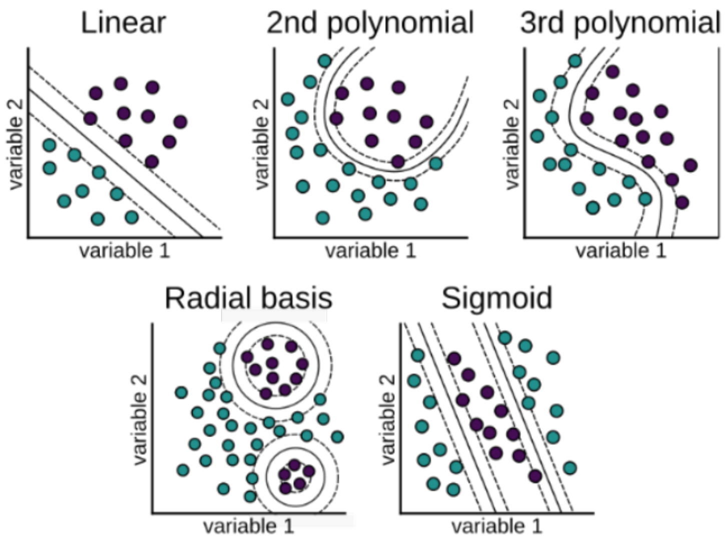


Figure 2.4: Kernel Patterns

tion, is shown in figure 2.5. This shows how data can be linearly separable in a different space and forms a complex separation in the original space:

Soft and hard margins – A support vector machine has a hyper parameter known as the c value. This allows for high or low amounts of miss classification, essentially allowing data points to fall within the margin and on the wrong side of the decision boundary as shown in figure 2.6.

These are referred to as soft (lots of misclassification) and hard (no misclassification) margins and are useful for when we want our model to generalize more. Usually, by allowing more misclassification, we achieve better generalisation while keeping stricter margins gives higher accuracy.

Advantages:

- Support vector machines are made for classification, which is ideal for

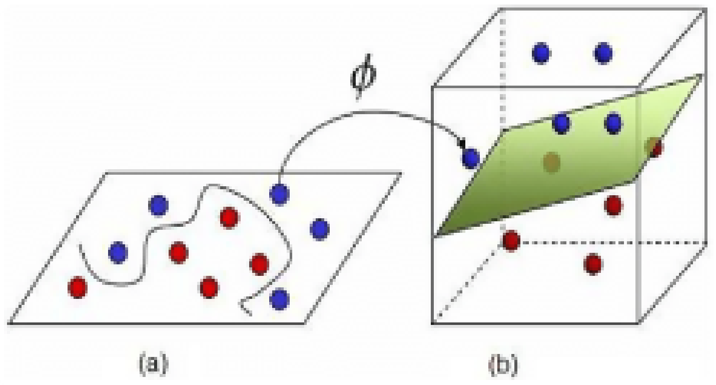


Figure 2.5: Transformation example

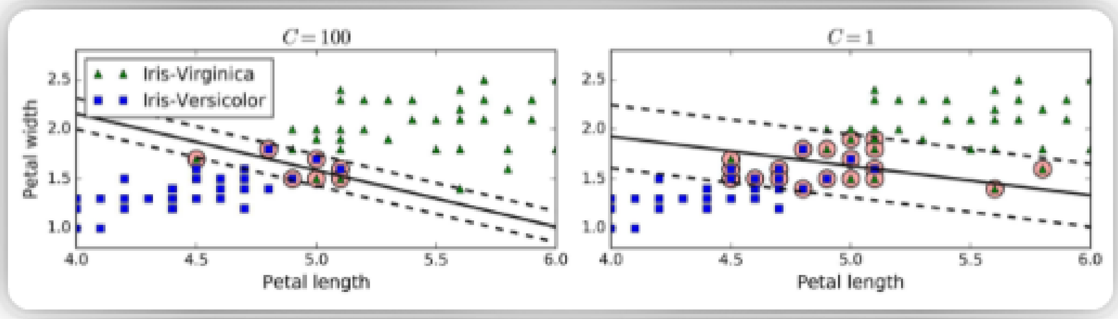


Figure 2.6: Effect of miss classification with soft and hard margins

my problem as I will have a large number of variables as each new word encountered is a new variable.

- Each emotion can be easily separated by a hyperplane and the use of misclassification will allow for certain words to fall on both sides of the decision boundary which they should.
- Due to text having to be Vectorised I will have many parameters as each new word will correspond to an additional vector position. Support vector machines are known for being good with dealing with many variables due to the increase in dimensions not increasing training time as much as other algorithms.
- Support vector machines generalize well due to the allowing of misclassification which will be key when dealing with text.

Disadvantages:

- Support vector machines have long training times with large amounts of data, due to considering all the data at once rather than adjusting as it looks through the data, but due to text containing little data and training only needed to be done once this should be too much of a problem for the machine.
- Support vector machines take a vector of all known words so positioning is lost, this is a key bit of information when detecting emotion for example “not happy” would be sad statement only because of the not before the happy but a support vector machine would not consider this positioning only the words present.

2.1.2 Neural Networks and 1D convolution

A neural network consists of 3 main sections, the input layer, output layer and hidden layer.

The input layer consists of all the data points intended to put into the network for picking up on patterns and trends. If we had a 20x20 image we could take the data from each pixel of the image and feed this into the input layer meaning we would need 400 input neurons.

The hidden layer is where all the pattern finding is done. Each neuron in this layer is a function of all the neurons in the previous layer that will be looking for some specific pattern in the data once it is trained. Every neuron we add can look for a different pattern and each layer we add allows the network to pick up on more complex patterns, this layer is trained via back propagation which adjusts the weightings in each function (neuron) depending on what outcome we got and what outcome we actually want in the output layer.

The output layer consists of the classes we would like the model to choose between, in this case we could have a output neuron for each emotion. The number output to each neuron is how likely the network thinks the input is that class, the neuron with the highest number is the guess the network would make. The network will have to be shown many labelled examples of the classes so that neurons can be adjusted to pick up on the patterns present in each class. A visual representation of this can be seen In figure 2.7.

A neural network is efficient due to it having the following features and ability

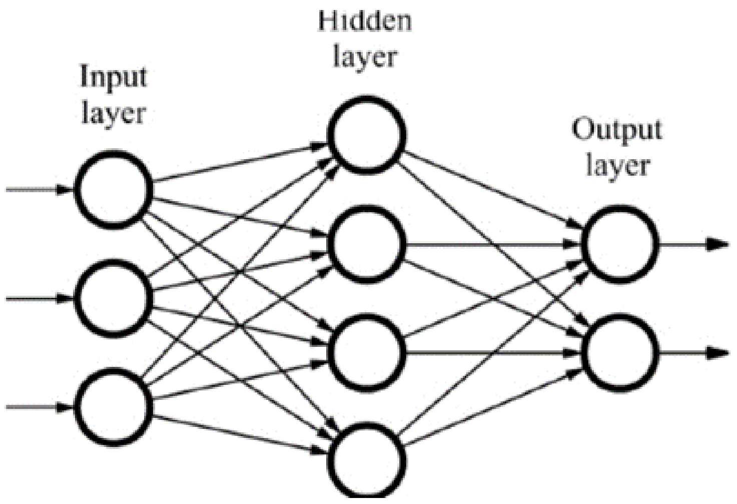


Figure 2.7: Structure of a basic neural net

to tune certain hyper parameters, these will all be key to my investigation and obtaining a high degree of accuracy:

Cost function – This allows us to evaluate the model by seeing how far the prediction of the model is off from the actual label. When the cost function is high, we know that the model is quite far off and needs its weights adjusting. The formula for cost function is shown in figure 2.8, it works by taking the sum of all the errors squared.

Back propagation and forward propagation – Forward propagation is the process of feeding data through the network. Once data is put through the input layer it is fed forward to the output layer where a prediction is made. Back propagation is the process of working backwards through the network once a prediction has been made to adjust the weightings in neurons to get a better output. The adjustments are made via gradient descent.

Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

Parameters: θ_0, θ_1

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Figure 2.8: Cost function equation

Gradient Descent – This is used to adjust a neurons weight to change the outcome of the final network to be more accurate. A neuron is adjusted down if it is found that it has high impact on a neuron that need to be lowered or raised if a neuron is having low impact on a neuron that needs to be lifted. It does this through performing calculus on the cost function and seeing which weights of which neuron that needs to be changed to take "steps" towards an optimum point. Gradient descent does have a problem however which is highlighted in figure 2.9 where a model can get caught in a local optimum rather than global, to combat this we use the learning rate.

Learning Rate – This is a hyper parameter that determines how big the "steps" in gradient descent are. Too large steps and you will never reach an optimal as gradient decent will overshoot the optimum but too small steps and you are likely to get caught in a local optimum and a small learning rate also leads to much longer training times. Examples of these are shown in figure 2.10 with a small learning rate for the first figure and a large one for the second:

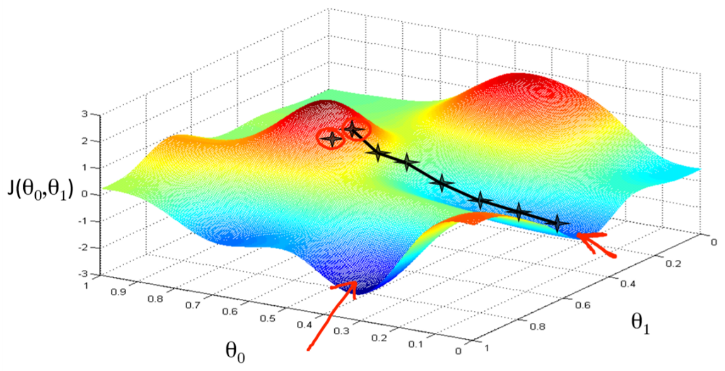


Figure 2.9: Gradient descent example

Activation functions – Activation functions are applied at each layer of the network; they change the output of a neuron by feeding its output it into the respective function. This is done to avoid dead nodes or nodes taking over. The main activation functions are listed in figure 2.11 with ReLU and sigmoid being the most popular.

Batch size – This determines how many examples are looked at before gradient descent is performed, and the weights of the network adjusted.

Bias Term – This is the constant that is added in at each layer to be taken account into the next layer. This can be though of as the y intercept of each function.

Dropout – This drops some neurons from the network on each round of training temporarily, to help with overfitting and dead neurons by not allowing the network to rely on one or a couple neurons.

Optimizer – Different algorithms for applying gradient descent and the learn-

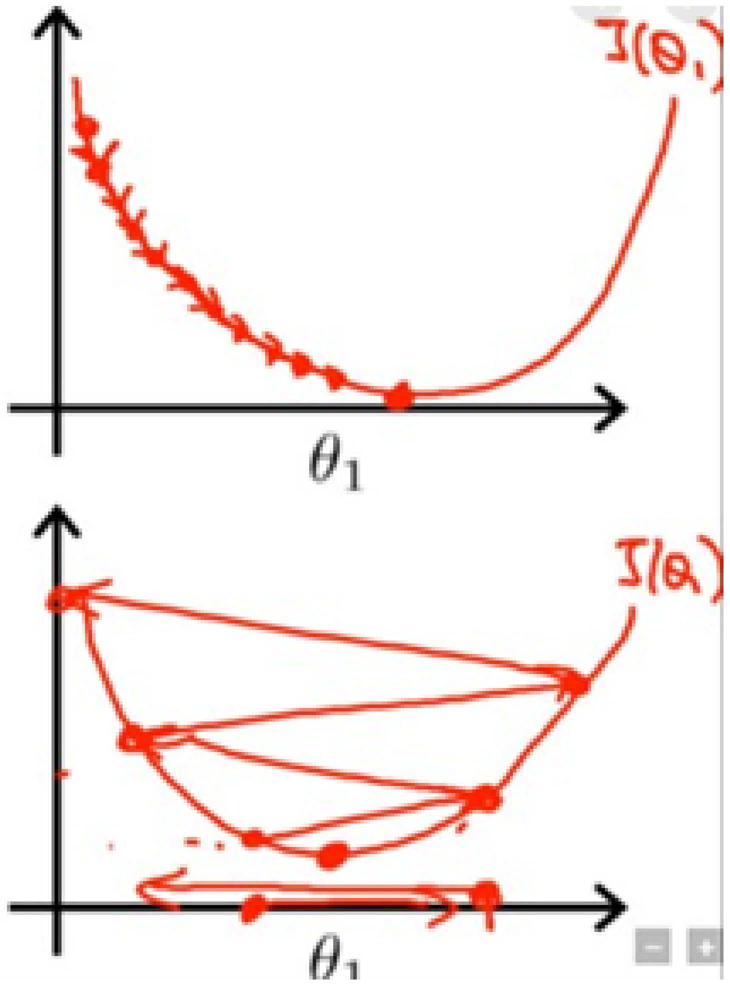


Figure 2.10: Too small learning rate vs Too large learning rate

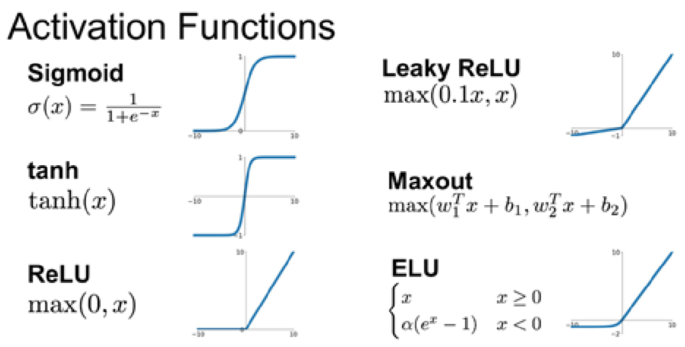


Figure 2.11: Activation function equations and visualization

ing rate for more efficient training.

Loss function – This is used to decide how the error is measured, only needs changing when switching between binary classification problems and multi-class classification problems.

Epochs – The number of times the model will be shown the same training data.

Deep neural nets – This simply refers to having multiple hidden layers. This allows our network to pick up on more complex patterns at the risk of over-fitting to the training data.

1D Convolution - A convolutional layer will help reduce noise and the amount of data points input from a vector. It will do this by creating a window to pass over the vector and combining the entries underneath the window via the weight in each position of the window. The window will be moved along the vector until each vector position has been seen by a position in the window. These weights will be adjusted through back propagation just like

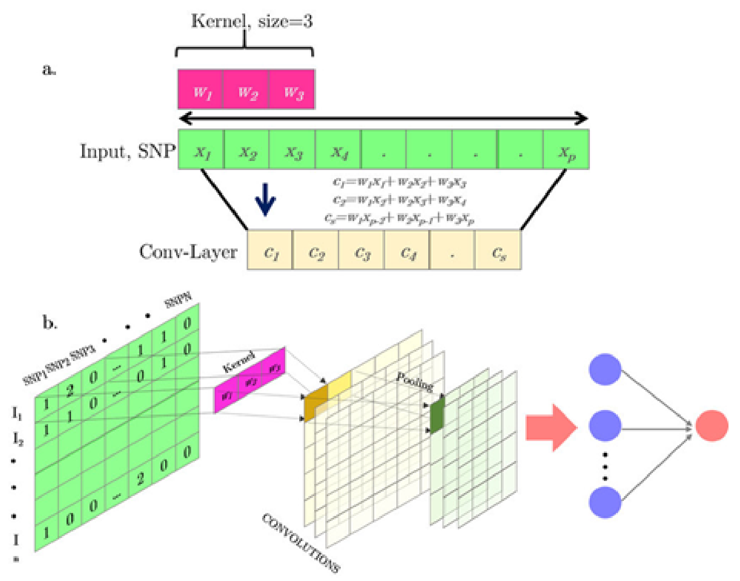

Kernel, size=3
a.
w_1 w_2 w_3
Input, SNP
x_1 x_2 x_3 x_4 x_p
$c_1=w_1x_1+w_2x_2+w_3x_3$
$c_2=w_1x_2+w_2x_3+w_3x_4$
$c_s=w_1x_{p-2}+w_2x_{p-1}+w_3x_p$
Conv-Layer
c_1 c_2 c_3 c_4 c_s
b.
SNP1 SNP2 SNP3
SNPN
I_1
I_2
I_n
Kernel
Pooling
CONVOLUTIONS

Figure 2.12: Windows passing over vector/Matrix example. Top = 1D convolution Bottom = 2D convolution

our neurons in the neural network. The convolutional layer also has a new hyper parameter known as the Kernel , this decides the size of the window passed over the data.

Advantages:

- Can pick up on complex patterns as well as look for multiple patterns and features at once. This allows for more complex data to be input and non-obvious patterns to be seen such as in looking at predicting the price of a house. One neuron could be looking for very old houses as these have a high price and certain features that increase or decrease their price. A second neuron could be looking at very new builds as these also have a higher price and discover features that are associated with new builds that effect the price.

- Large amounts of data can be processed at once or some at a time as the network adjusts after ever few examples (based of the batch size) rather than after looking at the whole data set. This allows for training to be started and stopped whenever and allows for you to start with a pre weighted model before training.
- The use of convolution allows me to look at different forums of input into our model rather than just set parameters and allows me to look at relationship between corresponding positions.

Disadvantages:

- Require large amounts of computation.
- The large number of hyper parameters leads to a long time spent tuning.
- The network itself amounts to a black box and it is often hard to see exactly what the network is doing, or “thinking” and what each neuron is exactly looking for. The best we can do is guess.
- Large amounts of data needed for training.

2.1.3 RNN and LSTM

These methods hold the same features of a neural network except they modify forward propagation slightly. The basic idea behind these two concepts is

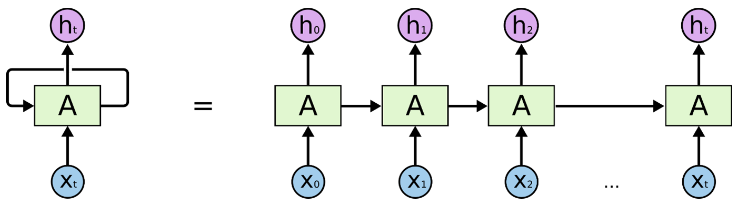


Figure 2.13: RNN example

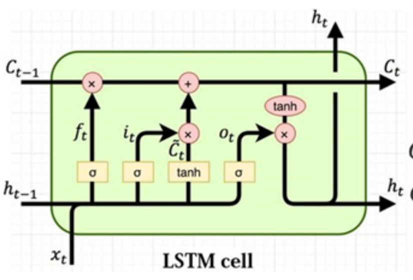
rather than feeding all the data at once, variables can be fed a few at a time or one at a time and be circulated through the network or stored in a neuron, eventually passing through a layer multiple time. This leads to the model having a sort of memory giving it the ability to analysis data where the data points are influenced by previous points, ie not independent.

A great example of where this is useful is in the stock market. Time series data is very important here and all stock prices are influenced by their previous price meaning for predicting of new prices being able to look back at old prices influence is invaluable.

RNN, Basic memory layer:

This works just like a normal neuron except each neuron is a function of the current state and all previous inputs and instead of data being put through all at once the data is looped over. Outputs can be taken in sequence or the last output can be considered. Essentially the output of each position is fed back round into the network and a new position is considered until all positions have been seen.

Long Short-term memory layer:



$$i_t = \sigma\big(x_t U^i + h_{t-1} W^i\big)$$
$$f_t = \sigma\big(x_t U^f + h_{t-1} W^f\big)$$
$$o_t = \sigma\big(x_t U^o + h_{t-1} W^o\big)$$
$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$
$$C_t = \sigma\big(f_t * C_{t-1} + i_t * \tilde{C}_t\big)$$
$$h_t = \tanh(C_t) * o_t$$

Figure 2.14: LSTM cell

The problem with the basic recurrent memory cell is that initial data is drowned out rather fast as information is lost at every step, this makes the model amount to having short term memory loss. LSTM hopes to solve this by giving value to an input and deciding whether it is useful information that will influence the rest of the data. The choice of whether data is included is taken into account using the logistics action function and stops useless data drowning out important previous data by discarding it and not letting it effect the overall output.

A LSTM Cell can be seen in figure 2.14. The path of the data is shown here and on the right hand side the formulas that the data is subject too are shown. Each is a different sum of weightings passed through an activation function. The weightings are learnt via back propagation.

Advantages:

- Considers sequencing and implements memory into the model. This is ideal for our task as words prior to a word do influence the mood they show for example “not happy” or “very angry”.

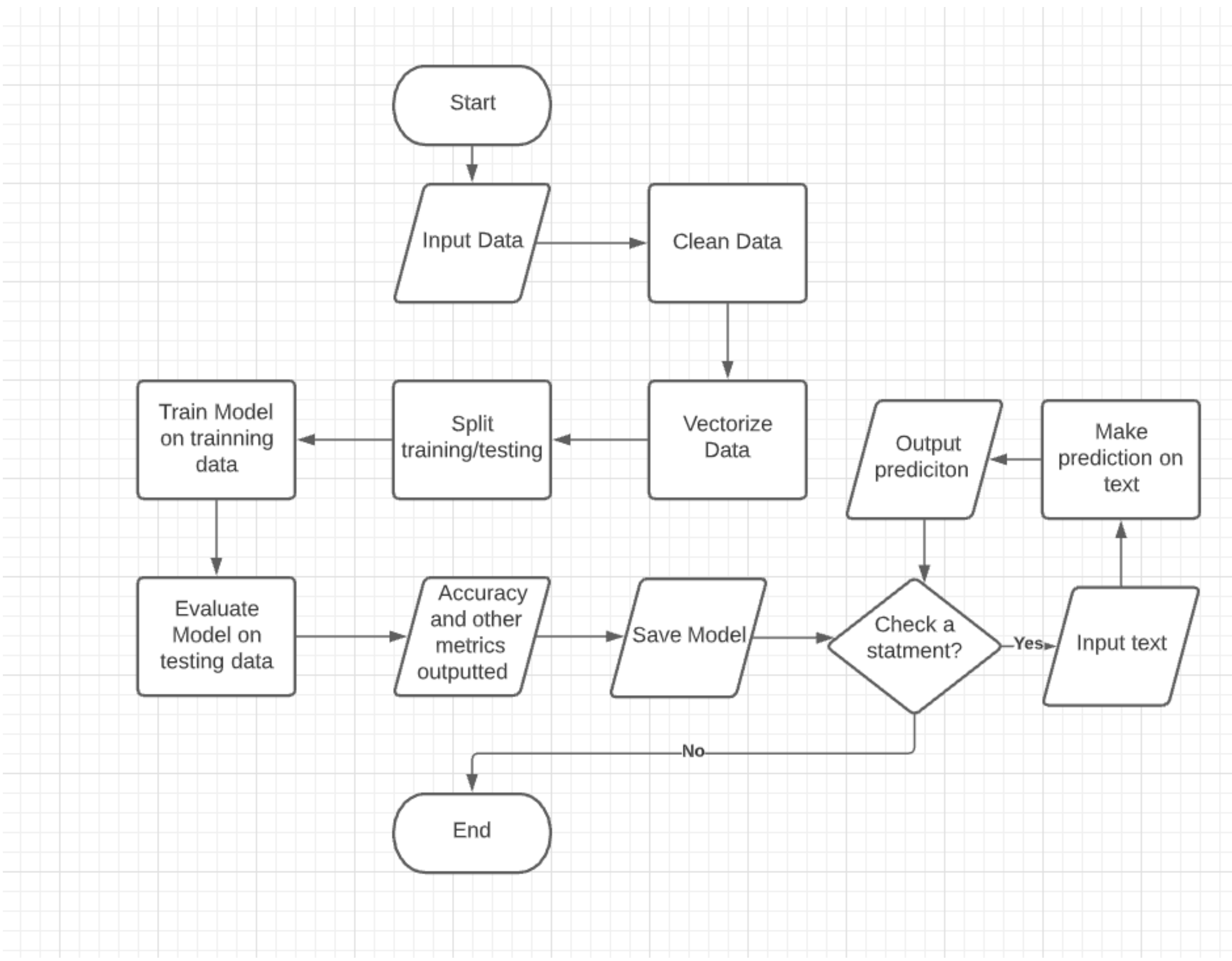
- Allows for data format where variables are not independent such as time series data or a sentence.

Disadvantages:

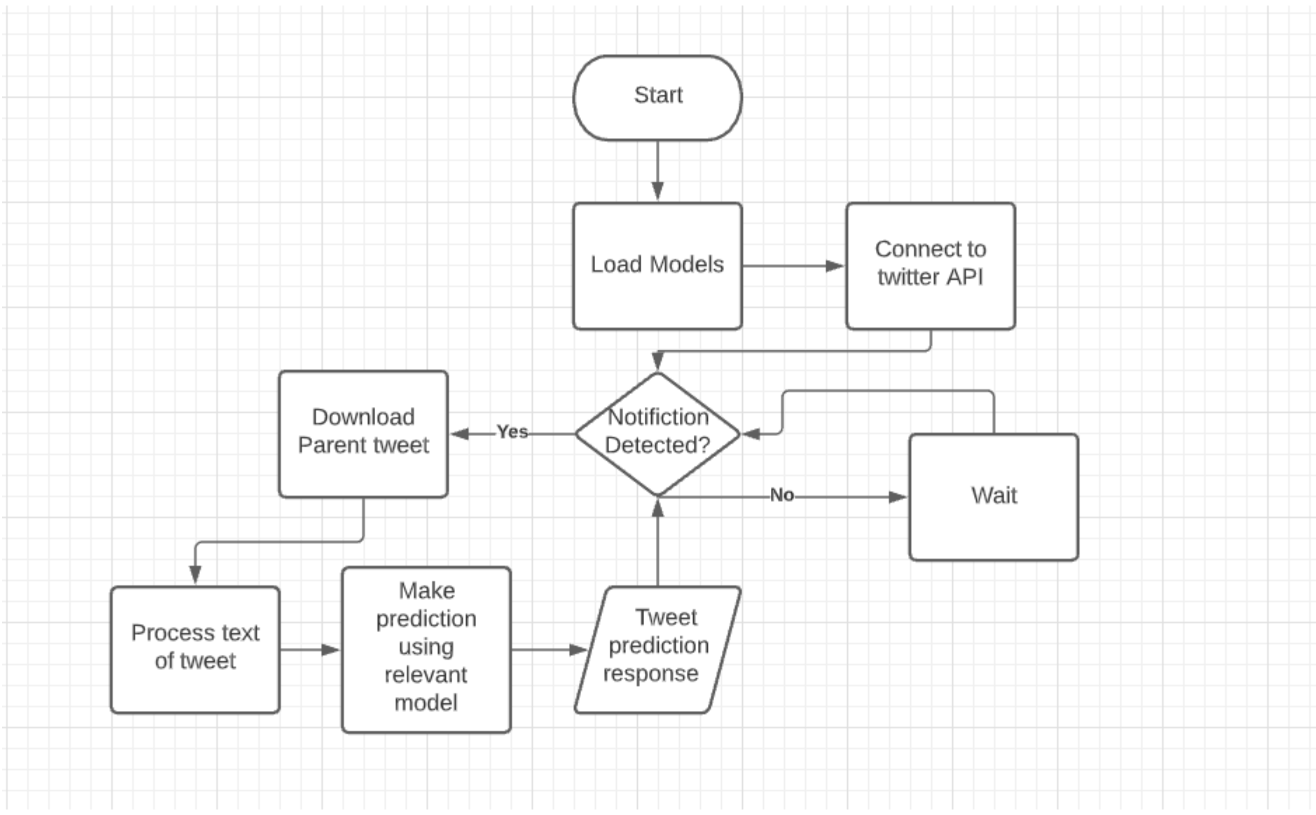
- Suffers heavily from the vanishing gradient problem and the exploding gradient problem.
- RNNs are not feasible in a deep model and so complex patters can be hard to pick up on
- Large amount of data needed.

2.2 Flowcharts

Creation of model



Deployment of model



2.3 Software and Libraries

Python 3.7 64 bit – Python will be the only language used for this project. It has great versatility when it comes to data manipulation and presentation, as well as leading machine learning capabilities.

Tensorflow 2.4.1 and keras 2.4.3 – These are the main libraries I will be using to do advanced machine learning. Keras serves at the front end allowing me to make complex neural networks and models rather easily and Tensorflow servers as the back end to run our calculations.

SK-Learn 0.24.0 – Used in the creation and training of support vector machines.

Numpy 1.19.3 and Pandas 1.1.5 – Both will be used for the handling and preparation of data from csv to model.

Matplotlib 3.3.3 – Matplotlib will be used for graphing data and metrics output by training.

Pygame 2.0.1 – Pygame will be used for light GUI creation to aid in the sorting of data.

Tweepy 3.10.0 – Tweepy will be used to connect to the twitter API allowing me to download tweets and monitor the associated twitter account. I will need to obtain approval from twitter for my project to gain API keys.

NLTK 3.5 and Enchant 3.2.0 – Both will be used in the processing and cleaning of raw text.

Cuda 12.0 and cuDNN 8.0 – These will be used to allow for models to be run on my graphics card speeding up training times considerably.

Joblib – Used for saving and importing SK-Learn models.

Chapter 3

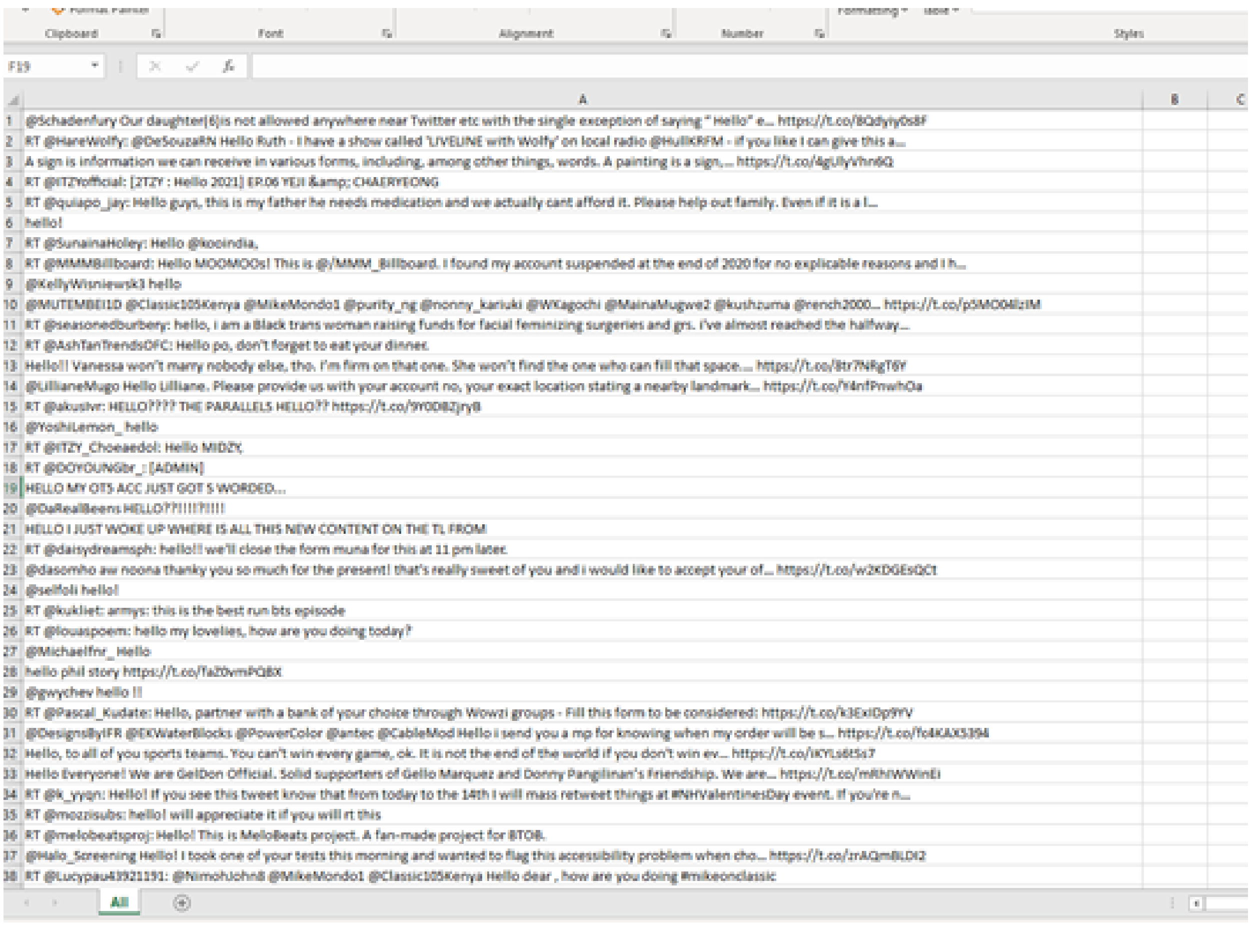
Implementation

3.1 Data Gathering

Most of my data will be self-gathered using the tweepy library to connect to the twitter api and download relevant tweets. The code for this can be seen in appendix A.1.1. Downloading the tweets is a simple process, all that needs to be stipulated is the search term which will download the most recent tweets containing the specified term. I also manipulated the dates of when it would be pulling tweets from as to not get any topical bias [9].

The code works as follows, a list of search terms is stipulated, for example "happy", "sad", "feeling", 100 tweets are downloaded that correspond to each search term and wrote line by line to a csv. The output of this is shown in figure 3.1.

Through this I downloaded around 13,000 tweets. Note: Sometimes lines can-



	A	B	C
1	@Schadenfury Our daughter(6)is not allowed anywhere near Twitter etc with the single exception of saying " Hello" e… https://t.co/BQdyiy0s8F		
2	RT @HareWolfy: @DeSouzaRN Hello Ruth - I have a show called 'LIVELINE with Wolfy' on local radio @HullKRFM - if you like I can give this a…		
3	A sign is information we can receive in various forms, including, among other things, words. A painting is a sign,… https://t.co/4gUlyVhn6Q		
4	RT @ITZYofficial: [2TZY : Hello 2021] EP.06 YEJI &amp; CHAERYEONG		
5	RT @quiapo_jay: Hello guys, this is my father he needs medication and we actually cant afford it. Please help out family. Even if it is a l…		
6	hello!		
7	RT @SunainaHoley: Hello @kooindia,		
8	RT @MMMBillboard: Hello MOOMOOs! This is @/MMM_Billboard. I found my account suspended at the end of 2020 for no explicable reasons and I h…		
9	@KellyWisniewsk3 hello		
10	@MUTEMBEI1D @Classic105Kenya @MikeMondo1 @purity_ng @nonny_kariuki @WKagochi @MainaMugwe2 @kushzuma @rench2000… https://t.co/p5MO04lzIM		
11	RT @seasonedburbery: hello, i am a Black trans woman raising funds for facial feminizing surgeries and grs. i've almost reached the halfway…		
12	RT @AshTanTrendsOFC: Hello po, don't forget to eat your dinner.		
13	Hello!! Vanessa won't marry nobody else, tho. I'm firm on that one. She won't find the one who can fill that space.… https://t.co/8tr7NRgT6Y		
14	@LillianeMugo Hello Lilliane. Please provide us with your account no, your exact location stating a nearby landmark… https://t.co/Y4nfPnwhOa		
15	RT @akuslvr: HELLO???? THE PARALLELS HELLO?? https://t.co/9Y0DBZjryB		
16	@YoshiLemon_ hello		
17	RT @ITZY_Choeaedol: Hello MIDZY,		
18	RT @DOYOUNGbr_: [ADMIN]		
19	HELLO MY OTS ACC JUST GOT S WORDED…		
20	@DaRealBeens HELLO??!!!!?!!!!		
21	HELLO I JUST WOKE UP WHERE IS ALL THIS NEW CONTENT ON THE TL FROM		
22	RT @daisydreamsph: hello!! we'll close the form muna for this at 11 pm later.		
23	@dasomho aw noona thanky you so much for the present! that's really sweet of you and i would like to accept your of… https://t.co/w2KDGEsQCt		
24	@selfoli hello!		
25	RT @kukliet: armys: this is the best run bts episode		
26	RT @louaspoem: hello my lovelies, how are you doing today?		
27	@Michaelfnr_ Hello		
28	hello phil story https://t.co/faZ0vmPQBX		
29	@gwychev hello !!		
30	RT @Pascal_Kudate: Hello, partner with a bank of your choice through Wowzi groups - Fill this form to be considered: https://t.co/k3ExiDp9YV		
31	@DesignsByIFR @EKWaterBlocks @PowerColor @antec @CableMod Hello i send you a mp for knowing when my order will be s… https://t.co/fo4KAX5394		
32	Hello, to all of you sports teams. You can't win every game, ok. It is not the end of the world if you don't win ev… https://t.co/iKYLs6tSs7		
33	Hello Everyone! We are GelDon Official. Solid supporters of Gello Marquez and Donny Pangilinan's Friendship. We are… https://t.co/mRhIWWinEi		
34	RT @k_yyqn: Hello! If you see this tweet know that from today to the 14th I will mass retweet things at #NHValentinesDay event. If you're n…		
35	RT @mozzisubs: hello! will appreciate it if you will rt this		
36	RT @melobeatsproj: Hello! This is MeloBeats project. A fan-made project for BTOB.		
37	@Halo_Screening Hello! I took one of your tests this morning and wanted to flag this accessibility problem when cho… https://t.co/zrAQmBLDI2		
38	RT @Lucypau43921191: @NimohJohn8 @MikeMondo1 @Classic105Kenya Hello dear , how are you doing #mikeonclassic		

Figure 3.1: Intial data gathered

not be downloaded due to encoding errors with emojis and certain characters hence the try, except portion of the code. I would like to have investigated the use of emojis however the error with importing them and the fact that different operating systems and keyboard applications used different codes to represent emojis made this near impossible.

The data now needed to be labelled with the relevant emotion. To aid me in this process I created a GUI with PyGame that allowed me to quickly label data. The code for this can be seen in appendix A.1.2.

The code works by loading in the data from the csv and presenting each tweet one at a time on the screen. The user then can click buttons that are labelled with the corresponding emotion. Once a button is pressed the tweet is saved in a new csv with its label and the next tweet is presented. You can see the application here in figure 3.2.

Clearly the tweet in figure 3.2 would be labelled as sad. Note also that tweets that made no sense or where confusing where just excluded and not added to the database.

Finally, some supplementary data was sourced from online Kaggle data bases for both sarcastic vs none and happy vs sad, links to these can be found in the references. Then I created 4 data sets, one for happy vs sad,, one for Happy VS Sad Vs None, one for angry vs none and one for sarcastic vs none[15][16].

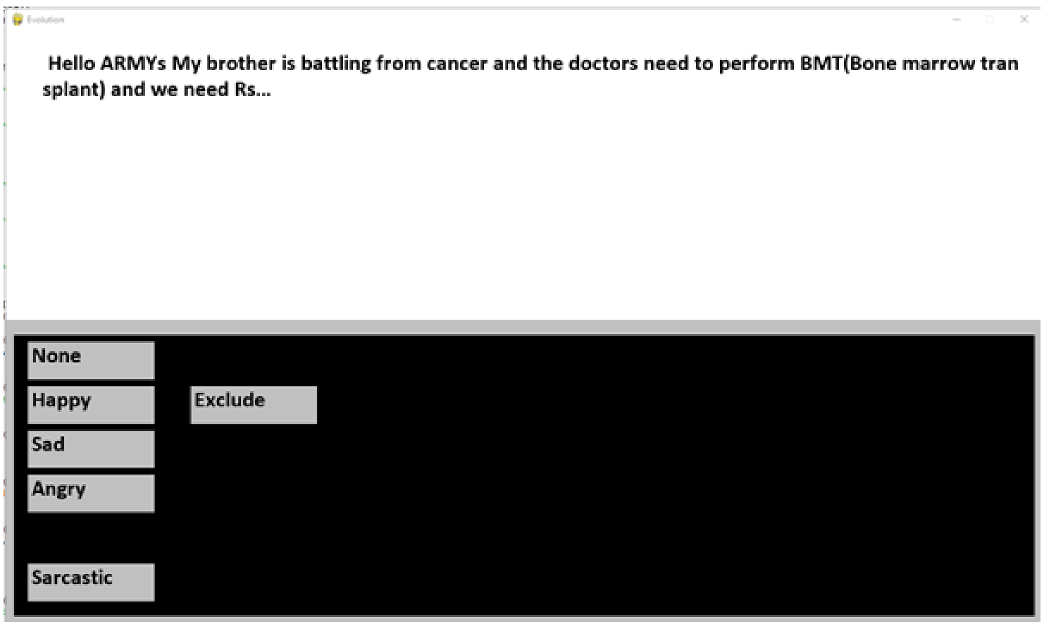

Hello ARMYs My brother is battling from cancer and the doctors need to perform BMT(Bone marrow tran splant) and we need Rs...

Figure 3.2: Data labelling program

3.2 Data Cleaning

At first the data was quite messy as shown in figure 3.1, this data would be almost impossible for a machine to read and understand so cleaning needed to take place. This data was cleaned several ways.

Firstly, I removed links and punctuation as I wanted to focus on the words as inputs to my models. You can see the code for this in appendix A.1.3.

Secondly, I used the Enchant library to go through each word in my tweets and check if they were an actual English word. If it was not it would try to guess what the word was supposed to be. This was great as a lot of people on twitter aren't too concerned with how they spell and the machine learning model wont be able to understand what they are saying. Code for this can be seen in appendix A.1.4 the results of the cleaning can be seen in figure 3.3:

Next, I converted the text into a vector where each position in the vector will correspond to a known word and the number in the vector position to the count of that word in the text. I then comma separates my list and allowed sk-learn to vectorize the data using the inbuilt function Count Vectorize (an example for which can be shown in figure 3.4).

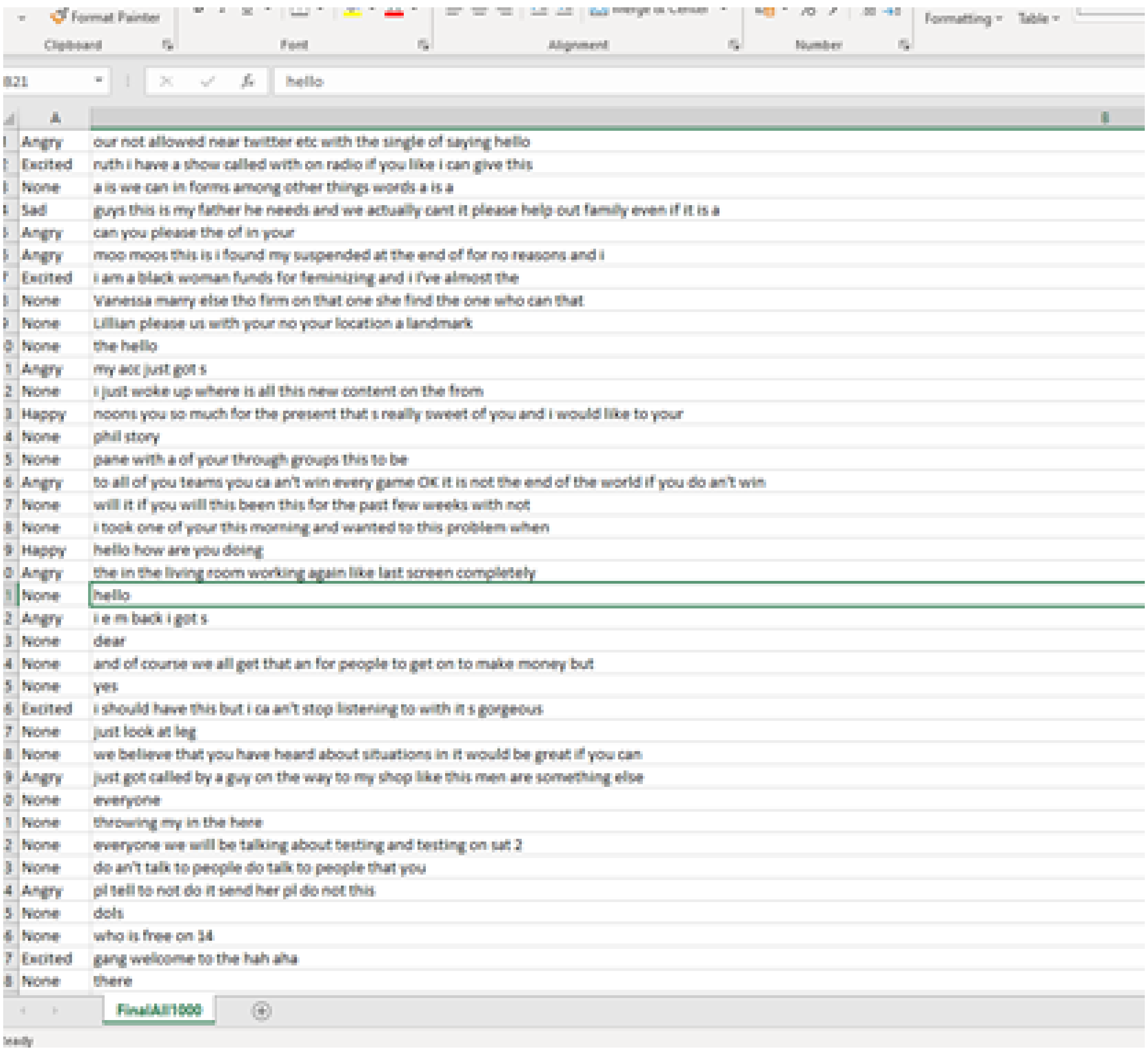
For Tensorflow I went with a slightly different approach, in anticipation of taking sequencing into account. I turned my text into time series data. I gave each word a unique index number and changed the words in each sentence too the corresponding number. I then filled a vector with these numbers, where the position of the number in the vector is its position in the sequence of the sentence and then pad the vector with nulls so that each vector is the same length. These vectors are what I will be using for input into my neural networks, you can see details of the code that does this in appendix A.2.3.

You can see the result in figure 3.5, the happy, sad and none labels where changed to 0,1,2 respectively.

3.3 Overview of training code

3.3.1 Basic models

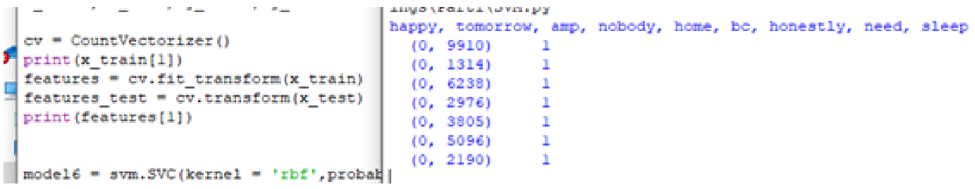
SVM:



B21 hello

	A	
1	Angry	our not allowed near twitter etc with the single of saying hello
2	Excited	ruth i have a show called with on radio if you like i can give this
3	None	a is we can in forms among other things words a is a
4	Sad	guys this is my father he needs and we actually cant it please help out family even if it is a
5	Angry	can you please the of in your
6	Angry	moo moos this is i found my suspended at the end of for no reasons and i
7	Excited	i am a black woman funds for feminizing and i i've almost the
8	None	Vanessa marry else tho firm on that one she find the one who can that
9	None	Lillian please us with your no your location a landmark
0	None	the hello
1	Angry	my acc just got s
2	None	i just woke up where is all this new content on the from
3	Happy	noons you so much for the present that s really sweet of you and i would like to your
4	None	phil story
5	None	pane with a of your through groups this to be
6	Angry	to all of you teams you ca an't win every game OK it is not the end of the world if you do an't win
7	None	will it if you will this been this for the past few weeks with not
8	None	i took one of your this morning and wanted to this problem when
9	Happy	hello how are you doing
0	Angry	the in the living room working again like last screen completely
1	None	hello
2	Angry	i e m back i got s
3	None	dear
4	None	and of course we all get that an for people to get on to make money but
5	None	yes
6	Excited	i should have this but i ca an't stop listening to with it s gorgeous
7	None	just look at leg
8	None	we believe that you have heard about situations in it would be great if you can
9	Angry	just got called by a guy on the way to my shop like this men are something else
0	None	everyone
1	None	throwing my in the here
2	None	everyone we will be talking about testing and testing on sat 2
3	None	do an't talk to people do talk to people that you
4	Angry	pl tell to not do it send her pl do not this
5	None	dols
6	None	who is free on 14
7	Excited	gang welcome to the hah aha
8	None	there

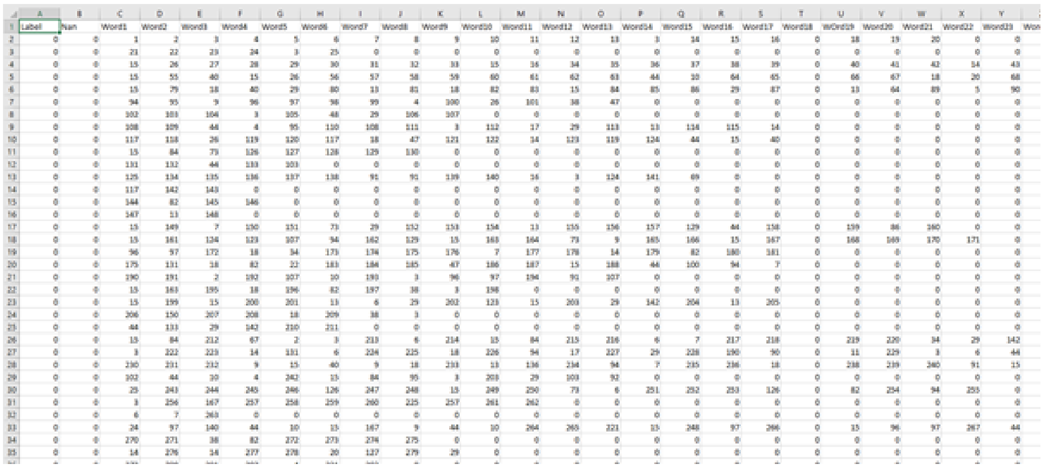
Figure 3.3: data after cleaning



```
cv = CountVectorizer()
print(x_train[1])
features = cv.fit_transform(x_train)
features_test = cv.transform(x_test)
print(features[1])


model6 = svm.SVC(kernel = 'rbf',probab
```

```
happy, tomorrow, amp, nobody, home, bc, honestly, need, sleep
  (0, 9910)     1
  (0, 1314)     1
  (0, 6238)     1
  (0, 2976)     1
  (0, 3805)     1
  (0, 5096)     1
  (0, 2190)     1
```

Figure 3.4: Vectorization example



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
1	Label	Nan	Word1	Word2	Word3	Word4	Word5	Word6	Word7	Word8	Word9	Word10	Word11	Word12	Word13	Word14	Word15	Word16	Word17	Word18	Word19	Word20	Word21	Word22	Word23
2	0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	3	14	15	16	0	18	19	20	0	0
3	0	0	21	22	23	24	3	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	15	26	27	28	29	30	31	32	33	15	16	34	35	36	37	38	39	0	40	41	42	14	43
5	0	0	15	55	40	15	26	56	57	58	59	60	61	62	63	44	10	64	65	0	66	67	18	20	68
6	0	0	15	79	18	40	29	80	13	81	18	82	83	15	84	85	86	29	87	0	13	64	89	5	90
7	0	0	94	95	9	96	97	98	99	4	100	26	101	38	47	0	0	0	0	0	0	0	0	0	0
8	0	0	102	103	104	3	105	48	29	106	107	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	108	109	44	4	95	110	108	111	3	112	17	29	113	13	114	115	14	0	0	0	0	0	0
10	0	0	117	118	26	119	120	117	18	47	121	122	14	123	119	124	44	15	40	0	0	0	0	0	0
11	0	0	15	84	73	126	127	128	129	130	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	131	132	44	133	103	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	125	134	135	136	137	138	91	91	139	140	16	3	124	141	69	0	0	0	0	0	0	0	0
14	0	0	117	142	143	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	144	82	145	146	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	147	13	148	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	15	149	7	150	151	73	29	152	153	154	13	155	156	157	129	44	158	0	159	86	160	0	0
18	0	0	15	161	124	123	107	94	162	129	15	163	164	73	9	165	166	15	167	0	168	169	170	171	0
19	0	0	96	97	172	18	34	173	174	175	176	7	177	178	14	179	82	180	181	0	0	0	0	0	0
20	0	0	175	131	18	82	22	183	184	185	47	186	187	15	188	44	100	94	7	0	0	0	0	0	0
21	0	0	190	191	2	192	107	10	193	3	96	97	194	91	107	0	0	0	0	0	0	0	0	0	0
22	0	0	15	163	195	18	196	82	197	38	3	198	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	15	199	15	200	201	13	6	29	202	123	15	203	29	142	204	13	205	0	0	0	0	0	0
24	0	0	206	150	207	208	18	209	38	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	44	133	29	142	210	211	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	15	84	212	67	2	3	213	6	214	15	84	215	216	6	7	217	218	0	219	220	34	29	142
27	0	0	3	222	223	14	131	6	224	225	18	226	94	17	227	29	228	190	90	0	11	229	3	6	44
28	0	0	230	231	232	9	15	40	9	18	233	13	136	234	94	7	235	236	18	0	238	239	240	91	15
29	0	0	102	44	10	4	242	15	84	95	3	203	29	103	92	0	0	0	0	0	0	0	0	0	0
30	0	0	25	243	244	245	246	126	247	248	15	249	250	73	6	251	252	253	126	0	82	254	94	255	0
31	0	0	3	256	167	257	258	259	260	225	257	261	262	0	0	0	0	0	0	0	0	0	0	0	0
32	0	0	6	7	263	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	0	0	24	97	140	44	10	15	167	9	44	10	264	265	221	13	248	97	266	0	15	96	97	267	44
34	0	0	270	271	38	82	272	273	274	275	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
35	0	0	14	276	14	277	278	20	127	279	29	0	0	0	0	0	0	0	0	0	0	0	0	0	0

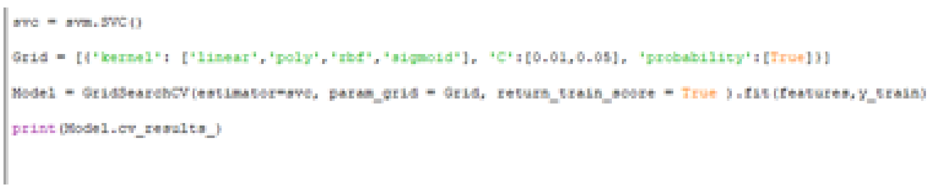
Figure 3.5: Data in time series format

At this point the data is cleaned and ready for training. Training is the same for all support vector machines, example code of a support vector machine being trained can be seen in appendix A.2.2 and B.1. Data is split into testing and training using a seed that will be universal across all models so that the same data is being looked at. It is the then turned into a vector and sent to the model. Initial training bought with it a few problems. Initially overfitting was quite high so the c values I was testing where lowered, and this mostly solved the problem. To further reduce overfitting, I removed words that had less than a count of 2 to reduce the number of variables which is a common way to address overfitting.

I also tried limiting the words so that only the top 1000 and 5000 words would be present, but this did not bring much higher accuracy's and was abandoned.

A range of models were tested for each of the data sets, example of this training can be seen in appendix B.1. Linear and radial showed initial strength as well as a c values between 1 and 0.05 for all data sets. All trained models can be seen in the dropbox linked in the references under SVM models[5]. The final model is chosen using a gird search to select the best model. Gird search lets me select what hyper parameters have been working well and train the model many times with different combinations of these parameters to find the best one. Across the board linear kernel performs the best with a c value of 0.1 but this is detailed more later. An example of a grid search can be seen in figure 3.6.

Neural Network:



```
svc = svm.SVC()

Grid = [{'kernel': ['linear','poly','rbf','sigmoid'], 'C':[0.01,0.05], 'probability':[True]}]

Model = GridSearchCV(estimator=svc, param_grid = Grid, return_train_score = True ).fit(features,y_train)

print(Model.cv_results_)
```

Figure 3.6: Grid search example

Next, I tried Neural Networks both deep and simple. Examples of the code for these can be seen in appendix A.2.4 and B.2. Data is again split and then fed to the model with a batch size of 200. The models are run for 10 epochs so see all the data 10 times and then is evaluated on the testing data. An example of this training and output can be seen in appendix B.2. Neural networks originally disappointed me with low accuracies especially for angry Vs none and sarcasm Vs none. To combat this, I changed my model's loss from categorical crossentropy to binary crossentropy this meant instead of 2 output neurons, one for each category, I would have one neuron with the activation function of sigmoid so that 1 = one class and 0 = second class. This saw decent upticks in the accuracy for these models and once I was happy with them I choose the final hyper parameters using a gird search.

The best models comprised of the softmax activation function and rmsprop optimizer with not too much of a difference between a deep and simple network. A range of models where trained and these can be seen in the dropbox linked in the references under Neural nets. Neural networks however did not come close to the accuracies of SVM showing SVM to the superior model [5].

While I was happy with these results, I was limited as the model does not consider the position of words in the text (as seen in appendix B.7), so I moved onto models that could address sequencing for higher accuracy.

3.3.2 Sequence models

The next models all train and look the same as a neural network. In fact, I copied the structure of my deep neural network for these next models and simply replaced the first layer with the layer for the technique I was working with. All the following models were tuned using the grid search function and can be seen in the dropbox linked in the references under RNN and LSTM [5].

1D Convolution:

1D convolution was the first sequencing model I tried. It works the same as the neural network in terms of training code except it has this layer added[12]:

```
modelNeuralNet.add(layers.Conv1D(128, kernel_size=2,activation="relu",
input_shape=(60,1)))
```

Additional tuning took place to determine the kernel size which decides how many points in succession are looked at for each data point. I found that a kernel size of two did the best and this makes sense as usually the word before a word in a sentence will have the most impact on its meaning. 1D convolution did perform better than neural network but could still not beat SVM on any of the datasets.

LSTM:

LSTM was a straightforward implementation and no more hyperparameter tuning was really needed as this stage. The model follows on from the 1D convolution model but replaces its unique layer with the following[11]:

```
modelNeuralNet.add(layers.Bidirectional(tf.keras.layers.LSTM(64,
return_sequences=True)))
```

LSTM ended up performing the highest out of all my models except SVM when it came to happy vs sad vs none.

Chapter 4

Results

4.1 Metrics

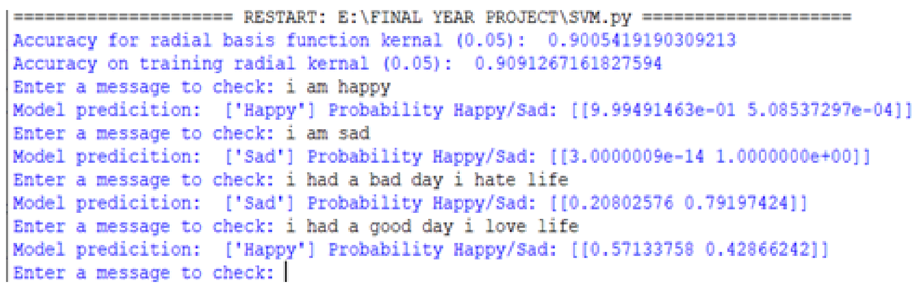
Training examples and final demos are given in section B of the appendix.

4.1.1 Basic models

These are the following best models for each investigation after much training and testing.

Happy vs Sad Vs None:

SVM with kernel = radial and c=0.05 was my first investigation with just happy vs sad on SVM, It is very good at detecting obvious messages but as you can see in figure 4.1 favoured sad over happy.



```
===================== RESTART: E:\FINAL YEAR PROJECT\SVM.py ====================
Accuracy for radial basis function kernal (0.05):  0.9005419190309213
Accuracy on training radial kernal (0.05):  0.9091267161827594
Enter a message to check: i am happy
Model predicition:  ['Happy'] Probability Happy/Sad: [[9.99491463e-01 5.08537297e-04]]
Enter a message to check: i am sad
Model predicition:  ['Sad'] Probability Happy/Sad: [[3.0000009e-14 1.0000000e+00]]
Enter a message to check: i had a bad day i hate life
Model predicition:  ['Sad'] Probability Happy/Sad: [[0.20802576 0.79197424]]
Enter a message to check: i had a good day i love life
Model predicition:  ['Happy'] Probability Happy/Sad: [[0.57133758 0.42866242]]
Enter a message to check: |
```

Figure 4.1:

The confusion matrix for this model can be seen in figure 4.2 and figure 4.3.

SVM with kernel = linear and c=0.1 was my best model for happy vs sad vs none. It is very good at Happy vs sad as would be expected from the previous model but struggles with happy vs none and sad vs none. This can be seen in the confusion matrix in figure 4.4 and 4.5.

Ultimately this would be the model I choose for using in my twitter bot demo for happy vs sad vs none.

Neural Networks couldn't get a model above 70 percent for either investigation, accuracy for the best neural net can be seen in figure 4.6.

Angry Vs None: SVM didn't perform as well here but still came out with come good accuracy's. Again a linear Kernel with a low C value performed the best. The confusion matrix for which can be seen in figure 4.7 and 4.8.

A demo for this model can be seen in figure 4.9.

Sarcasm Vs None: I spent some time working with this model but unfortunately even when accuracy's came above guessing rates the model didn't

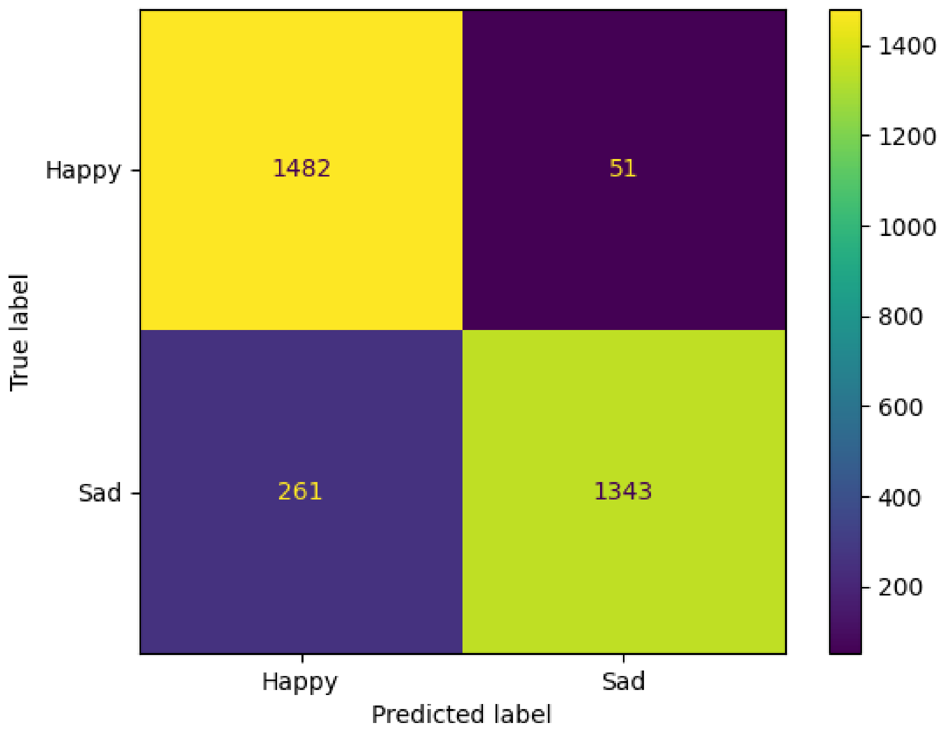


Figure 4.2: Testing data Happy Vs Sad SVM

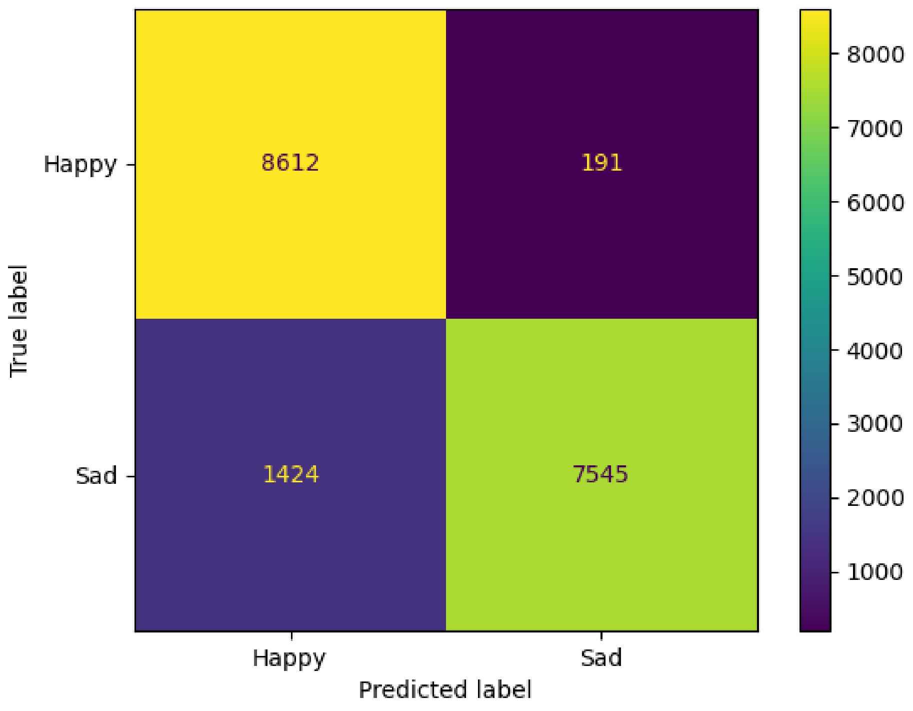


Figure 4.3: Training data Happy Vs Sad SVM

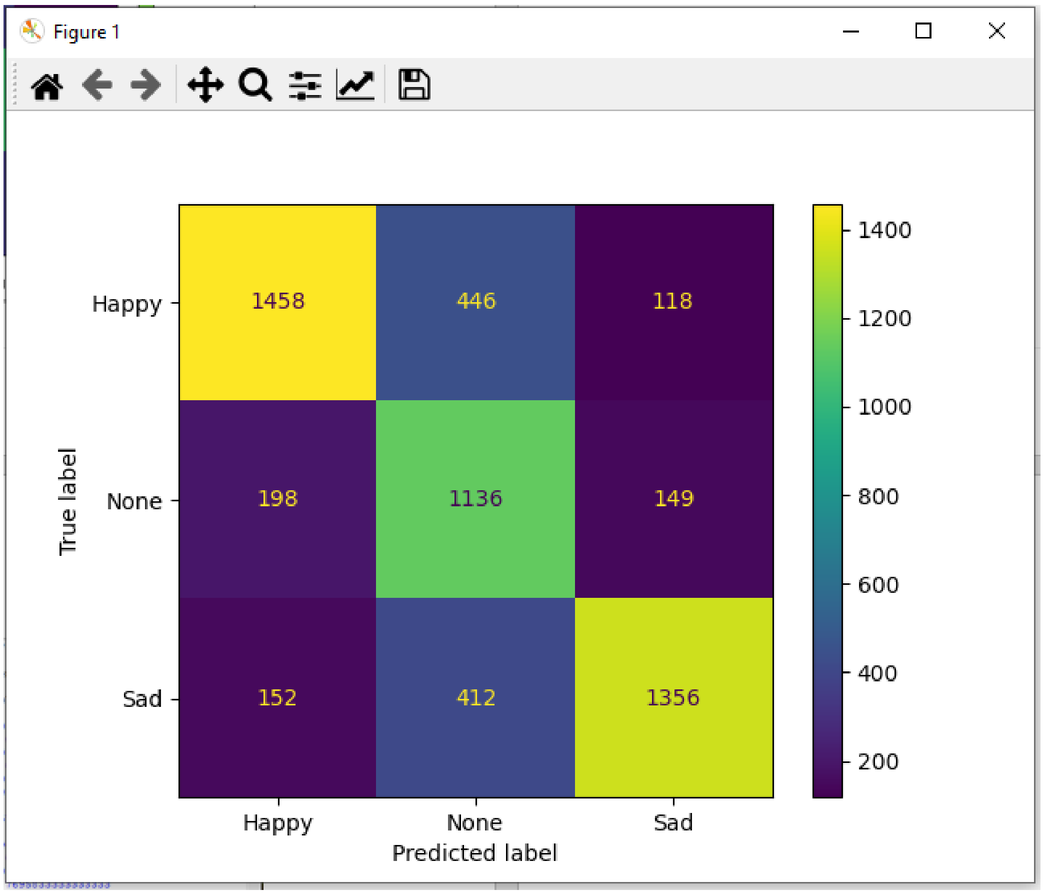


Figure 4.4: Testing data Happy Vs Sad Vs None SVM

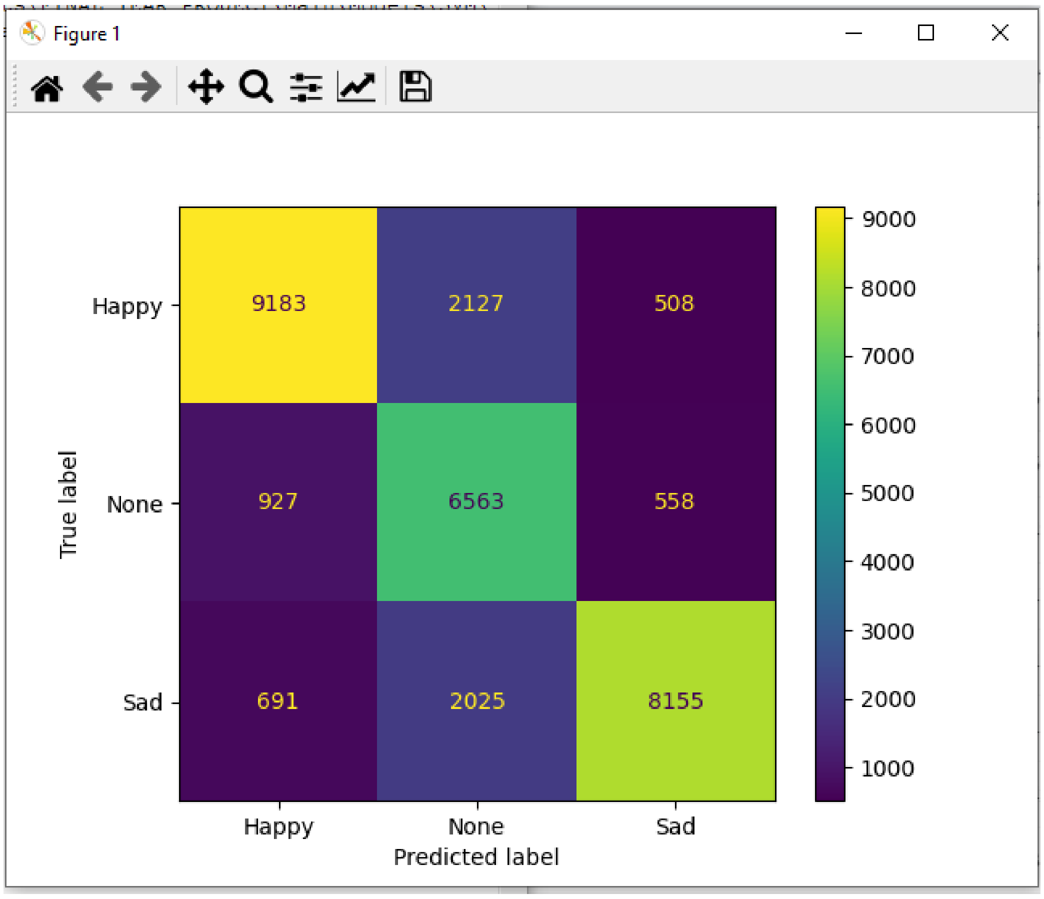
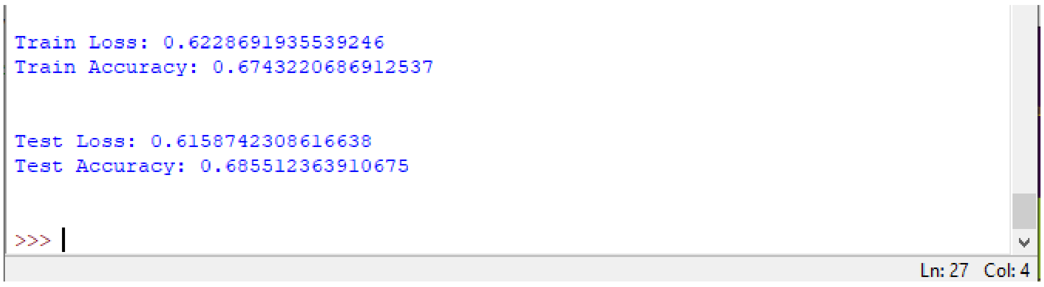


Figure 4.5: Training data Happy Vs Sad Vs None SVM



```
Train Loss: 0.6228691935539246
Train Accuracy: 0.6743220686912537


Test Loss: 0.6158742308616638
Test Accuracy: 0.685512363910675


>>> |
```

Figure 4.6: Neural Network Accuracy for Happy Vs Sad

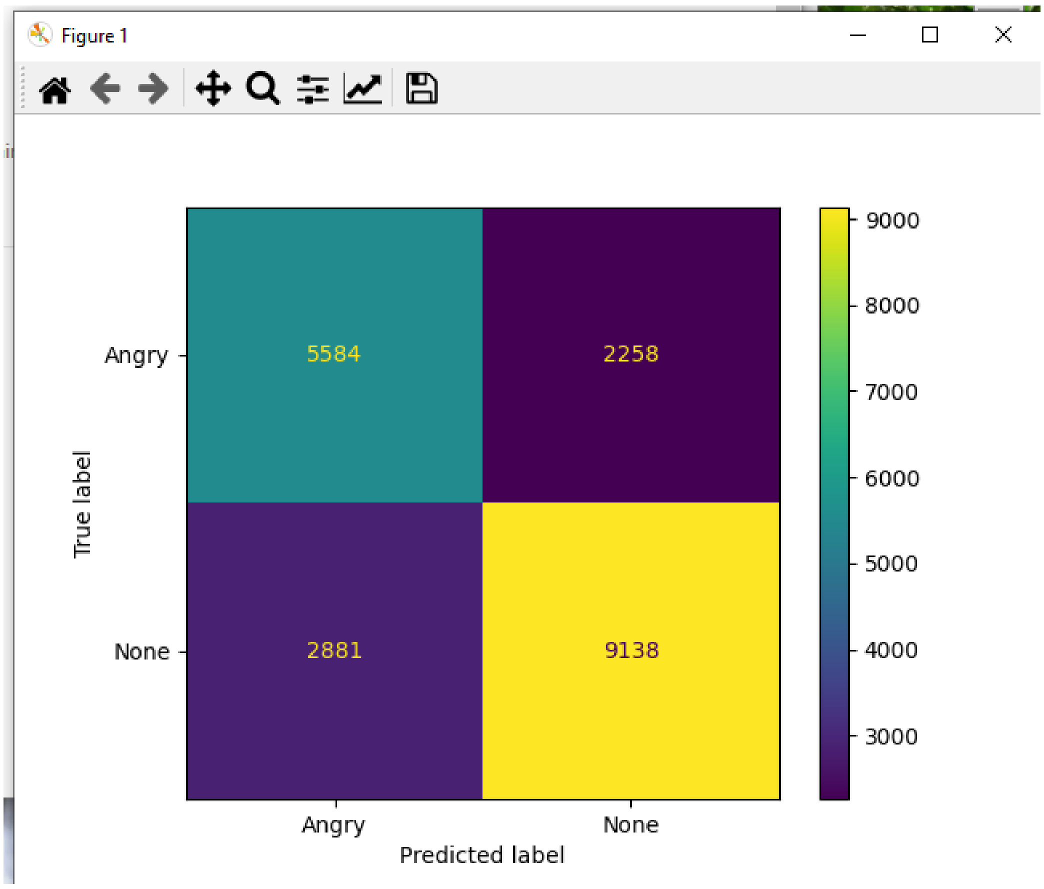


Figure 4.7: Training data Angry Vs None SVM

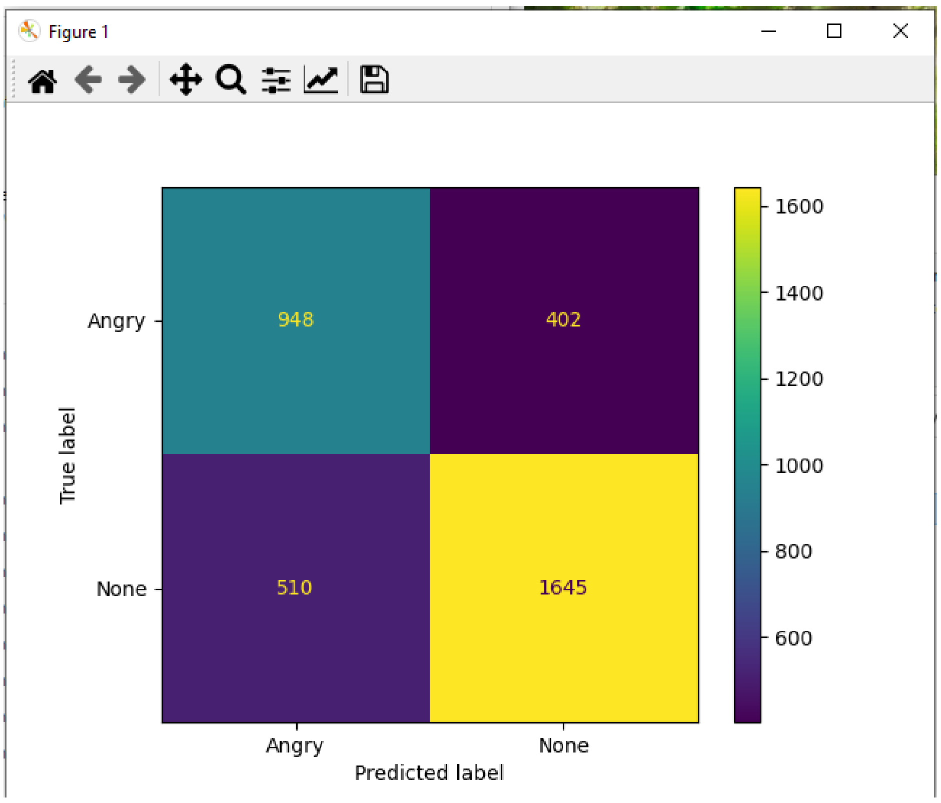
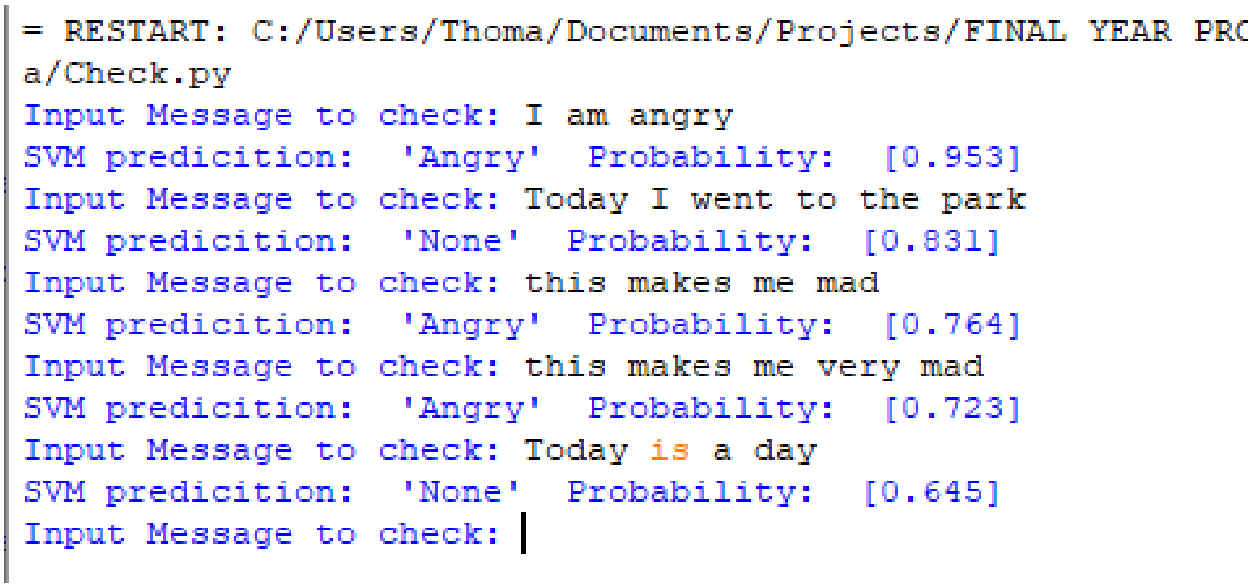
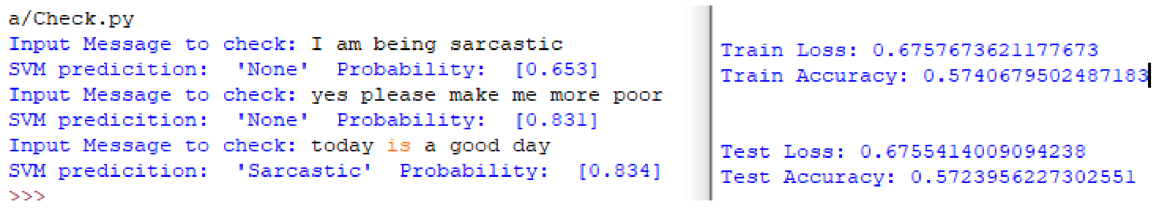


Figure 4.8: Testing data Angry Vs None SVM



```
= RESTART: C:/Users/Thoma/Documents/Projects/FINAL YEAR PRO
a/Check.py
Input Message to check: I am angry
SVM predicition:  'Angry'  Probability:  [0.953]
Input Message to check: Today I went to the park
SVM predicition:  'None'  Probability:  [0.831]
Input Message to check: this makes me mad
SVM predicition:  'Angry'  Probability:  [0.764]
Input Message to check: this makes me very mad
SVM predicition:  'Angry'  Probability:  [0.723]
Input Message to check: Today is a day
SVM predicition:  'None'  Probability:  [0.645]
Input Message to check: 
```

Figure 4.9: Demo of SVM Angry Vs None



```
a/Check.py
Input Message to check: I am being sarcastic
SVM predicition:  'None'  Probability:  [0.653]
Input Message to check: yes please make me more poor
SVM predicition:  'None'  Probability:  [0.831]
Input Message to check: today is a good day
SVM predicition:  'Sarcastic'  Probability:  [0.834]
>>>
```

```
Train Loss: 0.6757673621177673
Train Accuracy: 0.5740679502487183


Test Loss: 0.6755414009094238
Test Accuracy: 0.5723956227302551
```

Figure 4.10: Sarcasm best accuracy. Clearly no better than guessing.

generalize at all and made no sense. You can see this in figure 4.10, this was the same for all models and so was dropped.

4.1.2 Sequence models

Happy Vs Sad: Only happy vs sad was performed for sequencing models. LSTM out performed CNNS throughout every investigation. LSTM did not however reach the accuracy's of SVMs for happy vs Sad. The best models all had 3 layers and used the soft max activation function.

LSTM Confusion matrix for happy is shown in figure 11. As we can see it under-performs compared too the SVM achiveing a accuracy of only 80 percent compared to SVMs 90+.

LSTM Demo can be seen in figure 4.12. This shows that while LSTM is less confident with predictions, sequenced words are having the right effect on the outcome.

The Best CNN accurcies can be seen in figure 4.13.

Angry Vs None: LSTM performed outstandingly at this task and achieved higher accuracy's than SVM. The best structure was the same for Happy Vs Sad and Angry Vs None. The accuracy on training and testing can be seen in figure 4.14 and a demo in figure 4.15.

This would be the model I chose for deployment.

CNN also performed better than SVM on this task, but not as good as LSTM. The Best CNN accurcies can be seen in figure 4.16.

4.2 Summary

Both types of models had their strengths and weaknesses. The basic models were much better at having a strong choice with sentences that have had a word that was obvious one way or the other. Sequence models where much

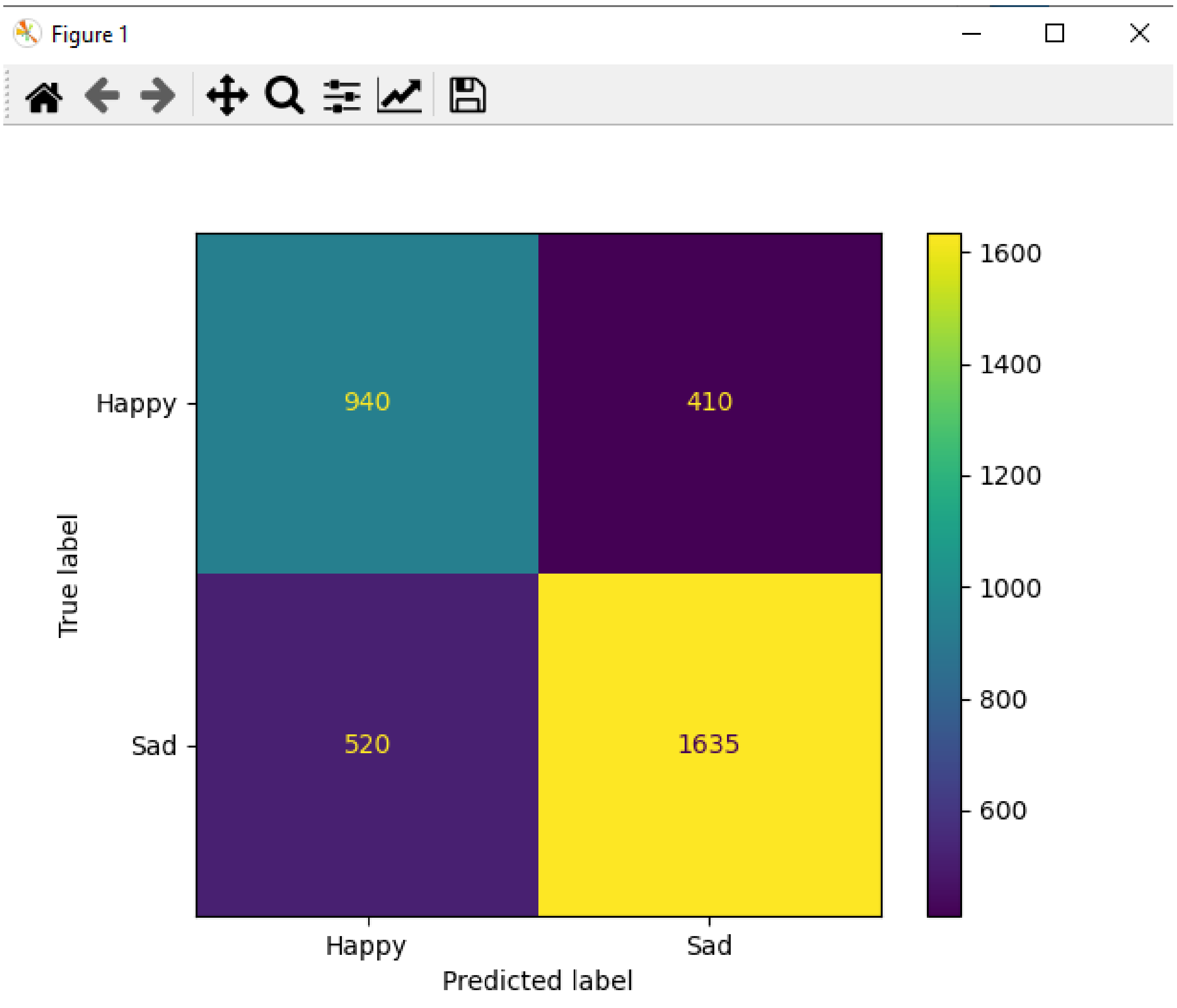
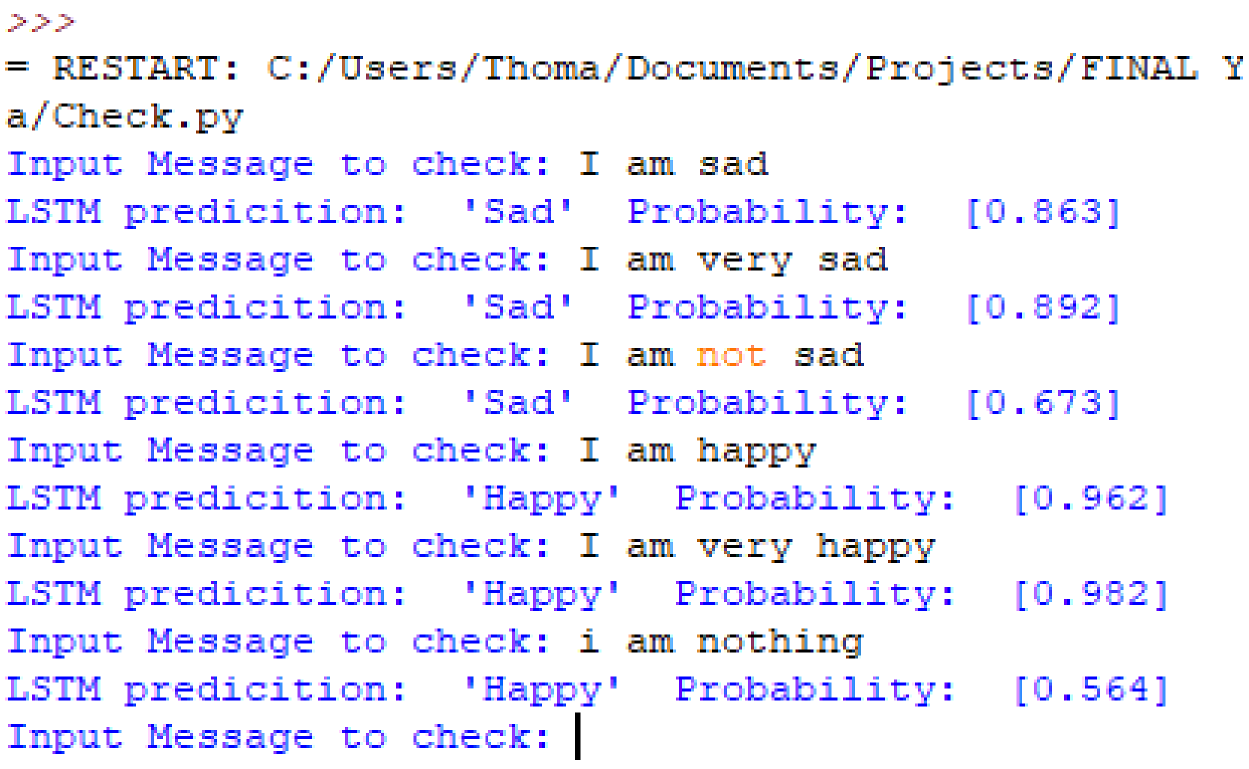
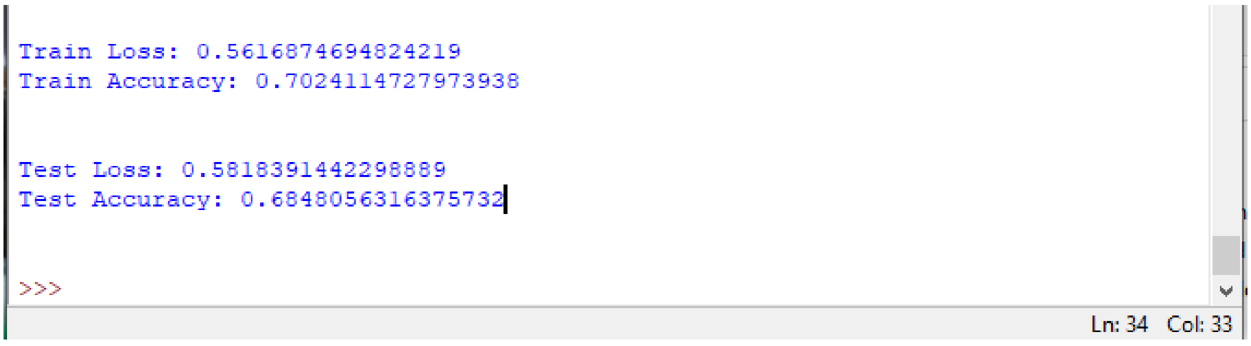


Figure 4.11: Confusion matrix LSTM happy vs sad



```
>>>
= RESTART: C:/Users/Thoma/Documents/Projects/FINAL Y
a/Check.py
Input Message to check: I am sad
LSTM predicition:  'Sad'  Probability:  [0.863]
Input Message to check: I am very sad
LSTM predicition:  'Sad'  Probability:  [0.892]
Input Message to check: I am not sad
LSTM predicition:  'Sad'  Probability:  [0.673]
Input Message to check: I am happy
LSTM predicition:  'Happy'  Probability:  [0.962]
Input Message to check: I am very happy
LSTM predicition:  'Happy'  Probability:  [0.982]
Input Message to check: i am nothing
LSTM predicition:  'Happy'  Probability:  [0.564]
Input Message to check:
```

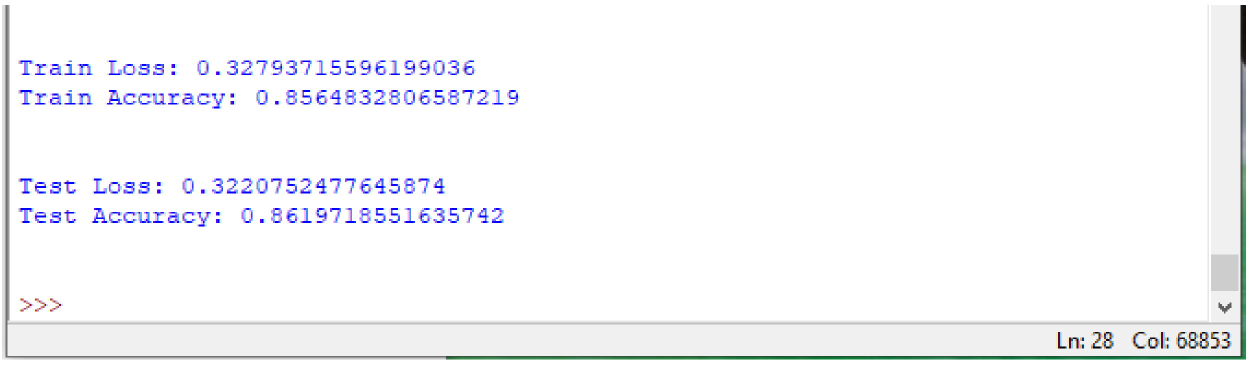
Figure 4.12: LSTM Happy Vs Sad Demo



```
Train Loss: 0.5616874694824219
Train Accuracy: 0.7024114727973938


Test Loss: 0.5818391442298889
Test Accuracy: 0.6848056316375732


>>>
```

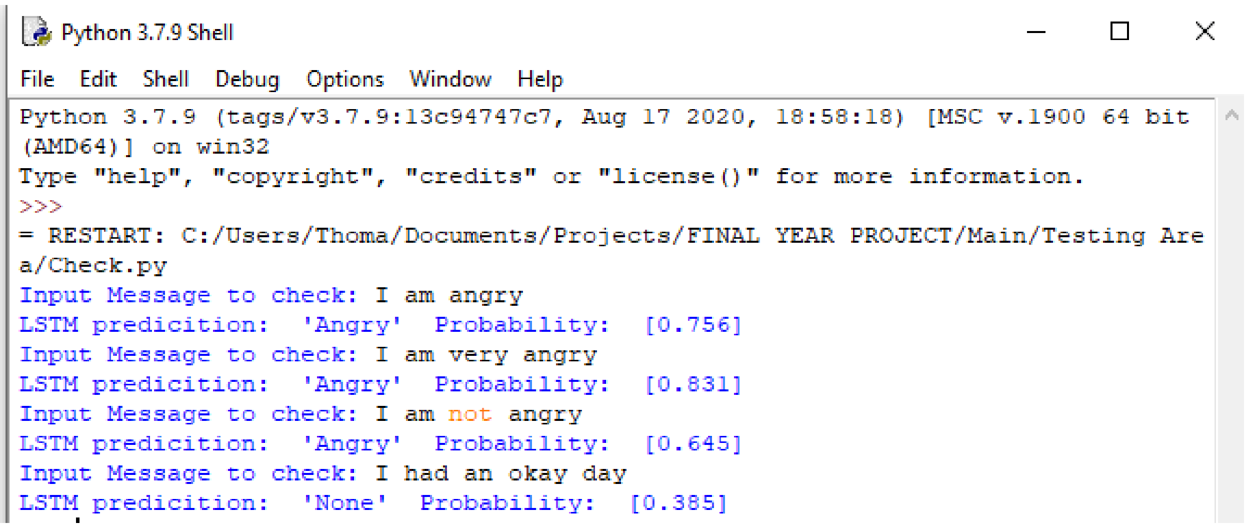
Ln: 34   Col: 33

Figure 4.13: CNN Happy Vs Sad accuracy



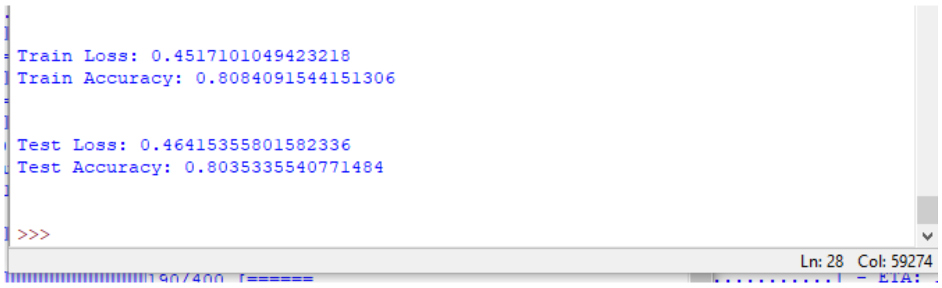
```
Train Loss: 0.32793715596199036
Train Accuracy: 0.8564832806587219


Test Loss: 0.3220752477645874
Test Accuracy: 0.8619718551635742


>>>
```

Figure 4.14: LSTM Angry Vs None accuracy



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 18:58:18) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/Thoma/Documents/Projects/FINAL YEAR PROJECT/Main/Testing Are
a/Check.py
Input Message to check: I am angry
LSTM predicition:  'Angry'  Probability:  [0.756]
Input Message to check: I am very angry
LSTM predicition:  'Angry'  Probability:  [0.831]
Input Message to check: I am not angry
LSTM predicition:  'Angry'  Probability:  [0.645]
Input Message to check: I had an okay day
LSTM predicition:  'None'  Probability:  [0.385]
```

Figure 4.15: LSTM Angry Vs None demo



```
Train Loss: 0.4517101049423218
Train Accuracy: 0.8084091544151306

Test Loss: 0.46415355801582336
Test Accuracy: 0.8035335540771484

>>>
```

Figure 4.16: CNN Angry Vs none accuracy

better with messages in the grey areas or with lots of influence words.

Sarcasm while getting some good accuracies did not perform well on any new data shown and so was left out for deployment.

Overall however, SVM came out with the highest accuracies, especially for happy vs sad so I will be using SVM for happy vs sad vs none in deployment and LSTM for angry Vs None as this had a stand out accuracy.

4.3 Deployment

Now that I had my best models trained and selected it was time to make the twitter bot to show off my models working. You can see the twitter bot code in appendix A.3.

It works by connecting to the twitter API every 5 minutes to check for new tweets with mentions of my twitter account. If it detects that a new tweet has been sent to the account, it will then download the tweets text along with the users ID.

My best models are then loaded in and fed the text after it has been cleaned and turned into a vector. The model that is the most confident in its prediction is used to assume the emotion. The user is then replied too, informing them of the emotion in the tweet. Demos for this can be seen in appendix C.

Chapter 5

Review

5.1 Assessment of findings

Overall basic models were much better at confidently classifying when an obvious statement was made such as “I am happy” but were much worse when text was less obvious between two emotions and could get confused. Basic models also could not handle statements with negatives or positives such as “not happy” or “very happy” while we see sequence models probabilities effected by these negatives and positives being present.

SVM not handling sequences and negatives/positives can be seen in appendix B.1 Figure B.7

Sequence models being affected by negatives and positives can be seen in figure 4.11.

Sequencing models were powerful but looking back required larger volumes of perfectly crafted data to be able to handle the obvious statements and making effector words (not,very) have more of an effect on the outcome.

The fact that linear kernel dominated was understandable as most of my data points where either on or off, 1 or 0 and transformations on this data would not change the shape much.

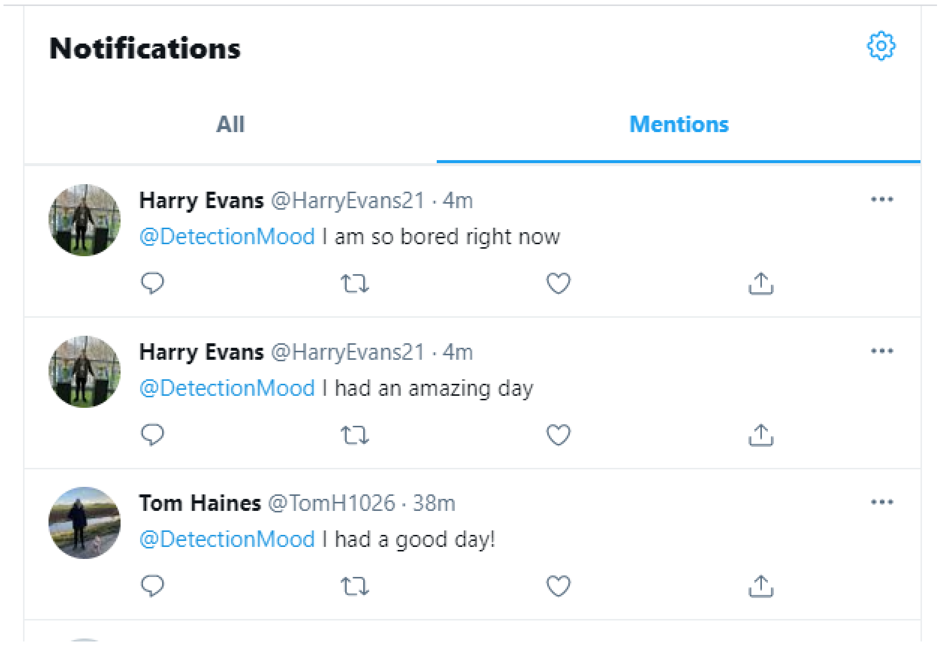
5.2 Deployment Review

Deployment worked fully as intended. A demo with actual people using the twitter bot can be seen in figure 5.1,5.2,5.3. Final deployment included Happy Vs Sad Vs None and Angry Vs None. All of these are identified well, and the bot can be run 24/7 to reply to people all the time. Further demos for the can be seen in appendix C.

5.3 What went well

Although a lot of time was spent cleaning the data, this really improved model accuracy and allowed me to investigate more complex models and understand how a machine can understand text data. The use of the time series format worked exceptionally well in the sequencing models and the cleaning of the text was much needed for bringing understanding to my data.

My large range of models allowed me to really investigate machine under-


Notifications

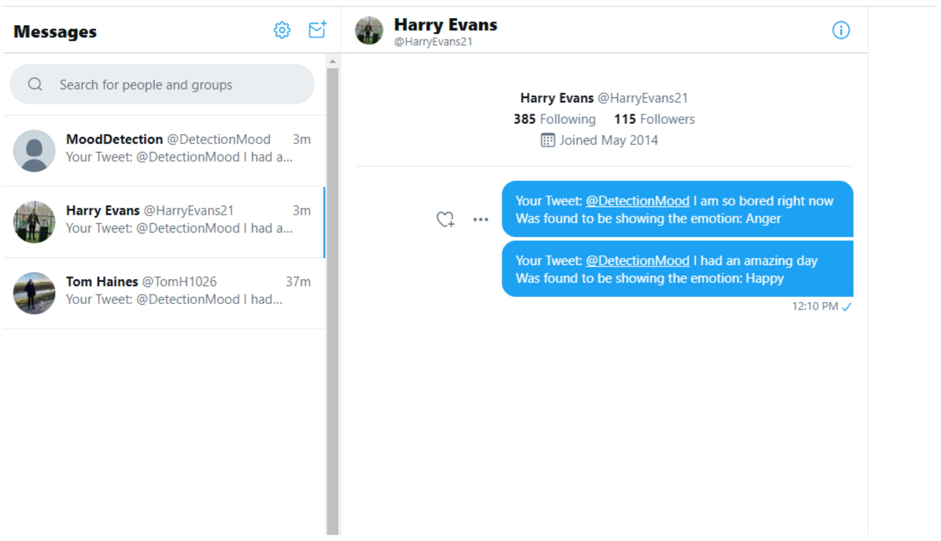
All Mentions

Harry Evans @HarryEvans21 · 4m
@DetectionMood I am so bored right now

Harry Evans @HarryEvans21 · 4m
@DetectionMood I had an amazing day

Tom Haines @TomH1026 · 38m
@DetectionMood I had a good day!

Figure 5.1: Messages tweeted at the bot


Messages

Search for people and groups

MoodDetection @DetectionMood 3m
Your Tweet: @DetectionMood I had a...

Harry Evans @HarryEvans21 3m
Your Tweet: @DetectionMood I had a...

Tom Haines @TomH1026 37m
Your Tweet: @DetectionMood I had...

Harry Evans
@HarryEvans21

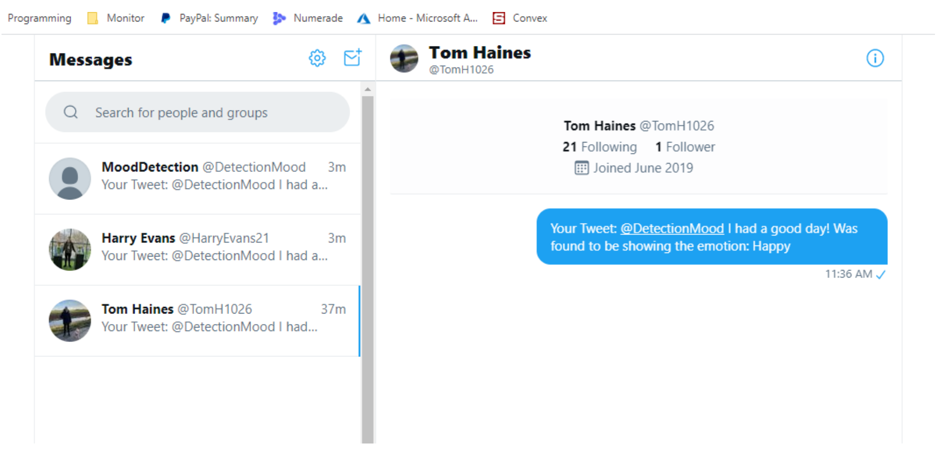
Harry Evans @HarryEvans21
385 Following 115 Followers
Joined May 2014

Your Tweet: @DetectionMood I am so bored right now
Was found to be showing the emotion: Anger

Your Tweet: @DetectionMood I had an amazing day
Was found to be showing the emotion: Happy

12:10 PM ✓

Figure 5.2: Messages tweeted back at user



MoodDetection @DetectionMood 3m
Your Tweet: @DetectionMood I had a...

Harry Evans @HarryEvans21 3m
Your Tweet: @DetectionMood I had a...

Tom Haines @TomH1026 37m
Your Tweet: @DetectionMood I had...

Tom Haines
@TomH1026

Tom Haines @TomH1026
21 Following 1 Follower
Joined June 2019

Your Tweet: @DetectionMood I had a good day! Was found to be showing the emotion: Happy
11:36 AM

Figure 5.3: Messages tweeted back at user

standing of text. Each model had its own strengths and weaknesses with basic models picking up on more obvious messages easier while sequencing models where better with the nuances.

Being able to deploy my models in the real world allowed me to understand how my models were actually working and put them to the test rather than just looking and training and testing accuracies. The bot worked incredibly well and has some obvious applications as well as being fun to make.

5.4 Problems and improvements

Given more time I would like to have investigated a few more emotions and paid more attention to moving from binary classification to multi class. Emotions could have included excitement and fear and would have made my bot more intelligent and more applicable to different areas. The use of multi class would also meant multiple models would not have to be used and compared on every piece of data.

I am not the first to investigate this problem or at least a similar problem. The use of a pretrained model (ie, taking their weight and structure as a starting point) that already identifies similar sentiment could have allowed for higher accuracies and less time spent training.

With a big application of these models being able to label large amounts of text data I would have liked to create a web app where people could drop in large amounts of text data to be labelled or to get relevant statistics on the emotions of the dataset.

A lot of the text data would have been very hard for a human to label and given more time I would like to have been more selective with what data I chose and used for a much deeper understand of NLP.

While data collection and cleaning were very key to this investigation it would be an understatement to say a good portion of my time on this project was spent doing that. In future I would have liked to leave more time for tuning and deployment.

SVM accuracy was high but this was mainly down to a few obvious words like good, great, and happy or bad, awful and sad being words that make the whole sentence emotion obvious. Given that I spent so much time on data collecting and training and given the previous statement decision trees may have been a good approach where I break down text and have the model look for certain trigger words that push it one way or the other and then identify if there is a negative or a positive effector before the trigger word as most sentences have one trigger word sometimes two so only a small tree would need to be constructed.

I would like to have investigated the use of emojis however the error with importing them and the fact that different operating systems and keyboard applications used different codes to represent emojis made this near impossible.

Although the range of models was good. Further exploration into into other techniques such as already mentioned decision trees and transformers could have been powerful tools or analysing the text also, Transformers in particular are known for being powerful when it comes to text classification [15].

Bibliography

[1] Geron, A. 2019. Hands on machine learning with scikit-Learn, Keras, and Tensorflow. 2nd Edition. Canada: O'Reilly Media.

[2] Ng, A. 2013. Machine Learning Seen at: https://www.coursera.org/learn/machine-learning

[3] Chollet, F . 2018. Deep learning with python. New York: Manning Publications

[4] Russell, S and Norvig, P. 2016 Artificial intelligence: a modern approach. 3rd Edition. Harlow: Pearson Education Limited

[5] Emment, T. 2021. Personal Additional referenced files. Seen at: https://www.dropbox.com/sh/394vdxvo52ap45h/AABY6TsqTyvYc3zfd4cRodgMa?dl=0

[6] IQBAL, M. 2020. Twitter Revenue and Usage Statistics. Seen at: https://www.businessofapps.com/data/twitter-statistics/

[7] Güntekin, B . 2018. Understanding Another Person's Emotions. Seen at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6135884/

[8] Harfoushi, O, Hasan, D, Obiedat, R. 2018. Sentiment Analysis Algorithms through Azure Machine Learning. ,pp. 1-10

[9] Tweepy Documentation Article. 2020. API Reference. Seen at: https://docs.tweepy.org/en/latest/api.html

[10] Tensorflow Documentation Article. 2021. Training and evaluation with the built-in methods. Seen at: https://www.Tensorflow.org/guide/keras/train$_and_evaluate$

[11] Tensorflow Documentation Article. 2021. Text classification with an RNN. Seen at: https://www.Tensorflow.org/tutorials/text/text$_classification_rnn$

[12] Tensorflow Documentation Article. 2021. Convolutional Neural Network (CNN). Seen at: https://www.Tensorflow.org/tutorials/images/cnn

[13] Kunchhal, R. 2020. The Mathematics Behind Support Vector Machine Algorithm Seen at: https://www.analyticsvidhya.com/blog/2020/10/the-mathematics-behind-svm/

[14] hefinioanrhys. 2019. Support Vector Machines Seen at: https://www.r-bloggers.com/2019/10/support-vector-machines-with-the-mlr-package/

[15] Rajapakse, T. 2019. A Hands-On Guide To Text Classification With Transformer Models Seen at: https://towardsdatascience.com/https-medium-com- chaturangarajapakshe-text-classification-with-transformer-models-d370944b50ca

Appendix A

Python Code

A.1 Data gathering and cleaning

A.1.1 Tweepy data gathering code

```
consumer_key = ’X4GdKJsPC5FwUypV4R4yQWlWk’
consumer_secret = ’zF9m8Sra9lDnXC7iNkLG8AzT9ZlN7OblRqR64GdSuiPFiOg4TE’
access_token = ’1318227698762874884-FPwVWzuBVEPZVRujJOPQBoYPygWlt2’
access_token_secret = ’8ERA3WilxXFPKOTtheA66Y54OPmah97klkuKg1rzXmKCk’

auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
api = tweepy.API(auth,wait_on_rate_limit=True)

csvFile = open(’All.csv’, ’a’)

csvWriter = csv.writer(csvFile)
SearchTerms = ["hello","happy","sad","the","sarcasm","holiday"
,"today","yesterday","feel","no"]
for x in SearchTerms:
    for tweet in tweepy.Cursor(api.search,q=x,count=100,lang=
    "en").items():
        print(tweet.text)
        try:
            csvWriter.writerow([tweet.text])
        except:
            print("Error collecting data line")


csvFile.close()
```

A.1.2 data labeling

```
run = True

csvFile1 = open('Catagory.csv', 'a', newline='')
csvWriter = csv.writer(csvFile1)


with open('AllClean.csv', newline='') as csvfile:

    Data = csv.reader(csvfile)

    Tweets = []
    for Row in Data:
        Tweets.append(Row[0])

pygame.init()
pygame.display.set_caption("Evolution")
win = pygame.display.set_mode((1400,800))
Mediumfont = pygame.font.SysFont('Calibri', 30, True, False)
WHITE = (255,255,255)
BLACK = (0,0,0)
GREY = (192,192,192)


##Edit this after closing
Number = 86
#Start Position
```

```
while run:
    csvFile1 = open('Catagory.csv', 'a', newline='')
    csvWriter = csv.writer(csvFile1)
    win.fill(WHITE)
    pygame.time.delay(100)
    for event in pygame.event.get():
        MousePos = pygame.mouse.get_pos()
        if event.type == pygame.QUIT:
            print(Number)
            pygame.quit()
            sys.exit()
        if event.type == pygame.MOUSEBUTTONDOWN:
            if (MousePos[0] in range (30,200)) and (MousePos[1] in
            range (480,530)):
                csvWriter.writerow(["Happy",Tweets[Number]])
                Number = Number + 1
            if (MousePos[0] in range (30,200)) and (MousePos[1] in
            range (420,470)):
                csvWriter.writerow(["None",Tweets[Number]])
                Number = Number + 1
            if (MousePos[0] in range (250,420)) and (MousePos[1] in
            range (480,530)):
                Number = Number + 1
            if (MousePos[0] in range (30,200)) and (MousePos[1] in
            range (540,590)):
                csvWriter.writerow(["Sad",Tweets[Number]])
                Number = Number + 1
            if (MousePos[0] in range (30,200)) and (MousePos[1] in
            range (600,650)):
                csvWriter.writerow(["Angry",Tweets[Number]])
                Number = Number + 1

            if (MousePos[0] in range (30,200)) and (MousePos[1] in
            range (720,770)):
                csvWriter.writerow(["Sarcastic",Tweets[Number]])
                Number = Number + 1
    pygame.draw.polygon(win, BLACK, [(0,400),(1400,400),(1400,800)
    ,(0,800)])
    pygame.draw.polygon(win, GREY, [(0,400),(1400,400),(1400,800)
    ,(0,800)],20)
```

```
pygame.draw.polygon(win, GREY, [(30,480),(200,480),(200,530)
,(30,530)])
text = Mediumfont.render("Happy", True, BLACK)
win.blit(text,(35,485))

pygame.draw.polygon(win, GREY, [(250,480),(420,480),(420,530)
,(250,530)])
text = Mediumfont.render("Exclude", True, BLACK)
win.blit(text,(255,485))

pygame.draw.polygon(win, GREY, [(30,420),(200,420),(200,470)
,(30,470)])
text = Mediumfont.render("None", True, BLACK)
win.blit(text,(35,425))

pygame.draw.polygon(win, GREY, [(30,540),(200,540),(200,590)
,(30,590)])
text = Mediumfont.render("Sad", True, BLACK)
win.blit(text,(35,545))

pygame.draw.polygon(win, GREY, [(30,600),(200,600),(200,650)
,(30,650)])
text = Mediumfont.render("Angry", True, BLACK)
win.blit(text,(35,605))

pygame.draw.polygon(win, GREY, [(30,720),(200,720),(200,770)
,(30,770)])
text = Mediumfont.render("Sarcastic", True, BLACK)
win.blit(text,(35,725))

if len(Tweets[Number]) > 100:
    Runs = 1
    Sentance = Tweets[Number][0:100]
    while len(Tweets[Number]) > (100*Runs):
        Sentance = Tweets[Number][(100*(Runs-1)):100*Runs:]
        text = Mediumfont.render(Sentance, True, BLACK)
        win.blit(text,(50,30+(35*(Runs-1))))
        Runs = Runs +1
    Sentance = Tweets[Number][(100*(Runs-1))::]
    text = Mediumfont.render(Sentance, True, BLACK)
    win.blit(text,(50,30+(35*(Runs-1))))
else:
    text = Mediumfont.render(Tweets[Number], True, BLACK)
    win.blit(text,(50,30))
pygame.display.update()
csvfile.close()
```

A.1.3 Data Cleaning, removal of hyperlinks, @s and punctuation

```
csvFile1 = open('Final.csv', 'a', newline='')
csvWriter = csv.writer(csvFile1)


with open('Catagory.csv', newline='') as csvfile:

    Data = csv.reader(csvfile)
    for row in Data:  # add predict for miss spelt
        Position = 0
        Sentance = row[1]

        Max = len(Sentance)
        while Position < Max:

            if Sentance[Position] == "@":
                Temp = Position
                while Sentance[Temp] != " " and Temp+1 < Max:
                    Temp = Temp +1
                Sentance = Sentance[0:Position:]+Sentance[Temp+1::]
                Max = len(Sentance)
            if Position + 4 < Max:
                if Sentance[Position] == "h" and Sentance[Position+1]
                == "t" and Sentance[Position+2] == "t" and Sentance[Position+3]
                "p" and Sentance[Position+4] == "s":
                    Temp = Position
                    while Sentance[Temp] != " " and Temp+1 < Max:
                        Temp = Temp +1
                    Sentance = Sentance[0:Position:]+Sentance[Temp+1::]
                    Max = len(Sentance)
            if Position < Max:
                if Sentance[Position] in ["#","?",".","!",",",":",";"]:
                    Sentance = Sentance[0:Position:]+Sentance[Position+2::]
                    Max = len(Sentance)

            Position = Position +1
        csvWriter.writerow([row[0],Sentance])
    csvfile.close()
```

A.1.4 Data Cleaning, removing non words and guessing of miss-spellings

```
import Enchant
d = Enchant.Dict("en_GB")
import csv
from nltk.tokenize import word_tokenize


csvFile1 = open('Master1.csv', 'a', newline='')
csvWriter = csv.writer(csvFile1)


with open('Master.csv', newline='') as csvfile:
    Position = 0
    Data = csv.reader(csvfile)
    for row in Data:
        english_words = []
        Text = word_tokenize(row[1])
        for word in Text:
            if d.check(word):
                english_words.append(word)
            else:
                english_words.append(d.predict(word)[0])
        Position = Position +1
        csvWriter.writerow([row[0], " ".join(english_words)])
    csvfile.close()
```

A.2 Model preparation and training

A.2.1 Sk-Learn preperation

```
dataframe = pd.read_csv("AngryVsNoneAll.csv")


y = dataframe["Label"]
x = dataframe["Text"]

x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.15, random_state=23)

cv = CountVectorizer()

features = cv.fit_transform(x_train.astype('U').values)
features_test = cv.transform(x_test.astype('U').values)
```

A.2.2 SVM Training code

```
model = svm.SVC(kernel = 'rbf',probability=True, C = 0.01)
model.fit(features,y_train)
print("Accuracy for radial basis function Kernel (0.01): ",
model.score(features_test,y_test))

print("Accuracy on training radial Kernel (0.01): ",
model.score(features,y_train))

joblib.dump(model,"SVMAngryVsNoneAllRFB0.01.pkl")
```

A.2.3 Creating time series data for Neural Net

```
import os

cwd = os.getcwd()


ListWord = ["NANA"]

for filename in os.listdir(cwd + "/Angry"):
    Final = cwd + "/Angry/" + filename
    f = open(Final, "r")
    try:
        Text = f.read()
        Sentance = Text.split(" ")
        for word in Sentance:
            if word in ListWord:
                Yes = True

            else:
                ListWord.append(word)

    except:
        print("Encoding error")



for filename in os.listdir(cwd + "/None"):
    Final = cwd + "/None/" + filename
    f = open(Final, "r")
    try:
        Text = f.read()
        Sentance = Text.split(" ")
        for word in Sentance:
            if word in ListWord:
                Yes = True

            else:
                ListWord.append(word)
    except:
        print("Encoding error")


for filename in os.listdir(cwd + "/Angry"):
    Final = cwd + "/Angry/" + filename
    f = open(Final, "r")
    Vector = ["0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0
    Position = 1

    Text = f.read()
    Sentance = Text.split(" ")
    TooLong = False
    for word in Sentance:
        Index = ListWord.index(word)
        Vector[Position] = str(Index)
        Position = Position + 1
        if Position > 58:
            Position = 1
            TooLong = True
    if not TooLong:
        New = cwd + "/AngrysVector/" + filename
        file = open(New, "w")
        file.write(", ".join(Vector))
        file.close()



for filename in os.listdir(cwd + "/None"):
    Final = cwd + "/None/" + filename
    f = open(Final, "r")
    Vector = ["0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0
    Position = 1
    Text = f.read()
    Sentance = Text.split(" ")
    TooLong = False
    for word in Sentance:
        Index = ListWord.index(word)
        Vector[Position] = str(Index)
        Position = Position + 1
        if Position > 58:
            Position = 1
            TooLong = True
    if not TooLong:

        New = cwd + "/NoneVector/" + filename
        file = open(New, "w")
        file.write(", ".join(Vector))
        file.close()

import csv


csvFile1 = open('HappyVsSadVsNoneVectorNEW.csv', 'a', newline='')
csvWriter = csv.writer(csvFile1)

List = []

with open('AngryVsNoneAll.csv', newline='') as csvfile:
    Data = csv.reader(csvfile)
    for row in Data:
        Sentance = row[1].split(", ")
        csvWriter.writerow([row[0],int(Sentance[0]),
        int(Sentance[1]),int(Sentance[2]),int(Sentance[3]),int(Sentance[4]),
        int(Sentance[5]),int(Sentance[6]),int(Sentance[7]),int(Sentance[8]),int(
                            int(Sentance[10]),
                            int(Sentance[11]),
                            int(Sentance[12]),int(Sentance[13]),int(Sentance[14]
                            int(Sentance[15]),int(Sentance[16]),int(Sentance[17]
                            int(Sentance[45]),int(Sentance[19]),
                            int(Sentance[20]),
                            int(Sentance[21]),int(Sentance[22]),int(Sentance[23]
                            int(Sentance[24]),int(Sentance[25]),
                            int(Sentance[26]),int(Sentance[27]),int(Sentance[46]
                            int(Sentance[29]),
                            int(Sentance[30]),
                            int(Sentance[31]),int(Sentance[32]),int(Sentance[33]
                            int(Sentance[34]),int(Sentance[35]),int(Sentance[36]
                            int(Sentance[37]),int(Sentance[47]),
                            int(Sentance[39]),
                            int(Sentance[40]),int(Sentance[41]),int(Sentance[42]
                            int(Sentance[43]),int(Sentance[44]),int(Sentance[45]
                            int(Sentance[46]),int(Sentance[47]),int(Sentance[48]
                            int(Sentance[49]),
                            int(Sentance[50]),int(Sentance[51]),int(Sentance[52]
                            int(Sentance[53]),int(Sentance[54]),int(Sentance[55]
                            int(Sentance[56]),int(Sentance[57]),int(Sentance[58]
                            nt(Sentance[59])
                            ])

    csvfile.close()
```

A.2.4 Neural Net training code

```
batch_size = 200
seed = 42



dataset = pd.read_csv("AngryVsNoneVectorNEW.csv")

dataset.head()


train=dataset.sample(frac=0.8,random_state=seed) #random state is a seed value
test=dataset.drop(train.index).sample(frac=1.0)


train_dataset_x = train.copy()
test_dataset_x = test.copy()

train_dataset_y = train_dataset_x.pop('Label')
train_dataset_y = to_categorical(train_dataset_y)
train_dataset_x = np.array(train_dataset_x)

test_dataset_y = test_dataset_x.pop('Label')
test_dataset_y = to_categorical(test_dataset_y)
```

```
test_dataset_x = np.array(test_dataset_x)





##SOFT MAX IMPORTANT

#basic neural net

modelNeuralNet = models.Sequential()
modelNeuralNet.add(layers.Dense(128, activation='softmax'))
modelNeuralNet.add(layers.Dense(2, activation='softmax'))


modelNeuralNet.compile(
      optimizer='rmsprop',
      loss='binary_crossentropy',
      metrics=['accuracy'])




modelNeuralNet.fit(train_dataset_x, train_dataset_y,
epochs=10,validation_data=(test_dataset_x, test_dataset_y), )

modelNeuralNet.save("SimpleNeuralNetModelAngryNone.tf")

test_loss, test_acc = modelNeuralNet.evaluate(test_dataset_x,test_dataset_y)




train_loss, train_acc = modelNeuralNet.evaluate(train_dataset_x, train_dataset_y
print("\n")
print('Train Loss: {}'.format(train_loss))
print('Train Accuracy: {}'.format(train_acc))
print("\n")
print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
print("\n")
```

A.2.5 Sequence model layers training code

```
#1d
modelNeuralNet = models.Sequential()
intput_shape=(train_dataset_x.shape[1], 1)
modelNeuralNet.add(layers.Conv1D(128, kernel_size=3,activation="relu",
input_shape=(60,1)))
modelNeuralNet.add(layers.Dense(64, activation='softmax'))
modelNeuralNet.add(layers.Dense(64, activation='softmax'))
modelNeuralNet.add(layers.Dense(1, activation='sigmoid'))
#LsTM
modelNeuralNet = models.Sequential()
intput_shape=(train_dataset_x.shape[1], 1)
modelNeuralNet.add(layers.Bidirectional(tf.keras.layers.LSTM(64,
return_sequences=True)))
modelNeuralNet.add(layers.Bidirectional(tf.keras.layers.LSTM(32)))
modelNeuralNet.add(layers.Dense(64, activation='softmax'))
modelNeuralNet.add(layers.Dense(1, activation='sigmoid'))
```

A.3 Twitter bot demo code

```
dataframe = pd.read_csv("ALL.csv")

stop_words = set(stopwords.words('english'))
####input your credentials here
consumer_key = 'NGeDyTOHtMgKMZBkJuC9icsKj'
consumer_secret = 'IYqpeBiUR4tHhp94Zmw9CF9nBAlwBUKThbIOfclSS88JkPOb9w'
access_token = '1318227698762874884-xRjxzFz3yaLRzgMsy25sOmOpBvtU5n'
access_token_secret = '6PtBMQ9Yc9sKyoOzrWnuCNfm4MPwysyOT1yyEtMBzxoLR'

auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
api = tweepy.API(auth,wait_on_rate_limit=True)
cwd = os.getcwd()
```

```
ListWord = ["NANA"]
modelNeuralNet = keras.models.load_model("LSTMeuralNetModelAngryVsNone.tf")

for filename in os.listdir(cwd + "/Angry"):
    Final = cwd + "/Angry/" + filename
    f = open(Final, "r")
    try:
        Text = f.read()
        Sentance = Text.split(" ")
        for word in Sentance:
            if word in ListWord:
                Yes = True

            else:
                ListWord.append(word)

    except:
        print("Encoding error")



for filename in os.listdir(cwd + "/None"):
    Final = cwd + "/None/" + filename
    f = open(Final, "r")
    try:
        Text = f.read()
        Sentance = Text.split(" ")
        for word in Sentance:
            if word in ListWord:
                Yes = True

            else:
                ListWord.append(word)

    except:
        print("Encoding error")

y = dataframe["Label"]
x = dataframe["Text"]
```

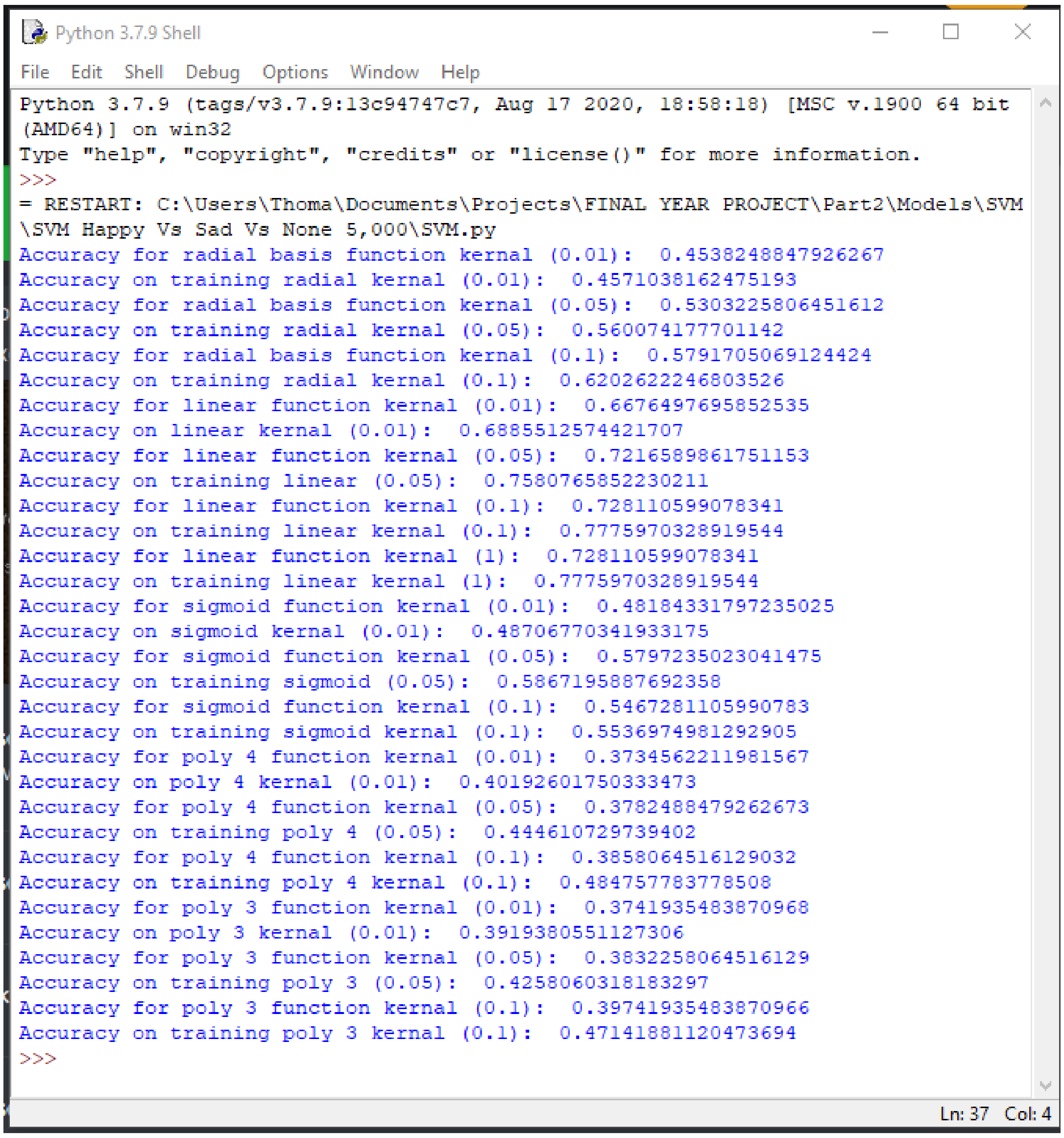
```
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.15, random_state=23)

cv = CountVectorizer()

features = cv.fit_transform(x_train.astype('U').values)
features_test = cv.transform(x_test.astype('U').values)
text = ""

modelSVM = joblib.load("SVMHappyVsSadVsNonelinear0.1.pkl")

SeenTweets = []
with open('Seen.txt', 'r') as fp:
    line = fp.readline()
    while line:
        SeenTweets.append(line[:-1])
        line = fp.readline()
while True:
    Alltweet = api.mentions_timeline(1)
    fp = open('Seen.txt', 'a')
    for Message in Alltweet:
        tweet = Message.text
        if Message.text not in SeenTweets:
            print(Message.text)
            Sentance = tweet.split(" ")
            Sentance.remove("@DetectionMood")
            fp.write(Message.text+"\n")
            Vector = [0] * 60
            Position = 1
            for word in Sentance:
                try:
                    Index = ListWord.index(word)
                    Vector[Position] = Index
                    Position = Position + 1
                except:
                    Temp = True
            Sentance = ", ".join(Sentance)
            Final = cv.transform([Sentance])
            print("SVM predicition: ", modelSVM.predict(Final),
            "Probability:",modelSVM.predict_proba(Final))
            Vector = np.array([Vector])
            Vector = Vector.reshape(Vector.shape[0],Vector.shape[1], 1)
            Vector = Vector.astype(np.float32)
            predictions = modelNeuralNet.predict(Vector)
            print("Neural Net Predicts:                  ", predictions)
            HappyVsSad = modelSVM.predict_proba(Final)
            AngryVsNone = predictions
            if HappyVsSad[0][0] > 0.65:
                Reply = "Your Tweet: " + Message.text +
                " Was found to be showing the emotion: Happy"
                api.send_direct_message(Message.user.id, Reply)
                print("Tweeting back Happy")
            elif HappyVsSad[0][2] > 0.65:
                Reply = "Your Tweet: " + Message.text +
                " Was found to be showing the emotion: Sad"
                api.send_direct_message(Message.user.id, Reply)
                print("Tweeting back Sad")
            elif HappyVsSad[0][1] > 0.60:
                Reply = "Your Tweet: " + Message.text +
                " Was found to be showing no emotion"
                api.send_direct_message(Message.user.id, Reply)
                print("Tweeting back None")
            else:
                if AngryVsNone[0][0] > 0.7 or AngryVsNone[0][0] < 0.3:
                    Reply = "Your Tweet: " + Message.text +
                    " Was found to be showing the emotion: Anger"
                    api.send_direct_message(Message.user.id, Reply)
                    print("Tweeting back Anger")
    time.sleep(300)
    fp.close
```

Appendix B

Training examples outputs

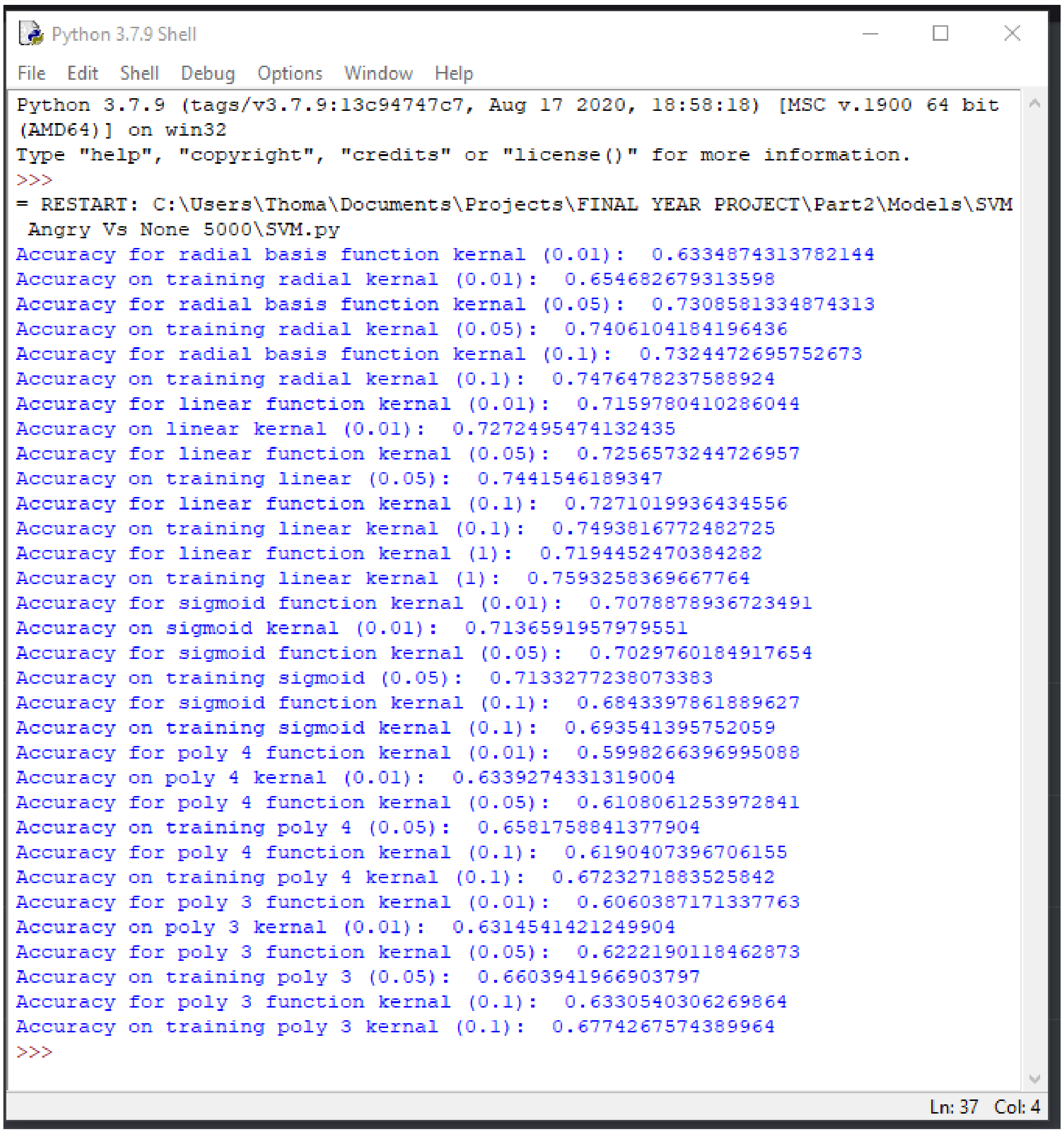
B.1 SVM

B.2 Neural Net and sequence models



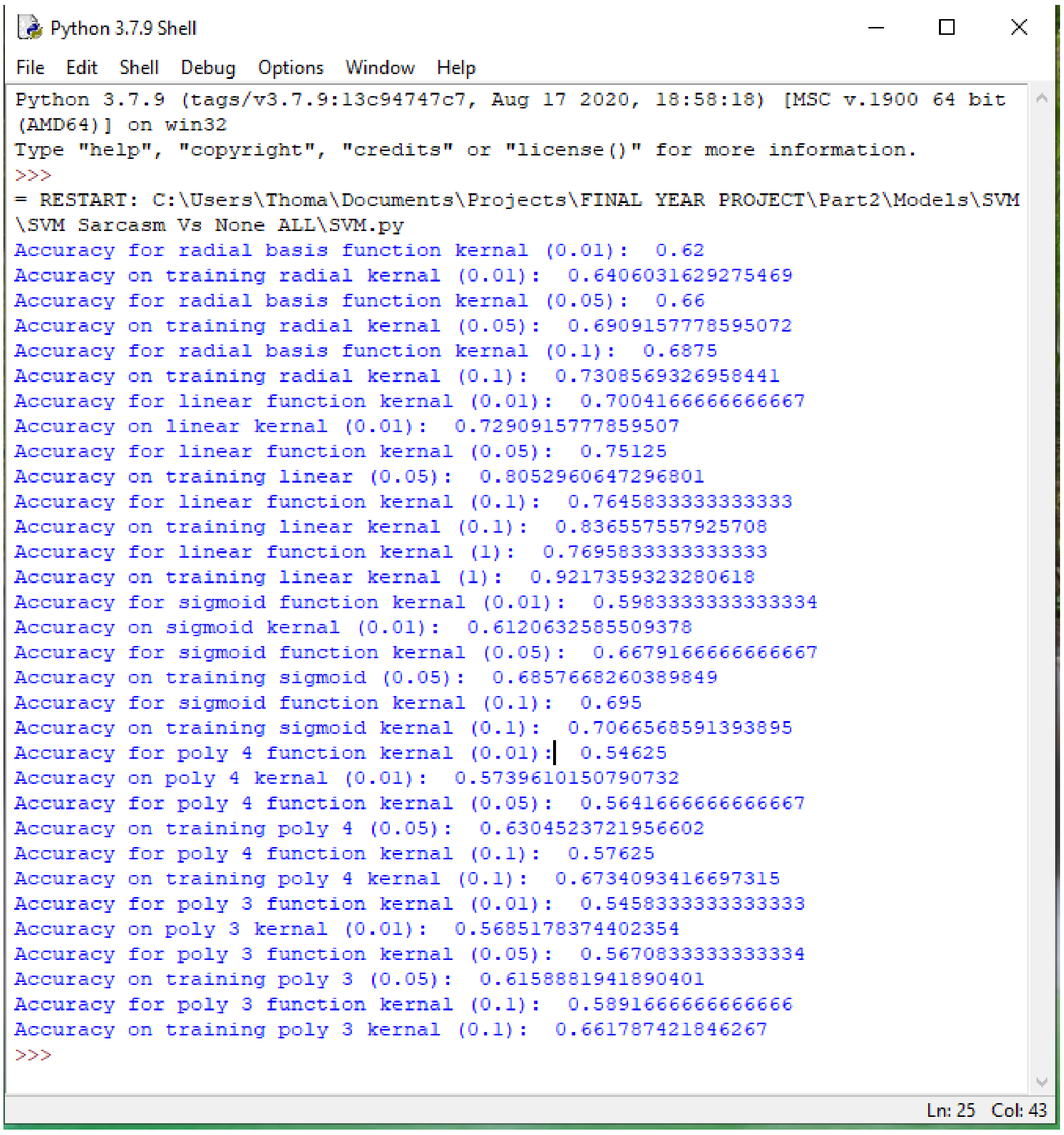
```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 18:58:18) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Thoma\Documents\Projects\FINAL YEAR PROJECT\Part2\Models\SVM
\SVM Happy Vs Sad Vs None 5,000\SVM.py
Accuracy for radial basis function kernal (0.01):  0.4538248847926267
Accuracy on training radial kernal (0.01):  0.4571038162475193
Accuracy for radial basis function kernal (0.05):  0.5303225806451612
Accuracy on training radial kernal (0.05):  0.560074177701142
Accuracy for radial basis function kernal (0.1):  0.5791705069124424
Accuracy on training radial kernal (0.1):  0.6202622246803526
Accuracy for linear function kernal (0.01):  0.6676497695852535
Accuracy on linear kernal (0.01):  0.6885512574421707
Accuracy for linear function kernal (0.05):  0.7216589861751153
Accuracy on training linear (0.05):  0.7580765852230211
Accuracy for linear function kernal (0.1):  0.728110599078341
Accuracy on training linear kernal (0.1):  0.7775970328919544
Accuracy for linear function kernal (1):  0.728110599078341
Accuracy on training linear kernal (1):  0.7775970328919544
Accuracy for sigmoid function kernal (0.01):  0.48184331797235025
Accuracy on sigmoid kernal (0.01):  0.48706770341933175
Accuracy for sigmoid function kernal (0.05):  0.5797235023041475
Accuracy on training sigmoid (0.05):  0.5867195887692358
Accuracy for sigmoid function kernal (0.1):  0.5467281105990783
Accuracy on training sigmoid kernal (0.1):  0.5536974981292905
Accuracy for poly 4 function kernal (0.01):  0.3734562211981567
Accuracy on poly 4 kernal (0.01):  0.40192601750333473
Accuracy for poly 4 function kernal (0.05):  0.3782488479262673
Accuracy on training poly 4 (0.05):  0.444610729739402
Accuracy for poly 4 function kernal (0.1):  0.3858064516129032
Accuracy on training poly 4 kernal (0.1):  0.484757783778508
Accuracy for poly 3 function kernal (0.01):  0.3741935483870968
Accuracy on poly 3 kernal (0.01):  0.3919380551127306
Accuracy for poly 3 function kernal (0.05):  0.3832258064516129
Accuracy on training poly 3 (0.05):  0.4258060318183297
Accuracy for poly 3 function kernal (0.1):  0.39741935483870966
Accuracy on training poly 3 kernal (0.1):  0.47141881120473694
>>>
```

Figure B.1: SVM Happy Vs Sad Vs None



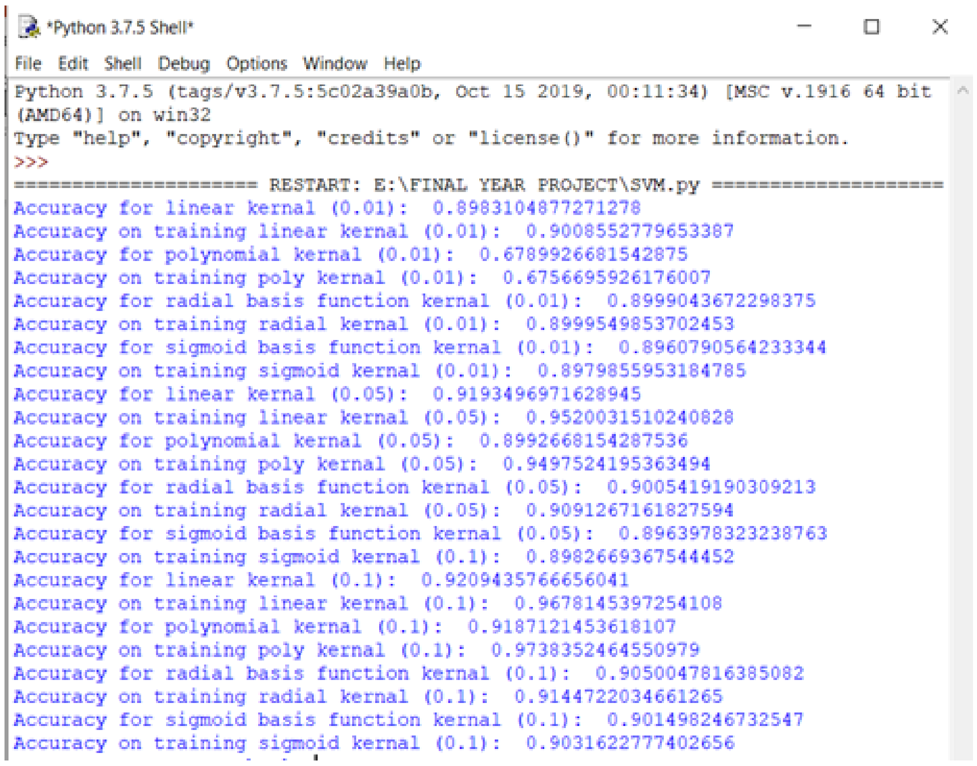
```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 18:58:18) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Thoma\Documents\Projects\FINAL YEAR PROJECT\Part2\Models\SVM
 Angry Vs None 5000\SVM.py
Accuracy for radial basis function kernal (0.01):  0.6334874313782144
Accuracy on training radial kernal (0.01):  0.654682679313598
Accuracy for radial basis function kernal (0.05):  0.7308581334874313
Accuracy on training radial kernal (0.05):  0.7406104184196436
Accuracy for radial basis function kernal (0.1):  0.7324472695752673
Accuracy on training radial kernal (0.1):  0.7476478237588924
Accuracy for linear function kernal (0.01):  0.7159780410286044
Accuracy on linear kernal (0.01):  0.7272495474132435
Accuracy for linear function kernal (0.05):  0.7256573244726957
Accuracy on training linear (0.05):  0.7441546189347
Accuracy for linear function kernal (0.1):  0.7271019936434556
Accuracy on training linear kernal (0.1):  0.7493816772482725
Accuracy for linear function kernal (1):  0.7194452470384282
Accuracy on training linear kernal (1):  0.7593258369667764
Accuracy for sigmoid function kernal (0.01):  0.7078878936723491
Accuracy on sigmoid kernal (0.01):  0.7136591957979551
Accuracy for sigmoid function kernal (0.05):  0.7029760184917654
Accuracy on training sigmoid (0.05):  0.7133277238073383
Accuracy for sigmoid function kernal (0.1):  0.6843397861889627
Accuracy on training sigmoid kernal (0.1):  0.693541395752059
Accuracy for poly 4 function kernal (0.01):  0.5998266396995088
Accuracy on poly 4 kernal (0.01):  0.6339274331319004
Accuracy for poly 4 function kernal (0.05):  0.6108061253972841
Accuracy on training poly 4 (0.05):  0.6581758841377904
Accuracy for poly 4 function kernal (0.1):  0.6190407396706155
Accuracy on training poly 4 kernal (0.1):  0.6723271883525842
Accuracy for poly 3 function kernal (0.01):  0.6060387171337763
Accuracy on poly 3 kernal (0.01):  0.6314541421249904
Accuracy for poly 3 function kernal (0.05):  0.6222190118462873
Accuracy on training poly 3 (0.05):  0.6603941966903797
Accuracy for poly 3 function kernal (0.1):  0.6330540306269864
Accuracy on training poly 3 kernal (0.1):  0.6774267574389964
>>>
```

Figure B.2: Angry Vs None



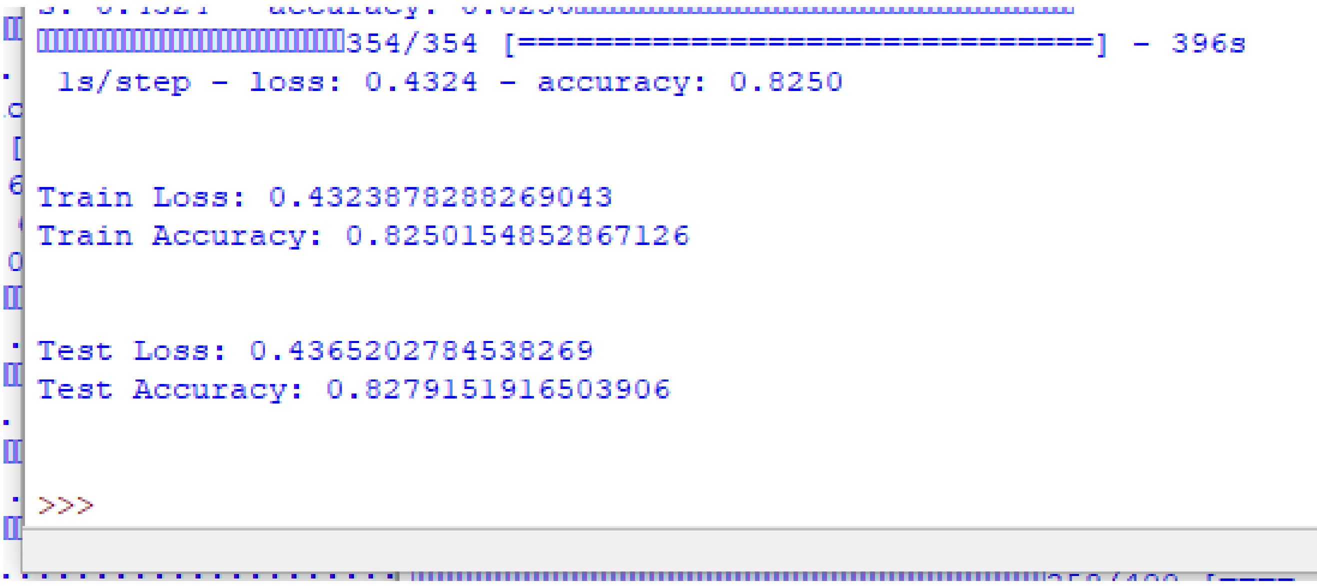
```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 18:58:18) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Thoma\Documents\Projects\FINAL YEAR PROJECT\Part2\Models\SVM
\SVM Sarcasm Vs None ALL\SVM.py
Accuracy for radial basis function kernal (0.01):  0.62
Accuracy on training radial kernal (0.01):  0.6406031629275469
Accuracy for radial basis function kernal (0.05):  0.66
Accuracy on training radial kernal (0.05):  0.6909157778595072
Accuracy for radial basis function kernal (0.1):  0.6875
Accuracy on training radial kernal (0.1):  0.7308569326958441
Accuracy for linear function kernal (0.01):  0.7004166666666667
Accuracy on linear kernal (0.01):  0.7290915777859507
Accuracy for linear function kernal (0.05):  0.75125
Accuracy on training linear (0.05):  0.8052960647296801
Accuracy for linear function kernal (0.1):  0.7645833333333333
Accuracy on training linear kernal (0.1):  0.836557557925708
Accuracy for linear function kernal (1):  0.7695833333333333
Accuracy on training linear kernal (1):  0.9217359323280618
Accuracy for sigmoid function kernal (0.01):  0.5983333333333334
Accuracy on sigmoid kernal (0.01):  0.6120632585509378
Accuracy for sigmoid function kernal (0.05):  0.6679166666666667
Accuracy on training sigmoid (0.05):  0.6857668260389849
Accuracy for sigmoid function kernal (0.1):  0.695
Accuracy on training sigmoid kernal (0.1):  0.7066568591393895
Accuracy for poly 4 function kernal (0.01):  0.54625
Accuracy on poly 4 kernal (0.01):  0.5739610150790732
Accuracy for poly 4 function kernal (0.05):  0.5641666666666667
Accuracy on training poly 4 (0.05):  0.6304523721956602
Accuracy for poly 4 function kernal (0.1):  0.57625
Accuracy on training poly 4 kernal (0.1):  0.6734093416697315
Accuracy for poly 3 function kernal (0.01):  0.5458333333333333
Accuracy on poly 3 kernal (0.01):  0.5685178374402354
Accuracy for poly 3 function kernal (0.05):  0.5670833333333334
Accuracy on training poly 3 (0.05):  0.615888194189040l
Accuracy for poly 3 function kernal (0.1):  0.5891666666666666
Accuracy on training poly 3 kernal (0.1):  0.661787421846267
>>>
```

Figure B.3: Sarcasm Vs None



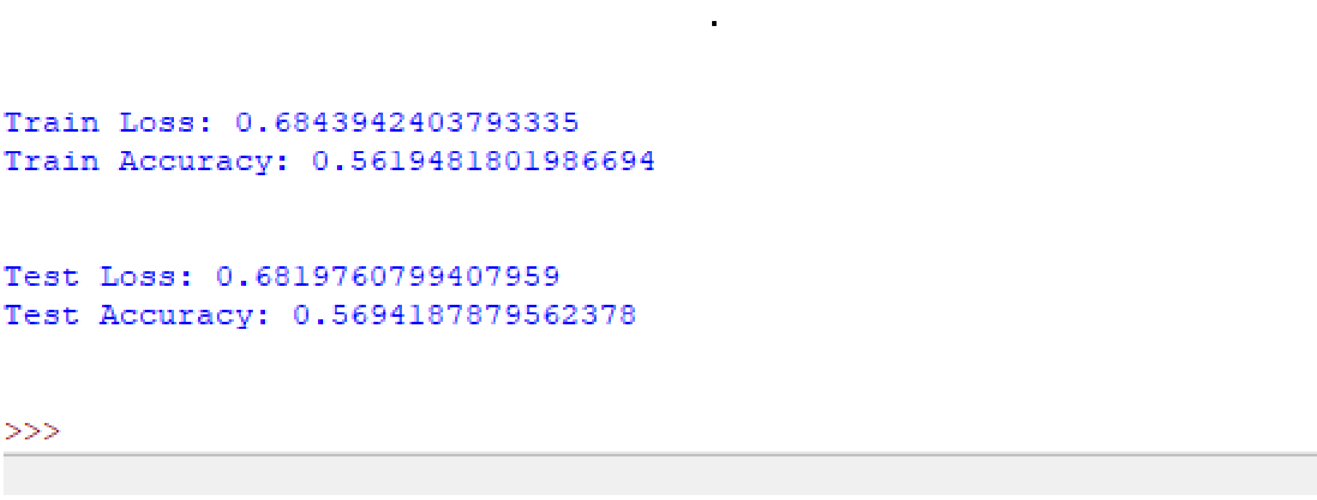
```
Python 3.7.5 (tags/v3.7.5:5c02a39a0b, Oct 15 2019, 00:11:34) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===================== RESTART: E:\FINAL YEAR PROJECT\SVM.py ====================
Accuracy for linear kernal (0.01):  0.8983104877271278
Accuracy on training linear kernal (0.01):  0.9008552779653387
Accuracy for polynomial kernal (0.01):  0.6789926681542875
Accuracy on training poly kernal (0.01):  0.6756695926176007
Accuracy for radial basis function kernal (0.01):  0.8999043672298375
Accuracy on training radial kernal (0.01):  0.8999549853702453
Accuracy for sigmoid basis function kernal (0.01):  0.8960790564233344
Accuracy on training sigmoid kernal (0.01):  0.8979855953184785
Accuracy for linear kernal (0.05):  0.9193496971628945
Accuracy on training linear kernal (0.05):  0.9520031510240828
Accuracy for polynomial kernal (0.05):  0.8992668154287536
Accuracy on training poly kernal (0.05):  0.9497524195363494
Accuracy for radial basis function kernal (0.05):  0.9005419190309213
Accuracy on training radial kernal (0.05):  0.9091267161827594
Accuracy for sigmoid basis function kernal (0.05):  0.8963978323238763
Accuracy on training sigmoid kernal (0.1):  0.8982669367544452
Accuracy for linear kernal (0.1):  0.9209435766656041
Accuracy on training linear kernal (0.1):  0.9678145397254108
Accuracy for polynomial kernal (0.1):  0.9187121453618107
Accuracy on training poly kernal (0.1):  0.9738352464550979
Accuracy for radial basis function kernal (0.1):  0.9050047816385082
Accuracy on training radial kernal (0.1):  0.9144722034661265
Accuracy for sigmoid basis function kernal (0.1):  0.9014982467325470
Accuracy on training sigmoid kernal (0.1):  0.9031622777402656
```

Figure B.4: SVM Happy Vs Sad



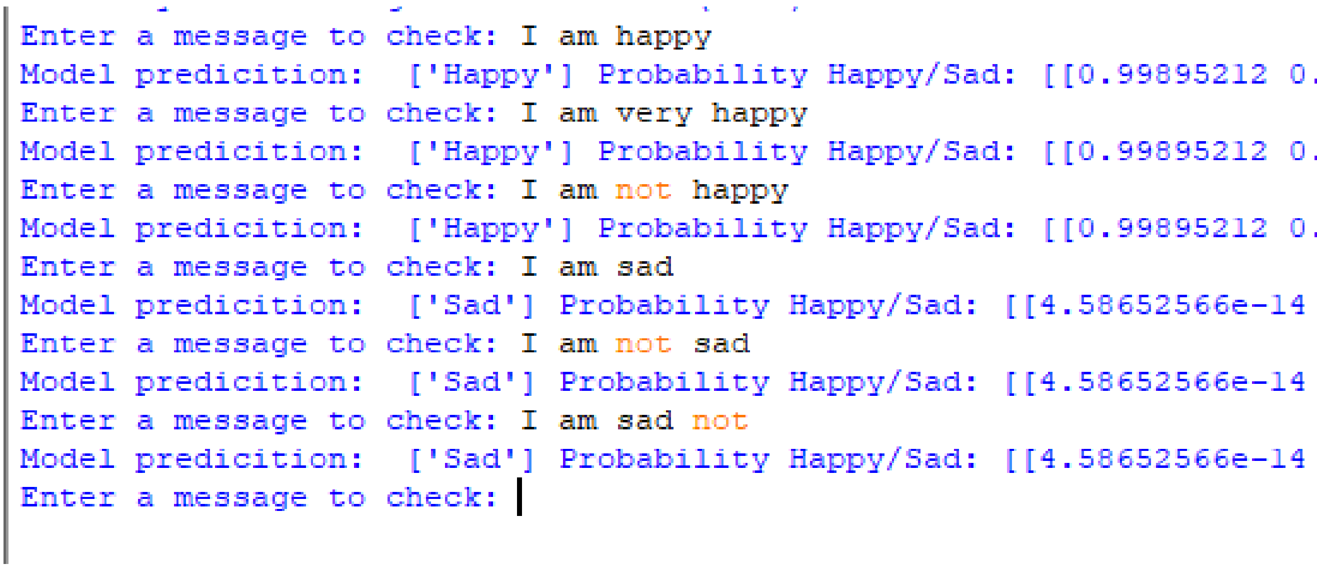
```
354/354 [==============================] - 396s
 1s/step - loss: 0.4324 - accuracy: 0.8250


Train Loss: 0.4323878288269043
Train Accuracy: 0.8250154852867126


Test Loss: 0.4365202784538269
Test Accuracy: 0.8279151916503906


>>>
```

Figure B.5: LSTM Angry vs None



```
Train Loss: 0.6843942403793335
Train Accuracy: 0.5619481801986694


Test Loss: 0.6819760799407959
Test Accuracy: 0.5694187879562378


>>>
```

Figure B.6: Basic Neural network Sarcastic Vs None

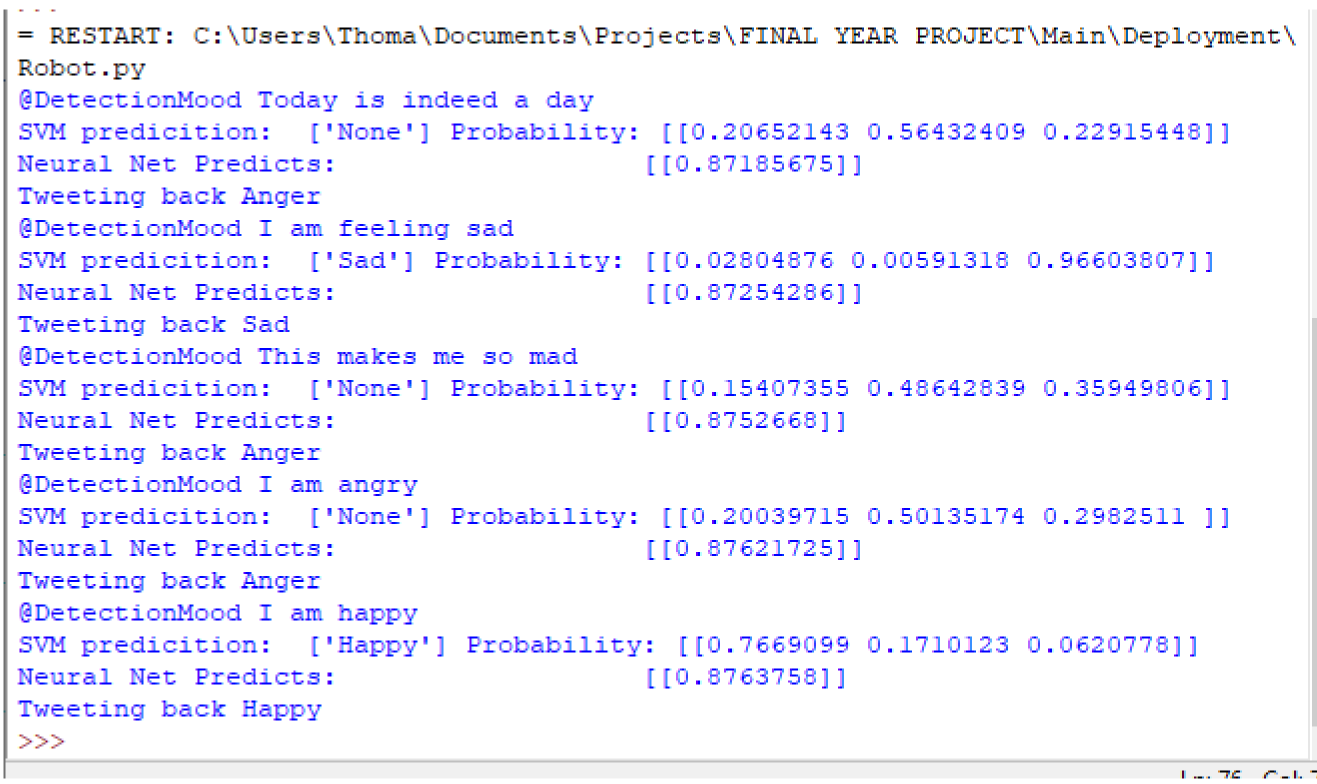


```
Enter a message to check: I am happy
Model predicition:  ['Happy'] Probability Happy/Sad: [[0.99895212 0.
Enter a message to check: I am very happy
Model predicition:  ['Happy'] Probability Happy/Sad: [[0.99895212 0.
Enter a message to check: I am not happy
Model predicition:  ['Happy'] Probability Happy/Sad: [[0.99895212 0.
Enter a message to check: I am sad
Model predicition:  ['Sad'] Probability Happy/Sad: [[4.58652566e-14
Enter a message to check: I am not sad
Model predicition:  ['Sad'] Probability Happy/Sad: [[4.58652566e-14
Enter a message to check: I am sad not
Model predicition:  ['Sad'] Probability Happy/Sad: [[4.58652566e-14
Enter a message to check:
```

Figure B.7: SVM not understanding sequencing

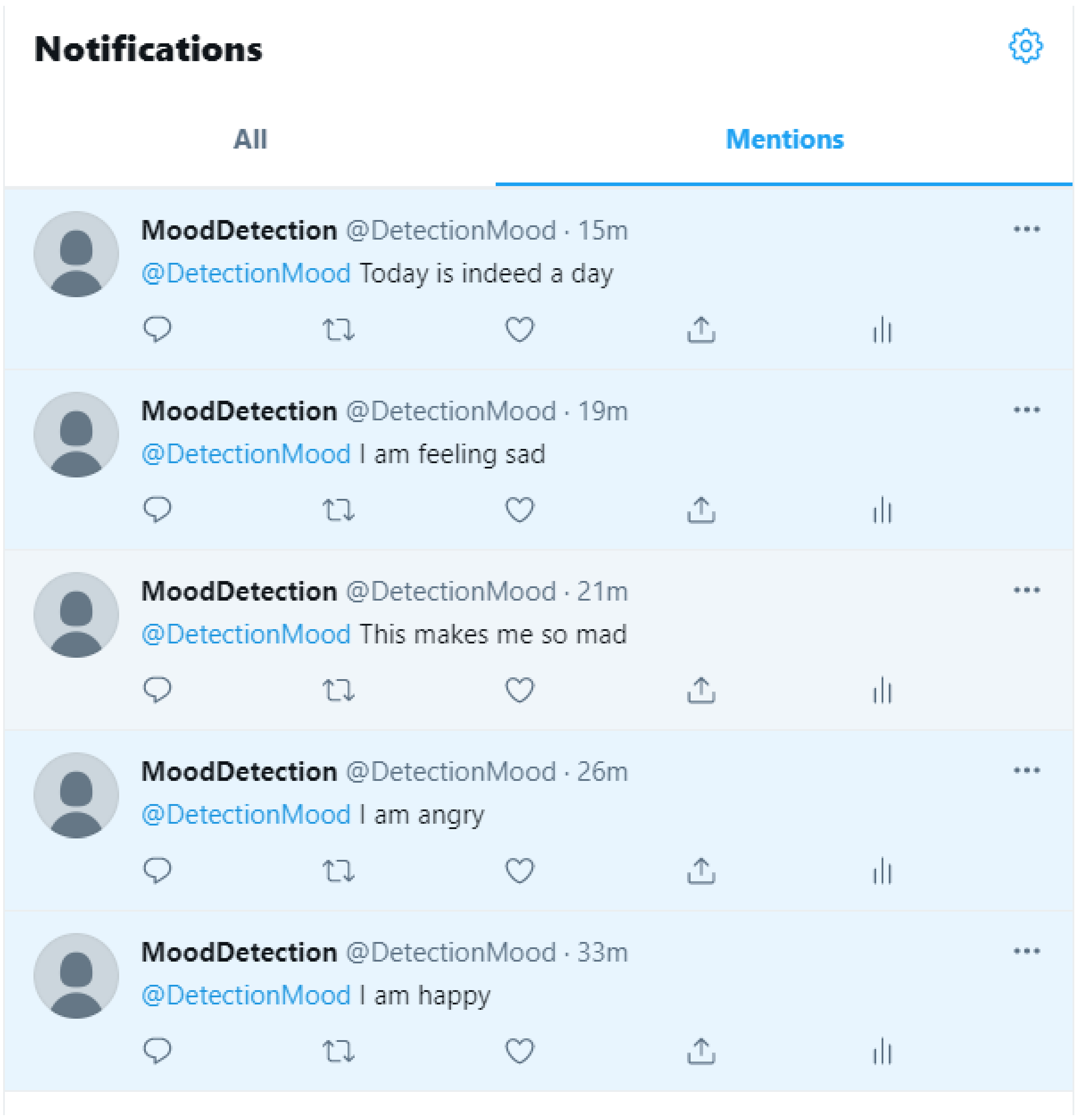
Appendix C

Demo Examples



```
= RESTART: C:\Users\Thoma\Documents\Projects\FINAL YEAR PROJECT\Main\Deployment\
Robot.py
@DetectionMood Today is indeed a day
SVM predicition:  ['None'] Probability: [[0.20652143 0.56432409 0.22915448]]
Neural Net Predicts:                    [[0.87185675]]
Tweeting back Anger
@DetectionMood I am feeling sad
SVM predicition:  ['Sad'] Probability: [[0.02804876 0.00591318 0.96603807]]
Neural Net Predicts:                    [[0.87254286]]
Tweeting back Sad
@DetectionMood This makes me so mad
SVM predicition:  ['None'] Probability: [[0.15407355 0.48642839 0.35949806]]
Neural Net Predicts:                    [[0.8752668]]
Tweeting back Anger
@DetectionMood I am angry
SVM predicition:  ['None'] Probability: [[0.20039715 0.50135174 0.2982511 ]]
Neural Net Predicts:                    [[0.87621725]]
Tweeting back Anger
@DetectionMood I am happy
SVM predicition:  ['Happy'] Probability: [[0.7669099 0.1710123 0.0620778]]
Neural Net Predicts:                    [[0.8763758]]
Tweeting back Happy
>>>
```

Figure C.1: Python code receiving tweets, analysing text and tweeting back result



MoodDetection @DetectionMood · 15m
@DetectionMood Today is indeed a day

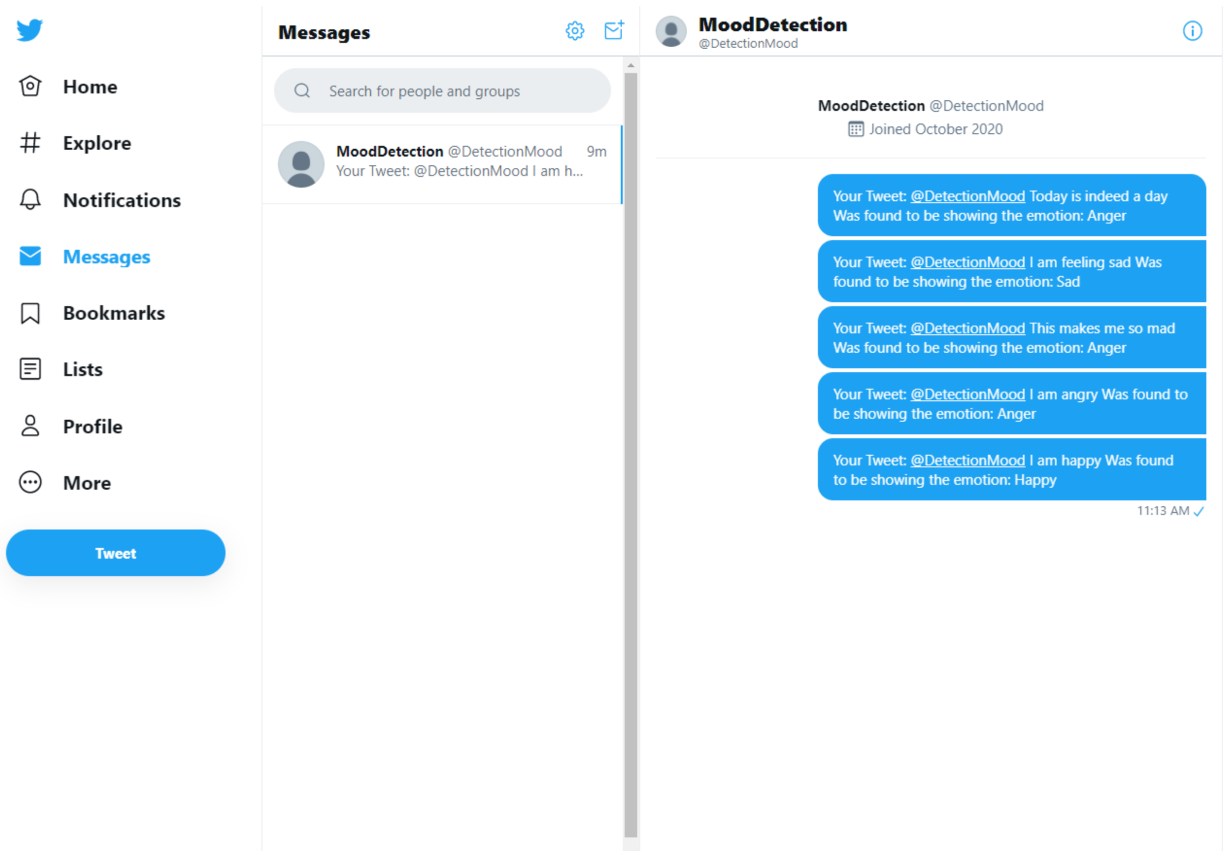
MoodDetection @DetectionMood · 19m
@DetectionMood I am feeling sad

MoodDetection @DetectionMood · 21m
@DetectionMood This makes me so mad

MoodDetection @DetectionMood · 26m
@DetectionMood I am angry

MoodDetection @DetectionMood · 33m
@DetectionMood I am happy

Figure C.2: Tweets tweeted at the twitter bot



MoodDetection @DetectionMood 9m
Your Tweet: @DetectionMood I am h...

MoodDetection
@DetectionMood

MoodDetection @DetectionMood
Joined October 2020

Your Tweet: @DetectionMood Today is indeed a day Was found to be showing the emotion: Anger

Your Tweet: @DetectionMood I am feeling sad Was found to be showing the emotion: Sad

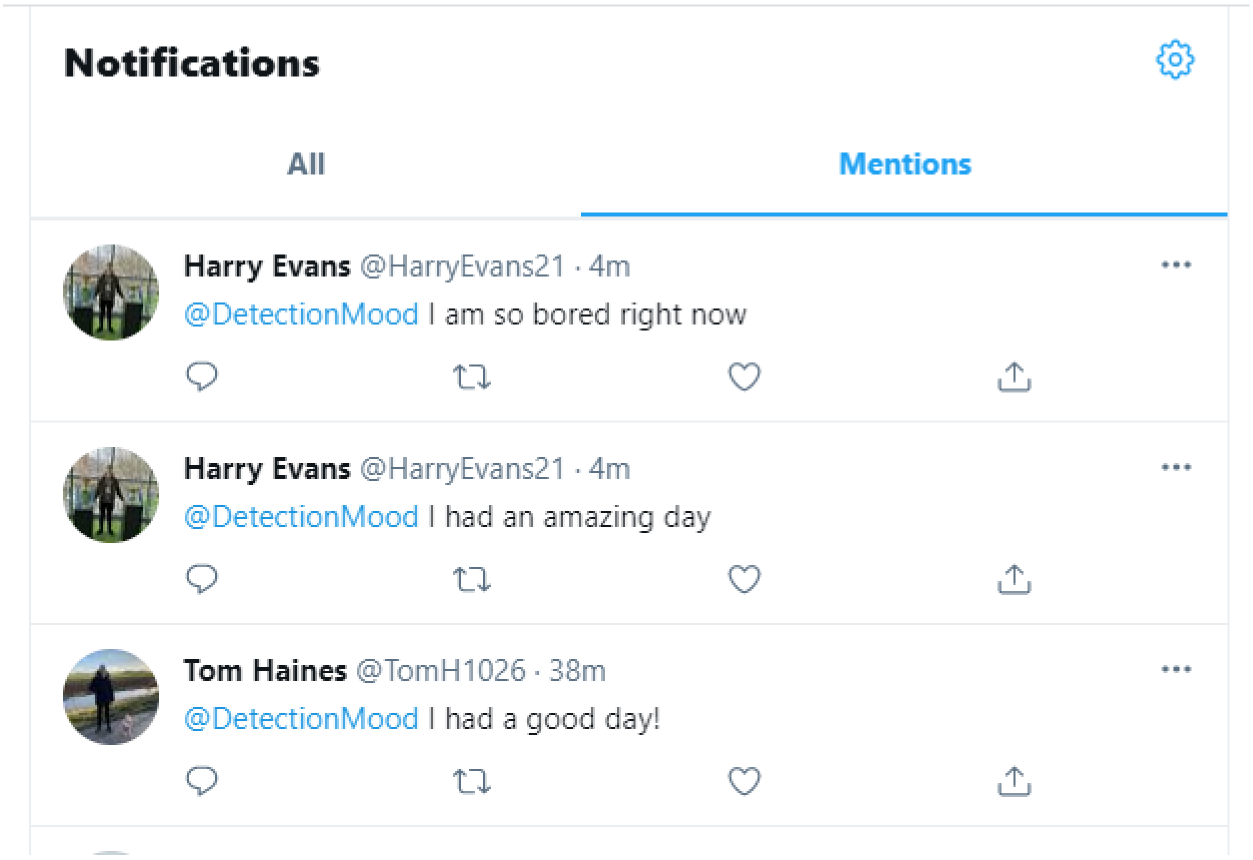
Your Tweet: @DetectionMood This makes me so mad Was found to be showing the emotion: Anger

Your Tweet: @DetectionMood I am angry Was found to be showing the emotion: Anger

Your Tweet: @DetectionMood I am happy Was found to be showing the emotion: Happy

11:13 AM ✓

Figure C.3: Message tweeted back at user



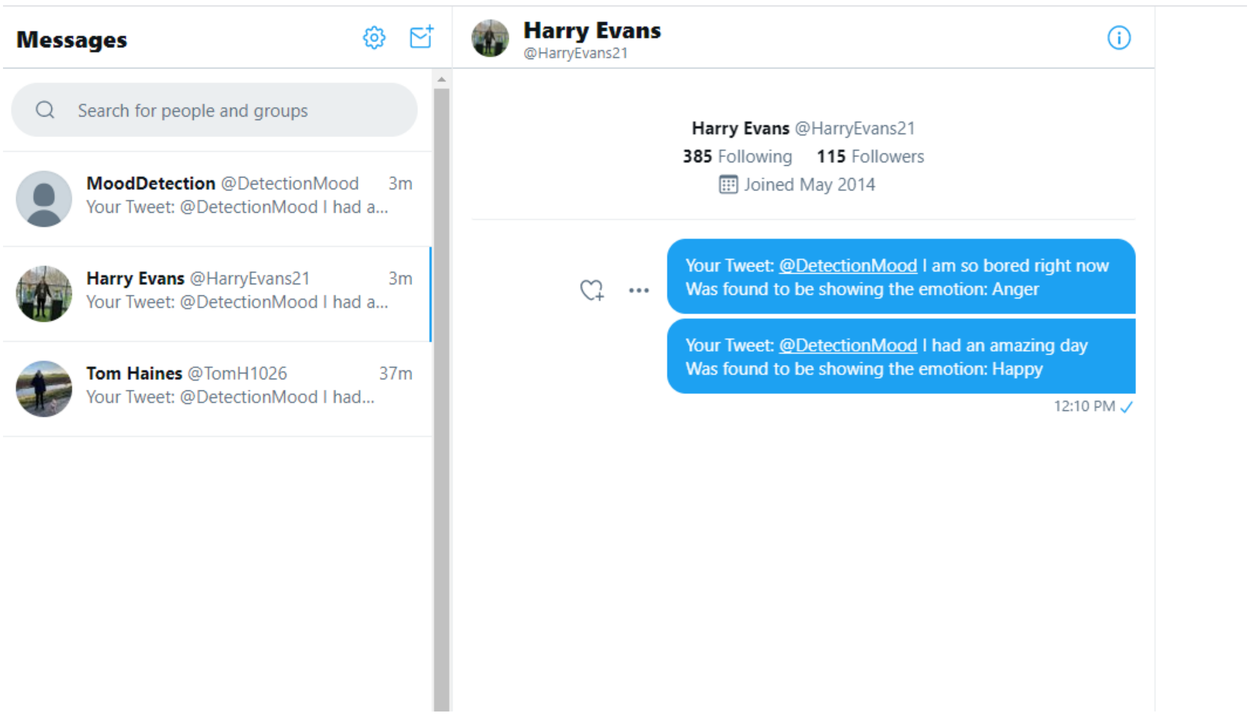
Notifications

Harry Evans @HarryEvans21 · 4m
@DetectionMood I am so bored right now

Harry Evans @HarryEvans21 · 4m
@DetectionMood I had an amazing day

Tom Haines @TomH1026 · 38m
@DetectionMood I had a good day!

Figure C.4: Tweets tweeted at the twitter bot



Messages

Search for people and groups

MoodDetection @DetectionMood 3m
Your Tweet: @DetectionMood I had a...

Harry Evans @HarryEvans21 3m
Your Tweet: @DetectionMood I had a...

Tom Haines @TomH1026 37m
Your Tweet: @DetectionMood I had...

Harry Evans
@HarryEvans21

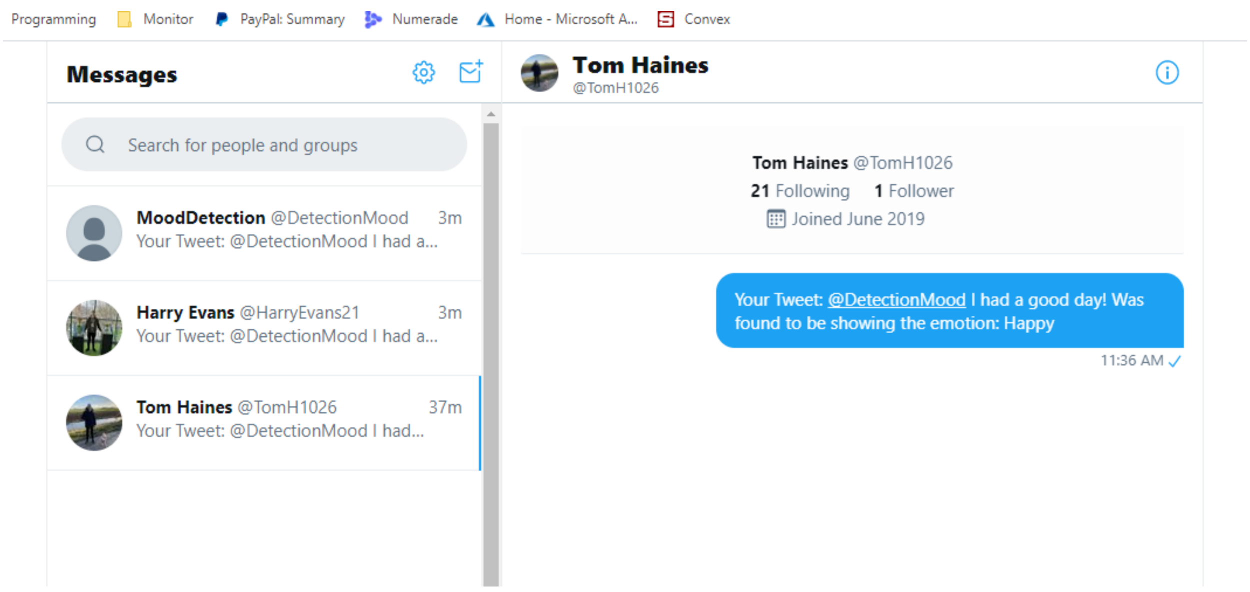
Harry Evans @HarryEvans21
385 Following 115 Followers
Joined May 2014

Your Tweet: @DetectionMood I am so bored right now
Was found to be showing the emotion: Anger

Your Tweet: @DetectionMood I had an amazing day
Was found to be showing the emotion: Happy

12:10 PM ✓

Figure C.5: Message tweeted back at user



Messages

Search for people and groups

MoodDetection @DetectionMood 3m
Your Tweet: @DetectionMood I had a...

Harry Evans @HarryEvans21 3m
Your Tweet: @DetectionMood I had a...

Tom Haines @TomH1026 37m
Your Tweet: @DetectionMood I had...

Tom Haines
@TomH1026

Tom Haines @TomH1026
21 Following 1 Follower
Joined June 2019

Your Tweet: @DetectionMood I had a good day! Was found to be showing the emotion: Happy

11:36 AM ✓

Figure C.6: Message tweeted back at user