The Social Dilemma - Sentiment Analysis

POSWAL, Abhishek & DALUWATTE, Dhulan Seniru
20557578 & 20553900
aposwal@connect.ust.hk & dsdaluwatte@connect.ust.hk

Dataset and Preprocessing

1. Dataset
   We decided upon using dataset 5 which is a Text Classification Dataset which could be found at the below mentioned link:
   https://www.kaggle.com/kaushiksuresh147/the-social-dilemma-tweets

   Column names in original dataset:
   - user_name
   - user_location
   - user_description
   - user_created
   - user_followers
   - user_friends
   - user_favourites
   - user_verified
   - date
   - text
   - hashtags
   - source
   - is_retweet
   - Sentiment
   -

   Below is their data type description along with the size of the dataset i.e. 14 columns and 20068 rows.

```
Dataframe Shape:  (20068, 14)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20068 entries, 0 to 20067
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   user_name         20067 non-null  object
 1   user_location     15860 non-null  object
 2   user_description  18685 non-null  object
 3   user_created      20068 non-null  object
 4   user_followers    20068 non-null  int64
 5   user_friends      20068 non-null  int64
 6   user_favourites   20068 non-null  int64
 7   user_verified     20068 non-null  bool
 8   date              20068 non-null  object
 9   text              20068 non-null  object
 10  hashtags          15771 non-null  object
 11  source            20068 non-null  object
 12  is_retweet        20068 non-null  bool
 13  Sentiment         20068 non-null  object
dtypes: bool(2), int64(3), object(9)
memory usage: 1.9+ MB
```

The two most important columns in this dataset are text and Sentiment. Although other columns might have minor or in some cases major impact, for our baseline models, we decided to use just the text column as the input and Sentiment as the target or label.

Further, for future advanced models, multiple features/columns such as “source”, “user_location”, “user_verified” will be taken into account as well.

The text column in the dataset contains the user’s tweet and the Sentiment column contains whether the sentiment associated with the tweet is Neutral, Negative or Positive. For the sake of simplicity, we mapped the sentiment to an integer value. Below is the mapping scheme:

Negative: 0, Neutral: 1, Positive: 2.

2. Preprocessing:

As imagined, there is a lot of noise in the dataset. The data in the text column consists:

- Text
    - Random Capitalization
    - Spelling errors
    - Punctuations
    - Newline
    - Multiple spacings
- Emojis
- Hyperlinks

Preprocessing this data for the baseline models will include lower casing alphabets, removing punctuations, newline, multiple spaces, emojis and hyperlinks.

This preprocessed data is then stored in a separate file named “preprocessed_data.csv”. This file contains 2 columns namely, “text” and

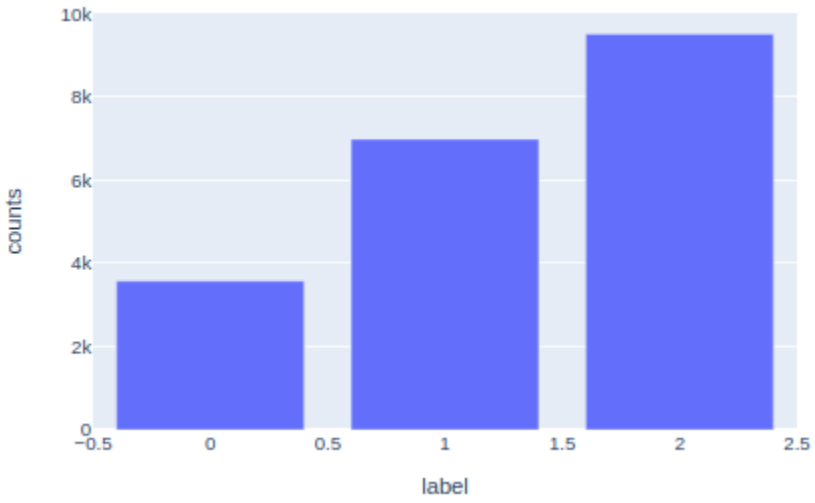
“label”. The “text” column contains processed text data and labels contain mapped values of Negative, Neutral and Positive sentiment i.e. 0, 1 and 2.

When working with Binary classification we had to preprocess twice once for keras and the second for pytorch. For keras we followed a similar technique except we used two mapped value negative which contained both negative and neutral and positive which had just positive.

Splitting of dataset:

The labels are then visualized to check how many 0’s, 1’s and 2’s are present and their ratio is then analyzed for test and train split.

Below is the sentiment label report.




```
Negative Sentiments:  3573
Neutral Sentiments:  6982
Positive Sentiments:  9513
```

As we can see, the number of negative sentiments in the report are much lower in ratio as opposed to neural and positive sentiments.

Due to this, we will separate the training set and testing set such that the test set contains all three sentiment classification in equal ratio i.e. 33.3% each and is 20% of the total dataset.

20% of the entire dataset = 20068 * 0.20 = 4014 (approx)
33.33% of 20% of entire dataset = 4014 * 0.3333 = 1337 (approx)

When we experimented with Binary classification we separated the training set and validation set such that the test set contains both sentiment classifications in equal ratio i.e. 50% of both classifiers and is 25% of the total dataset.

1. **Problems Encountered:**

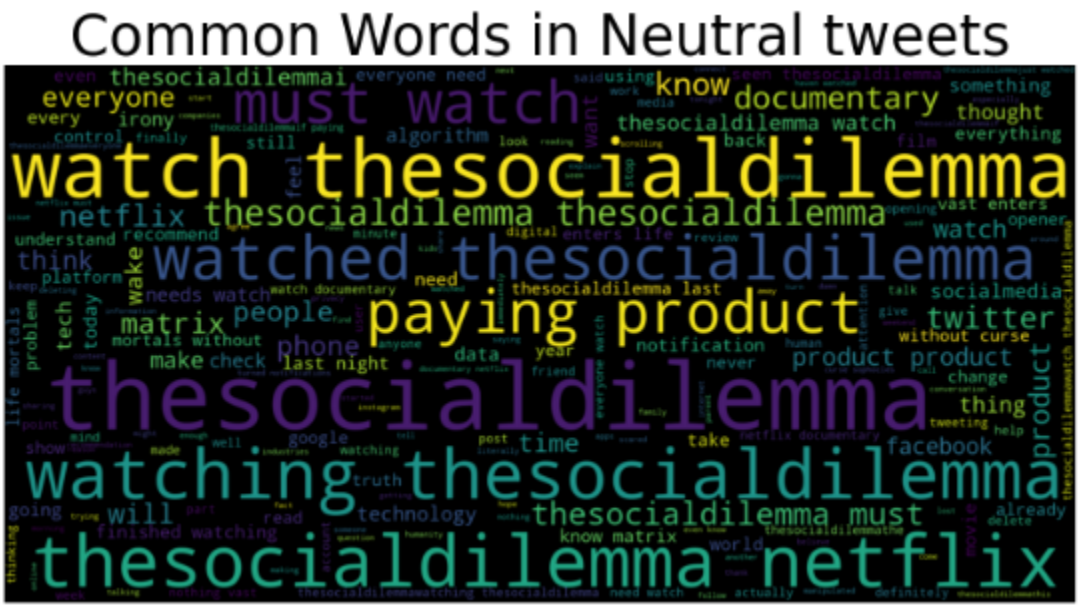
During Training:

While training it was found that some of the rows in data after preprocessing became empty. Therefore, they were either to drop empty rows or fill them with some values in order for us to predict it's output.

2.1 Drop Empty Rows:

We then decided to take a quick peek into the null row labels to see the dominant labels whose text column was null. Surprisingly, we found that all of them were labelled '1' aka "Neutral".

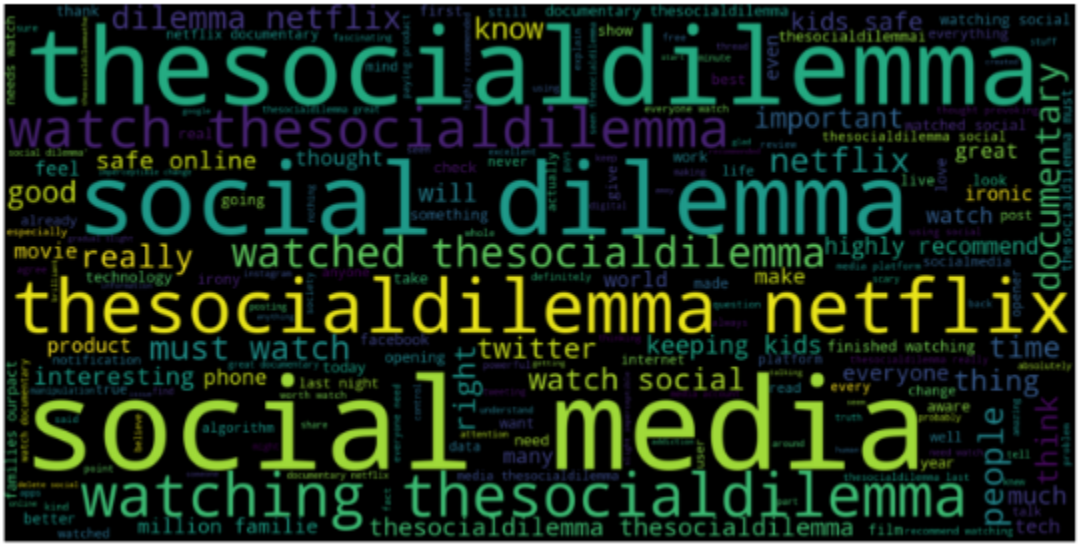
Based on this discovery, we decided to fill all the null rows with the most commonly found words in neutral labelled text data. Below is the matrix that describes the most commonly found words in the neutral data rows.



“thesocialdilemma” was apparently the most common word. This seemed a bit off from what we would have expected and before reaching the final conclusion and putting “thesocialdilemma” or “netflix” in our empty rows, we decided to do a further study on the common words that appeared in negative and positive sentiments. The results were not surprising. Below are their matrices.

Common Words in Negative tweets

Common Words in Positive tweets



As we may have expected, both “thesocialdilemma” and “netflix” were quite common in negative and positive tweets too. Therefore, we decided to remove those words from our “text” column in the preprocessing to see its impact in the later models.

2.2 Issues with Pytorch and multiple classification

We had spent a significant time attempting to use pytorch for creating an RNN model for multiple classification.

```
def prediction (predicted_outputs):
```

```
    preds = []
    for i in predicted_outputs:
        if i < (2/3):
            preds.append(0)
        elif i > (4/3):
            preds.append(2)
        else:
            preds.append(1)
    return torch.FloatTensor(preds).to(device)
```

The above snippet shows one of the attempts that we made to predict the output. However we were also not completely sure of the threshold we were using and as such we decided to switch to keras for multiple classification. However this was not completely in vain as when we were creating the Binary classification we were able to apply some lessons we learnt such as how to pad and process the dataframe more efficiently.

2.3 Concerning posters who posted several times

When it came to individuals who posted several times we questioned whether it was more beneficial to consider all the tweets as one person's entire message or to not and choosing what was the most common sentiment. We ended up deciding that it would be more harmful as it would reduce the size of the dataset and several tweets could be only slightly negative but one would be very positive making the sentiment neutral but the calculation would make it negative.

2.4 Concerns about misidentification

We were unsure how to handle the scenario where there would be misclassified data. However, the professor said that this is generally a case and that our model should be robust.

Computing Environment, Machine Learning Methods and Parameter Settings

1. **Hardware Computing Environment:**
   There were 2 computing environments used. Both Google Colab and Jupyter Notebook were used on a linux desktop with 6 Gb GPU RAM with CPU i7. Google Colab was generally used by both members to allow sharing code efficiently and at ease. Jupyter was used on the Linux system just to maintain proper organisation and to run more models concurrently (3 models were able to be run 2 using Colab on 2 different devices and 1 on the linux). The final product of the Binary Classification was made in Google Colab. The final Product of the Multi Class Classification was made in jupyter

2. **Software Computing Environment:**
   We generally used models from scikit-learn, Pytorch and Keras for the majority of our baseline and final model as well. Given that we had a thorough enough understanding of the models from the lectures, tutorial and the previous programming assignments, not much time was spent on building the architecture from ground up. As an open end project, we took the architectures that were already built and tweaked the parameters a bit for the most part to help us understand more on building up the knowledge prematurely provided to us.

   While some of the work was based off of the preexisted work, the work has been properly cited and referenced.

   2.1 Below is the description of the version of the library used in the Multiclass and Binary Sentiment analysis.

```
(pd.__version__) 1.1.5

(nltk.__version__)3.2.5

(gensim.__version__)3.6.0

(sns.__version__)0.11.1
```

(torch.__version__) 1.8.1+cu101

(re.__version__) 2..21

(np.__version__) 1.19.5

Torchsummary 1.3.0

-

3. **Machine Learning Task:**

   The task as the topic or heading of the project suggests was quite trivial. Given a string containing meaningful text that can be deciphered by humans as an input, the aim of the model is to predict or output the best befits the sentiment of the text as closely as possible. There are two separate ways that sentiments are detected in this project. First one includes the classification of data in three different classes i.e. multiclass classification.

   In the multiclass classification of sentiment analysis, we divide the sentiments into 3 different subclasses i.e. Neutral, Negative and Positive. However, in binary classification the sentiments are only divided into 2 different subclasses i.e. Negative and Positive. The second possible way is to use binary classification thus choosing between 2 classes we do this by sampling neutral as negative thus oversampling the negative field as well. The reason for removing Neutral class entirely shall be discussed in detail in the experiments below.

   The analysis of the models, parameters and hyperparameters used for the models involving multi and binary classification will be presented in the later parts of the report.

   For the input we perform dimensionality reduction with the help of Principal Component Analysis. We only pick or choose the features or dimensions that we think are most important and then later identify and rectify the layers based on experiments. For example, initially the input is just the

"text" attribute but later we also make use of the attributes such as "location" and device

Our outputs are accuracies for training and validation

4. **Machine Learning Models:**

Models Used:

- RNN (Recurrent Neural Networks)
- LSTM (Long Short Term Memory)
- GRU (Gated Recurrent Unit)
- XGBoost (Gradient Boosting Framework)
- SVC (Support Vector Classification)
- Random Forest Tree
- Decision Tree
- Logistics regression (3 different feature extraction method used i.e. binary, count based and tg-dif)

5. **Parameter Settings:**

Parameters Used:

In Linear Regression

- Activation layers: (sigmoid and softmax)
- Learning rate (0.001)
- Solver (saga, sag, and liblinear)
- Penalty (l2)
- Verbose (1, 3)

In Decision Tree, and Random Forest

- max_depth (20)
- criterion (entropy)
- max_features (auto and log2)
- random_state(4211)

In Support Vector Classification

- Verbose(true)
- Kernel(rbf)
- C(5)
- random_state(4211)

In Binary Classification

All RNN Layer had

- Criterion BCELoss (used as we used sigmoid during the model build)
- Optimizer Adam
- Epochs 15

Source Code

- Please find the source code attached to the zip file and run it either or jupyter notebook or google colab. It is recommended to use google colab in case the CPU/GPU of the computer is not powerful enough. View the ReadMe file to see how to run

Experiments Performed

1. **Add data from more columns:**

Adding data from more columns into text column of our train and test csv files helps such as adding the source i.e. Android, iPhone, Web… because people using different devices might have different opinions.

2. Tuning hyperparameters: The major hyperparameters such as learning rate, solver, penalty and verbose were mainly used to tune the model. Many different times was the model analyzed and to showcase the comparison between different models, we made 12 linear regression models which can still be seen in our code. With a baseline model performing as with accuracy rate as low as 69%, after tuning the hyperparameters our final regression performed with decently high testing accuracy rate i.e. as high as 85%.

3. Dropping empty rows and dropping most common words: This was a part of preprocessing as mentioned above. The common words frequently appeared in all the 3 classes were: ‘thesocialdilemma”, “netflix”, “tweet” and other stopwords. Nothing but a simple preprocessing method was involved in the first method of preprocessing i.e. removing extra space, hyperlinks etc., however, more noise was not removed such as extra stopwords. This was only improved in model

4. Adding most common words based on the labels

5. Experimenting with different models: In total, we used 10 models on just one type of classification. This does not factor in the models with different

hyperparameters and the same models used on different preprocessing methods.

6. Oversampling/Undersampling: Due to the oversampling in positive sentiments, no matter what model we built, we found that it would be rigged to some extent. There was no way to solve this problem but to either oversample the less sampled labels by making duplicates. This from our discovery did not improve our model to much extent.

   As a result, we decided to make the model a binary classification problem. To achieve this, we decided to change all the neutral labels to negative. Now, we understand that one might question this theory of ours, but for the sake of experiment we had to try. Since the labels were neutral, we initially thought it would slightly improve our basic models such as Decision Tree, Random Forest, XGBoost and Support Vector Machine but the results we got were unexpected. The difference in these models in terms of accuracy for binary classification when compared to multiclass classification was as high as 40-50%.

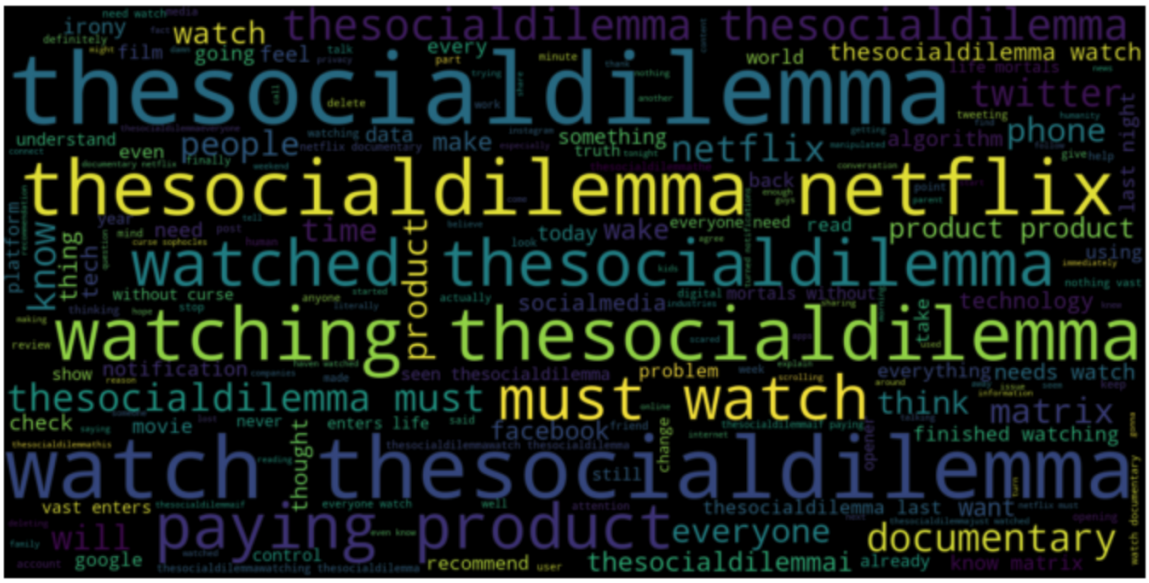
   Initially, the thought came that we implemented them incorrectly for either one of the classification, however, as surprising as these results were, they were correct.

   Therefore, changing our labels and methods of oversampling and undersampling can drastically change our results.

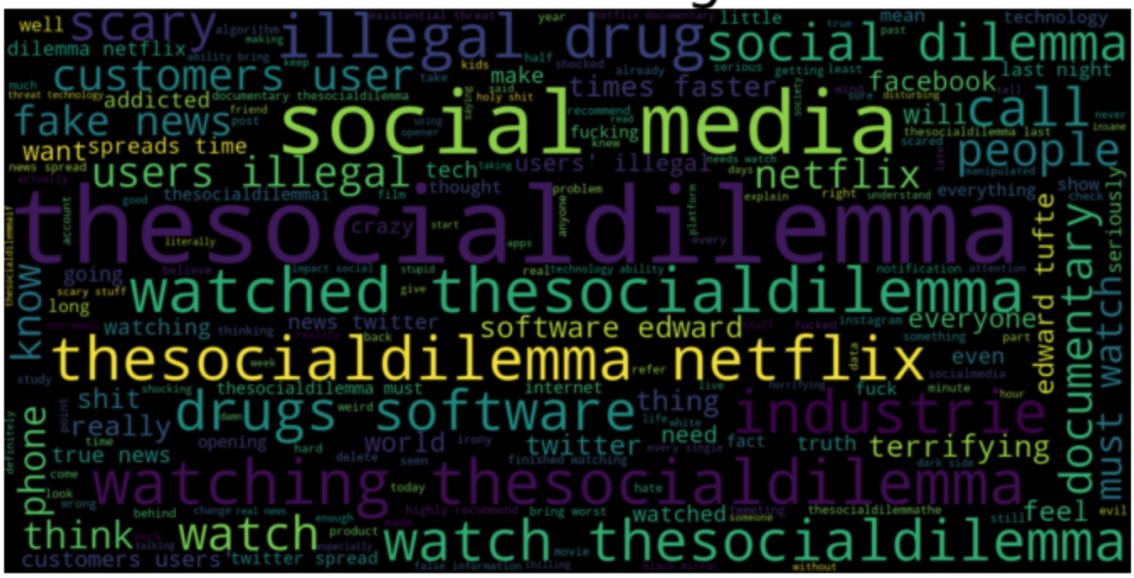
Visualization and discussion of results

Preprocessed common words (preprocessor_1):

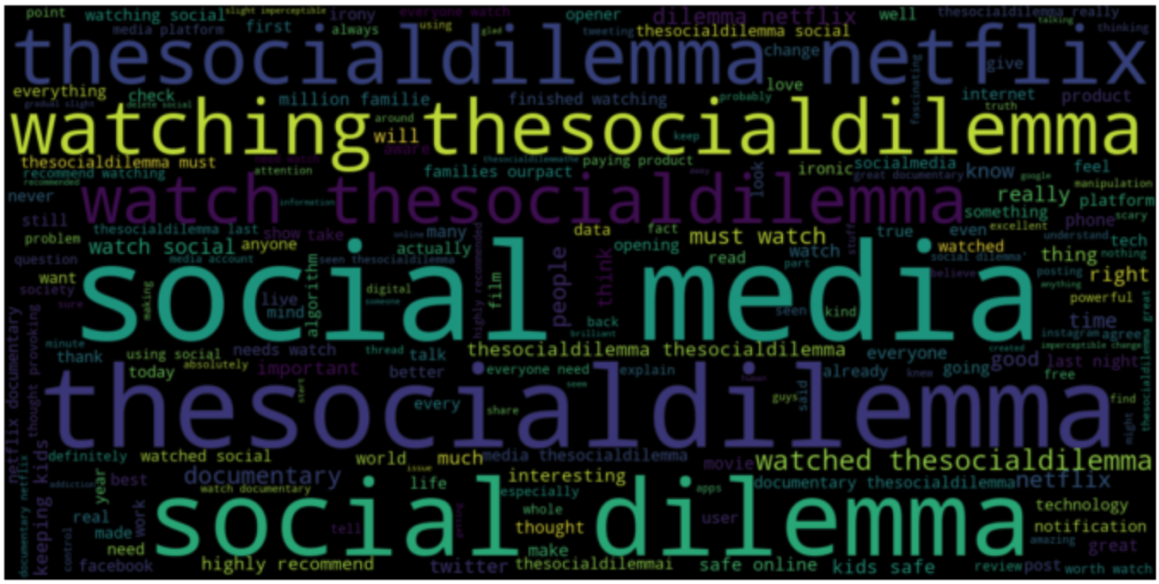
Common Words in Neutral tweets



Common Words in Negative tweets

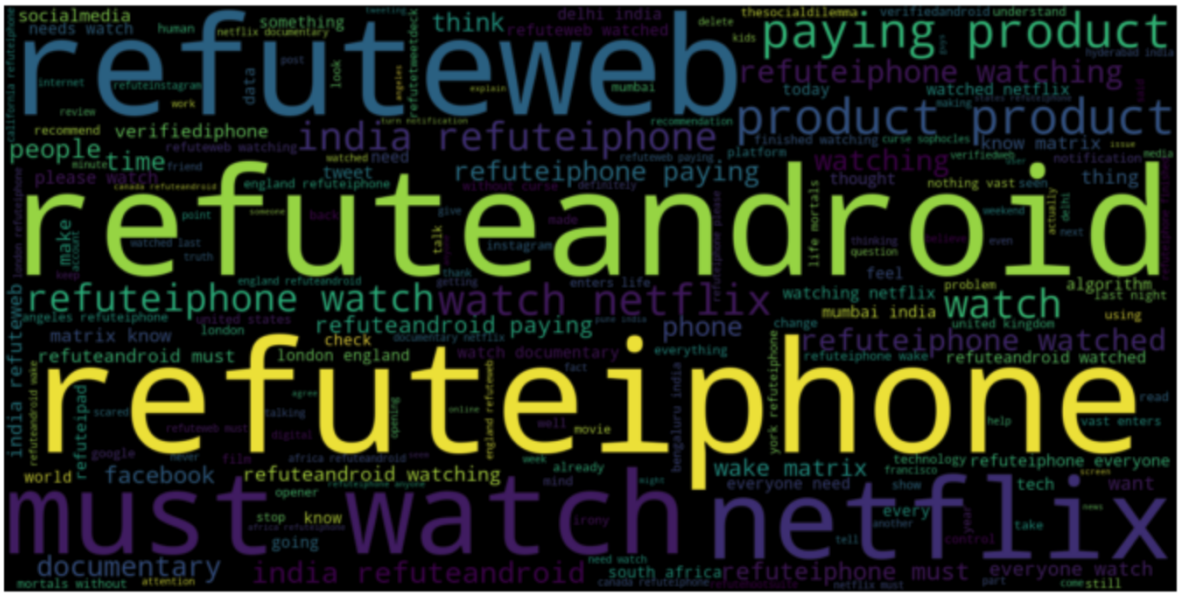


Common Words in Positive tweets

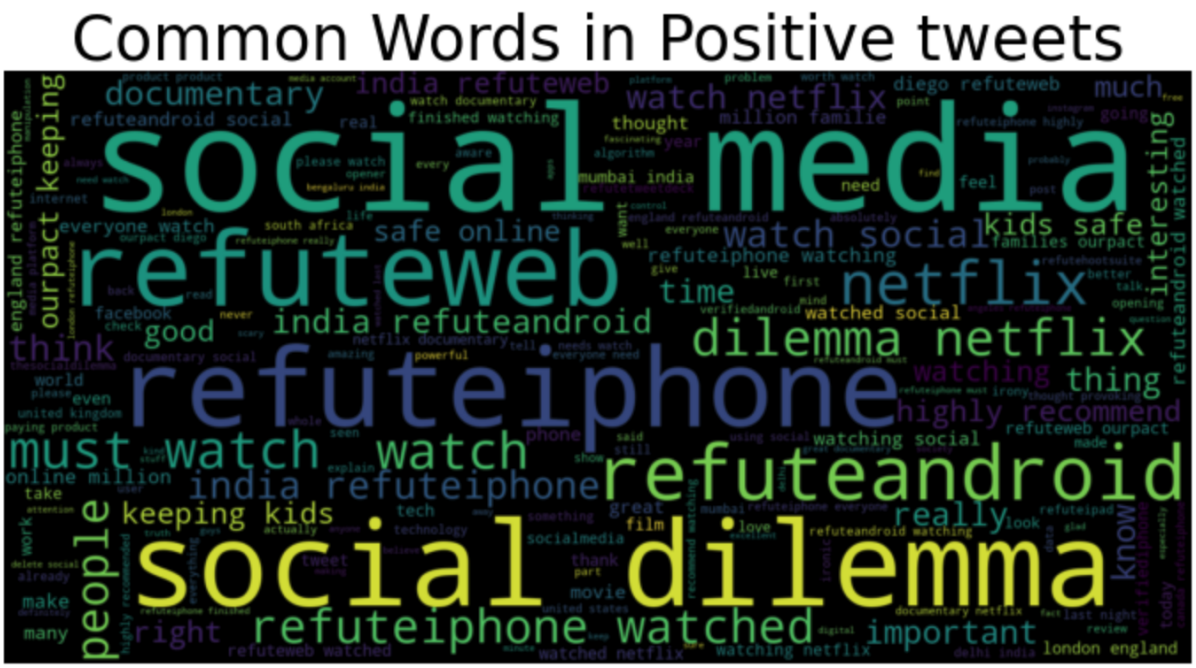


Preprocessed common words (preprocessor_1):

Common Words in Neutral tweets



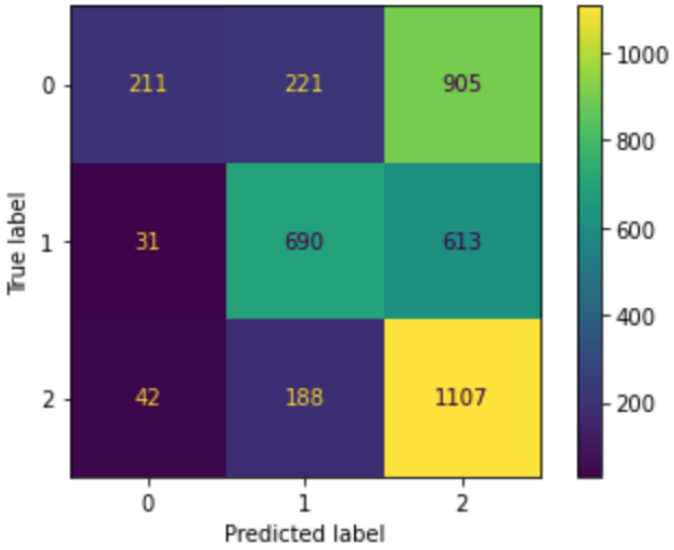
Common Words in Negative tweets



1. Confusion Matrix

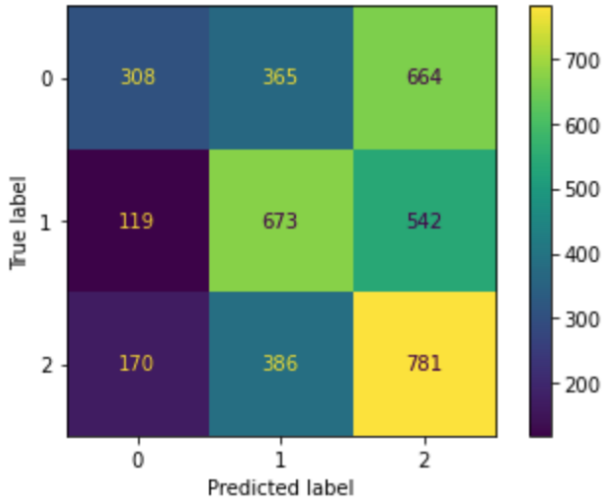
1.1 **With preprocessing method 1**

```
Training Accuracy [XGB]:  0.8443363880057353
Validation Accuracy [XGB]: 0.500998003992016
f1 score [XGB]: 0.462035607814459
```



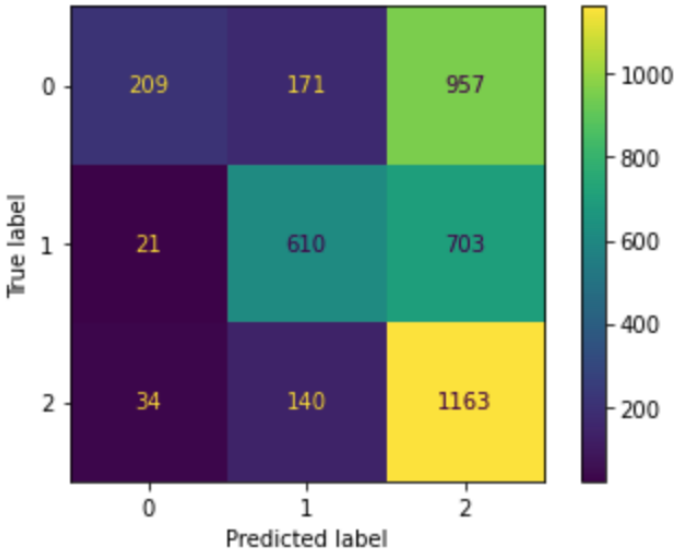
a.

```
Training Accuracy [DecisionTree]:  0.9998129792406957
Validation Accuracy [DecisionTree]: 0.43962075848303395
f1 score [DecisionTree]: 0.42544032631795115
```



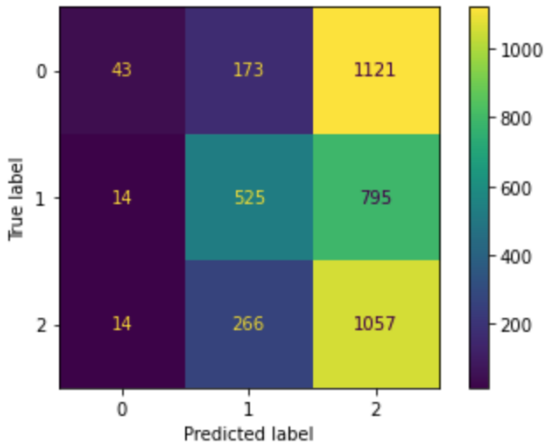
b.

```
Training Accuracy [RandomForest]:  0.9998129792406957
Validation Accuracy [RandomForest]: 0.4945109780439122
f1 score [RandomForest]: 0.45368180662106405
```



c.

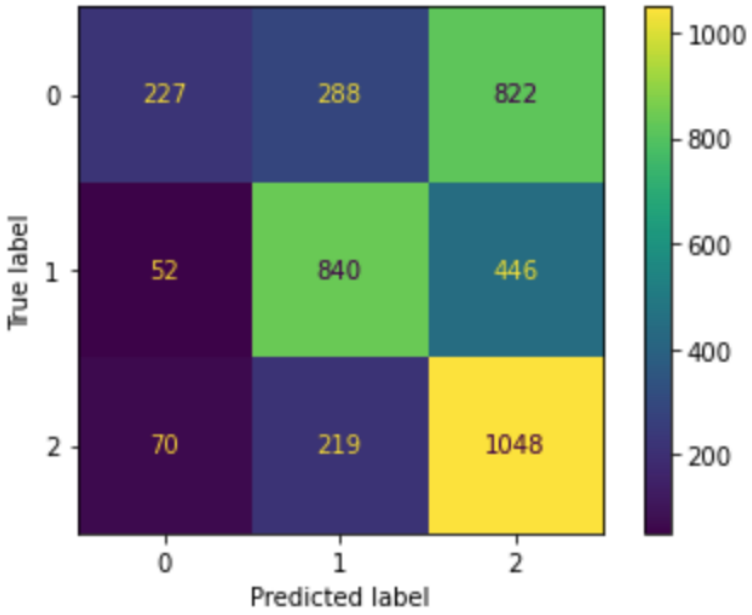
```
Training Accuracy [SupportMachineVector]:  0.6295742160713172
Validation Accuracy [SupportMachineVector]: 0.40543912175648705
f1 score [SupportMachineVector]: 0.336071561102163
```



d.

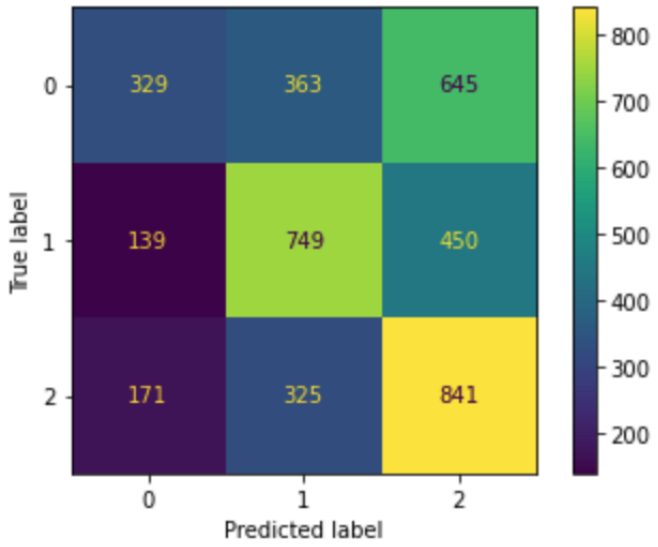
1.2 **With different preprocessing method 2 and extra added features:**

```
Training Accuracy [XGB]:  0.8494020926756353
Validation Accuracy [XGB]: 0.5271684945164506
f1 score [XGB]: 0.489617159432863
```



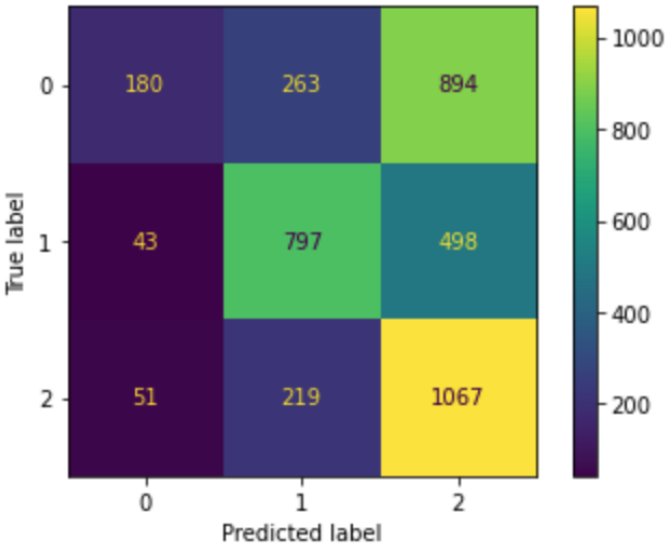
a.

```
Training Accuracy [DecisionTree]:  0.999813153961136
Validation Accuracy [DecisionTree]: 0.4783150548354935
f1 score [DecisionTree]: 0.4622584673430353
```



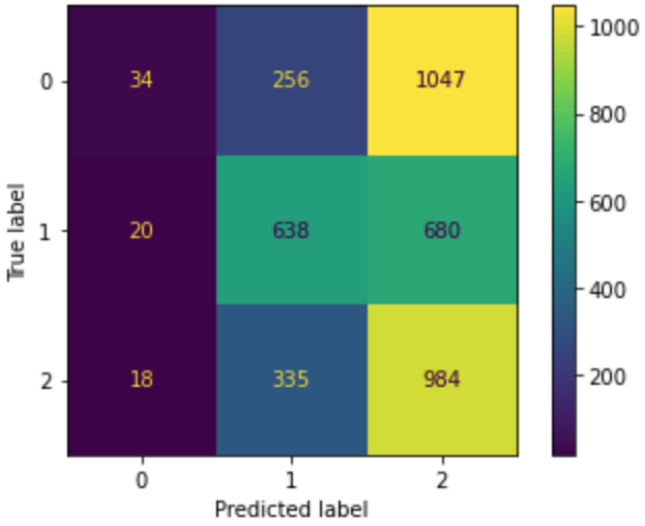
b.

```
Training Accuracy [RandomForest]:  0.999813153961136
Validation Accuracy [RandomForest]: 0.5094715852442672
f1 score [RandomForest]: 0.4649455303953435
```



c.

```
Training Accuracy [SupportMachineVector]:  0.6044469357249627
Validation Accuracy [SupportMachineVector]: 0.4127617148554337
f1 score [SupportMachineVector]: 0.3438733587178098
```

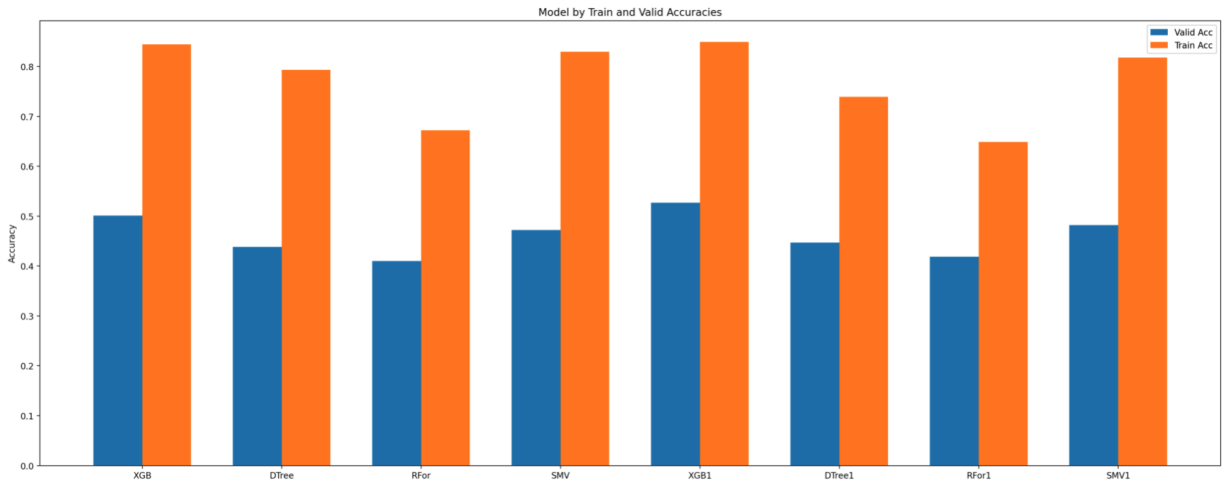


d.

None of the first eight training, validation accuracies, f1_score and confusion matrices are as appealing. There is no mystery to it. It is because all those models are highly algorithm based. Decision tree is a tree that determines the probabilities based on yes/no questions on every node until it reaches a leaf node. SVM similar to Logistics Regression finds the perfect threshold values to predict the outcome. Granted that it does not perform well here, but SVM does perform well at other places.

Now moving onto the difference between 1.1 and 1.2. In 1.1, as previously mentioned, nothing major in terms of preprocessing was done. Typical stopwords from the library and extra space and line removal was done.

However, in 1.2, we had mixed different features or dimensions together and further preprocessing was applied in hopes of getting a better accuracy rate. By providing more features to the model, we presented more questions for models like decision trees to be asked. Henceforth, making its accuracy better but not as drastically as we had hoped for. We had also removed data that was serving no use such as the hash tags that appeared in all the classification labels for instance, “tweet”, and “netflix”. Even this did not make much of a difference in our models rate of convergence or learning.



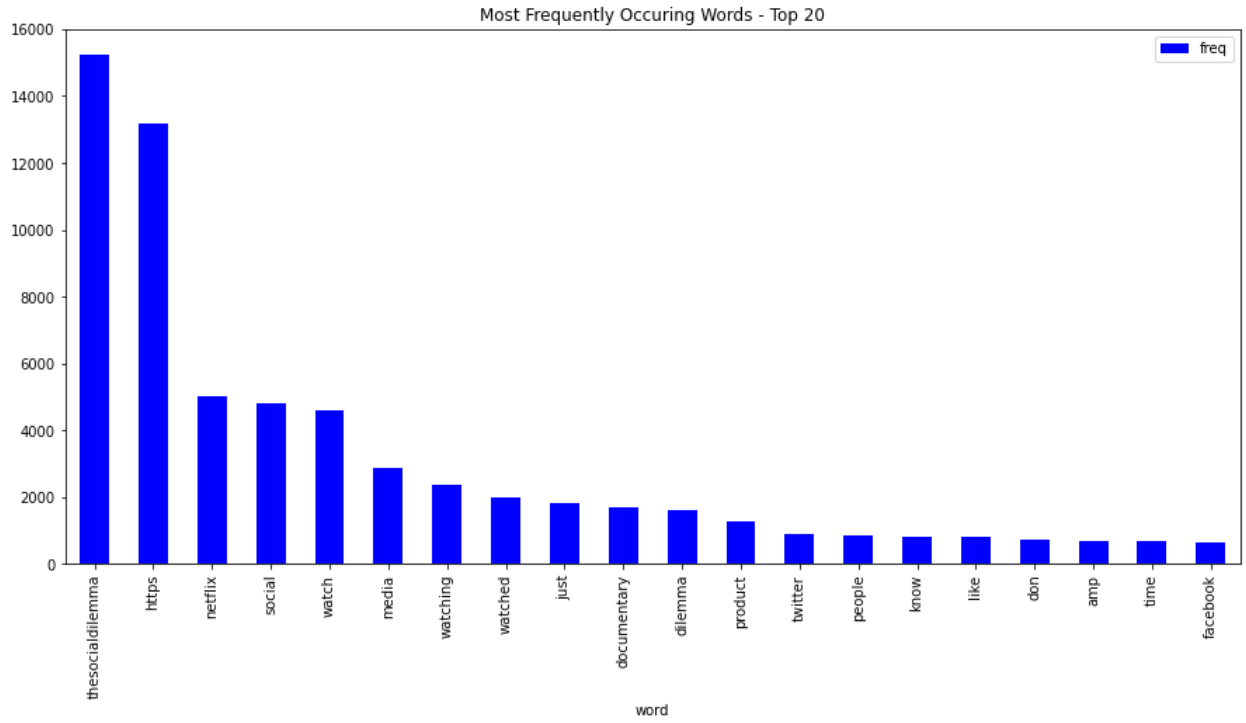
The above data clearly defines all the models and their accuracy rates. The ones marked with “1” at the end of their name are the models that were trained from preprocess method 2 and others from preprocess method 1. Now as you can see, the results with method 2 are slightly better as aforementioned due to the introduction of more dimensions such as “source”, “user_location” and “user_verified” in addition to the “text” attribute. Now, these dimensions also introduced noise which we realized later.

We could have improved our model by removing that noise but we were not sure if that would make any difference and under these circumstances we decided to use typical LSTM and GRU models to improve our accuracies.

Binary Classification Experiment and Results

We wanted to see if we could make a better model with binary classification than with multiple classification and these were our experiments

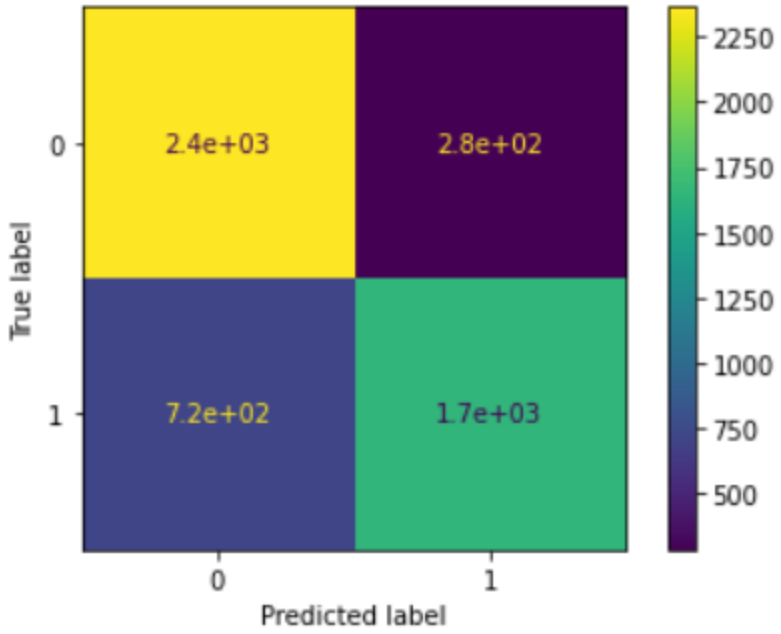
Showcase of most common words



First tested how the models that were currently being used faired with binomial classification

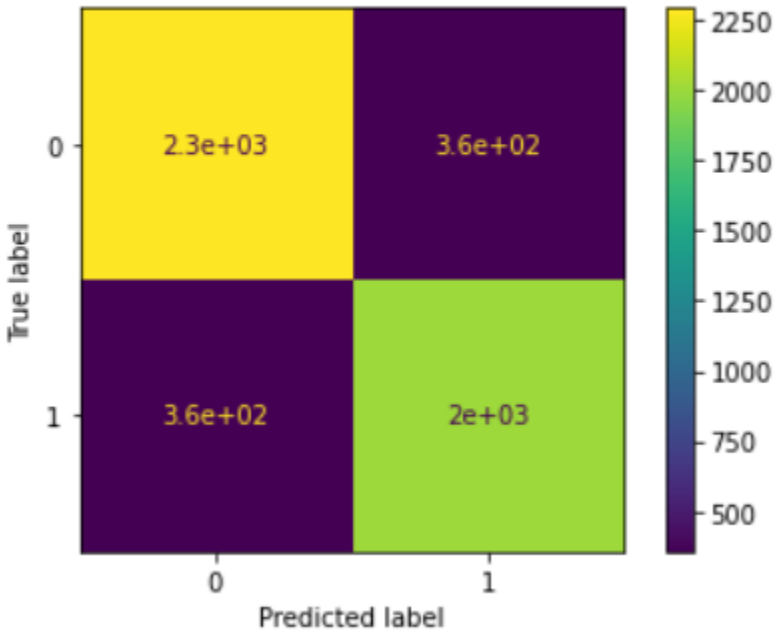
XGB

```
Training Accuracy [XGB]:  0.812637034084114
Validation Accuracy [XGB]: 0.8002790512258322
f1 score [XGB]: 0.7977730804891426
```



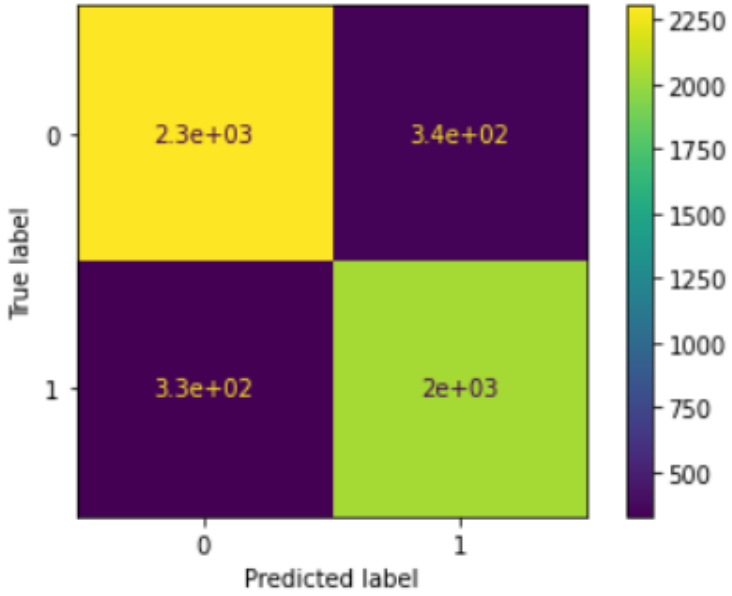
Decision Trees

```
Training Accuracy [DecisionTree]:  0.9991362700152814
Validation Accuracy [DecisionTree]: 0.8568865856089296
f1 score [DecisionTree]: 0.8568801287299453
```



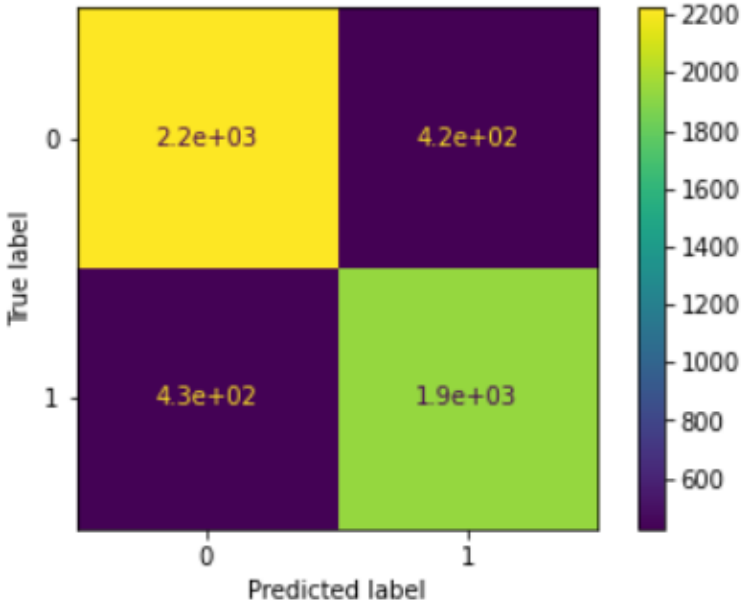
RandomForest

```
Training Accuracy [RandomForest]:  0.9991362700152814
Validation Accuracy [RandomForest]: 0.8664540562088898
f1 score [RandomForest]: 0.8664764441075761
```



Support Machine Vector

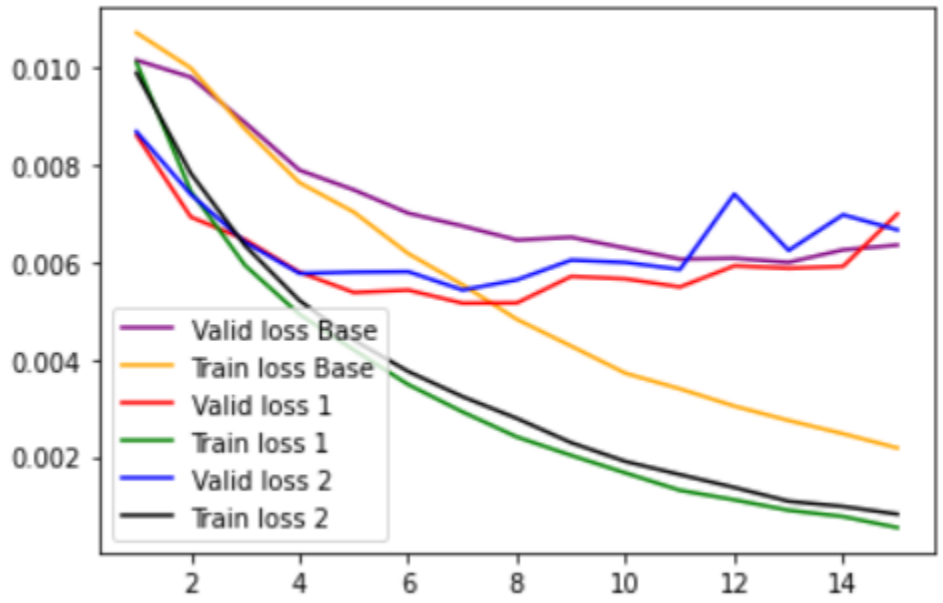
```
Training Accuracy [SupportMachineVector]:  0.9596704537904458
Validation Accuracy [SupportMachineVector]: 0.8289814630257126
f1 score [SupportMachineVector]: 0.8289658117257686
```



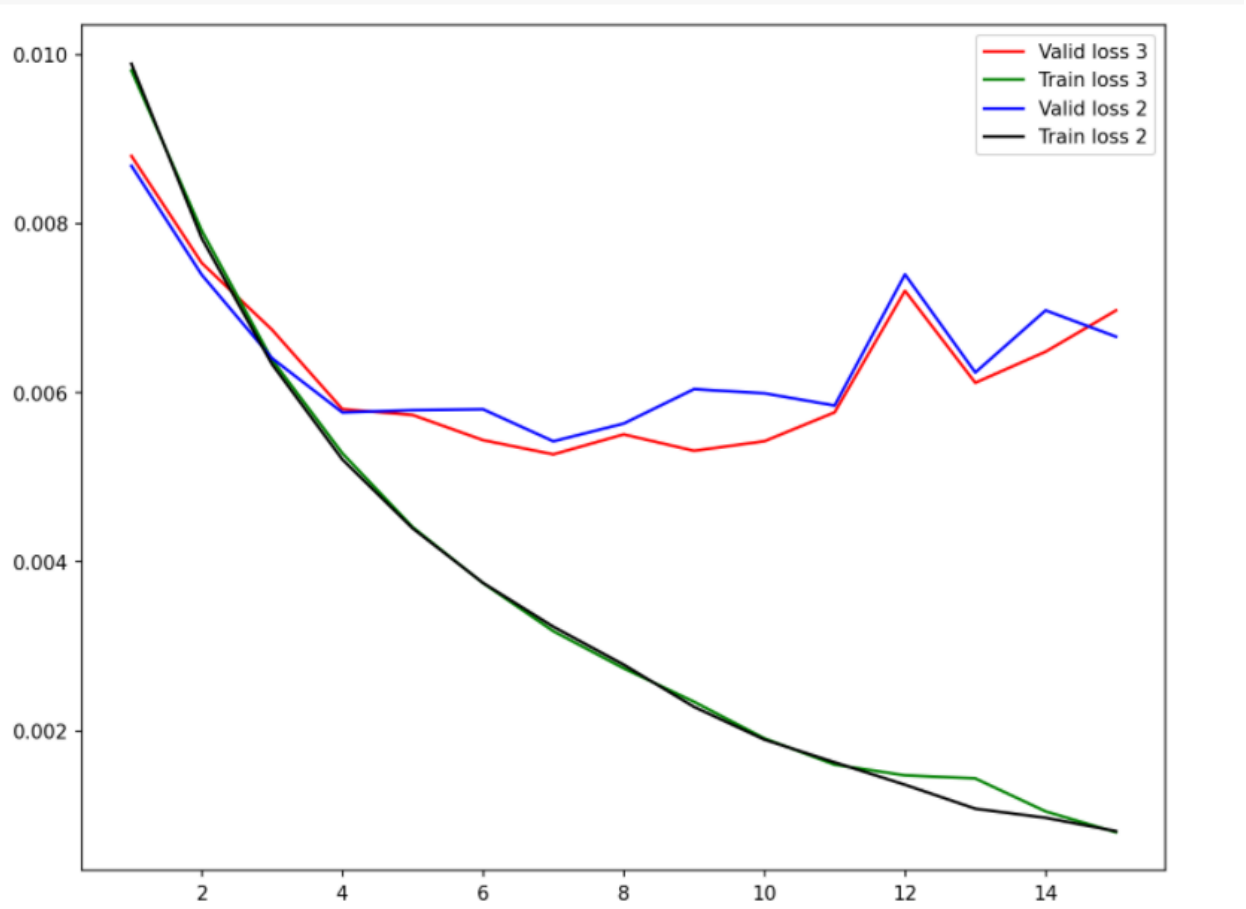
Through our above results we can see that they do. We assume that this is the case as it is binary.
We then used pytorch to build 9 RNN models to improve upon keras these model which we implemented were

- RNN 0: A baseline RNN model with 1 embedding layer with embedding dimensionality of 50, 1 RNN layer of 1 RNN cell of hidden dimensionality 64, 1 dropout layer with p=0.1 and a fully connected layer followed by a sigmoid then a squeeze
  `Epoch [15/15], Train Loss: 0.0022, Train Acc: 0.9611, Valid Loss: 0.0063, Valid Acc: 0.8701`
- RNN 1 was the same as baseline but it used GRU cell instead
  `Epoch [14/15], Train Loss: 0.0008, Train Acc: 0.9865, Valid Loss: 0.0059, Valid Acc: 0.9028`
- RNN 2 was the same as baseline but it used LSTM cell instead
  `Epoch [11/15], Train Loss: 0.0016, Train Acc: 0.9660, Valid Loss: 0.0058, Valid Acc: 0.8875`
- RNN 3 was the same as baseline but it used a bidirectional LSTM cell instead
  `Epoch [15/15], Train Loss: 0.0008, Train Acc: 0.9859, Valid Loss: 0.0070, Valid Acc: 0.8890`
- RNN 4 was the same as baseline but it used LSTM cell and used a GLoVe pre-trained word embedding (glove.6b.50) that was frozen
  `Epoch [14/15], Train Loss: 0.0047, Train Acc: 0.8747, Valid Loss: 0.0057, Valid Acc: 0.8454`
- RNN 5 was the same as baseline but it used LSTM cell and used a GLoVe pre-trained word embedding (glove.6b.50) that was fine tuned for this model
  `Epoch [8/15], Train Loss: 0.0012, Train Acc: 0.9749, Valid Loss: 0.0037, Valid Acc: 0.9343`
- RNN 6 was the same as baseline but it used LSTM cell and used a random vector as a word embedding and was frozen
  `Epoch [15/15], Train Loss: 0.0044, Train Acc: 0.8857, Valid Loss: 0.0084, Valid Acc: 0.7950`
- RNN 7 was the same as baseline but it used LSTM cell and used a normal xavier weight initialization method
  `Epoch [13/15], Train Loss: 0.0004, Train Acc: 0.9934, Valid Loss: 0.0051, Valid Acc: 0.9307`
- RNN 8 was the same as baseline but it used LSTM cell and used a normal kaiming weight initialization method
  `Epoch [13/15], Train Loss: 0.0005, Train Acc: 0.9921, Valid Loss: 0.0055, Valid Acc: 0.9179`

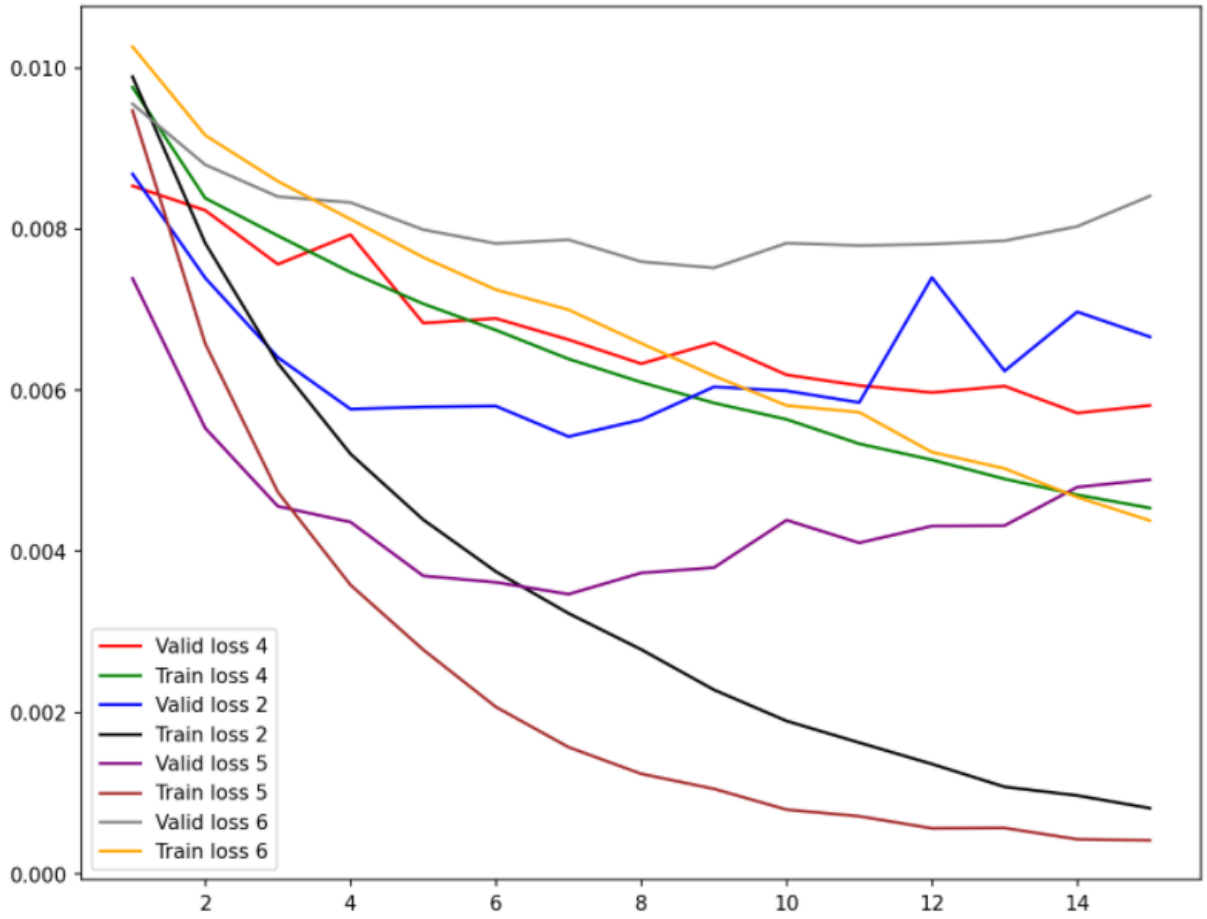
Plots of RNN Train and Validation Losses for every epoch:



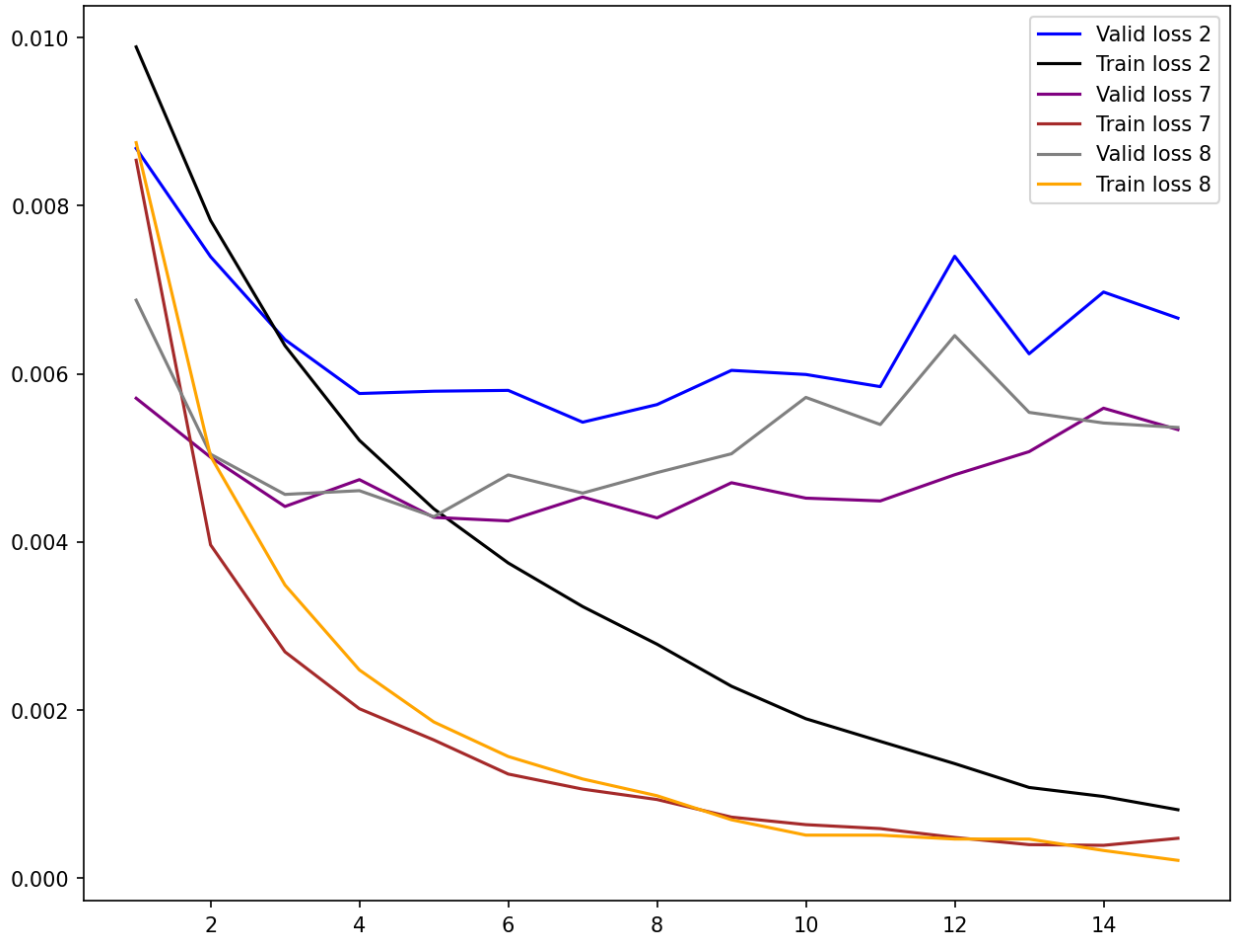
This showcases that Model 2 is the best and that RNN is weaker than GRU and very weak in comparison to LSTM



The difference between models 2 and 3 is insignificant at this level meaning bidirectionality did not matter

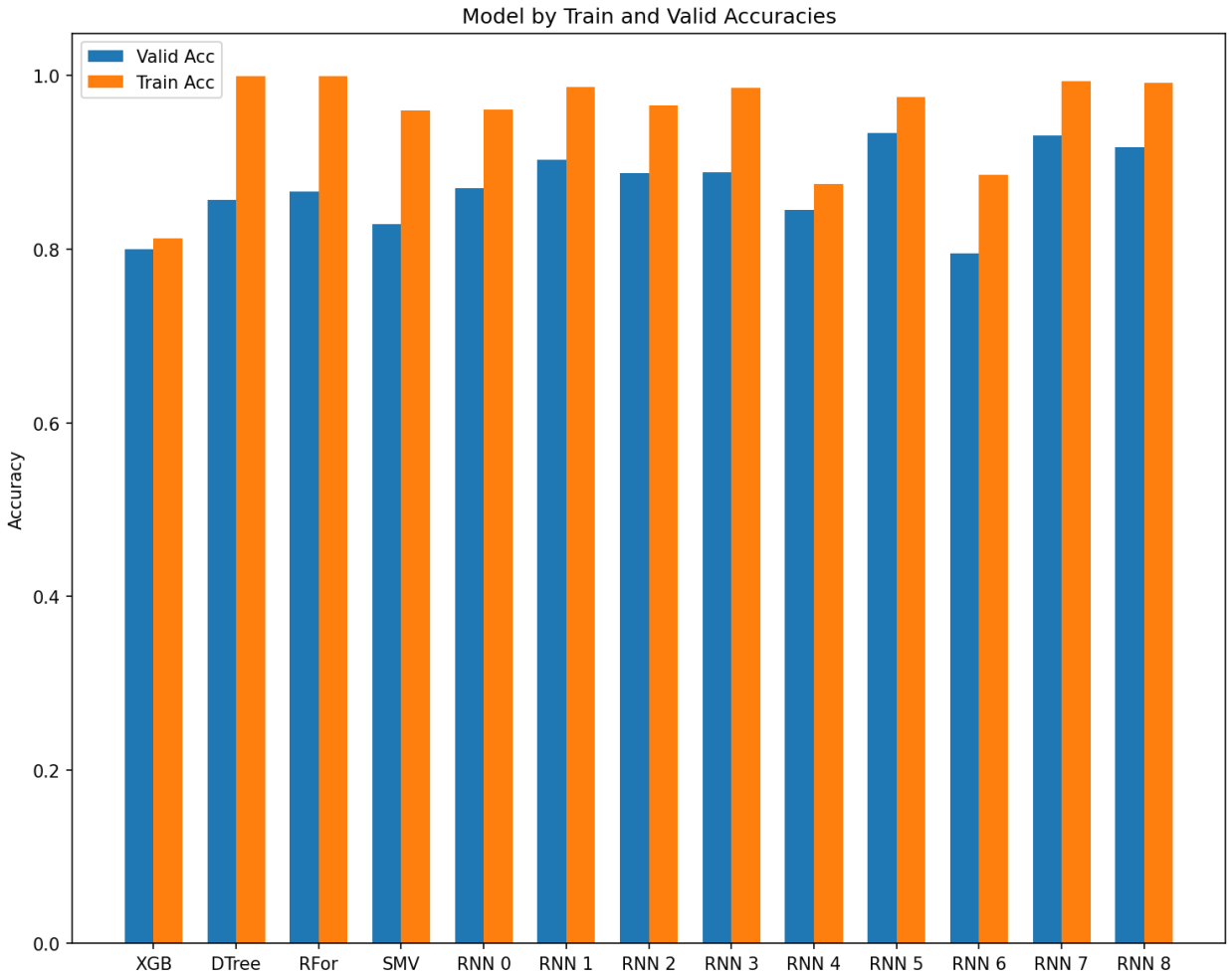


Model 5 performs exceptionally well due to both being pre trained and being fine tuned to the model



Showcases that initialization is definitely helpful

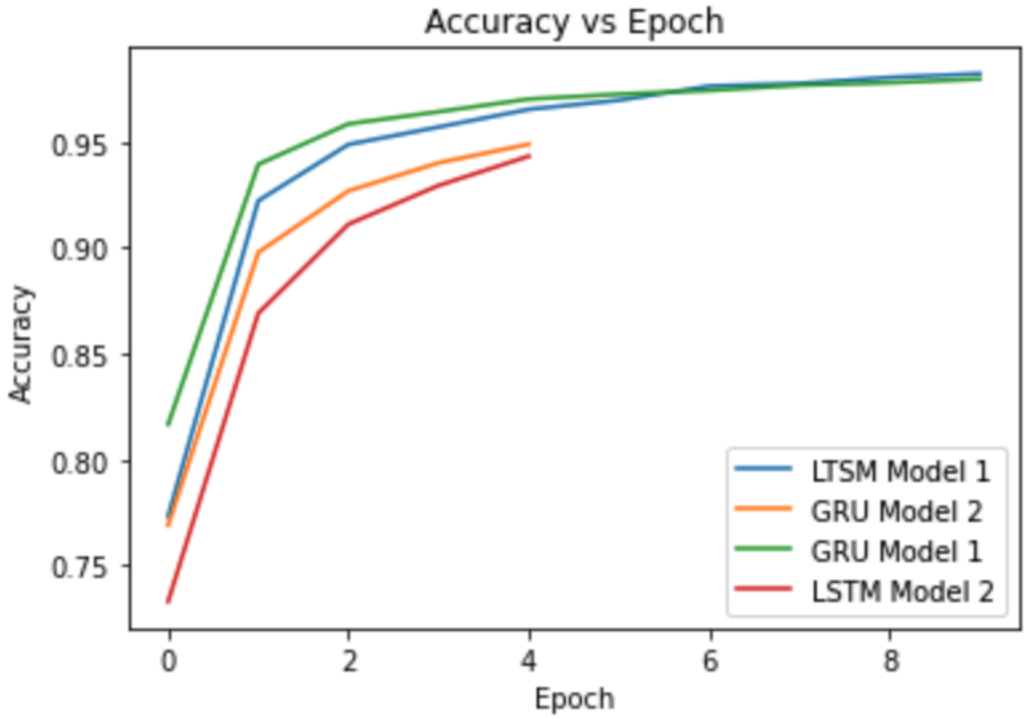
Final Result considering Accuracy using bar charts
Bar chart showing difference in train and validation accuracies with Binary classification



Above is for binary classification

Through our bar charts we can see that binary classification is more accurate as this probably as there is less leeway to make mistakes. Models RNN 5, 7, and 8 were considerably the best with model RNN 5 having the highest accuracy

Multiclass Model Winner:



The LSTM Model 1 was by far the best in terms of validation accuration which peaked at 91% and followed by GRU Model 1 with 89% validation accuracy.

Model 1's won because they did not have further noise introduced by the data from different dimensions.

Video Presentation

Please click on the link provided below to direct you to our video presentation.

https://youtu.be/ZgsfceAYnNc

References:

1. https://github.com/sharmaroshan/Twitter-Sentiment-Analysis/blob/master/Twitter_Sentiment.ipynb
2. https://www.kaggle.com/ivxn99/simple-tensorflow2-keras-classification
3. Spring 2021 COMP 4211 PA3 of Dhulan Seniru Daluwatte 20553900
4. Spring 2021 COMP 4211 PA3 of Abhishek Poswal 2020557578