Identifying Hate Speech in Social Media

Member 1	Bonthu Harsha Vardhan Reddy	harshareddy97@vt.edu
Member 2	Anudeep Kumar Reddy Guntaka	ganudeepreddy21@vt.edu
Member 3	Bharath Kusuluru	bharathk@vt.edu

Introduction:

The boom in social media has been fueled by the human impulse to communicate, socialize and form personal connections. It has helped us create interconnected worlds, where people are free to express themselves, and have conversations about a myriad of topics. However, this has also led to an increase in toxic, disrespectful, and often lewd remarks. Using social media as a curtain of anonymity, people continue to pour their hatred, dissent, pernicious and obscene comments with less fear of consequences. Therefore, it is essential to root out such toxic content, especially considering that most of the young people use social media on a regular basis. We should prevent consumption of toxic media by children or teens as that can negatively impact their mental health. Our project aims to make social media a safer space for people to share their opinions in a respectful way and ensure that the derogatory content is flagged and removed accordingly.

Project Problem Statement:

Our aim in this project is to identify if a comment is toxic or not and flag toxic comments for removal. Toxic remarks are ubiquitous in Facebook pages, Instagram and YouTube comments, tweets and reddit threads. This will be extremely beneficial to social media companies (Facebook, YouTube, Tiktok) since it bypasses the need for someone to manually scrape out toxic content; our algorithm will help them do it in real-time. Researchers and social scientists can use the algorithm to identify patterns of hate speech and individuals who try to inflict them. Our model should work for generic conversations and using our machine learning algorithms we will eventually be able to build a robust system that can predict the level of toxicity on future unseen instances.

Dataset:

This project will utilize the "**Jigsaw Unintended Bias in Toxicity Classification**" dataset from Kaggle.com found at the following location: https://www.kaggle.com/c/jigsaw unintended-bias-in-toxicity-classification/. Jigsaw sponsored this effort and provided annotation of this data by human raters for various toxic conversational attributes. This dataset was extracted from the Civil Comments platform and public comments from their platform were available between September 2015 and November 2017. This data is stored in comma separated values (CSV) format consisting of 1.8 million records across 46 features. The features which provide the most valuable insight here are id, comment_text, and target.

Table -1 : **Representing the Dataset**

Feature	**Type**	**Description**
id	Numerical	The id of the comment (e.g.: 5764010)
comment_text	Categorical	Text of the individual comment (e.g.: This is a very silly plan)
target	Numerical	If the value >=0.5, then the comment is toxic, otherwise non-toxic

Pre-processing:

The most important attribute for us is "comment_text" upon which we will apply our natural language processing and try to classify if it's toxic or not. The first step of pre-processing will be cleaning the comment_text such that our machine learning algorithm will be able to understand it.

1) Exploratory Data Analysis:

While performing the Data Analysis we found that our dataset is a bit skewed in terms of the toxic and non-toxic comments.

The total number of records in the dataset = **1,780,823**

The number of Toxic records = **106,438**

The number of Non-toxic records = **1,674,385**

There are a total 46 features, some attributes containing sub-levels of toxicity mentioned as: [severe_toxicity, obscene, threat, insult, identity_attack, sexually_explicit]. Since these subcategories would have higher correlation to toxicity, using them for our classification would defeat the purpose. We can perhaps in the next iteration of the project try to classify the comment into further subcategories. There are other identity columns as well such as 'atheist', 'buddhist', etc. However, they're too sparse to be useful. We also analysed the column "comment_text" and used a boxplot to identify the range of number of words in a given comment. The mean = **51.35**, median = **35**. So, we decided that a good sequence length would be **50**. Therefore, we as of now will only be using "comment_text" for classification of toxicity.

2) Cleaning attribute comment_text:

a) Remove special characters: We will remove any characters like &, @, and so on. b) Remove Non-ASCII characters: We will remove all the unnecessary Unicode characters apart from the standard ASCII-set

c) Remove Numbers: It's hard to ascertain correspondence with numbers and toxicity, so we will remove it

d) Remove punctuation: We will remove punctuation like ',', '!' and so on

Remove whitespaces: We will trim the whitespaces

f) Convert to lowercase: To make sure our model case insensitive

g) Remove stop words: We will use nltk's library of stop-words for English and remove them as they are not much useful here

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

3) Solving the imbalanced Dataset issue:

Imbalanced classifications pose a challenge for predictive modeling as most of the machine learning algorithms used for classification were designed around the assumption of an equal number of examples for each class. This results in models that have poor predictive performance, specifically for the minority class. As found out in the data analysis, our dataset is skewed towards non-toxic comments. In terms of percentage, non-toxic is 94% of the dataset and toxic is 6%. Every classification algorithm will then be biased, so we can resolve them in the following ways:

a) **Oversampling**: We try to duplicate the records of our toxic data and try to match it to the non-toxic. This will create artificially low variance for our toxic class. Also the number of records will be too high and our ML models may consume lots of time for it.

b) **Under sampling**: Here we take only a sample of the non-toxic data (equal or proportionate to the toxic one) and therefore, in this smaller dataset we can run our ML models. This will lead to a loss in data, which can underwhelm our model, but given that the total count of records will be greater than 200,000 it should be enough.

c) **Combining under sampling and ensemble**: This can be a better implementation, where we divide the non-toxic data into subsets of the size of toxic data. Then train our model multiple times, each time taking 1 subset from the non-toxic and using the toxic dataset. We will use the following technique:

Synthetic Minority Oversampling Technique (SMOTE):

SMOTE works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample at a point along that line. This will increase our sample size from 200K to 500K. Although the number of minority samples will be less than that of the majority, we have effectively increased the dataset size.

4) Feature Generation:

We generated feature space from sequences of words by using TF-IDF. We used max features as 10000. So, we get a feature space of shape 10000 for each instance.

For the LSTM, we can generate the embedding layer by the following methods:

a) Self-train Embedding layer: The embedding is learnt along the model during the training phase.

b) Transfer learning:
We used the pre-trained word vectors which were trained on 2 billion tweets with 27 billion tokens, known as Glove.twitter.27B embeddings.
These embeddings contain weights that were trained on a huge dataset, which means that they might contain useful information based on textual parsing. Given the context of toxic words classification, we can train this layer further on our dataset to improve the quality of our embeddings.

Methods and Models:

This is a binary classification problem and there are multiple machine learning algorithms which are good at it. However, we should also consider that we are solving the problem using natural language processing, so we should use algorithms which are good at textual classification.

1) Naive Bayes:

Among the different Bayesian algorithms, we will choose Bernoulli's Naive Bayes as it is empirically known to perform better with binary classification. There isn't much hyper tuning required, as Naive Bayes is a simple algorithm.

2) Support Vector Machine (SVM)

Since this is an NLP problem, the decision boundary will likely be non-linear. Therefore, we are not using linear SVM. SVM will take lots of time to train. (Ideally it would be more than 10 hours given our current laptop specifications).
So we will take a smaller sample (sub-set) and apply SVM. We will only take 40,000 records and perform the training as well as testing.
We used the following hyperparameters:
a) C (regularization parameter) = 1.0
b) kernel = 'rbf' (radial basis function)

3) Decision Trees:

There are multiple hyperparameters to tune over here and we have explored the following ones:
a) max_depth = [200, 500, 1000]
b) criterion : [Gini, entropy]
c) max_features: [default=None, auto]
d) min_samples_split = [default=2, 5]

4) Random Forests:

Random Forests use an ensemble technique, so we can expect an improvement from

Decision Trees. But given the power constraints, we weren't able to try many hyperparameters. We used the following:

a) n_estimators = 500

b) random_state = 13

5) Logistic Regression

Logistic regression is simple and efficient when applied to textual data, one thing we need to ensure is that there are enough iterations for the optimizer 'lbfgs' to converge. We apply it both for the normal data and SMOTE data, and compare the results. Since the number of minority classes is nearly half of that of the majority class, SMOTE doesn't perform that well as expected given the increase in size.

a) max_iter = 10,000

5) Long-Short-Term Memory (LSTM):

LSTM is an artificial recurrent neural network, which has feedback connections. This will be useful during parsing of textual semantics as it can understand the context better. For the self-train embeddings, We generate the feature representation of each sequence by using a self-train embedding layer, which generates the feature vector of length 10000 (50 x 200) [where 200=embedding dimension, 50=sequence length]. We use an LSTM layer with 50 units and in the final output layer we have an output node with sigmoid activation that generates a value between 0,1. We have used the following hyperparameters:

a) optimizer: adam ('adam' is the empirically best optimizer function)

b) loss_function: binary_crossentropy

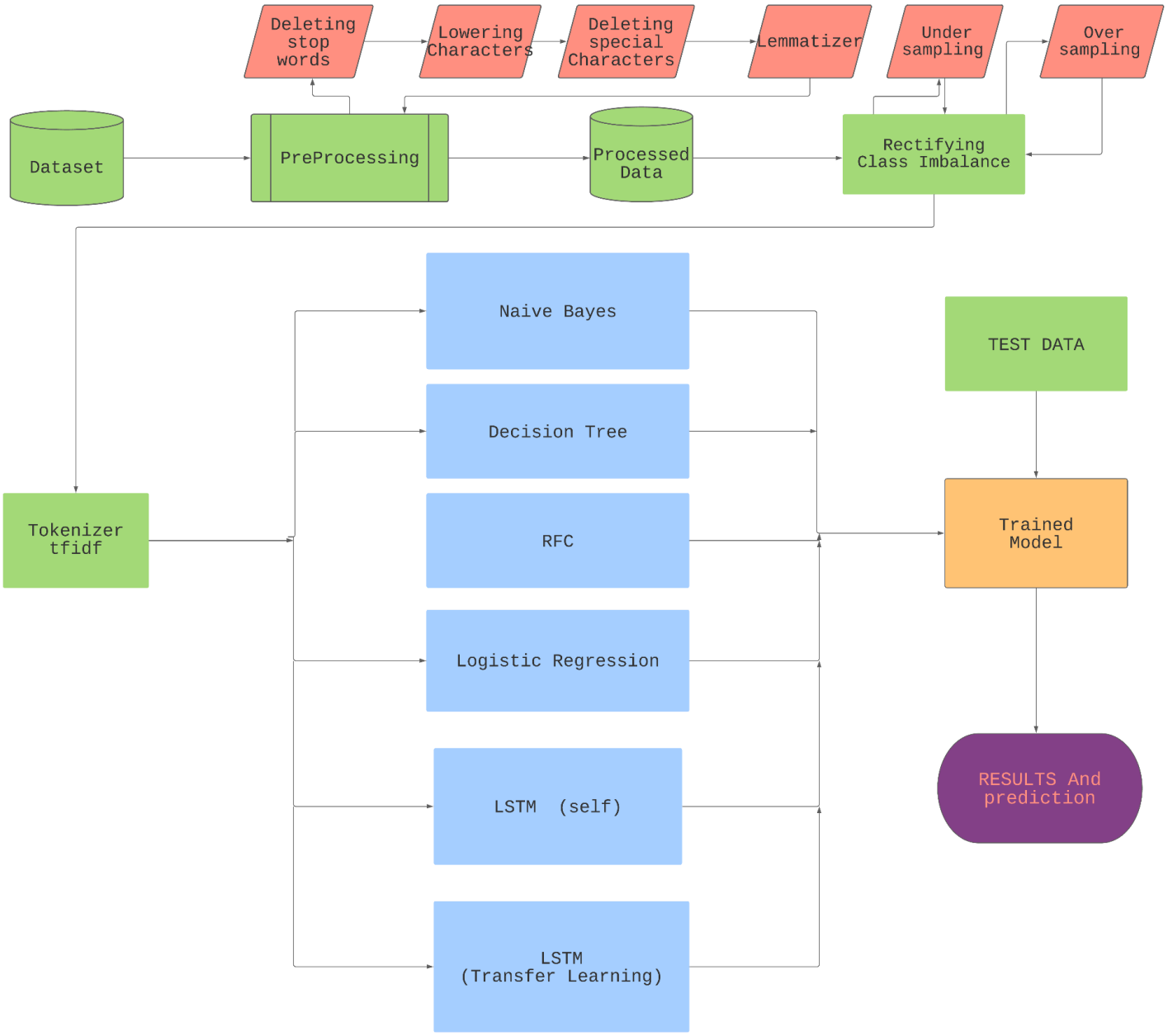
c) batch_size: 64

d) epochs: 10

e) learning rate = 0.01

f) activation function = 'sigmoid'

ML PIPELINE:



Results:

Table 2 - Table Representing the results of the different Classifiers

Sample Size	Classifier Name	Hyper parameters	F1 score Mean	F1 score standard Deviation
Under sample	Naive Bayes	N/A	**0.7544**	0.0018
Over sample	Naive Bayes	N/A	0.7107	0.0047
Under sample	SVM	Kernel = 'rbf'	**0.8392**	0.0008
Under sample	Decision Tree	max_depth = 200	0.8515	0.00169
Under sample	Decision Tree	max_depth = 500 min_samples_split = 5	0.8534	0.00215
Under sample	Decision Tree	max_depth = 500 max_features = auto	0.7944	0.00608
Under sample	Decision Tree	max_depth = 500 max_features = auto criterion = entropy	0.7970	0.0051
Under sample	Decision Tree	max_depth = 1000 max_features = auto	0.7937	0.0060
Under sample	Decision Tree	max_depth = 200	0.8521	0.0021
Under sample	Decision Tree	Max_depth = 200	**0.8605**	0.0136
Under sample	Random Forest	n_estimators = 500	0.8893	0.0013
Under sample	Logistic Regression	N/A	**0.9059**	0.0001
Over sample	Logistic Regression	N/A	0.8543	0.0261
Under sample	LSTM	Embeddings = 'self trained' Embedding_vector length = 200 Optimizer = 'adam'	**0.9107**	N/A
Under sample	LSTM	Embeddings = 'transfer learning'	**0.9133**	N/A

Model	**Area Under Curve (AUC)**
SVM	0.8447
Random Forest	0.8887
LSTM self train embedding	0.9692
LSTM with transfer learning	**0.9717**

We are using the following metrics to evaluate the performance of our models:

1) **Cross-Validation**: We are using cross-validation, which is the preferred way over the train_test_split. We are using the cv = 5, which means that 80% will be used for training and 20% will be used for testing.
2) **Receiver Operating Curve (ROC)**: Since our dataset is imbalanced, even after performing rectification, the ROC curve is a good metric to compare the performance of our deep learning models. The Area under curve (AUC) for the LSTM is displayed.
3) **F1-Score**: This is a more robust measure than accuracy, when the preference is for the positive class. We have displayed both the mean and standard deviation of the F1-scores.

Conclusion:

We have visualized certain instances of our toxic and nontoxic words using wordClouds. In the toxic comments, when we visualize it there isn't a plethora of bad words, which was generally expected. So, we can attest that the model we are building isn't a trivial one that just considers the presence or absence of toxic words, but rather builds upon the context of it.

We have run a total of 5 different ML algorithms starting from the most simple Naive Bayes. Naive Bayes is generally expected to run well on textual data, because it has a tendency to capture the presence of core factors (here the toxic word vector). The shortcoming of Naive Bayes is that it fails to capture the context of the occurrence of the words.
The reason we suspect to have gotten a low score on the under and oversampled synthetic data is because the number of the majority class is twice that of the minority class, affecting the Bayesian probability. We got a mean F1-Score of **0.75**, which forms our basis for comparison with other algorithms.

SVM is known to be effective, however in the TF-IDF vectorizer, we are generating 10,000 top features, which makes our data very-high dimensional. Coupled with the fact that we have more than 200K records and computer resources at hand, we ran for a limited 40K records. SVM performed decently well with a mean F1-Score of **0.839**.

Next we planned to run Decision Trees, which have the capability of identifying strong attributes. This will be helpful to identify the toxic attributes while classifying the record. We performed a plethora of hyperparameter tuning in Decision Trees. For 200 max_depth, we were able to get good scores. We tried using max_features to see if there was an improvement in F1 Score. However, it took a dip, but the model trained quickly. So we increased the depth to 500, but there

wasn't any significant improvement. So we tried using min_samples_split and changing the criterion to 'entropy' and got an improvement but it was very slight. Then we bumped up the depth to 1000, still there wasn't much improvement.
So we decided that for max_depth=200, we have got a reasonably good model. Then, we bumped up size by using the one from SMOTE and got the F1-score in cross-validation to be **0.86.**

Ideally, the next step would be to go to Random forests, since they're an ensemble technique and use Decision trees for their base classifier. There is a huge scope for hyperparameter tuning, however we are significantly limited by resources to do a GridSearchCV on this. So we chose the n_estimators to be 500 for our Random Forest and got a good F1-Score of **0.89.**

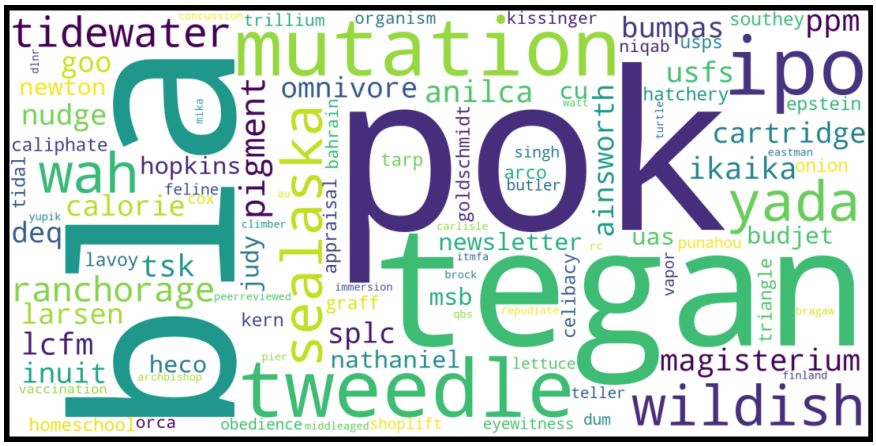
Then we took a turn towards Logistic regression, which is empirically known to perform well for text data sets. There isn't any hyperparameter tuning to be done. LR performed extremely well, which took us a bit by surprise, but we believe it's able to capture the toxic words' probability distribution quite well. We got a mean F1-score of **0.90.**

For textual data prediction, deep learning has been the industry standard, which we wanted to test and replicate. LSTM is known to perform well on sequential data given it's capability to remember the context of the words. We have used an empirically known 'adam' optimizer that converges faster. We used two models, first where we trained the model with a self-trained embedding layer and achieved a high F1-score of **0.91.**
In the second model we leveraged the power of transfer learning, we retrained the model using the Glove embedding layer prepared from Twitter data. We achieved a minor increase in F1-Score which resulted in our best F1 score **0.913**, an increase of **0.3%** from LSTM.

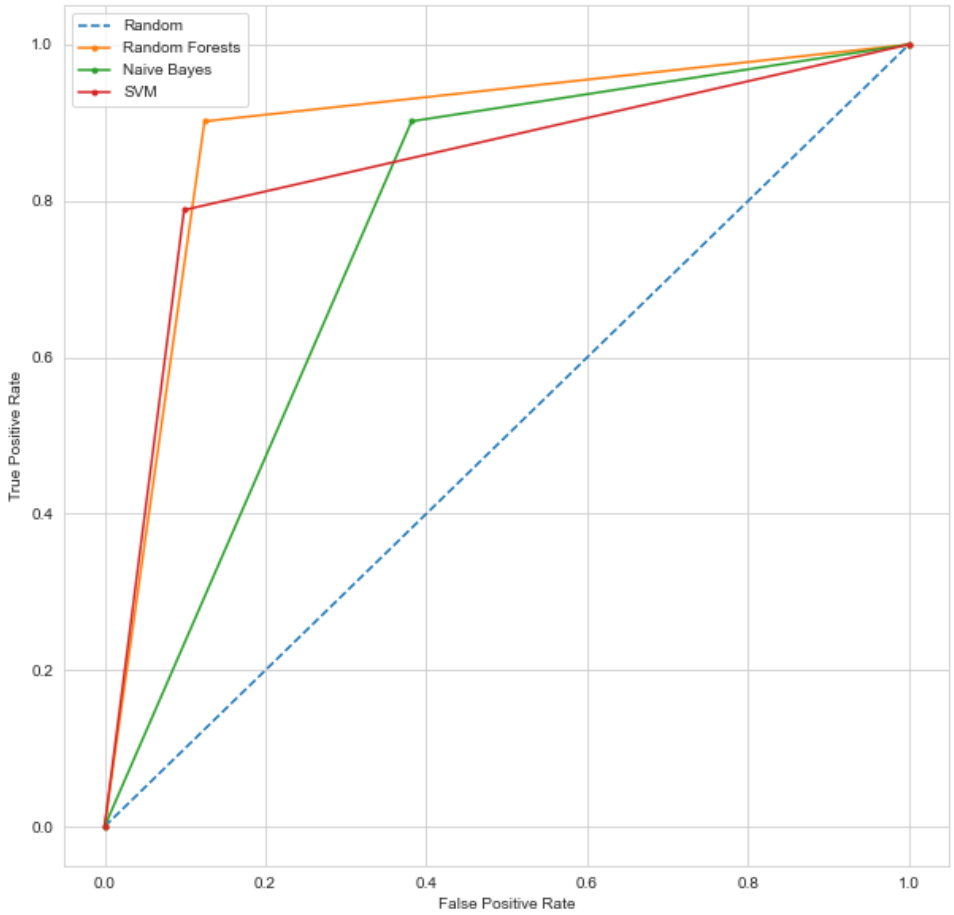
INDIVIDUAL CONTRIBUTION :

Member 1	Bonthu Harsha Vardhan Reddy	• **Exploratory Data Analysis** • **Solving Class Imbalance** • **Decision tree and Random Forest** • **Ablation testing and Analysis**
Member 2	Anudeep Kumar Reddy Guntaka	• **Naive Bayes** • **SVM** • **ML Pipeline** • **Results performance and analysis**
Member 3	Bharath Kusuluru	• **Cleaning the text** • **Feature Generation** • **LSTM and Logistic Regression** • **Conclusion and Analysis**

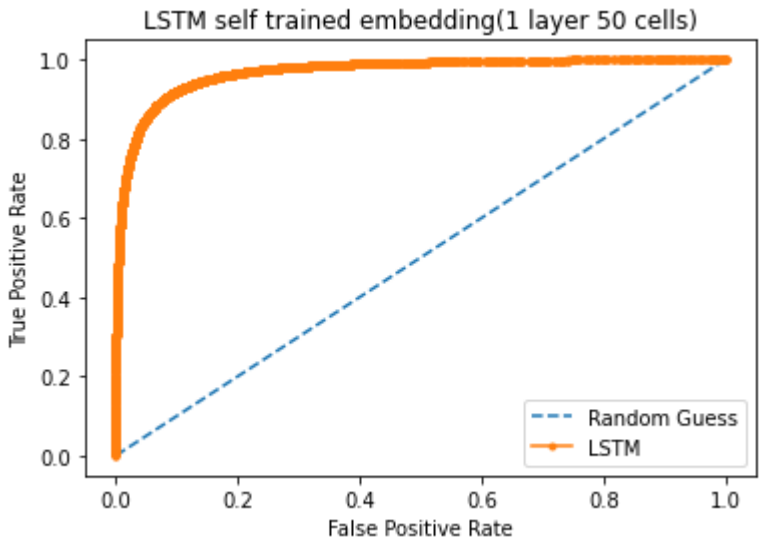
APPENDIX:



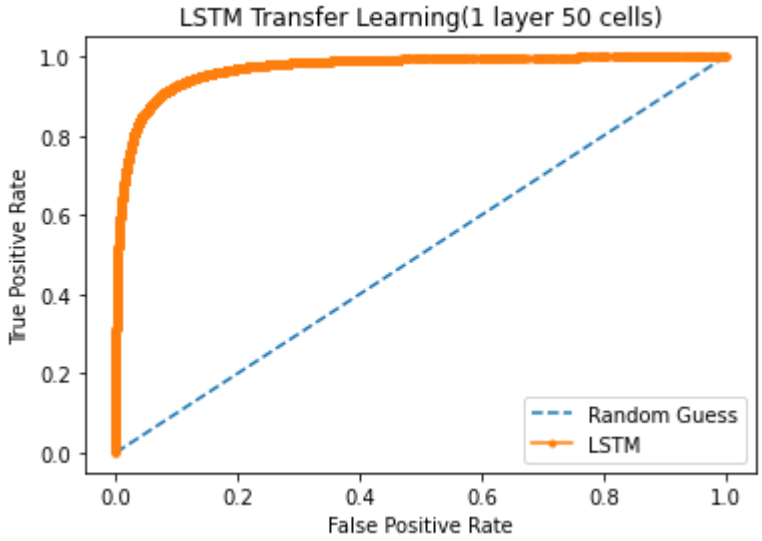
Picture 1: Word Cloud for top features



Picture 2: ROC Curves for ML Algorithms (for a reference)



Picture 3: LSTM with self-training embeddings ROC curve



Picture 4: LSTM with transfer learning embeddings ROC curve