# Determination of Toxic Comments and Analysis for the Minimization of Unintentional Model Bias

Md Shahidullah Kawsar
*kawsar@mail.usf.edu*
University of South Florida, Tampa, USA

**Abstract:** Online conversations can be toxic and subjected to threats, abuse, or harassment. Online conversation toxicity can be defined as rude, disrespectful, make somebody leave a discussion, stop expressing oneself, and even give up on looking for different opinions. To help improving online conversation and analyze the negative online behaviors, the goal of this project is to build a deep learning model that can perform binary classification, determine toxic and non-toxic comments, and analysis of minimization of the unintended bias concerning identity features such as race, gender, sex, religion, and disability, etc. The proposed Logistic Regression model with the TFIDF vectorizer achieved 94.5% accuracy, and for a small dataset, GRU, and LSTM model achieved 100% accuracy.

**Keyword: NLP, RNN, GRU, LSTM, TFIDF**

## I. Introduction:

The dataset is collected by the Conversation AI [1] team, a research initiative founded by Jigsaw [2] and Google organized a Kaggle competition in 2019: Jigsaw Unintended Bias in Toxicity Classification [3]. The dataset was collected from Civil Comments platform from 2015 to 2017. The dataset is open for the researchers in order to understand and improve the civility in online conversations. Jigsaw sponsored the efforts made by Civil Comments platform and extended by adding toxicity subtype attributes and identity variables through human raters.

When the Conversation AI team first built toxicity models, they found that the models incorrectly learned to associate the names of frequently attacked identities with toxicity. Models predicted a high likelihood of toxicity for comments containing those identities, e.g. 'gay', even when those comments were not actually toxic such as "I am a gay woman". This unintended model bias occurred because training data was pulled from the available sources where unexpectedly, certain identities are overwhelmingly referred to in offensive ways. Training a model from the data with these imbalances risks simply mirroring those biases back to users. Using a dataset labeled for identity mentions, toxicity subtype attributes and number of different reactions of the comments, the goal is to optimize a metric which can measure unintended bias and develop strategies to reduce unintended bias in machine learning models, and also build models that can work well for a wide range of online conversations and discussions. The primary goal of this project is to design an accurate supervised and deep learning model that can perform binary classification in order to determine toxic and non-toxic comments.

## II. Dataset Information:

The dataset contains 1.8 million observations and 45 features. The "comment_text" column contains the comment from diverse range of conversations and "target" column indicates how toxic a comment is and target value $\geq 0.5$ means comment is toxic. Additional toxicity subtype attributes are severe toxicity, obscene, identity attack, insult, threat, and sexually explicit. Reaction variables are funny, wow, sad, likes, disagree. Identity variables can be classified into five categories.
Gender: male, female, transgender, other gender
Sex: heterosexual, homosexual gay or lesbian, bisexual, other sexual orientation
Religion: Christian, Jewish, Muslim, Hindu, Buddhist, atheist, other religion
Race: black, white, Asian, Latino, and other race or ethnicity
Disability: physical disability, intellectual or learning disability, psychiatric or mental illness, other disability

Toxicity labels were obtained from the human raters. Each comment was rated by 10 to thousand raters. Raters marked each comment as very toxic, toxic, hard to say or no toxic. They also tried to guess the gender targeted in the comment etc. Here are some examples of the dataset.

Comment: *"haha you guys are a bunch of losers."*
Target Label (Toxicity): 0.89
Severe toxicity: 0.02
Identity attack: 0.02
Insult: 0.87
All others: 0.0

Comment: *"The woman is basically a slave."*
Target Label (Toxicity): 0.83

Identity attack: 0.83
Insult: 0.83
Female: 1.0
All others: 0.0

Comment: *"I love the idea of upvoting entire articles."*
Target Label (Toxicity): 0.0
All others: 0.0

## III. Exploratory Data Analysis

Fig 01 represents the distribution of the target variable which has two categories such as non-toxic (0) and toxic (1). Target variable < 0.5 belongs to non-toxic and target variable ≥ 0.5 belongs to toxic comments. From fig 01, we can see that the dataset is highly imbalanced. Among 1.8M observations, 92% (1.66M) of the data belongs to non-toxic and only 8% (0.14M) of the data belongs to toxic comments.

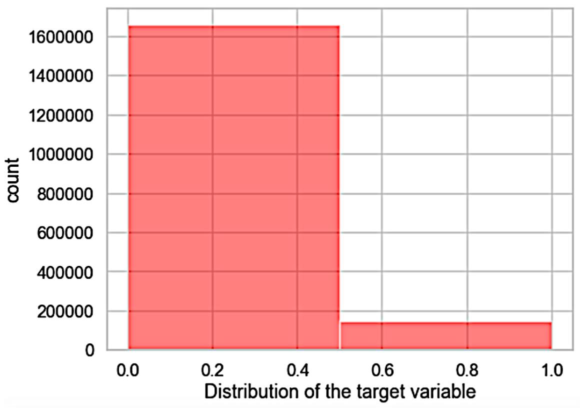


Fig 01. Distribution of the target variable. Target variable < 0.5 belongs to non-toxic (0) and target variable ≥ 0.5 belongs to toxic (1) comments.

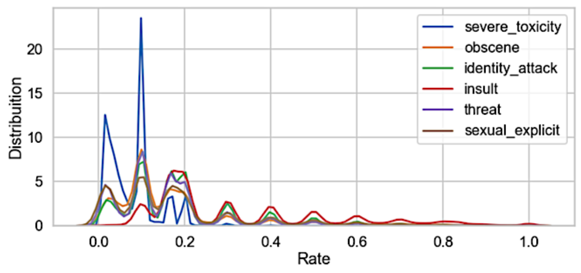


Fig 02. Distribution of the toxicity subtype attributes.

Fig 02 represents the distribution of the toxicity subtype attributes such as severe toxicity, obscene (offensive or disgusting), identity attack, insult, threat and sexually explicit (sexual harassment). Most of the attributes belong to the value less than 0.5.

Fig 03 represents the time series analysis of the ratio of toxic comments and gender identity variables such male, female, transgender, and other gender from 2015 to 2017. The dataset contains news articles from different sources. From the fig 03, we can see that transgender people are mostly targeted in the online toxic comments. We can also observe that there is a sharp increase of toxic comments regarding transgender people during July 2017 maybe because US President Donald Trump twitted about a ban on transgender people serving in the military. []

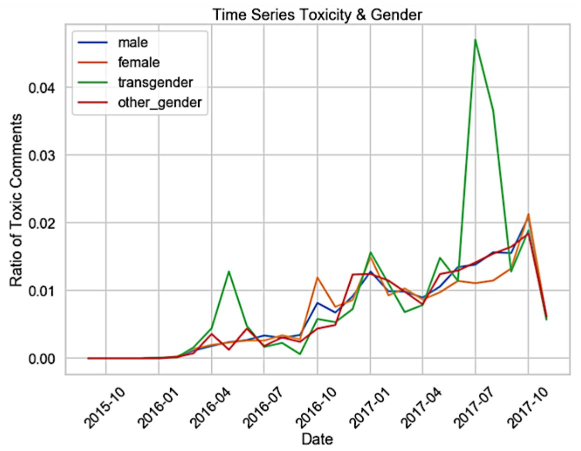


Fig 03. Time series analysis of the ratio of toxic comments and gender identity variables.

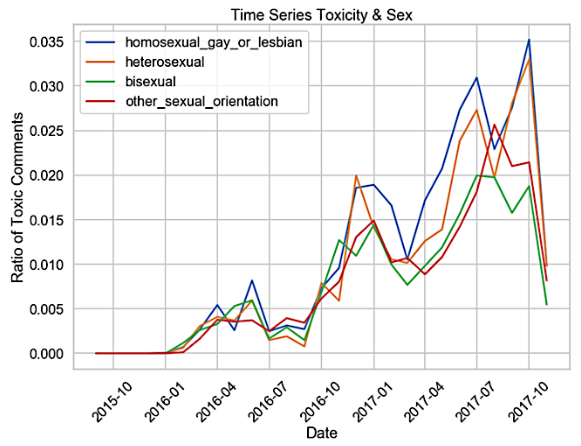


Fig 04. Time series analysis of the ratio of toxic comments and sex identity variables.

Fig 04 represents the time series analysis of the ratio of toxic comments and sex identity variables such homosexual gay or lesbian, heterosexual, bisexual, and other sexual orientation. Before the shutdown of the Civil Comments, at October 2017, the ratio of toxic comments targeting homosexual gay or lesbian, and heterosexual reached maximum.

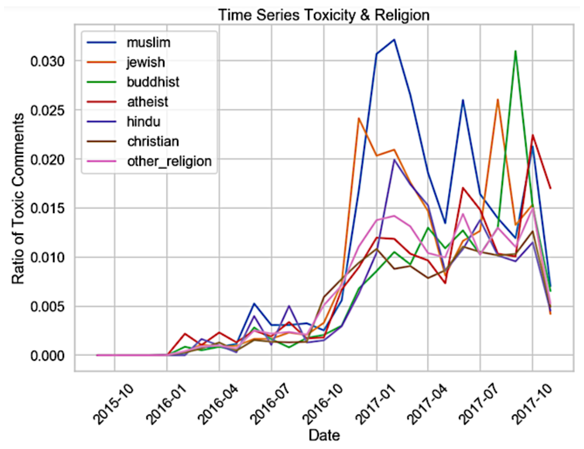


Fig 05. Time series analysis of the ratio of toxic comments and Religion identity variables.

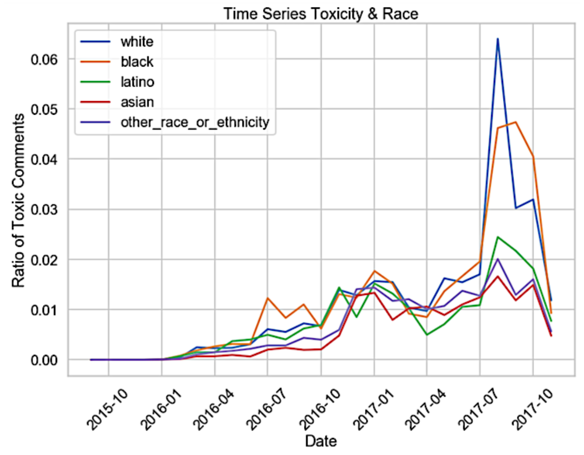


Fig 06. Time series analysis of the ratio of toxic comments and Race variables.

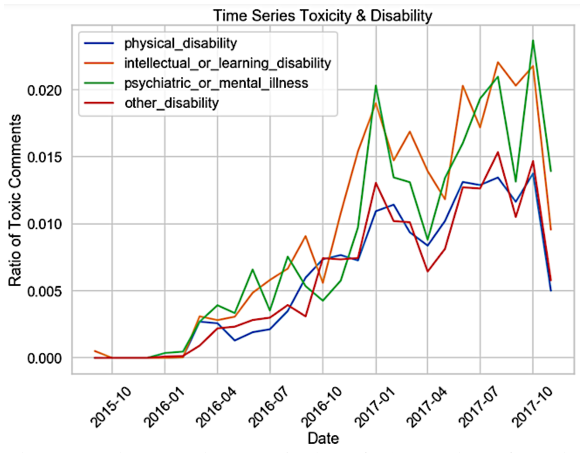


Fig 07. Time series analysis of the ratio of toxic comments and Disability variables.

Fig 05 represents the time series analysis of the ratio of toxic comments and religion related identity variables such Christian, Jewish, Muslim, Hindu, Buddhist, atheist, other religion. From the fig 05, it's clear that Muslim people were highly targeted for toxic comments in online conversations maybe because of the current global fear of extremism.

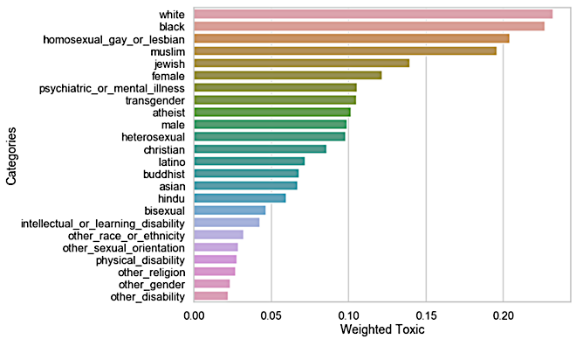


Fig 08. Percent of toxic comments related to different identities using target and population amount of each identity as weights.

Fig 06 represents the time series analysis of the ratio of toxic comments and Race identity variables such as white, black, Asian, Latino, and other race or ethnicity. It's still surprising that people live in modern society and also target human beings by their color especially the white and black.

Fig 07 represents the time series analysis of the ratio of toxic comments and disability identity variables such as physical disability, intellectual or learning disability, psychiatric or mental illness, and other disability.

From fig 08, among these 24 identity variables, the most targeted identities are white, black, homosexual gay or lesbian, Muslim, Jewish, female etc. Here, for each observation we have a value of target variable which represents how toxic the comment is. Each identity variable has also a value between 0 to 1 to identify how much they have been targeted. Using these two aspects, we can find which identities are more frequently related to toxic comments. To determine the weighted toxicity, first calculate the sum of the product of each identity variable with the target variable. Then, find the ratio of this sum of product to the number of identity variables which are greater than zero. For example, for the identity variable 'white':

$$\text{Weighted toxic} = \frac{\sum_{i \in N} identity\ variable_i \times target\ variable_i}{Number\ of\ identity\ values\ greater\ than\ 0} = \frac{6946.94}{29948} = 0.23$$

Where N is the total number of observations.

In order to understand the relation between different variables with the target variable, heatmap of correlations has been generated in two steps. First, a heatmap is created at fig 09 to take a look at the pairwise correlations between target variable and the reaction variables in the comments such as sad, wow, funny, likes, disagree. The strongest correlation value 0.34 exists between 'sad' and 'disagree' maybe because of the content of comments that people disagree, make them sad as well. Looking at the correlation to the target variable, the correlations are very weak. Maybe the reason is the data is collected from the Civil Comments platform from 2015 to 2017 and these data are not coming from popular social networking sites such as Reddit, Facebook, Instagram or Twitter where reactions are very popular. This can be an explanation for the low number of reaction votes and the weak correlation.

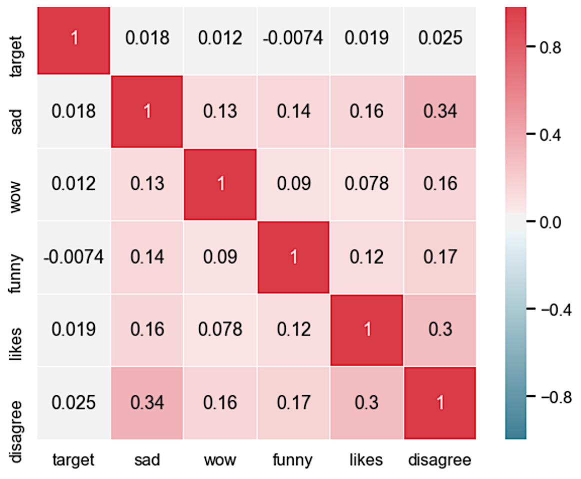


Fig 09. Correlation heatmap of target variable and the reaction variables in the comments.

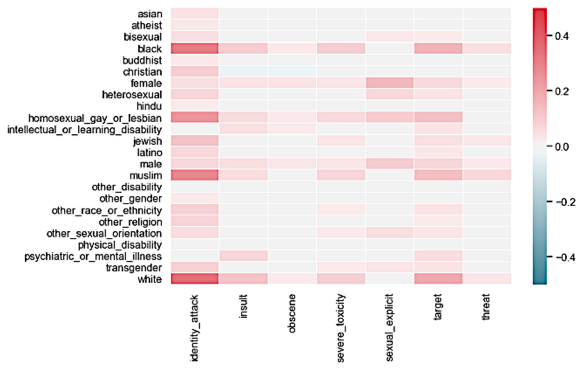


Fig 10. Correlation heatmap of target variable, toxicity subtype attributes and the identity variables.

Another correlation heatmap is generated at fig 10 to observe the relationship between target variable, toxicity subtype attributes and the identity variables. The subtype attribute 'identity attack' has higher positive correlation value with the race white and black, religion Muslim, and sex homosexual gay or lesbian. The exact same identity variables have high correlation with the target variable which verify the weighted toxicity of the identity variables at fig 08.

Overall, the additional data, coming along with the target variable and the comments, seems not really contributing to predict the toxicity. Thus, I decided on excluding the information for the following preprocessing and modeling. This does not mean that for other datasets from other platforms will lead to the same conclusions.

## IV. Data Preprocessing:

Before applying for predictive modeling, text data requires special preparation. The text must be parsed to remove words, called tokenization. Tokenization can turn a string into tokens which means smaller chunks. It's one of the steps of text preprocessing for natural language processing. Tokenization gives more granular control over the process. Then the words need to be encoded as integers or floating-point values to use as input to a machine learning algorithm, called feature extraction or vectorization.

**Count Vectorizer:** An encoded vector is returned with a length of the entire vocabulary and an integer count for the number of times each word appeared in the document. One disadvantage of simple counts is that some words will appear many times and their large counts won't be meaningful in the encoded vectors [6], [9].

**TFIDF Vectorizer:** TFIDF is Term Frequency and Inverse Document Frequency. Term frequency summarizes how often a given word appears within a document. Inverse document frequency downscales words that appear a lot across documents. Counts and frequencies can be very useful, but one limitation of these methods is that the vocabulary can become very large. This, in turn, will require large vectors for encoding documents and impose large requirements on memory and slow down algorithms [7], [9].

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right) \ldots. \ \ldots. (1)$$

$w_{i,j}$ = TFIDF weight for token $i$ in comment $j$
$tf_{i,j}$ = number of occurrences of token $i$ in the comment $j$
$df_i$ = number of comments that contain token $i$
$N$ = total number of comments

From equation 1, if the ratio of the total number of comments (N) and the number of comments that contain token $i$ is close to 1, TFIDF weight for token $i$ in comment $j$ ($w_{i,j}$) becomes close to zero. When the number of occurrences of token $i$ in the comment $j$ ($tf_{i,j}$) is small, the weight is also small.

**Hashing vectorizer:** A smart work is to use a one-way hash of words to convert them to integers. The clever part is that no vocabulary is required, and one can choose an arbitrary-long fixed length vector. The limitation is that the hash is a one-way function so there is no way to convert the encoding back to a word. [6], [9].

## V. Background Theory of Machine Leaning Algorithms:

### Logistic Regression:

Logistic Regression is used when the dependent variable (target) is categorical [10]. Here, our goal is to predict whether a comment is toxic (1) or non-toxic (0).

The posterior probability of class $C_1$can be written as a logistic sigmoid function $\sigma(.)$ acting on a linear function of the feature vector $\phi$,

$$p(C_1|\boldsymbol{\phi}) = \sigma(\boldsymbol{w}^T\boldsymbol{\phi}) \ldots\ldots (2)$$

$$\text{where } p(C_2|\boldsymbol{\phi}) = 1\text{-}\ p(C_1|\boldsymbol{\phi})$$

For a dataset $\{\boldsymbol{\phi}_n, \boldsymbol{t}_n\}$, where the target variable $t_n \in \{0,1\}$ and the basis function vector $\boldsymbol{\phi}_n = \boldsymbol{\phi}(x_n)$ with n = 1, …., N, the likelihood function is,

$$p(\boldsymbol{t}|\boldsymbol{w}) = \prod_{n=1}^{N} y_n^{t_n}\{1 - y_n\}^{1-t_n} \ldots\ldots (3)$$

$$\text{where } \boldsymbol{t} = (t_1, \ldots, t_N)^T$$
$$\text{and } y_n = p(C_1|\boldsymbol{\phi}_n)$$

By taking the negative logarithm of the likelihood function, the cross-entropy error function is,

$$E(\boldsymbol{w}) = -lnp(\boldsymbol{t}|\boldsymbol{w})$$
$$= -ln\left[\prod_{n=1}^{N} y_n^{t_n}\{1 - y_n\}^{1-t_n}\right]$$
$$= -\sum_{n=1}^{N}\{t_n lny_n + (1 - t_n)ln(1 - y_n)\}$$
$$\ldots\ldots\ldots (4)$$

where $y_n = \sigma(\boldsymbol{w}^T\phi_n)$

Taking the gradient of the error function with respect to the **w**, we get,

$$\nabla E(\boldsymbol{w}) = \sum_{n=1}^{N}(y_n - t_n)\phi_n$$
$$\ldots\ldots (5)$$

### Recurrent Neural Network (RNN):

In order understand the working principle of RNN, we need to know how to perform back propagation. The propagation follows two directions: vertical (between input and output) and horizontal (going through time). Due to this horizontal direction, back propagation is referred back as propagation through time.

In the forward propagation, a hidden state $a_t$ is computed that will carry past information by applying the linear combination over the previous step and the current input. The output $\hat{y}_t$ is computed only at the last hidden state often by applying a sigmoid or softmax activation function.

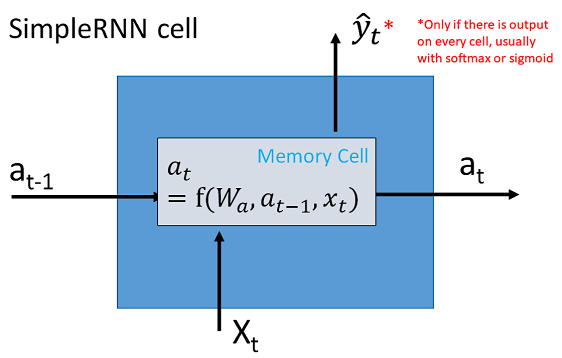

SimpleRNN cell
$\hat{y}_t$*
*Only if there is output on every cell, usually with softmax or sigmoid
$a_{t-1}$
Memory Cell
$a_t = f(W_a, a_{t-1}, x_t)$
$a_t$
$X_t$

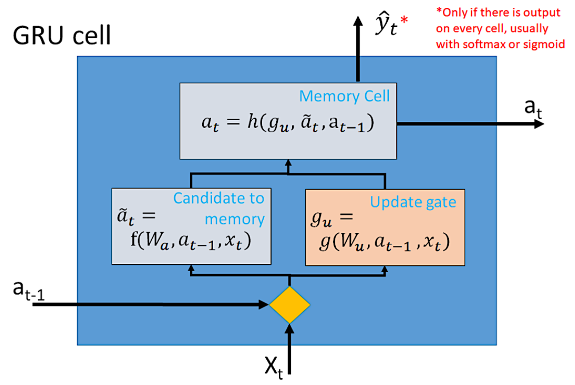
Fig 11. Architecture of a simple RNN cell [7].

$$a_t = f(W_a, a_{t-1}, x_t) = f(W_a, f(W_a, a_{t-2}, x_{t-1}), x_t)$$
$$\ldots\ldots (6)$$

For binary classification problem, the loss function can be binary cross-entropy. From the equation 6, the second step combines the results from the first step, and then receive the second word as input. The weight matrix $W_a$ is used on all steps, which means the weights are shared among all the inputs. During the back-propagation phase, we need to compute the derivatives of the loss function with respect to the weight matrix $W_a$ using the chain rule because the output $\hat{y}_t$ depends on the $a_t$ which is dependent on $W_a$. Besides, $a_t$ also depends on $a_{t-1}$ which depends on Wa and so on! Thus, we need to consider the contribution of every previous step by summing up their derivatives with respect to the weight matrix $W_a$. Computing derivatives lead to,

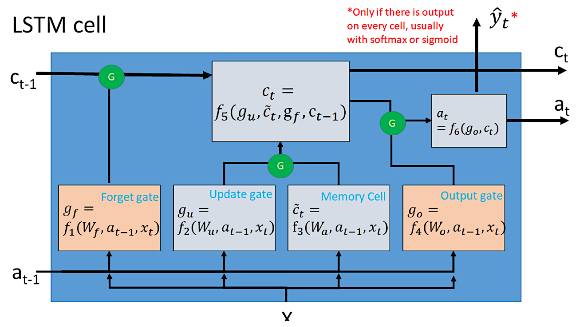
$$\frac{\partial a_t}{\partial W_a} = (W_a)^{t-1}g(X) \ldots\ldots (7)$$

During the computation of the gradients of the loss function with respect to the weight matrix, $(W_a)^{t-1}$ is multiplied by a term. Intuitively, if the values of the matrix are below one, $(W_a)^{t-1}$ can converge to 0. If the values are above one it will diverge to infinity. Vanilla RNN suffers from exploding and vanishing gradient problem. By limiting the size of the gradients or scaling them, the exploding gradients problems can eb solved. To overcome the limitation of vanishing gradients, better initialize the weight matrix as an orthogonal matrix makes their multiplication always be equal to 0. Second solution is to use regularization which controls the size of the entries. Third solution is to use ReLU instead of tanh/sigmoid/softmax, the derivative becomes a constant, thus doesn't increase or decrease exponentially. The best approach is to apply LSTM or GRU cells.



GRU cell
$\hat{y}_t$*
*Only if there is output on every cell, usually with softmax or sigmoid
Memory Cell
$a_t = h(g_u, \tilde{a}_t, a_{t-1})$
$a_t$
Candidate to memory
$\tilde{a}_t =$ f$(W_a, a_{t-1}, x_t)$
Update gate
$g_u = g(W_u, a_{t-1}, x_t)$
$a_{t-1}$
$X_t$

Fig 12. Architecture of a GRU cell [7].



LSTM cell
*Only if there is output on every cell, usually with softmax or sigmoid
$\hat{y}_t$*
$c_{t-1}$
$c_t$
$a_t$
G
$c_t = f_5(g_u, \tilde{c}_t, g_f, c_{t-1})$
$a_t = f_6(g_o, c_t)$
Forget gate
$g_f = f_1(W_f, a_{t-1}, x_t)$
Update gate
$g_u = f_2(W_u, a_{t-1}, x_t)$
Memory Cell
$\tilde{c}_t = f_3(W_a, a_{t-1}, x_t)$
Output gate
$g_o = f_4(W_o, a_{t-1}, x_t)$
$a_{t-1}$
x

Fig 13. Architecture of an LSTM cell [7].

**Steps of Gated Recurrent Unit (GRU):**
1. Compute candidate $\tilde{a}_t$ that will carry the present information (fig 12).
2. Compute the update gate (GU) that will determine if the candidate $\tilde{a}_t$ will be used as memory state or if we keep the past memory state $a_{t-1}$ .
- If the update gate is 0, the network keeps the previous hidden state.
- If the update gate is 1, it uses the new value of $\tilde{a}_t$.
- Other values will be a combination of the previous and the candidate memory state. During training it tends to close to 0 or 1.

**Gates of Long Short-Term Memory (LSTM):**
1. The forget gate determines if the previous state $c_{t-1}$ state should be forgotten or not (fig 13).
2. The update gate determines if the candidate hidden state $\tilde{c}_t$ should be forgotten or not.
3. The output gate determines if the new hidden state $c_t$ should be forgotten or not.

GRU and LSTM cells add gates to the equations, the gradients are no longer only dependent on the memory cell state. The derivatives of the loss function with respect to the weight matrix depends on all the gates and the memory cell, summing each of its parts, making the total gradient stop converging to zero or diverging. On every step, if the gradient is exponentially increasing or decreasing, we expect the training phase to adjust the value of the corresponding gate accordingly to stop this vanishing or exploding gradient tendency.

In order to further improvement of the model, we can add embedding layer, increase the number of layers, tune the hyperparameters, increase vocabulary size or accept longer sentences with more memory cells.

Using embedding layer, vectorization of a language model can be done. Embedding layer reduces dimension because one-hot encoding of the tokens for a very big vocabulary (billions of words) demands a lot of memory. Besides, dense representations of the words and the implementation gives surprisingly nice understanding of the tokens. Most importantly embedding layer can be used for transfer learning. Only one disadvantage of embedding layer is it demands training lots of parameters to learn this representation and can make training slower.

RNN models can overfit even with a few epochs like 10. If the model overfits, we can test different batch sizes because RNN models are sensitive to them since the batch size determines the number of updates in the weights that will be performed. Adding dropout layers and using the parameter recurrent dropout can add extra noise to the training data, forcing the model to be more general and reduce overfitting. The parameter dropout on RNN layers removes a percentage of the input data, while the recurrent dropout removes a percentage of the memory cell. In order to overcome overfitting, we can also apply max pooling which contains the parameter pool size that determines the window to look for the maximum value.

**Advanced Approach: Transfer Learning**
Instead of training a model from scratch, it's possible to use models pre-trained on a large dataset and then fine-tune them for specific natural language tasks. Google AI's Bidirectional Encoder Representations from Transformers (BERT) [4] have revolutionized the field of transfer learning in natural language processing by using language modelling during pre-

training, which has improved the state-of-the-art for a variety of tasks in NLP.

Two popular word embeddings are GloVe [13] and fastText [14]. These embeddings map each word onto a low-dimensional vector d commonly chosen to be roughly 300. Words that appear in similar contexts, like 'man' and 'male' should have similar embeddings.

Practitioners of deep learning for NLP typically initialize their models using pre-trained word embeddings, bringing in outside information, and reducing the number of parameters that a neural network needs to learn from scratch. Without GPU, transfer learning is beyond the scope of this project.

## VI. Experimental Setup:

**A. System Specification:** The device used for this project is a MacBook Pro 2017 which has operating system MacOS Mojave, Processor 2.3 GHz dual-core Intel Core i5, and Memory 8GB.

**B. Tokenization and Feature Extraction:** The scikit-learn library offers easy-to-use tools to perform both tokenization and feature extraction of the text data.

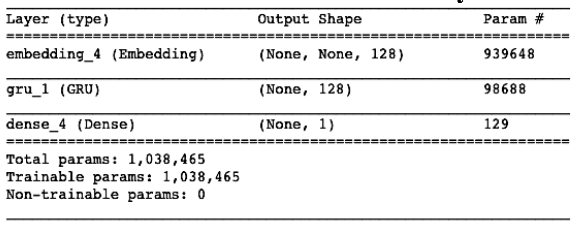
- Convert text to word count vectors with CountVectorizer.
- Convert text to word frequency vectors with TfidfVectorizer.
- Convert text to unique integers with HashingVectorizer.

**C. Solver:** I used the solver Stochastic Average Gradient (SAG) method for logistic regression which optimizes the sum of a finite number of smooth convex functions. Like stochastic gradient (SG) methods, the iteration cost of the SAG method is independent of the number of terms in the sum. However, by incorporating a memory of previous gradient values the SAG method achieves a faster convergence rate than black-box SG methods. It is faster than other solvers for large datasets, when both the number of samples and the number of features is large [12].

**D. Optimizer:** Adam [11] (adaptive moment estimation) is an adaptive learning rate method and computes individual learning rates for different parameters. Adam uses estimations of first moment (mean) and second moment (variance) of gradient to adapt the learning rate for each weight of the neural network.

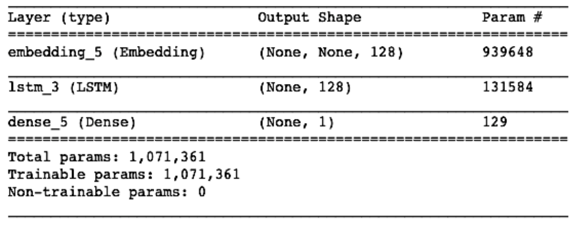
**E. Model Summary:** Table 1a represents the GRU model summary and table 1b represents the LSTM model summary. From these tables, we can see that LSTM model has higher number of trainable parameters than GRU model.

**Table 1a. GRU model summary**



```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, None, 128)         939648
_________________________________________________________________
gru_1 (GRU)                  (None, 128)               98688
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 129
=================================================================
Total params: 1,038,465
Trainable params: 1,038,465
Non-trainable params: 0
_________________________________________________________________
```

**Table 1b. LSTM model summary**



```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_5 (Embedding)      (None, None, 128)         939648
_________________________________________________________________
lstm_3 (LSTM)                (None, 128)               131584
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 129
=================================================================
Total params: 1,071,361
Trainable params: 1,071,361
Non-trainable params: 0
_________________________________________________________________
```

## VII. Results and Discussion:

**A. Confusion Matrix:**

For logistic regression with three different types of vectorizer, I use 70% of the data for training and rest 30% of the data for testing. For simple RNN, GRU and LSTM model I use 800 observations for training and 200 observations for testing. The average runtime per epoch for the simple RNN model, GRU model, and the LSTM model are respectively 33 seconds, 231 seconds, and 310 seconds. LSTM takes the highest time which makes it very difficult run without GPU.

From table 2a-c, the highest number of correctly classified non-toxic comments can be found from the count vectorizer with logistic regression. On the other hand, the highest number of correctly classified toxic comments can be found from the hashing vectorizer with logistic regression. Table 2d and 2e look exactly same because in this case, the designed GRU model and LSTM model perform exactly same. To get the correct classification of the toxic comments, the predicted target variable was modified to >= 0.47 for the toxic comments which gave the 100% correct classification.

**Table 2a. TFIDF vectorizer & Logistic Regression**

| | | Predicted | |
|---|---|---|---|
| | | Non-toxic | toxic |
| True Class | Non-toxic | 491188 | 6646 |
| | toxic | 22012 | 21617 |

**Table 2b. Count vectorizer & Logistic Regression**

| | | Predicted | |
|---|---|---|---|
| | | Non-toxic | toxic |
| True Class | Non-toxic | 493874 | 3960 |
| | toxic | 28285 | 15344 |

**Table 2c. Hashing vectorizer & Logistic Regression**

| | | Predicted | |
|---|---|---|---|
| | | Non-toxic | toxic |
| True Class | Non-toxic | 492175 | 5659 |
| | toxic | 23071 | 20558 |

**Table 2d. TFIDF vectorizer & GRU**

| | | Predicted | |
|---|---|---|---|
| | | Non-toxic | toxic |
| True Class | Non-toxic | 188 | 0 |
| | toxic | 0 | 12 |

**Table 2e. TFIDF vectorizer & LSTM**

| | | Predicted | |
|---|---|---|---|
| | | Non-toxic | toxic |
| True Class | Non-toxic | 188 | 0 |
| | toxic | 0 | 12 |

**B. Classification Report:**

In order to understand the confusion matrix elaborately, we need to understand the concepts of precision, recall, and F1 score. For each class, the precision measures how accurately the model is predicting the class.

$$Precision_{class} = \frac{Correct_{class}}{Predicted_{class}} \ldots\ldots (8)$$

Recall measures if the classes are being correctly classified.

$$Recall_{class} = \frac{Correct_{class}}{N_{class}} \ldots\ldots (9)$$

**Table 3. Classification report**

| Machine Learning Approach | Train Data size | Test data size | Accuracy | ROC curve area | | Precision | Recall | F1 score |
|---|---|---|---|---|---|---|---|---|
| Tf-idf + Logistic Regression | 1.26M | 0.54M | 0.945 | 0.741 | 0 | 0.96 | 0.99 | 0.97 |
| | | | | | 1 | 0.76 | 0.50 | 0.60 |
| Count + Logistic Regression | 1.26M | 0.54M | 0.913 | 0.672 | 0 | 0.95 | 0.99 | 0.97 |
| | | | | | 1 | 0.79 | 0.35 | 0.49 |
| Hashing + Logistic Regression | 1.26M | 0.54M | 0.945 | 0.73 | 0 | 0.96 | 0.99 | 0.97 |
| | | | | | 1 | 0.78 | 0.47 | 0.59 |
| Tf-idf + Simple RNN | 800 | 200 | 0.0 | - | 0 | - | - | - |
| | | | | | 1 | - | - | - |
| Tf-idf + LSTM | 800 | 200 | 1.0 | 1 | 0 | 1 | 1 | 1 |
| | | | | | 1 | 1 | 1 | 1 |
| Tf-idf + GRU | 800 | 200 | 1.0 | 1 | 0 | 1 | 1 | 1 |
| | | | | | 1 | 1 | 1 | 1 |

Precision is a metric that measures how well the model is predicting two classes, while recall measures how well a class is being classified. If the precision is high for one class, we can trust the model when it predicts that class. When the recall is high for a class, we can be assured that the class is well understood by the model.

F1 score is a weighted harmonic average between precision and recall.

$$F1\ Score_{class} = 2 \times \frac{Precision_{class} \times Recall_{class}}{Precision_{class} + Recall_{class}} \ldots\ldots (10)$$

From table 3, for logistic regression, to classify the non-toxic comments, TFIDF and Hashing vectorizer perform better than count vectorizer. The recall and F1score for non-toxic comments, all logistic regression models have same performance. These models suffer from the classification error of the toxic comments.

**C. ROC curve:**

Fig 14. compares the receiver operating characteristic (ROC) curve for the Logistic Regression model with different vectorizer. We know that, in ROC curve, the best model is which has area close to 1. From Table 3, among the linear regression models, the ROC curve area is the highest 0.741 for the TFIDF vectorizer.

For accuracy comparison the TFIDF and Hashing vectorizer with logistic regression give the best accuracy 94.5%. For the GRU and The LSTM model, the dataset is small but enough to propose a simple deep learning model to perfectly classify the non-toxic and toxic comments from the online discussions.

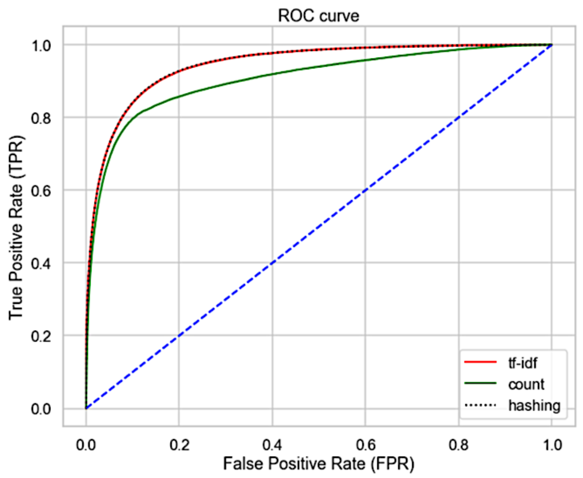


Fig 14. Comparison of the receiver operating characteristic (ROC) curve for the Logistic Regression model with different vectorizers.

**D. Training Accuracy vs Validation Accuracy:**
For the deep learning models, both the GRU and LSTM model have exactly same graph for the Training Accuracy vs Validation Accuracy (fig 15). The validation accuracy reached to 94% accuracy after the first epoch and converged with the training accuracy.

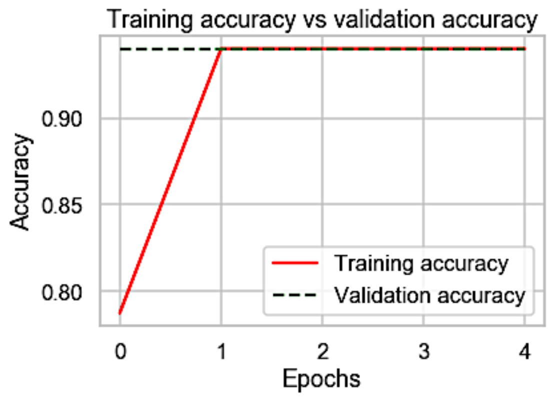


Fig 15. Training accuracy vs validation accuracy of the LSTM model.

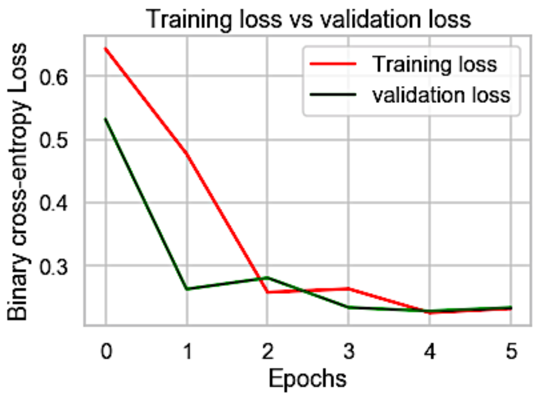


Fig 16. Training loss vs validation loss of the LSTM model. Here the loss function is binary cross-entropy.

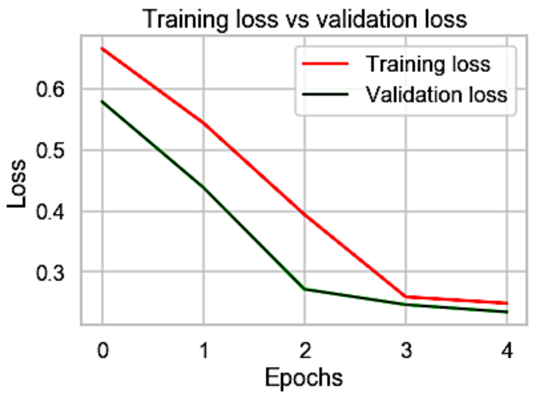


Fig 17. Training loss vs validation loss of the GRU model. Here the loss function is binary cross-entropy.

**E. Binary Cross-entropy Loss:**
Fig 16. displays the training loss vs validation loss of the LSTM model. Here the loss function is binary cross-entropy. The model converged after the fifth epoch. The training loss vs validation loss of the GRU model is represented at fig 17.

## VIII. Conclusion:
It was my first-time experience in Kaggle competition as well as the NLP problem. Without GPU support, the classification task was challenging due to the extremely long runtime of deep learning models. The proposed logistic regression model achieved 94.5% accuracy for the binary classification of non-toxic and toxic comments with TFIDF and hashing vectorizer for the 1.26M training data and 0.54M test observations. The proposed GRU and LSTM model achieved 100% accuracy for a small portion of the data. In the future, it's possible to apply the LSTM model with transfer learning and GPU support on the 1.8M observations for training and 97k publicly available test data. The advanced data analytics course and this final project gave me a comprehensive knowledge of deep learning, natural language processing, and, most importantly, it gave me the confidence to explore more difficult machine learning problems.

## IX. References:
[1] https://conversationai.github.io/
[2] https://jigsaw.google.com/
[3] Jigsaw Unintended Bias in Toxicity Classification, https://www.kaggle.com/c/jigsaw-unintended-bias-in-toxicity-classification/overview
[4] Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding." (2018).
[5] "Trump Says Transgender People Will Not Be Allowed in the Military", The New York Times, July 26, 2017 www.nytimes.com/2017/07/26/us/politics/trump-transgender-military.html
[6] Hacking Scikit-Learn's Vectorizers, towardsdatascience.com/hacking-scikit-learns-vectorizers-9ef26a7170af
[7] "Recurrent Neural Networks for Language Modeling in Python," www.datacamp.com/courses/recurrent-neural-networks-for-language-modeling-in-python
[8] "Introduction to Natural Language Processing in Python," www.datacamp.com/courses/natural-language-processing-fundamentals-in-python
[9] "Advanced NLP with spaCy,"

www.datacamp.com/courses/advanced-nlp-with-spacy
[10] Christopher M. Bishop, "Pattern Recognition and Machine Learning," Springer, 2006
[11] Adam — latest trends in deep learning optimization, towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c
[12] stackoverflow.com/questions/38640109/logistic-regression-python-solvers-defintions
[13] Jeffrey Pennington, Richard Socher, Christopher D. Manning, "GloVe: Global Vectors for Word Representation."
[14] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jegou, T. Mikolov, "FastText.zip: Compressing text classification models."