# Machine Learning Engineer Nanodegree

## Capstone Project

Ramkumar Narayanan

October 15th, 2018

### Definition

### Project Overview

Social media serves as a mean to express their thoughts or feelings about different subjects. It can be a political view or about a new product launched in the market or even about a movie.

An average of 2 hours and 15 minutes per day is spend on social networks. Facebook users generate 4 million comments every minute. Every second, on average, around 6,000 tweets are tweeted on Twitter which corresponds to over 350,000 tweets sent per minute, 500 million tweets per day and around 200 billion tweets per year. When we start analyzing the data from social media and perform sentiment analysis we would be able to obtain great deal of insights regarding that particular subject. It is like a crowd funded survey without any formal questionnaire at zero expense.

We are planning to conduct sentiment analysis in our company based on social media #hashtags right after a significant company milestone. I am hoping this project will be a foundation to get started in the domain of NLP which eventually I can apply at my work.

### Problem Statement

Discussing things that we care about can be difficult. The threat of abuse and harassment online means that many people stop expressing themselves and give up on seeking different opinions. Platforms struggle to effectively facilitate conversations, leading many communities to limit or completely shut down user comments. Many companies are working on tools to improve healthy online conversations and curb negative online behaviors like toxic comments (i.e. comments that are rude, disrespectful or otherwise likely to make someone leave a discussion).

So the objective is to build a multi-label classification model that is capable of detecting different types of toxicity like threats, obscenity, insults, and identity-based hate among a list of comments posted online. By doing this we can help online discussion become more productive and respectful.

Below is a brief overview of the solution being implemented

**Step-1: Data Explorations**
Before starting the project, I would like to go through the data thoroughly. I need to understand on what basis the comments present in the training data are labelled.

**Step-2: Data pre-processing and noise removal:**
Apply noise removal technique from nltk to remove texts which are not relevant to the context of the data. For e.g.: URLS, punctuations, acronyms, stopwords etc. and then to tokenize/sequence the text with keras preprocessing packages

**Step-3: Feature Engineering:**
To analyze the preprocessed data, the text is converted into features. Depending upon the usage, text features can be constructed using various techniques – Syntactical Parsing, Entities / N-grams / word-based features, Statistical features, and word embedding. In this project word embedding is performed using Glove dataset and crawl vectors.
**Step-4: Neural Net Model**
The final step is to create the Recurrent Neural Net model to train using the training data. Here LSTM and GRU networks are implemented. Based on the learning rate hyper parameters are tweaked and results are analyzed. The final ROC score is calculated against the test data provided.

## Metrics

Since this is a multi-label classification task final evaluation metric is (ROC AUC) score. Area Under the Receiver Operating Characteristic (ROC), or simply ROC curve, is a graphical plot which illustrates the performance of a binary classifier system as its discrimination threshold is varied. It is created by plotting the fraction of true positives out of the positives (TPR = true positive rate) vs. the fraction of false positives out of the negatives (FPR = false positive rate), at various threshold settings. TPR is also known as sensitivity, and FPR is one minus the specificity or true negative rate.

So according to kaggle the model is evaluated on the mean column-wise ROC AUC. In other words, the score is the average of the individual AUCs of each predicted column. The reason why kaggle uses ROC curve is to avoid class imbalance. Picking up a label which occurs most of the time may lead to high accuracy score but which in turn is wrong.For example, if 99% of observations had False labels, always responding False would result in a 99% accuracy.
The output file (test file) must predict a probability for each of the six possible types of comment toxicity (toxic, severe_toxic, obscene, threat, insult, identity_hate). Below is the sample test file

Test file sample:

id,toxic,severe_toxic,obscene,threat,insult,identity_hate
00001cee341fdb12,0.5,0.5,0.5,0.5,0.5,0.5
0000247867823ef7,0.5,0.5,0.5,0.5,0.5,0.5

# Analysis

## Data Exploration

The data is given as a comma separated value file with comments and the corresponding sentiments. The dataset is a list of comments from Wikipedia's talk page edit.

The comments in the dataset falls under one or more categories listed below.

1. Toxic

2. Severe Toxic
3. Obscene
4. Threat
5. Insult
6. Identity hate

The Dataset has 159,571 comments categorized into 6 different categories. If the number under any category is '1' then the comment falls under that category and '0' if the comment doesn't fall under the category. There are no personal information (no pii or spii data) about the user and it is a publicly available dataset in Kaggle. Below is a sample of the data set

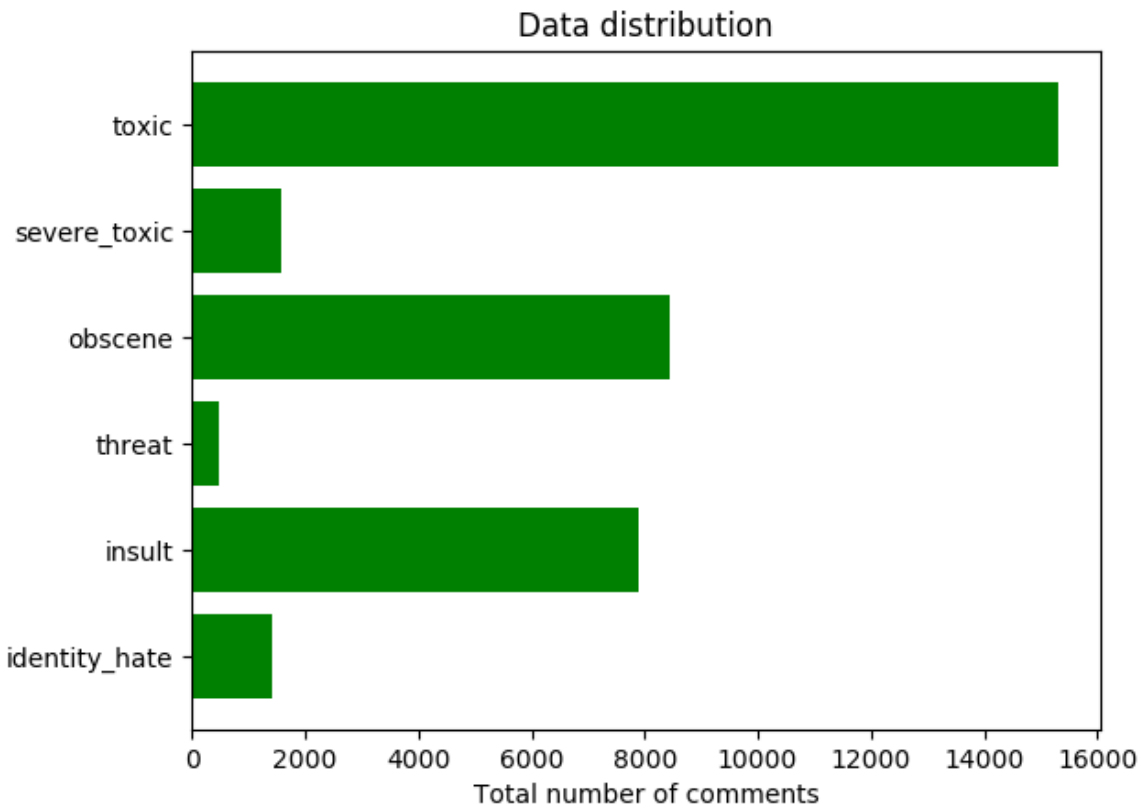
**DISCLAIMER : Dataset has obscene words!!**

```
A pair of jew-hating weiner nazi schmucks.
```

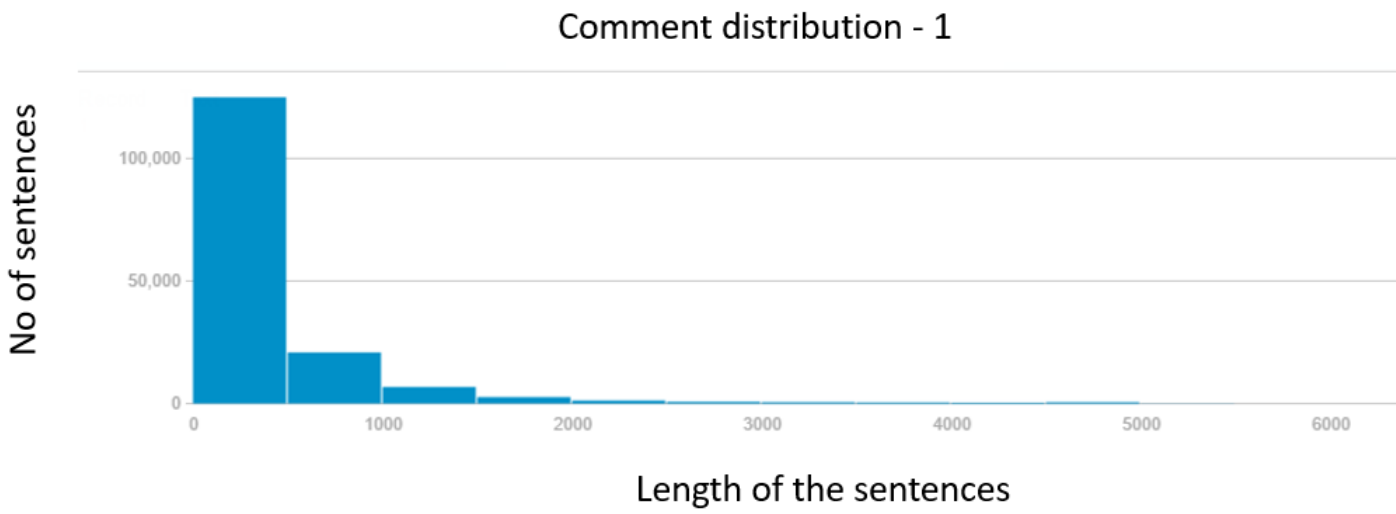
| Record # | id | comment_text | toxic | severe_toxic | obscene | threat | insult | identity_hate |
|---|---|---|---|---|---|---|---|---|
| 1 | 0002bcb3da6cb337 | COCKSUCKER BEFORE YOU PISS AROUND ON... | 1 | 1 | 1 | 0 | 1 | 0 |
| 2 | 001810bf8c45bf5f | You are gay or antisemmitian? | 1 | 0 | 1 | 0 | 1 | 1 |
| 3 | 00190820581d90ce | FUCK YOUR FILTHY MOTHER IN THE ASS, DRY! | 1 | 0 | 1 | 0 | 1 | 0 |
| 4 | 0020e7119b96eeeb | Stupid peace of shit stop deleting my stuff ass... | 1 | 1 | 1 | 0 | 1 | 0 |
| 5 | 0020fd96ed3b8c8b | =Tony Sidaway is obviously a fistfuckee. He lov... | 1 | 0 | 1 | 0 | 1 | 0 |
| 6 | 0028d62e8a5629aa | All of my edits are good. Cunts like you who r... | 1 | 0 | 1 | 0 | 1 | 0 |
| 7 | 0036621e4c7e10b5 | Would you both shut up, you don't run wikipe... | 1 | 0 | 0 | 0 | 1 | 0 |
| 8 | 00472b8e2d38d1ea | A pair of jew-hating weiner nazi schmucks. | 1 | 0 | 1 | 0 | 1 | 1 |
| 9 | 00686325bcc16080 | You should be fired, you're a moronic wimp w... | 1 | 0 | 0 | 0 | 1 | 0 |
| 10 | 006b94add72ed61c | I think that your a Fagget get a oife and burn i... | 1 | 0 | 1 | 1 | 1 | 1 |
| 11 | 006d11791d76b9f3 | REPLY ABOVE: | 0 | 0 | 0 | 0 | 1 | 0 |
| 12 | 006e87872c8b370c | you are a stupid fuck | 1 | 1 | 1 | 0 | 1 | 0 |
| 13 | 0086998b34865f93 | Fuck you, block me, you faggot pussy! | 1 | 0 | 1 | 0 | 1 | 0 |
| 14 | 008e0818dde894fb | Kill all niggers. | 1 | 0 | 1 | 0 | 1 | 1 |

Id - Unique key to represent a user
Comment_text - Raw comment data

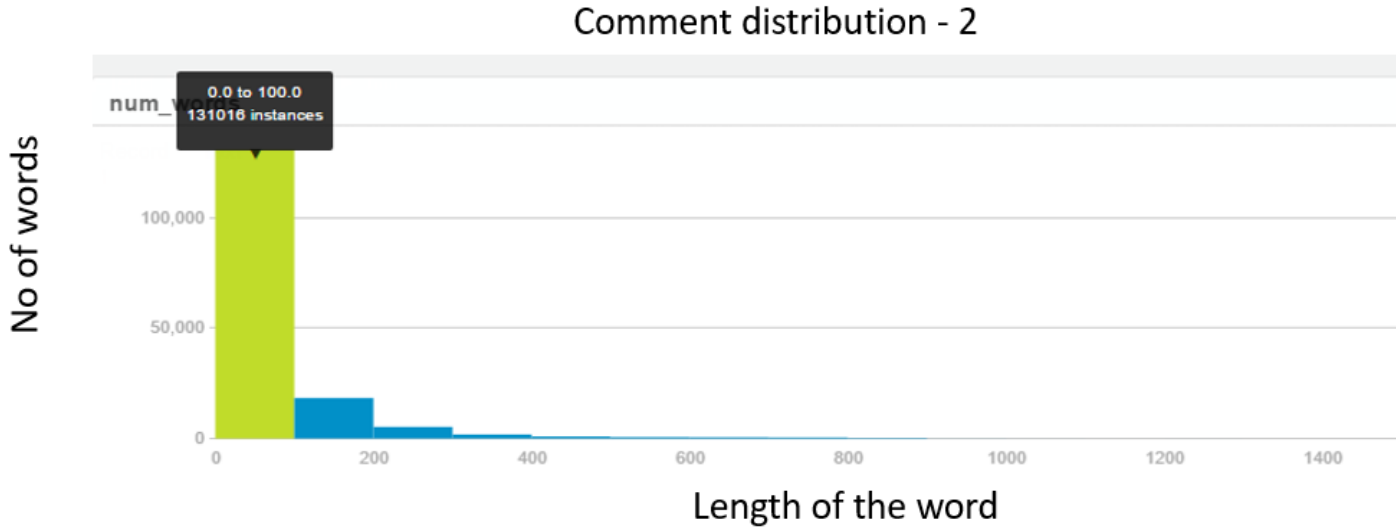
Out of the 159,571 comments
15,294 falls under Toxic
1,595 falls under Severe Toxic
8,449 falls under Obscene
478 falls under Threat
7,877 falls under Insult
1,405 falls under identity_hate
Remaining comments are neutral.




The comments present in the training file is of varying length. The graph below shows the distribution of the comment field in the train file
Basic Statistics:
Max length of a comment: 5895
Min length of a comment:  6
Mean length of a comment:  394
Total number of comments:  159,571
Total number of comments with length between 0 to 500:  125,172
Total number of comments with length between 500 to 1000: 20,904
Total number of comments with length more than 1000: 13,495
Total number of comments containing URLs: 4804




Total number of words in the comment field: 10,732,858
Max number of words in a comment: 1411
Mean number of words in a comment: 36
Total number of comments with 0 to 100 words: 131,016
Total number of comments with 100 to 200 words: 18,389
Total number of comments with more than 200 words: 10,166




The dataset is a mixture of words, emoticons, symbols and URLS. URLS and reference to other people don't contribute in predicting any sentiment. Since all the comments are free text field, words are often misspelled, sentence has punctuation where it is not necessary and words has many repeated letters.

Similar to the training set the test set has 153,164 records with just Id and comment_text field.

# Algorithms and Techniques:

The solution is to come up with the probability of a comment present in the test data falling under any of the above mentioned category.

**Word Embedding:**

A word embedding is a learned representation for text where words that have the same meaning have a similar representation. They are class of techniques where individual words are represented as real-valued vectors in a predefined vector space. Each word is mapped to one vector and the vector values are learned in a way that resembles a neural network.
Two types of word embeddings are used in the project to train the network.

1. Glove
2. Faststext's English vector

**Glove:**

Global Vectors for Word Representation also called as Glove is an unsupervised learning algorithm for obtaining vector representations for word and is used as the 1st embedding layer. Glove algorithm is an extension to the word2vec method for efficiently learning a standalone word embedding from a text corpus, developed by Pennington, et al. at Stanford. Classical vector space model representations of words were developed using matrix factorization techniques such as Latent Semantic Analysis (LSA) that do a good job of using global text statistics but are not as good as the learned methods like word2vec at capturing meaning and demonstrating it on tasks like calculating analogies. Analogy here means syntactic and semantic regularities in language. For example, learning male/female relationship. By subtracting the "man-ness" from "King" and adding "women-ness" results in the word "Queen", capturing the analogy "king is to queen as man is to woman". The induced vector representation here, "King – Man + Woman = Queen".

GloVe is an approach to marry both the global statistics of matrix factorization techniques like LSA with the local context-based learning in word2vec.

Rather than using a window to define local context, Glove constructs an explicit word-context or word co-occurrence matrix using statistics across the whole text corpus. The training objective of Glove is to learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence. Owing to the fact that the logarithm of a ratio equals the difference of logarithms, this objective associates (the logarithm of) ratios of co-occurrence probabilities with vector differences in the word vector space. Because these ratios can encode some form of meaning, this information gets encoded as vector differences as well and the output tabulates how frequently words co-occur with one another in a given corpus. Populating this matrix requires a single pass through the entire corpus to collect the statistics. For large corpora, this pass can be computationally expensive, but it is a one-time up-front cost. Subsequent training iterations are much faster because the number of non-zero matrix entries is typically much smaller than the total number of words in the corpus. For this reason, the resulting word vectors perform very well on word analogy tasks, such as those examined in the word2vec package.

**Faststext's English vector:**

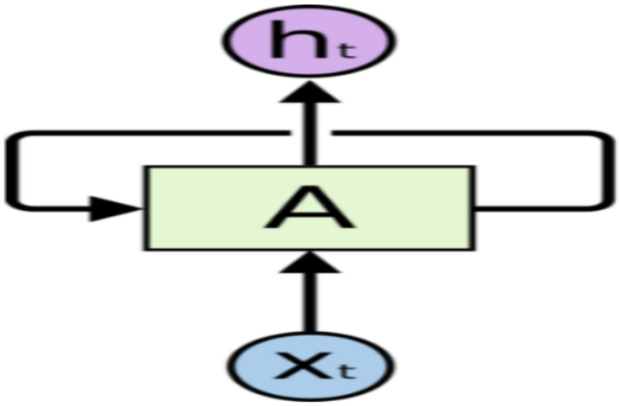
Similar to Glove , crawl-300d-2M.vec is a pre-trained word vectors trained using fastText.It has 2 million word vectors trained on Common Crawl which is a nonprofit organization that crawls the web and freely provides its archives and datasets to the public. Common Crawl's web archive consists of petabytes of data collected since 2011 .

**Neural Net architecture:**

As comments or any sentence for that matter are sequential data which contains words in some order a Bi-directional Recurrent Neural Network is implemented using LSTM network. One another important reason to choose LSTM network is to make sure the network derives and remembers sentence context from the previous time steps as well.

**RNN:**

A recurrent neural network (RNN) is a class of artificial neural network where connections between nodes form a directed graph along a sequence. This allows it to exhibit temporal dynamic behavior for a time sequence. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition[1] or speech recognition.




A simple recurrent neural network is represented in the above figure. Each node at a time step takes an input from the previous node and this can be represented using a feedback loop. The same can be expanded to multiple time step. At each time step, we take an input x_i and a_i-1(output of the previous node) and perform computation on it and produce an output h_i. This output is taken and given to the next node. This process continues until all the time steps are evaluated.
However using a simple RNN is that as the time steps increase, it fails to derive context from time steps which are much far behind.

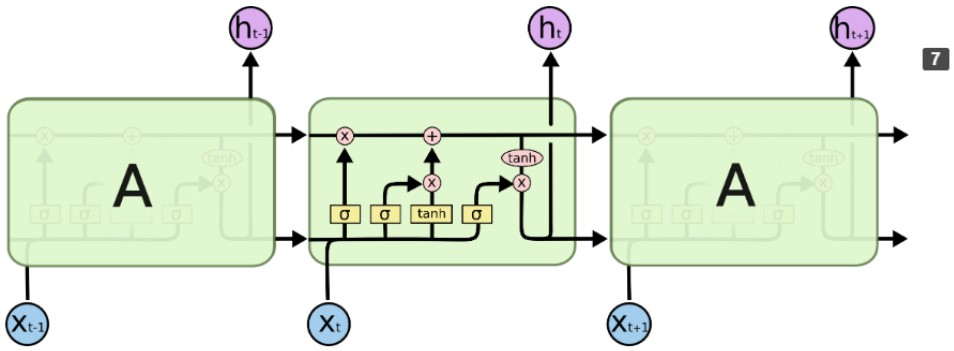
In order to classify the comment into one of the 5 category the neural net architecture should have the ability to derive context from time steps which are much far behind. As each comment is free text field authored by different type of users the neural network using a simple RNN will not aide in this task. The problem can be attributed to the cause of vanishing gradients as RNN is able to remember only short-term memory sequences.

To address this a Bidirectional LSTM network is used to train the data. The structure of an LSTM network remains the same as an RNN, whereas the repeating module does more operations. Enhancing the repeating module enables the LSTM network to remember long-term dependencies.

LSTM similar to RNN has a chain like structure, but the repeating module is structured differently. Instead of having a single neural network layer, there are four, interacting in a very special way.

Below is an illustration of a LSTM architecture.

---




As LSTM has the ability to remove or add information to the cell state using the gate architecture which will be very useful in training our model. There are 3 gates on what information will pass through.

$$gate_{forget} = \sigma(W_{fx}X_t + W_{fh}h_{t-1} + b_f)$$
$$gate_{input} = \sigma(W_{ix}X_t + W_{ih}h_{t-1} + b_i)$$
$$gate_{out} = \sigma(W_{ox}X_t + W_{oh}h_{t-1} + b_o)$$

For example, in the sentence

“Stupid peace of shit stop deleting my stuff ***hole go die and fall in a hole go to hell!”

Neural net should remember all the 5 classes of toxic words in a sentence only then it can predict the probability of a sentence being in a particular class at the same time it need not remember other irrelevant information like name, place etc. This decision is made by a sigmoid layer called the “forget gate layer.” It looks ht−1 and xt, and outputs a number between 0 and 1 for each number in the cell state Ct−1. 1 represents “completely keep this” while a 0 represents “completely get rid of this.”

The second step what new information we're going to store in the cell state.

3 equations to update the cell state and the hidden state:

$$\tilde{C} = \tanh(W_{cx}X_t + W_{ch}h_{t-1} + b_c)$$
$$C_t = gate_{forget} \cdot C_{t-1} + gate_{input} \cdot \tilde{C}$$
$$h_t = gate_{out} \cdot \tanh(C_t)$$

This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer

creates a vector of new candidate values, Ct~, that could be added to the state. In the next step, both are combined to

create an update to the state.

For example:

"so everytime i reset my modem my ip changes **ck you petty sexless ugly ass desperate no life.. no status in society
**al retentive wiki admins
(the site is great.. but the lower level admins have no life and can't handle the little authority they have)"
The sentence is classified as toxic, severe_toxic as well as obscene. Depending upon the word construct the layer first

stores that the sentence is toxic and then updates it with the information that it is also severe_toxic and obscene as well.

After updating old cell state, Ct−1, into the new cell state Ct, the gate decides what it is going to output as ht.

$$h_t = gate_{out} \cdot \tanh(C_t)$$

Using the same example above it outputs severe toxic words. The following layer will know what those words are and how they are structured together(rules) that makes it severely toxic .In the case of above example the neural net will a learn that a severe_toxic word will always be a toxic word as well but an obscene word need not be severe_toxic word always.

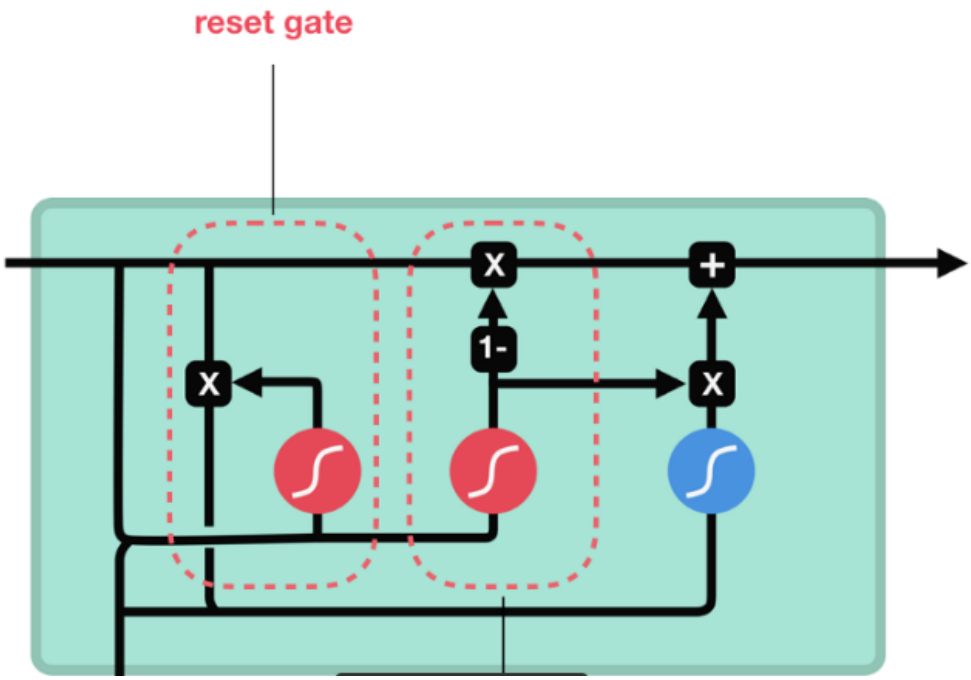
LSTM implemented in this project is Bidirectional. It involves duplicating the first LSTM layer in the network so that there are now two layers side-by-side. After which input sequence is provided to the first layer and a reversed copy of the input sequence to the second layer. By this way neural net may be trained using all available information in the past as well as future of a specific timeframe. The reason for using bidirectional in this project is due to mimic exactly how humans process the sentences. Even after going through first few words or the part of the sentence human may jump to conclusion but only when the future context is known (where the sentence actually leads) human can make correct decision. To achieve this in neural net and to

make the neural net understand the distinction between tasks that are truly online – requiring an output after every input – and those where outputs are only needed at the end of some input segment.

**GRU Network:**

To improve the accuracy of the result GRU network is also implemented on top of LSTM network. The output of the LSTM network is fed as input to the GRU network.

The main difference between LSTM and GRU is the Gate architecture. GRU's got rid of the cell state and used the hidden state to transfer information. It also only has two gates, a reset gate and update gate.




**Update gate:**

The update gate acts similar to the forget and input gate of an LSTM. It decides what information to throw away and what new information to add.

**Reset Gate:**

The reset gate is another gate is used to decide how much past information to forget.

GRU's has fewer tensor operations; therefore, they are a little speedier to train than an LSTM architecture. Researchers and engineers usually try both LSTM and GRU to determine which one works better for their use case. Here both are implemented to increase the accuracy. But there is no significant increase in accuracy with using both LSTM and GRU as opposed to just LSTM.

# Methodology:

## Data Preprocessing:

Data cleanup is the first order of business for many data-driven projects after exploration. As it is a free text field and real people will almost always will not use proper English in social media without data cleanup a meaningful accurate model cannot be created. To achieve this task nltk package and Keras text preprocessing tool called text_to_word_sequence() function is used.

1. Removing all the stop words using nltk.corpus package. It includes all the common stop words in English language.
2. Removing all irrelevant characters such as any non-alphanumeric [^A-Za-z0–9'] characters.

---

{"wasn't", 'same', 'to', 'be', 'i', 'from', 'd', 'through', 'whom', 'them', 'an', 's', 'yourselves', "don't", 'but', 'won', 'wi
th', 'nor', 'doesn', 'that', "hadn't", 'during', "won't", 'of', "you'd", "didn't", 'below', 'm', 'isn', 'those', "weren't", 'ot
her', 'down', 'its', "mightn't", 'most', 'needn', 'at', 'once', 'himself', 'few', '.', '}', "shouldn't", 'out', 'hers', 'might
n', 'doing', 'not', 'hadn', 'such', 'does', "haven't", "she's", "should've", 'being', 'have', 'into', 'or', "doesn't", 'yours',
',', 'who', 'had', 'itself', 'her', 'am', 'above', '[', 'it', 'weren', 'aren', 'about', 'your', "isn't", 'now', 'all', 'will',
'what', 'me', 'yourself', 'do', 'just', "you're", 'against', 'there', 'you', 'until', 'been', 'hasn', 'has', 'don', 'wasn', 'co
uldn', 'after', 'themselves', 'didn', 'both', 'should', 'as', 'these', 'because', "couldn't", 'for', '"', 'haven', 'he', "sha
n't", 'very', 'only', 'by', 'll', 'y', 'again', 'why', 'ma', 'up', 'each', 'the', "mustn't", 've', 'shouldn', 'mustn', 'if',
't', 'are', 'having', "you'll", "hasn't", 'was', "it's", 'she', "aren't", 'shan', 'off', 'ourselves', 'own', 'when', 'theirs',
'we', ';', 'they', "that'll", 'how', '{', 'ours', 'were', 'no', 'while', ')', 'wouldn', "you've", 'on', 'o', 'which', 'any', 't
oo', 'than', 'some', 'his', 'is', 'then', 'and', 'over', 'further', 'between', 'so', 'can', 'in', 're', 'our', 'him', 'here',
"'", "wouldn't", 'more', 'a', 'under', '(', 'did', 'before', ':', 'ain', "needn't", 'this', ']', 'my', 'myself', 'herself', 'th
eir', 'where'}

1. Text is Tokenized by separating it into individual words.
1. All characters are converted into lower case in order to treat words such as “hello”, “Hello”, and “HELLO” the same
1. From Tokenizer class, text_to_sequence() function from Keras is used to fit on both training data and test data.
1. A set of unique words from the document is created with a vocabulary size is set at 10,000 words.
1. Once fit the Tokenizer provides 4 attribute
    1. word_counts: A dictionary of words and their counts.
    1. word_docs: A dictionary of words and how many documents each appeared in.
    1. word_index: A dictionary of words and their uniquely assigned integers.
    1. Document_count:An integer count of the total number of documents that were used to fit the Tokenize

## Implementation:

### Word Embedding:

First word embedding is using glove.840B.300d.txt with utf-8 encoding. The embedding matrix is created with shape(number of words *embedding dimension). In this project max number of words considered is 100,000 and embedding dimension is chosen as 300. Glove embedding is completed with the following result

No of word vectors - **2196017**

Total number of null word embedding - **23510**

Second word embedding is using crawl-300d-2M.vec with utf-8 encoding. As above embedding matrix is created with shape(number of words *embedding dimension). In this project max number of words considered is 100,000 and embedding dimension is chosen as 300.

Sample Embedded matrix with Common Crawl

```
[ -4.82100010e-01   8.85000005e-02  -7.81999975e-02  -7.46000037e-02
  -7.41000026e-02  -1.40000004e-02   2.10400000e-01   2.96499997e-01
   1.77800000e-01   2.38600001e-01   2.87600011e-01   3.69199991e-01
   1.35700002e-01   5.55899978e-01   2.47000009e-02  -1.18100002e-01
  -5.82999997e-02   2.19999999e-03  -1.43999998e-02  -3.82400006e-01
  -3.58300000e-01   1.94999993e-01  -2.10999995e-02   6.27999976e-02
  -2.46099994e-01   1.25599995e-01   1.03200004e-01   4.00999993e-01
```

**Neural Net architecture:**

When both the embedding is completed, they are combined together to be fed as the input to the neural net.

A bi-directional LSTM network is created with number of units as 10.

Return_sequence is set to true to return the last output in the output sequence.

A bi-directional GRU network is create with number of units as 10

Return_sequence is set to true to return the last output in the output sequence.

To reduce the dimensionality which in turn reduces overfitting two Global Pooling layers are added. Global average pooling layer and Global Max Pooling layer.

Followed by pooling layer two fully connected Dense layers are added and one is activated through ‘relu’ and the final through sigmoid activation function.

Finally, model is compiled with ‘binary crossentrophy’ loss function and ‘Adam’ Optimizer

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_1 (InputLayer) | (None, 250) | 0 | |
| embedding_1 (Embedding) | (None, 250, 300) | 30000000 | input_1[0][0] |
| embedding_2 (Embedding) | (None, 250, 300) | 30000000 | input_1[0][0] |
| concatenate_1 (Concatenate) | (None, 250, 600) | 0 | embedding_1[0][0]<br>embedding_2[0][0] |
| bidirectional_1 (Bidirectional) | (None, 250, 20) | 48880 | concatenate_1[0][0] |
| bidirectional_2 (Bidirectional) | (None, 250, 20) | 1860 | bidirectional_1[0][0] |
| global_average_pooling1d_1 (Glo | (None, 20) | 0 | bidirectional_2[0][0] |
| global_max_pooling1d_1 (GlobalM | (None, 20) | 0 | bidirectional_2[0][0] |
| concatenate_2 (Concatenate) | (None, 40) | 0 | global_average_pooling1d_1[0][0]<br>global_max_pooling1d_1[0][0] |
| dense_1 (Dense) | (None, 200) | 8200 | concatenate_2[0][0] |
| dense_2 (Dense) | (None, 6) | 1206 | dense_1[0][0] |

### Refinement

Word embedding was the most complicated part. Glove and Crawl vectors were close to 10 GB in size. To start with I didn't know how to use a ".vec" file with my training data. Had to refer multiple documentation and few implementation by github users to finally make it work. Going forward I will reuse the same code for word embedding in my future NLP project.

First the model was embedded with quarter of Glove data as it is 4 GB in total size. This lead to an 81 percent accuracy in predicting the toxic class. In order to improve the accuracy full Glove data was added along with Crawl vectors and bi – directional LSTM.

Model was fitted with 6 epochs initially with 300 units in the neural net. The model ran for 21 hours without any output. I reduced the number of units to 10 and epochs to 3. As we can see from the output accuracy didn't change much with increased number of epochs.

## RESULTS

### Model Evaluation and Validation

Since this is a multilabel classification task final evaluation metric is (ROC AUC) score. Area Under the Receiver Operating Characteristic (ROC), or simply ROC curve, is a graphical plot which illustrates the performance of a binary classifier system as its discrimination threshold is varied. So according to kaggle the model is evaluated on the mean column-wise ROC AUC. In other words, the score is the average of the individual AUCs of each predicted column.
By the same methodology model  and thanks to the readily available word embedding I was able to achieve an accuracy score of 98% after training on 143,613 samples and validating on 15,958 samples.

Below is the snapshot of the model output

```
Train on 143613 samples, validate on 15958 samples
Epoch 1/3
 - 1178s - loss: 0.0426 - acc: 0.9835 - val_loss: 0.0438 - val_acc: 0.9827
Epoch 2/3
 - 1290s - loss: 0.0399 - acc: 0.9842 - val_loss: 0.0424 - val_acc: 0.9834
Epoch 3/3
 - 1333s - loss: 0.0375 - acc: 0.9850 - val_loss: 0.0423 - val_acc: 0.9829
```

Robustness:

To check for Robustness I changed the batch size and the validation split. Previously I kept the fraction of the training data to be used in validation as 10% of the training data.The model will set apart this fraction of the training data, will not train on it, and will evaluate the loss and any model metrics on this data at the end of each epoch. I changed it to 20% .Changed the number of epochs from 6 to 3 to 1 and the number of samples per gradient update from 80 to 40 . With all these changes the accuracy score always remained above 98% . I modified the training data by removing threat column and trained it using 5 labels. The result came back unchanged at 98%.

## Justification

The output is present in the submission file which has the probability of every comment being in any of the of the six possible types of comment toxicity (toxic, severe_toxic, obscene, threat, insult, identity_hate).

## Benchmark

There are two benchmark model against which this is compared. One is a Random Forest Classifier model posted by a Kaggle user and the other one is a simple Logistic Regression model created by me. The Random Forest Classifier model resulted in an accuracy score of 91.5% while the Logistic Regression classifier created by me lead to an 97% accuracy.

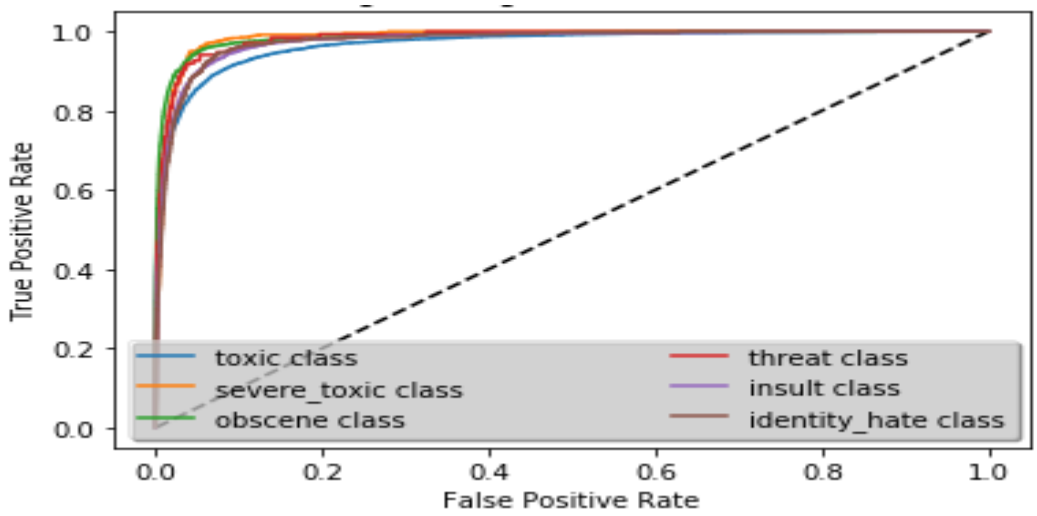
```
losses = []
test_predictions  = {'id': test['id']}
valid_predictions = {'id': valid['id']}

for class_name in class_names:
    train_target = train[class_name]
    classifier = LogisticRegression(solver='sag')

    cv_loss = np.mean(cross_val_score(classifier, train_features, train_target, cv=3, scoring='roc_auc'))
    losses.append(cv_loss)
    print('CV score for class {} is {}'.format(class_name, cv_loss))

    classifier.fit(train_features, train_target)
    test_predictions[class_name] = classifier.predict_proba(test_features)[:, 1]
    valid_predictions[class_name] = classifier.predict_proba(valid_features)[:, 1]
```

CV score for class toxic is 0.9703823806373831
CV score for class severe_toxic is 0.9850204552076671
CV score for class obscene is 0.9840357267528151
CV score for class threat is 0.9854014125653093
CV score for class insult is 0.9774041293366643
CV score for class identity_hate is 0.9741863733994759

The final result produced in this project through LSTM/GRU networks exceeded my expectation. I wanted to achieve an accuracy score better than the supervised model which was the benchmark and I was able to achieve an accuracy score of 98%. The Kaggle leaderboard has accuracy of over 99% with RNN models. I am glad that I am able to achieve a similar result.

## Conclusion

To validate the model and to avoid getting trapped into accuracy paradox created the ROC curve for various classes. In a Receiver Operating Characteristic (ROC) curve the true positive rate (Sensitivity) is plotted in function of the false positive rate (100-Specificity) for different cut-off points. All the curves plotted follows close to the left hand border and then top of the ROC space. This indicates that maximum number of space is available under the curve indicating text accuracy. Prediction for all the six classes has an ROC score above 0.95




### Reflection

Even though the data is highly unstructured it is astounding to find that a deep neural network has the ability to learn from this data and classify it accurately. Going forward as bulk of the data being generated are going to

be unstructured in nature, be it social media comments, photos or texts, manually going through each and every comment and flagging it is nearly impossible. At the same time since it is a free text field we cannot write rules based programming to tackle this problem as well. Machine learning especially deep learning is the only viable option for text/photo classification.

One of the difficult aspect in the above problem is understanding the context. The grammar in the sentence or word construct or spelling are almost always incomplete. Apart from just using stop words I had to find few other non-alphanumeric characters that I had to remove while data preprocessing.
Second difficulty was in understanding the different classes of classification. For example, there is no clear difference between a severe_toxic and obscene class. Final challenge I faced was the training time. When the number of epochs and units were high the network didn't complete its training even after 18 hours.

### Future Direction:

I am planning to further expand this learning by applying similar technique on large datasets using pyspark on a GPU to predict the sentiment in real-time. The learning from this could be directly applied on various projects at work.
Moreover, in many real world comments there are a high usage of emoticons, GIF files and link to other URLS. For our analysis we removing all these while predicting a sentiment. If those emotions are also captured we could derive more insight from the data.

## Reference:


https://nycdatascience.com/blog/student-works/predicting-stock-movement-of-hang-sengs-components/

https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

https://towardsdatascience.com/introduction-to-sequence-models-rnn-bidirectional-rnn-lstm-gru-73927ec9df15

https://blog.insightdatascience.com/how-to-solve-90-of-nlp-problems-a-step-by-step-guide-fda605278e4e

https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge#evaluation

http://www.letscodepro.com/Twitter-Sentiment-Analysis/

https://datahack.analyticsvidhya.com/contest/practice-problem-twitter-sentiment-analysis/

https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python/

http://scikit-learn.org/stable/modules/model_evaluation.html#roc-metrics

https://www.analyticsvidhya.com/blog/2018/02/natural-language-processing-for-beginners-using-textblob/#comment-151335

https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python/

http://scikit-learn.org/stable/modules/cross_validation.html

```
for class_name in class_names:
    train_target = train[class_name]
    classifier = LogisticRegression(solver='sag')

    cv_loss = np.mean(cross_val_score(classifier, train_features, train_target, cv=3,
scoring='roc_auc'))
    losses.append(cv_loss)
    print('CV score for class {} is {}'.format(class_name, cv_loss))

    classifier.fit(train_features, train_target)
    test_predictions[class_name] = classifier.predict_proba(test_features)[:, 1]
    valid_predictions[class_name] = classifier.predict_proba(valid_features)[:, 1]

print('Total CV score is {}'.format(np.mean(losses)))
```

```
CV score for class toxic is 0.9703823806373831
CV score for class severe_toxic is 0.9850204552076671
CV score for class obscene is 0.9840357267528151
CV score for class threat is 0.9854014125653093
CV score for class insult is 0.9774041293366643
CV score for class identity_hate is 0.9741863733994759
Total CV score is 0.979405079649886
```