Machine Learning Engineer Nanodegree

Capstone Project

Prateek Gupta
June 17th, 2018

I. Definition

Project Overview

This project aims to identify and classify a toxic comment during online conversation. Social networking sites and user groups allow people to come together and have an open discussion on topics they care about. Platforms intend to facilitate conversations without being harassed by people with different opinions. Certain platforms differentiate themselves by allowing mature conversations, while discouraging harsh comments.

This is a case of a *multi label classification* problem, which can be tackled by using Supervised learning algorithms.

Problem Statement

Being able to successfully predict the toxicity class of a comment can have immense benefits across the board for social media applications and can be help in keeping online discussions free of harassment and abuses. For this problem, we want to create a machine learning model that can be used to identify and classify toxic comments during online conversation. The deep learning model will be able to take a new conversation as input, pass it through the network and classify the level of toxicity. Some comments can fall under multiple buckets and hence the problem becomes a multi-label classification problem. In our current problem, a comment can have multiple labels (among toxic, severe_toxic, obscene, threat, insult and identity_hate).

Metrics

Since this is a classification problem, we will be using *ROC AUC* function under the *sklearn* library to compute the area under the Receiver Operation Characteristics Curve

(ROC AUC) from the prediction scores. The AUC values between 0.5 to 1 denotes an excellent classifier.

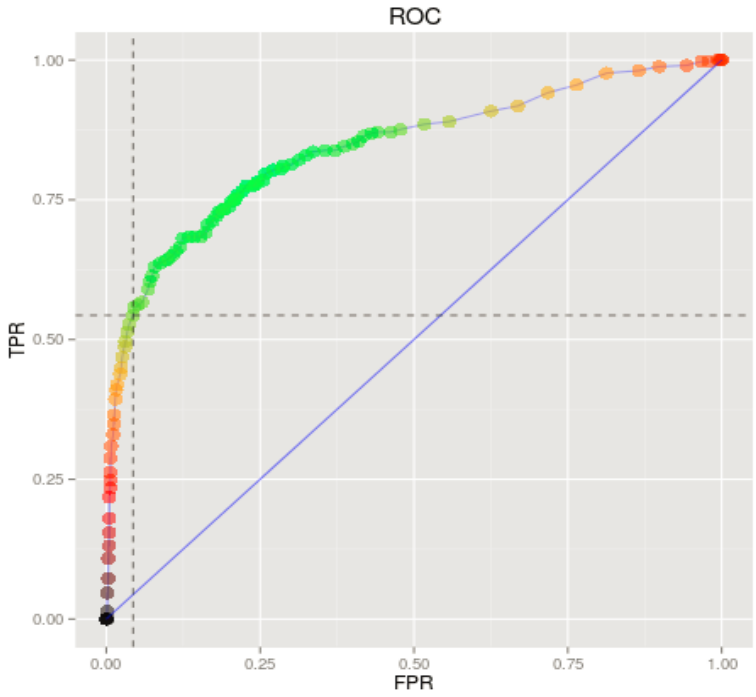
ROC is used to understand how to balance false positives and false negatives. For an imbalanced classification problem, accuracy as an evaluation metric, does not consider in-class misclassification. For example, marking all comments as nontoxic, may give us a good overall accuracy, but the model may work very poorly for identifying individual classes.

$$ROCx(\theta) = False\ Positive\ Rate(\theta) = \frac{False\ Positive(\theta)}{False\ Positive(\theta) + True\ Negative(\theta)}$$

$$ROCy(\theta) = True\ Positive\ Rate(\theta) = \frac{True\ Positive(\theta)}{False\ Negative(\theta) + True\ Positive(\theta)}$$

In terms of *hypothesis testing* where rejecting the null hypothesis is considered a positive result, the False Positive Rate corresponds to *Type I* error, the False Negative Rate corresponds to *Type II* error.

Plotting True Positive Rate against False Positive Rate, gives us the ROC curve. Calculating area under the curve (AUC) can be used to score the model. An excellent model should score value very close to 1.



Source: https://www.joyofdata.de/blog/illustrated-guide-to-roc-and-auc/

Additionally, we will be benchmarking the model using *zero rule algorithm*. For classification problems, *zero rule algorithm* is to predict the class value that is most common in the training dataset. In our case, we will be predicting all labels as non-toxic and then use it as another benchmarking.

II. Analysis

Data Exploration

In this section, we will first look at the sample data and understand the various columns.

We extract some sample Data for the training dataset

```
train.iloc[10:17]
```

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
10	0005300084f90edc	"\nFair use rationale for Image:Wonju.jpg\n\nT...	0	0	0	0	0	0
11	00054a5e18b50dd4	bbq \n\nbe a man and lets discuss it-maybe ove...	0	0	0	0	0	0
12	0005c987bdfc9d4b	Hey... what is it..\n@ | talk .\nWhat is it.....	1	0	0	0	0	0
13	0006f16e4e9f292e	Before you start throwing accusations and warn...	0	0	0	0	0	0
14	00070ef96486d6f9	Oh, and the girl above started her arguments w...	0	0	0	0	0	0
15	00078f8ce7eb276d	"\n\nJuelz Santanas Age\n\nIn 2002, Juelz Sant...	0	0	0	0	0	0
16	0007e25b2121310b	Bye! \n\nDon't look, come or think of comming ...	1	0	0	0	0	0

Here, comment_text contains the comment from the user. Columns, toxic, severe_toxic, obscene, threat, insult and identity_hate contains binary values (0/1). Id 0005c987bdfc9d4b, has been classified as a toxic comment (column toxic has value 1 for that particular comment).

Exploratory Visualization

> In this section, we will present a visualization and provide our understanding. Some of the charts that we intend to look are frequency distribution charts and correlation matrix.
>
> The frequency distribution chart will tell us if the classes are imbalanced and correlation matrix will be used to identify labels which are highly correlated to each other.

- Frequency Distribution

	count
toxic	15294
obscene	8449
insult	7877
severe_toxic	1595
identity_hate	1405
threat	478

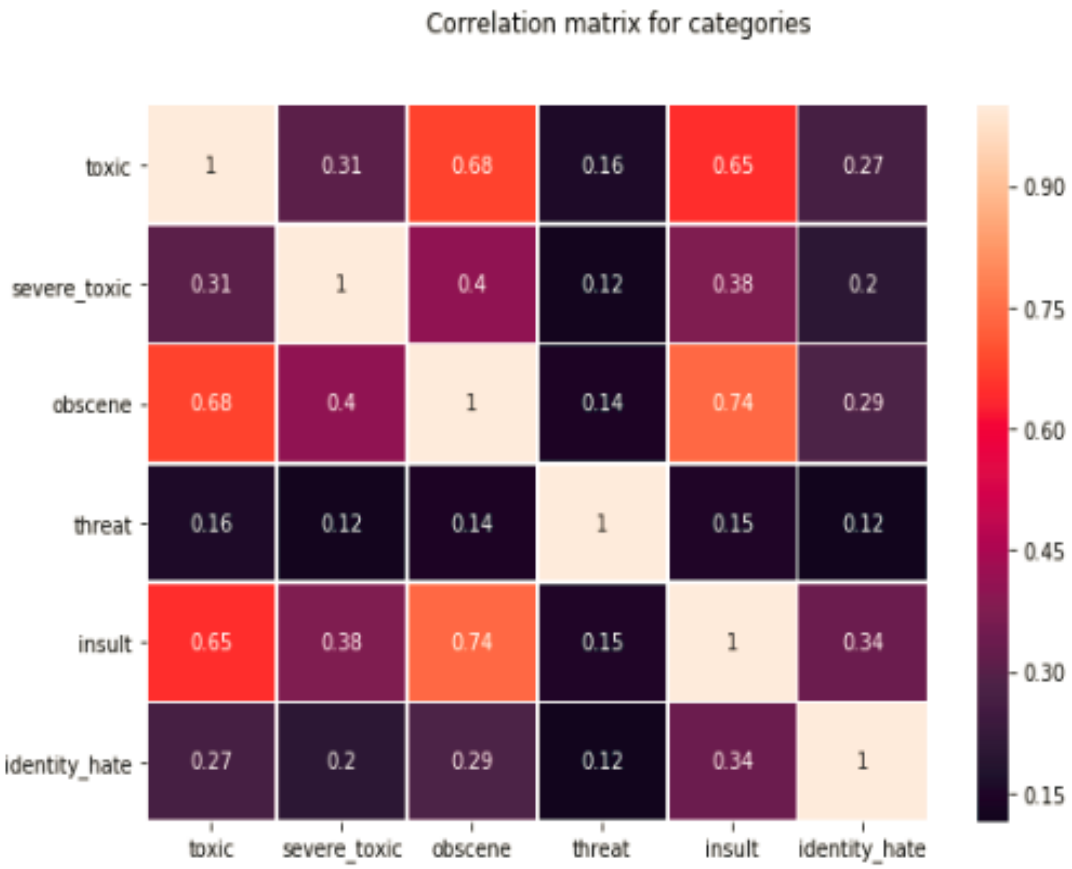
The frequency distribution chart tells us that the data has highly imbalanced classes. Three major labels are toxic, obscene and insult. The count reduces drastically for labels severe_toxic, identity_hate and threat.

	toxic	severe_toxic	obscene	threat	insult	identity_hate	count
0	0	0	0	0	0	0	143346
1	1	0	0	0	0	0	5666
2	1	0	1	0	1	0	3800
3	1	0	1	0	0	0	1758
4	1	0	0	0	1	0	1215
5	1	1	1	0	1	0	989
6	1	0	1	0	1	1	618
7	0	0	1	0	0	0	317
8	0	0	0	0	1	0	301
9	1	1	1	0	1	1	265

From this table, we see certain comments falling under multiple labels. Majority of the comments are clean.

Next, we will try to see the correlation among labels.

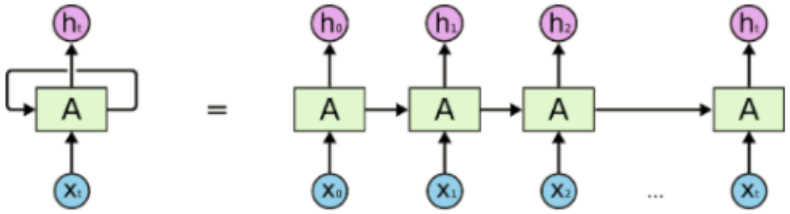
Correlations Between target variables



From the above correlation matrix, we find labels with obscene, toxic and insults to be highly correlated with each other.

Algorithms and Techniques

For this problem, we intend to use recurrent neural network. Persistence of information is important for understanding context from a sentence. Traditional neural networks do not have this capability. RNN (Recurrent neural network) can address this issue by allowing information to pass through loops. In cases, where the gap between the relevant information and the place where it is needed is small, RNN can be used to learn past information.

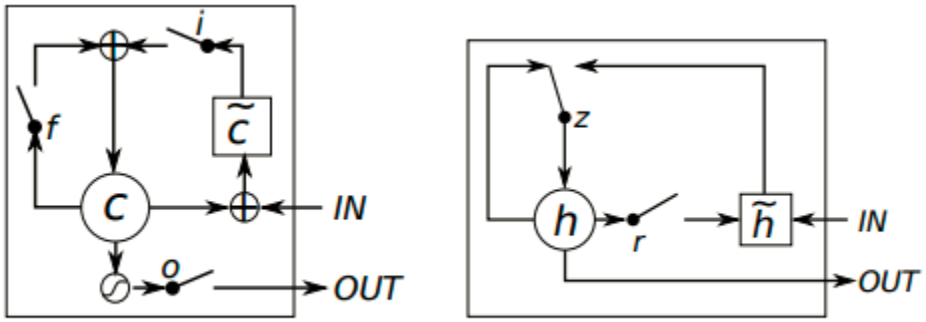


An unrolled RNN

a) An Unrolled RNN- Network takes both the output of network from the previous timestep as input and uses the internal state from the previous time step as a starting point for the current time step.

LSTM (Long Short Term Memory) is a special type of RNN, which can learn to connect the information with long term dependencies.

GRU (Gated Recurrent Unit) uses gating mechanism in RNN. The absence of output gate reduces the number of parameters as that of LSTM and provides better performance over LSTM over smaller datasets.



(a) Long Short-Term Memory (b) Gated Recurrent Unit

a) LSTM - i, f and o are the input, forget and output gates respectively. c and $\tilde{c}$ denote the memory cell and the new memory cell content. A single unit makes decision by considering the current input, previous output and previous memory. It then generates a new output and alters its memory. If we shut the forget gate, no old memory will be kept and if we open it, all old memory will pass through. New memory will come in through a T shaped joint and merge with old memory. Exactly how much new memory should come in is controlled by the input gate. The output gate is controlled by the new memory, the previous output value and the current input value. This gate will control how much new memory should output to the next LSTM unit.
b) GRU – r and z are the reset and update gates, h and $\tilde{h}$ are the activation and the candidate activation. Reset gate is used from the model to decide how much of the past information to forget. Update gate holds information for the current unit and passes it down the network. It determines what to collect from the current memory content and what to collect from the previous steps.

Source:

1. A Beginner's Guide to Recurrent Networks and LSTMs https://deeplearning4j.org/lstm.html
2. Long short-term memory – Wikipedia https://en.wikipedia.org/wiki/Long_short-term_memory
3. Gated Recurrent Unit – Wikipedia https://en.wikipedia.org/wiki/Gated_recurrent_unit
4. LSTM Diagram Explanation- https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714
5. GRU Diagram Explanation - https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

Benchmark

As this is a Kaggle competition, we will be submitting the solution via a late submission feature. Kaggle will bench mark our model against other models and provide a score. The AUC values between 0.5 to 1 denotes an excellent classifier. The winning entry has a score of 0.9885 and a personal goal would be to get a score 0.95+ ROC AUC result.

Further, we will be using Bidirectional LSTM as a benchmarking model and compare the performance with that of Bidirectional GRU.

III. Methodology

Data Preprocessing

The data for natural language processing is highly unstructured and noisy in nature. To achieve better insights and build algorithms, it is necessary to clean the data.

1. Converting to Unicode and removing special characters

   Before

   ```
   Explanation\nWhy the edits made under my username Hardcore Metallica Fan were reverted? 
   ```

   After

   ```
   Explanation Why the edits made under my username Hardcore Metallica Fan were reverted?
   ```

2. For the new features created (caps_vs_length and words_vs_unique), the data was scaled to 0 mean and unit variance using the *StandardScaler* class in *sklearn.preprocessing* module

3. Stopwords – stopwords were not removed as presence of multiple referrals can point to a certain toxicity

4. Words were tokenized using keras preprocessing text module and upper capped to 20000 features

5. We used embedding layers instead of one hot encoding. One hot encoded vectors are high-dimensional as well as sparse. In a big dataset, this will not be computationally efficient. Embedding layers also provide the model to take relationships in language into consideration, which is not possible in one hot encoding.

   For data corpus, we used GloVe Twitter (unsupervised learning algorithm for obtaining vector representations for words)

   Source: https://machinelearningmastery.com/develop-word-embedding-model-predicting-movie-review-sentiment/

   Example for embedding vector for word 'hate'

```
raw_embedding['hate']
```

```
array([  8.78119990e-02,   1.55090004e-01,   1.90430004e-02,
         3.27939987e-01,  -6.28650010e-01,   3.83540004e-01,
         1.08739996e+00,  -2.26730004e-01,  -2.04949994e-02,
         1.97310001e-01,  -9.79510043e-03,  -4.54270005e-01,
        -1.02190006e+00,  -1.37490004e-01,  -4.46559995e-01,
        -4.90740001e-01,  -2.70590007e-01,  -1.48629993e-01,
        -4.65490013e-01,   7.83940032e-02,   6.30289972e-01,
        -1.65370002e-01,   3.91750008e-01,  -4.56999987e-02,
         4.64299992e-02,   1.02579999e+00,  -2.03060001e-01,
         2.76069999e-01,  -4.07189995e-01,  -2.65980005e-01,
        -2.86119998e-01,  -5.63709974e-01,  -4.23070014e-01,
         5.34690022e-01,  -6.23390019e-01,  -2.11080000e-01,
        -2.16120005e-01,   1.72849998e-01,   8.18099976e-01,
```

Implementation

Main Packages Used:

1. Pandas – 0.22.0
2. Numpy – 1.14.2
3. Sklearn – 0.19.1
4. Keras – 2.1.5

The model is implemented in the following way:

1. Slicing the embedding vector to take max_features into vocabulary size, instead of using the complete vocabulary
2. Using embedding vector to encode the words into vectors
3. Passing the vectors through bidirectional GRU (See Model Summary Image)
4. Combining the output of bidirectional layer with input features
5. Pooling the layers to reduce computation
6. Connecting to a dense layer with sigmoid activation
7. Compiling the model with binary cross entropy as loss function, adam optimizer with controlled gradient clipping

Model Summary:

```
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input_2 (InputLayer)         (None, 50)                0
_________________________________________________________________________________________________
EmbeddingLayer (Embedding)   (None, 50, 200)           4000000     input_2[0][0]
_________________________________________________________________________________________________
BidirectionalGRU (Bidirectional [(None, 50, 80), (No 57840       EmbeddingLayer[0][0]
_________________________________________________________________________________________________
global_average_pooling1d_1 (Glo (None, 80)           0           BidirectionalGRU[0][0]
_________________________________________________________________________________________________
global_max_pooling1d_1 (GlobalM (None, 80)           0           BidirectionalGRU[0][0]
_________________________________________________________________________________________________
input_1 (InputLayer)         (None, 2)                 0
_________________________________________________________________________________________________
concatenate_1 (Concatenate)  (None, 202)           0           global_average_pooling1d_1[0][0]
                                                               BidirectionalGRU[0][1]
                                                               global_max_pooling1d_1[0][0]
                                                               input_1[0][0]
_________________________________________________________________________________________________
dense_1 (Dense)              (None, 6)             1218        concatenate_1[0][0]
=================================================================================================
Total params: 4,059,058
Trainable params: 59,058
Non-trainable params: 4,000,000
_________________________________________________________________________________________________
```

Model Fitting

1. To avoid over activation of specific nodes, spatial drop out with 0.5 rate is used
2. Number of epoch is set to 5, and number of folds in KFold cross validation is set to 5
3. A custom function – ROCAUCEvaluation, is written to record roc_auc_score at epoch end
4. Keras metric – 'accuracy' is used to track performance during epoch run

List of Algorithms Used:

1. Bidirectional LSTM
2. Bidirectional GRU
3. Zero Rule Algorithm

   Bidirectional LSTM and Zero Rule Algorithm are used as alternative models for benchmarking.

Refinement

For refining the model, I tweaked the following:

1. Epochs: Single Pass through the entire dataset while training the model
2. num_folds: Splits dataset into k folds with each fold set once as validation set, while the remaining k-1 folds form the training set
3. max_features: maximum number of words to keep, based on word frequency
4. features: identifying features that can represent the underlying problem to the model
5. Hidden layers: Layers between input and output layers, where neurons take a set of weighted inputs and produce an output through an activation function

For this problem, increasing the epoch and num_folds increased the training time drastically with improvement in accuracy. Increasing the vocab size increased the accuracy, and it started decreasing (below 95%) in value post 30000 words. Features such as character length, word length and number of upper words were considered. Finally, derived metrics, caps_vs_length and words_vs_unique are used in the model to decrease the number of parameters in the model.

Increasing the complexity of model by adding hidden layers (bidirectional GRU+LSTM) did not increase accuracy than a single bidirectional GRU,

IV. Results

Model Evaluation and Validation

Parameters and Kaggle Scoring

	Epochs	num_folds	max_features	Private Score - ROC AUC Score	Public Score - ROC AUC Score
Bidirectional GRU	5	5	20000	0.9784	0.9813
Bidirectional GRU + Bidirectional LSTM	1	2	20000	0.9657	0.9682
Zero Rule Algorithm	-	-	-	0.5	0.5

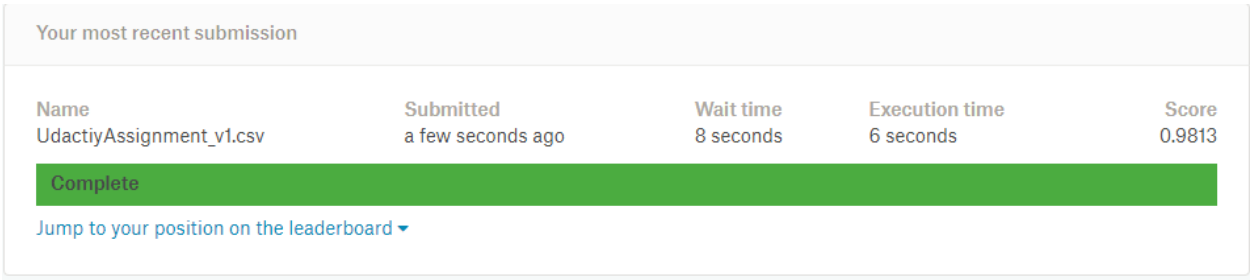
As observed from the results, Bidirectional GRU performs better than both bidirectional GRU+bidirectional LSTM combination and Zero Rule Algorithm in terms of ROC-AUC. The training time for bidirectional GRU is higher than the other two models because of higher number of epochs.

Additional Information on Models

Kaggle Scores:

Here we are showing the Kaggle score we obtained on making the submissions for different models

Bidirectional GRU (Final Model)



Bidirectional GRU+ Bidirectional LSTM (Alternate Model)

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
UdactiyAssignment_v1.csv	a few seconds ago	2 seconds	6 seconds	0.9670

Complete

Jump to your position on the leaderboard

Zero Rule Algorithm: Predicting all labels as non toxic

Name	Submitted	Wait time	Execution time	Score
UdactiyAssignment_ZR.csv	just now	0 seconds	4 seconds	0.5000

Complete

Jump to your position on the leaderboard

Bidirectional GRU+Bidirectional LSTM Model Summary

Layer (type)	Output Shape	Param #	Connected to
input_10 (InputLayer)	(None, 50)	0	
EmbeddingLayer (Embedding)	(None, 50, 200)	4000000	input_10[0][0]
spatial_dropout1d_4 (SpatialDro	(None, 50, 200)	0	EmbeddingLayer[0][0]
BidirectionalLSTM (Bidirectiona	(None, 50, 80)	77120	spatial_dropout1d_4[0][0]
BidirectionalGRU (Bidirectional	[(None, 50, 80), (No	29040	BidirectionalLSTM[0][0]
global_average_pooling1d_4 (Glo	(None, 80)	0	BidirectionalGRU[0][0]
global_max_pooling1d_4 (GlobalM	(None, 80)	0	BidirectionalGRU[0][0]
input_9 (InputLayer)	(None, 2)	0	
concatenate_4 (Concatenate)	(None, 202)	0	global_average_pooling1d_4[0][0] BidirectionalGRU[0][1] global_max_pooling1d_4[0][0] input_9[0][0]
dense_4 (Dense)	(None, 6)	1218	concatenate_4[0][0]

Total params: 4,107,378
Trainable params: 107,378
Non-trainable params: 4,000,000

Justification

Based on the improved Kaggle score, the final tuned model can be deemed as a satisfactory solution. A single bidirectional GRU with more epochs performed better than a bidirectional GRU+LSTM combination with lesser epochs. It also performed better than the zero rule algorithm. We are also able to beat the personal target of 0.95+ ROC AUC score.

V. Conclusion

Free-Form Visualization

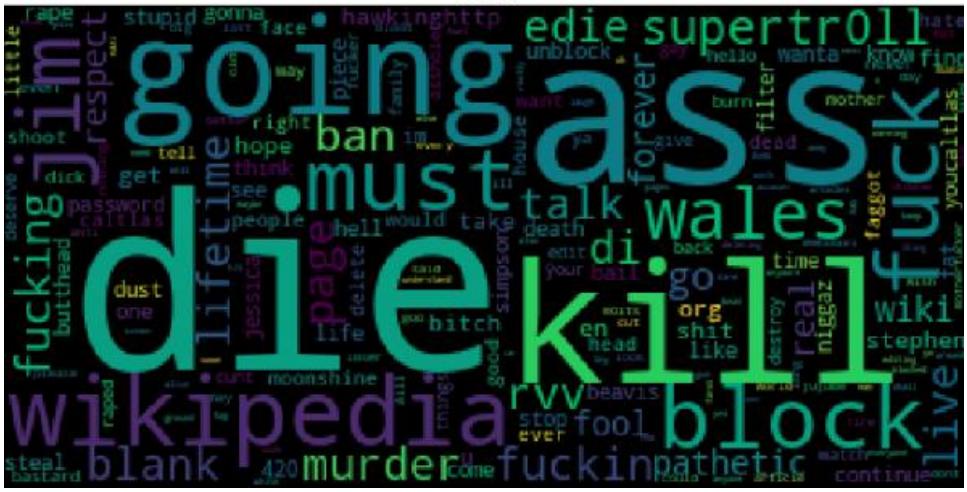
The word cloud gives an idea the words that are appearing in higher frequencies. Comparing the data with the correlation results, the presence of certain words makes it fall in multiple buckets (the word cloud for labels – toxic, insult and obscene is similar to each other. These words can be used to make special features for increasing accuracy.

Below, we are plotting the word cloud for toxic, severe_toxic, obscene, threat, insult, identity_hate.

severe_toxic

obscene

threat



insult

identity_hate

Reflection

This project can be summarized as the sequence of following steps:

1. Searching for a relevant problem across Kaggle and deciding between a classification or a regression problem
2. Understanding the data and visualizing various aspects of dataset
3. Preprocessing the data
4. Feature engineering and selection
5. Deciding the algorithms to be used to solve the problem
6. Applying the selected algorithm and compare results with Kaggle benchmarks
7. Hyperparameter tuning to get the best results with trade off between computation time and accuracy

Step 1 was the most challenging. I approached the problem by reading up material on recurrent neural network, LSTM and GRU. This included listening to video lectures from Andrew NG's deeplearning.ai course. Understanding embedding layers and its application proved to be another difficult task. Hyperparameter tuning was the biggest bottleneck due to absence of a GPU and the hardware limitation imposed by Kaggle on running the code on their server. Number of epochs and number of folds had to be significantly reduced.

Visualization was fun as there were multiple overlaps across labels. I tried implementing Venn diagram and then resorted to tried and tested correlation tables.

Improvement

The solution can be improved in the following ways:

1. Performing more aggressive feature engineering – From the word cloud, we see that particular words can be tagged to multiple classes. Creating a vocabulary that can take care of grammars, spell checks and variations of the word can be used to create flags/features. This method can be used for increasing accuracy.
2. Using fasttext and GloVe vocabularies in conjunction to create a larger vocabulary - More vocabulary can help in identifying better relationships among words and improve classification.
3. Performing aggressive hyperparameter tuning - Increasing number of epochs may further improve the model. Usually, "the learning path" of model will indicate if more epochs would enhance the results. We also try increasing the number of folds and the batch size. We can visualize the history output during model fitting and tune accordingly.

References

1. Kaggle Toxic Comment Classification Challenge https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge
2. A Beginner's Guide to Recurrent Networks and LSTMs https://deeplearning4j.org/lstm.html
3. Long short-term memory – Wikipedia https://en.wikipedia.org/wiki/Long_short-term_memory
4. Gated Recurrent Unit – Wikipedia https://en.wikipedia.org/wiki/Gated_recurrent_unit

5. Conversation AI https://conversationai.github.io/
6. GloVe - https://nlp.stanford.edu/projects/glove/
7. Embedding Vectors- https://machinelearningmastery.com/develop-word-embedding-model-predicting-movie-review-sentiment/
8. ROC - https://www.joyofdata.de/blog/illustrated-guide-to-roc-and-auc/
9. Zero Rule Algorithm - https://machinelearningmastery.com/implement-baseline-machine-learning-algorithms-scratch-python/
10. LSTM Diagram Explanation- https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714
11. GRU Diagram Explanation - https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be