Toxic Comments Classifier (Team 899129)

Satvik Sunkam Ramaprasad
satviksunkam.ramaprasad@iiitb.org
IMT2016008

Raghavan G V
raghavan.gv@iiitb.org
IMT2016099

Ananth Shreekumar
ananth.shreekumar@iiitb.org
IMT2016129

***Abstract*—As the internet is gaining popularity day by day, more people actively express their views in various forums by writing blogs and comments. Some of the comments may be vituperative or abusive, and it is the responsibility of the platform to identify such content and take appropriate action. Since it is impossible to manually search for such comments, there is a need to build a model that can classify text into different levels of obscenities.**

***Index Terms*—Machine Learning, Logistic Regression, XGBoost, Random Forest, NB - SVM, Blending, Stacking**

I. Introduction

The Toxic Comments Classifier Problem was hosted on Kaggle by The Conversation AI team, a research initiative founded by Jigsaw and Google (both a part of Alphabet). They are working on tools to help improve online conversation. One area of focus is the study of negative online behaviors, like toxic comments.

Here, we try to build a model that classifies an input text into categories - toxic, severe_toxic, obscene, threat, insult, and identity_hate. A given piece of text could belong to none, one or many of these categories. Hence, the problem is a MultiLabel Classification Problem.

II. Data

The data set was provided on Kaggle. The data set included two files:

- train.csv - the training set
- test.csv - the test set

The training data set contained 140K data points. The test data set contains 10000 data points.

Data Attributes for train.csv

- id - unique id value for a comment
- comment text - the comment content
- toxic - boolean field to indicate toxic
- severe_toxic - boolean field to indicate severe_toxic
- obscene - boolean field to indicate obscene
- threat - boolean field to indicate threat
- insult - boolean field to indicate insult
- identity_hate-boolean field to indicate identity_hate

The test data set contains 10000 data points.

Data Attributes for test.csv

- id - unique id value for a comment
- comment text - the comment content

Additionally, the dataset also contained a sampleSubmission.csv file, which was a sample submission file in the correct format for reference.

III. Data Preprocessing

Since the given comment_text was just the comments taken off of the internet, preprocessing the data was required to make it 'cleaner'. We had several ideas in mind.

Preprocessing steps

- Remove punctuation
- Remove excess white space
- Stemming of words
- Convert to lowercase
- Remove accents
- Grammar modifications like "don't" $\rightarrow$ "do not"
- Replacing emoticons such as ":)" $\rightarrow$ "smile"
- Removing stop words such as "the", "a"
- Lemmatization of words

We first removed excess white spaces. This did not affect our score. Tfidf vectorizer appears to handle excess white spaces. Then, we tried to remove punctuation from the comments, as we thought punctuation will confuse the model. However, contrary to our belief, the punctuation did actually help the model, so we decided to include punctuation in the comment text.

We then decided to clean the data using grammar. We replaced words like 'it's' with 'it is', ''nt' with ' not', in the hope that the model could recognize words better.

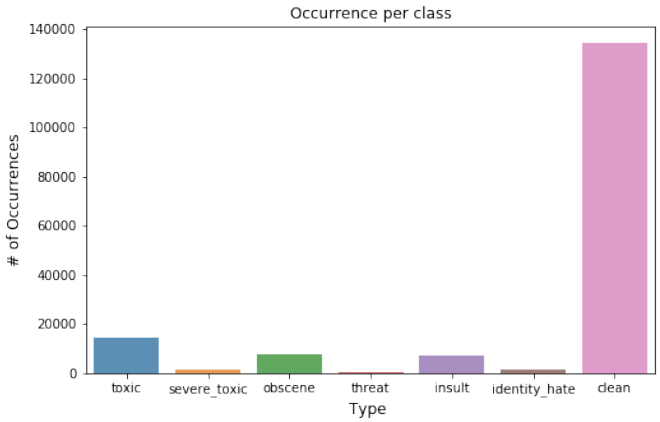
However, this did not improve our score at all, so we decided to use the original comments.

We tried applying Stemming and lemmatization using the nltk library. Both approaches gave marginal improvement in cross validation score. However on submission, Kaggle reported a lower score. The process was extremely slow and resource costly. Therefore we decided to proceed without this preprocessing step.

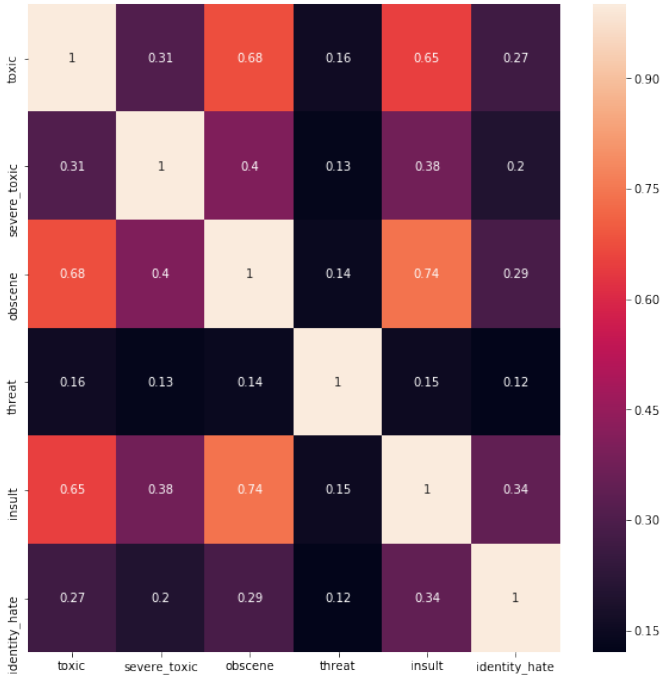
Converting all text to lowercase and removal of accents helped. Further, removal of stop words helped.

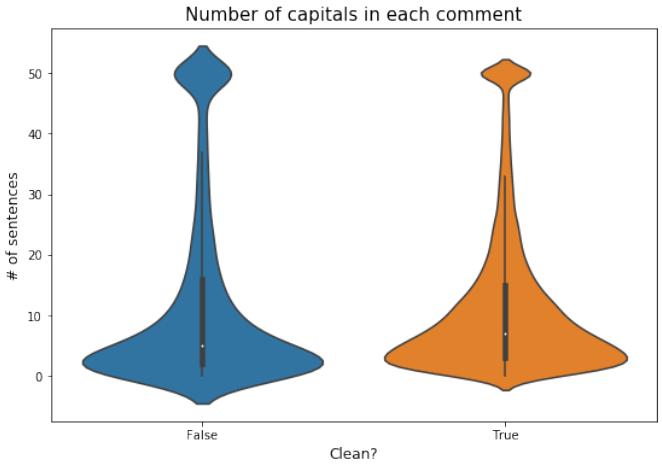
IV. Exploratory Data Analysis

Our training dataset was extremely skewed as shown below: We tried to look at the correlation between the



labels using the heatmap given below. A high correlation between two labels means that those two labels usually occur together. We tried to check if the number of capital letters in 'clean' comments actually varies from the 'dirty' comments. However, as the violin plot below shows, there is almost no difference between the two.





V. Feature Engineering

A. Extracting simple features

We believed, since the only feature we were given was the comment_text, adding more features from the comment_text was likely to increase the model accuracy. We figured that the fraction of capital letters in a given text might be indicative of vituperative text. Hence, we added a feature that was the number of capital letters in the comment, normalized by the length of the comment. Similarly, the number of punctuations in a given text could also indicate abusive content. We added another feature that was the number of punctuations in the comment, normalized by the comment length.

Extracted Features

- Length of comment
- Number of capitals
- Number of capitals/ Length of comment
- Number of exclamation marks
- Number of question marks
- Number of symbols ($%#* and others)
- Number of smilies

These features can be seen in `/jupyter/SimpleFeatures.ipynb`.

B. Selection of important words/tokens

Our data set was very large. Therefore, Tf-idf extracts a lot of words from the data set. However, we noticed that, by limiting the vectorizer using parameters like min_df, max_df and max_features, we were losing out on a few important words. This was more pronounced, when we tried using bi-grams and tri-grams. We believed that the high number of features would slow down the training process.Also, the model might have difficulty converging.

We decided to select important words and tokens by considering their absolute correlation with the label. We used a CountVectorizer, set it to binary and found the exhaustive vocabulary of the text. We then found absolute correlation with the word/token and the class(toxic, obscene etc). We used absolute correlation, because we needed tokens with high positive correlation as well as those with high negative correlation.

Since the data set was very large, we had to split the process into several batches and finally aggregate the results. The process took several hours using multiple PCs. The results can be found in `/features` folder. The code can be found in `/src/FeatureEngineering.py` and `/src/FeatureAggregator.py`

VI. Code structure

We have used object oriented code right from the start. We realized that lot of the code for the different models are similar. However the features and preprocessing are not similar. Therefore we created an abstract base class called `BaseModel`. All models inherit from it and implement the functions.

This structure also highly helped in implementing our custom blender class `Blender`. Since each model was a separate class and can be pickled individually, we could run our blending process smoothly and fast.

We used jupyter for development, all notebooks can be found in `jupyter`. The code from jupyter notebooks has been exported to `src`. Submission files for all models can be found in `answers`.

```
Toxic Comment Classifier
├── jupyter
│   └── EDA2.ipynb
├── src
│   ├── BaseModel.py
│   ├── LogisticRegression3.py
│   ├── NBSvm1.py
│   └── Blender2.py
├── answers
│   └── LogisticRegression3_submission.csv
├── input
│   ├── train.csv
│   ├── test.csv
│   ├── cleaned_train.csv
│   └── cleaned_test.csv
├── pickle
├── features
│   └── toxic
│       └── 1.json
├── results
└── requirements.txt
```

VII. Models and Training

A. Model using simple features

File: `/src/SimpleFeatures.py`

To start off, We used the simple features that we extracted in section V-A. We used logistic regression to build the model. The cross validation score was 0.71. Stacking the model with itself, the score rose to 0.75, which was pretty good considering that this model was built using a small portion of the data set. However, when we tried appending these features with the vectorized features from tf-idf, the score remained unchanged. We then decided to try both stacking and blending. Unfortunately, stacking did not help at all. Blending was not applicable, because other models were giving a cross validation score of 0.98, so the score was bound to decrease.

B. Model using word tokenizer

File: `/src/LogisticRegression1.py`
We used TfIdfVectorizer to build a word tokenizer. We used an ngram range of (1, 1), so that each word is considered a feature. This gave a decent cross validation score of 0.98178 with logistic regression. When we tried using higher ngrams such as bi-grams and tri-grams, the score seemed to decrease.

C. Model using character tokenizer

File: `/src/LogisticRegression2.py`
Using the TfIdfVectorizer we built a character tokenizer and an ngram range (2, 4) gave a better score of 0.98422 with logistic regression.

D. Model using both words and character tokenizer

File: `/src/LogisticRegression3.py`
We stacked the features of both word tokenizer and character tokenizer. This significantly improved the score to 0.98512. Using grid search and fine tuning, the score only increased marginally. We seemed to have hit an upper limit of 0.9855.

E. XGBoost and RandomForest

File: `/src/RandomForest1.py`,
When we tried XGBoost, the model score was very far behind logistic regression. Even random forest was slightly behind logistic regression. Considering the large number of features, we believe that XGBoost and Random Forest overfit on the data. This would explain the low score.

F. NB-Svm

File: `/src/NBSvm1.py`
Since we got the best score with Logistic Regression, we decided to use some similar models, like Naive Bayes SVM. By using NB-Svm, we got our best single model score of 0.98669. The model was trained on both word and character tokenized features.

G. Blending and Stacking

We used blending of multiple models to get our best score. We decided not to use Voting Classifier, because that would need to train all the models individually for each blend. Moreover, it was not easy to set the weights for the different estimators. Therefore we built our own blender. Stacking decreased the score. So we decided to not go with it.
File: `/jupyter/Blender2.ipynb`.

We got our best score by blending NBSvm, LogisticRegression and RandomForest. The kaggle public leaderboard score was 0.98777.

VIII. Results and Analysis

The table below shows the models that we created along with their scores. The evaluation metric used here is the AUC-ROC score. Single Model scores are based on cross validation. Blended models score are based on Kaggle public leaderboard.

Model	AUC-ROC
Simple Logistic Regression	0.70522
XGBoost	0.80147
Random Forest	0.97358
Logistic Regression (Words)	0.98178
Logistic Regression (Chars)	0.98422
Logistic Regression (Chars and Words)	0.98512
Blend Logistic Regression and Random Forest	0.98589
Naive Bayes SVM	0.98669
Blend Logistic Regression, Naive Bayes SVM and Random Forest	0.98777

Confusion Matrix and Area under the curve graphs are shown in appendix.

Pickle file can be found here: Google Drive

Acknowledgment

The authors would like to thank the professors for their continued guidance, support and encouragement. Further, the authors would like to thank the teaching assistants for their valuable time and encouragement. A special thanks to Team Dark Knights (Tejas Kotha, Tanmay Jain) for having intellectual discussions regarding various approaches in preprocessing, feature selection, model selections, and more.

References

[1] "Scikit-learn documentation." [Online]. Available: http://scikit-learn.org/stable/documentation.html

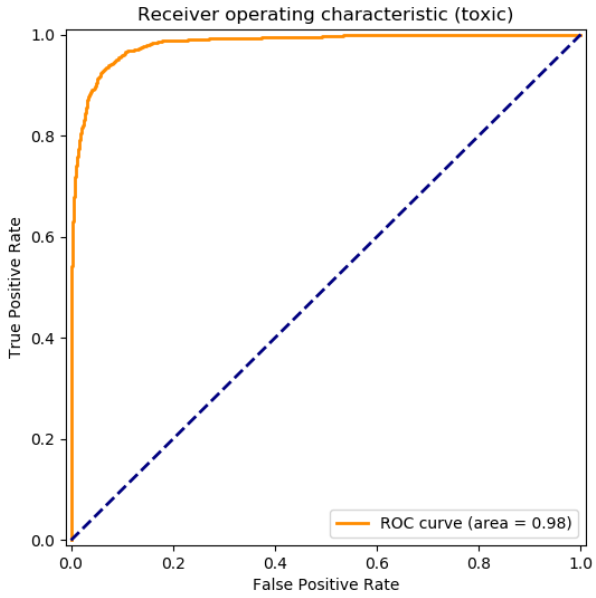
[2] "Pandas documentation." [Online]. Available: https://pandas.pydata.org/pandas-docs/stable/

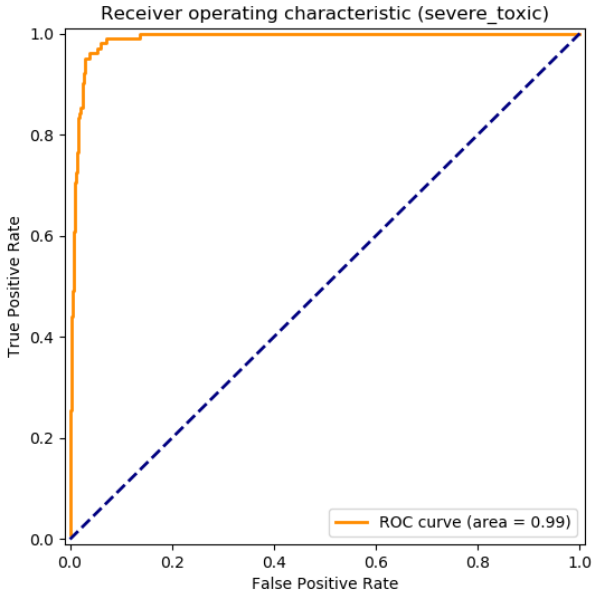
[3] "Numpy documentation." [Online]. Available: https://docs.scipy.org/doc/

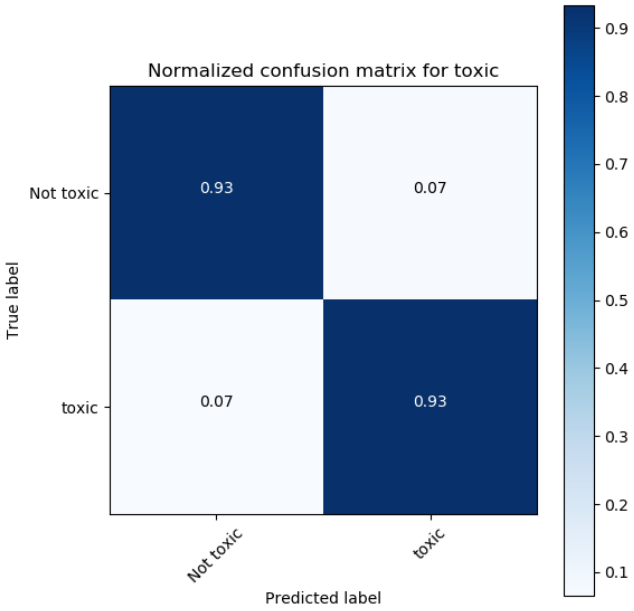
[4] "Matplotlib documentation." [Online]. Available: https://matplotlib.org/contents.html

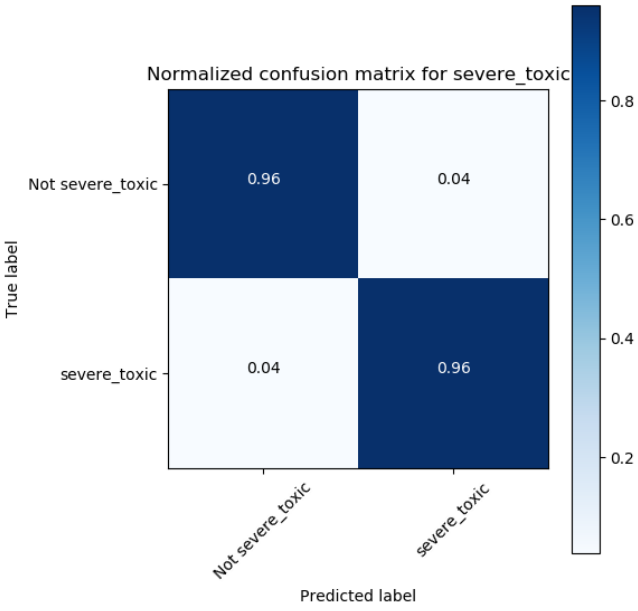
[5] S. Wang and C. D. Manning, "Baselines and bigrams: Simple, good sentiment and topic classification," in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2*, ser. ACL '12. Stroudsburg, PA, USA: Association for Computational Linguistics, 2012, pp. 90–94. [Online]. Available: http://dl.acm.org/citation.cfm?id=2390665.2390688

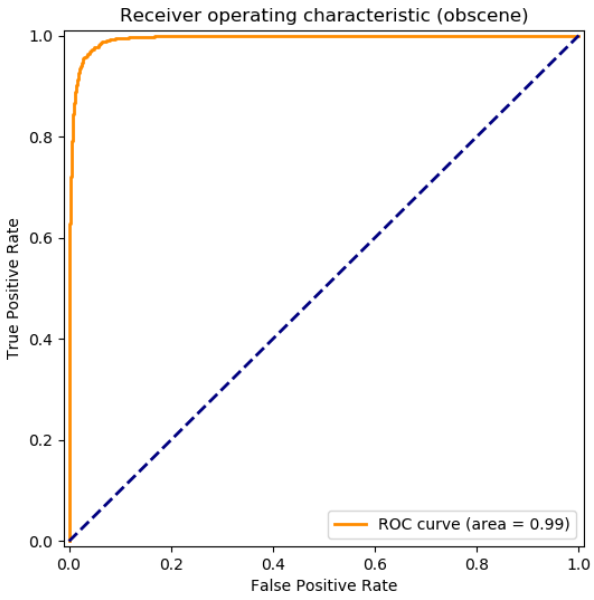
IX. Appendix

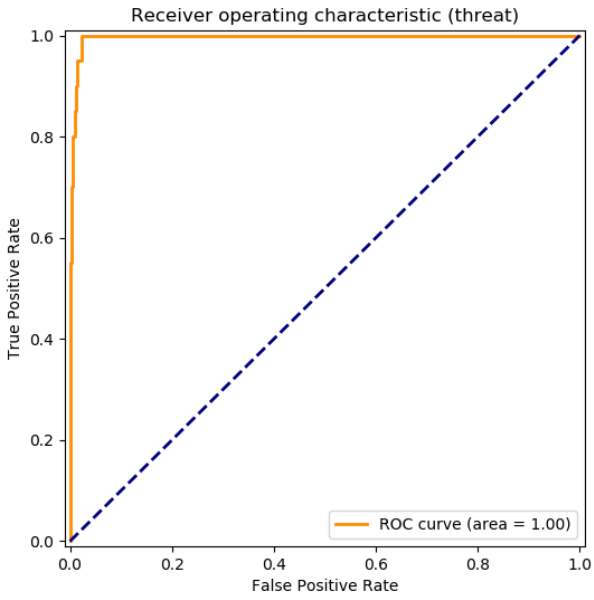

Receiver operating characteristic (toxic)
True Positive Rate
False Positive Rate
ROC curve (area = 0.98)
0.0
0.2
0.4
0.6
0.8
1.0

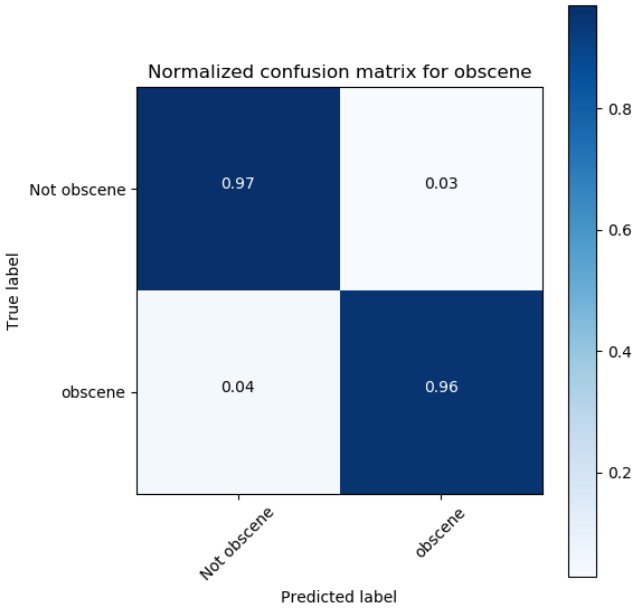

Receiver operating characteristic (severe_toxic)
True Positive Rate
False Positive Rate
ROC curve (area = 0.99)
0.0
0.2
0.4
0.6
0.8
1.0

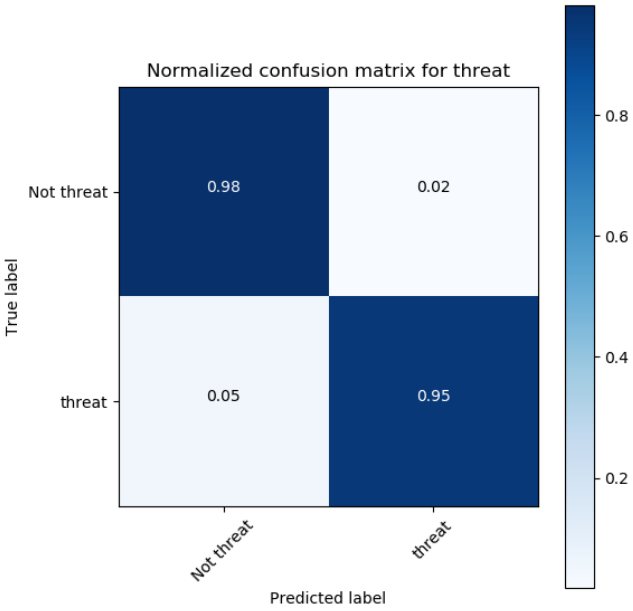

Normalized confusion matrix for toxic
Not toxic
toxic
True label
0.93
0.07
0.07
0.93
Not toxic
toxic
Predicted label
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

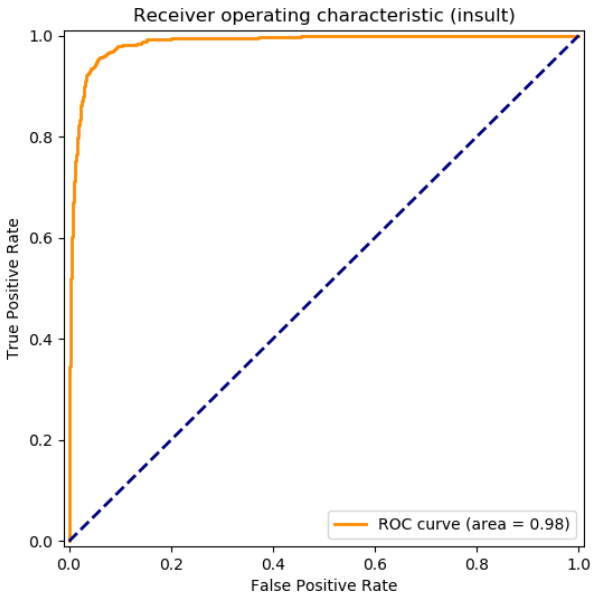

Normalized confusion matrix for severe_toxic
Not severe_toxic
severe_toxic
True label
0.96
0.04
0.04
0.96
Not severe_toxic
severe_toxic
Predicted label
0.8
0.6
0.4
0.2

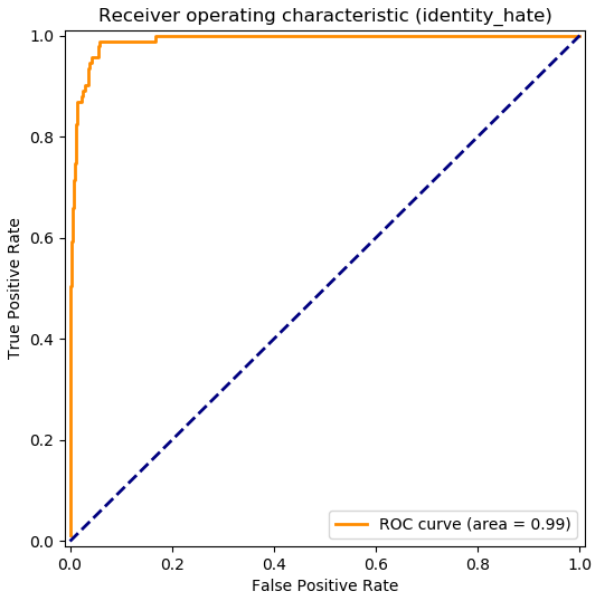

Receiver operating characteristic (obscene)
True Positive Rate
False Positive Rate
ROC curve (area = 0.99)
0.0
0.2
0.4
0.6
0.8
1.0

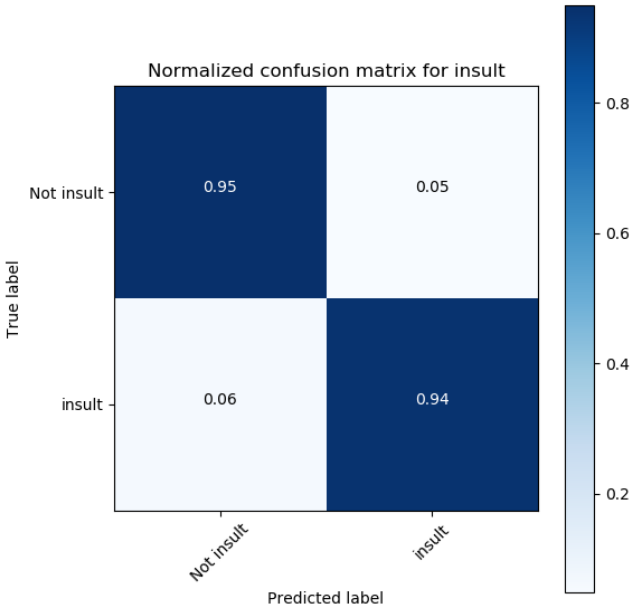

Receiver operating characteristic (threat)
True Positive Rate
False Positive Rate
ROC curve (area = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0

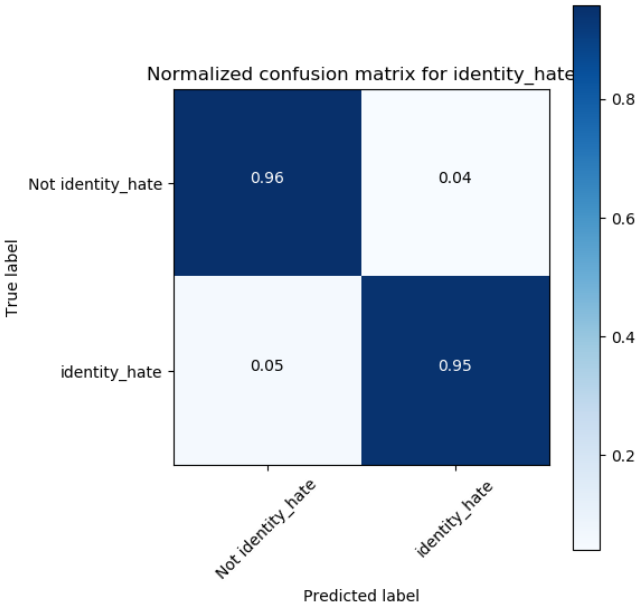

Normalized confusion matrix for obscene
True label
Predicted label
Not obscene
obscene
0.97
0.03
0.04
0.96
0.2
0.4
0.6
0.8


Normalized confusion matrix for threat
True label
Predicted label
Not threat
threat
0.98
0.02
0.05
0.95
0.2
0.4
0.6
0.8


Receiver operating characteristic (insult)
True Positive Rate
False Positive Rate
ROC curve (area = 0.98)
0.0
0.2
0.4
0.6
0.8
1.0


Receiver operating characteristic (identity_hate)
True Positive Rate
False Positive Rate
ROC curve (area = 0.99)
0.0
0.2
0.4
0.6
0.8
1.0


Normalized confusion matrix for insult
True label
Predicted label
Not insult
insult
0.95
0.05
0.06
0.94
0.2
0.4
0.6
0.8


Normalized confusion matrix for identity_hate
True label
Predicted label
Not identity_hate
identity_hate
0.96
0.04
0.05
0.95
0.2
0.4
0.6
0.8