

**Final Report: Toxic Comment Classification**

**Submitted to:**

Prof Qun Liu

School of Computing

Dublin City University

**Submitted By:**

Lavleen Bhat

**17210637**

lavleen.bhat2@mail.dcu.ie

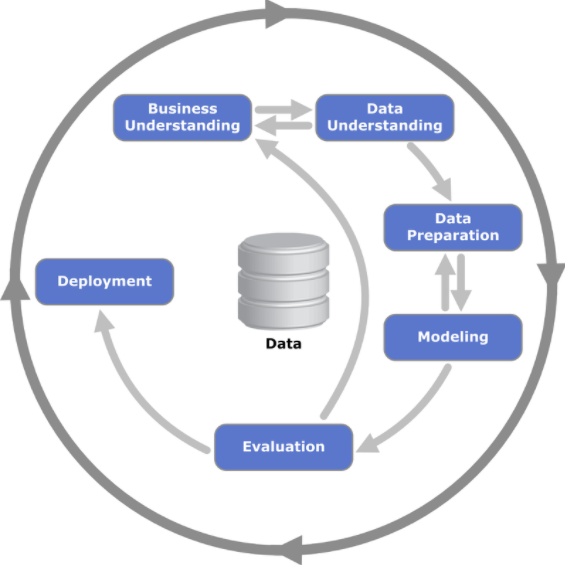
Subject: Machine Learning, CA684

**April 30, 2018**

**Declaration:**

In submitting this project, I declare that the project material, which I now submit, is my own work. I make this declaration in the knowledge that a breach of the rules pertaining to project submission may carry serious consequences.

**Introduction:**

The project aims to classify the comments into different labels of toxicity – toxic, severe toxic, obscene, threat, insult and identity hate. The project was divided into 6 different stages using CRISP-DM model.

* Business Understanding
* Data Understanding
* Data Preparation
* Modeling
* Evaluation
* Conclusion

**Technologies used:** Python, Keras, Scikit-learn

1. **Business Understanding**

**1.1 Objective:** The main object of the project is to distinguish and classify the comments into different categories of toxicity. Comment can be classified into more than one labels.

**1.2 Motivation**: With the increasing use of social media, there has been a huge wave of revolution within people to speak their own mind. One side-effect of which is, people can go anonymous and write whatever they like. Most of the times, people write negative comments which spread hatred and can be obscene. Current methodologies to handle such comments is largely manual. The users report the negative comments and then the moderator take the relevant action manually. In most of the cases, websites had to disable their comment section. During the process of flagging the comment and taking appropriate action, the damage has already been done. To avoid this time-consuming moderation process, there should be methods/tools by which we can detect such comments and prevent people from posting such comments altogether. The project aims to classify such social media comments in multiple categories.

1. **Data Understanding**

**2.1 Dataset**: The dataset is available on Kaggle. The dataset has one file for training dataset and one file for test dataset. The dataset has comments from Wikipedia’s talk page edits.

The comments in training dataset are classified in the below categories. One comment can be present in more than one category/label.

**Labels(Targets):** Toxic, Severe Toxic, Obscene, Threat, Insult, Identity Hate

**Columns:** ID, Comment

Number of records in training data: 159571

Number of records in test data: 153164

Test data is present in a separate file which is unlabelled. So, I’m predicting the probability of comments for each label. For validation, I have split the training dataset into 20% test dataset and 80% training dataset.

1. **Data Preparation**
   1. **Data Cleaning:** There are no null values present in the dataset and so didn’t have to spend time on imputation of null values. To clean the text comments, the punctuation has been removed from the comment before processing the text.
   2. **Tokenize and Vectorize:** I have used two different approaches for pre-processing the data before passing to Machine Learning models – Tokenizer and Vectorization.
   3. **Train test split:** Because the test data provided is unlabelled, to validate and check the accuracy of our model, I have split the training data keeping 20% for validation and testing.
   4. **Up-sampling:** As the dataset is imbalanced and has only 10% of toxic comments among whole dataset, the number of samples in the toxic class has been increased to 47.5 % by artificially picking up only the toxic comment in current dataset and appending to the training dataset.
2. **Modelling:**

Initially, the objective of the project was to only classify the comments but as I progressed through the project, I decided to do a comparison of different approaches to handle the same data. The approaches can be divided as below:

Model 1: Initial Classification (Baseline)

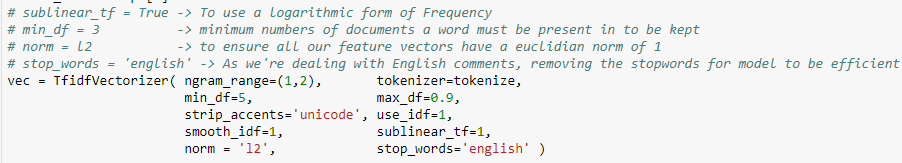
Model 2: Classification after up-sampling

Model 3: Classification using LSTM after up-sampling

**Model 1: Initial Classification**

For initial classification of the comments, I used Logistic Regression and Linear SVC models provided by scikit-learn python library.

1. **Term Frequency-Inverse Document Frequency (TF-IDF):** The program cannot understand the raw text directly and hence, we need to provide the text data in an understandable form to the models. For this, TF-IDF vectorizer has been used to convert the comments into matrix of vectors.



Here, I have merged the tokenization step with TF-IDF vectorizer. To tokenize the text, I’m removing the text punctuation for text cleaning.

As the text data is English only, I have used STOP\_WORDS = ‘ENGLISH’ for removing unnecessary words.

1. **Logistic Regression:** The first model implemented is Logistic Regression to build a baseline model for this project. Logistic Regression is the simplest classification algorithm to classify the text into multiple labels.

After converting the text into vector format, these feature vectors are passed to the Logistic Regression model. I have trained the model twice, once using unigrams and second time using bi-grams. The results were significantly different.

1. **Linear Support Vector Classification (Linear SVC):** Logistic Regression model gave a good accuracy of prediction which made me suspect and try a different machine learning model for the task of classification. I applied Linear SVC for getting an idea of the accuracy and for a comparison against previous model.

Same feature vectors which were used for previous model has been used for Linear SVC model as well. The model was trained twice – with unigrams and with bi-grams.

**Model 2: Classification after up-sampling**

As the model accuracy from above two models was suspiciously good without signification difference, I decided to take a look back at the dataset and found that the percentage of total toxic comments (including all categories) present in the training dataset was only 10% of the whole dataset which made me curious to find out and check for the effect of increasing the sampling of toxic comments in the training dataset. I tried this step to mainly rule out the possibility of overfitting or if there was in-sufficient data for the model to train on toxic comments.

1. **Up-sampling:** So, I increased the number of toxic comments (process explained in Data Processing section) in the training dataset to 47.5% of the total comments. So, the dataset is almost balanced to start working on.
2. **Logistic Regression:** After balancing the dataset, I applied same models to the modified dataset.

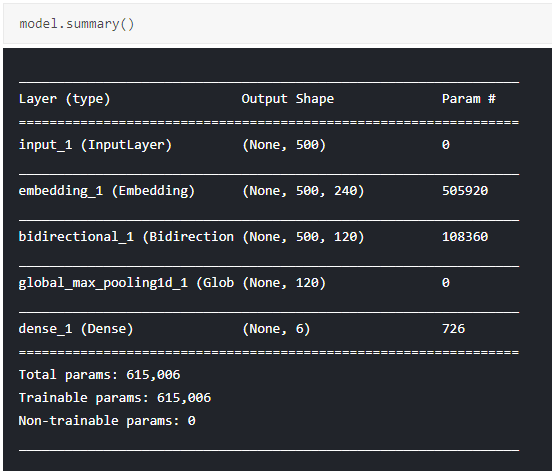
As the results for both, Logistic Regression and Linear SVC in previous modelling step were similar, I did not apply Linear SVC model for this step.

**Model 3: Classification using LSTM after up-sampling**

As the models were giving realistic results after balancing the dataset in previous step, the accuracy was still low. So, I decided to try the Recurrent Neural Network and check if the prediction accuracy can be improved using RNN. For this task, I used LSTM model provided by Keras python library. Below are the steps involved for applying LSTM to this dataset.

1. **Tokenize:** As the model cannot process raw text, the text comments have been tokenized first using Tokenizer function provided by Keras library. After tokenization and converting the training dataset into tokens, next step was to pad the generated sequences.
2. **Pad Sequences:** The comments are varying in length and largest percentage of comments have length between 0 and 500 characters. So, the sequences are truncated to the maximum length of 500 characters to make the model run efficiently.
3. **Model:** LSTM model has 5 different layers for completing the task.
   1. **Input Layer:** This step is to initialize the model with dimension of our input data.
   2. **Embedding Layer:** I have used Embedding layer provided by Keras instead of using pre-trained embeddings. This embedding layer has input dimension as length of the total comments and output dimension as 240. Output dimension defines the size of the output vector space in which each word will be embedded.
   3. **Bidirectional LSTM layer:** I have used bi-directional LSTM layer using GRU provided by Keras library. The reason for using GRU is that it controls the flow of information just like LSTM without using memory unit and are computationally more efficient.
   4. **Max-pooling Layer:** Here, global max pooling layer has been added for pooling operation to get a 2D matrix output.
   5. **Dense Layer:** This densely connected NN layer is activated with a ‘SIGMOID’ function for activation. Sigmoid function is better for classification task but for future work, we can try other activation functions such as ‘RELU’.
   6. **Model Layer:** Finally, we fit the model to our training dataset with batch size 100 for faster training and epochs = 2 for iterations. Validation data is used from the test data obtained after train\_test\_split function.

Here is the model summary showing all the layers.



1. **Evaluation**

For evaluation of the first two models, I have used cross validation score for checking the accuracy of classification for each class. For third model, I have used ‘BINARY\_CROSSENTROPY’ loss and Keras accuracy for checking the accuracy of LSTM model.

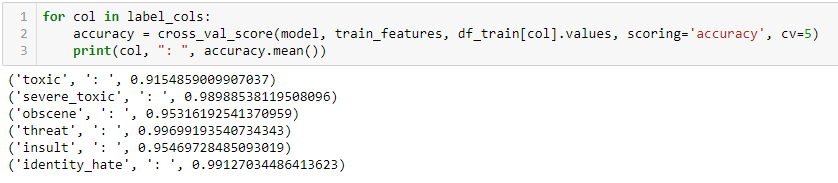
Here are the results:

**Model 1: Initial Classification**

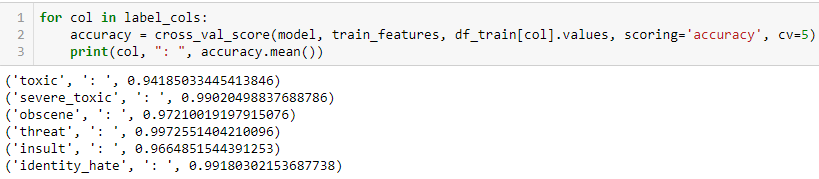
In the initial classification, I have taken the mean of the accuracy of prediction of each class.

1. **Logistic Regression:** Below are the accuracy metrics for logistic regression.

Uni-gram:

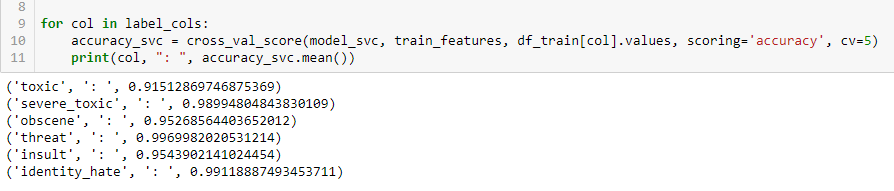


Bi-gram:

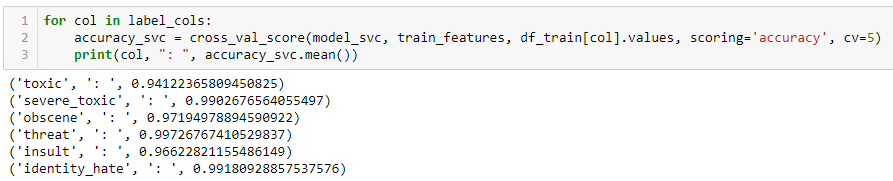


1. **Linear SVC:** Below are the accuracy metrics for Linear SVC model.

Uni-gram:



Bi-gram:

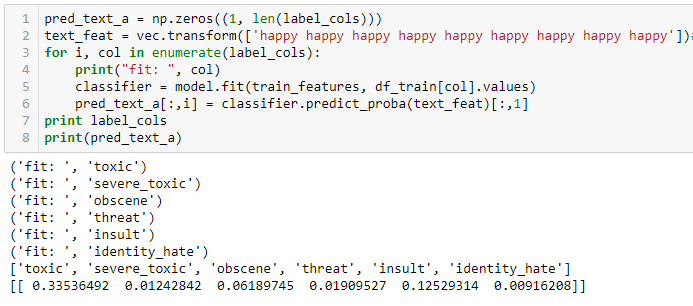
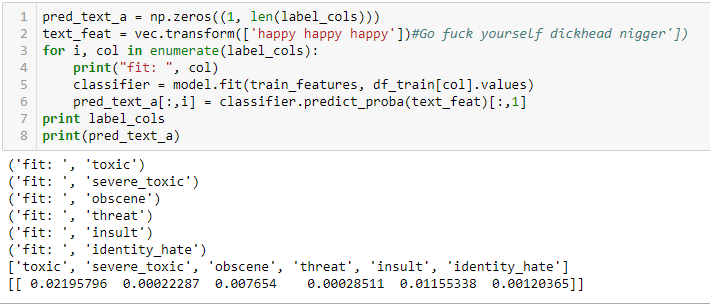


As it’s clearly visible, there is not significant difference between the prediction accuracy of both the models. This compelled me to check for the overfitting of model due to insufficient data for both the classes (toxic, non-toxic). This lead me to next model.

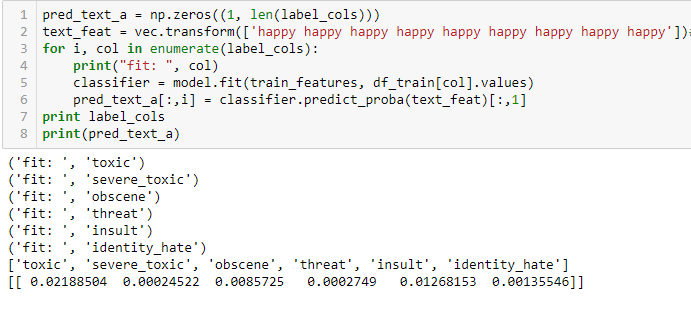
Another notable observation made after first model implementation was that when you try to implement the model on a random text, the results are not very good when predicting the toxicity of non-toxic comments. The reason for this is the use of unigrams and bi-grams. Bi-grams give better results.

If you keep increasing the number of same words in the sentence as shown below, the toxic probability will also increase. So, apparently, too much of happiness is a little toxic.

Unigram results:

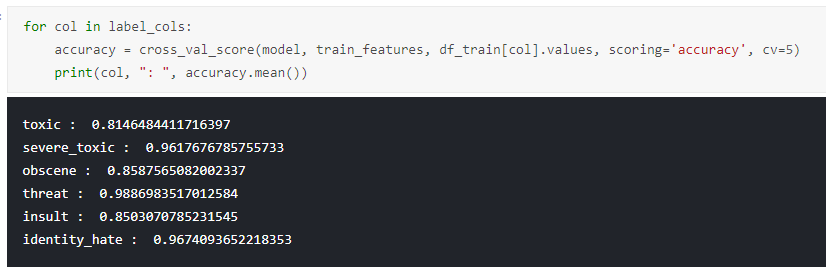


Bigram results:

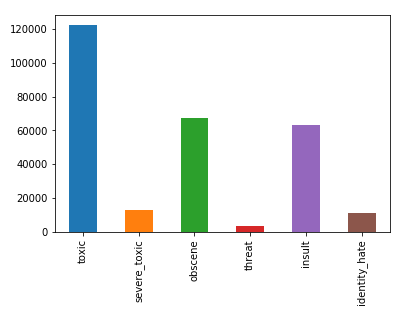


**Model 2: Classification after up-sampling**

After increasing the data for our toxic classes and balancing the dataset to have toxic comments around 47.5%, here are the results for the models applied to this modified dataset.



As the metrics show that the prediction accuracy has decreased drastically. Another thing to notice here is that, along with imbalance in the classes – toxic and non-toxic, there is imbalance in the data present for these labels (toxic, severe toxic, obscene, threat, insult, identity hate) as well. Here is the graph showing that.

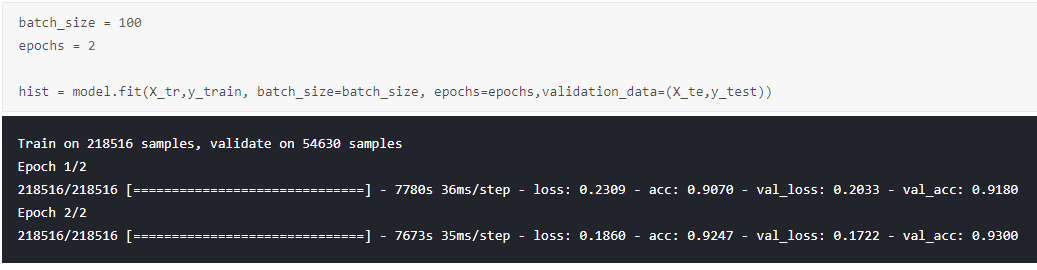


The graph above clearly explains that with the increase in amount of data available for training, the lower the accuracy. This brings to the conclusion that Logistic Regression and Linear SVC are not the best fit models for the task of comments classification.

**Model 3: Classification using LSTM after up-sampling**

After concluding that Logistic Regression and Linear SVC are not good models for this task, I tried Recurrent Neural Network. RNN models such as LSTM are said to be one of the good models for text classification task. So, I tried this model and here are the results.

I chose epochs to be 2 because of the limited CPU and memory capacity available. I’m certain if we increase the number of iterations, the accuracy will also increase. Additionally, for faster calculation, I chose batch size to be 100. If we reduce the batch size to optimum level, the accuracy will also increase.



The above image shows the accuracy for LSTM model. I’m certain that if the number of iterations is increased, prediction accuracy will also improve.

**Conclusion:**

So, after trying 3 different techniques for the classification of toxic comments, it can be concluded that LSTM performs way better than Logistic Regression and Linear SVC models. It’s also significant to mention that use of bigrams for text classification give better results than unigrams. Also, as we have observed, the samples of data present also impact the prediction accuracy. If we have an imbalanced dataset, the results will be biased and cause overfitting. Increasing the sampling and artificially balancing the dataset makes a huge impact on the results of model predictions.

**References:**

1. Bouazizi et. al. - Sentiment Analysis: from Binary to Multi-Class Classification
2. Zhang et. al. - Deep Learning for Sentiment Analysis: A Survey
3. Chu et. al. - Comment Abuse Classification with Deep Learning
4. Sax et. al. - FlameWars: Automatic Insult Detection
5. Hosseini et. al. - Deceiving Google’s Perspective API Built for Detecting Toxic Comments
6. Georgakopoulos et. al. - Convolutional Neural Networks for Toxic Comment Classification
7. <https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge>
8. <https://towardsdatascience.com/multi-class-text-classification-with-scikit-learn-12f1e60e0a9f>
9. <http://deeplearning.net/tutorial/lstm.html>
10. <https://www.geeksforgeeks.org/twitter-sentiment-analysis-using-python/>
11. <https://streamhacker.com/2010/05/10/text-classification-sentiment-analysis-naive-bayes-classifier/>
12. <https://machinelearningmastery.com/metrics-evaluate-machine-learning-algorithms-python/>

**GitHub Repository:**

<https://github.com/lavleenbhat/CA684-Machine-Learning.git>