Toxic Comment Classification Project

Jose Luis Martin A20410665, Student, IIT, Martin Berger A20410866 , Student, IIT, Mridula Tunga A20408428, Student, IIT , and Antoine Gargot A20410860, Student, IIT

[1]IIT, Illinois Institute of Technology, 3300 S Federal St, Chicago, IL 60616

Abstract – Expressing oneself on public forums these days is a challenging task because of the threat of abuses and harassments. Platforms struggle to effectively facilitate conversations, leading many communities to limit or completely shut down user comments. The Conversation AI team, a research initiative founded by Jigsaw and Google (both a part of Alphabet) are working on tools to help improve online conversation. One area of focus is the study of negative online behaviors, like toxic comments (i.e. comments that are rude, disrespectful or otherwise likely to make someone leave a discussion). So far they've built a range of publicly available models served through the Perspective API, including toxicity. But the current models still make errors, and they don't allow users to select which types of toxicity they're interested in finding (e.g. some platforms may be fine with profanity, but not with other types of toxic content). We will be trying to build a multi-headed model that's capable of detecting different types of toxicity like threats, obscenity, insults, and identity-based hate better than Perspective's current models. The dataset that we have to study will be comments from Wikipedia's talk page.

I. OVERVIEW

A - Relevant literature

As this project is currently on Kaggle as a competition, we can study different kernel and model from different competitor who wanted to share it. Some models with logistic regression seems to give good final result and most of the people emphasis on the fact that preprocessing the data and take care of stop words is really important. [1]

B - Proposed methodology

First we cleaned and preprocessed our data in order to extract new features and generate tokens, remove missing values and stop words. Then, we studied our data in order to see the distribution of different label regarding occurrences of reviews linked to it. We also studied link between label and review (some reviews will share different labels).

After cleaning the data and studying it, we implemented different models. Some that we saw in class such as logistic regression, Naive Bayes and few other new models like LSTM which is part of Neural Networks (Tensor Flow);

In order to analyze the validity of our results, we predicted the probability of each label for every comments, working as in a binomial problem for each label. We ended up with 6 confusion matrices from which we will draw some ROC curves and work on classification rate. Based on the 6 ROC curves, we computed the AUC value for each of them. We used cross validation to make sure that we are not over fitting our model. For general values, we studied the prediction accuracy of each model with a specific label, followed by F1 measure to study the right classification for toxic comments.

II. DATA OVERVIEW

Our training dataset consist of 159571 wikipedia comments that have been classified between 6 labels according to how offensive they would be. The 6 labels that characterize the dataset are the following:

Toxic, Severe Toxic, Obscene, Threat, Insult, Identity Hate. Each text and comment can have several label assignation when they are categorized as toxic. For instance, some can be seen as toxic, obscene and insult in the same time.During the analysis, comments that do not qualify for any of these labels will be referred as "Clean" or "None".

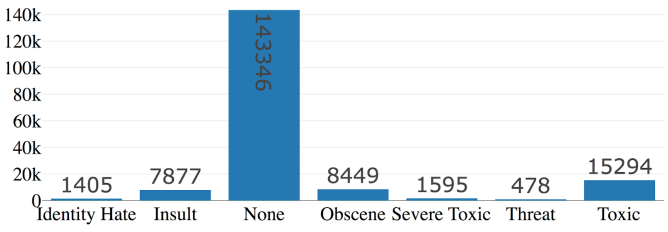

FIG 1 : OBSERVATION REPARTITION OVER LABELS

As we can see from the figure above, our dataset has a very unbalanced label repartition. This may be problematic in order to train an accurate model. We will have to take it into account.

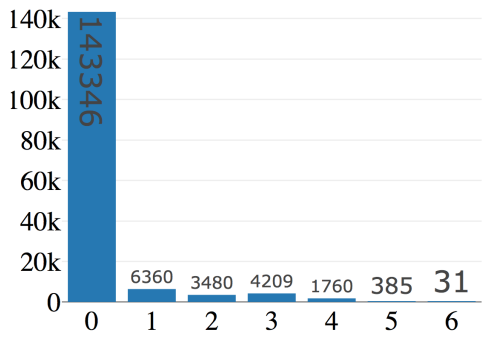

FIG 2 : NUMBER OF OBSERVATIONS WITH MULTIPLE LABELS

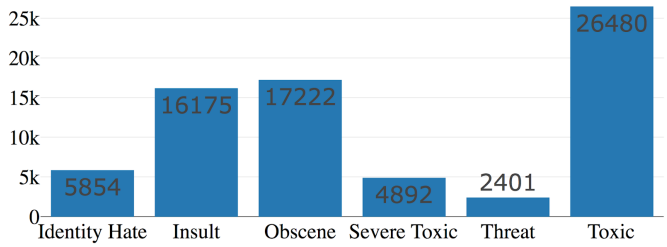

FIG 3 : SIZE OF THE VOCABULARY FOR LABELS

Regarding the size of the vocabulary, we can see that it is highly correlated with the number of occurrence of each label.

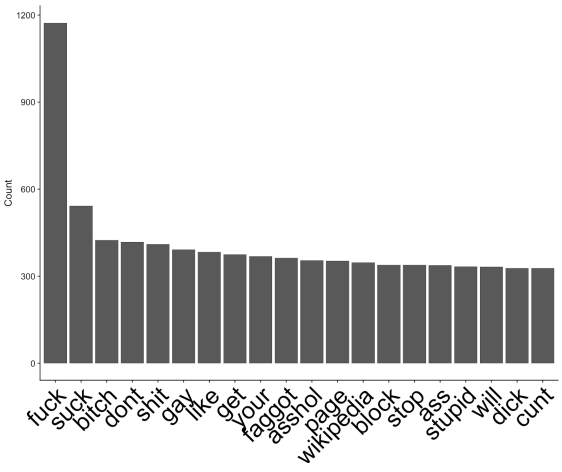

FIG 4 : TOXIC COMMENT TOP 20 WORDS

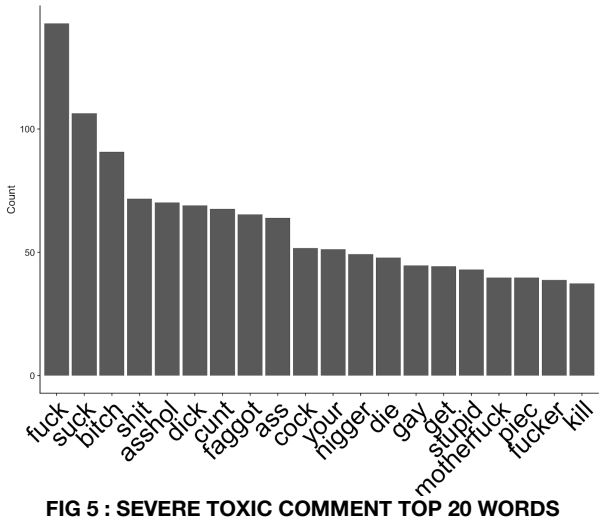

FIG 5 : SEVERE TOXIC COMMENT TOP 20 WORDS

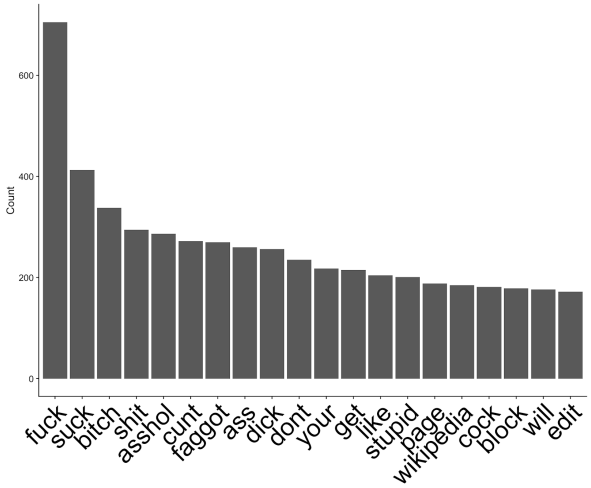

FIG 6 : OBSCENE COMMENT TOP 20 WORDS

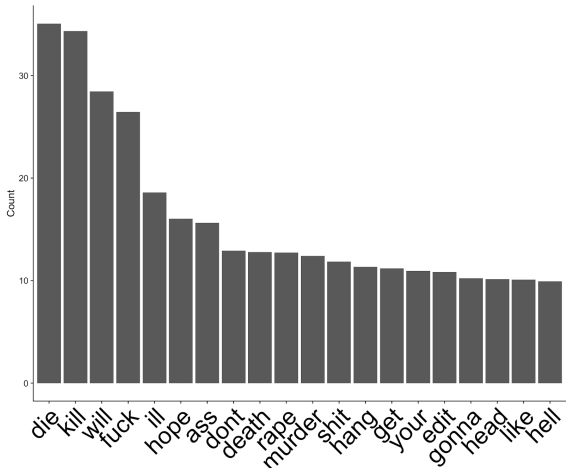

FIG 7 : THREAT COMMENT TOP 20 WORDS

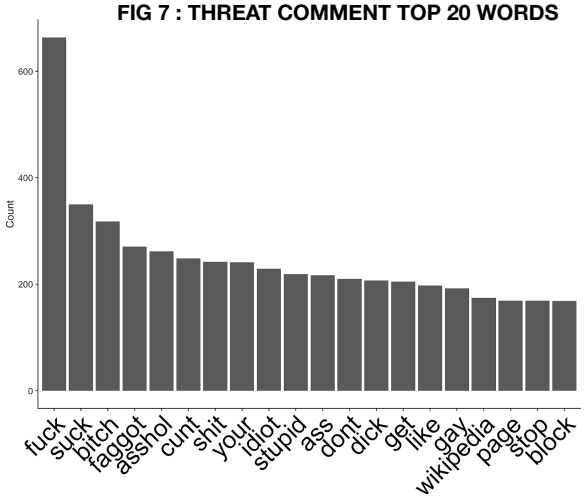

FIG 8 : INSULT COMMENT TOP 20 WORDS

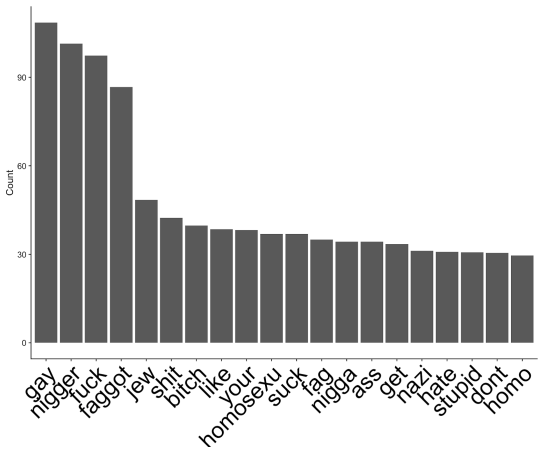

FIG 9 : HATE COMMENT TOP 20 WORDS

As we can see above, some words often come up in different Top20 Words ranking such as “fuck”, “suck”, “bitch”. These words will often be a grate marker to figure out a comment is toxic or not. However, we will need to focus on other words that make it up to the top 20 to be able to find the appropriate type of toxicity. For example, we can notice that for the threat comments, the words “die”, “kill” and “murder” find themselves in a better position than in any order labels.

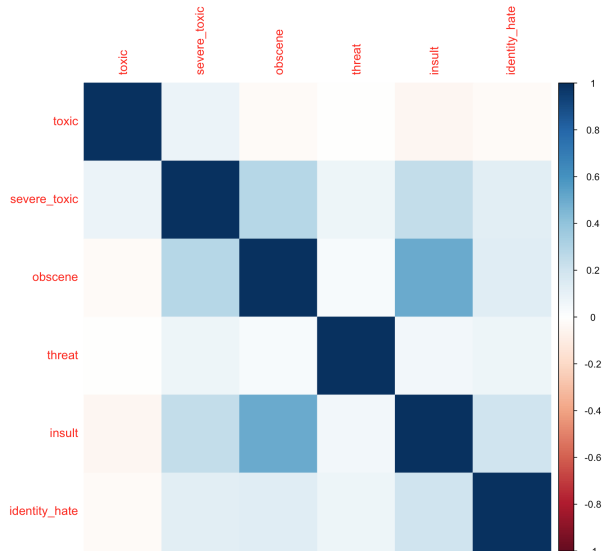

FIG 10 : CORRELATION MATRIX BETWEEN LABELS

Looking at the correlation matrix, we can notice that severe toxic comment tend to be correlated to insults and obscenity.

III. DATA PROCESSING

In order to preprocess our data, we used several pipelines for distinct treatment steps of it. Before processing and cleaning the text data from each comment, we wanted to extract relevant feature for this problem :

- The length of the comment
- The number of capital letter
- The ratio of capital letter among the whole text
- The number of exclamation marks
- The number of punctuation marks
- The number of words
- The number of symbols inside the text.

After extracting those feature from the data, we were able to clean it. First, we worked on comment tokenization using different preprocess step. We wanted to put all comments in lowercase, removing special character from them, erase the bad spaces in the text, remove unique character inside the text and finally create token from all the corpus with stemming each word (remove the expansion of word in order to merge conjugations, grammar rules modification into a unique token).

After working on the preprocess of our data, we wanted to create a document-term matrix. In order to do so, we created a vocabulary from all our corpus of 4000 different words. Words were selected if they appeared in at least 3 times,if they appear less than in 50% of the documents (remove recurrent words) and if those were not stop-words. With this final vocabulary, we created a vector which was the different feature of our document-term matrix. Thanks to this vocab, we created a document-term matrix for the entire corpus.

We added some weight to our document term matrix based on the Tf-Idf formula in order to give less weight to recurrent words and more weight to relevant words inside the corpus.

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

IDF(t) = log_e(Total number of documents / Number of documents with term t in it)

Finally, we merge the document term matrix with additional features that we computed before preprocessing the data.

IV. NAIVE BAYES

We first trained our Naive Bayes model on the entire dataset but we ended up with terrible results. Those were probably due to the fact that the dataset was very unbalanced as we saw in the data overview. Therefore, in order to solve this problem we took all the observations that had at least one toxic label and we took as many without any toxic label. This gave us a dataset of 32350 observations. With this dataset, we did apply a 10-Fold in order to evaluate our results with a minimum amount of bias. Furthermore, we did variating the smoothing parameter alpha from 0 to 1. In the graph below, you will find the resume of our results.

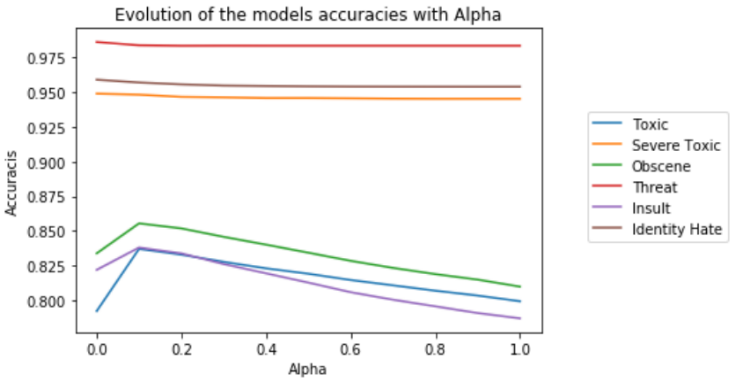

FIG 11 : EVOLUTION OF THE MODEL ACCURACY WITH ALPHA

We can see in the plot above, that the Obscene, Toxic and Insult labels are the most affected by the variation of alpha when it comes to accuracy. However, as we saw in the data overview, these labels tend to be correlated which explain the same behavior.

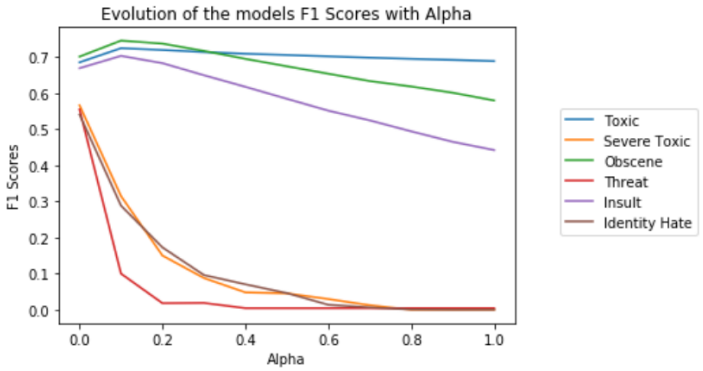

FIG 12 : EVOLUTION OF THE MODEL F1 SCORE WITH ALPHA

In the above we can see how having a high value of alpha will negatively impact our model. Therefore, we should focusing on keeping a low smoothing coefficient.

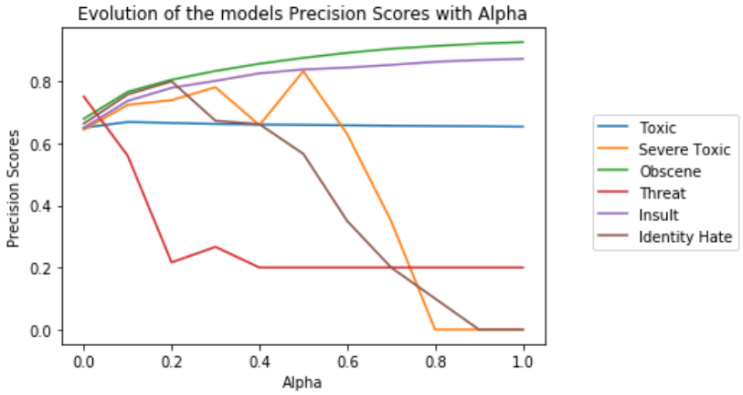

FIG 13 : EVOLUTION OF THE MODEL PRECISION WITH ALPHA

The precision of each model variates a lot depending on the value of alpha. In this case, we should choose an alpha value that enable the best precision to most of the model. Here again, a low alpha value seems to be the best solution.

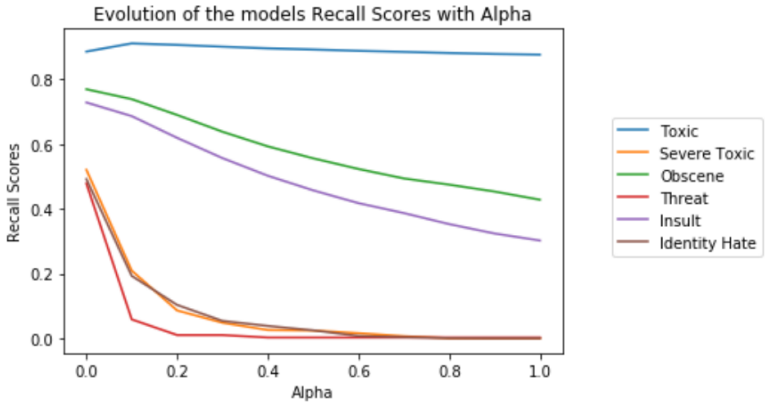

FIG 14 : EVOLUTION OF THE MODEL RECALL WITH ALPHA

Just as with the previous plot, we can notice that a low alpha value would offer us a best option. We will see in detail the comparison the metrics for an alpha equal to 0 and equal to 0.1

	Measures/Label	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	Accuracy	0.792179	0.949026	0.833818	0.986275	0.821917	0.959073
1	F1_Score	0.685120	0.566078	0.700894	0.554553	0.668888	0.540678
2	Precision	0.650796	0.644401	0.679737	0.751325	0.650648	0.664262
3	Recall	0.885518	0.520205	0.769189	0.478104	0.728196	0.491646

FIG 15 : MODEL MEASURES FOR ALPHA = 0

	Measures/Label	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	Accuracy	0.837032	0.948284	0.855425	0.983895	0.837991	0.957063
1	F1_Score	0.724267	0.314748	0.745210	0.100038	0.702978	0.288180
2	Precision	0.669516	0.724531	0.766765	0.560714	0.737548	0.758176
3	Recall	0.910745	0.208744	0.738534	0.058204	0.686116	0.192532

FIG 16 : MODEL MEASURES FOR ALPHA = 0.1

We can assume that we have better results with a low value of alpha because the smoothing coefficient is used to avoid giving a probability of zero when a unknown word appears in the dataset. Because most of the offensive words appears in the training set, there is no need to give to much importance for other new words that would add noise.

Comparing the two values of alpha, we can notice that a smoothing value equals to zero offers better results. However, we can notice that the three labels that have a low recall score are : severe_toxic, threat and identity_hate. This is due to the fact that we have too few observations about this label(respectively 1495,478 and 1405). We would need more data to perform a better model.

V. LOGISTIC REGRESSION

Appropriate regression analysis to conduct when the dependent variable is binary. However, we tried our hands on implementing logistic regression on multi label data. We split the train set into 80-20 train test split and carried out the process. Preprocessing step for the data included removing numbers, removing words whose length is less than 2.

A vocabulary was created with token vectorization with which a Document Term Matrix was created. Term Frequency Inverse Document Frequency was calculated to find how important the words are for various levels of toxicity. Now, with the final obtained clean dataset, logistic regression model was fit.

A cross validation approach was followed to fit the model with logistic regression which was carried out in a loop for each label.

The predicted model looked like this:

	id	toxic	severe_toxic	obscene	threat	insult	identity_hate
1	0000997932d777bf	0.0189705568	0.0019188034	0.0047913665	0.0007309999	0.0037624869	0.0019278840
5	0001d958c54c6e35	0.0286691487	0.0009923556	0.0323233664	0.0004283634	0.0255175482	0.0040011252
7	0002bcb3da6cb337	0.9965078771	0.0758724728	0.9635744116	0.0022519545	0.5945125170	0.0099592776
11	0005300084f90edc	0.0002839812	0.0004026656	0.0009164614	0.0001453301	0.0003700149	0.0002089342
15	00070ef96486d6f9	0.0755905024	0.0018826387	0.0129517935	0.0008172589	0.0129216651	0.0050152329

FIG 17 : SAMPLE OF PREDICTED PROBABILITIES WITH LOGREG

As seen, the probability of each of the comment Ids for particular label is prominent. After this, we assign a label or not based on a threshold of 50% for our data.

```
prediction[1:5,]
test_label[1:5,]
```

	id	toxic	severe_toxic	obscene	threat	insult	identity_hate
1	0000997932d777bf	0	0	0	0	0	0
5	0001d958c54c6e35	0	0	0	0	0	0
7	0002bcb3da6cb337	1	0	1	0	1	0
11	0005300084f90edc	0	0	0	0	0	0
15	00070ef96486d6f9	0	0	0	0	0	0

	id	toxic	severe_toxic	obscene	threat	insult	identity_hate
1	0000997932d777bf	0	0	0	0	0	0
5	0001d958c54c6e35	0	0	0	0	0	0
7	0002bcb3da6cb337	1	1	1	0	1	0
11	0005300084f90edc	0	0	0	0	0	0
15	00070ef96486d6f9	0	0	0	0	0	0

FIG 18 : COMPARISON BETWEEN ACTUAL AND PREDICTION

Below table depicts the results obtained after obtaining confusion matrix and AUC for each classification :

Label	Accuracy	Precision	Recall	F1	AUC
Toxic	0,956	0,963	0,869	0,913	0,813
Severe	0,99	0,99	0,56	0,665	0,624
Obscene	0,97	0,98	0,88	0,93	0,819
Hate	0,99	0,99	0,67	0,792	0,63
Threat	0,997	0,99	0,51	0,67	0,58
Insult	0,97	0,977	0,791	0,864	0,76

FIG 19 : SCORE SUMMARY FOR EACH LABEL

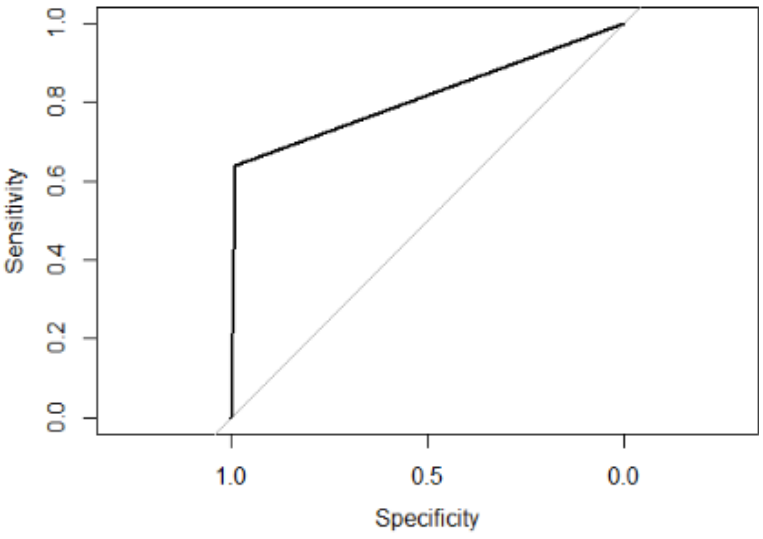

FIG 20 : ROC CURVE OF THE TOXIC PREDICTION

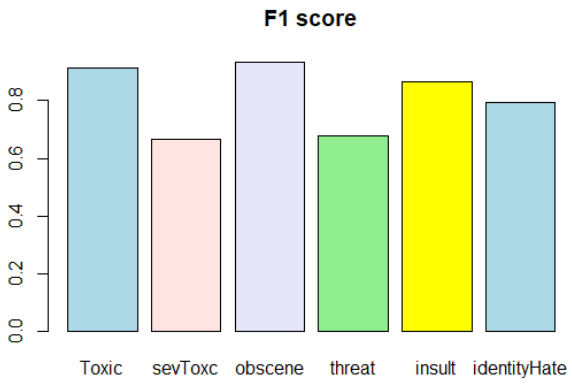

FIG 21 : F1 SCORE FOR EACH LABEL

V. LDA

We tried to implement LDA on multi-labeled data to see if LDA is a suitable model to predict out the labels for a specific comment. We trained the model with an 80% of the training data and used the rest for the test. We preprocessed the data same as logistic regression and performed the models for each label.

For all labels we got the confusion matrix and all had an accuracy of 1, Sensitivity of 1 and a Specificity of 1. Showing that the model for all labels performed with a success of 100%. This results are not logical as its impossible to have a model that is able to distinguish the labels perfectly.

Our next approach was to perform Latent Dritchlet Allocation for all the data without making any selection on the labels and we tried to cluster the output using PCA to see if the model was able to classify in different cluster all the labels. After performing several rounds changing the level off clusters we could conclude that Latent Drictchlet Allocation can actually separate regular comments from toxic comments. Results are shown below:

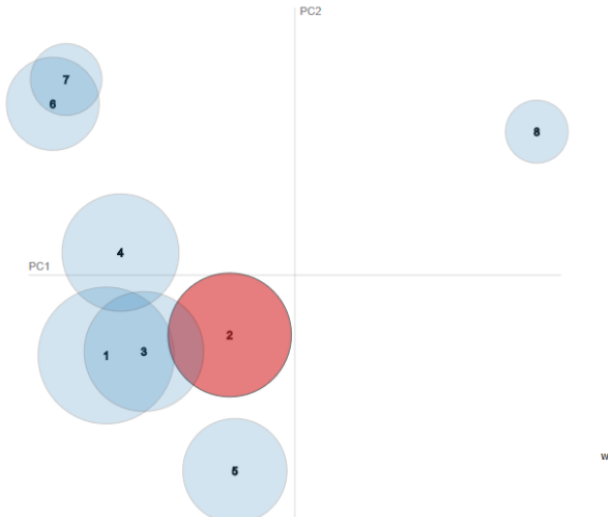

FIG 22 : LDA CLUSTER FOR NON-TOXIC COMMENTS

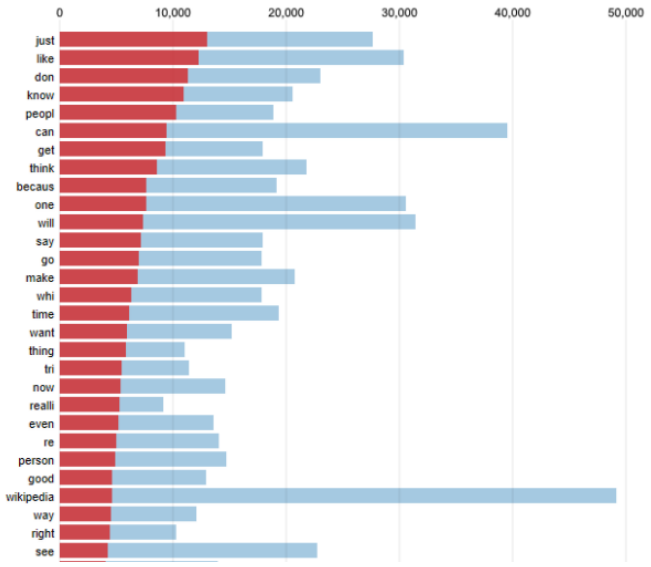

FIG 23 : LDA TOPIC FOR NON-TOXIC COMMENTS

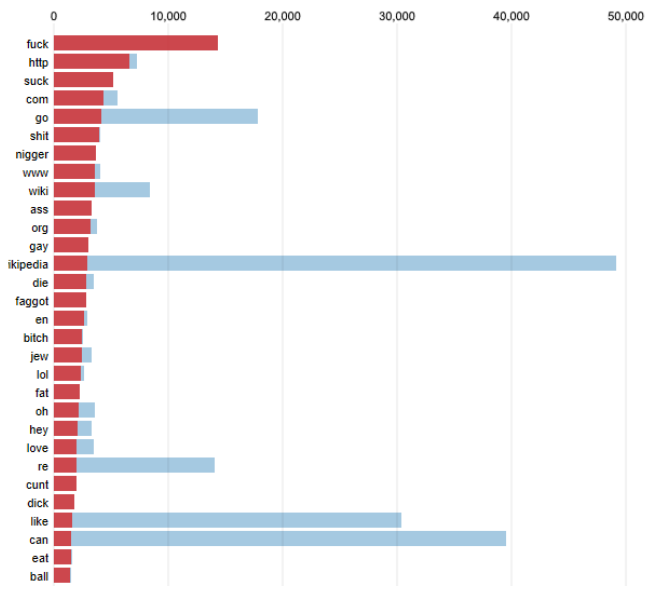

FIG 25 : LDA TOPIC FOR TOXIC COMMENTS

When adding more topics Latent Dritchlet Allocation was capable of differentiating bag of words of the same topic such as religion, culture, political, toxic, regular….etc, but we were not able to distinguish between labels of toxicity. We concluded that Latent Dritchlet Allocation can be used as a first filter to determine if the comment is toxic or not.

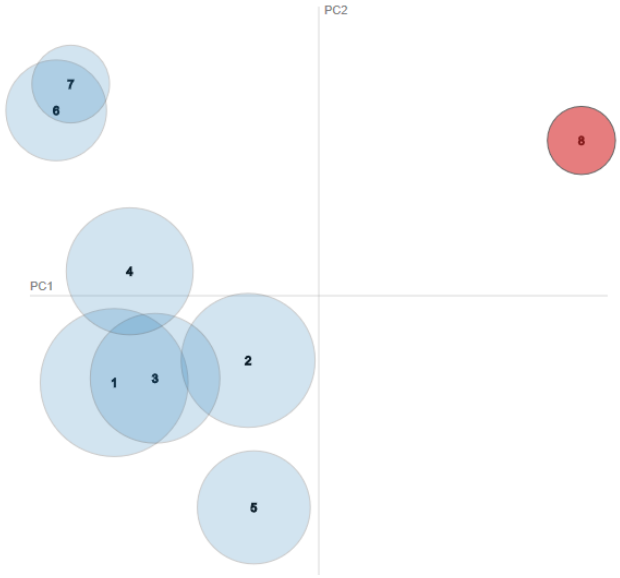

FIG 24 : LDA CLUSTER FOR TOXIC COMMENTS

VI. LSTM

We tried another final approach to work with toxic text classification using new technologies such as recurrent networks in deep learning field. For creating our first recurrent neural network, we worked with the well know tensorFlow library created by Google using Keras as a library to manage this backend. Regarding the fact that a structure of a neural network has a relatively different approach than machine learning algorithm, we work on a different data preprocessing.

For this model, we used 20,000 different features (words in our dictionary). We parsed our tokens inside a sequence which is, for each observation, an array of words id from the dictionary where the word is present in the current text.

LSTM neural layer requires the data to be of fixed length, that is the same number of features, but the comments can have various lengths and hence the indexing length might vary. Hence we go for padding where we set a maxlen allowed to a specific number. In order to find this optimal number, we plotted the total number of words per comment inside a histogram.

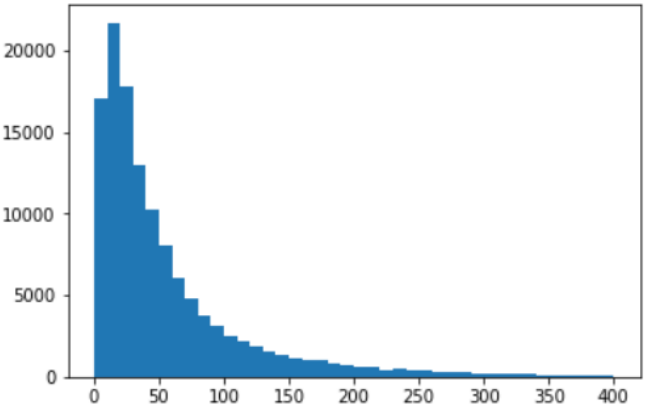

FIG 26 : NUMBERS OF WORDS PER COMMENT

Thanks to this plot, we can see that a length of 50 can easily represent our whole data set. In order to cover more data and regarding the fact that adding 150 more neurons for our LSTM is not too much computation for our machines, we chose to use the optimal length of 200 words.

We create pad sequences of our comments using 200 as a max length. This will had 0 to shorter comment which missed some tokens.

Let's now speak about the structure of our network. First of all, input function is used to create and define a standalone Input layer that specifies the shape of input data. The input layer takes a shape argument that is a tuple that indicates the dimensionality of the input data. When input data is one-dimensional, the shape must explicitly leave room for the shape of the mini-batch size used when splitting the data when training the network. Therefore, the shape tuple is always defined with a hanging the last dimension when the input is one-dimensional.

Outputs from the Input() layer is passed on to the embedding layer where the words are defined in a vector space depending on the surrounding words, the output of the embedding layer is a list of coordinates of the words in the vector space. Basically, it's a mapping of the original input data into some set of real-valued dimensions, and the "position" of the original input data in those dimensions is organized to improve the task. So, similar words might be put on the same dimensional. Hence the overall dimensions are reduced drastically. The distance between words is used to determine the relevance of concepts.

In LSTM, we feed the output of one layer as an input to the next layer. The final output is taken after some number of recursions. As the main task of a recurrent neural network. We want our LSTM to produce output with dimensions as 60. Taking input from the previous layers, LSTM runs 200 times, passing the coordinates of the words each time.

The model x obtained after filling LSTM will be a 3D model, we need to convert the same into a 2D one, hence we use GlobalMaxPool() layer.

In order to get a generalization of the data, we remove some part of the data so that the next layer handles missing data forcefully Dropout(0.1) disables 10% of the nodes.

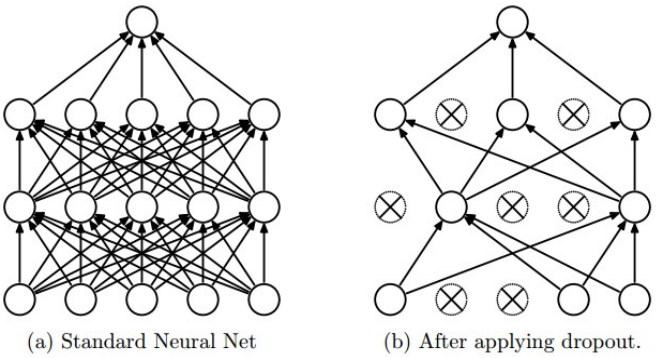

FIG 27 : DROPOUT REPRESENTATION ON A NEURAL NETWORK

The output of Dropout is given as input to a "Relu" for reduced likelihood of vanishing gradient. The ReLU is the most used activation function in the world right now. Since, it is used in almost all the convolutional neural networks or deep learning. This function activation function avoids too much neuron to die, when the gradient fall to 0.

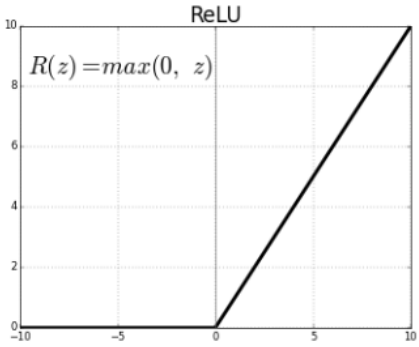

FIG 28 : RELU ACTIVATION CURVE

The dimension of the output is set to 50 Again a Dropout of 10% is achieved and the output is now given to a sigmoid function. The sigmoid function produces an output between 0 and 1, hence we achieve a binary classification for each of the 6 labels.

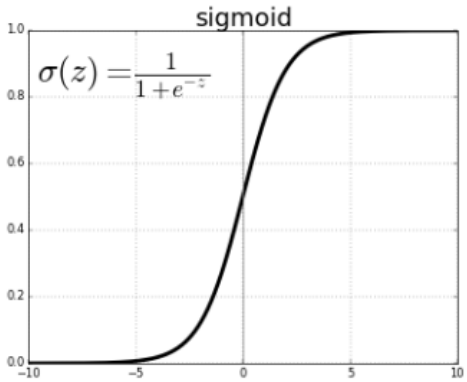

FIG 29 : SIGMOID ACTIVATION CURVE

Using Adam optimization algorithm that can use instead of the classical stochastic gradient descent procedure to update network weights iterative based on training data using Root Mean Square Propagation and Adaptive Gradient Algorithm.

We'll feed in a list of 32 padded, indexed sentence for each batch and split 10% of the data as a validation set. This validation set will be used to assess whether the model has overfitted, for each batch. The model will also run for 2 epochs which is enough regarding the algorithm and the amount of data.

We actually should consider the total training size/batch size, that many numbers of batches pass through our algorithm in each epoch. Typically, you'll split your test set into small batches for the network to learn from, and make the training go step by step through your number of layers, applying gradient-descent all the way down. All these small steps can be called iterations.

Our final structure of the network is the following :

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 200)	0
embedding_1 (Embedding)	(None, 200, 128)	2560000
lstm_layer (LSTM)	(None, 200, 60)	45360
global_max_pooling1d_1 (Glob	(None, 60)	0
dropout_1 (Dropout)	(None, 60)	0
dense_1 (Dense)	(None, 50)	3050
dropout_2 (Dropout)	(None, 50)	0
dense_2 (Dense)	(None, 6)	306

Total params: 2,608,716
Trainable params: 2,608,716
Non-trainable params: 0

FIG 30 : LAYERS STRUCTURE OF THE NETWORK

```
Train on 114890 samples, validate on 12766 samples
Epoch 1/2
114890/114890 [==============================] - 1078s 9ms/step - loss: 0.0770 - acc: 0.9762
- f1: 0.4786 - val_loss: 0.0502 - val_acc: 0.9816 - val_f1: 0.6600
Epoch 2/2
114890/114890 [==============================] - 1121s 10ms/step - loss: 0.0461 - acc: 0.9829
- f1: 0.6679 - val_loss: 0.0477 - val_acc: 0.9816 - val_f1: 0.6682
```

FIG 31 : FINAL RESULTS OF OUR NETWORK

The training model performs over 0.983 in accuracy but has an F1 of 0.67 due to the fact that some labels have a small number of observation. For instance, threat label has 478 observations which are really small compared to the 159571 total observation. It will be interesting to add more observations relative to this label to have good recall for our model. We can see on the previous figure that the second epoch was really useful not regarding the accuracy of the classification but more on the recall of the final classification, as you can see, the F1 score of our classification went from 0.48 to 0.68. By trying to run our model in a third epoch we didn't notice much improvement of our model.

At the end, we can see that the model is really good for classifying non toxic comments over the internet.

VII. CONLUSION

Looking back at the challenge that was posed by this project, We can definitely say that it enabled us to implement some concept that we learned at IIT. Furthermore, it was also an opportunity to learn new algorithms and to step into the realm of deep learning.

Regarding the dataset that we used, we confirmed the importance of the quality of the input data to train our models. Indeed, a poorly distributed dataset was often the source of error. However, it was an interesting challenge to handle such a dataset.

Despite the fact, that the dataset wasn't in our favor, we still managed to optimize our models in order to have the best results as possible. Indeed, even if the Naive Bayes lacked of some training data to perform well, we still managed to have good results using a logistic regression and a LSTM.

References :

[1] : Toxic comments - Insight into datasets - Kernel from Kaggle Competition

[2] : Understanding LSTM Networks, https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[3] Deceiving Google's Perspective API Built for Detecting Toxic Comments, https://arxiv.org/pdf/1702.08138.pdf https://web.stanford.edu/class/cs224n/reports/2762092.pdf

[4] Pipes in R Tutorial For Beginners, https://www.datacamp.com/community/tutorials/pipe-r-tutorial

[5] Develop Your First Neural Network in Python With Keras Step-By-Step, https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/