Online harrasment on social media and knowledge sharing platforms

Anish Philip
W266: *MIDS Final Project Writeup*
anishphilljoe@berkeley.edu

A lot of past study in this area has been focused on identifying the source and psychology of the online harassments across various online platforms, but only recently has the detection of toxic comments/ cyber bullying come into focus. Jigsaw and Google are working to help identify negative comments at scale using language models and making them publicly available (ex. Perspective API). Artificial intelligence and Machine language concepts have the power to identify the trends and patterns of the online harassment across different platforms. Various organizations are continuously working on improving the current methodologies to identify the negative comments (threats, toxic or obscene comments, identity hate, insult) and the population segment involving in these acts as victims and initiators. While there is a long way, many ML models are already capable of identifying the hate speech 80-85% of the time. Much of this work is based on the existing sentiment analysis and spam detection algorithms

Keywords—Natural Language Processing, CNN, RNN, LSTM, Bi-directional LSTM, Word Embeddings, Twitter, NBA, Wikipedia

I. Introduction

The goal of the project was to compare and execute various strategies involved in the design and execution of a typical NLP project and its application in real-world problems. This document will go through in brief, the different pre-processing steps, word embedding selection scenarios, and execution of a few Artificial neural network based supervised learning models to detect and categorize hate speech into 6 different categories. While the learning and validation is done based on a database on online comments from Wikipedia's talk pages, the model is also ran over a million tweets scrapped from official Twitter accounts of NBA players to see if there is a correlation between negative tweets and player performance (based on player-match level data scraped from ESPN).

II. Approach

A. Data set

Five different sets of data sets were leveraged from various sources. The easiest being the Wikipedia comments with labeled training and test from Kaggle. IMDB movie reviews data is fetched form keras library. The most effort intensive part was to scrape the Twitter and ESPN data. Live and historic tweets were scraped for 381 NBA players, 80 media partners, 30 teams and 200 fans using Tweepy listener. 60k Match by match stats were scrapped from stats.espn.com for the latest 3 seasons (2016-18). All of these datasets other than ESPN stats were leveraged to build the word embeddings. In addition, a randomly selected 600k more tweets were included for the processing

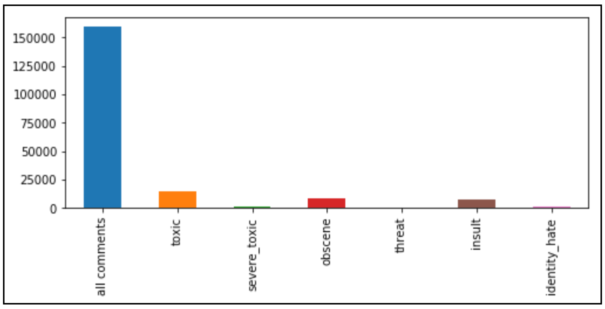


Image1: distribution of all comments by category of hate comments

B. Pre-processing

Once the data was loaded and EDA completed, appropriate pre-processing steps were executed to ensure the most optimal results. Following preprocessing steps were applied:

Tokenization: Two different methods from NLTK library were explored – Regular Word tokenization and Twitter tokenization. Based on the frequency distribution of the most common words I choose to go ahead with Twitter tokenization for its robustness in cases with apostrophes and slightly lower average number of tokens per comment.

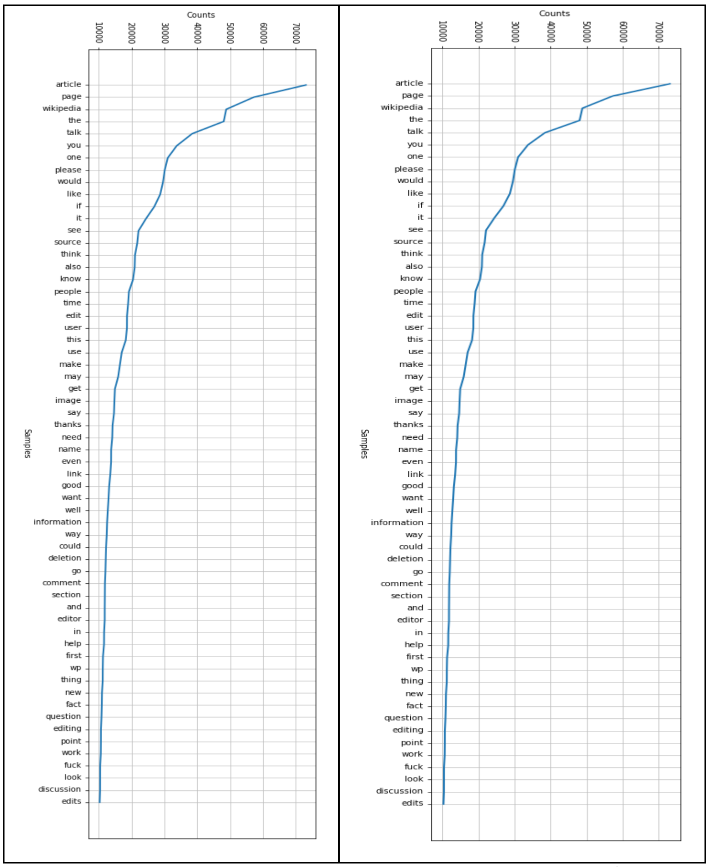


Image 2: Frequency distribution showing regular sentence to word (left) and twitter tokenization (right)

Removing Stop-words: Stop words are very frequent within and across document and generally does not provide any differentiation power in a text classification problem. Hence these words (such as 'is', 'am', 'are', 'this', 'a', 'an', 'the', etc.) can be considered as noise for such NLP tasks

Lexicon normalization: After weighing Lemmatization and Stemming for this step, lemmatization was considered for its understanding of context and sophistication
Lemma: "Better" -> "Good"
Stem: "Connection" -> "Connect"

Emoticons, Number, Special characters and punctuations were removed to reduce the noise. In addition to these Twitter handle tagging, web links and Twitter reserved keywords were also removed

After pre-processing certain records (comments/ reviews/ tweets) were removed that has one or no words remaining. Especially the tweets with web addresses and emoticons.

C. *Methodology*

For baseline calculations none of the preprocessing steps were applied. A bi-directional LSTM model was executed using keras library with a pre-trained Word2vec model built using Google News corpus of 3B words and 300 dimensions for each word. This model achieved a very good accuracy

Category	Accuracy	Precision
Toxic (N=6,088)	93.26%	63.22%
Server toxic (N=367)	99.41%	47.24%
Obscene (N=3689)	96.32%	67.2%
Threat (N=211)	-	-
Insult (N=3425)	95.79%	62.92%
Identity hate (N=712)	-	-

Table 1: Baseline Accuracy and Precision %

The performance of a model was based on the Accuracy, precision and Recall based on the following formula

Precision *= True Positive/ (True Positive + False Positive)*

Accuracy *= (True Positive + True Negative)/ (True Positive + True Negative + False Positive + False Negative)*

Recall *= True Positive/ (True Positive + False Negative)*

All models were optimized with Adam optimizer for 'Binary cross entropy' loss function. Binary cross entropy is actually a combination of sigmoid activation function plus cross entropy. Although a research claims that Categorical Cross-Entropy loss, or Softmax loss worked better than Binary Cross-Entropy loss in their multi-label classification problem. This will be a good next step for future extension of this project but in current scope only 'Binary cross entropy' loss function is being considered

III. WORD EMBEDDINGS

16 different word embedding model were created using the following corpus of text:

Wikipedia comments (Training)	Wikipedia comments (test)	Tweets (NBA specific)	Tweets (Random)	IMDB Movie Review
~150k	~150k	~1m	~0.6m	~0.6m

Word2Vec model from genism library was leverage to generate 16 different predictive word embeddings (5 CBOW and 11 skip grams). Following hyper parameters were modified to identify the best suited word embedding model:

- Number of epochs: Number of training epochs/ iterations over the corpus
- Min count: Ignores all words with total frequency lower than this
- Vector size: Number of dimensions that a word is represented as
- Window size: Maximum distance between the current and predicted word within a sentence
- Model Type: CBOW or Skip-gram: Skip-gram tries to predict the surrounding words given a context word whereas CBOW does the opposite

Although skip-gram models took longer to train, they provided better results with the google analogy test set. Below are the results

A. *CBOW Models*

category	model1	model2	model3	model4	model5
capital common countries	31.05	38.16	37.37	33.95	38.16
capital world	23.41	28.21	29.65	30.13	33.85
currency	1.16	2.33	2.33	2.33	3.49
city in state	10.58	12.72	23.67	30.09	25.81
family	66.01	68.95	69.28	69.93	73.2
gram1 adjective to adverb	7.14	10.22	9.48	7.14	6.4
gram2 opposite	17	20.83	16.67	15.33	16.17
gram3 comparative	55.32	57.38	39.76	44.21	47.14
gram4 superlative	32.07	38.39	28.85	28.74	22.07
gram5 present participle	54.3	54.73	54.95	53.33	50.54
gram6 nationality adjective	49.58	53.12	66.05	65.2	64.36
gram7 past tense	38.26	37.48	35.7	32.93	36.42
gram8 plural	20.77	24.77	19.54	20.31	21.23
gram9 plural verbs	18.75	18.01	13.24	14.34	15.44

Table 3: % match in Google Analogy data set test

Model	Epochs	Min. word Freq	Vec Size	Window
Model 1	10	1	200	1
Model 2	20	1	200	5
Model 3	10	1	200	15
Model 4	10	1	300	15
Model 5	10	2	300	15

Table 4: Definition of all CBOW models

B. *Skip-gram model*

Skip gram provided better performance than CBOW models but were also slow to train.

category	model6	model7	model8	model9	model10	model11	model12	model13	model14	Google News	model15	model16
capital common countries	26.32	50	53.95	55.53	60	52.89	50	58.68	54.74	**83.6**	63.42	**73.57**
capital world	19.93	42.98	45.5	47.18	49.34	47.54	38.06	48.5	46.22	**82.72**	53.3	**53.58**
currency	0	0	0	0	1.16	0	3.49	0	0	**39.84**	0	**0**
city in state	14.25	33.94	26.54	31.99	32.72	37	24.53	32.84	34.8	**74.64**	35.79	**33.47**
family	58.5	37.25	41.18	42.48	41.83	52.61	28.76	41.83	41.18	**90.06**	47.66	**54.97**
gram1 adjective to adverb	5.67	6.65	5.17	4.19	7.02	4.06	4.68	5.17	6.16	**32.27**	8.37	**8.25**
gram2 opposite	14.17	11	11.83	7.67	10.83	7.17	8	7.33	7.17	**50.53**	9.4	**10.26**
gram3 comparative	40.4	27.78	24.13	19.84	20.16	20.95	24.05	18.41	19.37	**91.89**	27.78	**33.73**
gram4 superlative	20.23	12.18	10.57	9.89	7.36	7.59	9.77	7.24	8.28	**88.03**	15.15	**13.23**
gram5 present participle	47.53	42.8	39.14	40.32	42.04	37.2	34.19	37.31	39.46	**79.77**	40.83	**48.49**
gram6 nationality adjective	51.89	66.51	72.98	74.36	74.13	76.44	66.13	76.14	73.75	**97.07**	74.18	**73.38**
gram7 past tense	29.3	26.6	26.67	25.11	22.19	22.19	25.18	21.69	22.9	**66.53**	30.16	**30.03**
gram8 plural	17.08	10.92	14	12.77	13.38	11.54	14.15	15.85	12.15	**85.58**	15.02	**18.33**
gram9 plural verbs	6.99	4.78	6.25	3.31	4.78	4.04	8.09	5.15	7.35	**68.95**	13.4	**11.44**

Table 4: % match in Google Analogy data set test (Skip-gram models)

Google news is the word embedding pretrained on Google news corpus with a large vocabulary. Although we are not using it directly. We are using it in as a standard for how the custom word embeddings should perform. Since the corpus that is used to train the Word2vec model is very small compared to the Google news corpus, the performances cannot be equivalent but still Model 15 and 16 are sufficiently performing. Below is the definition of each model, you will notice the reason behind the model 15 and 16 outperforming all other models is not just the configuration but the additional data that is feed for training (imdb movie reviews and another half a mill random tweets)

Model	Epochs	Min. word Freq	Vec Size	Window
Model 6	10	1	200	5
Model 7	20	2	200	10
Model 8	10	2	200	15
Model 9	20	1	200	15
Model 10	20	2	200	15
Model 11	20	1	300	15
Model 12	20	1	100	15
Model 13	20	2	300	15
Model 14	20	5	200	15
Model 15**	20	2	200	15
Model 16**	20	2	200	15

**Model 15 and 16 have additional data. Model 15 considers an additional set of imdb movie reviews and 16 has random Tweets plus the regular training data including imdb data

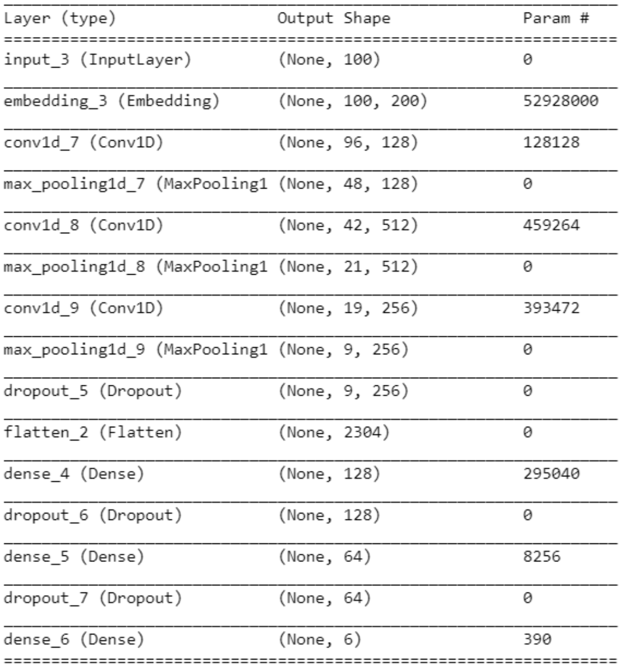
After this exercise we can say with some level of confidence that we don't need a large vector to represent words in larger vocabulary. Word embedding between 100 to 300 should work just fine. Longer vector representation doesn't necessarily add more accuracy and from the above data we can see 200 dim vector representation provides decent results. This would probably be more strengthened when we look at some of the LSTM models in the next segment. A LSTM with 50-60 hidden layers with a fully connected 50-100 neuron could potentially outperform a MLP of 1000 neuron. Although a pre-trained vector could be easier to use and would provide similar accuracy score eventually, it was important to go through this exercise to understand the important hyperparameters that help optimize the NLP models.

IV. CNN, LSTM, STACKED-LSTM AND BI-DIRECTIONAL LSTM

Let’s take a quick look at a few models that we put together to accurately calculate the problem at hand. Which is to categorize negative comments into 6 categories

A. Convolutional Neural Network

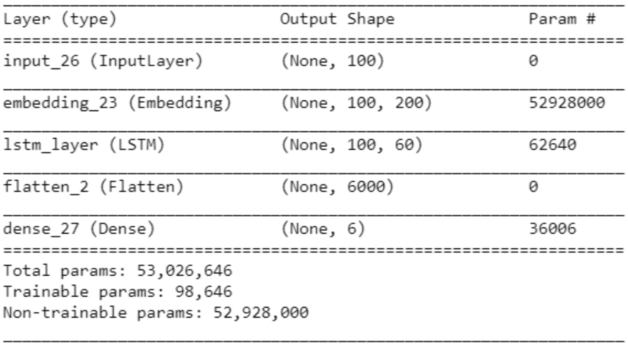
3 1D Convolution Layers with, Max pooling and recurrent dropout layers to avoid overfitting



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         (None, 100)               0
_________________________________________________________________
embedding_3 (Embedding)      (None, 100, 200)          52928000
_________________________________________________________________
conv1d_7 (Conv1D)            (None, 96, 128)           128128
_________________________________________________________________
max_pooling1d_7 (MaxPooling1 (None, 48, 128)           0
_________________________________________________________________
conv1d_8 (Conv1D)            (None, 42, 512)           459264
_________________________________________________________________
max_pooling1d_8 (MaxPooling1 (None, 21, 512)           0
_________________________________________________________________
conv1d_9 (Conv1D)            (None, 19, 256)           393472
_________________________________________________________________
max_pooling1d_9 (MaxPooling1 (None, 9, 256)            0
_________________________________________________________________
dropout_5 (Dropout)          (None, 9, 256)            0
_________________________________________________________________
flatten_2 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               295040
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_5 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_7 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_6 (Dense)              (None, 6)                 390
=================================================================
```

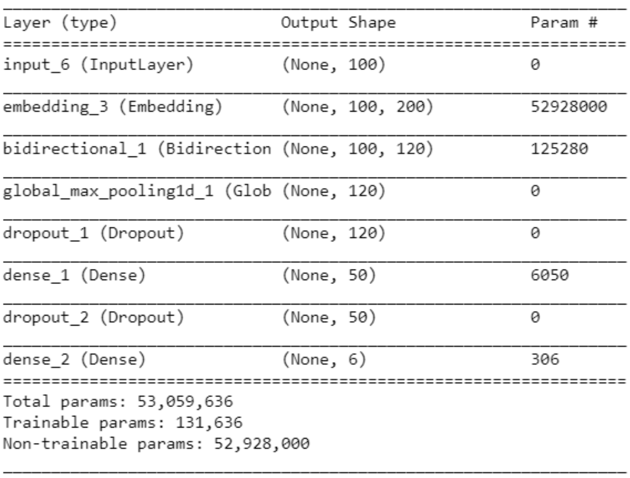
B. Simple LSTM Layer

Simple LSTM config with 100 hidden neurons



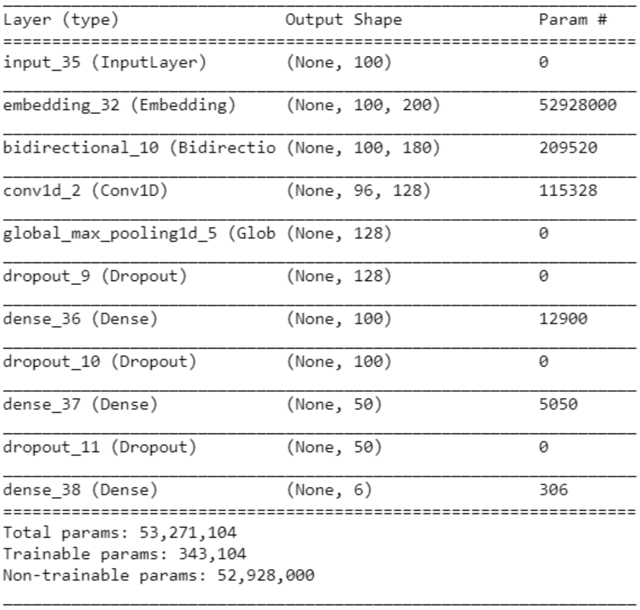
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_26 (InputLayer)        (None, 100)               0
_________________________________________________________________
embedding_23 (Embedding)     (None, 100, 200)          52928000
_________________________________________________________________
lstm_layer (LSTM)            (None, 100, 60)           62640
_________________________________________________________________
flatten_2 (Flatten)          (None, 6000)              0
_________________________________________________________________
dense_27 (Dense)             (None, 6)                 36006
=================================================================
Total params: 53,026,646
Trainable params: 98,646
Non-trainable params: 52,928,000
_________________________________________________________________
```

C. Bi-Directional LSTM



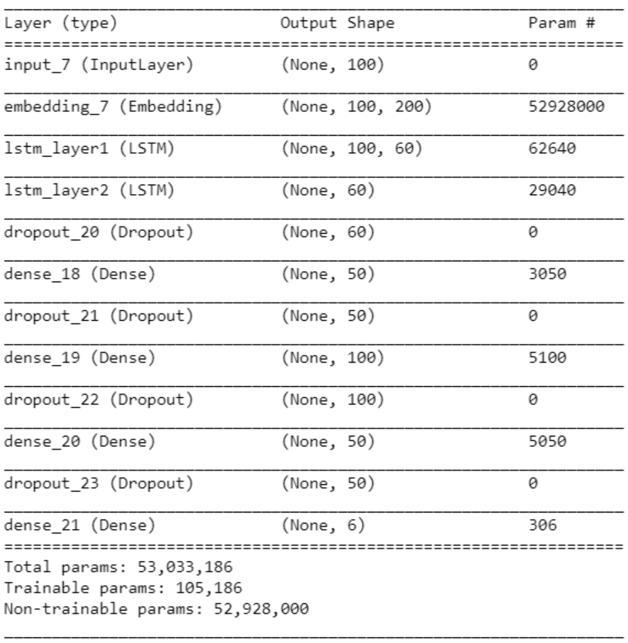
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         (None, 100)               0
_________________________________________________________________
embedding_3 (Embedding)      (None, 100, 200)          52928000
_________________________________________________________________
bidirectional_1 (Bidirection (None, 100, 120)          125280
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 120)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 120)               0
_________________________________________________________________
dense_1 (Dense)              (None, 50)                6050
_________________________________________________________________
dropout_2 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 306
=================================================================
Total params: 53,059,636
Trainable params: 131,636
Non-trainable params: 52,928,000
_________________________________________________________________
```

D. Bi-Directional LSTM with Convolutional Layer



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_35 (InputLayer)        (None, 100)               0
_________________________________________________________________
embedding_32 (Embedding)     (None, 100, 200)          52928000
_________________________________________________________________
bidirectional_10 (Bidirectio (None, 100, 180)          209520
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 96, 128)           115328
_________________________________________________________________
global_max_pooling1d_5 (Glob (None, 128)               0
_________________________________________________________________
dropout_9 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_36 (Dense)             (None, 100)               12900
_________________________________________________________________
dropout_10 (Dropout)         (None, 100)               0
_________________________________________________________________
dense_37 (Dense)             (None, 50)                5050
_________________________________________________________________
dropout_11 (Dropout)         (None, 50)                0
_________________________________________________________________
dense_38 (Dense)             (None, 6)                 306
=================================================================
Total params: 53,271,104
Trainable params: 343,104
Non-trainable params: 52,928,000
_________________________________________________________________
```

E. Stacked LSTM



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_7 (InputLayer)         (None, 100)               0         
_________________________________________________________________
embedding_7 (Embedding)      (None, 100, 200)          52928000  
_________________________________________________________________
lstm_layer1 (LSTM)           (None, 100, 60)           62640     
_________________________________________________________________
lstm_layer2 (LSTM)           (None, 60)                29040     
_________________________________________________________________
dropout_20 (Dropout)         (None, 60)                0         
_________________________________________________________________
dense_18 (Dense)             (None, 50)                3050      
_________________________________________________________________
dropout_21 (Dropout)         (None, 50)                0         
_________________________________________________________________
dense_19 (Dense)             (None, 100)               5100      
_________________________________________________________________
dropout_22 (Dropout)         (None, 100)               0         
_________________________________________________________________
dense_20 (Dense)             (None, 50)                5050      
_________________________________________________________________
dropout_23 (Dropout)         (None, 50)                0         
_________________________________________________________________
dense_21 (Dense)             (None, 6)                 306       
=================================================================
Total params: 53,033,186
Trainable params: 105,186
Non-trainable params: 52,928,000
_________________________________________________________________
```

V. References

[1] Ex Machina: Personal Attacks Seen at Scale 2017 International World Wide Web Conference Committee

[2] Deceiving Google's Perspective API Built for Detecting Toxic Comments ArXiv 2017

[3] Modeling the Detection of Textual Cyberbullying 2011, Association for the Advancement of Artificial Intelligence

[4] Text categorization with support vector machines: learning with many relevant features In Proceedings of ECML-98, 10th European Conference on Machine Learning (Chemnitz, DE, 1998), pp. 137–142

Model	Toxic			Obscene			Insult		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall
CNN	90.83%	53.35%	29.85%	94.69%	56.77%	33.88%	94.81%	52.8%	30.6%
Simple LSTM	92.74%	59.87%	72.16%	96.05%	64.86%	69.1%	95.89%	61.89%	60.7%
Bi-Directional LSTM	92.81%	59.94%	74.06%	96.2%	66.63%	68.53%	96.03%	64.0%	59.27%
Bi-Directional LSTM with Conv layers	93.09%	61.72%	72.49%	95.89%	62.31%	72.81%	96.33%	68.41%	58.69%
Stacked LSTM	93.26%	63.22%	69.94%	96.32%	68.42%	67.2%	95.79%	62.92%	52.41%

Note: Additional models are present in the code. These are the ones that were identified as the better ones

In summary, it is preferable to tray many architectures and cross-validate them with smallest number of neurons possible and then start building on them depending on the computational resource and development time available. In all these models there were very limited variation in accuracy although some additional analysis would need to be done in order to enhance some of these models that were constrained by computational resource or development time.

Code: https://github.com/anishphilljoe/NLP/blob/master/Final_Submission.ipynb
Extended Results: https://github.com/anishphilljoe/NLP/blob/master/Online%20Comments%20Classification.pptx