# Machine Learning Engineer Nanodegree
# Toxic Comment Classifier

Stephen O'Kennedy

July 4, 2018

## 1 Definition (approx. 1-2 pages)

### 1.1 Project Overview

Since it's inception the internet has allowed people from most parts of the world to freely communicate, debate, and collaborate with each other over a wide range of topics and projects. Platforms like Github, Hackernews, Twitter, Wikipedia, etc. form the foundations for which these interactions can take place. Many of these communities have standards and rules in place to facilitate conversations, and to prevent these communities from being hijacked, or destroyed by toxic behaviour. It is becoming increasingly harder to regulate and enforce these standards. In fact Facebook are currently hiring more and more moderators to sift through questionable content [7].

Conversation AI [1] are working to provide tools to help improve online conversation[4]. One area that they're focusing on is the study of negative online behaviours, like toxic comments[4]. As their Kaggle page states, the current models in use for detecting toxic comments still make errors, and they don't allow users to be able to identify the types of toxicity they're interested in finding. For example some platforms may be fine with comments that contain profanities. Ultimately we want to define a model that can perfrom sentimental analysis on Wikipedia comments.

As we can see from Pak and Paroubek [9] work there is a significant body of research into sentimental analysis. Their work focused on sentiment analysis which is fairly similar domain to what we'll be working on. Pak and Paroubek defined a sound methodology for pre processing text using the bag of words technique. They proceeded to use both SVM and Naive Bayes classifiers to identify if the sentiment of a tweet to was Positive, Neutral, or Negative, and found that Naive Bayes classifier was more accurate. Furthermore, they were able to boost their classifier's accuracy by using bigrams when they vectorised their tweets. It would appear that their work could be used as the basis for a baseline model for evaluations [9].

### 1.2 Problem Statement

Given a dataset that contains a large number of Wikipedia comments which have been labelled by human moderators for toxic behaviour. We want to

create a model that predicts the probability of different types of toxicity for each comment.

To solve this problem we'll begin by establishing benchmark models to prove that this problem is solvable. We'll prove this by having a model achieve an evaluation metric score of greater than 50%. The two baseline models we'll look to use are MultinomialNB and NbSvmClassifier. However, before we do that we'll need to perform text pre-process. For our baseline models we'll use the bag of words technique. Raschka and Mirjalili [11] explain that the **bag of words** technique allows us to represent text as numerical feature vectors. The idea behind the bag of words model is to simply create a vocabulary of unique tokens, and count the occurrence of each word in our dataset. We'll aim to take this a step further and attempt to assess word relevancy via term frequency-inverse document frequency. We will delve deeper into what this means in section 3.1.

Once we've processed our dataset and evaluated our baseline models we'll aim to meet and hopefully surpass our baseline models with the use of Convolutional and Recurrent Neural Networks (CNN and RNN). We will also be looking to use a different text preprocessing technique known as Word2Vec.

Both CNN and RNN fall within a sub field of machine learning known as deep learning [11]. We will be using Google's Tensorflow [5] as our backend, which is a highly optimised library that can efficiently train a neural network through the use of powerful GPU's. Our intention is to not directly interact with the tensorflow library. Instead we'll rely on Keras [2] as our interface for building neural networks.

Ultimately we will want our deep neural networks to surpass our baseline models. In the below section we'll be discussing the metrics that we will use to evaluate our models.

## 1.3 Metrics

Models will be evaluated by using the ROC AUC metric. Each comment in the test data set will need to be labelled with the predictions for each type of toxicity appearing in each comment.

ROC is the receiver operating characteristic curve. It is a graphical plot that displays the discrimination threshold of a binary classifier, which is what we'll need to build. Our threshold ($T$), which is used to classify a data-point as either positive or negative, is by default set 0.5. We take the true positive rate ($TPR$) and false positive rate ($FPR$) for all scores and plot a curve. Calculating the AUC (area under the curve) will reduce the curve down to a single value, $1 \geq A < 0$. Where $A$ is the AUC. If $A$ is close to 1.0 we've got a perfect classifier, if however it is 0.5 of lower than our classifier is doing little more than guessing [3].

The formula is:

$$A = \int_{\infty}^{-\infty} TPR(T)FPR'(T)dT = \int_{-\infty}^{\infty}\int_{\infty}^{-\infty} I(T' > T)f_1(T')f_0(T)dT'dT = P(X_1 > X_0)$$

# 2 Analysis (2-4) pages

## 2.1 Data Exploration

The dataset we'll be using is stored in a CSV format. Our dataset contains 159571 rows and 8 columns. Below are the first five rows of our dataset.

| id | comment text | toxic | severe toxic | obscene | threat | insult | identity hate |
|---|---|---|---|---|---|---|---|
| 000099... | Explanation\nWhy... | 0 | 0 | 0 | 0 | 0 | 0 |
| 000103f... | D'aww! He matches this... | 0 | 0 | 0 | 0 | 0 | 0 |
| 000113f... | Hey man, I'm really... | 0 | 0 | 0 | 0 | 0 | 0 |
| 0001b41... | "\nMore\nI... | 0 | 0 | 0 | 0 | 0 | 0 |
| 0001d95... | You, sir, are my hero... | 0 | 0 | 0 | 0 | 0 | 0 |

Table 1: First five rows of the toxic comment dataset

As we can see in Table 1 we have 8 columns. The id column is of no real use to us. However, we'll want to draw our focus onto the other seven columns that make up our dataset. The first column we see is the comment_text it contains a mix upper and lower case words as well as escaped characters, punctuation and other non alphanumeric characters. The other columns can contain either 0 or 1 and act as boolean values for the different types of toxicity. The different types of toxicity are as follows: [4]

- toxic
- severe_toxic
- obscene
- threat
- insult
- identity_hate

We first began our initial analysis of the comments by finding the minimum, maximum and mean length. We found that the shortest comments were 6 characters long and the longest were 5000. We also see that the average was approximately 394 characters long. From looking at these values we know that our dataset has no missing value entries.

We then began looking at the breakdown of what percentage of the comments fell into the various types of toxicity. Below are our findings

```
Percentage of toxic comments: 9.58%
Percentage of severe_toxic comments: 1.00%
Percentage of obscene comments: 5.29%
Percentage of threat comments: 0.30%
Percentage of insult comments: 4.94%
Percentage of identity_hate comments: 0.88%
Percentage of clean comments: 89.83%
```

We can see above that our data set is very unbalanced. With none of the labels coming near 10% of the data set let alone anywhere near 50%. When we

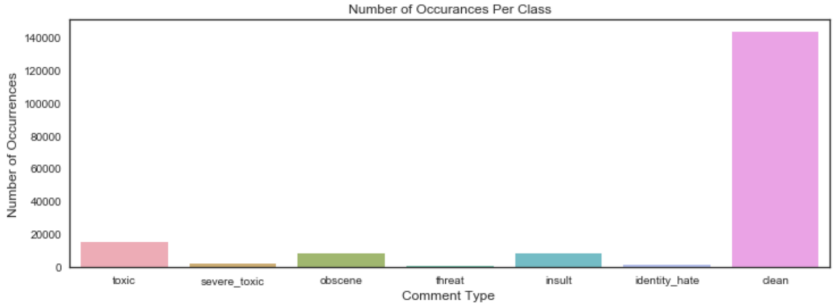
go to train our network we want to be sure that we take steps to ensure our models give a true output of it's "accuracy". Therefore, we can initially rule out the use of the arithmetic mean (A.K.A accuracy) as our evaluation method. Furthermore, given that we're trying to perform to build a classifier that can output multiple classification the ROC AUC evaluation metric would be a good choice.

Lastly, we performed some statistical analysis on the sentence and word counts in our datasets, and below are our findings. The most interesting statistic that we want to draw our attention to is that 75% percentile for comment_word_count which is 75. This value can then be used later to decide on how to best pre process the each comment.

| | comment_word_count | comment_sentance_count |
|---|---|---|
| **count** | 159571.000000 | 159571.000000 |
| **mean** | 67.273527 | 8.536376 |
| **std** | 99.230702 | 28.581049 |
| **min** | 1.000000 | 1.000000 |
| **25%** | 17.000000 | 3.000000 |
| **50%** | 36.000000 | 5.000000 |
| **75%** | 75.000000 | 9.000000 |
| **max** | 1411.000000 | 4945.000000 |

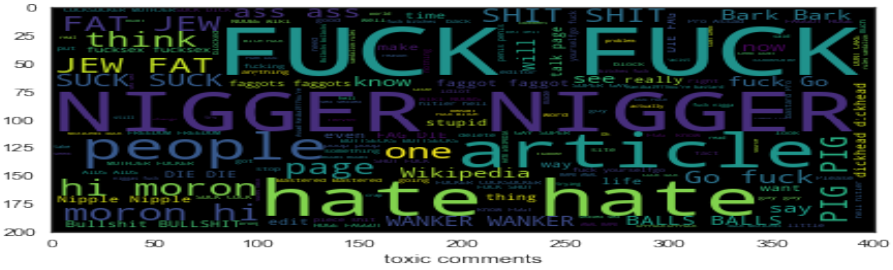
## 2.2 Exploratory Visualization

After our initial data exploration we began working on the visualisations of the data set.




As we can see in the above figure most of our comments are marked as clean. This means that they do not fall into any of the other categories of toxic comments. We can also see that toxic, obscene and insult labelled comments are the most common types of toxic behaviour and the remaining three appear to much rarer in the dataset.

The next visualisation we performed are word clouds. Word clouds show the frequency of words by increase the size relative to others. Please note that given the topic of the dataset we are dealing with, offensive and racially derogatory words will be present in the following visualisation.


FUCK
NIGGER
article
people
hate
toxic comments

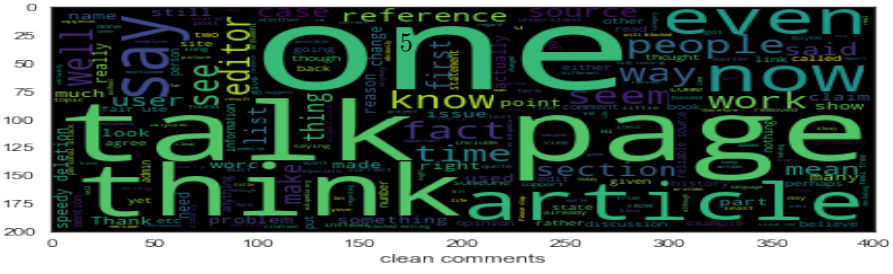

ass
SUCK
FUCK
SHIT
severe_toxic comments


NIGGER
FUCK
SHIT
SUCK
ass
obscene comments


ass
DIE
MUST
WALES
JIM
threat comments


NIGGER
FUCK
FAT
JEW
HATE
insult comments


NIGGER
FAT
JEW
FAGGOT
SUCK MEXICANS
identity_hate comments


one
talk
page
think
article
clean comments

As we can see in the above visualisation we see that there are some words that come up very frequently in the different categories of comments. One noticeable word that does appear frequently is "Die" in the threat comments word cloud. Furthermore from this visualisation we can see that there are some words that are seemingly common across the different toxic comments levels. This does highlight that there's a significant overlap between the different categories. And perhaps, there is a strong correlation between them.

## 2.3 Algorithms and Techniques

Given that this is an natural language processing (NLP) problem we'll want to make use of several techniques and algorithms to solve this classification problem. The first problem we have is to convert our comments from raw text to vectors that in turn can be fed into our models. There are two approaches that we decided the first is the bag of words technique in conjunction with term frequency-inverse document, which is a way of assessing word relevancy [11]. The other approach is to use Word2Vec algorithm to create vectors that maintain a semantic relationship between other words [8].

Once the comments in our dataset have been converted into vectors by both techniques described above, we will be able to simply feed them into our models in order to train them. We will also need to perform the similar steps on our testing data.

There are two deep learning algorithms that my provide effective solutions to us, that we wish to discuss. These algorithms are Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN).

As Raschka and Mirjalili describes, CNNs are a family of models that were inspired by the how the visual cortex of the human brain works when recognizing objects. CNNs essentially navigate over a matrix by taking strides and analysing the elements that fall within that stride.

Unlike other deep neural networks, or other classical machine learning algorithms, RNNs rely on sequential data. This means that order matters [11]. RNNs were designed to model sequences and are capable of rembering past information and process new events accordingly. RNNs have found a home in domains that rely on data to be sequenced for example image captioning, and text generation. They may also be capable of learning sentiment. Which could prove to be an effective solution to our problem. Raschka and Mirjalili explains that the RNN networks can have one more LSTM layer. Where the hidden layers receive inputs from the input layer, like other layers, but also from other hidden layers. The effect this has is that it gives a neuron "memory".

## 2.4 Benchmark

Given that the dataset is part of a Kaggle competition, we could use the public leader boards as our benchmark. While this does gives us scores that we could try to surpass, the test data that is used to evaluate those models was part of a dataset that we don't have access to the labels, we will be unable to include the scores as part of the code supplied with this project. Therefore, we'll be establishing our own benchmarks.

To do this we'll be using the bag of words technique to pre-process the data and then through the use of Naive Bayes and Naive Bayes - Support Vector Machine (NBSVM) we'll be able to create our benchmarks.

We chose Naive Bayes because not only are the models relatively simple, they are also known to be very robust when performing sentimental analysis [10]. We also looked at NBSVM as another model to use as our baseline. According to Wang and Manning NBSVM performs well on snippets and longer documents, for sentiment, topic and subjectivity classification, and is often better than previously published results. Wang and Manning also note that use of Logisitic Regressoion with naive bayes do in fact yield similar results. With the added benefit of being efficient. Therefore, in the implementation included in the code base, we use the Logisitic Regressoion.

The benchmark models were defined with the following hyper parameters:

```
MultinomialNB
        alpha = 0.01
NbSvmClassifier
        C = 1.0
        n_jobs = 1
```

Both models achieved an ROC AUC score of 0.9101 and 0.9826 respectively. Our future CNN and RNN should aim to surpass these scores.

# 3 Methodology (approx. 3-5 pages)

## 3.1 Data Processing

As we discussed in section 2.3 we're working on a form of sentimental analysis on comments and we will need to convert that text into a numerical feature vectors. We also mentioned that we are using two approaches to process the text, the bag of words model and word2vec.

We used the bag of words model to process the data for our baseline models. Raschka and Mirjalili describe the bag of words model as follows:

"The bag of words model is where we create a vocabulary of unique tokens, for example words, from the entire set of documents. We construct a feature vector from each document that contains the counts of how often each word occurs in the particular document."

When we performed this tokenisation step we also strip away all escaped characters and punctuation. Furthermore we set them to lower case. We also break our comments into to both uni and bi-grams. We do this because a collection of words can have a different meaning than if the words were recorded individually. While we were tokenising our text with the bag of words model we went one step further and used term frequency-inverse document frequency (tf-idf) to assess the word relevance. Raschka and Mirjalili explains that the frequently occurring words typically don't contain useful of discriminatory information. The tf-idf can be described as the product of the term frequency and the inverse document frequency.

$$tf - idf(t, d) = tf(t, d) \times idf(t, d)$$

Where $t$ refers to the number of times $t$ occurs, $d$ refers to the document. $idf$ is defined as:

$$idf(t, d) = \log \frac{n_d}{1 + df(t, f)}$$

Here $n_d$ is defined as the total number documents, and $df(d, t)$ is the number of documents $d$ that contain the term $t$. The log is used to ensure that low frequency terms are not given too much weight[11].

Fortunately much of the leg work was done by Pedregosa et al.. They provide us with TfidfVectorizer model which implements most of this.

Word2Vec on the other hand is where we encode each word in our dataset to as a numeric feature vector [8]. However, we try to maintain the semantic relationship between words. This is done by tokenising each word into a vector. Then we split the vectorised document into bi-grams and using one word as the input and the other as the output we train a deep neural network [8]. The idea behind this is that we can use this model to vectorise new text and attempt to capture the relationship between different words. By maintaining this semantic link between words, we hope to build a deep neural network that can capture more nuanced "meaning" behind comments.

In this project we didn't train our own model, as we would need to a comprehensive dataset to be our vocabulary. While we could have split our dataset

in to training and testing data. Our model would probably not generalise too well when dealing with new words. We therefore decided to leverage an existing pre-trained word2vec model.

The pre-trained model is Bojanowski et al.'s fastText. It is model trained on wikipedia and each vector is represented in 300 dimensions. It uses Bojanowski et al.'s skip-gram model.

To use this model with our own dataset we simply imported the library as a dictionary, with the key being the word and the value being the vector, and performed a lookup.

## 3.2 Implementation

We first began with the implementation of our baseline models. This involved us with splitting our dataset into training and testing sets. We decided to use 33% of our dataset as our testing set. We then created our bag of words model to convert the text in our dataset into numerical feature vectors.

Up to this point we have mostly relied on Pedregosa et al.'s sklearn library to process our data, and our intentions were to also use the Multinomial Naive Bayes algorithm, which comes out of the box with this library, as well. However, given that we're trying to decide if a comment can be categorised as a certain type of toxic comment and each category is not mutually exclusive with other categories. Therefore we needed to work around this issue by creating a classifier for type of category of comment. We achieved this by creating a list of classifiers that are trained for each category of comments.

Furthermore for each classifier we had to output a score $x$ that is $0 \geq x \leq 1$ in a vector format. The main reasoning behind this was that it was part of the acceptance criteria for the Kaggle competition [4]. Fortunately sklearn classifiers are able to to output the probability of a prediction via the predict_proba method.

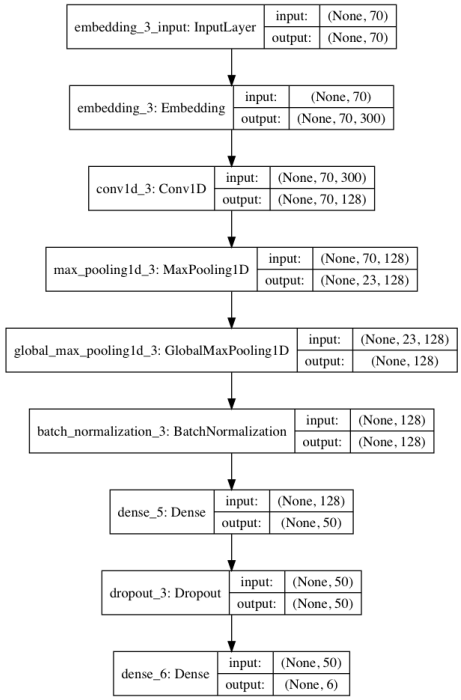
When we followed the above implementation for the MultinomialNB model we achieved a score 0.9101. We had followed the same steps for the NBSVM classifier. However, we had to implement the classifier ourselves. You can find the implementation we used in our benchmark tests in the toxic_comment_classifier notebook. The implementation follows the design pattern that other classifiers in the sklearn library uses. By doing this we can leverage the wide range of tools that are provided by the sklearn library.

When we began working on our deep neural network we decided to use the keras library to build our networks. The main reason behind this is that they provide an easy, intuitive way of building complex networks in a declarative manner. However, before we began implementing our network we began processing out data. Our first step was to load the dataset and split it into training and testing data. We decided to hold back 20% of the dataset for testing. We then moved on to process our text data.

With our baseline models we had simply used the bag of words model to convert convert our comments into numerical feature vectors. We first began by assigning unique tokens for each word in our dataset. However, we needed to ensure that each comment vector we have is of a fixed size. We experimented with different sizes and found that 70 entries in length was the optimum length. We settled on this number as we felt it was the best trade off between having

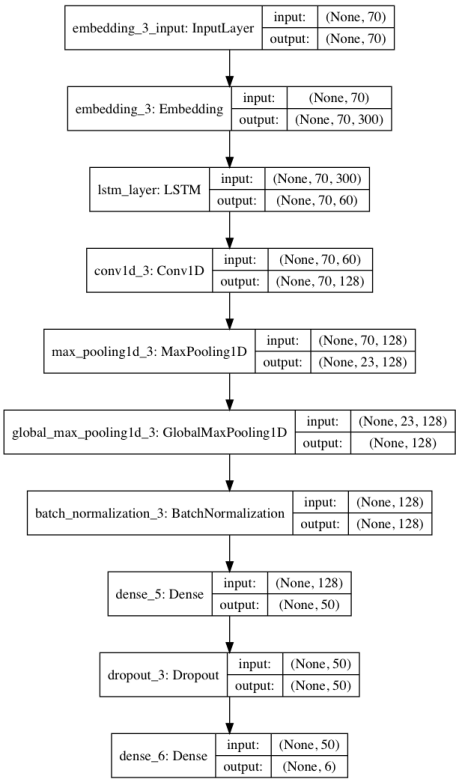
data that was complex enough to identify nuanced patterns, but we didn't suffer too much from the curse of dimensionality. If comments exceeded this length we simply truncated them. If however, we were handling shorter comments then we would simply pad out the comments until they were at the appropriate length. The next step we began working on was the embedding matrix. According to Raschka and Mirjalili, the idea behind embedding is to a feature learning technique that that can be utilised to automatically learn the salient features to represent the words in the dataset. While we can build our own embedding matrix, we instead decided to leverage a pre-trained embedded layer. We did this by using Bojanowski et al.'s fastText pre-trained vectors.

We then began building our Convolutional Neural Network (CNN). We can see below the CNN architecture that we used in this project.




In the above figure we can see that we have embedding input layer that will take our tokenised text and pass it though our embedding layer. We then pass the outputs to a convolutional layer. From this layer perform max pooling and global max pooling to reduce the dimensions of our outputs. We then proceed to normalise our batches. After this we pass the outputs to a dense layer and finally we pass our output to the output layer. The output layer contains 6 nodes and we used the sigmoid activation function as this will output values from 0 to 1.

The final neural network we wanted to build was a Recurrent Neural Network (RNN). The RNN architecture we designed can be seen below. The architecture is broadly the same as the CNN we defined earlier. However, instead of having a Conv1D layer we instead use LSTM layer.




## 3.3 Refinement

When we designed our initial CNN and RNN architectures, we deliberately chose relatively simple architectures. When we trained and tested them on our testing data. They had outperformed our baseline models. Albeit marginally. We therefore made some minor, manual tweaks to our process. With regards to our baseline models we did attempt to improve the computational speed of them by batching the training data and using thread pools to train and predict the data in parallel. This was to allow us to perform a more automated refinement process. However, due to time constraints we focused on other parts of the project.

# 4 Results (approx. 2-3 pages)

When undertaking the problem of creating a toxic comment classifier, we had decided to create our own benchmarks using classical machine learning techniques and algorithms. After creating and defining our baseline models and scores we began working on our deep learning models. With the aim of surpassing our baseline models. We can the ROC AUC scores for both our baseline, CNN and RNN models in Table 2

| multinomial | SVM NB | CNN | RNN |
|---|---|---|---|
| 0.91016 | 0.98265 | 0.98186 | 0.98280 |

Table 2: ROC AUC scores

As we can see the in Table 2 the multinomial naive bayes classifier was the worst performing classifier of the four. It had achieved a score of 0.91016. Which, given its relative simplicity isn't by any stretch of the imagination, bad. There is of course an opportunity to use hyper parameter tuning to try and create a more satisfactory model. When we were testing this model we performed a train/test split, where 33% of the dataset was held back for testing. Purely on this basis we would that the data that the model was tested on was not seen by the model. Therefore we can say that the multinomial naive bayes model is able to generalise to unseen data, but it does so with a poorer performance when compared to the other models. We also found that the model is not robust enough for the problem. If we looked at some examples of toxic comments we can see that there are some obvious characteristics of threatening comments. Below are two examples:

```
Comment 1:
'Hi! I am back again!\nLast warning!\nStop undoing my edits or die!'

Comment 2:
"I'm also a sock puppet of this account...SUPRISE!!\n--sincerely,\n
The man that will track you down from the Internet and kill you"
```

We tokenised this text and had our multinomial naive bayes classifier calculate the probability of these comments of being threatening. Below are the results:

```
Comment 1: 0.0008037202832728238
Comment 2: 0.0014198664949907798
```

As we can see our classifier was not able to correctly identify these threatening comments. As a result I would not trust this model.

The other benchmark model we developed was the SVM Naive Bayes classifier. This model performed considerable better than the Multinomial Naive Bayes. It's ROC AUC score is 0.98265. We feel that this is a considerably more robust classifier. The classifier was tested on the two above sample comments. Below are the results.

```
Comment 1: 0.0008037202832728238
Comment 2: 0.0014198664949907798
```

As we can see in Table 2 the ROC AUC score for the CNN is 0.98186 which is a significantly higher score than the Multinomial Naive Bayes classifiers. The architecture that we used is quite simple, and while it may not match the SVM NB classifier, we do feel that there is scope to improve the architecture of the CNN and be able to match and if not surpass the SVM NB classifier. However we do feel that the final parameters of the model are not appropriate. As we mentioned we did not develop our CNN architecture to be more complex. Furthermore we didn't try running training our CNN over more than 10 epochs. We found that as we trained up to 10 epochs the time it takes to train the model is increased and the performance decreased.

We tested our CNN classifier on Comment 1 and Comment 2 to see how well the model generalises to unseen data and to see if the model is robust enough to deal with classifying toxic comments. We can below the threat scores for the CNN model:

```
Comment 1: 2.4979036e-02
Comment 2: 9.5797271e-01
```

We can see that the model has given a poor probability score of whether or not a the comments are toxic or not. We would have held the view that the CNN would've performed better than at classifying these two comments. However, we would like to highlight that the model is simple and there is scope to perform hyper parameter tuning to develop and identify a ore robust model.

The final model we would like to report our results on is our RNN model. Given that RNN networks leverage Long-Short-Term-Memory (LSTM) layers allow "knowledge" to be "remembered" we had anticipated that this model would be the best performing of the four we have trained. Once again we have developed a relatively simple RNN model. And as such we are sure that we there is scope for further improvement of the model. As we can see in Table 2, the RNN model achieved a score of 0.98280. Which is a an improvement of 0.00015 over the SVM NB model.

Like the other models the RNN model was tested on the two sample threat comments and the results are below.

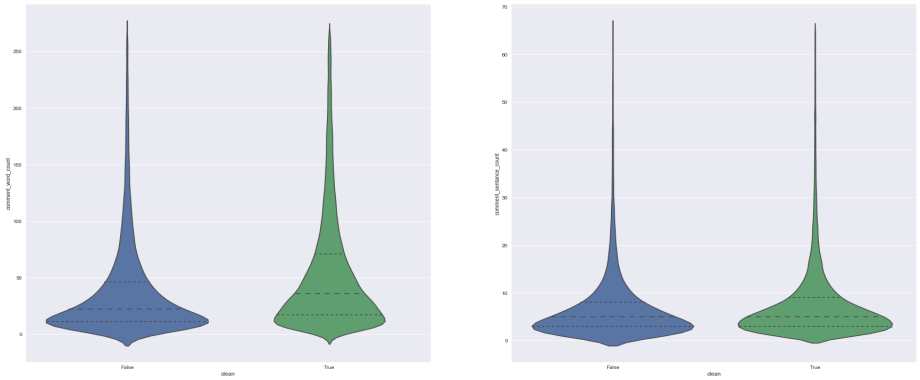
```
Comment 1: 0.00419312
Comment 2: 0.0778641
```

We can that unlike the SVM NB model it was unable to correctly classify comment 1 as a threat, and it was also unable to correctly classify comment 2. However we did find that RNN was able to classify comment 2 as a toxic comment with a probability score of 0.6163382. We would fee that the model is robust enough for explicit toxic comments in most cases. However, we feel that it fails on more nuanced comments.

Ultimately, we would feel that more extensive evaluation of models could be performed to validate them. Given that our evaluation is based on how well our models perform across all categories and not just on one particular category, there is valid reasoning in evaluating a model on each category independently and using another metric such as an f1 score.

# 5 Conclusion

## 5.1 Free-Form Visualization



One of the important qualities of the project we wished to emphasis here is some of the latent feature we didn't use directly in our project. But we feel are worth mentioning. We felt it would be interesting if people who composed toxic comments used more words or typed more sentences. Such features could be useful in future works determine if there's a relationship between toxic comments and word or sentence length.

We therefore generated the diagram depicted above are two violin plots that depict the commonality of the different lengths for the number of words in the left plot and the number of sentences in the right plot. We had removed outliers where the $z$-score was $< 2$. We did this to ensure that we could make it clear what the distributions were for the rest of the dataset. The "clean" comments column was a value we engineered to filter out comments that fell into none of the existing toxic comment categories. The $x$-axis tells us if the comments were clean or not, $y$-axis informs us the number of words and sentences for clean and non-clean comments.

Interestingly we can see that there non-clean comments seem to be shorter in terms of word count, and clean comments appear to be more evenly distributed. With regards to the sentence length there doesn't appear to a difference in sentence lengths that is as striking as the word count. We feel that the word count could provide us with an additional insight in how to more effectively process our data.

## 5.2 Reflection

From the outset we were rather fortunate with the data we were working with. It was mostly complete and there were a little work in the way of data munging. Furthermore the task at hand was essentially sentimental analysis and as such there's a wealth of research in how to pre-process the data and what algorithms to use. We took the opportunity to use the tried and tested bag of words technique as well as the newer word2vec techniques to not only tokenise the text but to try and allow our models to infer the importance of words and phrases. We also attempted to use some out of the box algorithms form sklearn as well as keras to build our models. But we did try to implement other algorithms that weren't part of those libraries.

We found that performing research on the area of NLP and sentimental analysis particularly interesting and how many individuals and teams across the world are using some classic and newer techniques to tackle a wide range of non-trivial problems. We also found that working with deep neural networks to be challenging and interesting.

The hardest part of the project we found was trying to develop an efficient work-flow. Throughout the project we found that we lost valuable time developing our models, due limitations of our work-flow. For example some of the preprocessing steps could be significantly sped up through the use concurrency, however jupyter notebooks don't quiet support concurrency. Furthermore, we found that we went down several rabbit holes that didn't always yield anything useful in solving this problem.

Ultimately, we found that our model does somewhat match our expectations for the solution to the problem. Do we feel that the model should be used in a general setting? It certainly depends. We feel that it could be used. But there is of course scope to improve the model's accuracy.

## 5.3 Improvement

We found that working on this problem there are several areas that we would aim to improve, that should yield better results. The first area we would look to do is develop a pipeline to enable a more systematic approach to building the networks and perform hyper parameter tuning through the use of the grid search technique. We feel that our CNN and RNN have a far more potential in solving our problem, and that through the use of the grid search technique and more access to a GPU enabled machine we could greatly improve the performance of our models.

While we have a high level overview of how CNN and RNN networks work. We're still scratching the surface of what are the best architectures to solve NLP related problems.

We certainly feel that there are better solutions that could surpass our solution as it is currently. In fact on Kaggle there are plenty of models that are capable of surpassing our models.

# References


[1] Conversation ai by conversationai. https://conversationai.github.io/. (Accessed on 05/02/2018).

[2] Keras documentation. https://keras.io/. (Accessed on 06/11/2018).

[3] Receiver operating characteristic - wikipedia. https://en.wikipedia.org/wiki/Receiver_operating_characteristic#Area_under_the_curve. (Accessed on 05/02/2018).

[4] Toxic comment classification challenge — kaggle. https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge. (Accessed on 05/02/2018).

[5] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 265–283, Berkeley, CA, USA, 2016. USENIX Association. ISBN 978-1-931971-33-1. URL http://dl.acm.org/citation.cfm?id=3026877.3026899.

[6] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[7] MICHAL LEV-RAM. Facebook and youtube's human moderators won't be enough — fortune. http://fortune.com/2018/03/22/human-moderators-facebook-youtube-twitter/, 03 2018. (Accessed on 05/02/2018).

[8] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[9] Alexander Pak and Patrick Paroubek. Twitter as a corpus for sentiment analysis and opinion mining. In *LREc*, volume 10, 2010.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011.

[11] Sebastian Raschka and Vahid Mirjalili. *Python Machine Learning, 2nd Ed.* Packt Publishing, Birmingham, UK, 2 edition, 2017. ISBN 978-1787125933.

[12] Sida Wang and Christopher 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*, pages 90–94. Association for Computational Linguistics, 2012.