# MACHINE LEARNING CAPSTONE REPORT
# Title: Toxic Comment Classification

Ziyao Tang

April 24, 2018

## Contents

# 1 Definition

## 1.1 Project Overview

This project is about online comments classification, which is based on a public Kaggle competition Toxic Comment Classification Challenge. Online discussion is everywhere, however, it can be difficult. According to 2016 Global Report on Online Commenting, there is a trend that high-profile news organizations (including NPR, CNN, The Verge, Toronto Star, Reuters and Popular Science) shutting down comment sections due to the abusive tone and poor quality of comments.

To combat such problem, the Conversation AI team, a research initiative founded by Jigsaw and Google, are working on tools to help improve online conversation. One of the studies focuses on the toxicity of online comments (i.e. comments that are rude, disrespectful or otherwise likely to make someone leave a discussion).

## 1.2 Problem Statement

The current models by the Conversation AI 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).

The goal of this project is to build a multi-headed classification model that's capable of detecting different types of toxicity like threats, obscenity, insults, and identity-based hate. Such improvements to the current model will hopefully help online discussion become more productive and respectful.

## 1.3 Metrics

The models will be evaluated on the mean column-wise ROC AUC. In other words, the score is the average of the individual area under the ROC curves of each predicted column.

ROC stands for Receiver Operating Characteristic (from Signal Detection Theory). It was initially developed for distinguishing noise from not noise in radars. Thus, it is a way of showing the performance of Binary Classifiers.

In binary classification, as shown in Figure 1, the prediction of each instance is often made on a continuous random variable $X$. Given a threshold $T$, the instance is positive if $X > T$, and negative otherwise. $X$ follows a probability density $f_1(x)$ if the instance actually belongs to class "positive", and $f_0(x)$ if otherwise. Thus, the True Positive Rate (TPR) and False

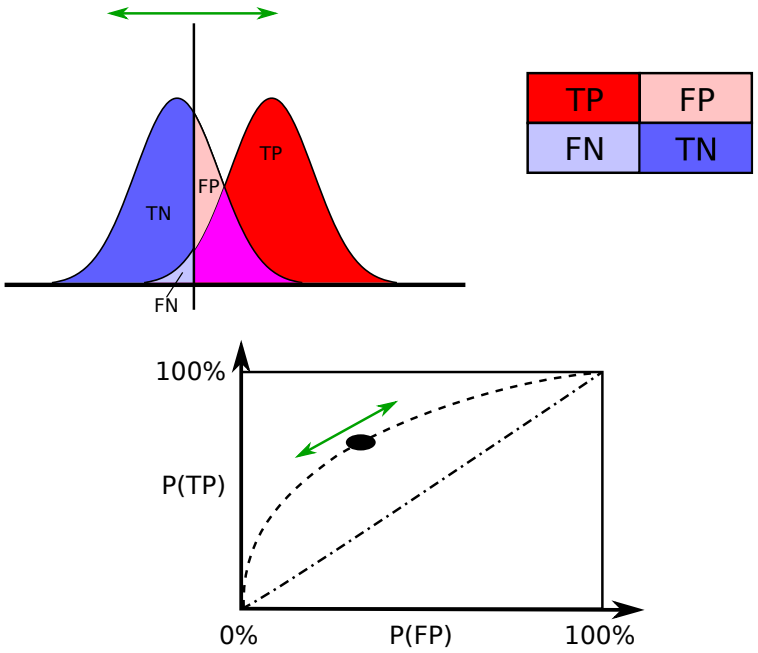


Figure 1: ROC curve

Positive Rate (FPR) are given by

$$TPR = \int_{T}^{\infty} f_1(x)dx$$

$$FPR = \int_{T}^{\infty} f_0(x)dx$$

ROC curve is created by plotting the True Positive Rate vs the False Positive Rate. Thus, the area under the curve can be calculated as

$$ROCAUC = \int_{\infty}^{-\infty} TPR(T)FPR^{'}(T)dT$$

The ROC AUC measures **discrimination**, in this project, the ability to correctly classify the comments with and without certain type of toxicity. The better the classifier, the closer ROC AUC score to 1. One thing to be noticed is that the Keras Library for deep learning does not provide ROC AUC as a metric function. To solve this, a callback function is implemented according to this online discussion. The function run predictions on all of validation data at the end of an epoch, then run the sklearn *roc_auc_score* on the predictions, and display the result.

More information can be found at ROC curve wikipedia and ROC Analysis.

# 2 Analysis

## 2.1 Data Exploration

The dataset contains about 160k human labeled comments from Wikipedia Talk pages. The labeled annotations are obtained by asking 5000 crowd-workers to rate Wikipedia comments according to their toxicity. The paper Ex Machina: Personal Attacks Seen at Scale describes in details the approach to generating a corpus of discussion comments from English Wikipedia.

The dataset has 159571 text records with 6 attributes (1 for toxic, 0 for non-toxic). Among those, the clean comments are 143346, which is about 90% of the total records. After check, there is no missing values in the columns. The types of toxicity are:

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

```
train.head()
```

| | id | comment_text | toxic | severe_toxic | obscene | threat | insult | identity_hate |
|---|---|---|---|---|---|---|---|---|
| **0** | 0000997932d777bf | Explanation\nWhy the edits made under my usern... | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 000103f0d9cfb60f | D'aww! He matches this background colour I'm s... | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 000113f07ec002fd | Hey man, I'm really not trying to edit war. It... | 0 | 0 | 0 | 0 | 0 | 0 |
| **3** | 0001b41b1c6bb37e | "\nMore\nI can't make any real suggestions on ... | 0 | 0 | 0 | 0 | 0 | 0 |
| **4** | 0001d958c54c6e35 | You, sir, are my hero. Any chance you remember... | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 2: the preview of records

Disclaimer: the dataset for this competition contains text that may be considered profane, vulgar, or offensive.

```
The 126104th record of "clean" comments is:
"

You say you ""received his approval"", but for what? To license under GFDL? To license under
a version of Creative Commons? To put in the public domain? We must use one of the standard l
icenses, because they cover all the legal angles, such as commercial use in printed version o
f WP. Just a ""permission to use in WP"" is completely insufficient.  "

The 2197th record of "toxic" comments is:
What the hell is wrong with this thing ? why are my changes not showing?

The 1087th record of "severe_toxic" comments is:
@Qwyrxian suck my dick

The 4812th record of "obscene" comments is:
stupid fucking volcano

The 115th record of "threat" comments is:
Please go and kill yourself. You clearly have nothing better to do on wikipedia than try and
fuck it up for other people. We have no need for your kind.  (talk)

The 5786th record of "insult" comments is:
hey Jingiby you hate Turkic people because you are a racist Slavic Bulgarian... but you hate
your Turkic ancestries... everybody hate you, racist pig!!85.107.110.87

The 793th record of "identity_hate" comments is:
Stop talking to me

Shut the fuck up you stupid cunt
```

Figure 3: the preview of *comment_text*

A *comment_text* sample for each subtype within the corpus is chosen randomly by the program, as shown in Figure 3. Take a moment to look at the text and here is what I see:

- There are obvious typos or spelling mistakes.
- There's punctuation like commas, apostrophes, quotes, question marks, and more.
- There are online usernames and IP address.

## 2.2 Exploratory Visualization

Some exploratory research has been done to better understand the dataset. The researched aspects include Class Imbalance, Multi-tagging, Correlation, Comment Length Distribution, and WordCloud Plot.

**Class Imbalance :** As shown in the Figure 4, about 90% of the records are clean comments.The toxicity is not evenly distributed in different subtypes. Toxic comments due to *obscene* and *insult* are more than those due to *threat* and *identity_hate*. This may create class imbalance problems.

**Multi-tagging :** In Figure 5 , it is shown that some comments have more than one tag. It is reasonable since a comment rated as *severe_toxic* is highly likely be rated as *toxic* at the same time. Other subtypes (eg. *threat*, *obscene*, *insult*, *identity_hate*) could be rated as *toxic* as well.

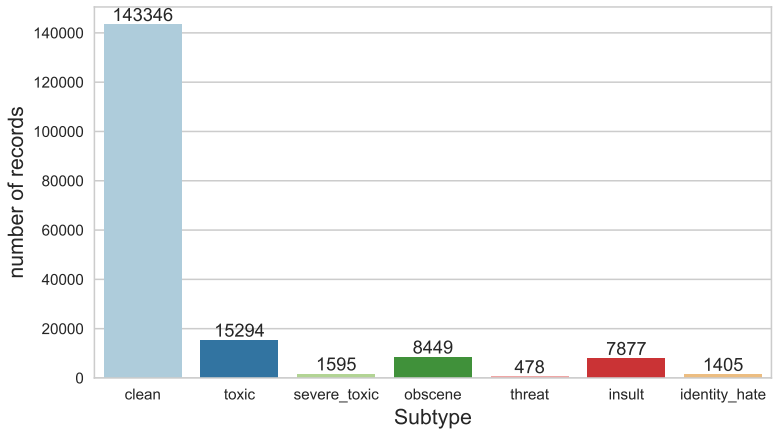


Figure 4: distribution of subtypes of the training set records

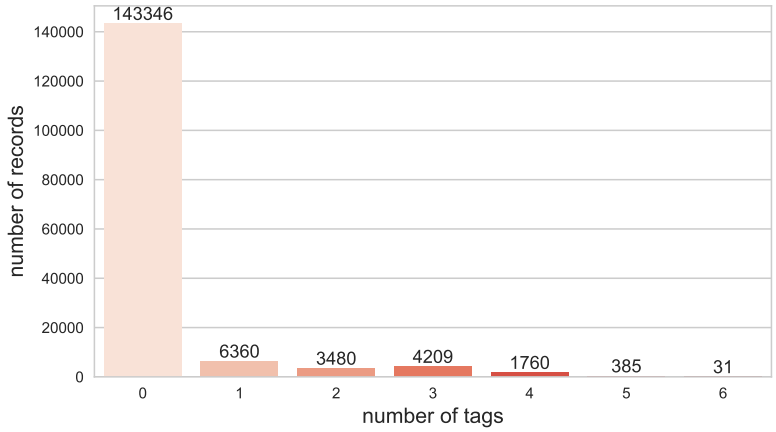


Figure 5: distribution of number of tags each comments has

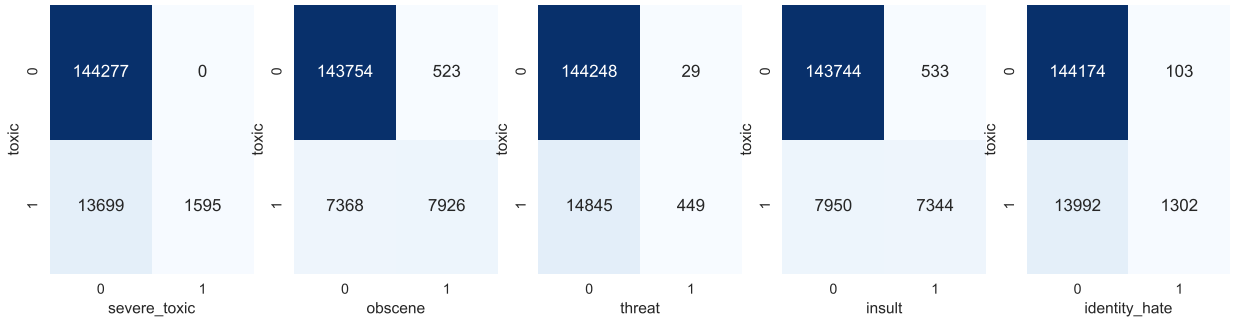


Figure 6: correlation between different subtypes

**Correlation :** Since the target variables are binary, we can apply tools like Confusion Matrix or Crosstab to inspect the correlation between types of toxicity. Figure 6 shows two interesting points. One is that a *severe_toxic* comment is always *toxic*, as we discussed in Multi-tagging section. The other is that other types (*threat*, *obscene*, *insult*, *identity_hate*) seem to be classified as *toxic* for most of them and still with a few exceptions.

**Comments Length Distribution :** Most of the comments are under 400 words. This will be performed again after preprocessing the text data for parameter setting.

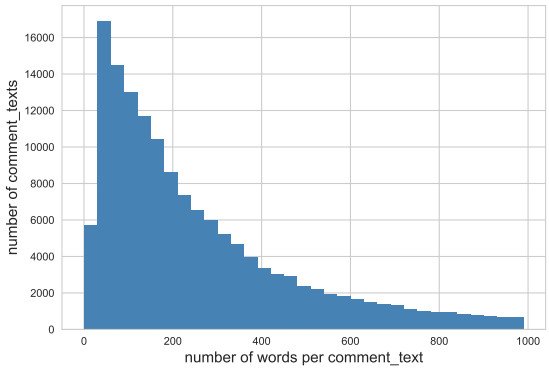


Figure 7: comments length distribution

**Wordclouds - Frequent Words :** In a Wordclouds plot, more frequent words appear bigger. Based on this example, here pictures are applied as masks: checkmark with white background for *clean* comments (as in Figure 8) and crossmark with black background for *toxic* comments (as in Figure 9).
Disclaimer: the text in plots on the next page may be considered profane, vulgar, or offensive..


Figure 8: wordcloud plot for *clean* comments


Figure 9: wordcloud plot for 6 types of *toxic* comments

## 2.3 Algorithms and Techniques

**The Bag-of-Words representation** creates a vocabulary of words and then create a sparse matrix of word counts for the words in the sentence that are present in the dictionary.*sklearn* provides 3 ways of creating Bag-of-Words representation.

- **Count Vectorizer** creates a matrix with frequency counts of each word in the text corpus
- **Hashing Vectorizer** creates a hashmap instead of a dictionary for vocabulary. This enables it to be more scalable and faster for larger text coprus.
- **TF-IDF Vectorizer** Term Frequency (TF) means the count of the words in the text corpus. Inverse Document Frequency (IDF) penalizes frequent words.

TF-IDF term weighting:

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

$$idf(t) = \ln \frac{1+n}{1+df(d,t)} + 1$$

where $n$ is the total number of documents, and $df(d,t)$ is the number of documents that contain term $t$.

In this previous work by Jigsaw on the same dataset, it is confirmed that "*character-level n-grams result in an impressively flexible and performant classifier for a variety of abusive language in English*". Thus, in this project, *n-grams TF-IDF Vectorizer* will be applied as for the benchmark Logistic Regression model.

**Convolutional Neural Network (CNN)** can be used for sentence classification, as described in this paper Convolutional Neural Networks for Sentence Classification. A standard model for document classification is to use an Embedding layer as input, followed by a one-dimensional convolutional neural network, pooling layer, and then a prediction output layer. Kernel size in the convolutional layer defines the number of words to consider as the convolution is passed across the input text document.

**Recurrent Neural Network (RNN)** are especially useful with sequential data because each neuron can use its internal memory to maintain information about the previous input. Since words sequence plays an important part in sentence meaning, this allows the network to gain a deeper understanding of the statement. Theoretically RNNs can handle context from the beginning of the sentence which will allow more accurate predictions of a word at the end. However, RNNs faded out from practice. This can be solved by adding a memory unit that can remember context from the beggining of the input. Two widely used units are Long Short Term Memory (LSTM) and Gated Recurrent Units (GRU).

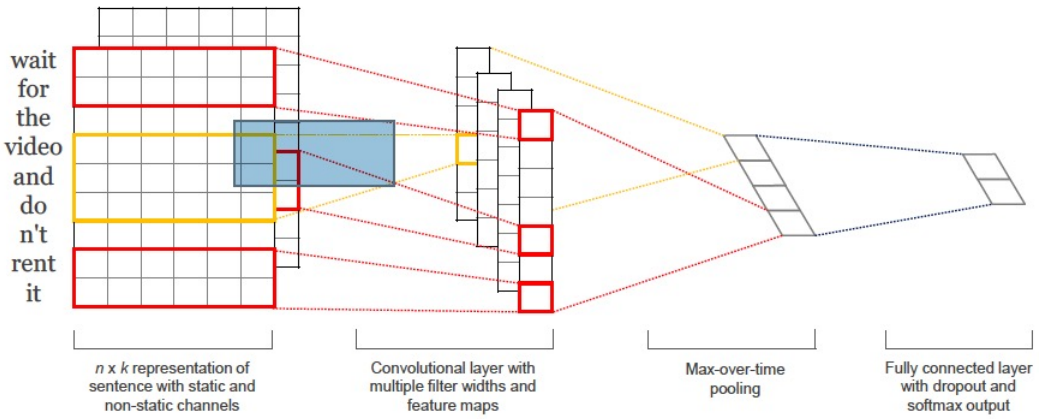


Figure 10: model architecture with two channels for one sentence, cited from *Y. Kim 2014*

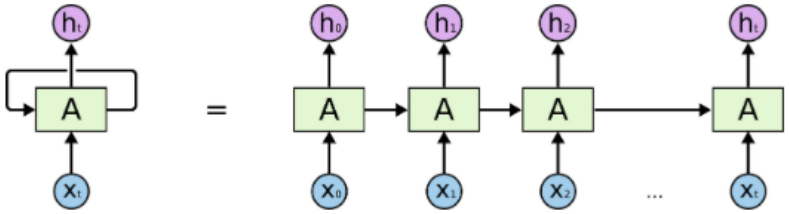


Figure 11: an unrolled recurrent neural network, cited from *colah's blog*

**Transfer Learning** is a machine learning technique where a model trained on one task is re-purposed on a second related task. It is common to perform transfer learning with natural language processing problems. Some available pre-trained models are Google's word2vec Model, Stanford's GloVe Model, and Facebook's FastText.

## 2.4 Benchmark

Since this is a classification problem, a Logistic Regression model with TF-IDF Vectorizer has been trained on the dataset. This yields 0.9707 in the submission leaderboard.

# 3 Methodology

## 3.1 Data Preprocessing

As discussed in section 2.1, a brief scan of the *comment_text* shows that there are:

- obvious typos or spelling mistakes
- punctuation like commas, apostrophes, quotes, question marks, and more
- online usernames and IP address

The following procedures has been implemented to preprocess the text comments:

- split words by white space
- remove punctuation from each string (*#$&*)
- remove non-alphabetic words (0123456789)
- convert words to lower case
- remove stopwords (*we, that, me, my, etc.*)
- remove short words (one letter)
- lemmatization (*better -> good, walking -> walk*)

An example of the preprocessing and summary of the comments length distribution are shown as below:

```
The 0th comment text in trainning set is:
Explanation
Why the edits made under my username Hardcore Metallica Fan were reverted? They weren't vanda
lisms, just closure on some GAs after I voted at New York Dolls FAC. And please don't remove
the template from the talk page since I'm retired now.89.205.38.27


The 0th comment text in trainning set after preprocessing is:
explanation edit make username hardcore metallica fan revert werent vandalisms closure gas vo
te new york dolls fac please dont remove template talk page since im retire
```

Figure 12: a comparison between *comment_text* before and after preprocessing

| | original | preprocessed |
|---|---|---|
| count | 159571 | 159571 |
| mean | 394 | 237 |
| std | 590 | 370 |
| min | 6 | 0 |
| 25% | 96 | 55 |
| 50% | 205 | 121 |
| 75% | 435 | 259 |
| max | 5000 | 5000 |

Table 1: summary of comments length distribution

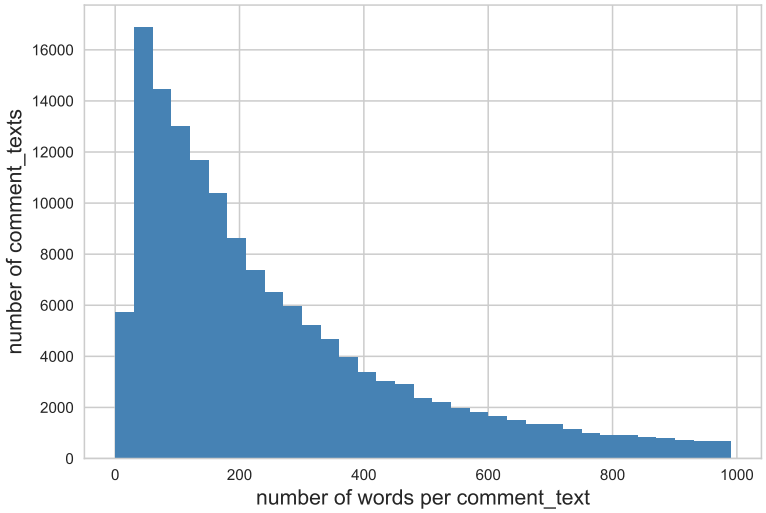


Figure 13: comment length distribution after preprocessing

After the text cleaning process, a *tokenizer* is applied to the cleaned text to encode the text into a sequence of numbers. Since the *comment_text* has different length, *word padding* is used to pad sequence to the same length (*maxlen*). According to Table 1, the 75% of comment length is 259 words. Thus, *maxlen* is set to 250.

## 3.2 Implementation

The classifiers were trained on the preprocessed training data. This was done in Kaggle kernels, and the prediction saved was further evaluated in the Kaggle competition submission board. This can be further divided into the following steps:

1. Load both the training and testing dataset into memory, and preprocess the *comment_text* as described in the previous section.
2. Implement helper functions:
   - *roc_callback()*: creates a callback with ROC AUC score at the end of each epoch.
   - *train_model()*: complies a model first, fits the training dataset, stores the best performance parameters in the checkpointer, loads the model with the best parameters and predicts on the testing dataset.
   - *create_embeddings_index()*: loads the pre-trained model and creates a word embedding dictionary.
   - *create_embeddings_matrix()*: looks up word vectors in the word embedding dictionary and creates a embeddings matrix.
3. Define the network architecture and training parameters. Apply *train_model()* to train the model and save the prediction result for further evaluation.

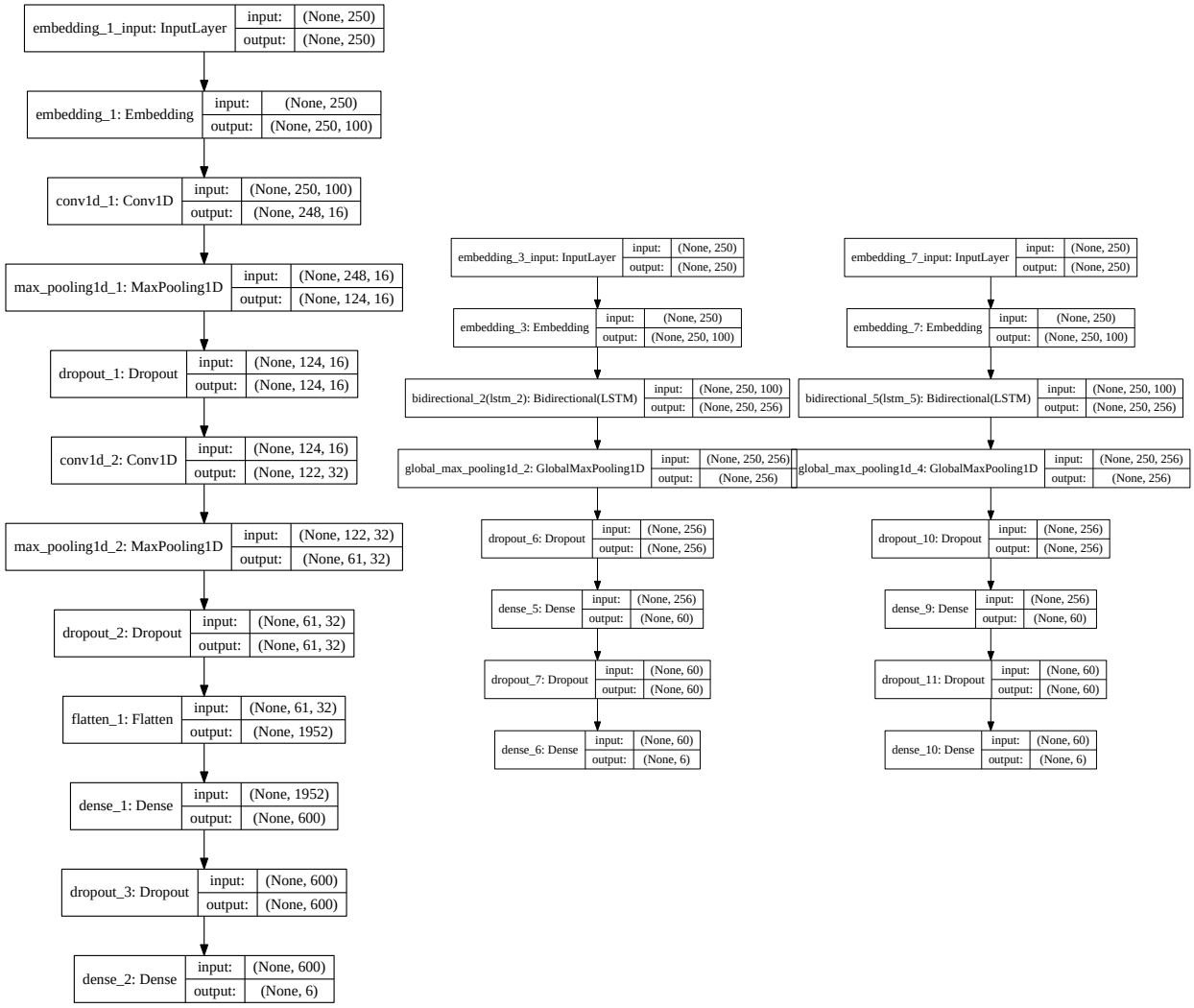


Figure 14: model structure from left to right: CNN, RNN, GloVe

Different model architectures were constructed for training and testing. Here, three of them are selected for discussion and comparison:

- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)
- Transfer Learning as Embedding (GloVe)

## 3.3 Refinement

A basic CNN model with a Convolutional layer, a Dropout layer, and two Dense layers was tested first. By changing parameters one at a time, the results are shown in Figure 15. The dropout rate can be set at 0.25 to get the best performance. The performance does not vary much when tuning the other parameters, such as the size of kernel, the number of filters and the number of nodes in dense layer. Thus, to improve the classifier, it is reasonable to focus

on a good model architecture than parameter tuning.

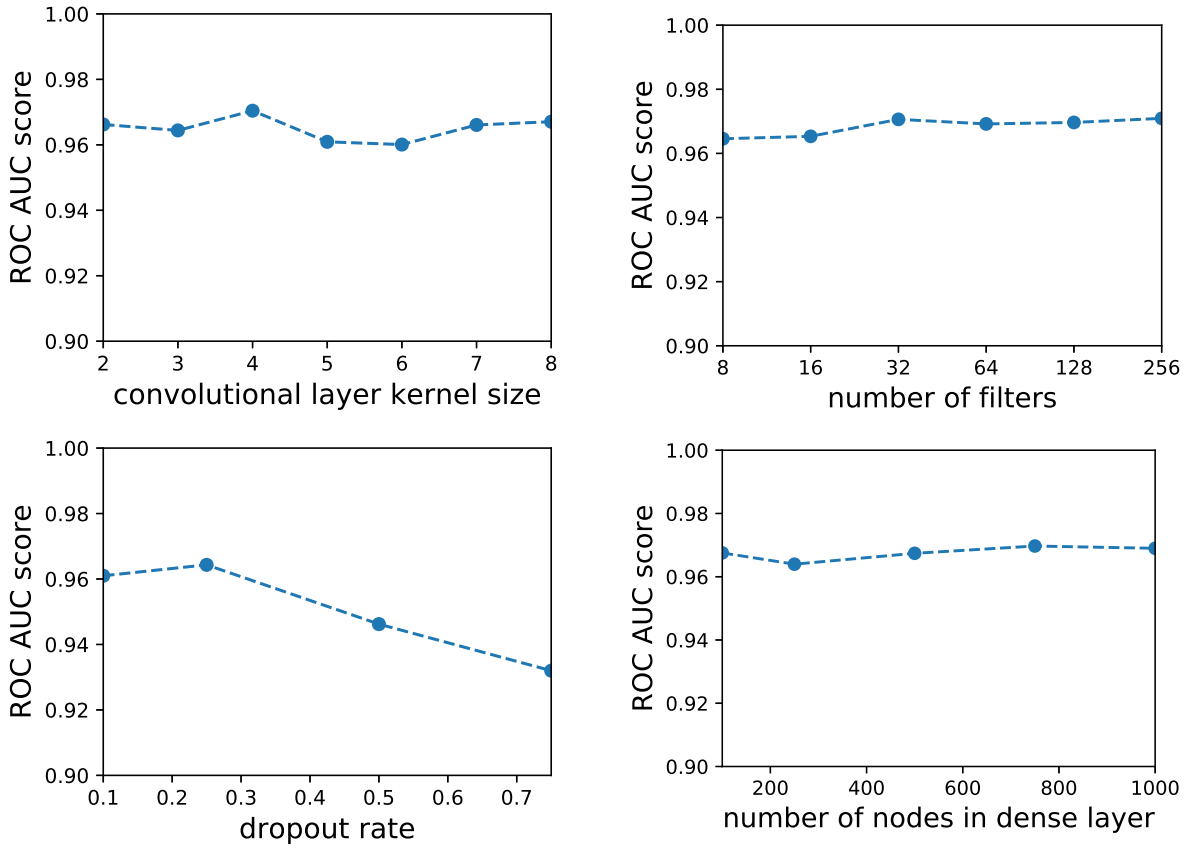


Figure 15: parameters tuning results

# 4 Results

## 4.1 Model Evaluation and Validation

As for each experiment, only the best model was saved along with the weights by applying *modelpointchecker* callback. The best model so far uses GloVe embedding layer with bidirectional LSTM layer.

| Model | ROC AUC training set | ROC AUC validation set | ROC AUC testing set |
|---|---|---|---|
| Logistic Regression | 0.9745 | 0.9745 | 0.9707 |
| CNN | 0.9746 | 0.9426 | 0.9434 |
| RNN (LSTM) | 0.9811 | 0.9777 | 0.9625 |
| GloVe + RNN | 0.9889 | 0.9851 | 0.9756 |

Table 2: summary of model performance

### 4.2 Justification

The Logistic Regression model, chosen as benchmark model, showed surprisingly good result by applying TF-IDF Vectorizer with n-grams. As discussed in section 2.3, the TF-IDF Vectorizer are word frequency scores that try to highlight words that are more interesting. In the comments classification problem, certain key words in a sentence can simply place it into a subtype of toxicity. TF-IDF helps to highlight these keywords. Also, n-grams (here set from 1 word to 6 words) is strongly recommended in the previous work by Jigsaw.

This study shows that RNN (LSTM and GRU) outperforms CNN in toxicity classification. How can this be explained? CNN is good at extracting position invariant features and RNN good at modeling units in sequence. In the paper Comparative Study of CNN and RNN for Natural Language Processing, it suggests that "*which DNN type performs better in text classification task depends on how often the comprehension of global/long-range semantics is required*". As for this project, the toxicity can be either determined by some key words or more often by the long-range semantics, thus RNN performs better.

By adding the GloVe embedding layer, the performance was further improved. Because the word embedding layer was trained with a more reasonable initializer.

## 5 Conclusion

### 5.1 Free-From Visualization

For each experiment, the model accuracy and model loss has been recorded in *history*, as in Figure 16, 17 and 18. The CNN model is quickly overfitted in 2 epochs, and the RNN model overfitted in 2-3 epochs. The model with GloVe embedding performs the best and slightly overfitted at around 7 epochs.

### 5.2 Reflection

As for this project, since most of the model complexity lay in the pre-trained embeddings, minor architecture changes made very little impact on AUC ROC score. Adding dense layers, gaussian vs. spatial dropout, etc. barely changed the overall score of the model. CNN and RNN performance comparison: the order of the words matters. The same phrase with two words swapped can mean completely different things. This meant that CNN approaches were difficult to work with, as they rely on max-pooling as a crutch.

More in general for a machine learning project, studying previous work and literature on similar topic can save a lot of time from trying the wrong directions. These two papers provide great insights in dealing with sentiment classification: Comparative Study of CNN

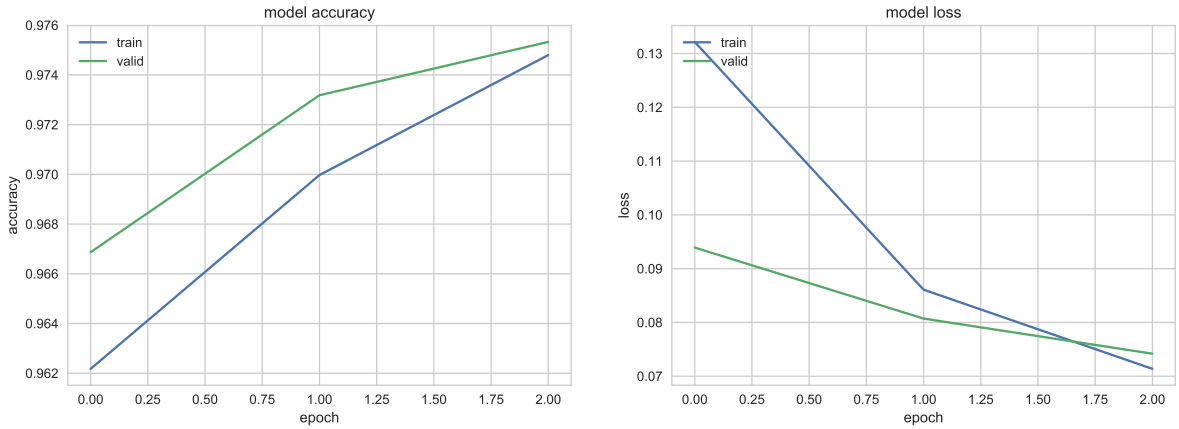


Figure 16: CNN model accuracy and model loss

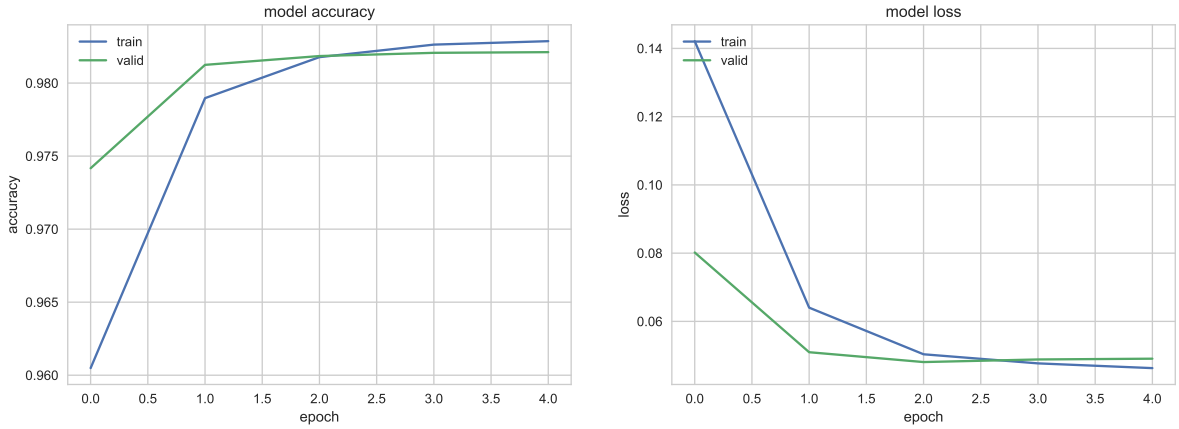


Figure 17: RNN model accuracy and model loss

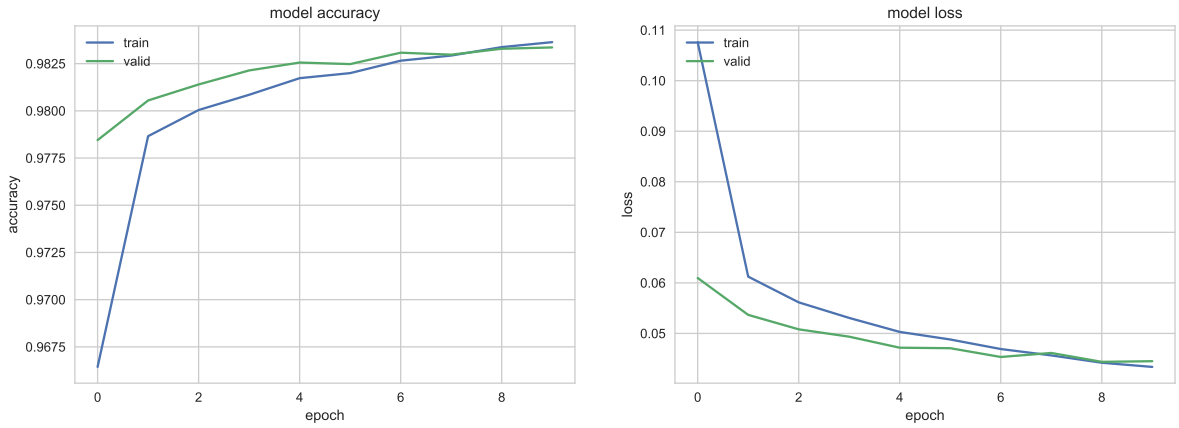


Figure 18: GloVe model accuracy and model loss

and RNN for Natural Language Processing and Ex Machina: Personal Attacks Seen at Scale. Also, the Kaggle online community is extremely welcome and kind in sharing great ideas and useful techniques.

## 5.3 Improvement

Due to the limited time and computational resource, there are some other approaches that can be experimented later for further improvement, such as:

- diverse pre-trained word embeddings (GloVe, word2vec, FastText);
- translations as train/test-time augmentation.