Toxic Comment Classification report

Susan Wang
Department of Computing Science
Simon Fraser University
jyw30@sfu.ca

Muzi Jiang
Department of Computing Science
Simon Fraser University
muzij@sfu.ca

Zewai Cai
Department of Computing Science
Simon Fraser University
zewei_cai@sfu.ca

Luoxi Meng
Department of Computing Science
Simon Fraser University
luoxim@sfu.ca

Abstract

This paper introduces an application of Natural Language Processing techniques to classify text into toxic and non toxic categories. By doing this work, we can create an effective model that can identify toxic comments and identify different types of toxic categories.

Our goal is to implement five models (baseline Support Vector Machine (SVM), Logistic Regression, Multi-Layer Perceptron (MLP), Long Short Term Memory (LSTM), and Convolutional Neural Networks (CNN)) using Python to identify toxic comments and classify the toxic comments as the following six classes: (toxic, severe_toxic, obscene, threat, insult, identity_hate). The dataset is from the Kaggle Toxic Comment Classification Challenge [4], which contains Wikipedia comments that have been labelled by human raters for toxic behaviour. The models were based on a previous research paper from Stanford [1] and our idea is to try to improve the performance of the existing models by tweaking it. We have implemented the baseline SVM model using the built in classifier from the sklearn library. We have found that training the SVM model is very slow and time consuming, so we tried using another built in classifier from sklearn which was the Logistic Regression classifier. We also implemented the Multi-Layer Perceptron model, CNN model and LSTM model using the Keras library. We will evaluate our data based on the ROC AUC curve and log loss for SVM and Logistic Regression. For the other models, we will evaluate the log loss, validation loss, accuracy, and validation accuracy. For all our models, we will evaluate the ROC AUC curve from the Kaggle training dataset. We will talk about our findings in the Experiment section.

1 Prior Related Work

From a previous paper at Stanford [1], models for Support Vector Machine (SVM), Long Short Term Memory (LSTM), Convolutional Neural Network (CNN) and Multilayer Perceptron (MLP) has been implemented by Khieu and Narwall to predict/classify toxic comments using the dataset from Kaggle with the Wikipedia comments. They used Binary Classification to determine whether a comment is toxic or non-toxic and standardized all models to use sparse categorical cross-entropy loss. They used Multi-Label Classification to identify which class of toxic comments a toxic comment belongs to and standardized across all models to use binary cross entropy loss. For word level analysis, they found that LSTM (using 3 layers and 32 output units for each layer) achieved the best score for the Binary Classification task with a 0.889 test accuracy, 0.925 precision, 0.850 recall, and highest F1 score of 0.886. For the character level analysis, they found that CNN (with 64 output units, kernel size 3, 100 character clipping) achieved the best score for the Binary Classification task with a 0.803 label accuracy, 0.805 precision, 0.796 recall, and 0.800 F1 score. However, the models for the character level analysis for the Binary Classification task did not perform as well as the models for the word level analysis. For the Multi-Label Classification, the LSTM model (using 3 layers and 32 output units for each layer) achieved the best score with a label accuracy of 0.927, a sentence accuracy of 0.735, a precision of 0.733, a recall of 0.681, and a F1 score of 0.706. In general, the LSTM is the best model for both Binary Classification (word level analysis) and Multi-Label classification. The CNN may have the best performance for the character level analysis for Binary Classification, but had worse scores compared to all the models used for word level analysis for the Binary Classification task.

2 Introduction

Nowadays, social networking and online communities are revolutionizing the way that people exchange information. However, toxic and abusive comments are becoming immensely prevalent in social networking websites and online communities. The toxic comments are sometimes used as a personal attack. Many people will give up participation in an online discussion to avoid harassment from those who do not share their opinions.

Suppose a social networking platform wants to create a healthy communication environment. In that case, there is a challenge to use an identification and prevention system to identify toxic comments to improve the online conversation environment. This project is going to analyze models that detect different types of malicious online behavior.

Our goal is to try to analyze the models and look at how we can improve the accuracy for classifying toxic comments. A model that can accurately determine if a comment is toxic will be important, so that a social networking site can correctly decide what comments should be removed or what other action to take when a toxic comment is posted. Certain types of toxic comments are more serious than others (e.g. violent threats may be more severe than insults), so a model will be needed to detect the types of toxic comments accurately so that social networking sites can decide what actions to take when considering a toxic comment and the type of toxic comment it is. For example, if a user is posting a lot of violent threats, the model should be able to detect that so a social networking site can decide if the user should be reported to the authorities. We need to look at what factors improve the accuracy for this type of classification problem.

3 Approach

3.1 Task

3.1.1 Binary Classification

Our binary classification task trains a model for each of the labels. We combine the results from all the classifiers and output the probabilities that each label is true predicted by each classifier. This approach is simple and easy to understand. The shortcoming lies in that the correlations between the labels are not used. But it does simplify the task and decompose it to several binary classification tasks. This is applied to both the SVM and the Logistic Regression models.

3.1.2 Multi-label Classification

The multi-label classification task feeds all the data to the training phase. Thus, it is complete and relatively complex. Note there might be an item labeled with more than one label. For example, an item may be labeled with both “severe_toxic” and “obscene”. This is applied to Multi-layer Perceptrons, LSTM, and CNN.

3.2 Support Vector Machine (Baseline)

We take our implementation of SVM with sklearn library as our baseline approach. The input features are TfIdf vectors transformed from the train and test dataset. We fit the TfidfVectorizer with the train dataset and use this model to transform the test dataset to obtain the Tfidf vectors. And then we feed the transformed vectors to SVC from the sklearn library. Note that this model follows the specifications we have above on the “Binary Classification” session, and the output predicted probabilities is a combined result from the output of the classifier for each label. We take SVM as our baseline approach and compute its ROC AUC score.

3.3 Logistic Regression

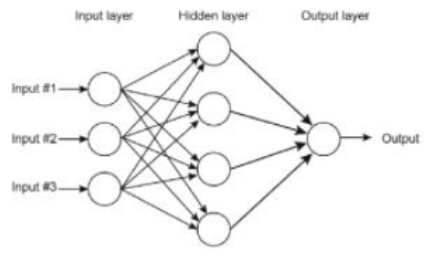
The training phase of SVM from sklearn library is time-consuming and the result is not satisfiable. Thus, we consider using Logistic Regression to provide a better result under the binary classification task. The input is still Tfidf vectors and we do some preprocessing work to make sure the model will converge in finite steps. Then we evaluate our model with cross validation and compute log loss. We also got the ROC AUC score of our model.

3.4 Neural network models

As mentioned above, we implemented three different neural networks in this project: Multilayer Perceptron, Long Short Term Memory, Convolutional Neural Network. We used Python’s deep learning API, Keras to implement these models. At the beginning, we converted the comment texts to word embeddings by GloVe word vectors and used the obtained word embeddings as the input of networks [2]. The output of the neural networks are probabilities of 6 classes.

3.4.1 Multilayer Perceptron

In this model, we used Tokenizer to do tokenization and then applied Dense layers from Keras to implement MLP. This network just used Dense layers, which are fully-connected layers. The network used ReLU activations between each hidden layer, and finally a sigmoid activation function would be used over the final output. The network would output 6 probabilities which corresponds to 6 classes. This model used Adam optimizer with 0.0005 learning rate and used binary cross entropy loss over labels.

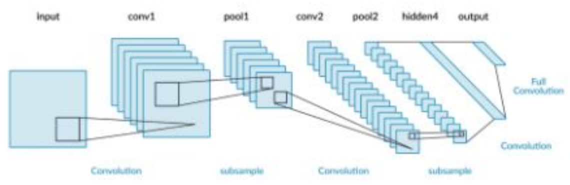


3.4.2 Long Short Term Memory

The implementation difference between MLP and LSTM is the structure of networks. In this model, two more useful layers are used, which are dropout layers and bidirectional LSTM layers. The dropout layer is used to select a few neurons rather than all neurons from the previous layer. Bidirectional LSTM is the core layer in this model. ReLU activations are used between hidden layers and a sigmoid activation is used over the final output. This model also used Adam optimizer with a 0.0005 learning rate and used binary cross entropy loss over labels.

3.4.3 Convolutional Neural Network

The last model is CNN, which usually used to process image data. In this model, we treat our comment texts as image data and do convolution operation. In this model, Conv1D layers and MaxPooling1D layers are added. ReLU activations are used between convolution layers and max pooling layers. A sigmoid activation is used over the final output. This model also used Adam optimizer with a 0.0005 learning rate and used binary cross entropy loss over labels.



4 Experiment

4.1 Data

Our group analyses the Kaggle Toxic Comment Classification Challenge dataset. It includes 159,517 annotated user comments collected from Wikipedia user talk pages. The comments labelled as toxic are divided by 6 classes (toxic, severe_-toxic, obscene, threat, insult, identity_hate). Comments can be associated with multiple classes. It frames the task as a multi-label classification problem. We use a csv file for the training data to train our model to identify types of toxic comments and create a csv file for the testing data to predict the type of toxic comments, given the comments.
Our data processing was done using Python. For the binary classification task, we train each label in the classes and output the probability for the predicted class.

4.2 Implementation

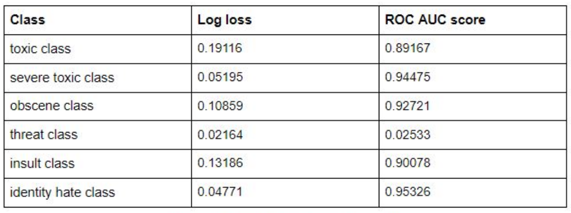
The models that we have implemented were the baseline Support Vector Machine (SVM) using the pre-built SVC classifier from the sklearn library, the Logistic Regression model that was also using a pre-built classifier from the sklearn library, and the Multi-Layer Perceptron, LSTM, and Concurrent Neural Network (CNN) models using the Keras library.
The code we used for constructing an embedding matrix, which is also input to the neural network, was a revised version of the code from https://stackoverflow.com/questions/37793118 /load-pretrained-glove-vectors-in-python. [2]
The code we applied for plotting the figures, which showed the comparison and tendency of scores during the training phase was based on the code from https://towardsdatascience.com/ machine-learning-recurrent-neural-networks-and-long-short-term-memory-lstm-python-keras-example-86001ceaaebc. [3]

4.3 Evaluation

We evaluate our implementation on Support Vector Machine (SVM) and Logistic Regression with log loss scores and ROC AUC scores. For the three neural networks (MLP, LSTM, and CNN), we evaluate them with log loss and accuracy scores for our training data. For the test data from the Kaggle competition, we will evaluate all the models with ROC AUC scores.

4.3.1 Baseline: Support Vector Machine (SVM)

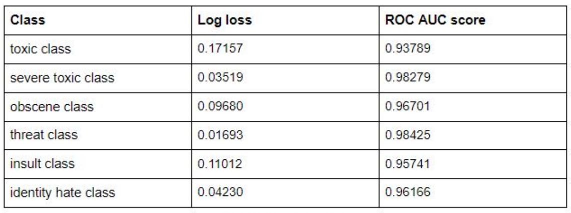
We used Tfidf vectors as our training/test input for SVM and Logistic Regression. For the models using SVM, we applied binary classification for each label. We transformed the Tfidf vectors using TruncatedSVD where n_components = 20. For the Logistic Regression, we applied TruncatedSVD with n_components=100 and n_iter=100 and the MaxAbsScaler as our transformation.



Class	Log loss	ROC AUC score
toxic class	0.19116	0.89167
severe toxic class	0.05195	0.94475
obscene class	0.10859	0.92721
threat class	0.02164	0.02533
insult class	0.13186	0.90078
identity hate class	0.04771	0.95326

Figure 1: SVC Results (imbalanced)

Figure 1 shows the scores for SVM for each class of toxic comments. Most of the comments have a high score over 89%, but the threat class is extremely low (0.02533). It could be due to the reason that we don't have enough training data for the threat class. We got a ROC AUC score of 0.73786 in Kaggle (test dataset) for our first attempt. The major differences between the scores from Kaggle and our validation set could be a result of overfitting.
To solve the problem of overfitting, we tried to balance the weight of each class. By setting class_weight='balanced' when calling the SVC sklearn classifier. This mode will automatically adjust weights inversely proportional to class frequencies in the input data as n_samples / (n_classes * np.bincount(y)).



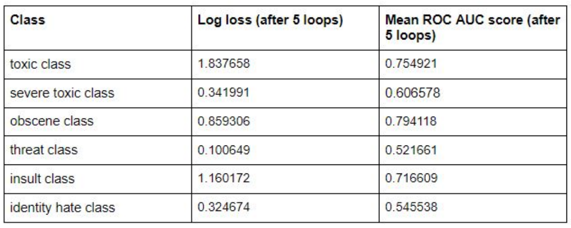
Class	Log loss	ROC AUC score
toxic class	0.17157	0.93789
severe toxic class	0.03519	0.98279
obscene class	0.09680	0.96701
threat class	0.01693	0.98425
insult class	0.11012	0.95741
identity hate class	0.04230	0.96166

Figure 2: SVC Results (balanced)

Figure 2 shows that after balancing the weight of each class, the log loss scores drop and the ROC AUC scores increase, especially the "insult class", whose ROC AUC score goes up to 0.98425. The ROC AUC Kaggle score for this balanced version of SVM is 0.91595 (on test dataset)
From this improvement, we found that balancing the weight of class is an efficient approach to solve the problem of overfitting on a specific class (with few training data).
By the implementation of our baseline approach, we find that 1) balanced SVC is already a good classifier for solving this problem, which is simple and effective, 2) we have to apply more techniques to improve the performance of advanced deep learning methods, in order to outperform this baseline approach.

4.3.2 Logistic Regression

Although the performance of Support Vector Machine (SVM) is good enough as a baseline approach, it is quite time-consuming. To deal with this issue, we also turn to Logistic Regression in order to find a simple faster solution to this problem.



Class	Log loss (after 5 loops)	Mean ROC AUC score (after 5 loops)
toxic class	1.837658	0.754921
severe toxic class	0.341991	0.606578
obscene class	0.859306	0.794118
threat class	0.100649	0.521661
insult class	1.160172	0.716609
identity hate class	0.324674	0.545538

Figure 3: Logistic Regression Results

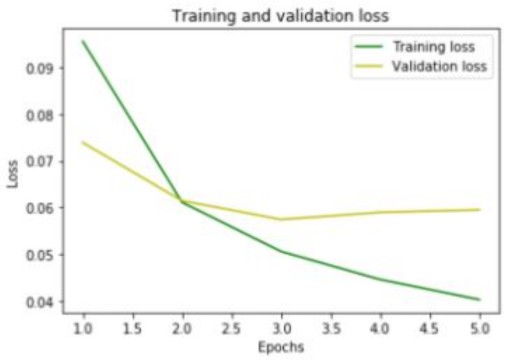
Figure 3 shows the scores for ROC AUC for each of the toxic comments for our Logistic Regression model. We implemented cross validation to evaluate the models. Our ROC AUC score is generally lower than the score from the SVM results because of cross validation. But when we submitted our predicted results to the Kaggle competition's test set, we got a ROC AUC score of 0.94415. It is likely that for the SVM model, we have overfitted our data since the score for our given data was significantly higher than the unseen data from the Kaggle competition.

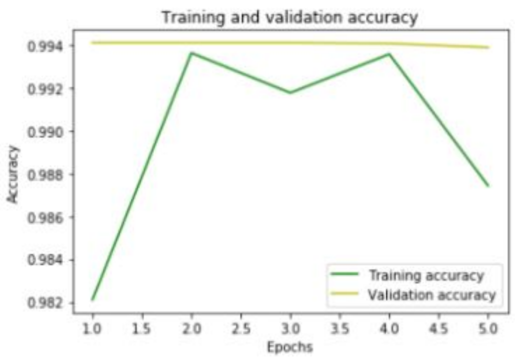
4.3.3 Multi-Layer Perceptron

For MLP, we tried different number of Dense layers, different number of epochs, different learning rates and different optimizers. Now, we used three dense layers of 256 units with ReLU activation and a final dense layer of 6 units with Sigmoid activation. The Multi-Layer Perceptron has a batch size of 32, a validation split set to 20% of the training data, and was trained over 5 epochs. We got a ROC AUC score of 0.96942 in Kaggle.

	loss	accuracy	val_loss	val_accuracy
Epoch 1	0.0955	0.9821	0.0739	0.9941
Epoch 2	0.0611	0.9936	0.0615	0.9941
Epoch 3	0.0506	0.9918	0.0574	0.9941
Epoch 4	0.0446	0.9936	0.0590	0.9941
Epoch 5	0.0404	0.9874	0.0595	0.9939

Figure 4: Results of MLP





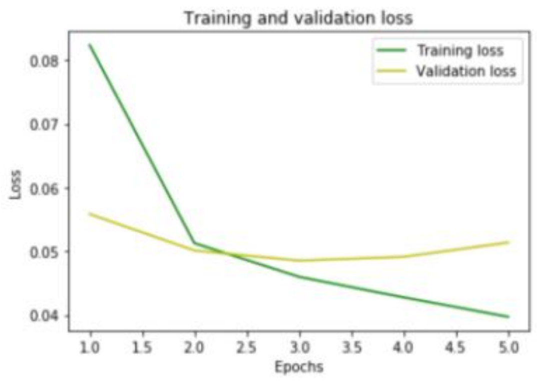
As we can see, training loss decreases over each epoch. However, the training accuracy curve fluctuates. We think it's reasonable because the changes are small, which is acceptable. The potential reason could be caused by the fact that after 1 epoch, the network nearly converges and the learning rate is still high in this case. Thus, the curve would fluctuate.

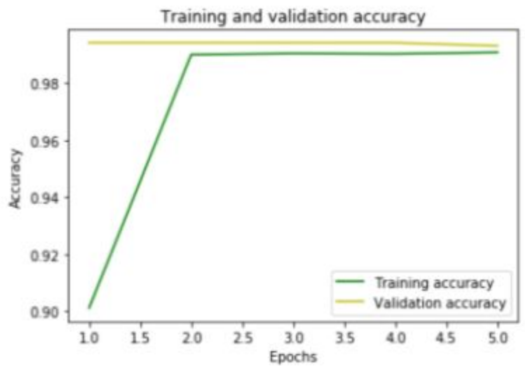
4.3.4 Long Short Term Memory

For LSTM, we tried different number of bidirectional LSTM layers, different number of epochs, different learning rates and different optimizers. Now, we used one bidirectional LSTM layer of 100 units, a global 1D max pool layer, and a dense layer with ReLU activation and a dropout layer. Finally, a dense layer of 6 units with Sigmoid activation is used. LSTM has a batch size of 32, a validation split set to 20% of the training data, and was trained over 5 epochs. We got a ROC AUC score of 0.97206 in Kaggle.

	loss	accuracy	val_loss	val_accuracy
Epoch 1	0.0824	0.9011	0.0588	0.9941
Epoch 2	0.0513	0.9900	0.0501	0.9941
Epoch 3	0.0460	0.9904	0.0485	0.9941
Epoch 4	0.0428	0.9902	0.0491	0.9941
Epoch 5	0.0397	0.9908	0.0514	0.9930

Figure 5: Results of LSTM

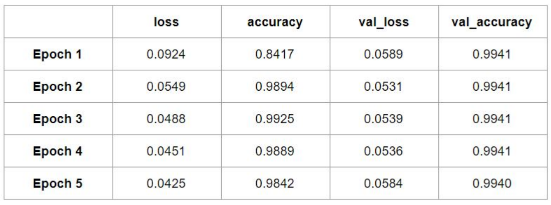




As we can see, the training loss decreases and training accuracy increases with increasing epochs (converging at epoch 2). Validation accuracy remained consistent with slight fluctuations for each epoch.

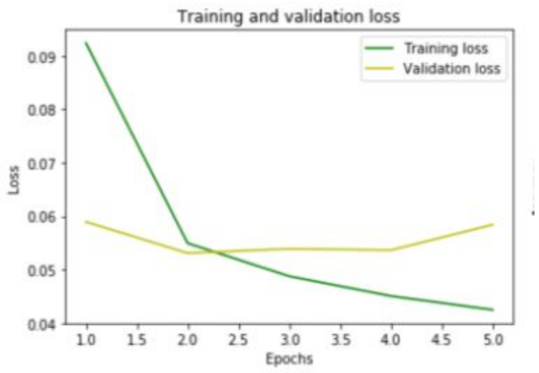
4.3.5 Convolutional Neural Network

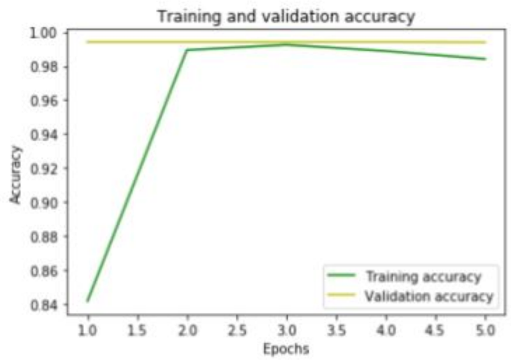
For CNN, we tried different number of convolutional layers and max pooling layers, different number of epochs, different learning rates and different optimizers. Now, we used 3 convolutional layers and 3 max pooling layers (1 global max pool layer). Finally, 2 dense layers follow. Some dropout layers are added between hidden layers with p=0.2. CNN has a batch size of 32, a validation split set to 20% of the training data, and was trained over 5 epochs. We got a ROC AUC score of 0.96000 in Kaggle.



	loss	accuracy	val_loss	val_accuracy
Epoch 1	0.0924	0.8417	0.0589	0.9941
Epoch 2	0.0549	0.9894	0.0531	0.9941
Epoch 3	0.0488	0.9925	0.0539	0.9941
Epoch 4	0.0451	0.9889	0.0536	0.9941
Epoch 5	0.0425	0.9842	0.0584	0.9940

Figure 6: Results of CNN



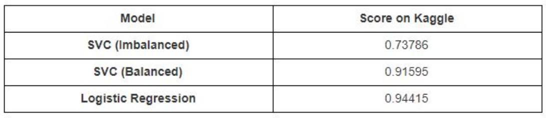


As we can see, the training loss decreases and training accuracy increases with increasing epochs, but eventually converges and slightly fluctuates at epoch 2.

4.4 Methods being compared

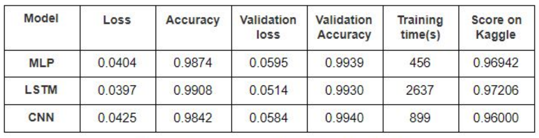
For SVM and Logistic Regression, we looked at the ROC AUC curve we got from our own training data for each of the 6 classes of toxic comments and the overall test dataset from Kaggle. We also looked at the log loss as well for each class from our training data as well.
For our neural network models (CNN, LSTM, MLP), we compared the loss, validation loss, accuracy, and validation accuracy we have seen on our training set for each epoch. From the test dataset on Kaggle, we also evaluated with the ROC AUC score.

4.5 Result



Model	Score on Kaggle
SVC (Imbalanced)	0.73786
SVC (Balanced)	0.91595
Logistic Regression	0.94415

Figure 7: Comparison of SVC and Logistic Regression



Model	Loss	Accuracy	Validation loss	Validation Accuracy	Training time(s)	Score on Kaggle
MLP	0.0404	0.9874	0.0595	0.9939	456	0.96942
LSTM	0.0397	0.9908	0.0514	0.9930	2637	0.97206
CNN	0.0425	0.9842	0.0584	0.9940	899	0.96000

Figure 8: Comparison of MLP, LSTM and CNN

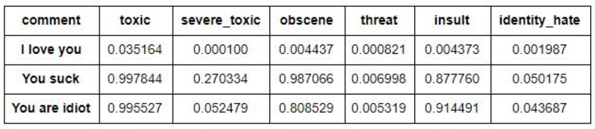
(Based on performances of three neural networks after 5 epochs)

4.5.1 Overall Quantitative Results

Figure 7 shows that the score of our baseline approach SVM is improved from 0.73786 to 0.91595 by means of balancing the weight of each class. For example, only a few data is labeled as "threat" in the training set and thus we tend to overfit this class. By rebalancing the weight, the problem is solved. Moreover, Logistic Regression gives a higher result than SVM. It is because training a Logistic Regression model is much more efficient than training a SVM model, which facilitates us to tune the parameters better.
Figure 8 shows that the performance of LSTM is the best, which has higher accuracy and lower loss and also performs best on test data as the ROC AUC score from the Kaggle test set shows. To improve the performance of networks, we tried to tune the number of dense layers in MLP, the number of bidirectional LSTM layers in LSTM, and the number of convolutional layers in CNN. Finally, we got the best performance of each model. But it's hard to tell which neural network is better, because it really depends on the structure of networks. Adding one layer or removing one layer may influence the performance of networks a lot.

4.5.2 Overall Qualitative Evaluation

The best performing model in our project is LSTM. We created some comment samples and tested the results. Figure 9 shows a table with comments and the probability that each comment belongs to each of the 6 classes.



comment	toxic	severe_toxic	obscene	threat	insult	identity_hate
I love you	0.035164	0.000100	0.004437	0.000821	0.004373	0.001987
You suck	0.997844	0.270334	0.987066	0.006998	0.877760	0.050175
You are idiot	0.995527	0.052479	0.808529	0.005319	0.914491	0.043687

Figure 9: Qualitative Evaluation of LSTM model

From the qualitative results, we can see the prediction is accurate. For example, "I love you" is a positive comment, which should not be labeled as any of 6 labels, and our prediction shows low probabilities on each label. However, for the other 2 negative comments, they got high probabilities on toxic, obscene and insult labels, which is reasonable. Therefore, we think our models are useful enough in real-world applications.

5 Conclusion

We have learned a lot from our project. To begin with, we better have a better insight of the work load that could be done. Our initial plan was to implement both word-level and character-level models. However, due to multiple reasons, we did not have time to implement the character-level model. Besides, we realized that neural networks are very useful in machine learning, because there is no limit to the number of input and output, while basically, some basic classifiers only predict on one label, which means one output. However, usually neural networks would take a longer time for training, because there are too many parameters to be learned. And how to design a very useful network in a specific problem is a challenge. For us, we would constantly try out different network structures, and then filter networks with good performance on validation data.

Some problems still remained in our project and they could be fixed. Because we have to compare the performances under the same hyperparameters, some results would not look perfect. For example, under small epochs, the curves of loss and accuracy fluctuate. We believe that it is usually caused by the learning rate. The solution could be setting a relatively proper initial learning rate and then using a scheduler to decay the learning rate after certain epochs. The reason behind this is that with training going, the results would converge, and if the learning rate is still high, it would be hard to obtain better parameters. In the future, we would like to implement a character-level version of each deep learning model. We will test on the difference of performances between word-level and character-level versions of each model and comment on which implementation outperforms the other. We would also want to implement a binary prediction task on comments. To be specific, we can classify each comment as toxic or non-toxic, which is also useful in real-world applications.

References

[1] Kevin Khieu and Neha Narwel. "CS224N: Detecting and Classifying Toxic Comments". In: (), pp. 1–8. URL: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/reports/6837517.pdf.

[2] *Load Glove*. URL: https://stackoverflow.com/questions/37793118/load-pretrained-glove-vectors-in-python.

[3] *Towards Data Science*. URL: https://towardsdatascience.com/machine-learning-recurrent-neural-networks-and-long-short-term-memory-lstm-python-keras-example-86001ceaaebc.

[4] *Toxic Comment Classification Challenge*. URL: https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data.