Toxic Comment Classification by Regression and Neural Networks

CSE802 Project Proposal

Zhuangdi Zhu Wei Wang Zhiming Jiang
Department of computer Science and Engineering
Michigan State University
East Lansing, Michigan, USA
{zhuzhuan, wangwe90, jiangzh7}@msu.edu

***Abstract*— In this project, we proposed approaches to automatically recognize toxic comments on social networks. Digital abuse is a critical issue worldwide. The approach of manually rating toxic comments works with a huge cost. Therefore, we propose classification models that can automatically detect different types of toxicity like threats, obscenity, insults, etc., using pattern recognition approaches. We use logistic regression as well as neural networks techniques to build different models and conducted evaluations to compare the performance of them.**

Keywords*— neural networks; natural language processing, regression;

I. INTRODUCTION

1) Motivation

In this project, we aim to provide a tool which recognizes toxic comments presented on social networks. Digital abuse is a critical issue worldwide. Due to the prevalence of social platforms such as YouTube and Twitter, people have more channels to express their opinions on the internet, for better or for worse. On one hand, the anonymity feature of the network leave people less constrained when making comments. Some people may make reckless comments without taking any consequences. On the other hand, some people may give inappropriate comments even without awareness because of the culture difference or misunderstanding.

The abuse of toxic comments on the internet can be harmful to both individuals and the society as a whole. For individuals, the threat of abuse and harassment online means that many people stop expressing themselves and give up on seeking different opinions. Victims of severe digital abuse may even end up with depression, taking the loss of reputation or privacy. For companies, they have to struggle to effectively facilitate conversations, leading many communities to limit or completely shut down user comments.

Depriving users' rights of expressing comments is not the right solution to digital abuse. Many online platforms and communities are hiring labors to manually rate toxic comments in order to take reactions. However, the manual rating approach works at large cost. First, manual detection of toxic comments is time consuming and non-efficient. Besides time, money spent on paying toxic comment raters is also a huge expenditure. Moreover, being exposed to toxic comments hours and days leave human raters under big pressure. All of these issues can be addressed if we build a detection model which automatically detects toxic comment.

2) Literature Review

Our work belongs to the field of sentiment analysis, which can be considered as an integration of natural language processing and pattern recognition. Much work can be found along these two lines [8-25]. In this project, we focus on the approaches used for feature extraction and model training.

a) Regression approaches for sentiment analysis

Logistic regression is widely used on natural language processing and has a strong support of theory and algorithm. For example, in topic classification, text categorization, detection of spam pages [13]. Logistic regression measures the relationship between an output variable Y(category) and one or more independent variables, which are usually continuous, by using probability scores as the predicted values of the dependent variable.

Recent studies show that for a wide range of text classification tasks and corpora, logistic regression achieves outstanding performance. In Zhang and Oles's 2001 paper [12], they applied logistic regression on text categorization problem and compared with other linear classification methods such as support vector machines, which showed an excellent performance on most of the experiments. In Li and Yang 2003 paper [11], they optimized loss function and model complexity for a set of algorithms include logistic regression and applied modified approaches on text categorization problem. The performance of logistic regression is competitive to many latest techniques.

Many efforts also have been made on speeding up training process. A large-scale Bayesian logistic regression approach was proposed on 2007 by Alexander and Lewis [19]. 4-gram hashed byte are used as features to train logistic regression classifier mainly on news articles and showed a great success on text classification problem of large dataset. In 2012, Ke and Meng, et al proposed LightGBM [18], a highly efficient gradient boosting decision tree algorithm, which can achieve an even better training speed on logistic regression with large dataset.

b) Deep learning approaches for sentiment analysis

With the blossom of large amount of corpus available on the internet, deep learning approaches using neural networks have earned increasing popularity the field of Sentiment. Researchers from Stanford University have proposed complex neural network models to conduct sentence-level sentiment analysis. An example is the Matrix-Vector Recursive Neural Network (MV-RNN), proposed by Socher *et al* [8]. This model takes input a parse tree of a sentence and generates the sentiment category of the root node of the parse tree by recursively building it in a bottom-up fashion.

Chung *et.al* compares different types of recurrent units in recurrent neural networks [9]. They found that more sophisticated units with a gating mechanism, such as a long short-term memory (LSTM) unit and a recently proposed gated recurrent unit (GRU) performs better when modeling long sequence data. Based on traditional GRU, Zhou et. al proposed a novel recurrent cell called Minimal Gate Unit (MGU) [10]. All of the above work inspires us to leverage recurrent neural network to build a comment classifier.

3) Problem Statement

In this project, we aim to build a multi-headed model that is capable of detecting different types of toxicity like threats, obscenity, insults, and identity-based. Especially, we abstract the problem as a multi-label classification problem: given a text x, we want to predict the probability that this text belongs to certain toxicity types: $p = P(x|D)$.

Here, x is a text composed of English words;p is a probability vector of dimension d, as we have d possible toxicity types; the value in each dimension of p indicates the likelihood that this text belongs to a certain toxicity type. We are given a dataset of D which is a collection of labeled texts. For each $t \in D$, we have a label y, which is a binary vector of dimension d.

II. METHEDOLOGIES

This project involves four stepwise: data collection & pre-processing, feature extraction, model training, and the performance evaluation.

A. Data preprocessing

The dataset we use is from Wikipedia's talk page edits. It contains a large number of Wikipedia comments which have been labeled by human raters for toxic behavior. The dataset is available online [1]. It contains around 160, 000 samples, each sample being a comment written by English. A sample is associated with six types of toxicity:

$$1.toxic;\ 2.severe-toxic\ \ 3.obscene;$$

$$4.threat;\ 5.insult;\ 6.identity-hate$$

Each toxicity type is labeled as either 1 or 0. It is worth noting that a comment can be labeled with more than one toxicity type. We take two steps to pre-process the dataset. First, for each text in the dataset, we clean it by removing some stopping words, special characters and punctuations. Stopping words are frequently used words such as *the, an, a, will, shall*, etc. Next, we tokenize each text into a list of words, so that we can get the vocabulary V of the whole dataset, where each word w in the vocabulary is a one-bit-hot vector: $w = [w_1, w_2, \dots, w_{|V|}]^t$, so that only one bit in w is one, and the dimension of this vector is vocabulary size.

Before we extract features from this dataset, we conducted a word-frequency analysis. We separate the training dataset into two groups: the neutral subset and the negative subset. The neutral subset contains all texts that are not labeled as any toxicity type, while negative subset contains texts that are labeled as at least one toxicity type. We use a graphical tool [2] to show the visualization of the neutral group in Figure 1. The visualization of the negative group is not shown because it may bring discomfort to our readers.

Fig. 1. Visualization of the netural dataset

B. *Feature Extraction*

In this section, we introduce three different approaches used for extracting features: count vectorization and TF-IDF vectorization approaches for training a regression model, and word embedding approaches for training neural network models.

1) *Count Vectorization*

Count vectorization is a procedure that calculates the occurrence(frequency) of a particular token appeared in one document normalized by all tokens across all documents. Token is learned when taking all documents into account. Each individual token occurrence is treated as a feature; the vector containing all token frequencies for a given document is used to train classifier model.

2) *Tf-idf Vectorization*

Tf-idf (term frequency-inverse document frequency) [4], is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. The tf-idf weight is often used in information retrieval and text mining. The importance of a word is related with the appearance and the frequency in the document. Tf-idf is often used by searching engines in scoring and ranking a document's relevance given a user query [5].

The tf-idf weight contains two parts, the normalized term frequency and the inverse document frequency. For a document d and a word t, tf-idf is calculated as following:

$$tf - idf_{t,d} = tf_{t,d} \times idf_t$$

$$tf_{t,d} = number\ of\ occurrences\ of\ t\ in\ d$$

$$idf_t = \log\left(\frac{number\ of\ documents\ containing\ t}{number\ of\ documents}\right)$$

3) *Word to Vector Representation*

In order to generate trainable features, we need to map each word into a vector representation. Initially, a straight-forward approach to represent words is using one-hot vector. So that each word is an $R|V| \times 1$ vector with all 0s, except one bit set to 1 to specify the index of that word in the sorted vocabulary, and |V| is the size of our vocabulary. This notation represents each word as a completely independent entity, without giving any notion of semantic similarity.

In order to improve this approach, we can map one-hot vectors to a smaller subspace which can encodes the semantic relationships between words, and this kind of mapping is called word-embedding. Word Embedding represent words in a continuous vector space where semantically similar words are mapped to nearby points. In another word, Word Embedding is a predictive model that predicts the occurrence of a certain word given some context information. It comes in two flavors: The Continuous Bag-of-Words model (CBOW) and the Skip-Gram model. The main difference between CBOW and Skip-Gram model is that, CBOW predicts a center word from the surrounding context, while Skip-Gram: does the inverse and predicts surrounding context words given a center word. In our project, we adopt the CBOW approach and implement it using the *Keras* python module.

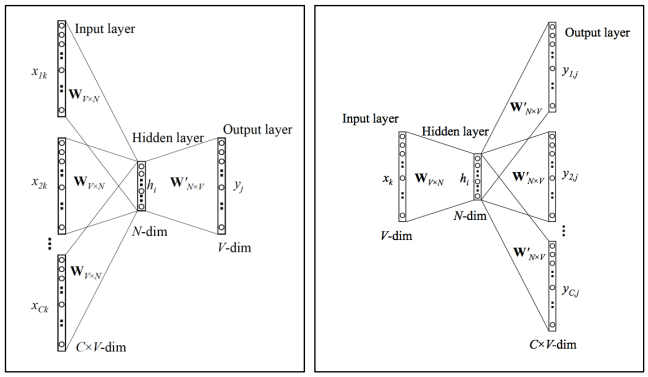

Fig. 2. CBOW versus Skip-Gram Embedding.

C. Model Training

In this section, we first designed and implemented three kinds of classifiers: a gradient-boost based classifier, a CNN based classifier, and a RNN based classifier. Next, we also combine the CNN and RNN model into an integrated network, in the hope of getting better performance than a single RNN or CNN module. We will compare the performance of each classification model in the Evaluation section.

1) XGBoost Classifier

Gradient boosting decision tree(GBDT) [14] is an optimized distributed gradient boosting framework. Because of its high efficiency and accuracy, it becomes more and more popular for solving various machine learning problems, for example, ranking problem [17], user behavior prediction [16] and multi-class classification [5]. There are several implementations of GBDT recently, such as XGBoost [20], pGBRT and LightGBM. In this project, we used LightGBM to implement our logistic regression model.

To further improve the computational efficiency, lightGBM applied Gradient-based One-Side Sampling(GOSS) and Exclusive Feature Bundling(EFB) strategy to exclude non-informative data instances and features so that improve the training speed greatly without losing accuracy. GOSS keeps data instances with larger gradients and randomly drop instances with small gradients to retain the accuracy of information gain estimation and speed up simultaneously. EFB are designed to dealing with datasets that have a large but sparse feature space. By greedily bundling mutually exclusive features, EFB can reduce the number of features.

Logistic regression is widely used on natural language processing, for example, in topic classification, text categorization, detection of spam pages. Logistic regression measures the relationship between an output variable $Y(category)$ and one or more independent variables, which are usually continuous, by using probability scores as the predicted values of the dependent variable. For input feature vectors $X = \{x_1, x_2 \dots x_n\}$, outcome ranges between 0 and 1 which represents the prediction of probability. For probability greater than 0.5, we would classify the observation as positive, otherwise, as negative. The probability of the out is assumed to follow a parametric model given by the sigmoid function:

$$P(Y = 1|X) = \frac{1}{1 + \exp\left[-(\beta_0 + \sum_{i=1}^{n} \beta_i x_i)\right]}$$

2) CNN-based classifier

Conventionally, CNN is widely applied to applications like image classification or object detection. We plan to leverage CNN for text classification for two reasons: first, we think that the important thing for toxicity content classification is to find sentiment features, like abuses or identity-hates, which are usually related one or two single words and can be extracted using a CNN. Second, we think that CNN converges faster than RNN due to its simple structure. Since each sample in the dataset is a short comment instead of a long paragraph/document, we can easily use padding approach to make it fit for a CNN classifier.

As we know, the major part of CNN is constructed by the convolutional layer, the pooling layer, and the fully-connected Layer.

The convolution layer is the building block of a CNN model. In this layer, a kernel matrix is used to conduct a dot production on each ‘slice’ of the training sample in order to extract features and filter out useless information. You can think this kernel matrix as a sliding window which traverses through the training sample, so that after the convolution layer, you will have a feature vector (or matrix, based on the structure of yur training sample) that can be used for further training.

The pooling layer is used to remove local variance and decrease the dimension of the feature vector, which therefore alleviates the problem of overfitting. There are two basic pooling approaches: max pooling and average pooling. We will explore each pooling approach and pick the optimal one in our following experiments.

After layers of convolution and pooling, we need activation functions to bring the needed non-linearity property. For the active function, we plan to use the classic one which is Relu [7]. Finally, we have a single vector which we can use for Fully connected layers.

In practice, we plan to use Python-Keras language to establish the CNN (convolution neural network) classifier. The model structure is given in Fig. 3.

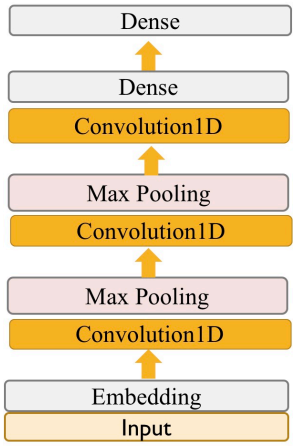

Fig. 3. CNN-based model

3) RNN-based Network Classifier

RNN is widely applied in many areas such as protein action recognition, speech recognition, machine translation and handwriting generation. It is a recurrent model that can learn order dependence in sequence prediction problems. As shown in Fig 3. In an RNN model, a chunk of neural network cells, looks at some input x_t and outputs a value h_t. A loop allows information to be passed from one step of the network to the next.

In this project, we explore three RNN variants: a LSTM network, a GRU network, and a Bidirectional RNN network, and show how we build a RNN-based model using these three variants.

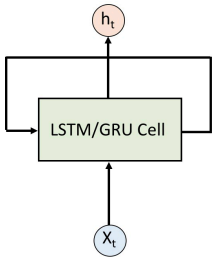

Fig. 4. RNN

a) RNN varitant 1: LSTM

Long Short-Term Memory (LSTM) is a variant of the vanilla RNN [7]. It differs from the standard RNN by having a new structure called memory cell. The memory cell is composed of four main elements: an input gate, a forget gate, an output gate, and a neuron with a self-recurrent connection (a connection to itself). The three gates contained in the memory cell can be represented as follows:

$$i_t = \sigma(U_i h_{t-1} + W_i x_t + b_i) \quad (1)$$

$$f_t = \sigma(U_f h_{t-1} + W_f x_t + b_f) \quad (2)$$

$$o_t = \sigma(U_o h_{t-1} + W_o x_t + b_o) \quad (3)$$

Where Eq. (1) – (3) represents the input gate, forget gate, and the output gate, respectively. Then the updated memory cell contains information both from the three gates and its original cell information:

$$c_t = f_t * c_{t-1} + i_t * \tanh(U h_{t-1} + W x_t + b)$$

The memory cell then conveys information to the updated hidden state:

$$h_t = o_t * \tanh(c_t)$$

Finally, the output layer looks like following:

$$y_t = V h_t + d$$

b) RNN variant 2: GRU

GRU network is also a recurrent neural network with gated cells, where each of the cell is called Gated Recurrent Unit (GRU). The idea behind a GRU is quite similar to that of a LSTM. Different from LSTM units, a GRU unit does not have an output gate. We show the cell structure of LSTM and GRU in Fig. 4.

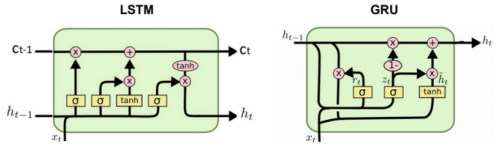

Fig. 5. LSTM cell versus GRU cell

c) RNN variant 3: Bidirectional RNN

The structure of a Bidirectional RNN is shown in Fig. 5. It is a combination two independent RNNs. The input sequence is fed in normal time order for one network, and in reverse time order for another. The outputs of the two networks are combined at each step, with the common combination being concatenation or summation. With this structure, Bidirectional RNN allows the networks to have both backward and forward information about the sequence at every time step.

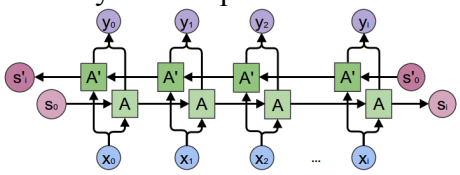

Fig. 6. Bidirectional RNN

Based on the abovementioned RNN variants, we designed a bidirectional RNN models using GRU cells. It contains two bidirectional RNN layers, with each layer containing 64 hidden cells. The final output layer is a dense layer with 6 units, as we have six toxicity types.

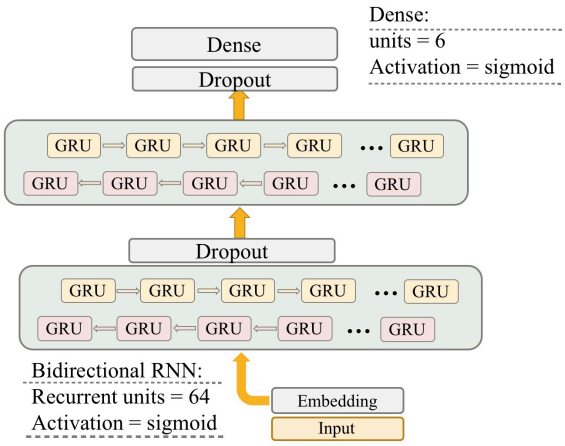

Fig. 7. Bidirectional RNN model using GRU cells

d) Integrated model using both RNN and CNN

Given the CNN based model and RNN based model, we propose an integrated network by merging the two models together, so the output feature map of the CNN model and the RNN model will be concatenated and fed into the final output layer. The structure of this integrated model is shown in Fig. 7.

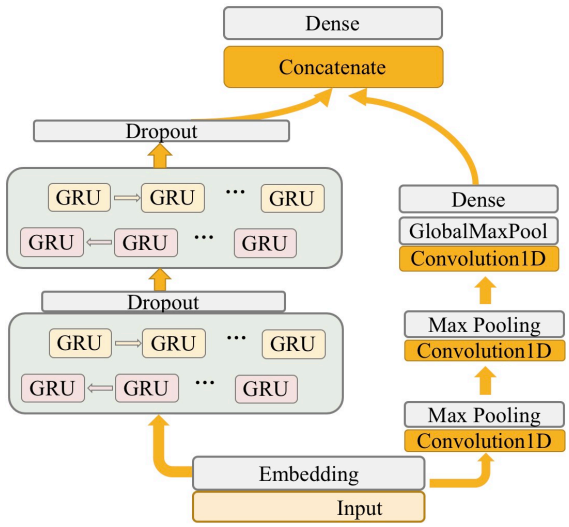

Fig. 8. Integrated model using both RNN and CNN

D. Performance Evaluation

We apply different models on a real-word dataset collected by Wikipedia's talk page edits. In this section, we will first introduce the metrics used for evaluation the model performance. Next, we evaluate the performance of each model. Finally, we compare the performance across different models.

1) Metrics

We adopt four metrics to evaluate the models' performance:

- Training accuracy
- Training loss
- Testing accuracy
- Testing loss

Especially, for loss function, we adopt binary cross entropy loss. Cross-entropy loss measures the performance of a classification model whose output is a probability value between 0 and 1. A perfect model would have a log loss of 0. Binary cross-entropy can be calculated as follows:

$$-(ylog(p) + (1-y)\log(1-p)$$

2) Regression model evaluation

For regression model, we evaluate the classification accuracy for each of the six toxicity types using the model called LightGBM Logistic regression. We use the average accuracy for the six types as the model's general performance. The detailed results are shown in Table 1.

Accuracy on each toxicity	1	2	3	4	5	6	Overall Accuracy
LLR Testing	97.27%	98.34%	99.41%	98.98%	97.98%	97.95%	98.32%
LLR Training	99.55%	99.79%	99.84%	99.99%	99.38%	99.97%	99.75%

TABLE. 1: EVALUATION RESULTS FOR REGRESSION BASED MODEL

3) *CNN model evaluation*

We evaluated our RNN model using the Wikipedia comments dataset. Please be noted that due to the large amounts of training samples available (160,000 samples), our models converge quickly after a few of epochs. Therefore, all of the following experiments are done with epoch = 1, batch size = 64.

For the CNN based model, we compare out model's performance with one benchmark model, which is a CNN model with one convolution layer and one max-pooling layer. The convolution layer is 125×4 in size, followed by a rectified linear unit (*relu*) activation layer.

We show the performance comparison in Table 2. We can see that our model outperforms the benchmark in training accuracy/loss and testing loss. For the testing accuracy, they achieve the same result. We also compare the convergence speed of these two models in Fig. 9. We can see that our model, although with a more complexed network structure, converges faster than the simpler version in terms of training loss, and achieves almost the same pace in terms of training accuracy.

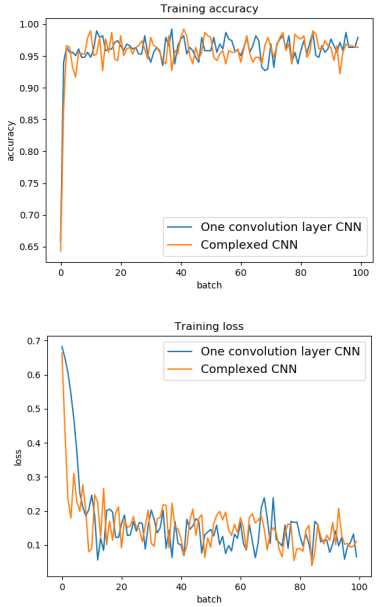

Fig. 9. Training performance for CNN-based models

Model Performance	Training Accuracy	Testing Accuracy	Training Loss	Testing Loss
One layer CNN	96.892%	97.932%	0.104	0.061
Our CNN model	**97.150%**	**97.932%**	**0.090**	**0.052**

TABLE. 2: EVALUATION RESULTS FOR RNN BASED MODEL

4) *RNN model evaluation*

We compare our model's performance against two benchmarks: the first benchmark is a standard LSTM network, i.e. a network with one regular RNN layer. The second is a LSTM network with a pre-trained embedding matrix. We show the training accuracy and training loss for the first 100 batches in Fig. 10. It can be observed that a pre-trained embedding matrix does improve the training performance with faster convergence speed. It can also be observed that our model, a bidirectional 2-layer RNN model using GRU cells, outperforms the two benchmarks with lower loss, higher accuracy, and faster convergence speed. Table 3 shows the detailed performance of each of the three models.

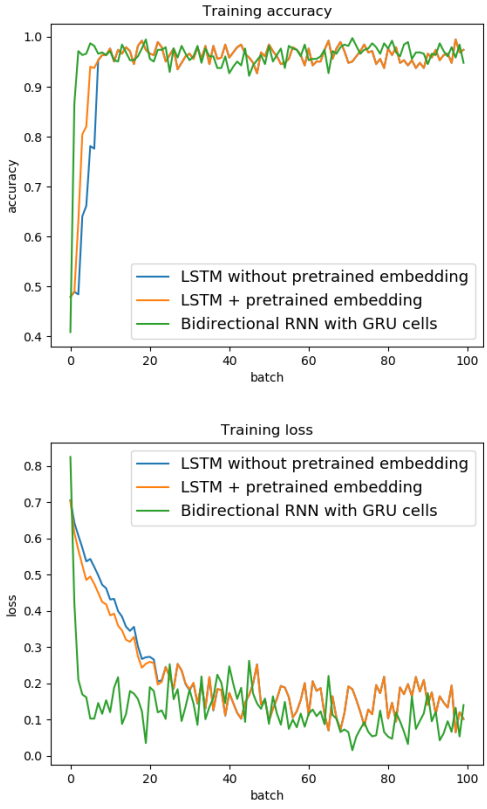

Fig. 10. Training performance for RNN-based models

Model Performance	Training Accuracy	Testing Accuracy	Training Loss	Testing Loss
LSTM	95.732%	96.418%	0.161	0.139
LSTM + Pre-embedding	95.907%	96.418%	0.159	0.140
Our Model	**98.089%**	**97.415%**	**0.052**	**0.077**

TABLE. 3: EVALUATION RESULTS FOR RNN BASED MODEL

5) Integrated model evaluation

Finally, we compare the performance of the integrated model with the single RNN-based model and CNN-based model. We show the best performance results of each kind of neural network in Table 4. Generally, the integrated model has the best performance in terms of testing/training accuracy. In terms of training loss, RNN model is slightly better than the integrated model.

Compared with the regression model whose testing accuracy is 98.32%, the integrated neural network model achieves a similar performance, with less than 0.2% of performance difference.

Model Performance	Training Accuracy	Testing Accuracy	Training Loss	Testing Loss
Our CNN model	97.150%	97.932%	0.090	0.052
Our RNN model	97.415%	98.089%	**0.052**	0.077
Our Integrated model	**97.534%**	**98.141%**	0.072	**0.052**

TABLE. 4: EVALUATION RESULTS FOR 3 NEURAL NETWORKS

III. CONCLUSION

In this project, we designed, implemented and analyzed different classification models for toxic comment detection using regression techniques and neural networks. We found that the regression model achieves the best performance, followed by an integrated neural network model using both RNN and CNN layers.

Our future work will be using ensemble learning to taking the benefit of both neural network and regression to build a more powerful classification model, as well as applying our models to various real-world dataset.

IV. CONTRIBUTIONS OF TEAMMEMBERS

In this project, we together designed, implemented, and evaluated different models wrote this report project together.

1. Wei Wang designed and implemented the regression-based model. She is also responsible for the feature extraction, model training, and evaluation of this model. She and Zhuangdi together finished the literature review collection rand report draft writing.
2. Zhuangdi Zhu designed and implemented the RNN-based model, as well as the integrated model using both RNN and CNN structure. She is responsible for the feature extraction, model training and evaluation of these two models. She wrote the Python & MATLAB code used for plotting the evaluation results for the neural network part.
3. Zhiming Jiang designed and implemented the CNN-based models. He is responsible for the feature extraction, model training and evaluation of the CNN model. He helped Zhuangdi on pre-processing the data for feeding neural networks, as well as model evaluation.

REFERENCES

[1] Data link for Kaggle competition: toxic comment classification: https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data

[2] Covington, Paul, Jay Adams, and Emre Sargin. "Deep neural networks for youtube recommendations." Proceedings of the 10th ACM Conference on Recommender Systems. ACM, 2016.

[3] Salton, Gerard, and Christopher Buckley. "Term-weighting approaches in automatic text retrieval." *Information processing & management* 24.5 (1988): 513-523.

[4] Ramos, Juan. "Using tf-idf to determine word relevance in document queries." *Proceedings of the first instructional conference on machine learning*. Vol. 242. 2003.

[5] Hanley, James A., and Barbara J. McNeil. "The meaning and use of the area under a receiver operating characteristic (ROC) curve." *Radiology* 143.1 (1982): 29-36.

[6] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9.8 (1997): 1735-1780.

[7] Nair, Vinod, and Geoffrey E. Hinton. "Rectified linear units improve restricted boltzmann machines." *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010.

[8] R. Socher, B. Huval, C. D. Manning, and A. Y. Ng. "Semantic compositionality through recursive matrix-vector spaces. " In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1201–1211. Association for Computational Linguistics, 2012.

[9] Chung, Junyoung, et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling." *arXiv preprint arXiv:*1412.3555 (2014)

[10] Zhou, Guo-Bing, et al. "Minimal gated unit for recurrent neural networks." *International Journal of Automation and Computing* 13.3 (2016): 226-234.

[11] Li, Fan, and Yiming Yang. "A loss function analysis for classification methods in text categorization." *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. 2003.

[12] Zhang, Tong, and Frank J. Oles. "Text categorization based on regularized linear classification methods." *Information retrieval* 4.1 (2001): 5-31.

[13] Genkin, Alexander, David D. Lewis, and David Madigan. "Sparse logistic regression for text categorization." *DIMACS Working Group on Monitoring Message Streams Project Report* (2005).

[14] Friedman, Jerome H. "Greedy function approximation: a gradient boosting machine." *Annals of statistics* (2001): 1189-1232.

[15] Li, Ping. "Robust logitboost and adaptive base class (abc) logitboost." *arXiv preprint arXiv*:1203.3491 (2012).

[16] Richardson, Matthew, Ewa Dominowska, and Robert Ragno. "Predicting clicks: estimating the click-through rate for new ads." Proceedings of the 16th international conference on World Wide Web. ACM, 2007.

[17] Burges, Christopher JC. "From ranknet to lambdarank to lambdamart: An overview." *Learning* 11.23-581 (2010): 81.

[18] Ke, Guolin, et al. "LightGBM: A highly efficient gradient boosting decision tree." *Advances in Neural Information Processing Systems*. 2017.

[19] Genkin, Alexander, David D. Lewis, and David Madigan. "Large-scale Bayesian logistic regression for text categorization." *Technometrics* 49.3 (2007): 291-304.

[20] Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. ACM, 2016.

[21] Tyree, Stephen, et al. "Parallel boosted regression trees for web search ranking." *Proceedings of the 20th international conference on World wide web*. ACM, 2011.

[22] Salton, Gerard, and Christopher Buckley. "Term-weighting approaches in automatic text retrieval." Information processing & management 24.5 (1988) : 513-523.

[23] Nasukawa, Tetsuya, and Jeonghee Yi. "Sentiment analysis: Capturing favorability using natural language processing." Proceedings of the 2nd international conference on Knowledge capture. ACM, 2003

[24] Liu, Bing, and Lei Zhang. "A survey of opinion mining and sentiment analysis." Mining text data. Springer US, 2012. 415-463.

[25] Cambria, Erik, et al. "New avenues in opinion mining and sentiment analysis." IEEE Intelligent Systems 28.2 (2013): 15-21.