Using Network to fix network -- Toxic Comment Classification via DeepLearning 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 aim to provide a tool which recognizes toxic comments presented on social networks. Digital abuse is a critical issue worldwide. The abuse of toxic comments on the internet can be harmful to both individuals and the society as a whole. Since the approach of manually rating toxic comments works with a huge cost, we aim to build a multi-headed model that can automatically detect different types of toxicity like threats, obscenity, insults, and identity-based. We aim to use different neural networks to build the model, and compare the powerfulness with traditional models such as regression**

Keywords—machine learning; natural language processing, deep learning; neural networks.

I. INTRODUCTION

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 (i.e. comments that are rude, disrespectful or otherwise likely to make someone leave a discussion) 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. The burst of negative events on the internet can generate large amount of information in a short time, which overwhelms the human raters. 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.

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; y 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 sample t in D, we have a label y, which is a binary vector of dimension d. In practice, we use a dataset of comments from Wikipedia's talk page edits.

II. PROPOSED METHODS

We consider four stepwise tasks in this project: data collection and 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 types of toxicity are:

1. $toxic$;
2. $severe - toxic$;
3. $obscene$;
4. $threat$;
5. $insult$;
6. $identity - hate$

The dataset is available online [1].

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 netural dataset

B. Feature Extraction

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.

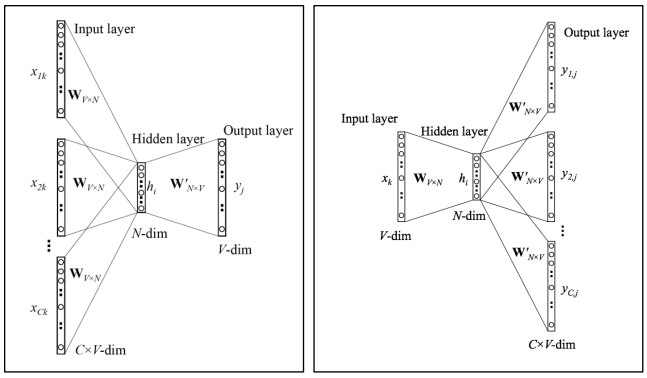

Fig. 2. CBOW versus Skip-Gram Embedding.

C. Model Training

We plan to work on three kinds of classifiers: a RNN based classifier, a gradient-boost based classifier, and a CNN based classifier. We will not only compare the performance difference among these classifiers, but also seek to use ensemble learning for a synthesized prediction.

1) LSTM Network Classifier

LSTM network classifier applies Recurrent Neural Network (RNN) using the Long Short-Term Memory (LSTM) [7] architecture. LSTM is a type of recurrent neural network that can learn order dependence in sequence prediction problems, which is widely applied in many areas such as protein action recognition, speech recognition,

machine translation and handwriting generation. LSTM introduces a new structure called a memory cell shown in Fig. 3. A memory cell is composed of four main elements: an input gate, a neuron with a self-recurrent connection (a connection to itself), a forget gate and an output gate.

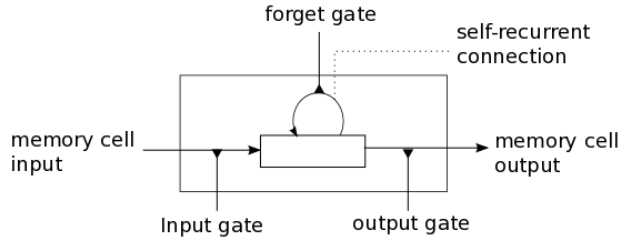

Fig. 3. LSTM memory cell

A memory cell together with outer recurrence like ordinary recurrent network construct LSTM network shown in Fig. 4. The network makes predictions based on the individual time steps of the sequence data.

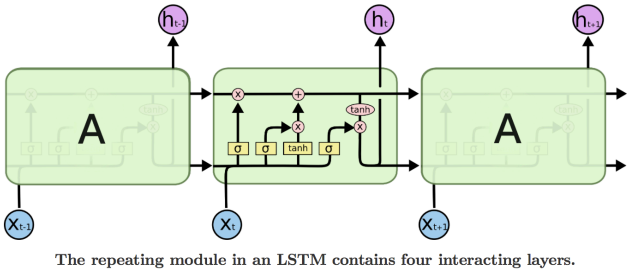

Fig. 3. LSTM network structure

2) XGBoost Classifier

XGBoost (Extreme Gradient Boosting), is an optimized distributed gradient boosting library. It provides a parallel tree boosting framework, using gradient boosting to speed up machine learning algorithms. We plan to use logistic regression classifier in XGBoost.

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:

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

3) CNN 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. We currently plan to explore multiple conversions and max-pooling layers and find the optimal structure.

D. Performance Evaluation

We plan to use ROC curve (receiver operating characteristic curve) and AUC (area under curve) [6] to evaluate the performance of all classifiers we implement. ROC curve illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.

The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold. For a binary classification problem. One of two labels positive or negative is assigned to the input. There are four possible outcomes from a binary classifier. If the outcome from a prediction is positive and the actual value is also positive, then it is called a true positive (TP); if the actual value is negative then it is a false positive (FP). If the outcome from a prediction is negative and the actual value is also negative, then it is called

a true negative (TN); if the actual value is positive then it is a false negative (FN). TPR and FPR are calculated as following:

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

AUC is the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one. AUC varies between 0 and 1, an uninformative classifier result in AUC of 0.5. Better classifier results in higher AUC.

III. Proposed Timeline

1. Mar. 18 ~ Mar. 24: Submit project proposal. Collect data and preprocess.
2. Mar. 25 ~ Apr. 14: Extract feature vectors from input data set. Implement algorithms.
3. Apr. 15 ~ Apr. 28: Apply methods on input feature vectors and measure the performance of each method.
4. Apr. 29 ~ May. 5: Write final project report and prepare the project presentation.

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.