Toxic Comment Detection and Classification

Hao Li
haoli94@stanford.edu

Weiquan Mao
mwq@stanford.edu

Hanyuan Liu
liuhy14@stanford.edu

Abstract

In the project, we implemented three models that could detect toxic comments with high accuracy. Naive Bayes-SVM is our baseline model, which achieved 68.33% F1 score and 87.57% EM score. Then two deep-learning models, LSTM and BERT, are fully investigated. We applied weighted loss to solve the imbalanced data problem. The best performance of a single model could achieve 81.19% F1 score and 95.54% EM score. In the end, two ensemble methods were used and increased the F1 and EM scores to 84.28% and 95.14%.

1. Introduction

Internet is an open discussing space for everyone to freely express their opinions. However, harassment and abuse are discouraging people from sharing their ideas and disturbing the internet environment. Platforms struggle to effectively facilitate conversations, leading many communities to limit or completely shut down user comments if it's toxic.

Motivated by this problem, we want to build technology to protect voices in conversation by machine learning models that can identify toxicity in online conversations, where toxicity is defined as anything rude, disrespectful or otherwise likely to make someone leave a discussion.

For our model, the input will be a comment. We used three different models to predict scores for "toxicity" (which is our target), "severe_toxicity", "obscenity", "identity_attack", "insult", "threat". Then the ensemble methods are studied to combine multiple learning algorithms and get the best result.

2. Related Work

Related research has looked into hate speech, online harassment, abusive language, cyberbullying, and offensive language. Generally speaking, toxic comment detection is a supervised classification task and can be approached by either manual feature engineering[2] or neural networks[4].

A large variety of machine learning approaches have been explored to tackle the detection of toxic language. Fahim Mohammad[3] compared traditional machine learning methods, from Naive Bayes (NB) and logistic regression (LR) to state-of-art neural networks. The performances of BiLSTM and NB-SVM are good and robust after data preprocessing compared to other models. Zimmerman et al[5] investigated ensemble models with different hyperparameters, which inspired us to combine various model architectures and different word embeddings for toxic comment classification.

Neural network approaches appear to be more effective[7], while feature-based approaches preserve some sort of explainability. In this paper we used NB-SVM as our baseline model, and later switched to explore deep neural networks (e.g. BiLSTM) and a new language representation model Bert[6], which stands for bidirectional encoder representations from transformers. Bert is designed to pretrain deep bidirectional representations from unlabeled text by jointly conditioning on both left and right contexts in all layers.

3. Dataset and Features

3.1. Data description

To train our models, we use the Civil Comments dataset from Kaggle.[1] The dataset comprises of over 1804000 rows. Each row contains a general toxic target score from 0 to 1, a comment text, scores under various labels such as severe toxicity, obscene, identity attack, insult, threat, asian, homosexual gay or lesbian, black, intellectual or learning disability, etc. and other information including the article id and rating, likes, disagrees of the comment and so on. The data distribution is shown in figure1, which is quite imbalanced for training. Some efforts to fix this problem will be discussed later.

The dataset is split into 80% as training set, 10% as dev set and 10% as test set.

3.2. Time series analysis

In order to have a better understanding of the data distribution, we first checked the time series for toxicity regarding to different identities. We mainly investigated four toxicity sub-type attributes.

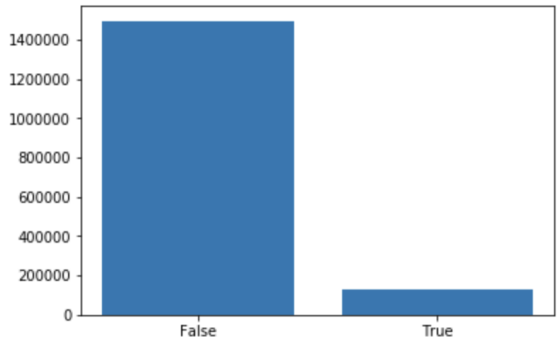
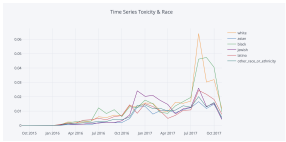


Figure 1: Data distribution



(a) toxicity: race

(b) toxicity: sexual orientation

(c) toxicity: religion

(d) toxicity: disability

Figure 2: time series analysis

As shown in figure 2, we can see that the data was collected between 2016 and 2017. And there was a interesting peak around Jan 2017. To better understand the underlying reason, we found out that the top three common words appear in the toxic comment are “trump”, “people” and“like”, which explains why there is a peak in Jan 2017.

3.3. Data preprocess

Based on the analysis, we did the following data preprocess:

1. Besides the common word contraction mappings, like “it’d” to ”it would”, we added word contraction mappings that were related to our dataset, like “Trump’s” to “Trump is”, “Trumpcare” to “Trump care”, “Obamacare” to “Obama care” and so on.

2. We collected some common misspell words and corrected them, such as “tRump” to “Trump”.

3. We translated some special Latin words and Emojis to English words.

4. Methods

4.1. Naive Bayes SVM Model

We combined Naive Bayes and Support Vector Machines to serve as our baseline model. SVM is built over NB log-count ratios in feature values, and it has been proven a strong and robust performer over all the presented tasks.

First, we will introduce the notation here. $f^i \in \mathcal{R}^{|V|}$ is the word count vector for comment i with label $y^i \in \{-1, 1\}$ indicating whether it’s toxic or not. V is the dictionary, the set of all words. f^i_j is the number of occurrences of word V_j in comment i. The log-count ratio vector is $r \in \mathcal{R}^{|V|}$.

$$r_j = log(\frac{1 + \sum_{i:y^i=1} f^i_j}{1 + \sum_{i:y^i=-1} f^i_j})$$

We formulated our model as a linear classifier, where the prediction for comment k in the test set is y^k.

$$y^k = sign(w^T x^k + b)$$

For multinomial event Naive Bayes model,

$$x^k = f^k, w = r, b = log(\frac{N_+}{N_-})$$

where N_+, N_- are the number of toxic and nontoxic comments in the training set.

For the SVM model with Naive Bayes log-count ratios in feature values,

$$x^k = r \circ f^k \text{(element-wise product)}$$

w and b are obtained by solving the optimization problem:

$$\min_{w,b} \frac{1}{2} w^T w$$

$$\text{s.t. } -y^i(w^T x^i + b)) + 1 \leq 0, \ i = 1, ..., n$$

4.2. LSTM model

Long short-term memory networks (LSTMs) are a special kind of RNNs designed to be capable of learning long-term dependencies. Vanilla RNNs can be tough to train on long sequences due to vanishing and exploding gradients caused by repeated matrix multiplication. LSTMs solve this problem by introducing a hidden cell and replacing the simple update rule of the vanilla RNN with a gating mechanism.

Frequently, the dependencies within sequential data, like sentences, are not just in one direction, but may be observed in both directions. Therefore, we used BiLSTMs, in which, Bidirectional layers exploit the forward and backward dependencies by combining the features obtained going in both directions simultaneously.

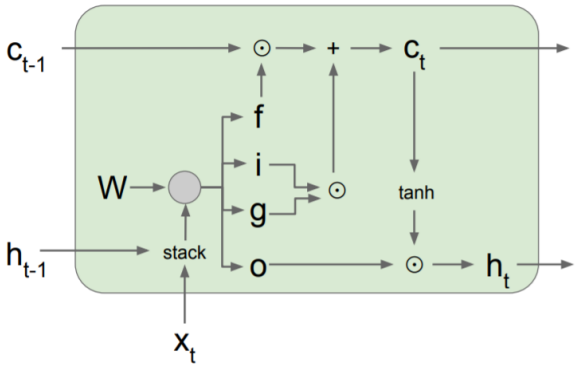


Figure 3: LSTM structure

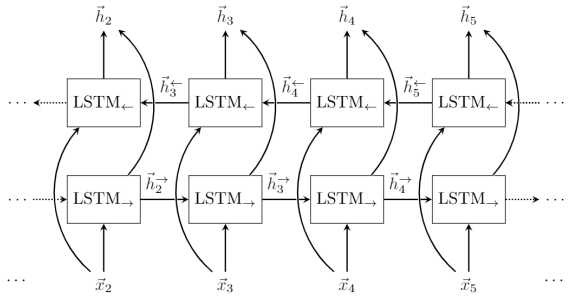


Figure 4: BiLSTM structure

We first converted every word in the input text to a vector based on word embeddings. Then we dropped out vectors at random positions to increase the robustness of our model. At the end of our model, in addition to the target score of toxicity, the model also predicted an auxiliary result, which not only included the score for toxicity, but also the scores for more specific characteristics of the comment text, such as "obscenity", "identity attack", "insult" and "threat". These scores are provided in the training set. Though these are not the target score we need from the model, the model predicted them to utilize these additional information in data.

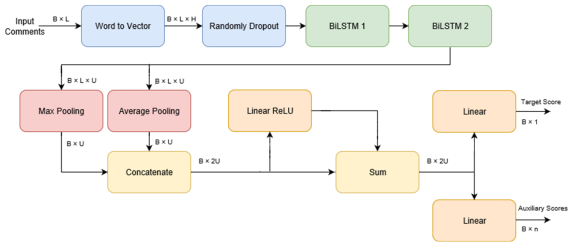


Figure 5: LSTM model

4.3. BERT model

Bidirectional Encoder Representations from Transformers(BERT) achieves state-of-the-art performance for eleven NLP tasks, including Question Answering and Sentence (and sentence-pair) classification tasks, through only fine-tuning the last layer. BERT has such noteworthy achievement because it learns a more powerful bi-directional representation than most of the previous approaches. BERTs architecture is mainly multi-layer bidirectional Transformer encoder with bidirectional self-attention mechanism. The encoder of BERT is pre-trained with two tasks, masked language model (MLM) and Next Sentence Prediction. These two objectives help encoder to learn both left and right contextual of a word in the sentence and provides significant support for downstream tasks like question answering.

We used the pre-trained BERT-Base model, which is cased and has 12 layer with 768-hidden, 12-heads, and 110M total parameters. Cased means that the true case and accent markers are preserved. To integrate the pre-trained encoder of BERT and fine-tune it to solve sentence classification problem, a final classification layer is added with weights $W \in R^{H \times K}$, where H is the encoded hidden size and K is the number of classifier labels. The label probabilities $P \in R^K$ are computed with a softmax layer, $P = \text{softmax}(CW^T)$.

4.4. Ensembling

In modern Machine Learning, Ensembling Methods are extensively used to combine multiple learning algorithms, preferably from different model classes, into an aggregate model with better performance than any single model. Each of the BERT and LSTM model we built make different predictions on probabilities of toxicity, and therefore we use Ensembling Methods in hope of utilizing the information extracted from all models. Specifically, we create an algorithm in combining these models.

Algorithm 1 Guided Random search + Weighted Average Ensembling

Input:
1. set of k models: $M \in R^k$
2. dev set: $\{(P_l = (probabilityoftoxicity)\}_{l \in 1,2...n_{dev}}$
3. prediction set: $\{P_m\}_{m \in 1,2...n_{pred}}$
4. max_num_iter

Output: Predictions on the prediction set

$best\_weight = Guided\_Random\_Search\_on\_Weight$(M, max_num_iter, dev set)

Return $Make\_Predictions$(M, $best\_weight$, prediction set)

Assume that we have a total of n models to ensemble, and for each inputting (Question, Paragraph) pair, model k

outputs

$$^{k}p_i, 0 \leq i \leq \text{len(dev examples)}$$

where $^{k}p_i$ is the predicted probability of the i-th dev example of the k-th model. Following this notation, our goal is now reduced to finding the best $w \in R^n$, such that

$$p_i := \sum_k w_k \times p_i^k, w \in R^n$$

reaches the most accurate prediction. With this motivation, we develop a pipeline(figure6) that learns the weights w, and make predictions on the dev set. The details are described in Algorithm1. In high level, our algorithm randomly assign weights to each model. With the weights learned, we re-do the predictions by taking the weighted average of the probabilities predicted by each model.

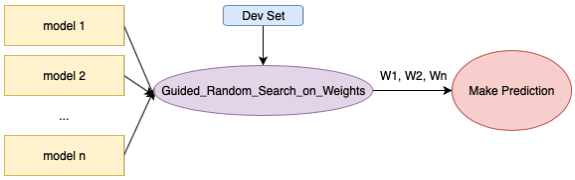


Figure 6: Pipeline for Guided Random Search + Weighted Average Ensembling

Another ensembling algorithm we would like to explore is most-confident voting. The idea is inspired from the data distribution. As we can see that there is more nontoxic comments than toxic comments in training set. The model tends to predict nontoxicity for many comments. Thus we would like to believe that as long as one of those model conforms a comment toxicity, it is more likely that it is the true ground.

5. Evaluation Method

We mainly used two types of evaluation metrics, Exact Match and F1 score. Exact Match (EM) is the percentage of outputs that match exactly the ground truth. F1 is the harmonic mean of precision and recall:

$$F_1 = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

where true positives are the toxic comments predicted as toxic, false positives are the nontoxic comments predicted as toxic, false negative are the toxic comments predicted as toxic.

6. Experiments and Results

6.1. Naive Bayes SVM Model

The evaluation of our baseline model, Naive Bayes SVM, is shown in Table 1. Naive Bayes SVM Model reached a high EM score. However, the F1 score was lower than EM by around 20%. It resulted from our model' tendency to predict comments as nontoxic. This can be explained by the imbalance of data, which had a much larger percentage of nontoxic comments than toxic comments.

Table 1: Naive Bayes Result

Description	Dev F1	Dev EM
Naive Bayes	68.33%	87.57%

6.2. LSTM Model

We trained the LSTM model for 4 epochs using the Adam optimizer with the initial learning rate 1e-3. We set a learning rate scheduler adjusting the learning rate at the end of every epoch during training. We used binary cross entropy loss as the loss function.

Table 2 shows the evaluation of the LSTM model. From the result, we can see that both EM score and F1 score are much higher than Naive Bayes SVM model. It is caused by the Naive Bayes model's ignorance of the relationship between words, which is a fatal problem. LSTM could remember the relationships and combine those meaning together to get the result, and it led to a more accurate result than our baseline model.

6.3. Weighted Loss LSTM

There are several methods put forward to solve the imbalanced data problem. We tried weighted loss to train our model. We set the weight of the loss of predictions for toxic comments, larger than the weight of the loss of predictions for nontoxic comments. From figure1, we can see that the rate of nontoxic comments and toxic comments is about 9:1. Thus we set the weights to be 0.9 for toxic comments and 0.1 for nontoxic comments. The results are shown in the Figure 2. The results indicated that weighted loss can be a very useful hyperparameter to fine-tune model trained on imbalanced data.

6.4. Data Preprocess with Contraction Mapping

Based on LSTM model discussed above, we tried to find whether the data preprocess with "Trump" words will work.

Table 2: Simple and weighted loss LSTM

Description	Dev F1	Dev EM
Simple LSTM	77.95%	95.37%
LSTM with weighted loss pair (0.9, 0.1)	81.12%	95.21%

We set weights of loss 0.5 and 0.5. We trained one LSTM model with contraction mapping and one without. The results are shown in table 3.

Table 3: Comparison between data preprocess

Description	Dev F1	Dev EM
With contraction mapping	77.95%	95.37%
Without contraction mapping	76.04%	95.38%

The results quite make sense. As mentioned before, the most frequent word in toxic comments is "Trump". Thus, if we preprocess those "Trump" words, the toxicity in the training set will be more obvious. The result reflects the effectiveness of contraction mapping. The improvement of F1 score shows that the model became better at detecting toxic comments.

6.5. BERT

Because of time limit, we trained BERT model with only one epoch and got results comparable to the LSTM model. Besides, batch size was set to be 32 due to the limitation of GPU memory. Learning rate was set to be 2×10^{-2}, which is recommended by the official guideline.

With the same setting, we also trained the BERT model using weighted loss (0.9 for toxic comments and 0.1 for nontoxic comments). The results are shown in table 4.

Table 4: BERT model and ensembling

Description	Dev F1	Dev EM
Simple BERT	77.37%	95.73%
BERT with weighted loss pair (0.9,0.1)	81.19%	95.54%

6.6. Ensemble Method

With the above models, we ran our ensemble algorithm and the results are listed in table 5.

The first method we used was most-confident voting, which takes the most confident result with the highest confidence. This could work because that the prediction mistakes mainly occurred on toxic comments and the F1 score was relatively lower than the EM score. So if we took the most confident result, the toxic comments would be more likely to be detected. As shown in the table, the F1 score increases without the compromise of decreasing EM score. The results were gotten from BERT with weighted loss, BERT without weighted loss and LSTM with weighted loss.

The second method is guided-random-research with weighted average ensembling. The results were gotten from Bert with weighted loss and LSTM with weighted loss.

Table 5: BERT model and ensembling

Description	Dev F1	Dev EM
Best model without ensembleing	81.19%	95.54%
Ensembling with most confident vote	84.28%	95.14%
Ensembling with guided weight	81.57%	95.50%

7. Conclusion

To sum up our work, we implemented three machine learning models, namely Naive Bayes-SVM model, LSTM and BERT. We used information from data visualization to preprocess data. Weighted loss performed very well in fixing the problem of imbalanced data. BERT can produce state-of-the-art work with only one output layer. We trained BERT with only one epoch and got results better than the LSTM model. At last, we used two ensemble methods to further improve the quantitative results.

For future work, we would like to try some other ways besides weighted loss to fix the problem of imbalanced data. BERT model brought us surprisingly good results and we would like to explore it if we have more time.

8. Acknowledgements

We would like to acknowledge the inspiration of the Naive Bayes SVM model and the base LSTM network model which provided us the starting point of our code:

https://www.kaggle.com/jhoward/nb-svm-strong-linear-baseline

https://www.kaggle.com/bminixhofer/simple-lstm-pytorch-version

References

[1] Cyphers. Pitchfork reviews dataset. `https://www.kaggle.com/bcyphers/pitchfork-reviews`, 2018.

[2] Z. Z. David Robinson and J. Tepper. Hate speech detection on twitter: Feature engineering v.s. feature selection. *The Semantic Web: ESWC 2018 Satellite Events*, pages 46–49, 2018.

[3] F. Mohammad. Is preprocessing of text really worth your time for toxic comment classification? *Int'l Conf. Artificial Intelligence*, pages 447–453, 2018.

[4] M. G. V. V. Pinkesh Badjatiya, Shashank Gupta. Deep learning for hate speech detection in tweets. *WWW'17'*, pages 759–760, 2017.

[5] U. K. Steven Zimmerman and C. Fox. Improving hate speech detection with deep learning ensembles. *LREC*, 2018.

[6] J. D. M.-W. C. K. L. K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *NAACL-HLT*, 2018.

[7] Z. Zhang and L. Luo. Hate speech detection: A solved problem? the challenging case of long tail on twitter. *CoRR*, 2018.