Toxic Comment Classification

Dian Qi
A14581615
diqi@ucsd.edu

Zhuo Liu
A14723302
zhl526@ucsd.edu

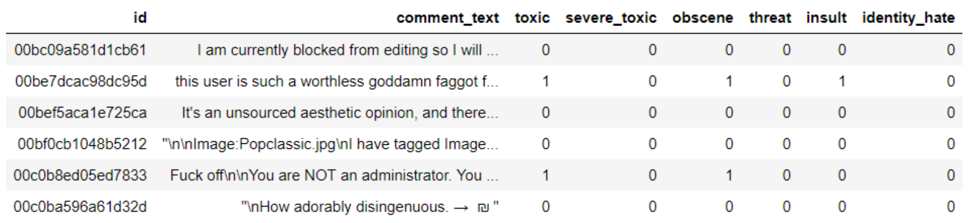
Xurui Zhang
A92120779
zxurui@ucsd.edu

Abstract

Classifying toxic comments online is significant but hard to achieve. We aim to automatically classify each comment into one of six overlapping categories. We compare the performance of different models, and design and implement a double-layered bidirectional LSTM with attention mechanism. The highest ROC AUC score it achieves on test set is 0.9841.

1 Introduction

Detecting and Classifying toxicity in online comments has become an increasingly important task. Given the magnitude of user-generated online comment and the impact of toxic comments, automatic controlling and moderation of these comments are necessary, which is essentially a natural language processing task. This project aims to design and implement a deep learning model that can efficiently classify toxic comment online. We will use dataset provided by the Toxic comment Classification Challenge on Kaggle[1]. The dataset is provided by Google's Jigsaw team, where each comment can have six overlapping types.



id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
00bc09a581d1cb61	I am currently blocked from editing so I will ...	0	0	0	0	0	0
00be7dcac98dc95d	this user is such a worthless goddamn faggot f...	1	0	1	0	1	0
00bef5aca1e725ca	It's an unsourced aesthetic opinion, and there...	0	0	0	0	0	0
00bf0cb1048b5212	"\n\nImage:Popclassic.jpg\nI have tagged Image...	0	0	0	0	0	0
00c0b8ed05ed7833	Fuck off\n\nYou are NOT an administrator. You ...	1	0	1	0	0	0
00c0ba596a61d32d	"\nHow adorably disingenuous. → ₪ "	0	0	0	0	0	0

Figure 1: A selection chosen particularly to show multiple labels of the corpus. Note that in reality, toxic comments only takes a small proportion of the dataset, and that samples from different categories are quite unbalanced.

To have a grasp of the six types of toxicity, `toxic`, `severe_toxic`, `obscene`, `threat`, `insult`, and `identity_hate`, a sample of entries in our dataset is depicted in Figure 1.

In this report, we will deliberate on our approach that consists of four parts: data preprocessing, model development, model deployment, and model evaluation. The evaluation metric we choose is ROC AUC, which will be compared with baseline models, as well as existing approaches. The rest of the paper is organized as follows. Section 2 is about data preprocessing. Section 3 presents our model and other baseline models. Section 4 is for conclusion and future work.

[1]https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge

2 Data Preprocessing

To transform raw data into uniform format, we first tokenize the input to remove punctuation and stop words, after which all the comments are changed to lowercase. We also pad or truncate the comments to ensure that each token list has the same length.

We choose different methods of word embedding for different models to compare. We use TF-IDF in logistic regression model and NB-SVM model, and use Word2Vec in Text-CNN model. For RNN, GRU, and LSTM models, we use the pretrained global vectors for word representation (GloVe) [1]. The double-layered bidirectional LSTM with attention that we implemented also uses GloVe. After that, we create a dictionary that stores indices of words that will be used to populate the word embedding matrix for neural methods. Misspelled or slang words that are not within the 400k words GloVe word embedding are mapped to the $\langle$unk$\rangle$ token.

Finally, we held back 20% of the training set as the dev set for testing the out-of-sample performance of our models.

Category	Occurences	Percentage
Threat	478	0.3%
Identity Hate	1405	0.9%
Severe Toxic	1595	1.0%
Insult	7877	4.9%
Obscene	8449	5.3%
Toxic	15294	9.6%
No label	143346	89.8%

Table 1: Number of occurences of different labels in the dataset and their percentages from least to max.

Since instances of different categories our dataset are quite unbalanced, as can be seen from Table 1, we assign each label a weight that is inverse proportional to their occurences. The weights for rare labels are greater than those of frequent ones, so their gradients will be larger in the sense that mistakes on rare labels are less ”tolerable”. Note that unlabeled data take a large proportion of the our dataset such that always guessing a comment as nontoxic will yield an accuracy score near 90%. In this situation, our choice of ROC AUC as evaluation metric makes more sense than accuracy.

3 Our Approach

3.1 Baseline Models

We implemented and chose several baseline models: Logistic Regression (LR), Naive Bayes-Support Vector Machine (NB-SVM), Vanilla Recurrent Reural Network (RNN), Gated Reccurent Unit (GRU), Long Short-Term Memory (LSTM). The bidirectional architectures of GRU and LSTM are also implemented with ease in Python using Pytorch.

Both LR model and NB-SVM models use TF-IDF to generate word embeddings, and concatenates character-level embedding and word-level embedding together to feed into logistic regression model from Sklearn. The main difference between these two models is that SVM is a hard classifier and choose support vectors that has the most discriminatory power in the training set, while LR is a probabilistic one. In our code they both use Stochastic Gradient Descent (SGD) to minimize the cross entropy loss function:

$$J(w) = -\sum_{i=1}^{m} y^{(i)} \log P(y=1) + (1-y^{(i)}) \log P(y=0) \tag{1}$$

The drawback of LR and NB-SVM models is that they cannot capture sequential information. However, the evaluation results turn out to be surprisingly good (Table 2). In this case, those linear models even outperforms some neural ones.

Category	ROC AUC Score
Toxic	0.9692
Severe Toxic	0.9876
Obscene	0.9839
Threat	0.9834
Insult	0.9774
Identity Hate	0.9739
Average	**0.9792**

Category	ROC AUC Score
Toxic	0.9745
Severe Toxic	0.9865
Obscene	0.9856
Threat	0.9875
Insult	0.9792
Identity Hate	0.9756
Average	**0.9815**

Table 2: ROC AUC score of (a). LR model, and (b). NB-SVM model.

We calculate ROC AUC score for each label and average over them:

$$\text{ROC AUC} = \int_{\infty}^{-\infty} \text{TPR}(T)\text{FPR}'(T)\, dT \tag{2}$$

where TPR is true positive rate, FPR is false positive rate.

We also implement recurrent architectures, such as vanilla RNN, RNN with GRU and LSTM cells. The output from the last time step is feeded into a fully-connected layer with sigmoid activations to get the probabilities of each label. The performances of RNNs with GRU or LSTM cells are similar, but both higher than that of vanilla RNN. Equations for GRU is therefore omitted here, they they can be found at [2]. The original LSTM equation [4] is show below:

$$\begin{aligned}
\boldsymbol{i}_t &= \sigma\left(\boldsymbol{x}_t \boldsymbol{W}^{(i)} + \boldsymbol{h}_{t-1}\boldsymbol{U}^{(i)}\right)\\
\boldsymbol{f}_t &= \sigma\left(\boldsymbol{x}_t \boldsymbol{W}^{(f)} + \boldsymbol{h}_{t-1}\boldsymbol{U}^{(f)}\right)\\
\boldsymbol{o}_t &= \sigma\left(\boldsymbol{x}_t \boldsymbol{W}^{(o)} + \boldsymbol{h}_{t-1}\boldsymbol{U}^{(o)}\right)\\
\tilde{\boldsymbol{c}}_t &= \tanh\left(\boldsymbol{x}_t \boldsymbol{W}^{(o)} + \boldsymbol{h}_{t-1}\boldsymbol{U}^{(o)}\right)\\
\boldsymbol{c}_t &= \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \circ \tilde{\boldsymbol{c}}_t\\
\boldsymbol{h}_t &= \boldsymbol{o}_t \circ \tanh(\boldsymbol{c}_t)
\end{aligned}$$

where $\boldsymbol{i}, \boldsymbol{f}, \boldsymbol{o}$ are the input, forget and output gates, respectively. $\boldsymbol{U}$ is the recurrent connection between previous and current hidden layers, $\boldsymbol{W}$ is the weighted matrix for input $\boldsymbol{x}$ of current hidden layer).

With the use of LSTM cells, we can capture sequential information, which is the feature of all RNNs, and prevent exploding gradient problems by input and forget gates. We can also add dropout[6] to prevent overfitting, and an example choice is $p_{\text{drop}} = 0.2$.

3.2 Our Model

We are inspired by the Stanford CS224n final project by Baumer, M. and Ho, A. [3], where they implement a Tensorflow framework of bidirectional LSTM with attention. We developed a double-layered bidirection LSTM with a different attention implementation using Pytorch. Attention mechanism is proposed by Yang *et al.* [5], and works by adding importance to most relevant words and use it in the process of prediction. The equation for attention mechanism is shown below:

$$\begin{aligned}
\boldsymbol{v}_t &= \tanh(\boldsymbol{h}_t \boldsymbol{W}_a + \boldsymbol{b}_a)\\
s_t &= \boldsymbol{v}_t \boldsymbol{u}_a^\top\\
\alpha_t &= \frac{\exp(s_t)}{\sum_{t=1}^{T}\exp(s_t)}\\
\tilde{\boldsymbol{h}} &= \sum_{t=1}^{T}\alpha_t \boldsymbol{h}_t
\end{aligned}$$

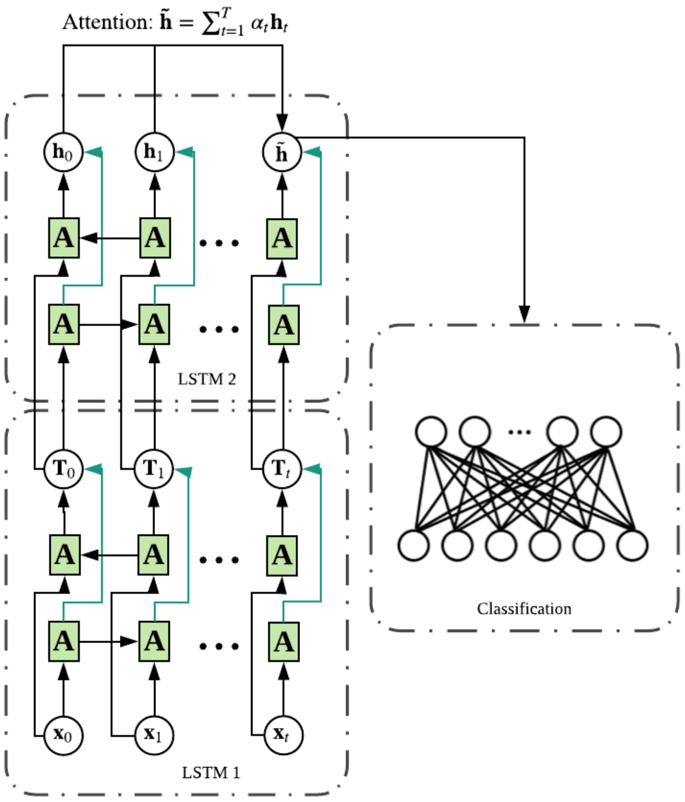


Figure 2: Visualization of the architecture of our model.

where $\boldsymbol{h}_t$ is hidden vectors of LSTM cell from time step t, $\boldsymbol{v}_t$ is the new word representation produced, $\boldsymbol{u}_a$ is the word context vector used to calculate s_t by dotting $\boldsymbol{v}_t$, α_t is the weights for hidden vectors from every time step by taking softmax of all s_t. $\tilde{\boldsymbol{h}}$ is the final output of LSTM that takes every weighted hidden vectors into account. Note that our implementation differs from the original one, since we use 3 layer MLP architecture:

$$\begin{aligned}
\boldsymbol{v}_{t_1} &= \text{ReLu}(\boldsymbol{h}_t \boldsymbol{W}_{a_1} + \boldsymbol{b}_{a_1}) \\
\boldsymbol{v}_{t_2} &= \text{ReLu}(\boldsymbol{v}_{t_1} \boldsymbol{W}_{a_2} + \boldsymbol{b}_{a_2}) \\
s_t &= \text{ReLu}(\boldsymbol{v}_{t_2} \boldsymbol{W}_{a_3} + \boldsymbol{b}_{a_3}) \\
\alpha_t &= \frac{\exp(s_t)}{\sum_{t=1}^{T} \exp(s_t)} \\
\tilde{\boldsymbol{h}} &= \sum_{t=1}^{T} \alpha_t \boldsymbol{h}_t
\end{aligned}$$

In brief justification is that MLP is an universal approximator, so the dot product similarity between the context vector and new word presentation can be approximated by our 3 layer MLP if PEs in each hidden layer are properly chosen.

The diagram of our model is shown in Fig.2. Note that since we are using double-layered bidirection LSTM, the output from every time step of the first LSTM serves as input for the second one from the same time step. Attention mechanism is applied only at the second LSTM. The result from the last time step in the second LSTM is feeded into the fulled-connected for classification. The choice of hyperparameters, e.g., dimensions of hidden layer, is discussed in the next section.

4 Experiment

4.1 Hyperparamter Tuning

Consider the magnitude of models that we are comparing our model against, we use a random 20% of the training set and GloVe of dimension 50 when developing. After a systematic search for hyperparameters, we changed to the full traning set and use GloVe of dimension 300. Current optimal hyperparameter settings are: $lr = 0.0002$ for Adam optimizer [7]; $p_{drop} = 0.25$, $\dim(\text{MLP}_1) = 768$, and $\dim(\text{MLP}_2) = 300$ for 3 layer MLP in `Attention` class; $p_{drop} = 0.2$ for LSTM, and $\dim(\text{hidden}_{LSTM}) = 128$. Note that dropout for LSTM is one at the output, rather than the recursive dropout at each cell. `torch.nn.LSTM` does not support recursive dropout, and we failed to implement it due to time limitation. We are particularly careful in preventing overfitting, which is partially due to our 3 layer MLP architecture for attention mechanism. An increase in dev loss usually occurs after 50 epoches.

Category	**ROC AUC Score**
Toxic	0.8943
Severe Toxic	0.9718
Obscene	0.9533
Threat	0.9884
Insult	0.9505
Identity Hate	0.9881
Average	**0.95770**

Table 3: Early stopping at epoch 10.

In an early stopping at epoch 10 (Table 3), we notice that the ROC AUC scores for rare labels, `Severe Toxic`, `Threat`, and `Identity Hate`, are higher than those of other frequent ones, which indicates that our inverse proportional class weight assignment take effect.

4.2 Evaluation

We compare our model against six other models, and calculate their average ROC AUC score over all the labels.

Model	**ROC AUC**
Bi-GRU+CNN	0.9857
Our Model	**0.9841**
Bi-LSTM	0.9813
NB-SVM	0.9815
LR	0.9792
LSTM	0.9743
Text-CNN	0.9724

Table 4: Comparison between models using mean ROC AUC from dev set

The result in Table 4 shows that our model rank the second. Bi-GRU+CNN has delicate choices of hyperparameters and unusual but surprisingly effective design of CNN after Bi-GRU, which cannot be fully explained by its author[2]. An interesting fact is that LR and SVM outperforms LSTM, and Text-CNN (also RNN and GRU, which are not listed above). Traditional machine learning techniques show their effectiveness in this particular task.

[2]http://konukoii.com/blog/2018/02/19/twitter-sentiment-analysis-using-combined-lstm-cnn-models

5 Conclusions and Future Work

We implemented a double-layered bidirectional LSTM with attention mechanism in Python using Pytorch that successfully performs toxic online comments classification. The best mean ROC AUC score of our model on validation set is 0.9841.

We have encounter several problems. The first one is we have to manually handle `LSTMCell` in Pytorch in order to add recursive dropout to `LSTM` in Pytorch. We failed to implement it due to time limitation. Otherwise, the perform of model would have been improved since it has been shown quite effective by many other researches, and we don't have to tune recursive dropout rate, whose best is known to be at 0.5. It would help with the overfitting problem of our model. Another problem is that there are many slang and misspelled words in online comments that are not covered by GloVe word embedding. If we can find a better way to handle those words, the data preprocessing part would have been much improved.

Our next goal is to first implement recursive dropout in Pytorch, and then try to integrate our model into a browser, e.g., plugin for Google Chrome, that helps user block toxic comments on specific websites based on their preferences.

References

[1] Pennington, J., Socher, R., & Manning, C. (2014) GloVe: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*.

[2] Cho, K., *et al*. (2014) Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, arXiv:1406.1078.

[3] Baumer, M., & Ho, A. (2018) Toxic Comment Categorization using Bidirectional LSTMs with Attention. In *Stanford CS224n Final Project Reports*

[4] Hochreiter, S. & Schmidhuber, J. (1997) Long Short-Term Memory. *Neural Computation*, 9(8):1735-1780.

[5] Yang, Z., *et al*. (2016) Hierarchical Attention Networks for Document Classification. *Proceedings of NAACL-HLT 2016* 14801489.

[6] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014) Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15 1 1929-1958.

[7] Kingma, D. P. & Ba, J. (2014) Adam: A Method for Stochastic Optimization, arXiv:1412.6980.

Acknowledgement

We would like to give special thanks to professor Leon Bergen, who walked us through both basic concepts and high-level architectures of Natural Language Processing models in lecture, and offered us insights into this final project. We also appreciate to all the endeavors of course staffs for preparing this enlightening course.