Toxic Comment Classification

Pranjali Ingole, Akanksha Agarwal, Kratika Kothari, Ekta Agrawal

1 Problem Statement

Build a multi-headed model that's capable of detecting different types of of toxicity like threats, obscenity, insults, and identity-based hate.

The background for the problem originates from the multitude of online forums, where-in people participate actively and make comments. As the comments some times may be abusive, insulting or even hate-based, it becomes the responsibility of the hosting organizations to ensure that these conversations are not of negative type. The task is thus to build a model which could make prediction to classify the comments to belong to one or more of the following categories — toxic, severe-toxic, obscene, threat, insult or identity-hate with either approximate probabilities or discrete values (0/1).

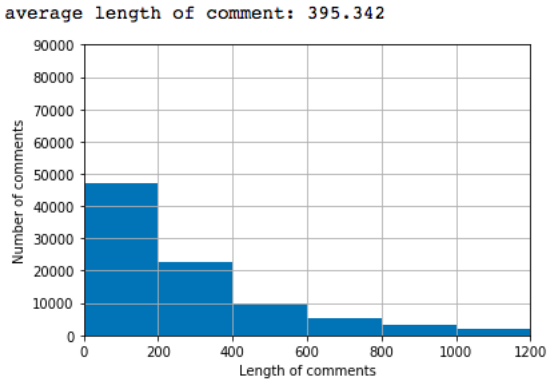
2 Data set

We have 65535 comments from Wikipedia's talk page edits 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

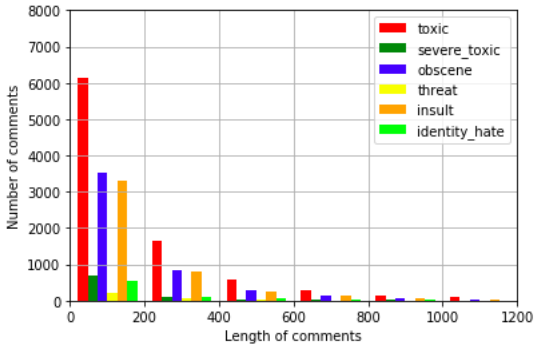
2.1 Data Analysis

Keeping length of comments on the independent axis with buckets of size 200, we counted the number of comments which had number of characters in that range. For instance, from the graph below we can see there are 10000 comments which have 400 to 600 characters.



It can be observed that comments are of varying lengths from less than 200 characters to 1200 characters. The majority of comments had length up to 200.

Further to understand the distribution of comments across the categories, we counted comments belonging to each of the different categories.



The graph plots the number of comments belonging to various categories. Toxic comments were highest in number, followed by obscene, insult, severe-toxic, identity-hate and threat.

3 Pre-Processing

1. Remove content specified in languages other than English.
2. Remove unnecessary data like HTML , XML tags, Wikipedia warnings
3. Punctuation, Numbers removal.
4. Stop word removal
5. Stemming and Lemmatizing

4 Word Embedding

- We have used Glove word embedding that are pre-trained on Stanford's Wikipedia data.
- Glove embedding vector size: 300
- For models that do not consider context such as Random Forest, SVM and Multi-Layer Perceptron, we have used average of word embedding of each word in Wikipedia comment as input to the model.
- For models that considers context such as LSTM, CNN, we have concatenated word embedding of each word in Wikipedia comment in same order as in the comment as input to the model.

5 Approach

We have implemented multiple supervised multi-label multi-class classifiers which will assign one or more labels out of six toxic labels mentioned above to each comment. If comment does not belong to any of the toxic label then it will be classified as non-toxic comment.

Below are the supervised models that we have implemented

5.1 Random Forest:

1. Input: Average Glove embedding for Wikipedia comment
2. 150 estimators
3. Output: Probabilities for each class

5.2 SVM:

Binary Relevance technique is used to train SVM for multi-label problem. It works by decomposing the multi-label learning task into a number of independent binary learning tasks (one per class label). SVM Configuration:

1. Input: Average Glove embedding for Wikipedia comment
2. Kernel: Linear, RBF
3. Regularization factor: 1
4. Output: Probabilities for each class

5.3 Multi-Layer Perceptron:

MLP Configuration details:

1. Input: Average Glove embedding for Wikipedia comment
2. Input Layer: 300 Neurons (embedding vector size)
3. Kernel Initializer: random uniform
4. No. of Hidden layers: 1
5. Activation function: Relu (hidden layer), softmax (output layer)
6. Loss function: Categorical Cross Entropy
7. Optimizer: RMSProp
8. Output Layer: 6 Neurons (no. of classes)
9. Output: Probabilities for each class

Hyper-Paramter Tuning using Cross Validation: Hyper-Parameter tuning is performed on MLP using 3-fold cross validation. Below parameter values are varied:

1. No. of neurons in hidden layer: 16, 32, 64
2. Learning rate of optimizer : 0.001, 0.002, 0.003
3. Batch Size: 64, 128
4. No. of epochs: 50, 100

Best model is obtained with below parameter values:

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
             estimator=<keras.wrappers.scikit_learn.KerasClassifier object at 0x7f28d7938748>,
             iid='warn', n_jobs=1,
             param_grid={'Learning_Rate': [0.001, 0.002, 0.003],
                         'No_Hidden_Neurons': [16, 32, 64],
                         'batch_size': [64, 128], 'epochs': [50, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=2)
Best score : 0.9941656065325153
Best params : {'Learning_Rate': 0.003, 'No_Hidden_Neurons': 16, 'batch_size': 64, 'epochs': 100}
```

5.4 LSTM:

LSTM configuration details:

1. Input: Glove embedding for wikipedia comments.
2. Embedding Layer: 20000 nodes(Maximum number of words in our vocabulary), 100(Embedding vector size)
3. Hidden Layers: 1 layer with 150 nodes
4. Output Layer: 7 nodes (1 node added as non-toxic)
5. Output: Probabilities of each class
6. Optimizer: Adam
7. Learning rate: 0.001
8. Loss Function: Sparse Categorical Cross Entropy
9. Activation Function: Softmax(output layer)
10. Batch size: 128
11. Number of epochs: 100

5.5 CNN:

CNN configuration details:

1. Input: Average Glove embedding for Wikipedia comment
2. Convolutional Layer: Number of filters:64, Kernel size:3
3. Global max pooling layer
4. Dense Layer with dropout
5. Output: Probabilities for each class

6 Evaluation Metric

To evaluate the model, We will use example based evaluation metrics where accuracy and loss is calculated for each example and then averaged across the test set.
As the given train data is very skewed i.e. very less percentage of total comments were actually toxic(less than 10%), We will use Hamming Loss or Log Loss as evaluation metric because they work well even with skewed data.

1. **Hamming Loss:** The hamming loss (HL) is defined as the fraction of the wrong labels to the total number of labels. For a multi-label problem, we need to decide a way to combine the predictions of the different labels to produce a single result. The method chosen in hamming loss is to give each label equal weight.

$$HL = \frac{1}{NL} \sum_{l=1}^{L} \sum_{i=1}^{N} Y_{i,l} \oplus X_{i,l},$$

Hamming Loss formula

2. **Log Loss:** Log Loss quantifies the accuracy of a classifier by penalizing false classifications. In order to calculate Log Loss the classifier must assign a probability to each class rather than simply yielding the most likely class.

$$-\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} y_{ij} \log p_{ij}$$

Log loss formula

7 Results:

MODEL	ACCURACY	HAMMING LOSS	LOG LOSS
Random Forest	89.65	3.90	0.42
SVM (Linear Kernel)	88.25	3.27	1.87
SVM (RBF Kernel)	87.55	3.32	1.67
Multi-layer Perceptron	79.49	5.62	1.12
LSTM	81.27	4.8	1.9
CNN	90.13	3.81	0.41