Toxic Comment Classification

Abhishek, MT18095
M-tech CSE
IIIT DELHI
abhishek19086@iiitd

Amrit, MT18022
M-tech CSE
IIIT DELHI
amrit18022@iiitd

Chirag, MT19089
M-tech CSE
IIIT DELHI
chirag19089@iiitd.ac.in

1 Problem Definition

The problem is about classifying the given comments into multiple toxic comment class label. Each label indicates how toxic the comment is i.e., toxicity level.

Dataset The dataset is taken from the kaggel. Dataset named toxic comment classification. The provided dataset for this problem contains 3 different files. First is, train.csv. It has 160k rows in which each row represents the class labels for each comment. Each data point has seven columns id, toxic, sever_toxic, obscene, threat, insult, identity_hate. The second file is, test.csv. It has only 2 columns in it i.e., id and comments. We are supposed to predict the label for these comments which are stored in this file. For the true value of these class labels and accuracy measure, the labels are given in another file named test_labels.csv. It has a column corresponding to every class label. Each column has only 2 possible values 0 or 1 in train data. 0 means that comment does not belong to the class and 1 means comment belongs to that class. In test_label -1 also a possible value which states that the corresponding comment is not used for labeling.

2 Problem Background

The comments are captured from Wikipedia. Wikipedia is a famous site where millions of people visit daily. The dataset contains approx 160k comments which is a large number of comments. The comments might have a different level of toxicity. Some might try to insult someone and some comments may intent to threaten any entity. It is a big task to classify those comments based on toxicity. This classification might help hide those comments or block some user who is not appropriate for such a big platform.

To tackle such a classification problem NLP(Natural Language Processing) plays a big role. NLP will help to understand the intent of the given comment by using some techniques like sentiment analysis, tf-idf, etc. Problem, we will try to solve this classification problem with various NLP and ML techniques like tf-idf based techniques, Naive-Bayes, Logistic Regression, CNN(Convolution Network), etc.

3 Simple Baseline Techniques

For the baseline model, n-gram based language model is used. We have trained and tested the model for all unigram, bigram and trigram.

3.1 Preprocessing

- In preprocessing, firstly Data points in which NaN or null values are are present those data point are removed.
- Then stop words are removed and as per the usage of model lemmatization and stemming are used for preprocessing.
- Stemming: Stemming algorithms work by cutting off the end or the beginning of the word, taking into account a list of common prefixes and suffixes that can be found in an inflected word. Used porter stemmer for implementing it in our code.
- Lematization: Lemmatization, on the other hand, takes into consideration the morphological analysis of the words. To do so, it is necessary to have detailed dictionaries which the algorithm can look through to link the form back to its lemma. Used Wornet lemmetizer for performing this lemmatizing task.
- Handling Multi-labeled Data : The given data is multi-labeled i.e., a single comment may belong to multiple classes at an instant. For

making it single labeled the class labels are merged. For example, if a comment belongs to both toxic and sever_toxic class then the new class label will be toxic:sever_toxic.

- Removing Garbage Data : The data given in test.csv there are some comments which have -1 as entry for all labels. This indicates that the data was never labeled and this data is not to be used for evaluating our model. That's why we drop such data points.

3.2 n-gram Based Language Model

- Unigram : In this part the language model is trained and tested with the usage of unigrams. The complete data including all the comments are divided into the unigrams for a particular class label. Then the probability measures are used to classify the comments into the given labels.
  Accuracy = 15.36%

- Bigram : In this part the language model is trained and tested with the usage of bigrams. The occuring frequency measures and probability measures are used to generate a language model using bigrams. This measures are used to classify the comments into the given class labels.
  Accuracy = 95.32%

- Trigram : In this part the language model is trained and tested with the usage of trigrams. The term frequency for each trigram is measured and probability for each trigram is also measured. These measures are used to generate a language model using trigrams and classify the comments into the given class labels.
  Accuracy = 97.53%

3.3 Naive-Bayes Classifier

In this approach on the basis of given data the probability is calculated for each and every comment being in the particular class on the basis of unigrams and sentence that class have. The probability is calculated as summation of the log of the fraction. The fraction is calculated as (1 + count of w in that class) divided by (total size of vocab of that class + vocab of whole data). This is also called as add-1 smoothing. This is done for handling the unseen data. As in the comments of test documents some unseen terms may appear and their count would be zero. So, they will make the whole probability 0.
Naive-Bayes label the comment with a class label with the class having maximum probability among all the probabilties which are calculated for each of the class.
Accuracy = 41.9743% It's accuracy is not that good because our data is multi-labeled but the naive-bayes will label the comment with only one class label.

4 Feature Engineering For advanced model

- Number of unique words in a comment : The count of unique words in a comment is measured for generating this feature. It will be a good feature because mostly when someone says something more toxic then he/she most probably say something which has repeated words more as compared to unique words.

- Number of words in a comment : When someone comments something toxic he/she tends to use fewer words.

- Length of comment : In toxic comments number of words is less but the length of the comment is either way too high or very less. The length means the alphabets each comment has.

- Number of punctuation : In toxic comments, the number of punctuation used will be less as compared to a non-toxic comment.

- Number of symbols or special symbols : Toxic comments have a very large number of special symbols in it. In toxic comment, the user tends to use "###" or "***" rather writing the complete word.

- Number of Stopwords : While writing the toxic comments the user most probably use less number of stopwords.

- Proportion of capital/small letters in a comment : In toxic comments, users tend to use similar type of letters i.e., either he/she will use the capital letter of whole comment or the small letters for the whole comment.

5 Other Techniques Applied

There are many classification and regression techniques which can be applied on the given data. We

have implemented some of those. Discussed as follows :

5.1 Tf-IDF based Cosine Similarity

The given data set is provided with a large number of Wikipedia comments which have been labeled by human raters for toxic behavior.The type of toxicity are
1 toxic
2 severe_toxic
3 obscene
4 threat
5 insult
6 identity_hate
We are given the all possible toxicity class of each comment. Our motive is to make tf-idf vctor of each type of toxicity class and find its cosine similarity.Then for a given sentence we find the cosine similarity of tf-idf vector sentence to tf-idf vector of each toxicity class vector.And class having highest cosine similarity for given sentence is class of that sentence.We label the sentence accordingly.

- **TF-IDF based cosine similarity** :-TF-IDF is short for term frequency–inverse document frequency. It is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. TF(Term Frequency) is taken as a ratio between simple count of that particular term in that particular question. IDF(Inverse Document Frequency) is taken as the log of the ratio between total number of questions and the number of questions in which that term appears. The formula used for calculation of tfidf is:

  tf = 1 + log(f)
  idf=log(N/(1+n))
  f = frquecy of given word in document
  N=total number of document
  n=# of document in which given word occurs

  A high weight in tf–idf is reached by a high term frequency (in the given document) and a low document frequency of the term in the whole collection of document.

- **Pre-processing to make tf-idf vector of toxic class**
  Here we make six documents each containing concatenated sentence of given class.So we have six documents each belonging to one class.These documents cover all the sentences in the training data.Than we make tf-idf vector of 6 document treating each document containing all words belonging to given class.

- **Finding Accuracy for tf-idf**
  As labeling given in test-label file also contains label in which a given sentence do not belong to any class i.e. all zero or all minus one.So first do labeling of test output and consider only those sentences to find accuracy which have one at least in one class.Here one in given class means that the sentence belong to that class.As in our data a sentence may belong to more than one class so we label the sentence with class which appears first i.e. Now we have both class label As we have class label of each test sentence and label find by cosine similarity so me easily find accuracy for given test data.

5.2 Decision Tree Classifier

Decision Tree is one of the most powerful and popular algorithm. Decision-tree algorithm falls under the category of supervised learning algorithms. It works for both continuous as well as categorical output variables.

- **Convert data into single class-labelled** As our data is multi-labelled i.e a test instance can belongs to multiple classes, so we have merged the class labels to get single-labelled class for each instance, This Process is done in two steps ://

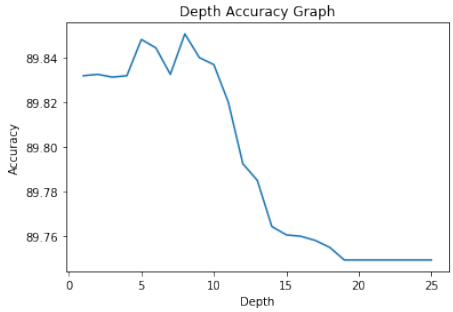
  First we have splitted class labels(in our case, from 6 to 58), i.e each instance belongs to only one class labels such that only the entry of class label under which instance belongs is 1 and rest all class label entries are 0 for that instance.//

  Then we merge the class labels into s single class label, such that for each instance, we find the class label for which it is 1 and assign the instance with that class label name.

- **Adding new Features** As the given dataset has only two features, one is 'id' and the other is 'comment text', so we are adding new features derived from the 'comment text' in order to make classifier more accurate

- **Convert dataset into encoded data** we are converting our dataset into labelled data sing the labelled encoder before applying the classifier.

- **Apply the Decision Classifier Model** we are applying the sklearn Decision tree classsifier in order to classify the data and using the gini index as measure of impurity in gini index.
  We are predicting the class labels on various depths of decision tree ([1:25]).
  For each Depth, we are splitting our encoded dataset into train and test part, train part is used to train the decision classifier model and the trained model is applied on the test part to predict the class labels.The splitting is done multiple times, for each split we calculate the accuracy then mean of all accuracies of each split for a particular depth
  Then we plot the accuracy graph for each depth, her is the plot of Depth vs Accuracy values :



5.3 Support Vector Machine

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimentional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

The Pre-processing steps that we have used in Decision tree are also applied here before applying the SVM model, in order to make data more meaningful.
There are three type of Parameters that we can tune into SVM model,described below:

- **Linear Kernel** The learning of the hyperplane in linear SVM is done by transforming the problem using some linear algebra. This is where the kernel plays role. For linear kernel the equation for prediction for a new input using the dot product between the input (x) and each support vector (xi) is calculated as follows: f(x) = B(0) + sum(ai * (x,xi))

- **Regulization** The Regularization parameter (often termed as C parameter in python's sklearn library) tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points.

- **Gamma** The gamma parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. In other words, with low gamma, points far away from plausible seperation line are considered in calculation for the seperation line. Where as high gamma means the points close to plausible line are considered in calculation.

- **Apply the Model** we have defined two terms regularization parameter and gamma. These are tuning parameters in SVM classifier. Varying those we can achieve considerable non linear classification line with more accuracy in reasonable amount of time. One more parameter is kernel. It defines whether we want a linear of linear separation. But this linear kernel generally takes more time than the others and gives the accurate hyperplane. In our SVM classifier, we are passing the 'gamma' as parameter to SVM Model which can do misclassification for some of the instance, but it is taking less time to train the model with this gamma parameter.
  We have splitted the train dataset into training and testing parts, and learn the SVM Model in training part and apply the model in the Testing part to predict class labels of instances in testing part. Then we evaluate the accuracy score.
  And We are getting the accuracy : 0.8990432820855615

5.4 Logistic Regression

- **Logistic Model** In statistics, the logistic model (or logit model) is used to model the probability of a certain class or event existing such as pass/fail, win/lose, alive/dead or healthy/sick. This can be extended to model several classes of events such as determining whether an image contains a cat, dog, lion, etc... Each object being detected in the image would be assigned a probability between 0 and 1 and the sum adding to one.

- **Pre-processing**
  1. We have three files, train, test, Submission
  2. We will load train and test file as pandas dataframe, and add a new Attribute to train dataframe represents Non-Toxic i.e if a particular instance is not Toxic i.e doesn't belongs to any toxic class, the Non-toxic attribute value will be 1 for that instance, otherwise 0.
  3. Check for any None Values in train data
  4. Then we will Apply TfidfVectorizer methods to find doc vectors of commenttext of train(as trainvector) and test data(as testvector).
  5. Then We apply LogisticRegression method on testvectors to find the probabilities of belonging to a particular class for each instance.
  6. Here are the accuracies of multiple instances for these six classes :

id	toxic	severe_toxic	obscene	threat	insult	identity_hate
00001cee341fdb12	0.999987829	0.106263793	0.999986693	0.002368913	0.962577866	0.094955729
0000247867823ef7	0.002872896	0.000603644	0.001893376	0.00010037	0.002226837	0.000342311
00013b17ad220c46	0.011754996	0.000863802	0.005587902	0.000101699	0.003209674	0.000296718
00017563c3f7919a	0.000960254	0.00022388	0.001141406	0.000170571	0.001056735	0.000297366
00017695ad8997eb	0.009957418	0.000484707	0.002009037	0.000131261	0.002395143	0.000351449
0001ea8717f6de06	0.004427786	0.000280444	0.001964433	0.000372824	0.003182199	0.000364454
00024115d4cbde0f	0.000535025	0.000155547	0.000825363	0.000101738	0.000704483	0.000410841
000247e83dcc1211	0.187793255	0.000365585	0.003324076	0.000132346	0.007759241	0.000341969
00025358d4737918	0.009362863	0.000135086	0.002518447	9.54E-05	0.003299132	0.000393867

5.5 Convolution Neural Network

CNNs, are similar to neural networks. These are made up of neurons with learnable weights and biases. Each neuron receives several inputs, takes a weighted sum over them, pass it through an activation function and responds with an output. Here, in our implementation there are several layers used such as embedding layer, 1D-convolution layer etc.

- CNN is trained with one input layer, one 1-D convolution layer, two hidden layers and one output layer to predict the output.

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 500)               0
_________________________________________________________________
embedding_1 (Embedding)      (None, 500, 240)          492480
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 500, 100)          96100
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 125, 100)          0
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 50)                5050
_________________________________________________________________
dropout_1 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 306
=================================================================
Total params: 593,936
Trainable params: 593,936
Non-trainable params: 0
```

Figure 1: Layers implemented in CNN

- **epochs:**The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset. In current approach, epochs are taken as 5.

```
143613/143613 [==============================] - 953s 7ms/step - loss: 0.0791 - acc: 0.9756 - val_loss: 0.0602 - val_acc: 0.9796
Epoch 2/5
143613/143613 [==============================] - 943s 7ms/step - loss: 0.0630 - acc: 0.9790 - val_loss: 0.0569 - val_acc: 0.9803
Epoch 3/5
143613/143613 [==============================] - 944s 7ms/step - loss: 0.0593 - acc: 0.9798 - val_loss: 0.0547 - val_acc: 0.9806
Epoch 4/5
143613/143613 [==============================] - 945s 7ms/step - loss: 0.0561 - acc: 0.9805 - val_loss: 0.0530 - val_acc: 0.9811
Epoch 5/5
143613/143613 [==============================] - 939s 7ms/step - loss: 0.0546 - acc: 0.9808 - val_loss: 0.0539 - val_acc: 0.9814
```

Figure 2: Training at different epochs

- **batch size:**The batch size defines the number of samples that will be propagated through the network.Our batch size is 32.

- **Optimizer:**Optimization algorithms helps us to minimize (or maximize) an Objective function (another name for Error function) E(x) which is simply a mathematical function dependent on the Model's internal learnable parameters which are used in computing the target values(Y) from the set of predictors(X) used in the model. In current approach, Adam optimizer is used.

- **Neuron Activation function:**Activation function decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it. The purpose of the activation function is to introduce non-linearity into the output of a neuron. In current approach, relu is used for hidden layers and sigmoid for output layer.

It shows the accuracy of 97%. because of the label given in our test data are not in appropriate format.

6 Experimental Results :

The accuracy for applied models are as follows :

Model Applied	Accuracy
N-gram based Language Model	Uni- 15.63%, Bi-95.32%, Tri-97.53%
Tf-idf Based cosine similarity	96%
Support Vector Machine	88.99%
Decision Tree Classifier	88.2%
Convolution Neural Network	97%
Naive-Bayes	47.9734%

Figure 3: Accuracy achieved by different models

7 Conclusion :

As per the accuracy values all the model shows a recommendable accuracy but that is because of the type of data given. As we generate some new features and applied the support vector machine , decision tree classifier, cosine based similarity on that data. SVM shows the high accuracy as compared to decision tree. Logistic Regression, convolution neural network provides the probability of each comment being in a particular class. Which was the motive to infer that how toxic a comment is. So, we can consider that probability as a toxicity metric.

References

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