Toxic Comment Classification

- Team Narang -

Project Report

By:

Name	Roll Number
Bhavesh Gulecha	MT2018024
Shubham Darokar	MT2018113
Tushar Narang	MT2018127

International Institute of Information Technology Bangalore

Contents

Chapter 1

Introduction

1.1 Problem Statement

Discussing things you care about can be difficult. The threat of abuse and harassment online means that many people stop expressing themselves and give up on seeking different opinions. Platforms struggle to effectively facilitate conversations, leading many communities to limit or completely shut down user comments.

One area of focus is the study of negative online behaviors, like toxic comments (i.e. comments that are rude, disrespectful or otherwise likely to make someone leave a discussion). So far range of publicly available models served through the Perspective API, including toxicity. But the current models still make errors, and they don't allow users to select which types of toxicity they're interested in finding (e.g. some platforms may be fine with profanity, but not with other types of toxic content).

In this competition, We are challenged to build a multi-headed model that's capable of detecting different types of of toxicity like threats, obscenity, insults, and identity-based hate better than Perspective's current models.

1.2 Data Description

Given a group of sentences or paragraphs, used as a comment by a user in an online platform, classify it 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).

In multi-label classification, data can belong to more than one label simultaneously. For example, in our case a comment may be toxic, obscene and insulting at the same time. It may also happen that the comment is non-toxic and hence does not belong to any of the six labels.

1.2.1 Dataset Labels

toxic
severe_toxic
obscene
threat
insult
identity_hate

Total Row in Training Data:149571
Total Row in Test Data:10000

Chapter 2

Exploratory Data Analysis (EDA)

Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns,to spot anomalies,to test hypothesis and to check assumptions with the help of summary statistics and graphical representations. It is a good practice to understand the data.

2.1 Label Distribution

Here we find count of labels in dataset to see which labels are imbalance and what is count of each label.

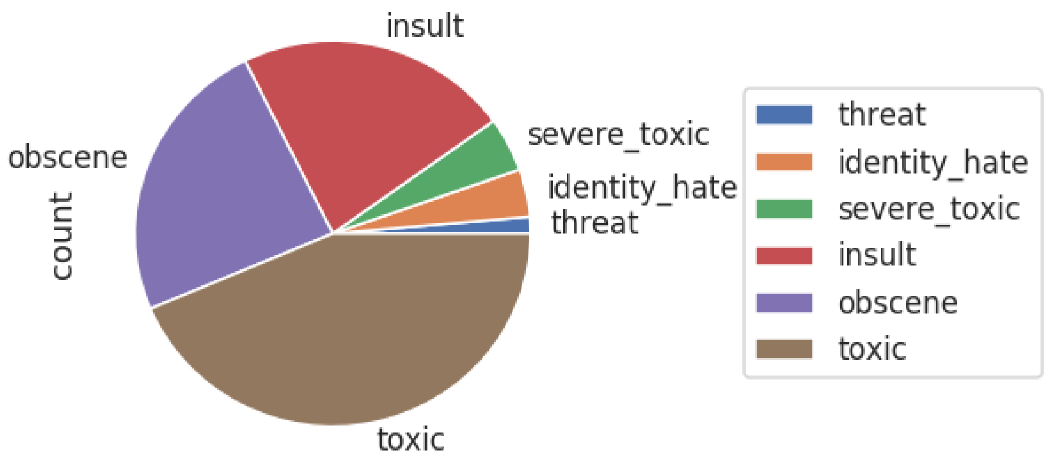


Figure 2.1: Label Distribution Without None Category

Threat is very imbalanced, which may make it a good candidate for upsampling.

Type	Count
none	134351
toxic	14352
obscene	7914
insult	7366
severe_toxic	1493
identity_hate	1314
threat	458

Table 2.1: Label And Its Count

The three major labels are :

1. toxic
2. obscene
3. insult

We can see several things :

1. As expected, the 'none' label is clearly ahead with 134351 comments
2. 'toxic', is the first 'real' label & is coming in all combination from rank 1 to 6

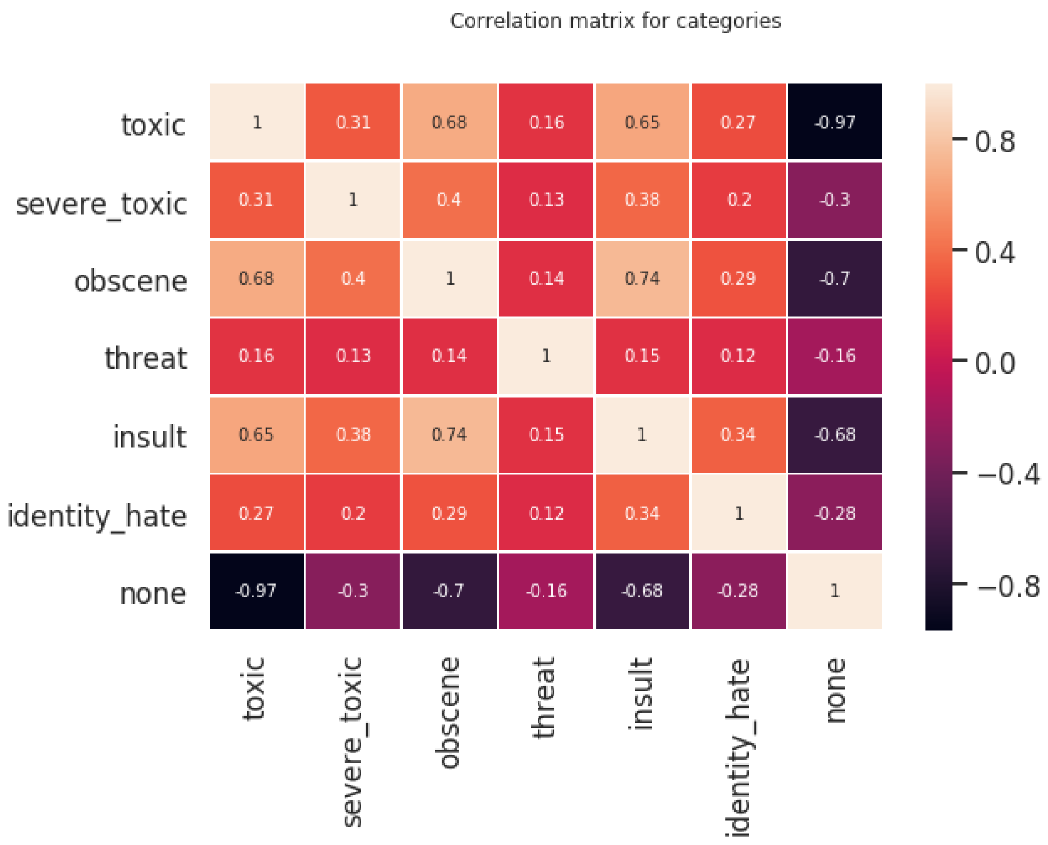
2.2 Correlation Between Variables

Correlation Matrix is used to investigate the dependency between multiple variables at the same time. The result is a table containing the correlation coefficients between each variable and the others.

A correlation matrix is a table showing correlation coefficients between sets of variables. Each random variable (X_i) in the table is correlated with each of the other values in the table (X_j). This allows you to see which pairs have the highest correlation.

The correlation matrix shows interesting things :

1. 'toxic' is clearly correlated with 'obscene' and 'insult' (0.68 and 0.65)
2. 'toxic' and 'severe_toxic' are only got a 0.31 correlation factor
3. 'insult' and 'obscene' have a correlation factor of 0.74



	toxic	severe_toxic	obscene	threat	insult	identity_hate	none
toxic	1	0.31	0.68	0.16	0.65	0.27	-0.97
severe_toxic	0.31	1	0.4	0.13	0.38	0.2	-0.3
obscene	0.68	0.4	1	0.14	0.74	0.29	-0.7
threat	0.16	0.13	0.14	1	0.15	0.12	-0.16
insult	0.65	0.38	0.74	0.15	1	0.34	-0.68
identity_hate	0.27	0.2	0.29	0.12	0.34	1	-0.28
none	-0.97	-0.3	-0.7	-0.16	-0.68	-0.28	1

Figure 2.2: Correlation Matrix

2.3 Word Cloud

Technique used for representing text data in which the size of each word indicates its frequency or importance.Significant textual data points can be highlighted using a word cloud.

Example 2.1 (Snipet)
Here is Code Snipet for Word Cloud

$$\begin{aligned} wordcloud =& WordCloud(background\_color =' black', max\_words = 200, \\ & max\_font\_size = 100, random\_state = 4561 \\ &).generate\_from\_frequencies(wc.to\_dict()['count']) \end{aligned} \tag{2.1}$$

.

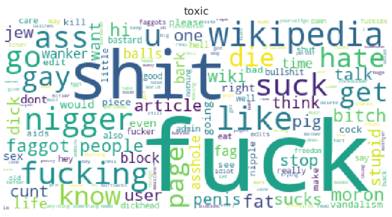


Figure 2.3: Toxic

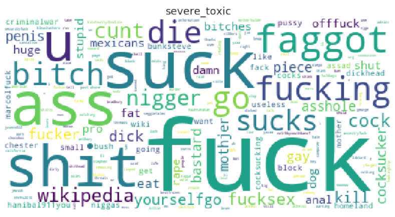


Figure 2.4: Severe_toxic

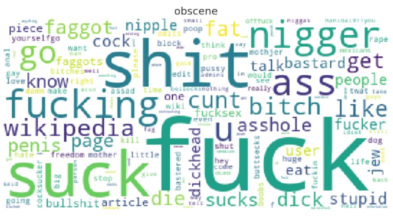


Figure 2.5: Obscene

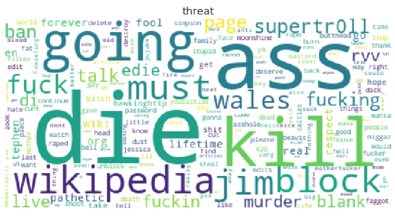


Figure 2.6: threat

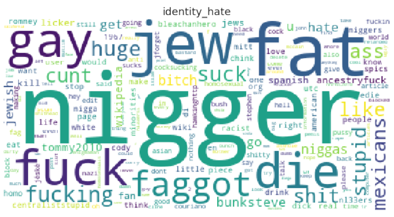


Figure 2.7: identity_hate

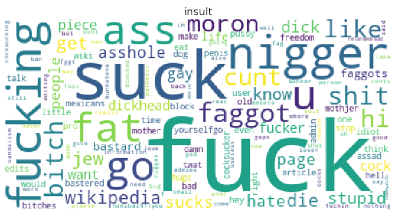


Figure 2.8: insult

Things Observed in our Data:

The vocabulary used in all categories is quite similar (expect for 'none' of course).Frequencies are varying a bit across (for example 'fuck' and 'suck')

Chapter 3

Data Pre-Processing

3.1 TF-IDF

We chose word frequency here to represent text features. However, Inverse Comment Frequency can be applied to Term Frequency Matrix to furthur improve our classifier.

Term Frequency (TF) is a scoring of the frequency of the word in the current Comment, whereas Inverse Document Frequency (IDF) is a scoring of how rare the word is across comments.

Why do we need to find rare words ? Terms that appear across many comments are less discriminating. TFIDF assigns weightage to words wrt other words in comment.

TF-IDF not only counts the frequency of a term in the given comment), but also reflects the importance of each term to the comment by penalizing frequently appearing terms(words) in most samples.

Example 3.1 (Math)
Formulas

$$\begin{aligned}TF(t) &= \text{No. of times term t appears in a comment /Total No. of terms in the comment}\\ IDF(t) &= \text{log(No. of comments in Term Frequency matrix/No. of comment term t appear)}\\ Value &= TF * IDF\end{aligned} \tag{3.1}$$

3.2 Word and Character Vectorizer

Example 3.2 (Code)
Code Snipet

$$\begin{aligned} word\_vectorizer = TfidfVectorizer(& \\ & sublinear\_tf = True, \\ & strip\_accents =' unicode', \\ & analyzer =' word', \\ & token\_pattern = r'1,', \\ & stop\_words =' english', \\ & ngram\_range = (1,2), min\_df = 2, max\_df = 0.5, \\ & max\_features = 60000 \\ &) \end{aligned} \tag{3.2}$$

Example 3.3 (Char Vectorizer Snipet)
Code Snipet

$$\begin{aligned} char\_vectorizer = TfidfVectorizer(& \\ & sublinear\_tf = True, \\ & strip\_accents =' unicode', \\ & analyzer =' char', \\ & stop\_words =' english', \\ & ngram\_range = (2,6), min\_df = 2, max\_df = 0.5, \\ & max\_features = 60000 \\ &) \end{aligned} \tag{3.3}$$

3.2.1 TFIDF Parameters

1)analyzer : string, ‘word’, ‘char’ or callable
Whether the feature should be made of word or character n-grams.

If a callable is passed it is used to extract the sequence of features out of the raw, unprocessed input.
.

2)stop_words : string ‘english’, list, or None (default)
If a list, that list is assumed to contain stop words, all of which will be removed from the resulting tokens. Only applies if analyzer == ’word’.

.

3)max_features : int or None, default=
If not None, build a vocabulary that only consider the top max_features ordered by term frequency across the corpus.We use max_features=60000

.

4)ngram_range : tuple (min_n, max_n)
The lower and upper boundary of the range of n-values for different n-grams to be extracted. All values of n such that min_n <= n <= max_n will be used.

.

5)max_df : float in range [0.0, 1.0] or int, default=1.0
When building the vocabulary ignore terms that have a comment frequency strictly higher than the given threshold.If float, the parameter represents a proportion of comments, integer absolute counts. This parameter is ignored if vocabulary is not None.We used max_df=0.5

.

6)min_df : float in range [0.0, 1.0] or int, default=1
When building the vocabulary ignore terms that have a comment frequency strictly lower than the given threshold.If float, the parameter represents a proportion of comment, integer absolute counts. This parameter is ignored if vocabulary is not None.We Used min_df=2

3.2.2 TFIDF Function Used:

1)transform(raw_documents, copy=True)[source]

Transform comments to comment-term matrix.
Uses the vocabulary and comment frequencies (df) learned by fit (or fit_transform).

Parameters: raw_documents : iterable an iterable which yields either str, unicode or file objects

copy : boolean, default True Whether to copy X and operate on the copy or perform in-place operations.

Returns:

X : sparse matrix, [n_samples, n_features]
Tf-idf-weighted comment-term matrix.

Chapter 4

Model Building

4.1 Models Used

Multiple models were trained on the data set and finally does weighted average of 5 models we have. The following models were selected for the average ensembling:

- Ridge Regression
- Logistic Regression
- Random Forest
- SGD (Stochastic gradient descent)
- XGBoost (eXtreme Gradient Boosting)

4.1.1 Ridge Regression

Ridge Regression is a technique for analyzing multiple regression data for ill-posed problems, which are problems that do not have a unique solution. These problems suffer from multicollinearity. In such cases, least squares estimates are unbiased, but their variances may become so large that they may be far from the true value. By adding a degree of bias to the regression estimates, ridge regression reduces the standard errors. In other words, this technique prevents learning a more complex model, so as to avoid the risk of over-fitting.

Minimizes the objective function:

$$||y - Xw||_2^2 + alpha * ||w||_2^2$$

Parameter Tuning:

alpha : { float, array-like } , shape (n_targets) Regularization strength; must be a positive float. Regularization improves the conditioning of the problem and reduces the variance of the estimates. The value of alpha selected was 29

fit_intercept : boolean Whether to calculate the intercept for this model. If set to false, no intercept will be used in calculations (e.g. data is expected to be already centered). The value for our model was set to true.

max_iter : int, optional Maximum number of iterations for conjugate gradient solver. In our model the value of max_iter was set to 150.

tol : float Precision of the solution. The value of tol was set to 0.0025

solver : {'auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga'} Solver to use in the computational routines In our model solver was set as 'sag'.

Result of Ridge on our project: In our project, the Ridge Regression gave the best result. The AUC-ROC obtained was **0.9869**

4.1.2 Logistic Regression

Logistic Regression is a supervised learning algorithm. It measures the relationship between the categorical dependent variable i.e. the target variable and one or more independent variables that are the by estimating probabilities using a logistic/sigmoid function.

Parameters Tuning:

C : float, default: 1.0 Inverse of regularization strength; must be a positive float. Smaller values specify stronger regularization.

solver : str, {'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'}, default: 'liblinear'. Algorithm to use in the optimization problem. The value was set to 'sag'.

verbose : int, default: 0 For the liblinear and lbfgs solvers set verbose to any positive number for verbosity.

Result of logistic Regression on our model: The AUC-ROC obtained was **0.9859**

4.1.3 Random Forest

Decision trees are a popular method for various machine learning tasks. Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance. A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

Parameters Tuning:

n_estimators : integer, optional (default=10) The number of trees in the forest. The value of n_estimators was set to 1000.

max_depth : integer or None, optional (default=None) The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

max_leaf_nodes : int or None, optional (default=None) Grow trees with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. The value of max_leaf_nodes was set to 18.

random_state : int, RandomState instance or None, optional (default=None) If int, random_state is the seed used by the random number generator;The value of random state was set to 21.

Result of Random forest on our model: The model gave AUC-ROC of **0.9687**

4.1.4 Stochastic Gradient Descent(SGD)

Stochastic gradient descent is also known as incremental gradient descent. It is an iterative method for optimizing a differentiable objective function, a stochastic approximation of gradient descent optimization. It is called stochastic because samples are selected randomly (or shuffled) instead of as a single group (as in standard gradient descent) or in the order they appear in the training set.

Parameters Tuning:

penalty : str, ‘none’, ‘l2’, ‘l1’, or ‘elasticnet’ The penalty (aka regularization term) to be used. Defaults to ‘l2’ which is the standard regularizer for linear SVM models. The value of penalty was set to ’l2’.

alpha : float Constant that multiplies the regularization term. Defaults to 0.0001 Also used to compute learning_rate when set to ‘optimal’. The value of set to 0.001 .

max_iter : int, optional The maximum number of passes over the training data (aka epochs). The value of max_iter was set to 200.

tol : float or None, optional The stopping criterion. If it is not None, the iterations will stop when (loss > previous_loss - tol). The value of tol was set to 0.20 .

random_state : int, RandomState instance or None, optional (default=None) The seed of the pseudo random number generator to use when shuffling the data. If int, random_state is the seed used by the random number generator. The value of random state was set to 42.

learning_rate : string, optional The learning rate schedule. The value of learning_rate was set to ’optimal’.

Result of SGD on our model: The AUC-ROC obtained was **0.9851**

4.1.5 XGBoost (eXtreme Gradient Boosting)

XGBoost (eXtreme Gradient Boosting) is an advanced implementation of gradient boosting algorithm. XGBoost is also known as ‘regularized boosting‘ technique because Standard GBM implementation has no regularization like XGBoost, therefore XGBoost helps to reduce overfitting.

Parameters Tuning:

max_depth [default=6] The maximum depth of a tree, same as GBM. Used to control over-fitting as higher depth will allow model to learn relations very specific to a particular sample. Should be tuned using CV. Typical values: 3-10. The value was set as 5.

subsample [default=1] Denotes the fraction of observations to be randomly samples for each tree. Typical values: 0.5-1. The value of subsample was set to 0.8

colsample_bytree [default=1] Denotes the fraction of columns to be randomly samples for each tree. Typical values: 0.5-1. The value of colsample_bytree was set to 0.8

objective [default=reg:linear] This defines the loss function to be minimized. The value of objective was set as binary:logistic

eval_metric [default according to objective] The metric to be used for validation data. The default values are rmse for regression and error for classification. The value of eval_metric was set as auc

seed [default=0] The random number seed. Can be used for generating reproducible results and also for parameter tuning. The value of seed was set as 23

Result of XGBoost on our model: The model gave AUC-ROC value of **0.9782**

4.2 Weighted Average Ensembling

All models are assigned different weights defining the importance of each model for prediction.Multiple predictions are made for each data point in averaging. In this method, we take an weighted average of predictions from all the models and use it to make the final prediction.

```
import pandas as pd
import numpy as np



logistic = pd.read_csv('/home/bhavesh/Desktop/Toxic Comment/logistic.csv')
random = pd.read_csv('/home/bhavesh/Desktop/Toxic Comment/random.csv')
xgboost = pd.read_csv('/home/bhavesh/Desktop/Toxic Comment/xgboost.csv')
ridge= pd.read_csv('/home/bhavesh/Desktop/Toxic Comment/ridge.csv')
sgb=pd.read_csv('/home/bhavesh/Desktop/Toxic Comment/submissionsgd.csv')

b1 = ridge.copy()
col = ridge.columns

col = col.tolist()
col.remove('id')
for i in col:
    b1[i] = (ridge[i] * 7 + logistic[i]
        *6+random[i]*3+xgboost[i]*3+sgb[i]*2) / 21

b1.to_csv('blendbg.csv', index = False)
```

Chapter 5

Conclusion

1) We have used Ridge,Logistic Regression,Stochastic Gradient Descent,XGBoost and Random Forest but we got best individual result with Ridge.

2)We tried other models like lasso,LGBM and SVM but we didn't get the expected results with them. We got AUC-ROC 0.5 ,0.92 and 0.87 respectively.

3) With Weighted Average Ensembling technique, we used all 5 models with different weights according to AUC-ROC value.so we gave higher weight to ridge and lower weight to random forest.

4)Models and Their AUC-ROC Value:

Model	AUC-ROC
Ridge	0.9870
Logistic Regression	0.9859
SGD	0.9850
XGBOOST	0.9782
Random Forest	0.9682

Table 5.1: Models and AUC-ROC

5) With Our Dataset which is somewhat already clean , we didn't get expected results with further cleaning and feature extraction like (length_comment, no_of_fwords etc.)

6) Position on Leader Board

Leaderboard	Rank
Public	2
Private	4

Bibliography

[1] A Comprehensive Guide to Ensemble Learning (with Python codes)
`https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/`

[2] sklearn.linear_model.Ridge
`https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html`.

[3] Calculation and Visualization of Correlation Matrix with Pandas
`https://datascience.stackexchange.com/questions/10459/calculation-and-visualization-of-correlation-matrix-with-pandas`.