Toxic Comment Classification

Introduction

"This project was taken from kaggle's Toxic Comment Classification Challenge"

The threat of abuse and harassment online is very real. Due to which people stop expressing themselves and give up on asking opinions.

Platforms struggle to effectively facilitate conversations, leading many communities to limit or completely shut down user comments.

The area of focus of the competition 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). 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.

We are provided with a large number of Wikipedia comments which have been labeled by human raters for toxic behavior. The types of toxicity are:

```
toxic
severe_toxic
obscene
threat
insult
identity_hate
```

We must create a model which predicts whether each comment belongs to each type of toxicity. This is a multilabel classification. Each comment can belong to more than one type of toxicity.

File Descriptions:

```
train.csv - the training set, contains comments with their binary labels
test.csv - the test set, you must predict the toxicity probabilities for these comments. To deter hand l
sample_submission.csv - a sample submission file in the correct format
test_labels.csv - labels for the test data; value of -1 indicates it was not used for scoring; (Note: fi
```

Lots load the necessary libraries

```
if (!require(tidyverse)) install.packages('tidyverse')
if (!require(tidytext)) install.packages('tidytext')
if (!require(DT)) install.packages('DT')
if (!require(tm)) install.packages('tm')
if (!require(SnowballC)) install.packages('SnowballC')
if (!require(caret)) install.packages('caret')
if (!require(Metrics)) install.packages('Metrics')
if (!require(pROC)) install.packages('pROC')
if (!require(MLmetrics)) install.packages('MLmetrics')
if (!require(e1071)) install.packages('e1071')
```

```
if (!require(xgboost)) install.packages('xgboost')
if (!require(wordcloud)) install.packages('wordcloud')
if (!require(readr)) install.packages('readr')

library(tidyverse)
library(tidytext)
library(DT)
library(stringr)
library(tm)
library(SnowballC)
library(caret)
library(Metrics)
library(pROC)
library(MLmetrics)
library(e1071)
library(xgboost)
library(wordcloud)
library(readr)
```

Note: For running the code in less time, uncomment the lines as mentioned in the middle of the code which allows subsetting the data appropriately. Then you can be running the model on a lesser data.

For downloading the kaggle dataset, it needs user to be enrolled in the competition in kaggle and accept its rules. This might be cumbersome when some one new decides to run the code. So will be downloading the data via a github link.

```
testpath <- "https://raw.githubusercontent.com/malyada/toxic_comment_classification/main/test.csv"
trainpath <- "https://raw.githubusercontent.com/malyada/toxic_comment_classification/main/train.csv"
sample_submissionpath <- "https://raw.githubusercontent.com/malyada/toxic_comment_classification/main/sa
test_labelspath <- "https://raw.githubusercontent.com/malyada/toxic_comment_classification/main/test_lab

test <- read_csv(testpath)
train <- read_csv(trainpath)
sample_submission <- read.csv(sample_submissionpath)
test_labels <- read.csv(test_labelspath)
```

Exploratory Data Analysis(EDA)

First step of EDA is cleaning the data. Let us remove all possible punctuation from the comment_text, and also remove rows whose comment_text is not available.

```
#removing punctuation
train$comment_text=gsub("'|\"|'|"|"|\"|\n|,|\\.|…|\\?|\\+|\\-|\\/|\\=|\\(|\\)|‘", "",
                        train$comment_text)
test$comment_text=gsub("'|\"|'|"|"|\"|\n|,|\\.|…|\\?|\\+|\\-|\\/|\\=|\\(|\\)|‘", "",
                       test$comment_text)
#removing texts which are NA
train <- train[!(is.na(train$comment_text)), ]
test <- test[!(is.na(test$comment_text)), ]
```

Lets us look at the data distribution among the classes.

```
cat('no. of observations:',(dim(train)[1]), "\n")
```

```
## no. of observations: 159571
```

```
cat('no. of toxic observations:', sum(train$toxic == 1), "\n")
```

```
## no. of toxic observations: 15294
```

```
cat('no. of severe_toxic observations:', sum(train$severe_toxic == 1), "\n")
```

```
## no. of severe_toxic observations: 1595
```

```
cat('no. of obscene observations:', sum(train$obscene == 1), "\n")
```

```
## no. of obscene observations: 8449
```

```
cat('no. of threat observations:', sum(train$threat == 1), "\n")
```

```
## no. of threat observations: 478
```

```
cat('no. of insult observations:', sum(train$insult == 1), "\n")
```

```
## no. of insult observations: 7877
```

```
cat('no. of identity_hate observations:', sum(train$identity_hate == 1), "\n")
```

```
## no. of identity_hate observations: 1405
```

```
cat('no category observations:', sum((train$toxic == 0) & (train$severe_toxic == 0) & (train$obscene ==
            (train$threat == 0) & (train$insult == 0) & (train$identity_hate == 0)), "\n")
```

```
## no category observations: 143346
```

We see that the data is highly skewed. No label data is 90% of the training data, the largest toxic class has 10% number of observations and the smallest toxic class has .003% of the observations

Lets look at the most popular words

```
head(train %>%
unnest_tokens(output = 'word', token = 'words', input = comment_text) %>% mutate(word <- wordStem(word))
anti_join(stop_words) %>%
count(word, sort = T), 15)
```

```
## # A tibble: 15 x 2
##    word            n
##    <chr>       <int>
##  1 article     53873
##  2 page        43994
```

```
##  3 wikipedia   34725
##  4 talk        31809
##  5 dont        25465
##  6 im          18453
##  7 people      17536
##  8 edit        17478
##  9 articles    17044
## 10 time        15135
## 11 information 11868
## 12 sources     10979
## 13 deletion    10736
## 14 pages       10229
## 15 editing     10074
```

The popular words seem fair because more than 90% of the sentences are good comments and do not belong to any toxic class.

Let us look at the wordcloud of the most frequent words for each toxic class.

```
set.seed(1234) # for reproducibility
#toxic word cloud
tmp <- train %>%
  filter(toxic == 1) %>%
  unnest_tokens(output = 'word', token = 'words', input = comment_text) %>%
  mutate(word <- wordStem(word)) %>%
  anti_join(stop_words) %>%
  count(word, sort = T)

wordcloud(words = tmp$word, freq = tmp$n, min.freq = 1, max.words=200, random.order=FALSE, rot.per=0.35
          colors=brewer.pal(8, "Dark2"), scale=c(4, 0.75))
```


```
#severe_toxic word cloud
tmp <- train %>%
  filter(severe_toxic == 1) %>%
  unnest_tokens(output = 'word', token = 'words', input = comment_text) %>%
  mutate(word <- wordStem(word)) %>%
  anti_join(stop_words) %>%
  count(word, sort = T)

wordcloud(words = tmp$word, freq = tmp$n, min.freq = 1, max.words=200, random.order=FALSE, rot.per=0.35
          colors=brewer.pal(8, "Dark2"), scale=c(4, 0.75))
```

```
#obscene word cloud
tmp <- train %>%
  filter(obscene == 1) %>%
  unnest_tokens(output = 'word', token = 'words', input = comment_text) %>%
  mutate(word <- wordStem(word)) %>%
  anti_join(stop_words) %>%
  count(word, sort = T)

wordcloud(words = tmp$word, freq = tmp$n, min.freq = 1, max.words=200, random.order=FALSE, rot.per=0.35
          colors=brewer.pal(8, "Dark2"), scale=c(4, 0.75))
```


```
#threat word cloud
tmp <- train %>%
  filter(threat == 1) %>%
  unnest_tokens(output = 'word', token = 'words', input = comment_text) %>%
  mutate(word <- wordStem(word)) %>%
  anti_join(stop_words) %>%
  count(word, sort = T)

wordcloud(words = tmp$word, freq = tmp$n, min.freq = 1, max.words=200, random.order=FALSE, rot.per=0.35
          colors=brewer.pal(8, "Dark2"), scale=c(4, 0.75))
```


```
#insult word cloud
tmp <- train %>%
  filter(insult == 1) %>%
  unnest_tokens(output = 'word', token = 'words', input = comment_text) %>%
  mutate(word <- wordStem(word)) %>%
  anti_join(stop_words) %>%
  count(word, sort = T)

wordcloud(words = tmp$word, freq = tmp$n, min.freq = 1, max.words=200, random.order=FALSE, rot.per=0.35
          colors=brewer.pal(8, "Dark2"), scale=c(4, 0.75))
```


```
#identity_hate word cloud
tmp <- train %>%
  filter(identity_hate == 1) %>%
  unnest_tokens(output = 'word', token = 'words', input = comment_text) %>%
  mutate(word <- wordStem(word)) %>%
  anti_join(stop_words) %>%
  count(word, sort = T)
```

```
wordcloud(words = tmp$word, freq = tmp$n, min.freq = 1, max.words=200, random.order=FALSE, rot.per=0.35
          colors=brewer.pal(8, "Dark2"), scale=c(4, 0.75))
```

We notice that there are many high frequency words in each toxic classes, that belongs to more than one class.

Lets get the data into the (more reliable than just frequency) tfidf per word per sentence format to make further analysis of the data per class. In short for each sentences we take tokens, stem and remove stopwords and for each word we find tf*idf

tf: frequency of the term in the document

idf: log(n/nt)

n: number of documents

nt: no of documents containing the term.

We arrange this in the sparse document term matrix form where each row is a observation, each column is a word and the value across them is the tfidf.

```
train$comment_text = iconv(train$comment_text, 'UTF-8', 'ASCII')
train$comment_text=str_replace_all(train$comment_text,"[^[:graph:]]", " ")
test$comment_text = iconv(test$comment_text, 'UTF-8', 'ASCII')
test$comment_text=str_replace_all(test$comment_text,"[^[:graph:]]", " ")
```

```
#creating a document term matrix with TfIdf for dataset and datasetTest

corpus = VCorpus(VectorSource(train$comment_text))

corpus = tm_map(corpus, content_transformer(tolower))
corpus = tm_map(corpus, removeNumbers)
corpus = tm_map(corpus, removePunctuation)
corpus = tm_map(corpus, removeWords, stopwords())
corpus = tm_map(corpus, stemDocument)
corpus = tm_map(corpus, stripWhitespace)

dtm_train = DocumentTermMatrix(corpus, control = list(weightning = function(x) weightTfIdf(x, normalize
#it would crash unless below step is done 12gb ram is insufficient if not done
dtm_train = removeSparseTerms(dtm_train, 0.99)
dataset = as.data.frame(as.matrix(dtm_train))
```

```
dataset$toxic = NULL
dataset$severe_toxic = NULL
dataset$obscene = NULL
dataset$threat = NULL
dataset$insult = NULL
dataset$identity_hate = NULL



corpus = VCorpus(VectorSource(test$comment_text))
corpus = tm_map(corpus, content_transformer(tolower))
corpus = tm_map(corpus, removeNumbers)
corpus = tm_map(corpus, removePunctuation)
corpus = tm_map(corpus, removeWords, stopwords())
corpus = tm_map(corpus, stemDocument)
corpus = tm_map(corpus, stripWhitespace)


dtm = DocumentTermMatrix(corpus, control = list(weightning = function(x) weightTfIdf(x, normalize = F))
dtm = removeSparseTerms(dtm, 0.99)
datasetTest = as.data.frame(as.matrix(dtm))
```

We are making the test and the train data set such that they have the same columns. We are also ensuring no data leakage in the procedure of doing so. test matrix will have the word columns of the train, the train will not have to worry about the columns in the test, which is what happens in general. test matrix is datasetTest train matrix is dataset

saving the progress we made so far

```
write.csv(dataset, 'dataset.csv', row.names = F)
write.csv(datasetTest, 'datasetTest.csv', row.names = F)
```

```
dataset = read.csv("dataset.csv")
datasetTest = read.csv("datasetTest.csv")
```

```
matrix_column_names <- colnames(dataset)
intersectnames <- intersect(colnames(dataset),colnames(datasetTest))
datasetTest = datasetTest[ , (colnames(datasetTest) %in% intersectnames)]
trainminustestcolms <- matrix_column_names[!(matrix_column_names %in% intersectnames)]
for (s in trainminustestcolms) {
  datasetTest[, s] <- 0
}
```

removing unnecessary variables

```
rm(corpus)
rm(dtm)
rm(dtm_train)
rm(tmp)
rm(intersectnames)
rm(matrix_column_names)
rm(s)
rm(sample_submissionpath)
```

```
rm(test_labelspath)
rm(testpath)
rm(trainminustestcolms)
```

Methods and Analysis:

As of now we are dealing with a multi label classification problem. Also the data distribution is very much skewed. We will be making 6 submodels one for each toxic class and the submodel i decides if the comment belongs to the toxic class i or not.

There are some ways to deal with the imbalanced dataset.

1. Under sampling of the majority class : We don't want to loose valuable data. So not using this technique.
2. Oversampling of the minority class : The data is so large.. over sampling of the minority class may double the total data. This may cause some computational complexities… given.. the same has to be done for all the 6 submodels and we tune the hyper parameters for each submodels for double the data.. This is definitely a solution…. Though it is expensive. This can be our last resort..
3. Applying weights/costs: applying costs to the false positive or false negative. This is also a good approach, now we have an additional cost hyper parameter to tune for each of the 6 submodels.
4. Treating it as anomaly detection. We can consider the minority samples as the positive class and build model for it. The other ones are the outliers. This seems to be a good to go approach.

Anomaly detection sounds very positive for our data. For anomaly detection we build a one vs others classifier for each toxic class.

The available options we have are logistic regression, svm. Since there are common words between the toxic classes, the dataset is likely to be highly non linear.

Using a radial kernel for one class classification is the perfect fit as radial kernel can bring out the most complex nonlinear margins of the data.

Among the optimization metrics for classification, accuracy cannot be used here due to the high imbalance of data. F1 score is the metric which better deals with the false positives and false negatives. It is more sensitive towards skewed data.

When the output taken as probabilities, ROC-AUC gives a better view about the performance of the data, as it takes in to account the performance at every possible threshold. But ROC-AUC is not sensitive towards skewed data.

Let us consider F1 score as an optimization metric, but, we would have a peek at the AUC-ROC too for a clearer approach.

F1 score is the harmonic (may or may not be weighted) mean of precision and recall. precision is, of all we detected as positive, how many are actually positive. recall is, of all the positive, how many were we able to predict correctly.

AUC-ROC is the graph between sensitivity and (1 - specificity) at various thresholds. Sensitivity is a measure of the proportion of actual positive cases that got predicted as positive (or true positive). Sensitivity is also termed as Recall. Specificity is the proportion of truly negative cases that were classified as negative. i.e true negative rate. Sensitivity = TP / (TP + FN)

Specificity = TN / (TN + FP)

Higher AUC / F1 indicates a better model.

Each one class classifier svm takes only the positive samples and models them. For any test point, it checks if the point belongs to the model or not. Let us start with modeling the one class classifier svm for each of the 6 toxic classes.

```r
#uncomment to run the code with less data
 #datasetTest <- datasetTest[1:100, ]
 #test_labels <- test_labels[1:100, ]
 #sample_submission <- sample_submission[1:100, ]
```

```r
#toxic
dataset2 = dataset
dataset2$toxic = factor(train$toxic)
dataset2 <- dataset2 %>% filter(toxic == 1)
#uncomment below line to run on small data
#dataset2 <- dataset2[1:100, ]
tune_out <-
    tune.svm(x = as.matrix(dataset2 %>% select(-toxic)), y = dataset2$toxic,
             type = 'one-classification',
             kernel = "radial",
             #can specify more options at nu and gamma for hyper parameter optimization
           #doing this to speed up running the code for the evaluator, else it might take
           #a very very long time
           nu = 0.1, #gamma will be default, 1/data dimension
           cross = 0) #to perform k fold cross validation, assign cross = k > 0
model <- tune_out$best.model

print(summary(model))
```

```
##
## Call:
## best.svm(x = as.matrix(dataset2 %>% select(-toxic)), y = dataset2$toxic,
##     nu = 0.1, type = "one-classification", kernel = "radial", cross = 0)
##
##
## Parameters:
##    SVM-Type:  one-classification
##  SVM-Kernel:  radial
##       gamma:  0.00245098
##          nu:  0.1
##
## Number of Support Vectors:  1588
##
##
##
##
## Number of Classes: 1
```

```r
predictionstoxic = ifelse(predict(model,datasetTest) == TRUE, 1, 0)
```

```r
#severe_toxic
dataset2 = dataset
dataset2$severe_toxic = factor(train$severe_toxic)
dataset2 <- dataset2 %>% filter(severe_toxic == 1)
#uncomment below line to run on small data
#dataset2 <- dataset2[1:100, ]
tune_out <-
```

```
    tune.svm(x = as.matrix(dataset2 %>% select(-severe_toxic)), y = dataset2$severe_toxic,
             type = 'one-classification',
             kernel = "radial",
             #can specify more options at nu and gamma for hyper parameter optimization
            #doing this to speed up running the code for the evaluator, else it might take
            #a very very long time
            nu = 0.1, #gamma will be default, 1/data dimension
            cross = 0) #to perform k fold cross validation, assign cross = k > 0
model <- tune_out$best.model

print(summary(model))
```

```
##
## Call:
## best.svm(x = as.matrix(dataset2 %>% select(-severe_toxic)), y = dataset2$severe_toxic,
##     nu = 0.1, type = "one-classification", kernel = "radial", cross = 0)
##
##
## Parameters:
##    SVM-Type:  one-classification
##  SVM-Kernel:  radial
##       gamma:  0.00245098
##          nu:  0.1
##
## Number of Support Vectors:  160
##
##
##
##
## Number of Classes: 1
```

```
predictionssevere_toxic = ifelse(predict(model,datasetTest) == TRUE, 1, 0)
```

```
#obscene
dataset2 = dataset
dataset2$obscene = factor(train$obscene)
dataset2 <- dataset2 %>% filter(obscene == 1)
#uncomment below line to run on small data
#dataset2 <- dataset2[1:100, ]
tune_out <-
    tune.svm(x = as.matrix(dataset2 %>% select(-obscene)), y = dataset2$obscene,
             type = 'one-classification',
             kernel = "radial",
             #can specify more options at nu and gamma for hyper parameter optimization
            #doing this to speed up running the code for the evaluator, else it might take
            #a very very long time
            nu = 0.1, #gamma will be default, 1/data dimension
            cross = 0) #to perform k fold cross validation, assign cross = k > 0
model <- tune_out$best.model

print(summary(model))
```

```
##
```

```
## Call:
## best.svm(x = as.matrix(dataset2 %>% select(-obscene)), y = dataset2$obscene,
##     nu = 0.1, type = "one-classification", kernel = "radial", cross = 0)
##
##
## Parameters:
##    SVM-Type:  one-classification
##  SVM-Kernel:  radial
##       gamma:  0.00245098
##          nu:  0.1
##
## Number of Support Vectors:  904
##
##
##
##
## Number of Classes: 1
```

```
predictionsobscene = ifelse(predict(model,datasetTest) == TRUE, 1, 0)
```

```
#threat
dataset2 = dataset
dataset2$threat = factor(train$threat)
dataset2 <- dataset2 %>% filter(threat == 1)
#uncomment below line to run on small data
#dataset2 <- dataset2[1:100, ]
tune_out <-
    tune.svm(x = as.matrix(dataset2 %>% select(-threat)), y = dataset2$threat,
             type = 'one-classification',
             kernel = "radial",
             #can specify more options at nu and gamma for hyper parameter optimization
           #doing this to speed up running the code for the evaluator, else it might take
           #a very very long time
           nu = 0.1, #gamma will be default, 1/data dimension
           cross = 0) #to perform k fold cross validation, assign cross = k > 0
model <- tune_out$best.model

print(summary(model))
```

```
##
## Call:
## best.svm(x = as.matrix(dataset2 %>% select(-threat)), y = dataset2$threat,
##     nu = 0.1, type = "one-classification", kernel = "radial", cross = 0)
##
##
## Parameters:
##    SVM-Type:  one-classification
##  SVM-Kernel:  radial
##       gamma:  0.00245098
##          nu:  0.1
##
## Number of Support Vectors:  48
##
```

```
##
##
##
## Number of Classes: 1
```

```
predictionsthreat = ifelse(predict(model,datasetTest) == TRUE, 1, 0)
```

```
#insult
dataset2 = dataset
dataset2$insult = factor(train$insult)
dataset2 <- dataset2 %>% filter(insult == 1)
#uncomment below line to run on small data
#dataset2 <- dataset2[1:100, ]
tune_out <-
    tune.svm(x = as.matrix(dataset2 %>% select(-insult)), y = dataset2$insult,
             type = 'one-classification',
             kernel = "radial",
             #can specify more options at nu and gamma for hyper parameter optimization
           #doing this to speed up running the code for the evaluator, else it might take
           #a very very long time
           nu = 0.1, #gamma will be default, 1/data dimension
           cross = 0) #to perform k fold cross validation, assign cross = k > 0
model <- tune_out$best.model

print(summary(model))
```

```
##
## Call:
## best.svm(x = as.matrix(dataset2 %>% select(-insult)), y = dataset2$insult,
##     nu = 0.1, type = "one-classification", kernel = "radial", cross = 0)
##
##
## Parameters:
##    SVM-Type:  one-classification
##  SVM-Kernel:  radial
##       gamma:  0.00245098
##          nu:  0.1
##
## Number of Support Vectors:  849
##
##
##
##
## Number of Classes: 1
```

```
predictionsinsult = ifelse(predict(model,datasetTest) == TRUE, 1, 0)
```

```
#identity_hate
dataset2 = dataset
dataset2$identity_hate = factor(train$identity_hate)
dataset2 <- dataset2 %>% filter(identity_hate == 1)
#uncomment below line to run on small data
```

```r
#dataset2 <- dataset2[1:100, ]
tune_out <-
    tune.svm(x = as.matrix(dataset2 %>% select(-identity_hate)), y = dataset2$identity_hate,
             type = 'one-classification',
             kernel = "radial",
             #can specify more options at nu and gamma for hyper parameter optimization
           #doing this to speed up running the code for the evaluator, else it might take
           #a very very long time
           nu = 0.1, #gamma will be default, 1/data dimension
           cross = 0) #to perform k fold cross validation, assign cross = k > 0
model <- tune_out$best.model

print(summary(model))
```

```
##
## Call:
## best.svm(x = as.matrix(dataset2 %>% select(-identity_hate)), y = dataset2$identity_hate,
##     nu = 0.1, type = "one-classification", kernel = "radial", cross = 0)
##
##
## Parameters:
##    SVM-Type:  one-classification
##  SVM-Kernel:  radial
##       gamma:  0.00245098
##          nu:  0.1
##
## Number of Support Vectors:  143
##
##
##
##
## Number of Classes: 1
```

```r
predictionsidentity_hate = ifelse(predict(model,datasetTest) == TRUE, 1, 0)
```

The output of one class classifier svm is usually a T or F, convert them in to 1, 0 and find the f1 score The test_labels are the solutions given by kaggle, the -1 indicates they are not relevant examples and are not included in the scoring. such examples are added in kaggle to prevent malpractice. So removing the datapoints with -1 in their labels

```r
submission <- sample_submission
submission$toxic = predictionstoxic
submission$severe_toxic = predictionssevere_toxic
submission$obscene = predictionsobscene
submission$threat = predictionsthreat
submission$insult = predictionsinsult
submission$identity_hate = predictionsidentity_hate

y_svm <- submission[(test_labels$toxic != -1 & test_labels$severe_toxic != -1 & test_labels$obscene != -
                     & test_labels$threat != -1 & test_labels$insult != -1 & test_labels$identity_hate

y_act <- test_labels[(test_labels$toxic != -1 & test_labels$severe_toxic != -1 & test_labels$obscene !=
                       & test_labels$threat != -1 & test_labels$insult != -1 & test_labels$identity_hat
```

F1 score metrics of the one class svm

```
#doing this to be able to calculate f1 score even if all the y_actuals or y_predicted belong to
#one class. If not done the recall may become 0/0 and this throws an error.
#to avoid this, in such cases, we change the class one one of the observations.
if(length(unique(y_act$toxic)) == 1) {
  y_act$toxic[1] = (1 - y_act$toxic[1])
}
if(length(unique(y_act$severe_toxic)) == 1) {
  y_act$severe_toxic[1] = (1 - y_act$severe_toxic[1])
}
if(length(unique(y_act$threat)) == 1) {
  y_act$threat[1] = (1 - y_act$threat[1])
}
if(length(unique(y_act$obscene)) == 1) {
  y_act$Obscene[1] = (1 - y_act$obscene[1])
}
if(length(unique(y_act$insult)) == 1) {
  y_act$insult[1] = (1 - y_act$insult[1])
}
if(length(unique(y_act$identity_hate)) == 1) {
  y_act$identity_hate[1] = (1 - y_act$identity_hate[1])
}

if(length(unique(y_svm$toxic)) == 1) {
  y_svm$toxic[1] = (1 - y_svm$toxic[1])
}
if(length(unique(y_svm$severe_toxic)) == 1) {
  y_svm$severe_toxic[1] = (1 - y_svm$severe_toxic[1])
}
if(length(unique(y_svm$threat)) == 1) {
  y_svm$threat[1] = (1 - y_svm$threat[1])
}
if(length(unique(y_svm$obscene)) == 1) {
  y_svm$Obscene[1] = (1 - y_svm$obscene[1])
}
if(length(unique(y_svm$insult)) == 1) {
  y_svm$insult[1] = (1 - y_svm$insult[1])
}
if(length(unique(y_svm$identity_hate)) == 1) {
  y_svm$identity_hate[1] = (1 - y_svm$identity_hate[1])
}
```

```
#onesvm
toxic <- F1_Score(y_act$toxic, y_svm$toxic)
severe_toxic <- F1_Score(y_act$severe_toxic, y_svm$severe_toxic)
obscene <- F1_Score(y_act$obscene, y_svm$obscene)
threat <- F1_Score(y_act$threat, y_svm$threat)
insult <- F1_Score(y_act$insult, y_svm$insult)
identity_hate <- F1_Score(y_act$identity_hate, y_svm$identity_hate)
#macroaverage of f1
onesvmf1 <- print((toxic + severe_toxic + obscene + threat + insult + identity_hate)/6)
```

```
## [1] 0.5341708
```

```r
cat("f1 score of one class svm at nu = 0.1 :", onesvmf1)
```

```
## f1 score of one class svm at nu = 0.1 : 0.5341708
```

Using one class svm at nu = 0.1, we got an okish f1 score. As experimented in google colab(required high ram), as nu parameter increases, the f1 score increased. With a nu of 0.5, we got an optimal f1 score of 0.86. (couldn't run with nu = optimal here as it needed more than 6 gb ram while running the code, generating the pdf) (We have a huge dataset)

nu : An upper bound on the fraction of margin errors (see User Guide) and a lower bound of the fraction of support vectors. Should be in the interval (0, 1]. At nu nearing 1, the model overfits the data.

With higher nu, the model can capture more nonlinearities in the data.

xgboost is another model which is designed to be sensitive towards the imbalanced data and is known to be capturing complex patterns of the data. so we will be try the xgboost model We will look at both AUC-ROC and f1 metrics for this model.

We will model 6 xgboosts one for each toxic class.

```r
set.seed(1)
#toxic
dataset2 = dataset
dataset2$toxic = as.factor(train$toxic)
levels(dataset2$toxic) = make.names(unique(dataset2$toxic))
#uncomment below 2 lines for running on small data
#p <- createDataPartition(train$toxic, times = 1, p = 0.01)
#dataset2 = dataset2[p[[1]], ]

formula = toxic ~ .
#kept method none and no grid options to speed up running things for the evaluator
#as the cv or boot strap or providing grid options take a large amount of time (more than 12 hours)
#and cannot be verified by the evaluator.
#We also have option to allow parallel processing in fitControl
fitControl <- trainControl(method="none",classProbs=TRUE, summaryFunction=twoClassSummary)

xgbGrid <- expand.grid(nrounds = 500,
                       max_depth = 6,
                       eta = .05,#learning rage
                       gamma = 0.001, #regularization parameter.
                       colsample_bytree = .8,
                       min_child_weight = 1,
                       subsample = 1)

set.seed(13)

model = train(formula, data = dataset2,
                method = "xgbTree",trControl = fitControl,
                tuneGrid = xgbGrid,na.action = na.pass,metric="ROC", maximize=FALSE)

predictionsToxic = predict(model,datasetTest,type = 'prob')$X1
predToxic = predict(model,datasetTest)
```

```
#severe_toxic calculation
set.seed(1)
dataset2 = dataset
dataset2$severe_toxic = train$severe_toxic
dataset2$severe_toxic = as.factor(dataset2$severe_toxic)
levels(dataset2$severe_toxic) = make.names(unique(dataset2$severe_toxic))
#uncomment below 2 lines for running on small data
#p <- createDataPartition(train$severe_toxic, times = 1, p = 0.01)
#dataset2 = dataset2[p[[1]], ]

formula = severe_toxic ~ .

set.seed(13)

model = train(formula, data = dataset2,
                 method = "xgbTree",trControl = fitControl,
                 tuneGrid = xgbGrid,na.action = na.pass,metric="ROC", maximize=FALSE)

predictionsSevereToxic = predict(model,datasetTest,type = 'prob')$X1
predSevereToxic = predict(model,datasetTest)
```

```
#obscene
set.seed(1)
dataset2 = dataset
dataset2$obscene = train$obscene
dataset2$obscene = as.factor(dataset2$obscene)
levels(dataset2$obscene) = make.names(unique(dataset2$obscene))
#uncomment below 2 lines for running on small data
#p <- createDataPartition(train$obscene, times = 1, p = 0.01)
#dataset2 = dataset2[p[[1]], ]

formula = obscene ~ .
set.seed(13)
model = train(formula, data = dataset2,
                   method = "xgbTree",trControl = fitControl,
                   tuneGrid = xgbGrid,na.action = na.pass,metric="ROC", maximize=FALSE)

predictionsObscene = predict(model,datasetTest,type = 'prob')$X1
predObscene = predict(model,datasetTest)
```

```
#threat
set.seed(1)
dataset2 = dataset
dataset2$threat = train$threat
dataset2$threat = as.factor(dataset2$threat)
levels(dataset2$threat) = make.names(unique(dataset2$threat))
#uncomment below 2 lines for running on small data
#p <- createDataPartition(train$threat, times = 1, p = 0.01)
#dataset2 = dataset2[p[[1]], ]

formula = threat ~ .
set.seed(13)
model = train(formula, data = dataset2,
```

```
                  method = "xgbTree",trControl = fitControl,
                  tuneGrid = xgbGrid,na.action = na.pass,metric="ROC", maximize=FALSE)

predictionsThreat = predict(model,datasetTest,type = 'prob')$X1
predThreat = predict(model,datasetTest)
```

```
#insult
set.seed(1)
dataset2 = dataset
dataset2$insult = train$insult
dataset2$insult = as.factor(dataset2$insult)
levels(dataset2$insult) = make.names(unique(dataset2$insult))
#uncomment below 2 lines for running on small data
#p <- createDataPartition(train$insult, times = 1, p = 0.01)
#dataset2 = dataset2[p[[1]], ]

formula = insult ~ .
set.seed(13)
model = train(formula, data = dataset2,
                  method = "xgbTree",trControl = fitControl,
                  tuneGrid = xgbGrid,na.action = na.pass,metric="ROC", maximize=FALSE)

predictionsInsult = predict(model,datasetTest,type = 'prob')$X1
predInsult = predict(model,datasetTest)
```

```
#identity_hate
set.seed(1)
dataset2 = dataset
dataset2$identity_hate = train$identity_hate
dataset2$identity_hate = as.factor(dataset2$identity_hate)
levels(dataset2$identity_hate) = make.names(unique(dataset2$identity_hate))
#uncomment below 2 lines for running on small data
#p <- createDataPartition(train$identity_hate, times = 1, p = 0.01)
#dataset2 = dataset2[p[[1]], ]
set.seed(13)
formula = identity_hate ~ .

model = train(formula, data = dataset2,
                method = "xgbTree",trControl = fitControl,
                tuneGrid = xgbGrid,na.action = na.pass,metric="ROC", maximize=FALSE)

predictionsHate = predict(model,datasetTest,type = 'prob')$X1
predHate = predict(model,datasetTest)
```

xgboost performance evaluation: Using AUC

```
#AUC
submission <- sample_submission
submission$toxic = predictionsToxic
submission$severe_toxic = predictionsSevereToxic
submission$obscene = predictionsObscene
submission$threat = predictionsThreat
submission$insult = predictionsInsult
```

```
submission$identity_hate = predictionsHate

y_prb <- submission[(test_labels$toxic != -1 & test_labels$severe_toxic != -1 & test_labels$obscene != -
                     & test_labels$threat != -1 & test_labels$insult != -1 & test_labels$identity_hate

toxic <- auc(y_act$toxic, y_prb$toxic)
severe_toxic <- auc(y_act$severe_toxic, y_prb$severe_toxic)
obscene <- auc(y_act$obscene, y_prb$obscene)
threat <- auc(y_act$threat, y_prb$threat)
insult <- auc(y_act$insult, y_prb$insult)
identity_hate <- auc(y_act$identity_hate, y_prb$identity_hate)
#macroaverage of auc
xgboostAUC <- (toxic + severe_toxic + obscene + threat + insult + identity_hate)/6
cat("AUC for xgboost", xgboostAUC, "\n")
```

```
## AUC for xgboost 0.8464095
```

f1 score for the xgboost model

```
submission <- sample_submission
submission$toxic = factor(ifelse(predToxic=='X1', 1, 0), levels = c(0, 1))
submission$severe_toxic = factor(ifelse(predSevereToxic=='X1', 1, 0), levels = c(0, 1))
submission$obscene = factor(ifelse(predObscene=='X1', 1, 0), levels = c(0, 1))
submission$threat = factor(ifelse(predThreat=='X1', 1, 0), levels = c(0, 1))
submission$insult = factor(ifelse(predInsult=='X1', 1, 0), levels = c(0, 1))
submission$identity_hate = factor(ifelse(predHate=='X1', 1, 0), levels = c(0, 1))

xgb_lbl <- submission[(test_labels$toxic != -1 & test_labels$severe_toxic != -1 & test_labels$obscene !=
                       & test_labels$threat != -1 & test_labels$insult != -1 & test_labels$identity_hat
```

```
if(length(unique(xgb_lbl$toxic)) == 1) {
  xgb_lbl$toxic[1] = (1 - xgb_lbl$toxic[1])
}
if(length(unique(xgb_lbl$severe_toxic)) == 1) {
  xgb_lbl$severe_toxic[1] = (1 - xgb_lbl$severe_toxic[1])
}
if(length(unique(xgb_lbl$threat)) == 1) {
  xgb_lbl$threat[1] = (1 - xgb_lbl$threat[1])
}
if(length(unique(xgb_lbl$obscene)) == 1) {
  xgb_lbl$Obscene[1] = (1 - xgb_lbl$obscene[1])
}
if(length(unique(xgb_lbl$insult)) == 1) {
  xgb_lbl$insult[1] = (1 - xgb_lbl$insult[1])
}
if(length(unique(xgb_lbl$identity_hate)) == 1) {
  xgb_lbl$identity_hate[1] = (1 - xgb_lbl$identity_hate[1])
}
```

```
toxic <- F1_Score(y_act$toxic, xgb_lbl$toxic)
severe_toxic <- F1_Score(y_act$severe_toxic, xgb_lbl$severe_toxic)
obscene <- F1_Score(y_act$obscene, xgb_lbl$obscene)
```

```
threat <- F1_Score(y_act$threat, xgb_lbl$threat)
insult <- F1_Score(y_act$insult, xgb_lbl$insult)
identity_hate <- F1_Score(y_act$identity_hate, xgb_lbl$identity_hate)
#macroaverage of f1
xgboostF1 <- (toxic + severe_toxic + obscene + threat + insult + identity_hate)/6
cat("f1 for xgboost", xgboostF1, "\n")
```

```
## f1 for xgboost 0.9826881
```

Results and Conclusions

We have looked at the distribution of the data, type of the problem and narrowed down our options to one class svm and xgboost. Their have their performance measures as below.

```
cat('one class svm at nu = 0.1 :', onesvmf1, "\n")
```

```
## one class svm at nu = 0.1 : 0.5341708
```

```
cat('xgboost:', xgboostF1, "\n")
```

```
## xgboost: 0.9826881
```

```
cat('xgboost predicting probabilities', xgboostAUC, "\n")
```

```
## xgboost predicting probabilities 0.8464095
```

For xgboost also(like one class svm at nu = 0.5), we got high auc and f1 score.

The above performance metrics are really good and getting such metrics is very difficult on a live project with many more complexities in the data. We were given a rather controlled data, unaffected by language differences, given that some people type their mother tongue in english,.. which makes is difficult categorize it.

Comparing one class svm at nu = 0.5 and xgboost, we may be going with xgboost because of its less memory cost and optimal performance. And we will consider the auc score of xgboost, as it indicates the performance of the model at every threshold.

Issues and Futurework:

Also we didn't use the smileys. Recently smileys can be used to express a wide range of emotions which also include toxic emotions. This should also be taken into account for more effective classification.

The upcoming test data can contain plethora of new words, we should keep training the model in regular intervals of time to keep the model updated.

The bad words can also be quoted by using symbols like @, #, ! between the letters. If some one comments in such a way, we may not be able identify it as a toxic class. We should include a parser which parses through the sentences, finds these symbols, try to replace them with similar looking letters and then pass it for the classification.

Using neural networks model for this kind of problem statements improves performance even further As they can capture more non linearities in the data structure.

We have made the testDocumentMatrix by using all the test data at a time. In real scenario, we may get only a few sentences at a time. In such case, we should figure out a way to convert each sentence in to the tfidf form, one at a time if needed.