"Predicting Probability of Each Type of Toxicity for Each Comment"

Abhishek Hupele

1-May-2018

Contents

1 Introduction

1.1 Problem statement

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

Our aim is to detect negative online behaviors, like toxic comments (i.e. comments that are rude, disrespectful or otherwise likely to make someone leave a discussion). 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

We are provided with a large number of Wikipedia comments which have been labeled by human raters for toxic behavior. Our task is to perform sensitivity analysis which predicts a probability of each type of toxicity for each comment.
The types of toxicity are:

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

Given below is a sample of the data set that we are using to predict the sensitivity of comments:

Table 1.1: the training set, sample data contains comments with toxicity

id	comment_text	toxic	severe_	obsc	thre	insu	identity_
0000997932	Explanation Why the edits mad	0	0	0	0	0	0
000103f0d9	D'aww! He matche	0	0	0	0	0	0
000113f07e	Hey man, I'm really	0	0	0	0	0	0
0001b41b1c	" More I can't make any re There appears to b	0	0	0	0	0	0

2 Methodology

Any predictive modeling requires that we look at the data before we start modeling. However, in text mining terms looking at data refers to so much more than just looking. Looking at text refers to exploring the text, cleaning the text data as well as visualizing the text data through graphs and plots. To start this process we will first clean the data by removing irrelevant , meaningless texts or characters which do not add valuable information to text data. than. Further, We can visualize that in a glance by looking at the frequency or term factors of the text.

Preprocessing is an important task and critical step in Text mining. In the area of Text Mining, data preprocessing used for extracting interesting and non-trivial and knowledge from unstructured text data. Information Retrieval (IR) is essentially a matter of deciding which documents in a collection should be retrieved to satisfy a user's need for information. Before the information retrieval from the documents, the data preprocessing techniques are applied on the target data set to reduce the size of the data set which will increase the effectiveness of IR System.

2.1 Exploratory Analysis

<u>Distribution of Word Length</u>

In figure 2.1 we have plotted a histogram showing the comment word length distribution. As visible in histogram , distribution is right skewed with maximum comments with word length case to a few hundreds.

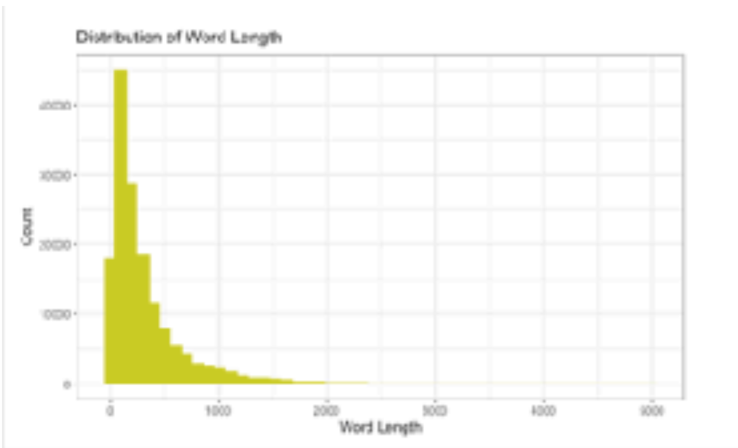


Figure 2.1 : Distribution of Word Length(See R Code in Appendix)

Most Frequent Words

We generate list of the most frequently occurring words in the comments (excluding stopwords).

```
# A tibble: 10 x 2
   word             n
   <fct>          <int>
 1 article        55907
 2 page           46189
 3 wikipedia      36640
 4 talk           32566
 5 edit           18237
 6 people         17835
 7 articles       16123
 8 time           15841
 9 information  12147
10 deletion     11375
```

Table 2.1 : Most Frequent Words (See R Code in Appendix)

Tokenisation of the Comments

The comments are broken up into words.The first 10 rows of comments broken up into words are shown below.

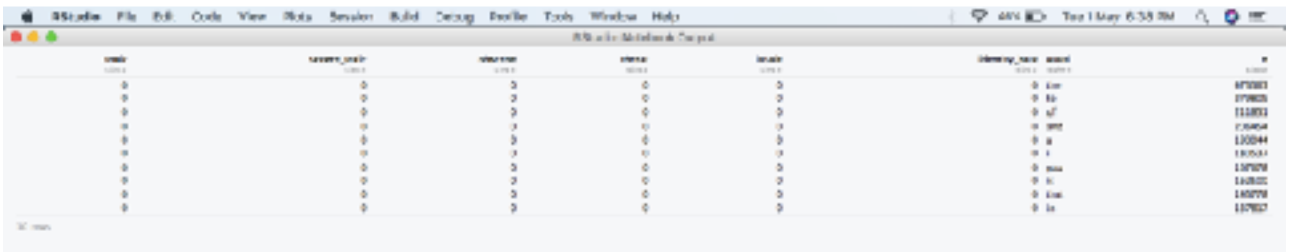


Table 2.2 : Tokenisation of the Comments (See R Code in Appendix)

Unique Categories of Text

The combinations of `toxic,severe toxic,obscene,threat,insult and identity hate` will create unique categories. We will display those categories here. There are 41 unique categories generated.

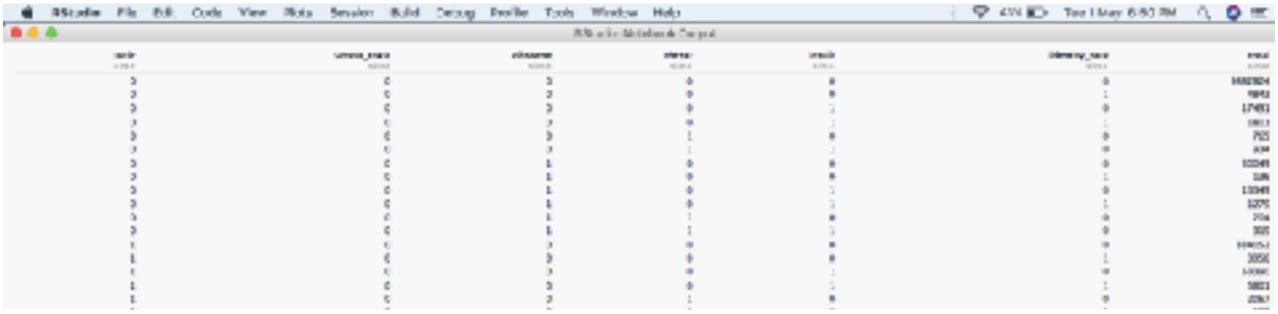


Table 2.3 : Unique Categories of Toxicity (See R Code Appendix)

<u>TF-IDF</u>

We wish to find out the important words in this `Toxic Comments`. Example for a patient , the most important word is **medicine**. Example for a cook, important words would be related to **food**.
We would explore this using a fascinating concept known as **Term Frequency - Inverse Document Frequency**.

A **document** in this case is the set of lines associated with a unique category determined by the various elements such as `toxic,severe toxic,obscene,threat,insult and identity hate`.
TF-IDF computes a weight which represents the importance of a term inside a document.

It does this by comparing the frequency of usage inside an individual document as opposed to the entire data set (a collection of documents).
The importance increases proportionally to the number of times a word appears in the individual document itself--this is called Term Frequency. However, if multiple documents contain the same word many times then you run into a problem. That's why TF-IDF also offsets this value by the frequency of the term in the entire document set, a value called Inverse Document Frequency.
TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document)
IDF(t) = log_e(Total number of documents / Number of documents with term t in it).
Value = TF * IDF

Twenty Most Important words

Here using **TF-IDF** , we investigate the **Twenty Most Important words**

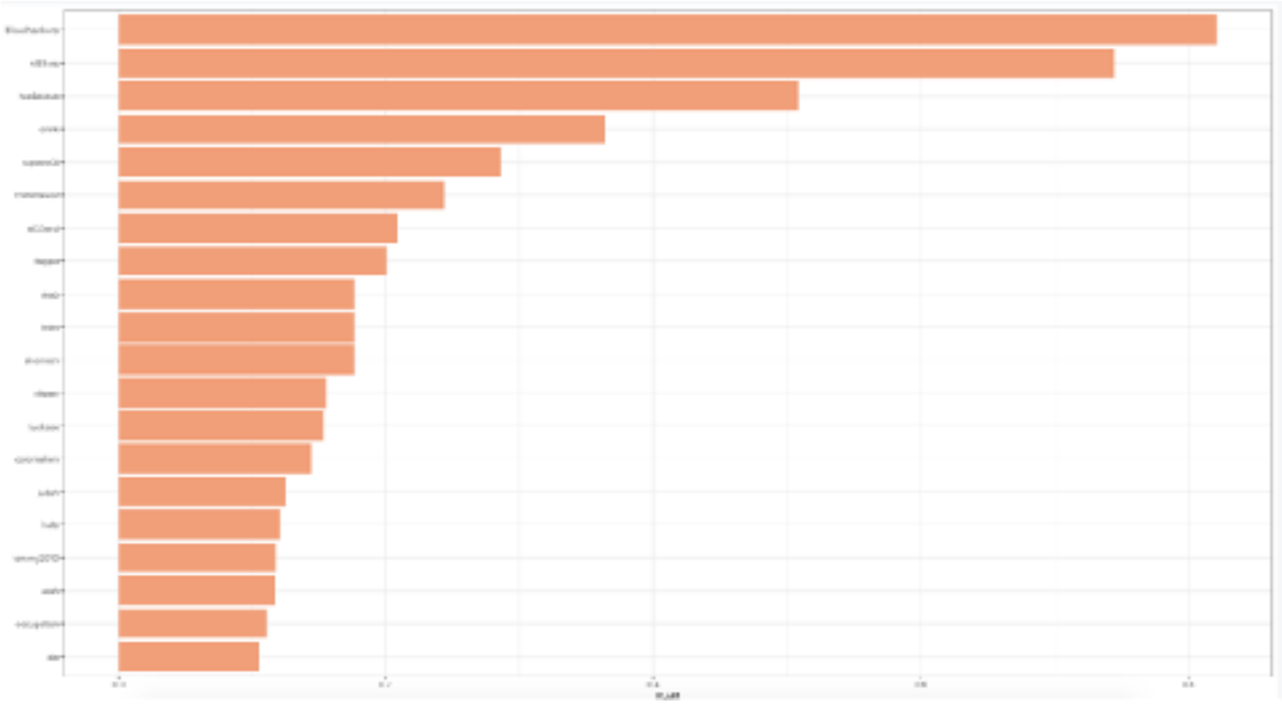
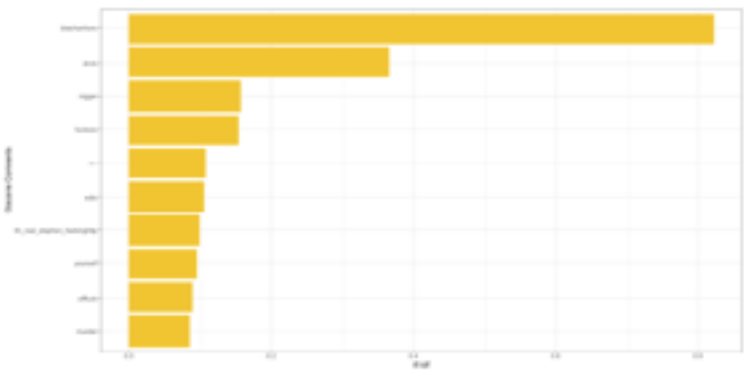
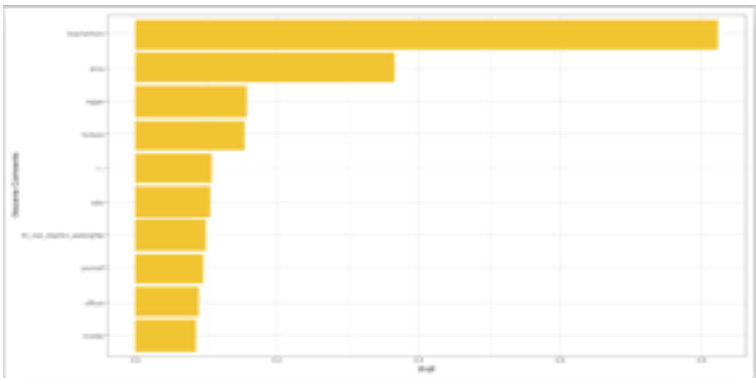


Figure 2.2 : TF-IDF , Important Words   (See R Code in Appendix)

We plot the TF-IDF for the Toxic Comments for each of the 6 categories

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





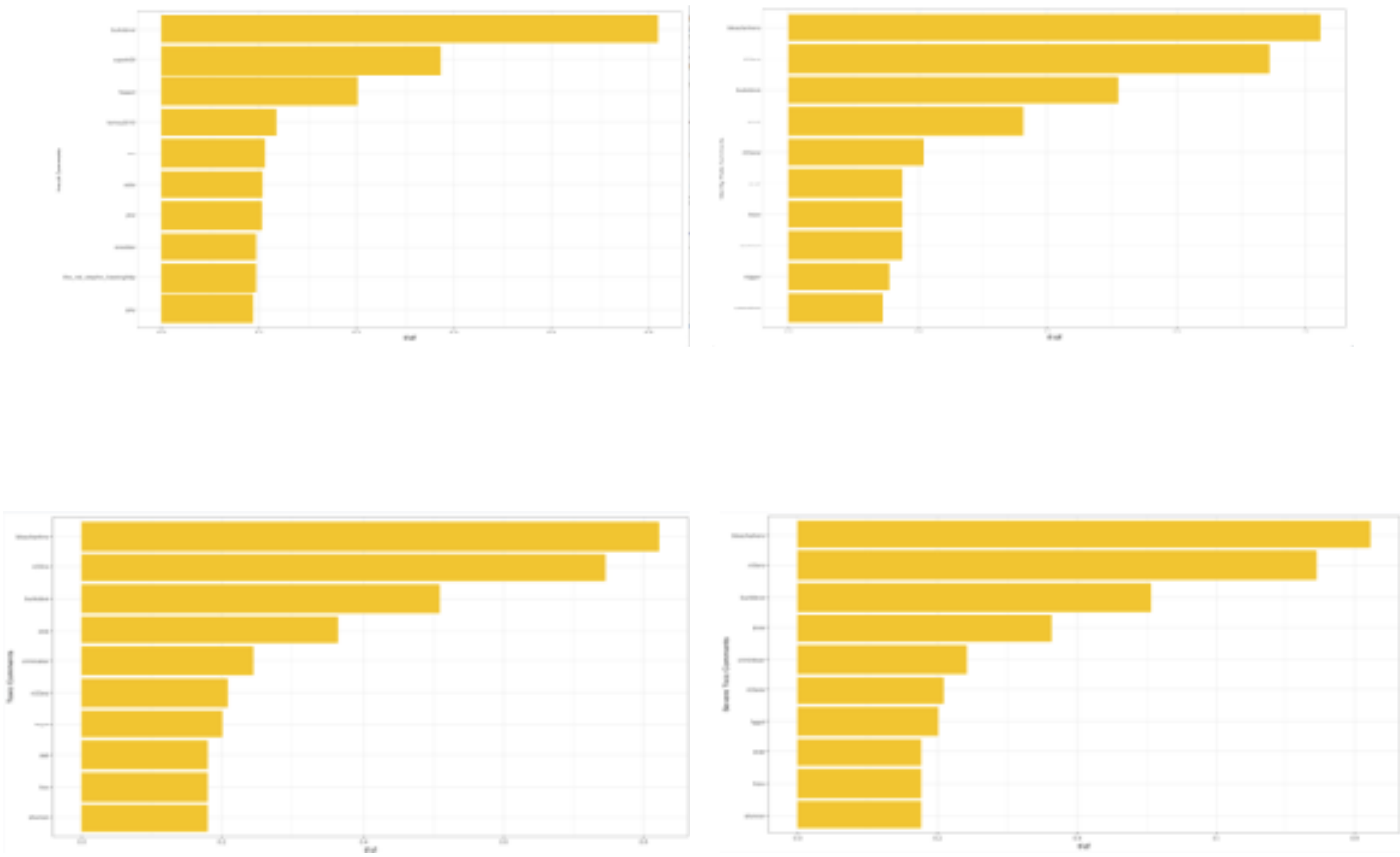


Figure 2.3 : TF-IDF plot, Toxicity Category wise (See R Code in Appendix)

Word Cloud for the Most Important Words

We show the **Fifty** most important words. This Word Cloud is based on the **TF- IDF** scores. Higher the score, bigger is the size of the text.

Figure 2.4 : Word Cloud (See R Code in Appendix)

2.2 Pre Processing

We incorporate text pre-processing techniques

a) Punctuation Marks - Remove punctuation marks from text. Like ? ! , ; [] () <>, . Also need to be careful for change in context of text getting changed due to removal, for example mr. john to mr John (mr stands for mister in 1st instance but could be misinterpreted as medical representative in 2nd)

b) Numbers - Remove numbers from the text content available, for example

3/12/91
Mar 13 1991
55 B.C
B-52
100.2.86.144

c) Case Folding - The whole point of lowercasing terms is to make them *more* likely to match, this job is done by case folding rather than by lowercasing. *Case folding* is the act of converting words into a (usually lowercase) form that does not necessarily result in the correct spelling, but does allow case-insensitive comparisons.

For instance, the letter ß, which is already lowercase, is *folded* to ss. Similarly, the lowercase ς is folded to σ, to make σ, ς, and Σ comparable, no matter where the letter appears in a word.

d) Stop Words - Many words in documents recur very frequently but are essentially meaningless as they are used to join words together in a sentence. It is commonly understood that stop words do not contribute to the context or content of textual documents. Due to their high frequency of occurrence, their presence in text mining presents an obstacle in understanding the content of the documents.

Stop words are very frequently used common words like ‘and’, ‘are’, ‘this’ etc. They are not useful in classification of documents. So they must be removed. This process also reduces the text data and improves the system performance.
e) White Spaces - Removes unnecessary extra space characters from text.
f) Stemming - Stemming is the process of conflating the variant forms of a word into a common representation, the stem. For example, the words: “presentation”, “presented”, “presenting” could all be reduced to a common representation “present”.

Both train and test data need to be pre-processed.

```
# Delete the leading spaces
library(stringr)
train$comment_text = str_trim(train$comment_text)

# Convert comment into corpus
library(tm)
traincorpus = Corpus(VectorSource(train$comment_text))

# Case Folding
traincorpus = tm_map(traincorpus, tolower)
# Remove Stop Words
traincorpus = tm_map(traincorpus,removeWords,stopwords('english'))
# Remove Punctuation marks
traincorpus = tm_map(traincorpus,removePunctuation)
# Remove Numbers
traincorpus = tm_map(traincorpus,removeNumbers)
# Remove unnecessary spaces
traincorpus = tm_map(traincorpus,stripWhitespace)
# Stemming
traincorpus = tm_map(traincorpus, stemDocument)
```

Now, We create the DTM for both Test and Train dataset. Sparse the dataset to remove words with less than 1% of occurrences to improve the efficiency of the model and decrease running time. Also, We need only those columns information from train dataset which are present in test dataset and only those columns in test dataset which can gather any meaningful insight on toxicity from train data set. Thus, we select common columns from both datasets

Project1_V2.Rmd* | test_dataset | plot_trainWords | total_words | trainWords

Filter

	dont	ever	get	guy	like	man	next	right	rule	thing	tim
1	1	1	2	1	1	1	1	1	2	1	
2	0	0	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	
5	0	0	0	0	0	0	0	0	0	0	
6	0	0	0	0	0	0	0	0	0	0	

	edit	just	made	new	now	page	pleas	remov
1	1	1	1	1	1	1	1	1
2	0	0	0	0	0	0	0	0
3	2	1	0	0	0	1	0	1
4	0	0	0	0	0	0	1	0
5	0	0	0	0	0	1	0	0

Figure 2.5 : screenshot shoeing DTM for common columns (See R Code in Appendix)

2.2 Modelling

In our early stages of analysis during pre-processing we have come to understand the words in comments that attribute to different toxicity levels in the train dataset. Based on the words in the every document probability can be assigned to the type of toxic category. We Fit Predictive Models over Different Tuning Parameters. Tuning Parameters can be derived for each toxic category.

We will be using XGBoost algorithm to calculate the various targets and predict the probabilities for each type of toxicity.

```
dataset2 = dataset
dataset2$toxic = train$toxic
dataset2$toxic = as.factor(dataset2$toxic)
levels(dataset2$toxic) = make.names(unique(dataset2$toxic))

formula = toxic ~ .

fitControl <- trainControl(method="none",classProbs=TRUE,
summaryFunction=twoClassSummary)

xgbGrid <- expand.grid(nrounds = 500, max_depth = 3, eta = .05,   gamma = 0,
                                 colsample_bytree = .8, min_child_weight = 1,
                                  subsample = 1)
```

```
set.seed(13)

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

predictionsToxic = predict(ToxicXGB,datasetTest,type = 'prob')
```

3 Conclusion

Predictive performance can be measured by comparing Predictions of the model with real values of the test variables, and calculating some average error measure. Need test dataset toxic categories values to measure the same.

Appendix - R Code, Extra Figures, Tables

Figure 2.1 : Distribution of Word Length

```
train$len = str_count(train$comment_text)
test$len = str_count(test$comment_text)

train %>%
  ggplot(aes(x = len)) +
  geom_histogram(fill= 'yellow3',bins = 50) +
  labs(x= 'Word Length',y = 'Count', title = 'Distribution of Word Length') +
  theme_bw()
```

Table 2.1 : Most Frequent Words

```
  train %>%
    unnest_tokens(word, comment_text) %>%
    filter(!word %in% stop_words$word) %>%
    count(word,sort = TRUE) %>%
    ungroup() %>%
    head(10)
```

Table 2.2 : Tokenisation of the Comments

```
trainWords <- train %>%
  unnest_tokens(word, comment_text) %>%
  count(toxic,severe_toxic,obscene,threat,insult,identity_hate,word) %>%
  ungroup()

head(trainWords,10)
```

Table 2.3 : Unique Categories of Toxicity

```
total_words <- trainWords %>%
  group_by(toxic,severe_toxic,obscene,threat,insult,identity_hate) %>%
  summarize(total = sum(n))

total_words
```

Figure 2.2 : TF-IDF , Important Words

```
Category =1:41

total_words$Category = Category

trainWords <- left_join(trainWords, total_words)

trainWords <- trainWords %>%
  bind_tf_idf(word, Category, n)


plot_trainWords <- trainWords %>%
  arrange(desc(tf_idf)) %>%
  mutate(word = factor(word, levels = rev(unique(word))))

plot_trainWords %>%
  top_n(20) %>%
  ggplot(aes(word, tf_idf)) +
  geom_col(fill = ‘orange’) +
  labs(x = NULL, y = "tf-idf") +
  coord_flip() +
  theme_bw()
```

Figure 2.3 : TF-IDF plot, Toxicity Category wise

```
# For ‘toxic’ comment
plot_trainWords %>%
  filter(toxic == 1 ) %>%
  top_n(20) %>%
  ggplot(aes(word, tf_idf)) +
  geom_col(fill = fillColor2) +
  labs(x = NULL, y = "tf-idf") +
  coord_flip() +
  theme_bw()


#For ‘Severe Toxic’ Comment
plot_trainWords %>%
  filter(severe_toxic == 1 ) %>%
  top_n(20) %>%
  ggplot(aes(word, tf_idf)) +
  geom_col(fill = fillColor2) +
  labs(x = NULL, y = "tf-idf") +
  coord_flip() +
  theme_bw()
```

Figure 2.4 : Word Cloud

```
plot_trainWords %>%
  with(wordcloud(word, tf_idf, max.words = 50,colors=brewer.pal(8, “Dark2")))
```

Figure 2.5 : screenshot shoeing DTM for common columns

```
colnamesSame = intersect(colnames(dataset),colnames(datasetTest))
dataset = dataset[ , (colnames(dataset) %in% colnamesSame)]
datasetTest = datasetTest[ , (colnames(datasetTest) %in% colnamesSame)]
```

Complete R Code : (*.Rmd)

````
#Load Library, setwd, import data files
```{r,message=FALSE,warning=FALSE}

library(tidyverse)
library(tidytext)
install.packages("tidytext", lib="/Library/Frameworks/R.framework/Versions/3.4/
Resources/library")
library(DT)
library(stringr)
library('wordcloud')
install.packages("igraph", lib="/Library/Frameworks/R.framework/Versions/3.4/
Resources/library")
library(igraph)
install.packages("ggraph", lib="/Library/Frameworks/R.framework/Versions/3.4/
Resources/library")
library(ggraph)
library(tm)
library(SnowballC)
library(caret)

rm(list=ls())

# setwd('/Users/abhishek/Desktop/Edwisor_Data Science Career/Aggregate_Notes/
Project_1')

train = read_csv("train.csv")
test = read_csv("test.csv")
submission = read_csv("sample_submission.csv")

```

# View the Data
```{r,message=FALSE,warning=FALSE}

head(train)
````
````

````
```

# Word Length Distribution
```{r,message=FALSE,warning=FALSE}

train$len = str_count(train$comment_text)
test$len = str_count(test$comment_text)

train %>%
  ggplot(aes(x = len)) +
  geom_histogram(fill= 'yellow2',bins = 50) +
  labs(x= 'Word Length',y = 'Count', title = paste('Distribution of Word Length ')) +
  theme_bw()

```




#Top Ten most Common Words
```{r,message=FALSE,warning=FALSE}

  train %>%
    unnest_tokens(word, comment_text) %>%
    filter(!word %in% stop_words$word) %>%
    count(word,sort = TRUE) %>%
    ungroup() %>%
    # mutate(word = factor(word, levels = rev(unique(word)))) %>%
    head(10)


```



#Tokenisation of the sentences

The sentences are broken up into words as shown below.

```{r,message=FALSE,warning=FALSE}

trainWords <- train %>%
  unnest_tokens(word, comment_text) %>%
  count(toxic,severe_toxic,obscene,threat,insult,identity_hate,word, sort = TRUE) %>%
  ungroup()

head(trainWords,10)
```



#Unique Categories of Text
````
````

````
The combinations of `toxic,severe toxic,obscene,threat,insult and identity hate` will create
unique categories. We will display those categories here.

```{r,message=FALSE,warning=FALSE}

trainWords <- train %>%
  unnest_tokens(word, comment_text) %>%
  count(toxic,severe_toxic,obscene,threat,insult,identity_hate,word) %>%
  ungroup()

total_words <- trainWords %>%
  group_by(toxic,severe_toxic,obscene,threat,insult,identity_hate) %>%
  summarize(total = sum(n))

total_words


```



#TF-IDF
## Twenty Most Important words
Here using **TF-IDF** , we investigate the **Twenty Most Important words**
```{r, message=FALSE, warning=FALSE}

Category =1:41

total_words$Category = Category

trainWords <- left_join(trainWords, total_words)

#Now we are ready to use the bind_tf_idf which computes the tf-idf for each term.
trainWords <- trainWords %>%
  bind_tf_idf(word, Category, n)


plot_trainWords <- trainWords %>%
  arrange(desc(tf_idf)) %>%
  mutate(word = factor(word, levels = rev(unique(word))))

plot_trainWords %>%
  top_n(20) %>%
  ggplot(aes(word, tf_idf)) +
  geom_col(fill = 'orange') +
  labs(x = NULL, y = "tf-idf") +
  coord_flip() +
  theme_bw()


```
````

````
#Various Categories of TF-IDF

##Toxic TF-IDF
We plot the TF-IDF for the Toxic Comments
```{r,message=FALSE,warning=FALSE}


plot_trainWords %>%
  filter(toxic == 1 ) %>%
  top_n(10) %>%
  ggplot(aes(word, tf_idf)) +
  geom_col(fill = 'yellow') +
  labs(x = 'Toxic Comments', y = "tf-idf") +
  coord_flip() +
  theme_bw()

```



##Severe Toxic TF-IDF
We plot the TF-IDF for the Severe Toxic Comments
```{r,message=FALSE,warning=FALSE}


plot_trainWords %>%
  filter(severe_toxic == 1 ) %>%
  top_n(10) %>%
  ggplot(aes(word, tf_idf)) +
  geom_col(fill = 'yellow') +
  labs(x = 'Severe Toxic Comments', y = "tf-idf") +
  coord_flip() +
  theme_bw()

```

##Obscene TF-IDF
We plot the TF-IDF for the Obscene Comments
```{r,message=FALSE,warning=FALSE}


plot_trainWords %>%
  filter(obscene == 1 ) %>%
  top_n(10) %>%
  ggplot(aes(word, tf_idf)) +
  geom_col(fill = 'yellow') +
  labs(x = 'Obscene Comments', y = "tf-idf") +
  coord_flip() +
  theme_bw()

```

##Threat TF-IDF
````

```
We plot the TF-IDF for the Threat Comments
```{r,message=FALSE,warning=FALSE}


plot_trainWords %>%
  filter(threat == 1 ) %>%
  top_n(10) %>%
  ggplot(aes(word, tf_idf)) +
  geom_col(fill = 'yellow') +
  labs(x = 'Threat Comments', y = "tf-idf") +
  coord_flip() +
  theme_bw()


# For ‘insult’ Comment

plot_trainWords %>%
  filter(insult == 1 ) %>%
  top_n(10) %>%
  ggplot(aes(word, tf_idf)) +
  geom_col(fill = 'yellow') +
  labs(x = 'Insult Comments', y = "tf-idf") +
  coord_flip() +
  theme_bw()


# For ‘identity hate’ Comment

plot_trainWords %>%
  filter(identity_hate == 1 ) %>%
  top_n(10) %>%
  ggplot(aes(word, tf_idf)) +
  geom_col(fill = 'yellow') +
  labs(x = 'Identity Hate Comments', y = "tf-idf") +
  coord_flip() +
  theme_bw()
```


#Word Cloud for the Most Important Words
We show the **Fifty** most important words. This Word Cloud is based on the **TF- IDF** scores. Higher the score, bigger is the size of the text.
```{r, message=FALSE, warning=FALSE}

plot_trainWords %>%
  with(wordcloud(word, tf_idf, max.words = 50,colors=brewer.pal(8, "Dark2")))

```


#Pre-rocessing
```{r,message =FALSE,warning=FALSE}
```
```

```
# Delete the leading spaces
library(stringr)
train$comment_text = str_trim(train$comment_text)
class(train$comment_text) # Class is 'Charcter'

# Convert comment into corpus
library(tm)
traincorpus = Corpus(VectorSource(train$comment_text))
writeLines(as.character(train$comment_text[10]))

# Case Folding
traincorpus = tm_map(traincorpus, tolower)
# Remove Stop Words
traincorpus = tm_map(traincorpus,removeWords,stopwords('english'))
# Remove Punctuation marks
traincorpus = tm_map(traincorpus,removePunctuation)
# Remove Numbers
traincorpus = tm_map(traincorpus,removeNumbers)
# Remove unnecessary spaces
traincorpus = tm_map(traincorpus,stripWhitespace)
# Stemming
traincorpus = tm_map(traincorpus, stemDocument)

########################################################################
###########
test$comment_text = str_trim(test$comment_text)
testcorpus = Corpus(VectorSource(test$comment_text))
testcorpus = tm_map(testcorpus, tolower)
testcorpus = tm_map(testcorpus,removeWords,stopwords('english'))
testcorpus = tm_map(testcorpus,removePunctuation)
testcorpus = tm_map(testcorpus,removeNumbers)
testcorpus = tm_map(testcorpus,stripWhitespace)
testcorpus = tm_map(testcorpus, stemDocument)

########################################################################
###########

dtm = DocumentTermMatrix(traincorpus)
dtm = removeSparseTerms(dtm, 0.99)
train_dataset = as.data.frame(as.matrix(dtm))
train_dataset$toxic = NULL
train_dataset$severe_toxic = NULL
train_dataset$obscene = NULL
train_dataset$threat = NULL
train_dataset$insult = NULL
train_dataset$identity_hate = NULL

########################################################################
###########

dtm = DocumentTermMatrix(testcorpus)
```

~~~~
dtm = removeSparseTerms(dtm, 0.99)
test_dataset = as.data.frame(as.matrix(dtm))

########################################################################
##############

colnamesSame = intersect(colnames(train_dataset),colnames(test_dataset))

train_dataset = train_dataset[ , (colnames(train_dataset) %in% colnamesSame)]
test_dataset = test_dataset[ , (colnames(test_dataset) %in% colnamesSame)]


########################################################################
##############


```

#Modelling using XGBoost
##Toxic Calculation
We calculate the various targets and predict the probablities
```{r,message=FALSE,warning=FALSE}



dataset2 = train_dataset
dataset2$toxic = train$toxic
dataset2$toxic = as.factor(dataset2$toxic)
levels(dataset2$toxic) = make.names(unique(dataset2$toxic))

formula = toxic ~ .

fitControl <- trainControl(method="none",classProbs=TRUE,
summaryFunction=twoClassSummary)

xgbGrid <- expand.grid(nrounds = 500,
                       max_depth = 3,
                       eta = .05,
                       gamma = 0,
                       colsample_bytree = .8,
                       min_child_weight = 1,
                       subsample = 1)


set.seed(13)

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

predictionsToxic = predict(ToxicXGB,test_dataset,type = 'prob')
~~~~

````
########################################################################
#############################

```


##Severe Toxic Calculation
```{r,message=FALSE,warning=FALSE}

dataset2 = train_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))

formula = severe_toxic ~ .

set.seed(13)

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

predictionsSevereToxic = predict(ToxicXGB,test_dataset,type = 'prob')

########################################################################
##################

```

##Obscene Calculation
```{r,message=FALSE,warning=FALSE}

dataset2 = train_dataset
dataset2$obscene = train$obscene
dataset2$obscene = as.factor(dataset2$obscene)
levels(dataset2$obscene) = make.names(unique(dataset2$obscene))

formula = obscene ~ .

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

predictionsObscene = predict(ObsceneXGB,test_dataset,type = 'prob')

########################################################################
##################

```

##Threat Calculation
````
````

````
```{r,message=FALSE,warning=FALSE}

dataset2 = train_dataset
dataset2$threat = train$threat
dataset2$threat = as.factor(dataset2$threat)
levels(dataset2$threat) = make.names(unique(dataset2$threat))

formula = threat ~ .

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

predictionsThreat = predict(ThreatXGB,test_dataset,type = 'prob')

######################################################################
##################

```



##Insult Calculation
```{r,message=FALSE,warning=FALSE}

dataset2 = train_dataset
dataset2$insult = train$insult
dataset2$insult = as.factor(dataset2$insult)
levels(dataset2$insult) = make.names(unique(dataset2$insult))

formula = insult ~ .

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

predictionsInsult = predict(InsultXGB,test_dataset,type = 'prob')

######################################################################
##################

```



##Identity Hate Calculation
```{r,message=FALSE,warning=FALSE}

dataset2 = train_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))

formula = identity_hate ~ .
````
````

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

predictionsHate = predict(HateXGB,test_dataset,type = 'prob')

################################################################################
####################

```

#Creating the Submissions
```{r,message=FALSE,warning=FALSE}

submission$toxic = predictionsToxic$X1
submission$severe_toxic = predictionsSevereToxic$X1
submission$obscene = predictionsObscene$X1
submission$threat = predictionsThreat$X1
submission$insult = predictionsInsult$X1
submission$identity_hate = predictionsHate$X1

# Write it to file
write.csv(submission, 'ToxicCommentsSubmission.csv', row.names = F)

```
````
````