

# Toxic Comment Classification

- Identify and classify toxic online comments

- Rohit Haritash  
- rohit.haritash@gmail.com  
30th April 2018

## Contents

Toxic Comment Classification .....	1
Chapter 1 .....	3
Introduction.....	3
1.1    Problem statement .....	3
1.2    Dataset.....	3
Chapter 2 .....	5
Methodology .....	5
2.1 Exploratory Data Analysis .....	5
2.1.1 Top 20 important toxic words .....	7
2.1.2 Top 20 important severe toxic words .....	7
2.1.3 Top 20 important obscene words .....	8
2.1.4 Top 20 important threat words.....	8
2.1.5 Top 30 important insult words.....	9
2.1.6 Top 30 important identity hate words .....	9
2.2 Text pre-processing.....	10
2.3 Modeling.....	10
2.3.1 Top ranked train features .....	11
Chapter 3 .....	15
Results & Discussion .....	15
3.1 Results.....	15
3.2 Models Evaluation .....	15
3.3 Result Table.....	16
3.4 Further Performance Enhancements .....	17
Appendix A - R Code .....	18
toxicCommentClassifEDA.R.....	18
xgboostClassifV1.R .....	21

## Chapter 1

# Introduction

---

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.

The Conversation AI team, a research initiative founded by Jigsaw and Google (both a part of Alphabet) are working on tools to help improve online conversation. 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 they've built a 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).

## 1.1 Problem statement

The challenge is to build a classification model that is capable of detecting different type of toxicity among online comments like threats, insult, identity hate better current models in use.

## 1.2 Dataset

Our dataset is consisting of comments from Wikipedia's talk edit pages. The training dataset consist of 8 columns which are

Column	Description
<i>id</i>	Id of comment
<i>comment_text</i>	Input comments
<i>toxic</i>	Target class
<i>sever_toxic</i>	Target class
<i>odscence</i>	Target class
<i>threat</i>	Target class
<i>insult</i>	Target class
<i>identity_hate</i>	Target class

**Table1. training dataset structure**

Below is the sample of training dataset.

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
1	0000997932d777bf	ExplanationWhy the edits made under my username Hardcore Metallica Fan were reverted They werent vandalism just closure on some GAs after I voted at New York Dolls FAC And please dont remove the template from the talk page since Im retired now892053827	0	0	0	0	0	0
2	000103f0d9cfb60f	Dawww! He matches this background colour Im seemingly stuck with Thanks talk 21:51 January 11 2016 UTC	0	0	0	0	0	0
3	000113f07ec002fd	Hey man Im really not trying to edit war Its just that this guy is constantly removing relevant information and talking to me through edits instead of my talk page He seems to care more about the formatting than the actual info	0	0	0	0	0	0
4	0001b41b1c6bb37e	Morel cant make any real suggestions on improvement I wondered if the section statistics should be later on or a subsection of types of accidents I think the references may need tidying so that they are all in the exact same format ie date format etc I can do that later on if noone else does first if you have any preferences for formatting style on references or want to do it yourself please let me knowThere appears to be a backlog on articles for review so I guess there may be a delay until a reviewer turns up Its listed in the relevant form eg Wikipedia:Good_article_nominations#Transport	0	0	0	0	0	0
5	0001d958c54c6e35	You sir are my hero Any chance you remember what page thats on	0	0	0	0	0	0

**Fig1. Training data sample**

A different test dataset was provided by Kaggle for submission purpose. As part of pre-processing and feature engineering, we are going to process both training and test dataset.

For the purpose of model validation and testing, main training data was split into cvtrain and cvtest.

Main test dataset was used for Kaggle submission.

## Chapter 2

# Methodology

---

The solution for this problem is divided into three parts. First basic level of exploratory data analysis(EDA) is performed. Results from EDA can be utilised to verify the feature selected by the model. In the second phase, pre-processing of text data is done. In the last phase, classification model was trained using XGBoost and some parameters were tuned for performance enhancement.

The models were trained on Microsoft Azure VM running Linux Ubuntu with 32 GB RAM and 4 processing cores.

## 2.1 Exploratory Data Analysis

The analysis will start with basic exploratory analysis. It will help us in understanding the structure and specific features for a particular target class. We have 6 target categories, so we have to find out which features are related to which class.

Below is a visualization of top 20 words in our input comments.

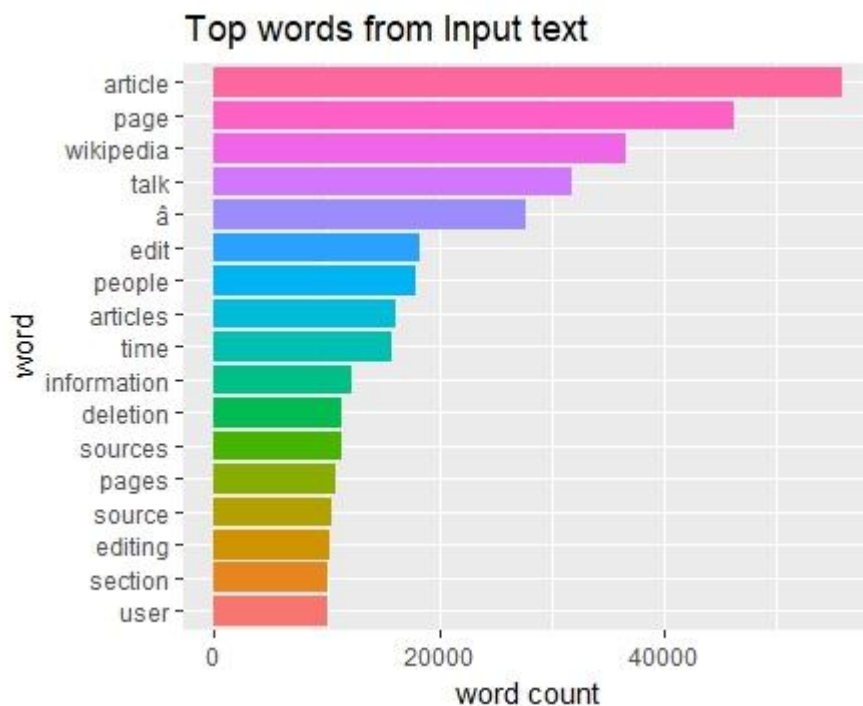


Fig.2 Frequency of top words from input comments

In the next step, frequency of words was calculated for each target class. Upon examining the result, it observed that a feature can exist in multiple target class. So we know that a single comment may belong to more than one target class. So a word can occur or present in many categories, which can be a combination of 6 target class. The combinations of `toxic, severe toxic, obscene, threat, insult and identity hate` will create unique categories. We will display those categories here. There exist 41 such cases where a single comment can belong to multiple categories. Below is a snap shot.

```
# A tibble: 41 x 7
# Groups:   toxic, severe_toxic, obscene, threat, insult [?]
  toxic severe_toxic obscene threat insult identity_hate total
  <int>      <int>    <int>  <int>  <int>      <int>    <int>
1     0          0        0      0      0          0 9928641
2     0          0        0      0      0          1  4854
3     0          0        0      0      1          0 17540
4     0          0        0      0      1          1  1801
5     0          0        0      1      0          0   765
6     0          0        0      1      1          0   934
7     0          0        1      0      0          0 30128
8     0          0        1      0      0          1   142
9     0          0        1      0      1          0 13369
10    0          0        1      0      1          1  1280
# ... with 31 more rows
```

**Fig.3 Comments with multiple categories**

Now we will look at the important words in each target categories. The importance of a word was calculated using **td-idf (Term frequency – Inverse document frequency) static**.

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.

Below are top 10 important words using td-idf.

Important Word	Category	TF-IDF
<i>bleachanhero</i>	36	0.820
<i>nl33ers</i>	29	0.744
<i>bunksteve</i>	31	0.508
<i>drink</i>	36	0.363
<i>supertr0ll</i>	6	0.286
<i>criminalwar</i>	28	0.243
<i>nl33ersi</i>	29	0.208
<i>faggot</i>	30	0.201
<i>shomron</i>	33	0.161
<i>fucksex</i>	35	0.153

**Table 2 – Top important words using TF-IDF**

After looking at the important words of overall input dataset, we will analysis top 20 important words of target class. This could give us a basic understanding about the important and relevant

features for our model. **These top words are calculate using tf-idf but without any text pre-processing.**

### 2.1.1 Top 20 important toxic words

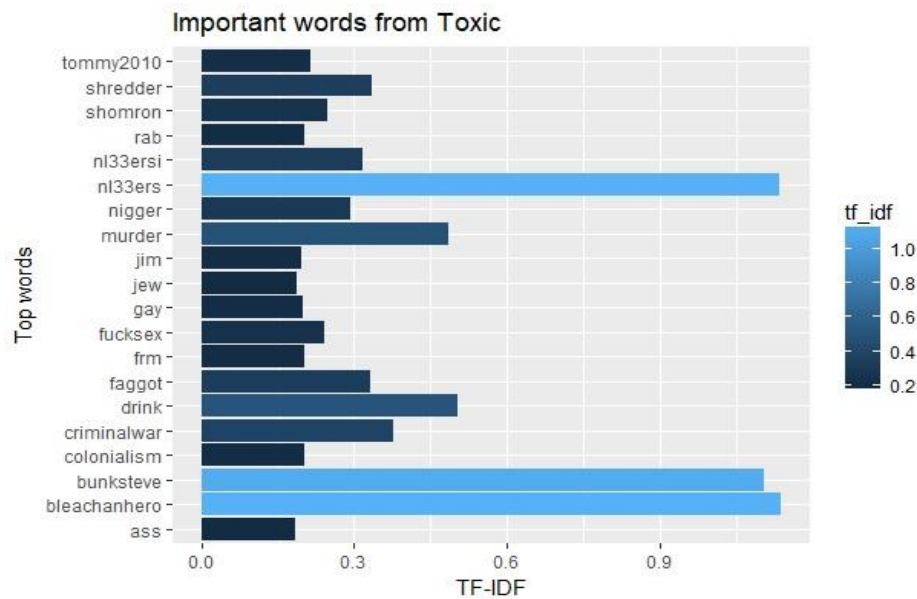


Fig.3 Important toxic words

### 2.1.2 Top 20 important severe toxic words

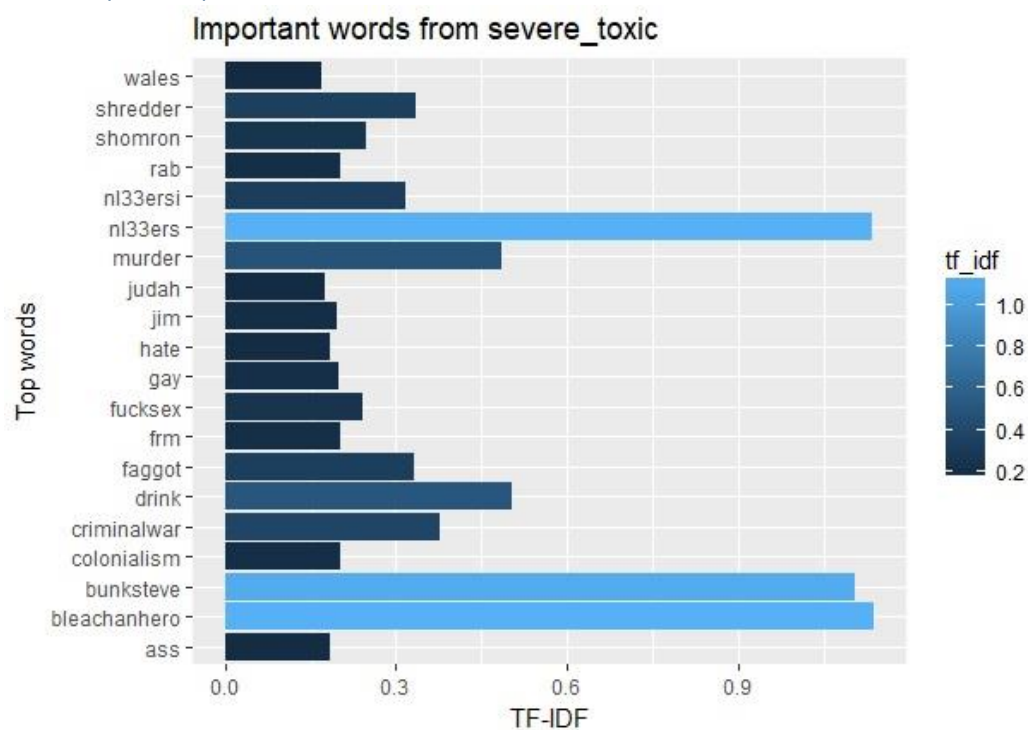


Fig.4 Important sever toxic words

### 2.1.3 Top 20 important obscene words

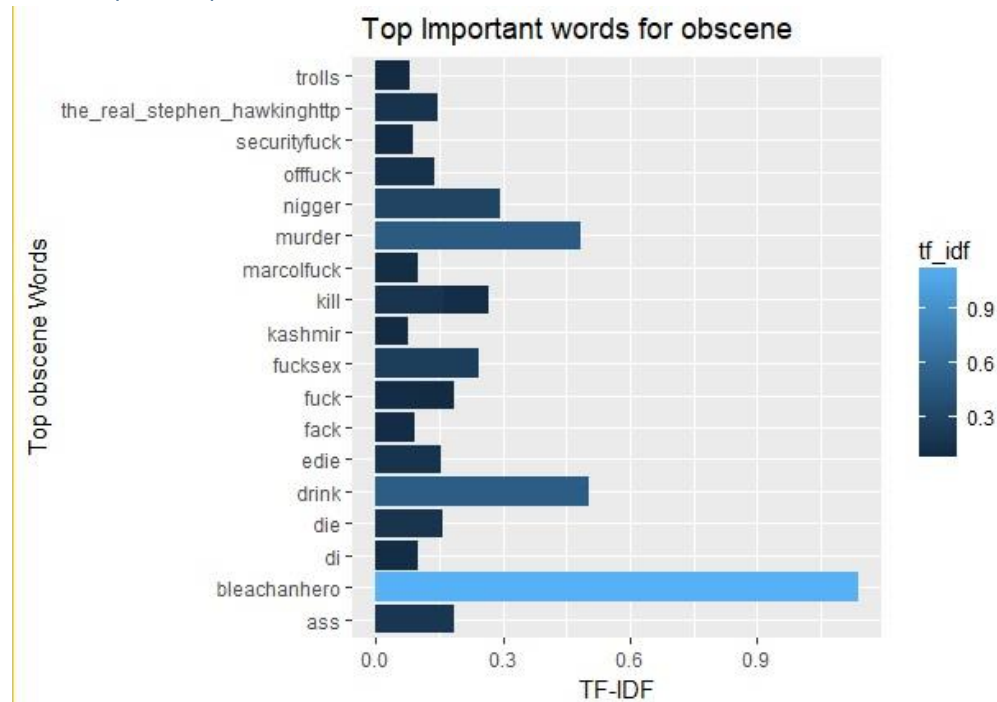


Fig.5 Important toxic words

### 2.1.4 Top 20 important threat words

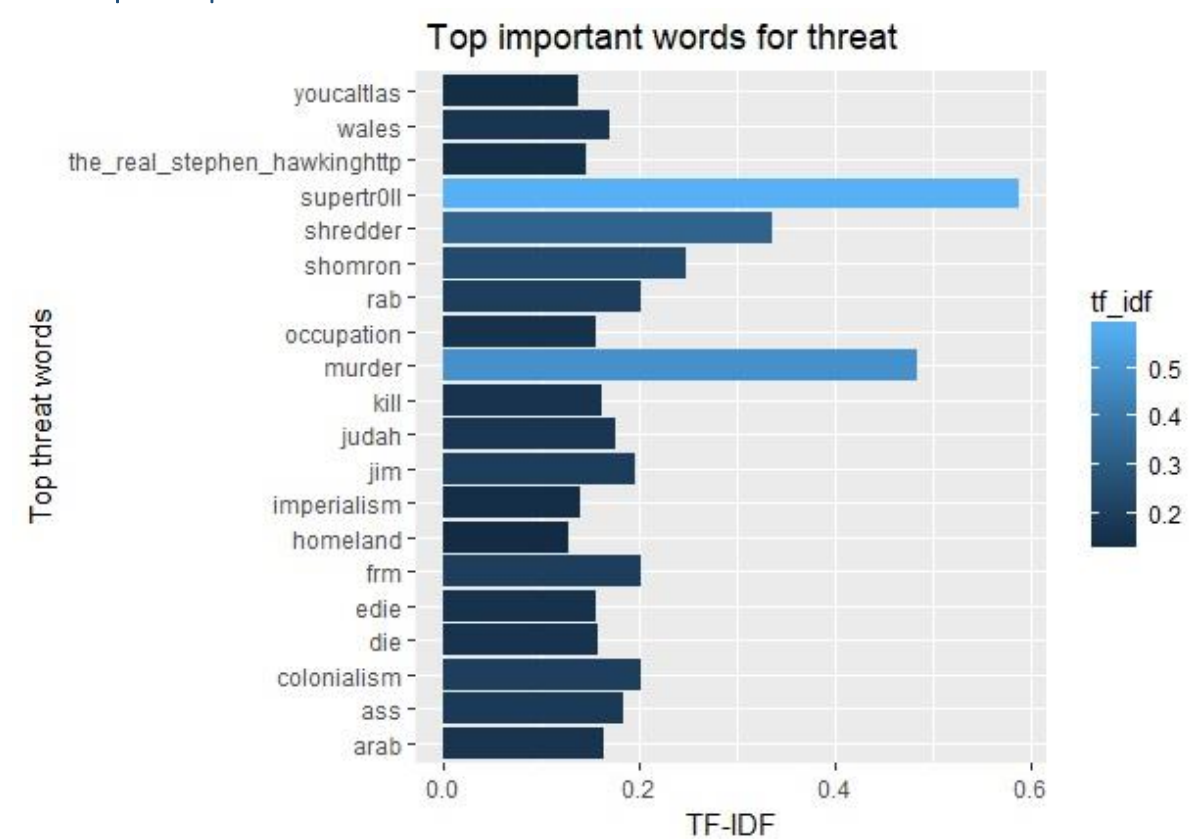


Fig.6 Important toxic words



Note – Increasing number of words for insult and identity hate to include more words.

### 2.1.5 Top 30 important insult words

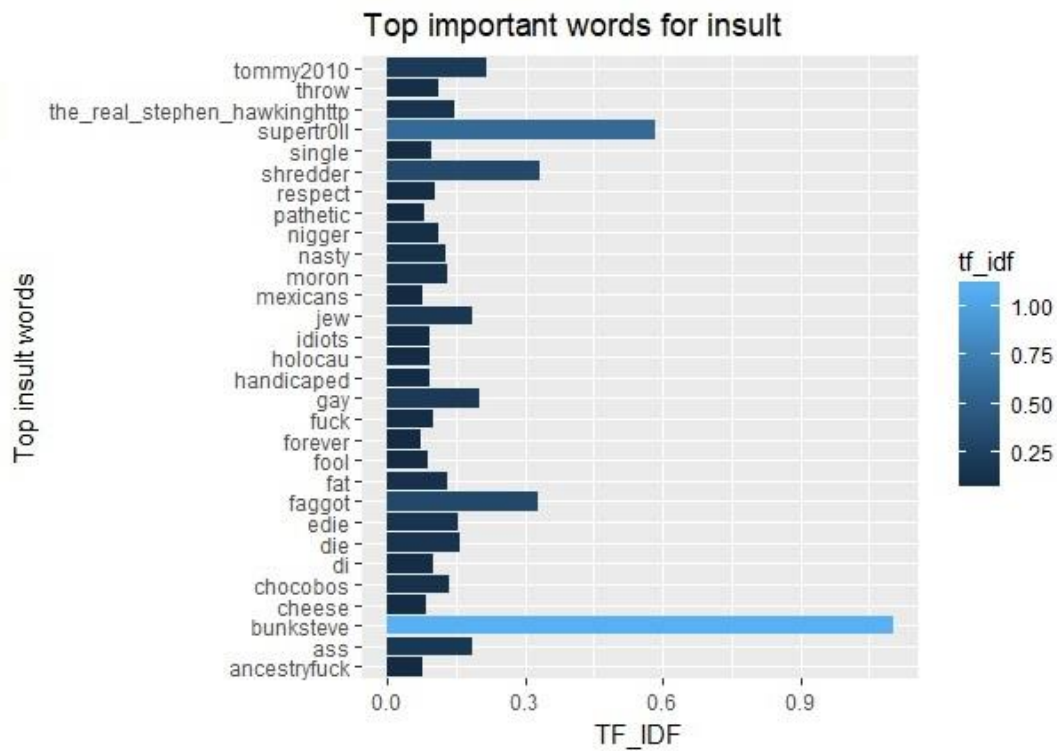


Fig.7 Important toxic words

### 2.1.6 Top 30 important identity hate words

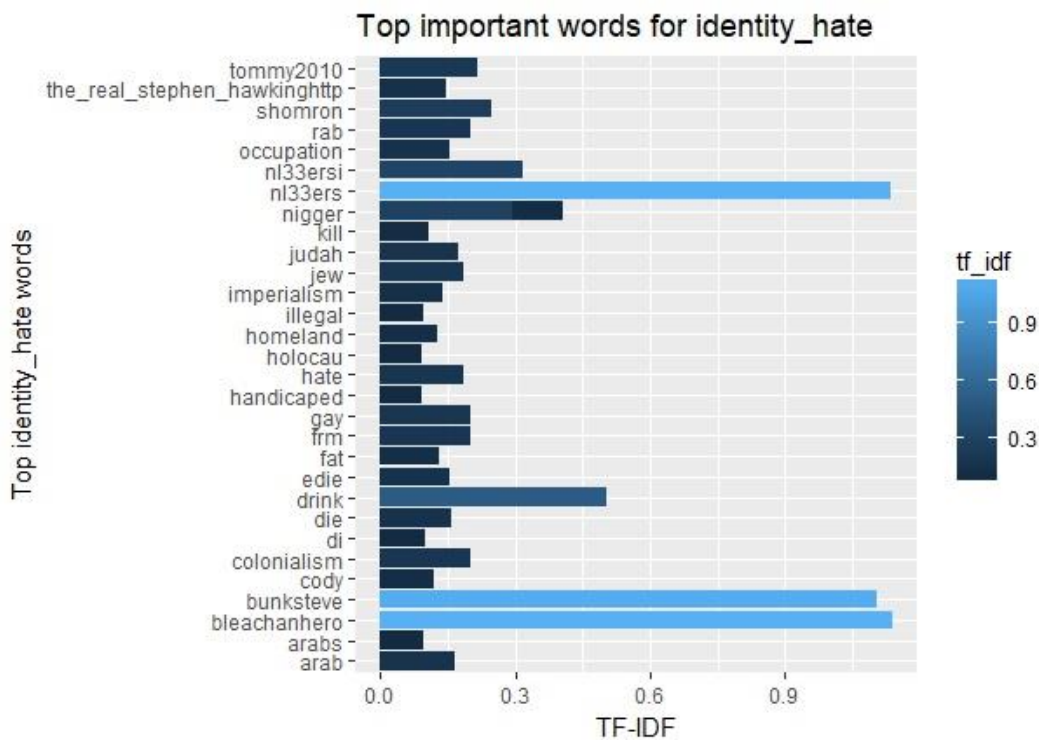
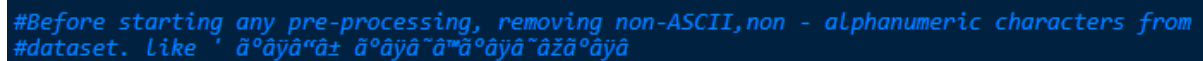


Fig.8 Important toxic words

## 2.2 Text pre-processing

After EDA, the textual data was pre-processed. Important features were extracted for the model. The whole process is described below.

Our input text consists of lots of non-ASCII characters, which can create nuisance during feature engineering. Before any pre-processing, all of the non-ASCII characters were removed.



```
#Before starting any pre-processing, removing non-ASCII, non - alphanumeric characters from  
#dataset. Like ' ā°āÿâ°â± ā°āÿâ°â°â°āÿâ°âžž°āÿâ
```

**Fig.9 Non-ASCII characters from input dataset**

Next all the non-alphanumeric and system keywords like '\n' were removed. Now the data was converted onto corpus. This is the first step of text pre-processing and model input preparation. All the text was converted into smaller case using case folding. All the numbers, punctuations and white spaces were removed. For the stop word removal, inbuilt stop words dictionary was not sufficient. So to remove noise and stop words a new dictionary was created using default 'stop words dictionary' and a custom list provided by <https://github.com/stopwords-iso/stopwords-en/blob/master/stopwords-en.txt>. Using this combined list, stop words were removed from the input. To further reduce the dimensions, stemming was performed. To create a classification model using xgboost, we need our input into numeric. So our corpus was converted into document-term matrix(DTM). In a DTM all the input comments were rows and highly important words (calculated using tf-idf) were features/columns.

The whole pre-processing was performed on main train and test dataset. Since both train and test datasets are different, so they both have different features. For model to work, we used intersection to get the common features from train and test. After this process a new train and validation/test dataset (cvtrain, and cvtest) were created from main train dataset. 'cvtrain' dataset was used for model training and 'cvtest' was used for model validation and testing.

## 2.3 Modeling

Many aspects were analysed before selecting XGBoost for this classification problem. Some of the most important reasons for choosing XGBoost were –

- Since it's a text analysis problem, it was evident from the early analysis that this has lots of predictors. A lots of computation was required to solve higher dimension problem. XGBoost supports automatic parallel computation on windows and linux. It can easily scale up to such computation intense problem.
- Xgboost was capable to work well on sparse and dense matrixes. A text mining problem wich used tf-idf can result is a highly sparse matrix. So XGBoost was selected.
- XGBoost is known to give very high prediction on both classification and regression setting.

- Variable importance can easily be estimated with XGBoost. It could help us in investigating whether right features were used in training.

Function **createDataPartition** from caret package was used to split train and validation/test splits to minimise biased. This classification problem consists of 6 target class. This problem was approached as six different sub-problems. Six different models were trained for each target class.

### 2.3.1 Top ranked train features

Importance variable for each model of version 3(v3) were calculated. Below are important or high ranking features for each target class used by models.

#### 2.3.1.1 Top ranked trained features for toxic model

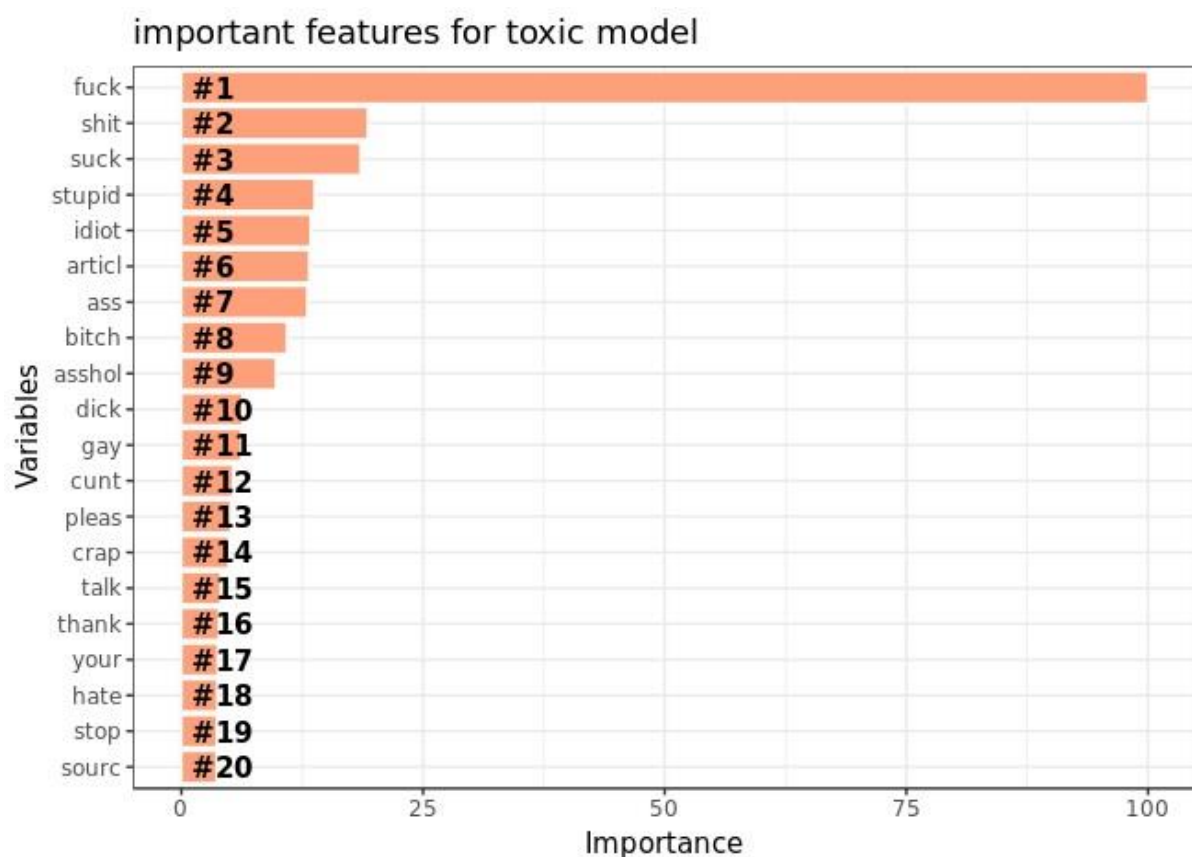


Fig.10 Top features for toxic model

### 2.3.1.2 Top ranked trained features for severe toxic model

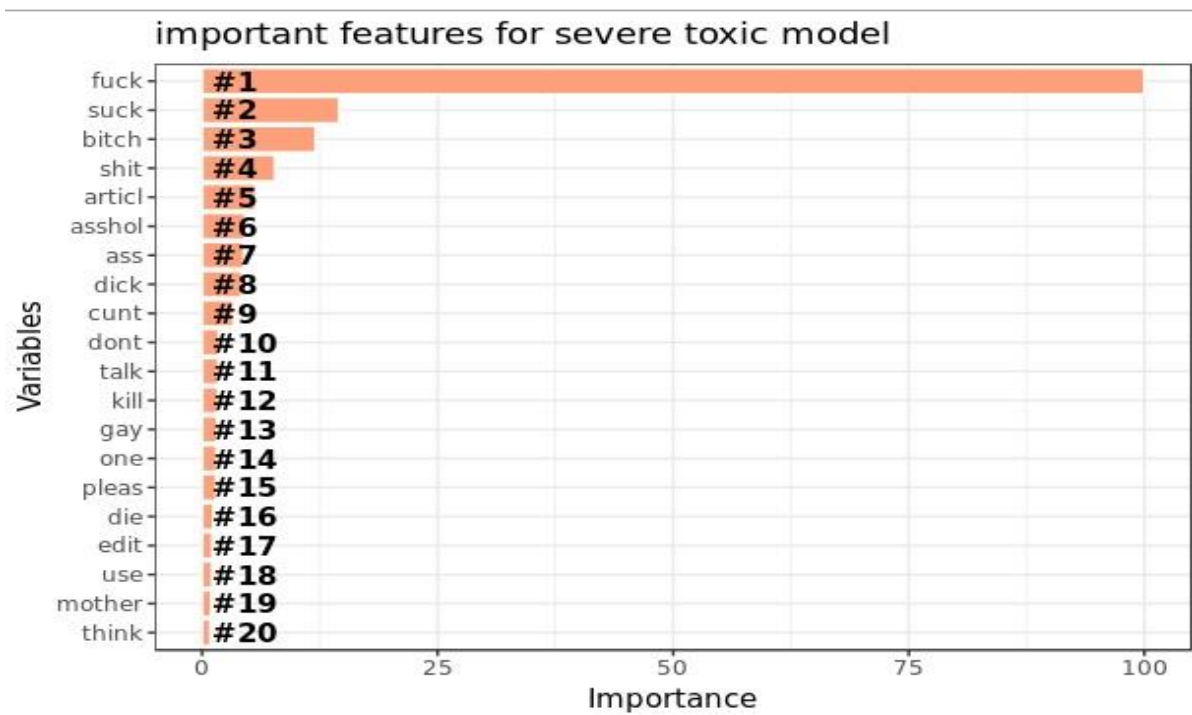


Fig.11 Top features for sever toxic

### 2.3.1.3 Top ranked trained features for obscene model

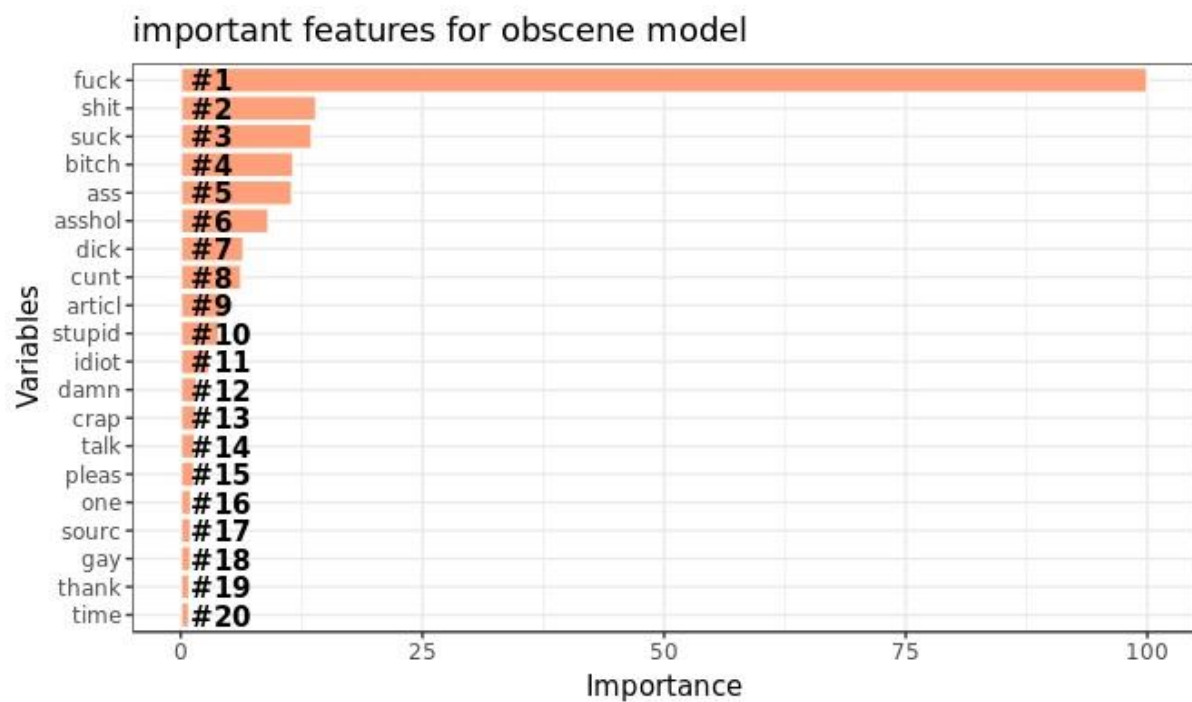


Fig.12 Top features for Obscene model

#### 2.3.1.4 Top ranked trained features for threat model

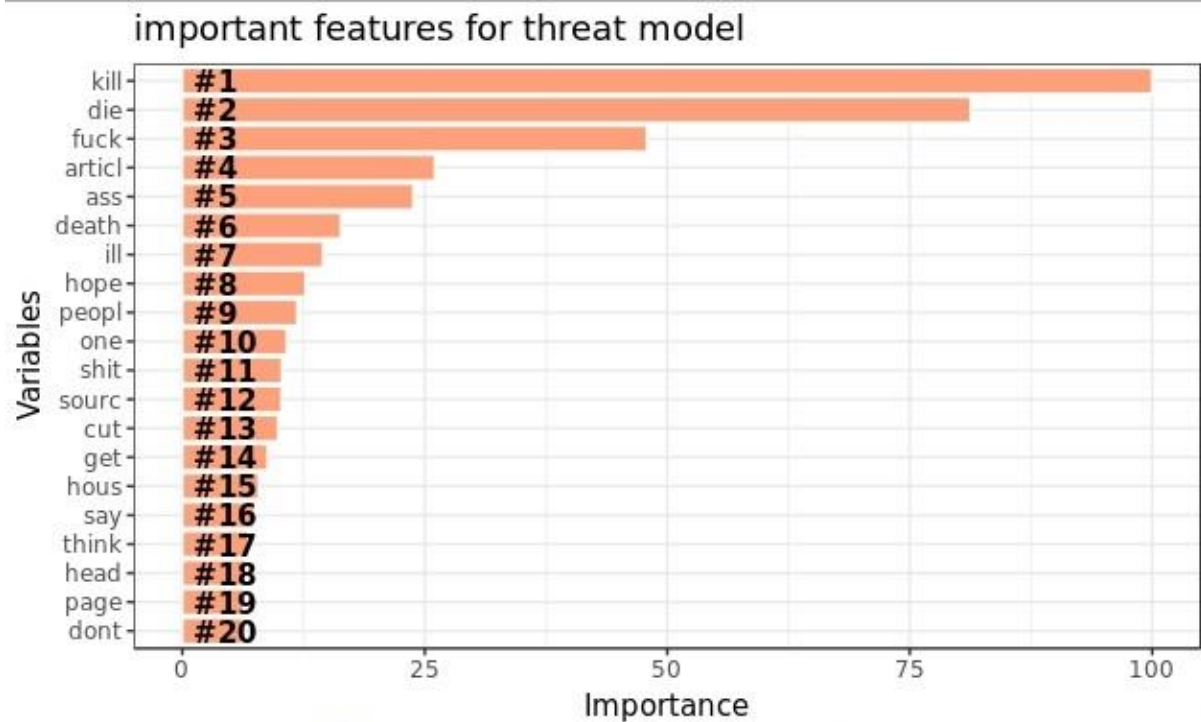


Fig.13 Top features for threat model

#### 2.3.1.5 Top ranked trained features for insult model

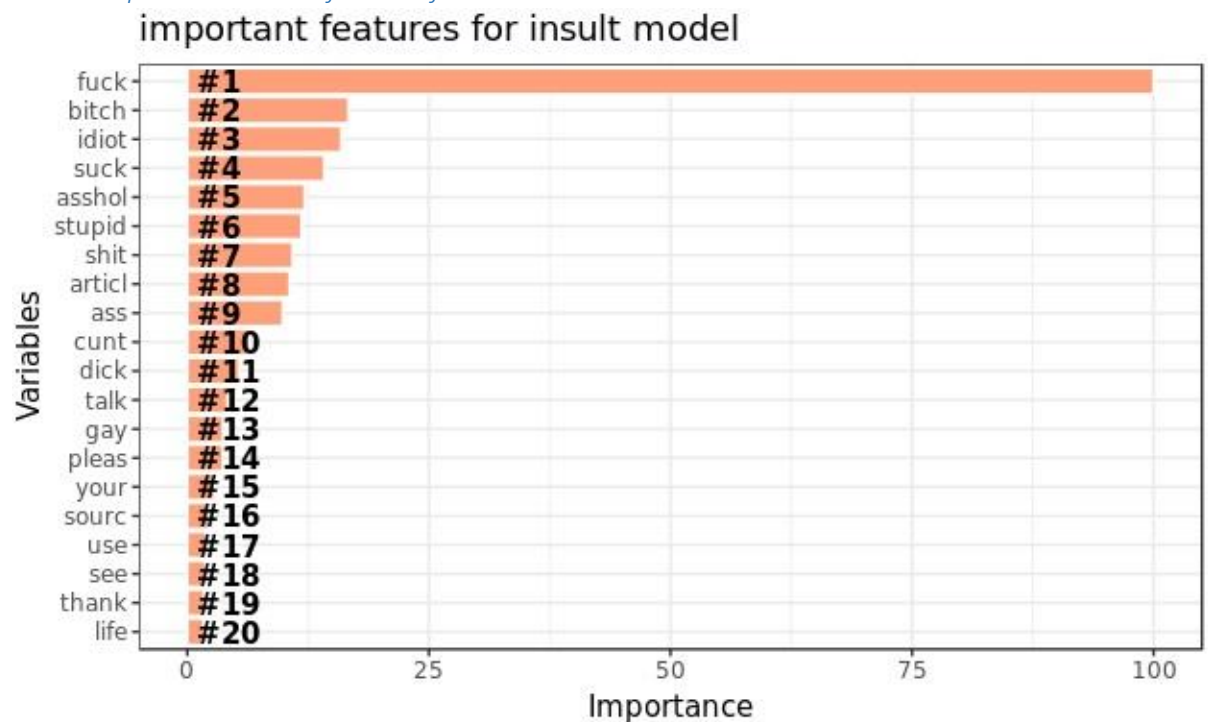


Fig.14 Top features for insult

#### 2.3.1.5 Top ranked trained features for identity hate model

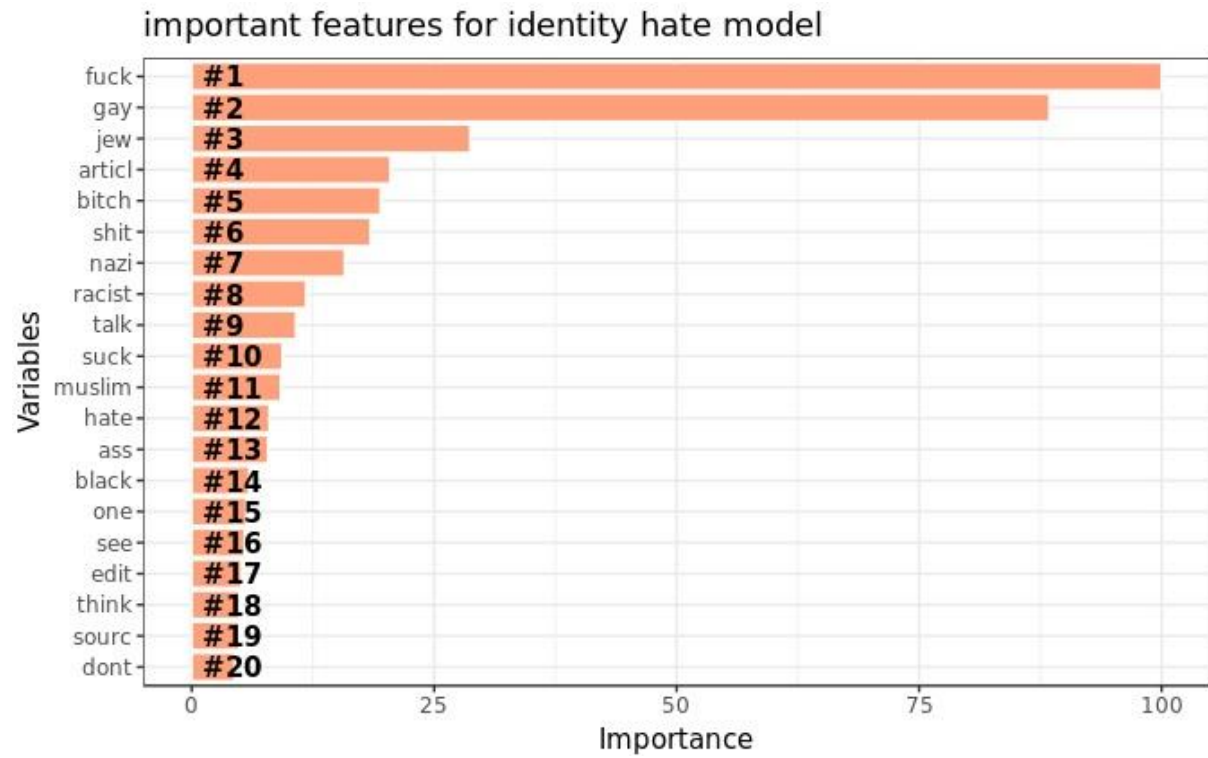


Fig.15 Top features for identity hate

As from these visualizations it is evident that variables/features used by the model make sense and were logical.

## Chapter 3

# Results & Discussion

### 3.1 Results

Total 6 models were trained for each target class. For all models, the neutral case when comment doesn't belong to any target class were the positive cases. Three different versions of control setting were used to train all the models 3 times. Parameters nround, max\_depth and eta were modified in different setting and version to see any visible performance change.

Version	nrounds	max_depth	eta	gamma	colsample_tree	min_child_weight	subsample
v1	100	6	0.1	0	0.8	1	1
v2	500	6	0.1	0	0.8	1	1
v3	500	8	0.09	0	0.8	1	1

Table 3 - Parameter Tuning Control versions

Results from all the models run are in table 5. From the table it was seen that **models for toxic, obscene and insult perform fairly while models for sever toxic, threat and identity hate performed poorly.**

### 3.2 Models Evaluation

To evaluate model performance on test/validation dataset confusion matrix was used. We were interested in true positive rate (sensitivity) and true negative rate (specificity). The models for toxic, obscene and insult performed fairly while performance of sever toxic, threat and identity hate models were poor. The test accuracy is very high due to dominating feature of test sensitivity. The results of cross validation and test evaluation using confusion matrix are shown in result table 5. From table 4 it can be seen that the reason for poor performance of sever toxic, threat and identity hate models were due to class imbalance problem.

Target class	Positive Cases	Negative Cases
toxic	144277	15249
severe toxic	157976	1595
obscene	151122	8449
threat	159093	478
insult	151694	7877
identity hate	158166	1405

Table 4 – Signs of class Imbalance problem (counts in yellow)

No evidence of overfitting was found since variation between cross-validation and test results was not much.

### 3.3 Result Table

Class	Control version	No of folds	samples	predictors	CV ROC	CV Sensitivity	CV Specificity	Test Accuracy	Test Sensitivity	Test Specificity
<b>toxic</b>	v1	3	111700	1072	90.08	99.46	46.61	94.25	99.38	45.9
	v2	3	111700	1072	92.18	99.24	50.71	94.55	99.17	50.94
	v3	5	111700	1072	92.3	99.25	51.18	94.58	99.15	51.48
<b>sever Toxic</b>	v1	3	111701	1072	96.2	99.88	13.07	99.07	99.9	16.32
	v2	3	111701	1072	96.28	99.8	20.32	99.04	99.85	19.25
	v3	5	111701	1072	96.45	99.81	18.71	99.07	99.85	21.34
<b>Obscence</b>	v1	3	111701	1072	94.43	99.48	64.2	97.56	99.51	62.71
	v2	3	111701	1072	95.33	99.45	64.17	97.68	99.54	64.33
	v3	5	111701	1072	95.22	99.06	64.12	96.9	99.35	65.48
<b>Threat</b>	v1	3	111701	1072	94.87	99.97	11.34	99.7	99.96	13.29
	v2	3	111701	1072	94.95	99.95	18.21	99.71	99.96	17.48
	v3	5	111701	1072	95.02	99.45	19.22	99.64	99.94	19.45
<b>Insult</b>	v1	3	111700	1072	92.4	99.25	45.33	96.5	99.27	43.25
	v2	3	111700	1072	93.85	99.12	50.3	96.67	99.1	49.98
	v3	5	111700	1072	93.9	99.05	52.32	96.42	99.24	54.69
<b>Identity Hate</b>	v1	3	111700	1072	91.99	99.96	9.8	99.16	99.95	10.21
	v2	3	111700	1072	91.99	99.91	14.12	99.15	99.92	12.35
	v3	5	111700	1072	90.12	99.96	10.06	99.16	99.15	9.7



## 3.4 Further Performance Enhancements

- To improve the model's performance, we can mitigate the class imbalance problem. We can either go under sampling or oversampling to balance the classes for model training.
- Due to hardware and time constraint, only 3 parameters were tuned and tried. We can try and tune other parameters also.
- During feature engineering, we can use n-gram approach and semantic analysis for feature selection.

# Appendix A - R Code

## toxicCommentClassifEDA.R

```
1. # Toxic Comment Classification
2.
3. rm(list = ls())
4. library(tidyverse)
5. library(tidytext)
6. library(ggplus)
7. library(tm)
8.
9.
10. #reading training dataset
11. train <-
12.   read.csv(
13.     "train.csv",
14.     sep = ',',
15.     header = TRUE,
16.     stringsAsFactors = FALSE
17.   )
18.
19. )
20. str(train)
21.
22. train$comment_text <-
23.   iconv(train$comment_text, 'UTF-8', 'ASCII', sub = "")
24.
25. #To remove all characters not in the list of alphanumeric and punctuation
26. train$comment_text <-
27.   str_replace_all(train$comment_text, "[^[:graph:]]", " ")
28.
29.
30. #After analyzing the train data, we have 8 columns :
31. #id - comment id
32. #comment_text - wikipedia comment
33. #toxic : comment clasiification class
34. #severe_toxic : comment clasiification class
35. #obscene : comment clasiification class
36. #threat : comment clasiification class
37. #insult : comment clasiification class
38. #identity_hate: comment clasiification class
39.
40.
41.
42. ##### Performing Exploratory data analysis #####
43.
44. # now look at top common words
45. visualize_topwords <- function(train) {
46.   train %>%
47.     unnest_tokens(word, comment_text) %>% #tokenizing comments
48.     anti_join(stop_words) %>% #removing stop words
49.     count(word, sort = TRUE) %>% #counting tokens
50.     filter(n > 10000) %>% #filtering words greater than 10000
51.     mutate(word = reorder(word, n)) %>%
52.     ggplot() +
53.     geom_col(mapping = aes(x = word, y = n, fill = word),
54.       show.legend = FALSE) +
55.     labs(x = 'word', y = 'word count', title = 'Top words from Input text') +
56.     coord_flip()
```

```

57. }
58.
59. visualize_topwords(train)
60.
61. #First step of any text analysis or text mining is tokenization.
62. #Converting our text(sentences or comments) into individual words or one-token-per-
    row format
63.
64. training_words <- train %>%
65.   unnest_tokens(word, comment_text)
66.
67. # counting frequency of a token in each target class
68. trainWords <- training_words %>%
69.   count(toxic,
70.         severe_toxic,
71.         obscene,
72.         threat,
73.         insult,
74.         identity_hate,
75.         word) %>%
76.   ungroup()
77.
78. # upon examining trainwords, it is observed that a word is present in multiple targ
    etclass.
79. #So we know that a single comment may belong to more than one target class. So a wo
    rd can occur
80. # or present in many categories, which can be a combination of 6 target class.
81. #Lets find out , how many such unique categories exist.
82.
83. words_by_cat <- trainWords %>%
84.   group_by(toxic, severe_toxic, obscene, threat, insult, identity_hate) %>%
85.   summarise(total = sum(n))
86. # total 41 such categories exist.
87.
88. words_by_cat$category = 1:41
89. str(words_by_cat)
90.
91. # we have 41 unique combined categories, Adding this information into trainwords.
92. final_trainwords <- left_join(
93.   trainWords,
94.   words_by_cat,
95.   by = c(
96.     "toxic",
97.     "severe_toxic",
98.     "obscene",
99.     "threat",
100.    "insult",
101.    "identity_hate"
102.  )
103. )
104.
105. #removing stop words from final_trainwords
106. final_trainwords <- final_trainwords %>%
107.   anti_join(stop_words)
108.
109. # now our dataset contains token with its count and to which category it bel
    ong.
110. #We will use tf-
    idf statistic to measure how important a word is a document. In our dataset,
111. # we have multiple documents(comments) belongs to many categories ie. toxic,
    threat,etc.
112.
113. #calculating tf-idf for each token
114. final_trainwords <- final_trainwords %>%
115.   bind_tf_idf(word, category, n)
116.

```

```

117.     #Lets examine some important words.ie words with high tf-idf
118.     final_trainwords %>%
119.       select(word, category, tf_idf) %>%
120.       arrange(desc(tf_idf))
121.
122.     #These are some of the important words and categories to which they belong
123.     # word          category tf_idf
124.     #<chr>          <int>  <dbl>
125.     #1 bleachanhero      36  0.820
126.     #2 nl33ers           29  0.744
127.     #3 bunksteve         31  0.508
128.     #4 drink             36  0.363
129.     #5 supertr0ll        6  0.286
130.     #6 criminalwar      28  0.243
131.     #7 nl33ersi          29  0.208
132.     #8 faggot            30  0.201
133.     #9 shomron           33  0.161
134.     #10 fucksex          35  0.153
135.
136.     #Let's visualize these important words per unique cartegories
137.     # Due to limitation of hardware and Rstudio top words of 41 unique categorie
s are plotted and stored
138.     #in a pdf file 'important_words_percat.pdf' .
139.     fillColor = "#F1C40F"
140.     gg1 <- final_trainwords %>%
141.       arrange(desc(tf_idf)) %>%
142.       group_by(category) %>%
143.       top_n(15) %>%
144.       ungroup() %>%
145.       ggplot() +
146.       geom_col(mapping = aes(
147.         x = word,
148.         y = tf_idf,
149.         color = category ,
150.         fill = tf_idf
151.       )) +
152.       labs(x = NULL, y = 'tf-idf') +
153.       # facet_wrap(~category, ncol = 2, scales = "free")
154.       coord_flip()
155.
156.     pdf('important_words_percatTF_IDF.pdf')
157.     facet_multiple(
158.       plot = gg1,
159.       facets = "category",
160.       ncol = 1,
161.       nrow = 1
162.     )
163.     dev.off()
164.
165.     # we can visuaalize top important class relevant to target categories
166.     plot_importantwords <- final_trainwords %>%
167.       arrange(desc(tf_idf))
168.
169.     #1. top tf-idf important words for Toxic
170.     plot_importantwords %>%
171.       filter(toxic == 1) %>%
172.       top_n(20) %>%
173.       ggplot() +
174.       geom_col(mapping = aes(x = word, y = tf_idf, fill = tf_idf)) +
175.       labs(x = 'Top words', y = "TF-
IDF", title = "Important words from Toxic") +
176.       coord_flip()
177.
178.     #2. top tf-idf important words for severe_toxic
179.     plot_importantwords %>%
180.       filter(severe_toxic == 1) %>%

```

```

181.         top_n(20) %>%
182.         ggplot() +
183.         geom_col(mapping = aes(x = word, y = tf_idf, fill = tf_idf)) +
184.         labs(x = "Top words", y = "TF-
IDF", title = "Important words from severe_toxic") +
185.         coord_flip()
186.
187.     #3. Top tf-idf important words for obscene
188.     plot_importantwords %>%
189.     filter(obscene == 1) %>%
190.     top_n(20) %>%
191.     ggplot() +
192.     geom_col(mapping = aes(x = word, y = tf_idf, fill = tf_idf)) +
193.     labs(x = "Top obscene Words", y = "TF-
IDF", title = " Top Important words for obscene") +
194.     coord_flip()
195.
196.     #4. Top tf-idf important words for threat
197.     plot_importantwords %>%
198.     filter(threat == 1) %>%
199.     top_n(20) %>%
200.     ggplot() +
201.     geom_col(mapping = aes(x = word, y = tf_idf, fill = tf_idf)) +
202.     labs(x = "Top threat words", y = "TF-
IDF", title = " Top important words for threat") +
203.     coord_flip()
204.
205.     #5. Top tf-idf important words for insult
206.     plot_importantwords %>%
207.     filter(insult == 1) %>%
208.     top_n(30) %>%
209.     ggplot() +
210.     geom_col(mapping = aes(x = word, y = tf_idf, fill = tf_idf)) +
211.     labs(x = "Top insult words", y = "TF_IDF", title = "Top important words fo
r insult") +
212.     coord_flip()
213.
214.     #6. Top tf-idf important words for identity_hate
215.     plot_importantwords %>%
216.     filter(identity_hate == 1) %>%
217.     top_n(30) %>%
218.     ggplot() +
219.     geom_col(mapping = aes(x = word, y = tf_idf, fill = tf_idf)) +
220.     labs(x = "Top identity_hate words", y = "TF-
IDF", title = "Top important words for identity_hate") +
221.     coord_flip()

```

## xgboostClassifV1.R

```

1. #loading requiried libraries
2. rm(list = ls())
3. library(tidyverse) #used in EDA
4. library(tidytext) #used in EDA
5. library(tm) #used in text-preprocessing
6. library(xgboost) #xgeboost classifier
7. library(caret) #use for train,test,cross validation
8. library(e1071)
9.
10. # Reading train and test data

```

```

11. train <-
12.   read.csv("train.csv", header = TRUE, stringsAsFactors = FALSE)
13. test <-
14.   read.csv("test.csv", header = TRUE, stringsAsFactors = FALSE)
15. head(train)
16. head(test)
17.
18. #Before starting any pre-processing, removing non-
    ASCII,non - alphanumeric characters from
19. #dataset. like ' ä°äÿâ"â± ä°äÿâ~â"ä°äÿâ~âžã°äÿâ
20. train$comment_text = gsub("'|\"'|'|"'"|\"|\n|,|\\.|_|\\.|\\+|\\\\-
    |\\/|=|\\(|\\)|'",
21.                             "",
22.                             train$comment_text)
23. test$comment_text = gsub("'|\"'|'|"'"|\"|\n|,|\\.|_|\\.|\\+|\\\\-
    |\\/|=|\\(|\\)|'",
24.                             "",
25.                             test$comment_text)
26. train$comment_text <-
27.   iconv(train$comment_text, 'UTF-8', 'ASCII', sub = "")
28. test$comment_text <-
29.   iconv(test$comment_text, 'UTF-8', 'ASCII', sub = "")
30. train$comment_text <-
31.   str_replace_all(train$comment_text, "[^[:graph:]]", " ")
32. test$comment_text <-
33.   str_replace_all(test$comment_text, "[^[:graph:]]", " ")
34.
35. # Including extra words to default stop words.
36. stopwords <-
37.   scan("stopwords.txt", what = list(NULL, name = character()))
38. myStopwords <- c(stopwords('english'), stopwords$name)
39.
40. ##### generic function for creating Document Term Matrix using tf-
    idf #####
41.
42. makeDTM <- function(dataset) {
43.   corpus <-
44.     Corpus(VectorSource(dataset$comment_text)) #creating corpus
45.   corpus <- tm_map(corpus, tolower) # case folding
46.   corpus <- tm_map(corpus, removeNumbers) # removing numbers
47.   corpus <- tm_map(corpus, removePunctuation)
48.   corpus <- tm_map(corpus, removeWords, myStopwords)
49.   corpus <- tm_map(corpus, stemDocument)
50.   corpus <- tm_map(corpus, stripWhitespace)
51.   dtm = DocumentTermMatrix(corpus,
52.                             control = list(
53.                               weighting = function(x)
54.                                 weightTfidf(x, normalize = FALSE)
55.                             ))
56.   dtm = removeSparseTerms(dtm, 0.997)
57.   labeledTerms = as.data.frame(as.matrix(dtm))
58.   colnames(labeledTerms) <- make.names(colnames(labeledTerms))
59.   return(labeledTerms)
60. }
61.
62. #calling function to prepare Document term matrix
63. labeledTerms_train = makeDTM(train)
64. labeledTerms_test = makeDTM(test)
65.
66. labeledTerms_train$toxic = NULL
67. labeledTerms_train$severe_toxic = NULL
68. labeledTerms_train$obscene = NULL
69. labeledTerms_train$threat = NULL
70. labeledTerms_train$insult = NULL
71. labeledTerms_train$identity_hate = NULL
72.

```

```

73. ##### preparing input for xgboost or a sanity check. This function will convert
    any character
74. #or factor variable into numeric factor
75. prepareFeatures <- function(labeledTerms) {
76.   features <- colnames(labeledTerms)
77.   for (f in features) {
78.     print(f)
79.     if ((class(labeledTerms[[f]]) == "factor") ||
80.         (class(labeledTerms[[f]]) == "character")) {
81.       levels <- unique(labeledTerms[[f]])
82.       labeledTerms[[f]] <-
83.         as.numeric(factor(labeledTerms[[f]], levels = levels))
84.     }
85.
86.   }
87.   return(labeledTerms)
88.
89. }
90.
91. #calling prepareFeatures
92. labeledTerms_train = prepareFeatures(labeledTerms_train)
93. labeledTerms_test = prepareFeatures(labeledTerms_test)
94.
95. dim(labeledTerms_train)
96. dim(labeledTerms_test)
97.
98. #After looking at the features of train_dataset and test_dataset, it seems that bot
    h have different features.
99. #For model to work, train and test should have same features. So We will use common
    feature for train and test datasets.
100.   commonFeatures <-
101.     intersect(colnames(labeledTerms_train), colnames(labeledTerms_test))
102.   train_dataset <-
103.     labeledTerms_train[, (colnames(labeledTerms_train)) %in% commonFeatures]
104.   test_dataset <-
105.     labeledTerms_test[, (colnames(labeledTerms_test)) %in% commonFeatures]
106.
107.   #sanity check
108.   dim(train_dataset)
109.   dim(test_dataset)
110.
111.
112.   ##### Generic function to plot n important variables of train
    ed model #####
113.
114.   plot_impVar <- function(xgbmodel, chartTitle) {
115.     importance = varImp(xgbmodel)
116.     varImportance <-
117.       data.frame(
118.         Variables = row.names(importance[[1]]),
119.         Importance = round(importance[[1]]$Overall, 2)
120.       )
121.
122.     #creating a rank variable based on importance
123.     rankImportance <- varImportance %>%
124.       mutate(Rank = paste0('#', dense_rank(desc(Importance)))) %>%
125.       head(20)
126.     fillColor = "#FFA07A"
127.     ggplot(rankImportance, aes(x = reorder(Variables, Importance),
128.                               y = Importance)) +
129.       geom_bar(stat = 'identity',
130.               colour = "white",
131.               fill = fillColor) +
132.       geom_text(
133.         aes(x = Variables, y = 1, label = Rank),
134.         hjust = 0,

```

```

135.         vjust = .5,
136.         size = 4,
137.         colour = 'black',
138.         fontface = 'bold'
139.     ) +
140.     labs(x = 'Variables', title = chartTitle) +
141.     coord_flip() +
142.     theme_bw()
143. }
144.
145. ##### Tunable parameters and control list for XGBoost #####
#####
146. fitControl <-
147.   trainControl(
148.     method = "cv",
149.     number = 5,
150.     classProbs = TRUE,
151.     savePredictions = TRUE,
152.     summaryFunction = twoClassSummary,
153.     allowParallel = TRUE
154.   )
155.
156.
157. xgbGrid <- expand.grid(
158.   nrounds = 500,
159.   max_depth = 8,
160.   eta = .1,
161.   gamma = 0,
162.   colsample_bytree = .8,
163.   min_child_weight = 1,
164.   subsample = 1
165. )
166.
167.
168. ##### XGB model for toxic category #####
#####
169.
170. train_datav1 = train_dataset
171. train_datav1$toxic = train$toxic
172. train_datav1$toxic = as.factor(train_datav1$toxic)
173. levels(train_datav1$toxic) = make.names(unique(train_datav1$toxic))
174.
175.
176. #splitting main training set into training and validation set.
177.
178. toxicIndex <- createDataPartition(train_datav1$toxic,
179.                                   p = .7,
180.                                   list = FALSE,
181.                                   times = 1)
182.
183. cvtrain <- train_datav1[toxicIndex, ]
184. cvtest <- train_datav1[-toxicIndex, ]
185.
186.
187.
188.
189. formula = toxic ~ .
190.
191. set.seed(123)
192. toxicXGB <- train(
193.   formula,
194.   data = cvtrain,
195.   method = "xgbTree",
196.   trControl = fitControl,
197.   tuneGrid = xgbGrid,
198.   metric = "ROC",

```



```

199.     maximize = FALSE
200.   )
201.   #Visualizing important features for toxic comments
202.   plot_impVar(toxicXGB, "important features for toxic model")
203.
204.
205.   #running prediction on validation/test set
206.   predXGBv1 <-
207.     predict(toxicXGB, cvtest[, -
208.       which(names(cvtest) == "toxic")], type = "prob")
209.     head(predXGBv1)
210.     predXGBv1 <- ifelse(predXGBv1 > 0.5, 'X0', 'X1')
211.     predXGBv1 <- data.frame(predXGBv1)
212.
213.   # confusion matrix
214.   confusionMatrix(predXGBv1$X0, cvtest$toxic)
215.
216.
217.   #prediction for submission test.
218.   predict_toxic = predict(toxicXGB, test_dataset, type = 'prob')
219.   head(predict_toxic)
220.
221.   ##### XGB model for severe toxic category #####
222.   #####
223.   train_datav1 = train_dataset
224.   train_datav1$severe_toxic = train$severe_toxic
225.   train_datav1$severe_toxic = as.factor(train_datav1$severe_toxic)
226.   levels(train_datav1$severe_toxic) = make.names(unique(train_datav1$severe_to
227.     xic))
228.   formula = severe_toxic ~ .
229.   severeIndex <-
230.     createDataPartition(
231.       train_datav1$severe_toxic,
232.       p = .7,
233.       list = FALSE,
234.       times = 1
235.     )
236.   cvtrain <- train_datav1[severeIndex, ]
237.   cvtest <- train_datav1[-severeIndex, ]
238.
239.
240.   set.seed(123)
241.   severe_toxicXGB <- train(
242.     formula,
243.     data = cvtrain,
244.     method = "xgbTree",
245.     trControl = fitControl,
246.     tuneGrid = xgbGrid,
247.     na.action = na.pass,
248.     metric = "ROC",
249.     maximize = FALSE
250.   )
251.   #Visualizing important features for severe toxic comments
252.   plot_impVar(severe_toxicXGB, "important features for severe toxic model")
253.
254.   predXGBv1 <-
255.     predict(severe_toxicXGB, cvtest[, -
256.       which(names(cvtest) == "severe_toxic")], type = "prob")
257.     head(predXGBv1)
258.     predXGBv1 <- ifelse(predXGBv1 > 0.5, 'X0', 'X1')
259.     predXGBv1 <- data.frame(predXGBv1)
260.

```

```

261.      # confusion matrix
262.      confusionMatrix(predXGBv1$X0, cvtest$severe_toxic)
263.
264.
265.
266.      #prediction for submission test.
267.      predict_severe_toxic = predict(severe_toxicXGB, test_dataset, type = 'prob')
268.
269.      head(predict_severe_toxic)
270.
271.      ##### XGB model for obscene category #####
272.
273.      train_datav1 = train_dataset
274.      train_datav1$obscene = train$obscene
275.      train_datav1$obscene = as.factor(train_datav1$obscene)
276.      levels(train_datav1$obscene) = make.names(unique(train_datav1$obscene))
277.      formula = obscene ~ .
278.
279.      obsceneIndex <- createDataPartition(train_datav1$obscene,
280.                                          p = .7,
281.                                          list = FALSE,
282.                                          times = 1)
283.
284.      cvtrain <- train_datav1[obsceneIndex, ]
285.      cvtest  <- train_datav1[-obsceneIndex, ]
286.
287.
288.      set.seed(123)
289.      obsceneXGB <- train(
290.        formula,
291.        data = cvtrain,
292.        method = "xgbTree",
293.        trControl = fitControl,
294.        tuneGrid = xgbGrid,
295.        na.action = na.pass,
296.        metric = "ROC",
297.        maximize = FALSE
298.      )
299.      #Visualizing important features for obscene comments
300.      plot_impVar(obsceneXGB, "important features for obscene model")
301.
302.      predXGBv1 <-
303.        predict(obsceneXGB, cvtest[, -
304.      which(names(cvtest) == "obscene")], type = "prob")
305.      head(predXGBv1)
306.      predXGBv1 <- ifelse(predXGBv1 > 0.5, 'X0', 'X1')
307.      predXGBv1 <- data.frame(predXGBv1)
308.
309.      # confusion matrix
310.      confusionMatrix(predXGBv1$X0, cvtest$obscene)
311.
312.
313.      #prediction for submission test.
314.      predict_obscene = predict(obsceneXGB, test_dataset, type = 'prob')
315.      head(predict_obscene)
316.
317.      ##### XGB model for threat category #####
318.
319.      train_datav1 = train_dataset
320.      train_datav1$threat = train$threat
321.      train_datav1$threat = as.factor(train_datav1$threat)
322.      levels(train_datav1$threat) = make.names(unique(train_datav1$threat))

```

```

323.     formula = threat ~ .
324.
325.     threatIndex <- createDataPartition(train_datav1$threat,
326.                                       p = .7,
327.                                       list = FALSE,
328.                                       times = 1)
329.
330.     cvtrain <- train_datav1[threatIndex, ]
331.     cvtest  <- train_datav1[-threatIndex, ]
332.
333.     set.seed(123)
334.     threatXGB <- train(
335.       formula,
336.       data = cvtrain,
337.       method = "xgbTree",
338.       trControl = fitControl,
339.       tuneGrid = xgbGrid,
340.       na.action = na.pass,
341.       metric = "ROC",
342.       maximize = FALSE
343.     )
344.     #Visualizing important features for threat comments
345.     plot_impVar(threatXGB, "important features for threat model")
346.
347.     predXGBv1 <-
348.       predict(threatXGB, cvtest[, -
which(names(cvtest) == "threat")], type = "prob")
349.     head(predXGBv1)
350.     predXGBv1 <- ifelse(predXGBv1 > 0.5, 'X0', 'X1')
351.     predXGBv1 <- data.frame(predXGBv1)
352.
353.
354.     # confusion matrix
355.     confusionMatrix(predXGBv1$X0, cvtest$threat)
356.
357.     #prediction for submission test.
358.     predict_threat = predict(threatXGB, test_dataset, type = 'prob')
359.     head(predict_threat)
360.
361.     ##### XGB model for insult category #####
362.     #####
363.     train_datav1 = train_dataset
364.     train_datav1$insult = train$insult
365.     train_datav1$insult = as.factor(train_datav1$insult)
366.     levels(train_datav1$insult) = make.names(unique(train_datav1$insult))
367.     formula = insult ~ .
368.
369.
370.     insultIndex <- createDataPartition(train_datav1$insult,
371.                                       p = .7,
372.                                       list = FALSE,
373.                                       times = 1)
374.
375.     cvtrain <- train_datav1[insultIndex, ]
376.     cvtest  <- train_datav1[-insultIndex, ]
377.
378.
379.     set.seed(123)
380.     insultXGB <- train(
381.       formula,
382.       data = cvtrain,
383.       method = "xgbTree",
384.       trControl = fitControl,
385.       tuneGrid = xgbGrid,
386.       na.action = na.pass,

```

```

387.     metric = "ROC",
388.     maximize = FALSE
389. )
390. #Visualizing important features for insult comments
391. plot_impVar(insultXGB, "important features for insult model")
392.
393. predXGBv1 <-
394.   predict(insultXGB, cvtest[, -
which(names(cvtest) == "insult")], type = "prob")
395.   head(predXGBv1)
396.   predXGBv1 <- ifelse(predXGBv1 > 0.5, 'X0', 'X1')
397.   predXGBv1 <- data.frame(predXGBv1)
398.
399.
400.   # confusion matrix
401.   confusionMatrix(predXGBv1$X0, cvtest$insult)
402.
403.
404.
405.   predict_insult = predict(insultXGB, test_dataset, type = 'prob')
406.   head(predict_insult)
407.
408.   ##### XGB model for identity_hate category #####
409.   #####
410.   train_datav1 = train_dataset
411.   train_datav1$identity_hate = train$identity_hate
412.   train_datav1$identity_hate = as.factor(train_datav1$identity_hate)
413.   levels(train_datav1$identity_hate) = make.names(unique(train_datav1$identity
_hate))
414.   formula = identity_hate ~ .
415.
416.
417.   identity_hateIndex <-
418.     createDataPartition(
419.       train_datav1$identity_hate,
420.       p = .7,
421.       list = FALSE,
422.       times = 1
423.     )
424.
425.   cvtrain <- train_datav1[identity_hateIndex, ]
426.   cvtest  <- train_datav1[-identity_hateIndex, ]
427.
428.
429.   set.seed(123)
430.   identity_hateXGB <- train(
431.     formula,
432.     data = cvtrain,
433.     method = "xgbTree",
434.     trControl = fitControl,
435.     tuneGrid = xgbGrid,
436.     na.action = na.pass,
437.     metric = "ROC",
438.     maximize = FALSE
439.   )
440.   #Visualizing important features for identity hate comments
441.   plot_impVar(identity_hateXGB, "important features for identity hate model")
442.
443.   predXGBv1 <-
444.     predict(identity_hateXGB, cvtest[, -
which(names(cvtest) == "identity_hate")], type = "prob")
445.     head(predXGBv1)
446.     predXGBv1 <- ifelse(predXGBv1 > 0.5, 'X0', 'X1')
447.     predXGBv1 <- data.frame(predXGBv1)

```

```
448.
449.     # confusion matrix
450.     confusionMatrix(predXGBv1$X0, cvtest$identity_hate)
451.
452.
453.
454.
455.     predict_identityhate = predict(identity_hateXGB, test_dataset, type = 'prob'
456. )
457.     head(predict_identityhate)
458.
459.     #creating submission for score
460.     submission <-
461.         read.csv(
462.             "sample_submission.csv",
463.             sep = ',',
464.             header = TRUE,
465.             stringsAsFactors = FALSE
466.         )
467.     submission$toxic = predict_toxic$X1
468.     submission$severe_toxic = predict_severe_toxic$X1
469.     submission$obscene = predict_obscene$X1
470.     submission$threat = predict_threat$X1
471.     submission$insult = predict_insult$X1
472.     submission$identity_hate = predict_identityhate$X1
473.
474.
475.     # Write it to file for submission
476.     write.csv(submission, 'toxicCommentClassifV2.csv', row.names = F)
```