Predicting Toxic Comments Classification

Somyanath Mohanty

19 May 2018

Contents

Chapter 1

Introduction

1.1 Problem Statement

Discussing things, you care about can be difficult. The threat of abuse and harassment online means that many people stop expressing themselves and give up on seeking different opinions. Platforms struggle to effectively facilitate conversations, leading many communities to limit or completely shut down user comments. The aim of the project is to build a system which helps the users in finding different types of toxic comments they are interested in finding.

1.2 Data

Our task is to build a model which will be capable of detecting different types of toxicity like threats, obscenity, insults, and identity-based hate better than current models. Given below is a sample of the data set that we are using to predict the different types of toxicity:

Table 1.1: Toxic Comments Sample Data (Columns 1-3)

Id	Toxic Comments	Toxic
0000997932d777bf	Explanation Why the…	0
0002bcb3da6cb337	COCKSUCKER BEFORE..	1
0005c987bdfc9d4b	Hey... what is it..	1

Table 1.1: Toxic Comments Sample Data (Columns 4-8)

Severe Toxic	Obscene	Threat	Insult	Identity Hate
0	0	0	0	0
1	1	0	1	0
0	0	0	0	0

As you can see in the above tables the only independent variable here is the “Toxic Comment”, using which we have to predict the probability of the different toxicity types “Toxic”, “Severe Toxic”, “Obscene”. “Threat”, “Insult”, “Identity Hate”.

Chapter 2

Methodology

2.1 Pre-Processing

Any predictive modelling requires that we look at the data before we start modelling. However, in text mining terms looking at data refers to so much more than just looking. Looking at data refers to tokenizing the sentences, cleaning them by removing punctuation marks, numbers, doing case folding, removing stop words, performing lemmatization, removing short forms, removing deconstructed texts etc.

```
# Remove all special chars, clean text and transform words
all_comments.clean <- all_comments.features %$%
  str_to_lower(comment_text) %>%
  # clear link
  str_replace_all("(f|ht)tp(s?)://\\S+", " ") %>%
  str_replace_all("http\\S+", "") %>%
  str_replace_all("xml\\S+", "") %>%
  # multiple whitespace to one
  str_replace_all("\\s{2}", " ") %>%

  # transform short forms
  str_replace_all("what's", "what is ") %>%
  str_replace_all("\\'s", " is ") %>%
  str_replace_all("\\'ve", " have ") %>%
  str_replace_all("can't", "cannot ") %>%
  str_replace_all("n't", " not ") %>%
  str_replace_all("i'm", "i am ") %>%
  str_replace_all("\\'re", " are ") %>%
  str_replace_all("\\'d", " would ") %>%
  str_replace_all("\\'ll", " will ") %>%
  str_replace_all("\\'scuse", " excuse ") %>%
  str_replace_all("pleas", " please ") %>%
  str_replace_all("sourc", " source ") %>%
  str_replace_all("peopl", " people ") %>%
  str_replace_all("remov", " remove ") %>%

  # multiple whitespace to one
  str_replace_all("\\s{2}", " ") %>%

  # transform deconstructed abusive text
  str_replace_all("(a|e)w+\\b", "") %>%
  str_replace_all("(y)a+\\b", "") %>%
  str_replace_all("(w)w+\\b", "") %>%
  str_replace_all("((a+)|(h+))(a+)((h+)?)\\b", "") %>%
  str_replace_all("((lol)(o?))+\\b", "") %>%
  str_replace_all("n ig ger", " nigger ") %>%
```

```
  str_replace_all("s hit", " shit ") %>%
  str_replace_all("g ay", " gay ") %>%
  str_replace_all("f ag got", " faggot ") %>%
  str_replace_all("c ock", " cock ") %>%
  str_replace_all("cu nt", " cunt ") %>%
  str_replace_all("idi ot", " idiot ") %>%
  str_replace_all("f u c k", " fuck ") %>%
  str_replace_all("fu ck", " fuck ") %>%
  str_replace_all("f u ck", " fuck ") %>%
  str_replace_all("c u n t", " cunt ") %>%
  str_replace_all("s u c k", " suck ") %>%
  str_replace_all("c o c k", " cock ") %>%
  str_replace_all("g a y", " gay ") %>%
  str_replace_all("ga y", " gay ") %>%
  str_replace_all("i d i o t", " idiot ") %>%
  str_replace_all("cocksu cking", "cock sucking") %>%
  str_replace_all("du mbfu ck", "dumbfuck") %>%
  str_replace_all("cu nt", "cunt") %>%
  str_replace_all("(?<=\\b(fu|su|di|co|li))\\s(?=(ck)\\b)", "") %>%
  str_replace_all("(?<=\\w(ck))\\s(?=(ing)\\b)", "") %>%
  str_replace_all("(?<=\\b\\w)\\s(?=\\w\\b)", "") %>%
  str_replace_all("((lol)(o?))+", "") %>%
  str_replace_all("(?<=\\b(fu|su|di|co|li))\\s(?=(ck)\\b)", "") %>%
  str_replace_all("(?<=\\w(uc))\\s(?=(ing)\\b)", "") %>%
  str_replace_all("(?<=\\b(fu|su|di|co|li))\\s(?=(ck)\\w)", "") %>%
  str_replace_all("(?<=\\b(fu|su|di|co|li))\\s(?=(k)\\w)", "c") %>%

  # clean nicknames
  str_replace_all("@\\w+", " ") %>%
  # clean digit
  str_replace_all("[[:digit:]]", " ") %>%
  # remove line breaks
  str_replace_all("\n", " ") %>%
  # remove graphics
  str_replace_all("[^[:graph:]]", " ") %>%
  # remove punctuation (if remain...)
  str_replace_all("[[:punct:]]", " ") %>%
  # remove alphanumeric text
  str_replace_all("[^[:alnum:]]", " ") %>%
  # remove single char
  str_replace_all("\\W*\\b\\w\\b\\W*", " ")  %>%
  # remove words with len < 2
  str_replace_all("\\b\\w{1,2}\\b", " ")  %>%
  # stripping multiple whitespace to one
  str_replace_all("\\s{2}", " ") %>%
  str_replace_all("\\s+", " ") %>%

  # creating an iterator
  itoken(tokenizer = tokenize_word_stems)
```

Here, we are cleaning the comments and then in the last we have created an iterator to parse over the input objects (words in our case) to create vocabularies, or DTM matrices.

2.2 Data Analysis

Text Mining is a lot different from basic data mining because in this there are no concept of outlier analysis or feature scaling but here we can check the correlation among different labels which will help us to understand the dependence of the different toxicity levels on each other. For that we can opt for correlational analysis.

```
library(corrplot)
# Creating a character vector of target variables
target = colnames(train)[3:8]

# Creating a dataset with only the target variables
target_data = train[target]

# Creating the correlation analysis plot
cor_plot = cor(target_data)
corrplot(cor_plot, method = "color", addgrid.col = "darkgray",
        outline = TRUE, addCoef.col = "white", number.digits = 2)
```

The correlation plot clearly shows that there is a correlation between “insult” & “obscene”, “toxic” & “obscene” and “toxic” & “insult”.

2.3 Modelling

2.3.1 Model Selection

Here we have to predict the probabilities of the different types of toxicity for a particular comment. So, this is basically a classification problem where we have to classify the comments according to the probability of a particular type of toxicity.

Most text mining and NLP modelling use bag of words or bag of n-grams methods. Despite their simplicity, these models usually demonstrate good performance on text categorization and classification tasks.

Let's briefly review some of the steps in a typical text analysis pipeline:

1. The researcher usually begins by constructing a document-term matrix (DTM) from input documents. In other words, the first step is to vectorize text by creating a map from words or n-grams to a vector space.
2. The researcher fits a model to that DTM. These models might include text classification, topic modelling, similarity search, etc. Fitting the model will include tuning and validating the model.
3. Finally, the researcher applies the model to new data.

Here we'll start our model building from the simplest to more complex model. So, for our classification problem the first model will be logistic regression, for which we'll be using "*glmnet*" package in R.

Let's start with the first step of the text mining pipeline i.e. create a document term matrix(DTM) using the input toxic comments. Here, each toxic comment corresponding to each id will be treated as one complete document.

2.3.2 Vectorization

To represent documents in vector space, we first have to create mappings from terms to term IDS. We call them terms instead of words because they can be arbitrary n-grams not just single words. We represent a set of documents as a sparse matrix, where each row corresponds to a document and each column corresponds to a term. This can be done in 2 ways: using the vocabulary itself or by feature hashing.
Here we'll use the vocabulary-based vectorization

```
# Creating a vectorizer to create a DTM.
# Vectorizer
t1 <- Sys.time()
vectorizer.dict <- create_vocabulary(all_comments.clean, ngram = c(1L, 2L) stopwords = stop_words_chr) %>%
  prune_vocabulary(term_count_min = 4, doc_proportion_max = 0.3, vocab_term_max = 10000)

print(vectorizer.dict)
```

```
vectorizer <- vectorizer.dict %>% vocab_vectorizer()
```

Here we collect unique terms from all documents and mark each of them with a unique ID using the "*create_vocabulary()*" function. Also, we can significantly reduce the training time and accuracy of our model by pruning the vocabulary. For example, we can find words "a", "the", "in", "I", "you", "on", etc in almost all documents, but they do not provide much useful information. Here we will remove pre-defined stopwords, very common and very unusual terms. We use an iterator to create the vocabulary.

Now that we have a vocabulary, we can construct a document-term matrix.

```
all_comments.dtm <- create_dtm(all_comments.clean, vectorizer) %>% normalize(norm = "l2")
print(difftime(Sys.time(), t1, units = 'sec'))
```

```
Number of docs: 312735
1678 stopwords: i, me, my, myself, we, our ...
ngram_min = 1; ngram_max = 2
Vocabulary:
                term term_count doc_count
    1:         pleas      53232     39783
    2:          fuck      49460     12424
    3:         sourc      47539     27924
    4:         delet      45402     21832
    5:         peopl      35615     24917
   ---
 9996:        bengal         93        49
 9997: histor_figur         93        87
 9998:       archaic         93        77
 9999:    post_delet         93        89
10000: exist_becaus         93        92
Time difference of 103.6366 secs
```

2.3.3 Logistic Regression

Now we are ready to fit our first model. Here we will use the "*glmnet*" package to fit a logistic regression model with an L1 penalty and 10-fold cross-validation.

```
# Training glmnet & predict toxicity
for (target in c("toxic", "severe_toxic", "obscene", "threat", "insult", "identity_hate")) {
  cat("\nTrain -->", target, "...")
  y <- factor(train[[target]])
  t1 <- Sys.time()
  glm.model <- cv.glmnet(train.matrix, y, alpha = 1, family = "binomial", type.measure = "auc",
```

```
                parallel = T, standardize = T, nfolds = 10, nlambda = 50)
  print(difftime(Sys.time(), t1, units = 'sec'))
  cat(" AUC:", max(glm.model$cvm))
  subm[[target]] <- predict(glm.model, test.matrix, type = "response", s = "lambda.min")
}
```

```
Train --> toxic ... AUC: 0.9668512Time difference of 107.3474 secs

Train --> severe_toxic ... AUC: 0.9870708Time difference of 128.2084 secs

Train --> obscene ... AUC: 0.9814219Time difference of 106.8772 secs

Train --> threat ... AUC: 0.9795836Time difference of 130.1804 secs

Train --> insult ... AUC: 0.9761061Time difference of 110.848 secs

Train --> identity_hate ... AUC: 0.9748461Time difference of 126.8065 secs
```

2.3.4 Extreme Gradient Boosting (XGBoost)

Now, for the more complex model here we will use *XGBoost* which is a scalable and accurate implementation of gradient boosting machines and it has proven to push the limits of computing power for boosted trees algorithms.
This is an ensemble method that seeks to create a strong classifier (model) based on "weak" classifiers. In this context, weak and strong refer to a measure of how correlated the learners to the actual target variable are. By adding models on top of each other iteratively, the errors of the previous model are corrected by the next predictor, until the training data is accurately predicted or reproduced by the model.
As our correlation analysis did proved that some of the labels are correlated to each other, so we can use *xgboost* as our next model.

```
# Applying xgboost
library(xgboost)
subm_xgb <- data.frame(id = test$id)
for (target in c("toxic", "severe_toxic", "obscene", "threat", "insult", "identity_hate")) {
   y <- train[[target]]
   t1 <- Sys.time()
   print(target)
   xgb.model <- xgboost(data = train.matrix, label = y,
                        booster = "gblinear", eval_metric = "auc",
                        objective = "binary:logistic", verbose = T,
                        nrounds = 4, alpha = 0.5)
   print(difftime(Sys.time(), t1, units = 'sec'))
   subm_xgb[[target]] <- predict(xgb.model, test.matrix)
}
```

```
Train --> toxic ...[1]  train-auc:0.975673
[2]     train-auc:0.980670
[3]     train-auc:0.982105
[4]     train-auc:0.982336
Time difference of 5.863597 secs

Train --> severe_toxic ...[1]   train-auc:0.990340
[2]     train-auc:0.991139
[3]     train-auc:0.991652
[4]     train-auc:0.992033
Time difference of 4.488909 secs

Train --> obscene ...[1]        train-auc:0.989472
[2]     train-auc:0.992251
[3]     train-auc:0.993111
[4]     train-auc:0.993278
Time difference of 4.726672 secs

Train --> threat ...[1] train-auc:0.985982
[2]     train-auc:0.986998
[3]     train-auc:0.988321
[4]     train-auc:0.989410
Time difference of 3.55719 secs

Train --> insult ...[1] train-auc:0.983736
[2]     train-auc:0.986420
[3]     train-auc:0.987377
[4]     train-auc:0.987385
Time difference of 4.228611 secs

Train --> identity_hate ...[1]  train-auc:0.983221
[2]     train-auc:0.985156
[3]     train-auc:0.986906
[4]     train-auc:0.988180
Time difference of 3.384799 secs
```

Chapter 3

Conclusion

3.1 Model Evaluation

Now that we have two models to predict the target variables, we need to decide which one to choose. There are several criteria that exist for evaluating and comparing models. We can compare the models using any of the following criteria:

1. Predictive Performance
2. Interpretability
3. Computational Efficiency

In Our case of Toxic Comment classification, the former two, Predictive Performance and Interpretability, do not hold much significance as we cannot compare the models as we don't have any real values of the target variables.

Therefore, we will use Computational Efficiency to compare the two models.

3.2 Computational Time

The only thing we check here is the time taken by the model to train and make the predictions for the test data. We can easily see that the *xgboost* model is nearly 50 times faster than the logistic regression method.

So, our choice of model here is *xgboost*.

Appendix – Complete R code

```
library(ggplot2)
library(data.table)
library(dplyr)
library(stringr)
library(ngram)
library(tokenizers)
library(tidyverse)
library(magrittr)
library(text2vec)
library(glmnet)
library(tm)
library(xgboost)
library(doParallel)
registerDoParallel(4)

train <- fread("../input/train.csv", key=c("id"))
test <- fread("../input/test.csv", key=c("id"))
stopwords.en <- fread("../input/stopwords-en.txt")
stowwords.custom <- c("put", "far", "bit", "well", "article", "articles", "edit", "edits", "page", "pages",
                      "talk", "page", "editor", "ax", "edu", "subject", "lines", "like", "likes", "line",
                      "uh", "oh", "also", "get", "just", "hi", "hello", "ok", "editing", "edited",
                      "dont", "use", "need", "take", "wikipedia", "give", "say",
                      "look", "one", "make", "come", "see", "said", "now", "wiki", "know", "talk", "read",
"hey", "time", "still", "user", "day", "want", "tell", "edit", "even", "ain't", "wow", "image", "jpg",
"copyright", "sentence", "wikiproject", "background color", "align", "px", "pixel",
                      "org", "com", "en", "ip", "ip address", "http", "www", "html", "htm",
                      "wikimedia", "https", "httpimg", "url", "urls", "utc", "uhm","username","wikipedia",
                      "what", "which", "who", "whom", "this", "that", "these", "those",
                      "was", "be", "been", "being", "have", "has", "had", "having", "do", "does", "did",
                      "doing", "would", "should", "could", "ought", "isn't", "aren't", "wasn't", "weren't",
"hasn't", "haven't", "hadn't", "doesn't", "don't", "didn't", "won't", "wouldn't", "shan't", "shouldn't",
                      "can't", "cannot", "couldn't", "mustn't", "let's", "that's", "who's", "what's", "here's",
                      "there's", "when's", "where's", "why's", "how's", "a", "an", "the", "and", "but", "if",
                      "or", "because", "as", "until", "while", "of", "at", "by", "for", "with", "about", "against",
                      "between", "into", "through", "during", "before", "after", "above", "below", "to", "from",
                      "up", "down", "in", "out", "on", "off", "over", "under", "again", "further", "then",
"once", "here", "there", "when", "where", "why", "how", "all", "any", "both", "each", "few", "more",
                      "most", "other", "some", "such", "no", "nor", "not", "only", "own", "same", "so", "than",
                      "too", "very","articl","ani")

train <- train %>% mutate(filter="train")
test <- test %>% mutate(filter="test")

all_comments <- train %>% bind_rows(test)

# Create some new features relative to use of punctuation, emoticons, ...
all_comments.features <- all_comments %>%

  select(id, comment_text) %>%
  mutate(length = str_length(comment_text),
```

```
        use_cap = str_count(comment_text, "[A-Z]"),
        cap_len = use_cap / length,
        use_lower = str_count(comment_text, "[a-z]"),
        low_len = use_lower / length,
        cap_rate = ifelse(is.null(use_cap / use_lower), 0, use_cap / use_lower),
        cap_odds = ifelse(is.null(cap_len / low_len), 0, cap_len / low_len),
        use_exl = str_count(comment_text, fixed("!")),
        use_space = str_count(comment_text, fixed(" ")),
        use_double_space = str_count(comment_text, fixed("  ")),
        use_quest = str_count(comment_text, fixed("?")),
        use_punt = str_count(comment_text, "[[:punct:]]"),
        use_digit = str_count(comment_text, "[[:digit:]]"),
        digit_len = use_digit / length,
        use_break = str_count(comment_text, fixed("\n")),
        use_word = str_count(comment_text, "\\w+"),
        use_symbol = str_count(comment_text, "&|@|#|\\$|%|\\*|\\^"),
        use_char = str_count(comment_text, "\\W*\\b\\w\\b\\W*"),
        use_i = str_count(comment_text, "(\\bI\\b)|(\\bi\\b)"),
        i_len = use_i / length,
        char_len = use_char / length,
        symbol_len = use_symbol / length,
        use_emotj = str_count(comment_text, "((?::|;|=)(?:-)?(?:\\)|D|P))"),
        cap_emo = use_emotj / length) %>%
  select(-id) %T>%
  glimpse()

# Overview of the features we have added
head(all_comments.features)

nrow(all_comments.features)

# Remove all special chars, clean text and transform words
all_comments.clean <- all_comments.features %$%
  str_to_lower(comment_text) %>%
  # clear link
  str_replace_all("(f|ht)tp(s?)://\\S+", " ") %>%
  str_replace_all("http\\S+", "") %>%
  str_replace_all("xml\\S+", "") %>%
  # multiple whitespace to one
  str_replace_all("\\s{2}", " ") %>%

  # transform short forms
  str_replace_all("what's", "what is ") %>%
  str_replace_all("\\'s", " is ") %>%
  str_replace_all("\\'ve", " have ") %>%
  str_replace_all("can't", "cannot ") %>%
  str_replace_all("n't", " not ") %>%
  str_replace_all("i'm", "i am ") %>%
  str_replace_all("\\'re", " are ") %>%
  str_replace_all("\\'d", " would ") %>%
  str_replace_all("\\'ll", " will ") %>%
  str_replace_all("\\'scuse", " excuse ") %>%
  str_replace_all("pleas", " please ") %>%
  str_replace_all("sourc", " source ") %>%
  str_replace_all("peopl", " people ") %>%
```

```
str_replace_all("remov", " remove ") %>%

# multiple whitespace to one
str_replace_all("\\s{2}", " ") %>%

# transform deconstructed abusive text
str_replace_all("(a|e)w+\\b", "") %>%
str_replace_all("(y)a+\\b", "") %>%
str_replace_all("(w)w+\\b", "") %>%
str_replace_all("((a+)|(h+))(a+)((h+)?)\\b", "") %>%
str_replace_all("((lol)(o?))+\\b", "") %>%
str_replace_all("n ig ger", " nigger ") %>%
str_replace_all("s hit", " shit ") %>%
str_replace_all("g ay", " gay ") %>%
str_replace_all("f ag got", " faggot ") %>%
str_replace_all("c ock", " cock ") %>%
str_replace_all("cu nt", " cunt ") %>%
str_replace_all("idi ot", " idiot ") %>%
str_replace_all("f u c k", " fuck ") %>%
str_replace_all("fu ck", " fuck ") %>%
str_replace_all("f u ck", " fuck ") %>%
str_replace_all("c u n t", " cunt ") %>%
str_replace_all("s u c k", " suck ") %>%
str_replace_all("c o c k", " cock ") %>%
str_replace_all("g a y", " gay ") %>%
str_replace_all("ga y", " gay ") %>%
str_replace_all("i d i o t", " idiot ") %>%
str_replace_all("cocksu cking", "cock sucking") %>%
str_replace_all("du mbfu ck", "dumbfuck") %>%
str_replace_all("cu nt", "cunt") %>%
str_replace_all("(?<=\\b(fu|su|di|co|li))\\s(?=(ck)\\b)", "") %>%
str_replace_all("(?<=\\w(ck))\\s(?=(ing)\\b)", "") %>%
str_replace_all("(?<=\\b\\w)\\s(?=\\w\\b)", "") %>%
str_replace_all("((lol)(o?))+", "") %>%
str_replace_all("(?<=\\b(fu|su|di|co|li))\\s(?=(ck)\\b)", "") %>%
str_replace_all("(?<=\\w(uc))\\s(?=(ing)\\b)", "") %>%
str_replace_all("(?<=\\b(fu|su|di|co|li))\\s(?=(ck)\\w)", "") %>%
str_replace_all("(?<=\\b(fu|su|di|co|li))\\s(?=(k)\\w)", "c") %>%

# clean nicknames
str_replace_all("@\\w+", " ") %>%
# clean digit
str_replace_all("[[:digit:]]", " ") %>%
# remove line breaks
str_replace_all("\n", " ") %>%
# remove graphics
str_replace_all("[^[:graph:]]", " ") %>%
# remove punctuation (if remain...)
str_replace_all("[[:punct:]]", " ") %>%
# remove alphanumeric
str_replace_all("[^[:alnum:]]", " ") %>%

# remove single char
str_replace_all("\\W*\\b\\w\\b\\W*", " ")  %>%
# remove words with len < 2
```

```
  str_replace_all("\\b\\w{1,2}\\b", " ")  %>%

  # multiple whitespace to one
  str_replace_all("\\s{2}", " ") %>%
  str_replace_all("\\s+", " ") %>%

  itoken(tokenizer = tokenize_word_stems)


# Calculate stopwords and rare words
stop_words_chr <- c(stopwords("en")) %>% c(stopwords.en$C1) %>% c(stowwords.custom)


library(corrplot)
# Creating a character vector of target variables
target = colnames(train)[3:8]

# Creating a dataset with only the target variables
target_data = train[target]

# Creating the correlation analysis plot
cor_plot = cor(target_data)
corrplot(cor_plot, method = "color", addgrid.col = "darkgray",
         outline = TRUE, addCoef.col = "white", number.digits = 2)


#Creating a vectorizer and then Pruning the Vocabulary to create a DTM.
# Vectorizer
t1 <- Sys.time()
vectorizer.dict <- create_vocabulary(all_comments.clean, ngram = c(1L, 2L),

stopwords = stop_words_chr) %>%
  prune_vocabulary(term_count_min = 4, doc_proportion_max = 0.3,

vocab_term_max = 10000)

print(vectorizer.dict)

vectorizer <- vectorizer.dict %>% vocab_vectorizer()
all_comments.dtm <- create_dtm(all_comments.clean, vectorizer) %>%

normalize(norm = "l2")
print(difftime(Sys.time(), t1, units = 'sec'))

# Number of rows (number of documents) & columns (number of unique terms) in the DTM
dim(all_comments.dtm)


# Preparing data for glmnet
all_comments.matrix <- all_comments.features %>%
  select(-comment_text) %>%
  sparse.model.matrix(~ . - 1, .) %>%
  cbind(all_comments.dtm)
```

```
# Prepare train and test set
names(all_comments)
train.set <- all_comments[, "filter"] == "train"
test.set <- all_comments[, "filter"] == "test"
train.matrix <- all_comments.matrix[train.set,]
test.matrix <- all_comments.matrix[test.set,]
subm <- data.frame(id = test$id)

# Training glmnet & predict toxicity
for (target in c("toxic", "severe_toxic", "obscene", "threat", "insult",

"identity_hate")) {
  cat("\nTrain -->", target, "...")
  y <- factor(train[[target]])
  t1 <- Sys.time()
  glm.model <- cv.glmnet(train.matrix, y, alpha = 0, family = "binomial",

type.measure = "auc",
                  parallel = T, standardize = T, nfolds = 10, nlambda =

50)
  cat(" AUC:", max(glm.model$cvm))
  subm[[target]] <- predict(glm.model, test.matrix, type = "response", s =

"lambda.min")
  print(difftime(Sys.time(), t1, units = 'sec'))
}

# Training xgbmodel and predict toxicity
subm_xgb <- data.frame(id = test$id)
for (target in c("toxic", "severe_toxic", "obscene", "threat", "insult",

"identity_hate")) {
  cat("\nTrain -->", target, "...")
  y <- train[[target]]
  t1 <- Sys.time()
  xgboost.model <- xgboost(data = train.matrix, label = y,
                           booster = "gblinear", eval_metric = "auc",
                           objective = "binary:logistic", verbose = T,
                           nfold = 5, nrounds = 4, alpha = 0.5)
  subm_xgb[[target]] <- predict(xgboost.model, test.matrix)
  print(difftime(Sys.time(), t1, units = 'sec'))
}
```