Stats 412 Final Project: Predicting Recommendations of New York Times Article Comments

Lucy Baden

6/6/2018

Introduction

In this project, we developed models to predict the number of recommendations, or upvotes, received by comments on articles in the New York Times. The training data consisted of two datasets: train_comments.csv, which contained 665396 comments, the number of recommendations received by each comment, and 32 additional features concerning the comments, and train_articles.csv, which contained 15 features for each of the 3445 articles the comments were taken from. We performed exploratory data analysis to examine these features and inform our decision to include them in our models and/or create new features. We then developed a number of different models, with random forest and gradient-boosted machines proving to be the most successful. Prediction accuracy was judged using both the root mean squared error (RMSE) and mean absolute error (MAE), which was the measurement used to judge the Kaggle competition. Model predictive accuracy was low in general, with randomly generated recommendations giving an MAE only about 0.8 worse than my best model on the test data.

Exploratory Data Analysis

Comment Data

We can see a summary of the comment data below:

```
‘data.frame’: 665396 obs. of 34 variables:
$ approveDate : int 1519852022 1518469135 1518385379 1517986719 1517945037 1517945011 1517945003
1517804923 1517525636 1517525624 . . .
$ articleID : Factor w/ 3570 levels “5a4980f57c459f246b63d649”,..: 1030 1030 1030 1030 1030 1030 1030 1030
1030 1030 . . .
$ articleWordCount : int 1322 1322 1322 1322 1322 1322 1322 1322 1322 1322 . . .
$ commentBody : Factor w/ 663261 levels “Congestion Pricing or Eliminate Double ParkingThere is a
cheaper and more effective remedy for New Yor”| ___truncated,..: 243535 247532 208698 245994 631974
393011 373869 86339 645636 131080 . . .
$ commentID : num 26156416 25930059 25912292 25864174 25855991 . . .
$ commentSequence : num 26156416 25930059 25912292 25864174 25855991 . . .
$ commentTitle : Factor w/ 2 levels “”,“n/a”: 1 1 1 1 1 1 1 1 1 1 . . .
$ commentType : Factor w/ 3 levels “comment”,“reporterReply”,..: 1 1 1 1 1 1 1 1 1 1 . . .
$ createDate : int 1519849555 1518458548 1518304930 1517980671 1517938058 1517916102 1517918817
1517762060 1517428315 1517505985 . . .
$ depth : int 1 1 1 1 1 1 1 1 1 1 . . .
$ editorsSelection : Factor w/ 4 levels “0”,“1”,“False”,..: 3 3 3 3 3 3 3 3 3 3 . . .
$ inReplyTo : int 0 0 0 0 0 0 0 0 0 0 . . .
$ newDesk : Factor w/ 42 levels “Arts&Leisure”,..: 35 35 35 35 35 35 35 35 35 35 . . .
$ parentID : num 0 0 0 0 0 0 0 0 0 0 . . .
$ parentUserDisplayName: Factor w/ 26258 levels “ W”,“–”,“-APR”,..: NA NA NA NA NA NA NA NA NA
NA . . .
$ permID : Factor w/ 579639 levels “25389168”,“25389460”,..: 416118 287473 276711 248274 243529 240115
```

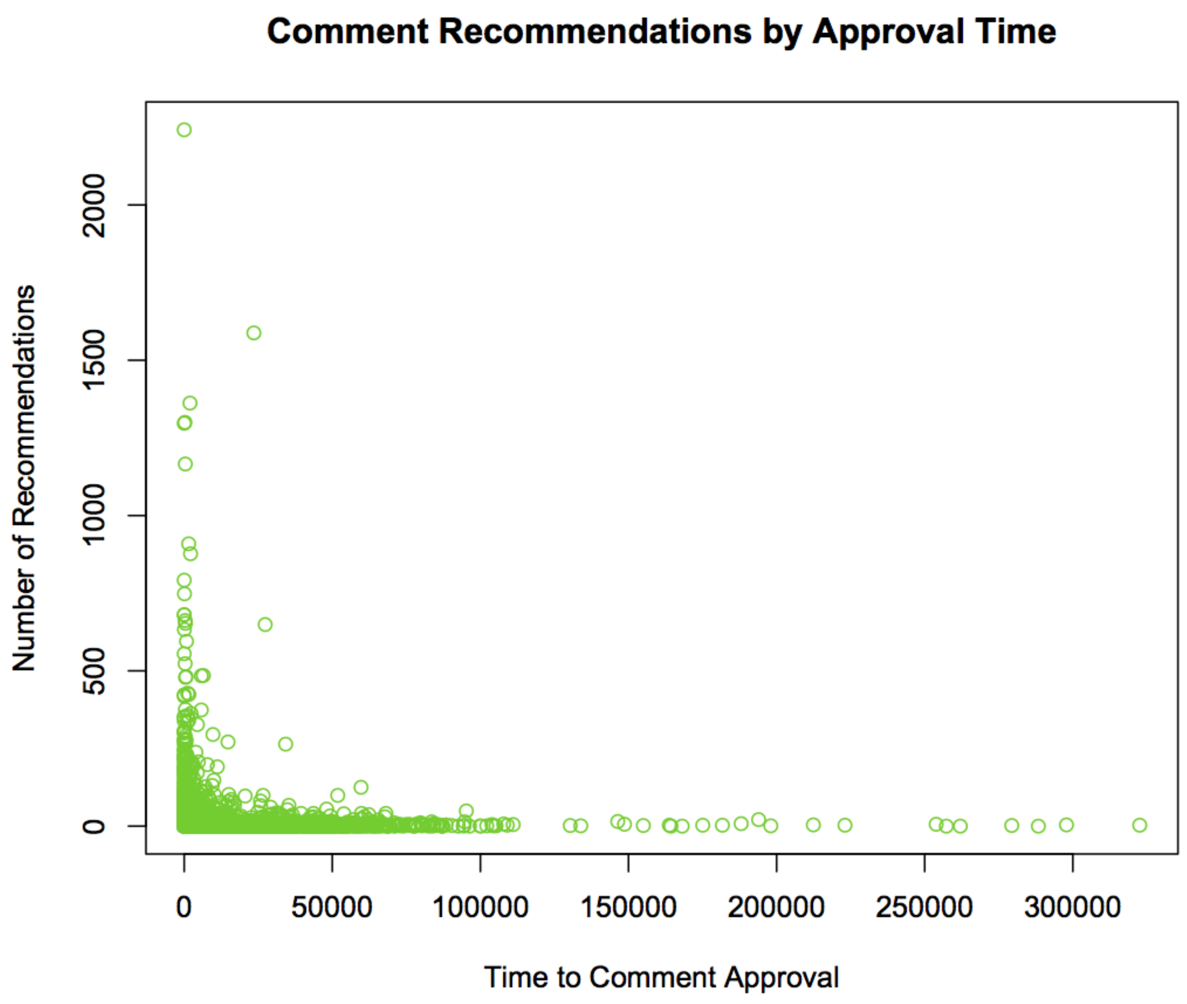
240284 227241 201515 205634 . . .
$ picURL : Factor w/ 5925 levels “http://graphics8.nytimes.com/images/apps/timespeople/none.png”,..: 452 452 452 452 452 452 452 452 452 452 452 . . .
$ printPage : int 5 5 5 5 5 5 5 5 5 5 . . .
$ recommendations : int 1 0 1 0 12 1 0 2 6 4 . . .
$ recommendedFlag : logi NA NA NA NA NA NA . . .
$ replyCount : int 0 0 1 0 0 0 0 0 0 0 . . .
$ reportAbuseFlag : logi NA NA NA NA NA NA . . .
$ sectionName : Factor w/ 49 levels “Africa”,“Americas”,..: 46 46 46 46 46 46 46 46 46 46 . . .
$ sharing : int 0 0 1 0 0 0 0 0 0 0 . . .
$ status : Factor w/ 1 level “approved”: 1 1 1 1 1 1 1 1 1 1 . . .
$ timespeople : int 0 1 1 0 1 1 1 1 1 1 . . .
$ trusted : int 0 0 0 0 0 0 0 0 0 0 . . .
$ typeOfMaterial : Factor w/ 14 levels “An Appraisal”,..: 9 9 9 9 9 9 9 9 9 9 . . .
$ updateDate : int 1519852022 1518469135 1518385379 1517986719 1517945037 1517945011 1517945003 1517804923 1517525636 1517525624 . . .
$ userDisplayName : Factor w/ 83239 levels “_DB“,”*W*“,”- roger-“,..: 23711 24784 50715 19404 22344 66928 81525 34996 65111 69485 . . .
$ userID : num 83288014 53167641 44043675 84748907 25854823 . . .
$ userLocation : Factor w/ 25829 levels”-“,”–“,”———-“,..: 25265 19916 4526 15331 4995 2986 24860 2791 24130 19916 . . .
$ userTitle : Factor w/ 14 levels”Columnist, New York Today“,..: NA NA NA NA NA NA NA NA NA NA . . .
$ userURL : logi NA NA NA NA NA NA . . .

We examined each feature and investigated the number of unique values and NA values they contained. Based on this, we removed five variables from the dataset: commentTitle, which contained only the values $< br/ >$, n/a, and NA; recommendedFlag, reportAbuseFlag, and userURL, which were NA for all comments; and status, which was approved for all comments.

We then explored the remainder of the features and their relationship with recommendations in more depth; notable features are discussed below.

createDate and approveDate

The features createDate and approveDate are the dates when the comments were created and approved respectively. Although the dates are in integer format, they can be converted to dates using January 1 1970 as the reference point. We also created a new feature, approvalTime, which was calculated as the difference betweeen these dates. The plot below shows some signs of a trend where comments that are approved quickly tend to get more recommendations in a random sample of 5000 comments:

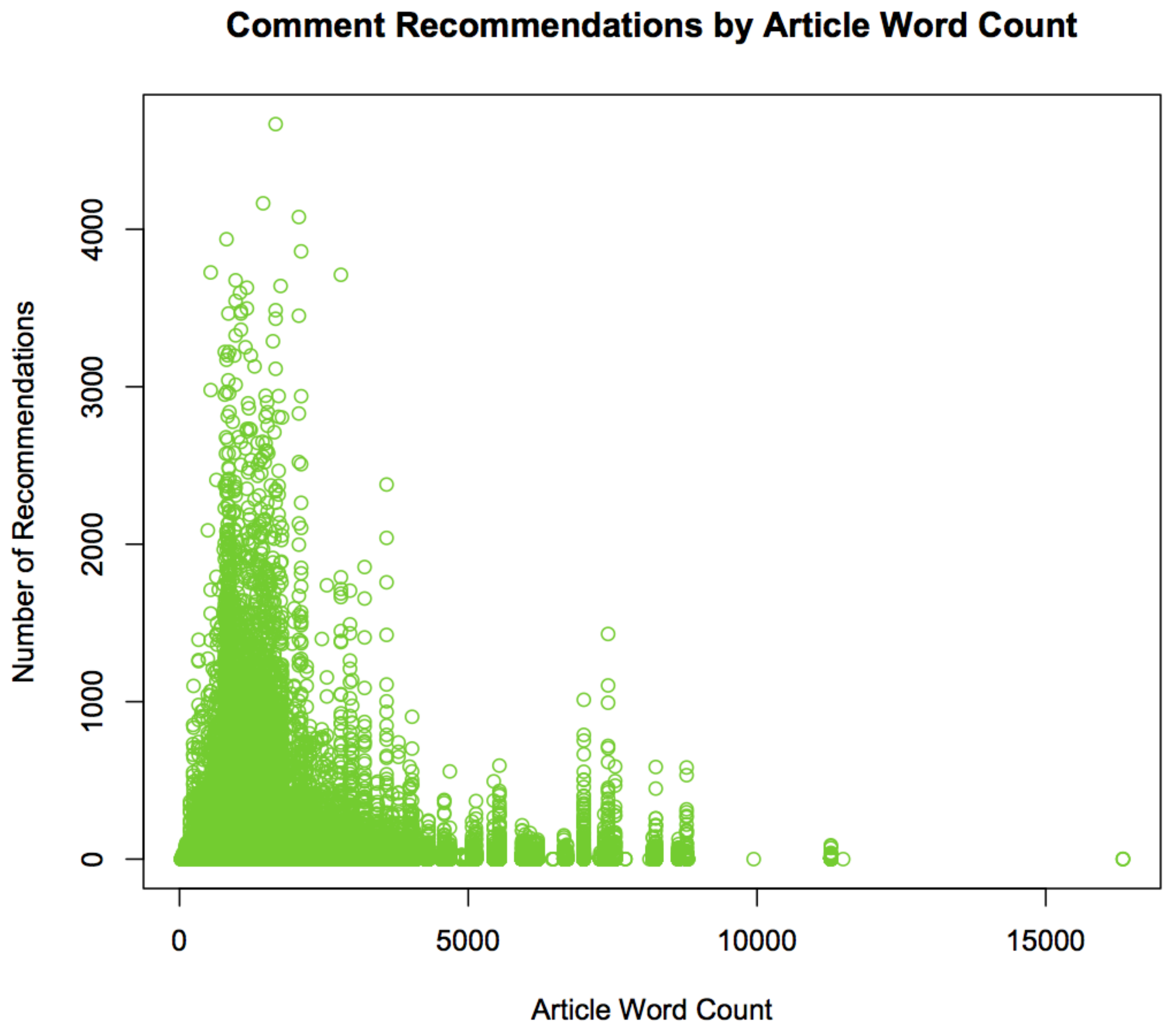


articleID

The train_comments dataset includes the articleID for each comment as well; there are 3570 unique article IDs. This value was used to join this dataset with the train_article dataset later, but is little use for predictive modeling given the number of levels. In contrast, there are only 3445 articles in the train_articles dataset. The comments without a matching article were given NA values for the various article features when the datasets were merged.

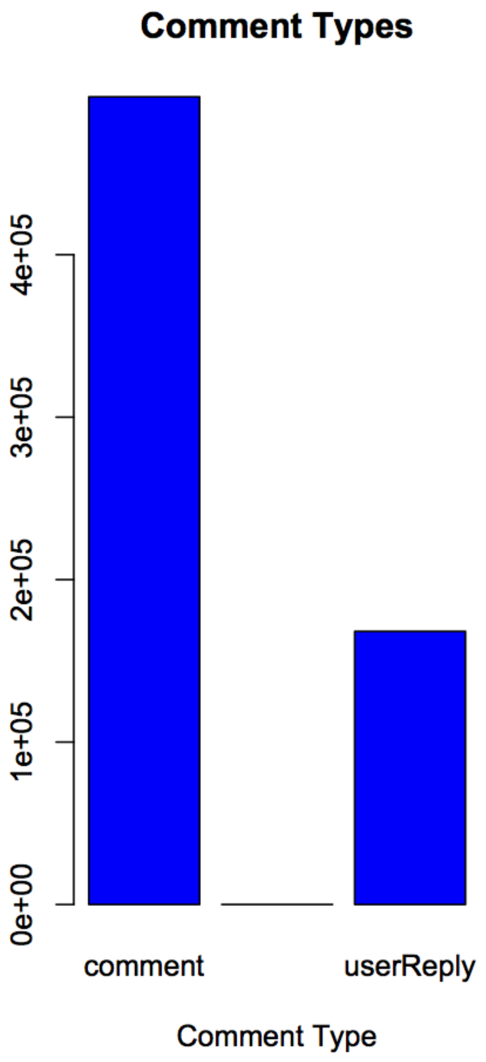
articleWordCount

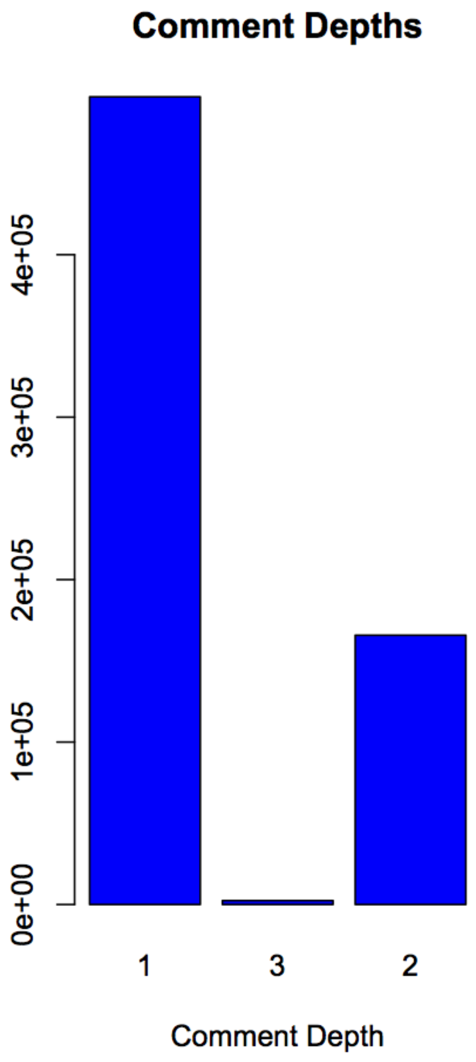
The articleWordCount feature lists the length of each article. As we can see in the plot below, there is some relation between the article's word count and the number of recommendations given to its comments, with shorter articles more frequently having comments with more recommendations.



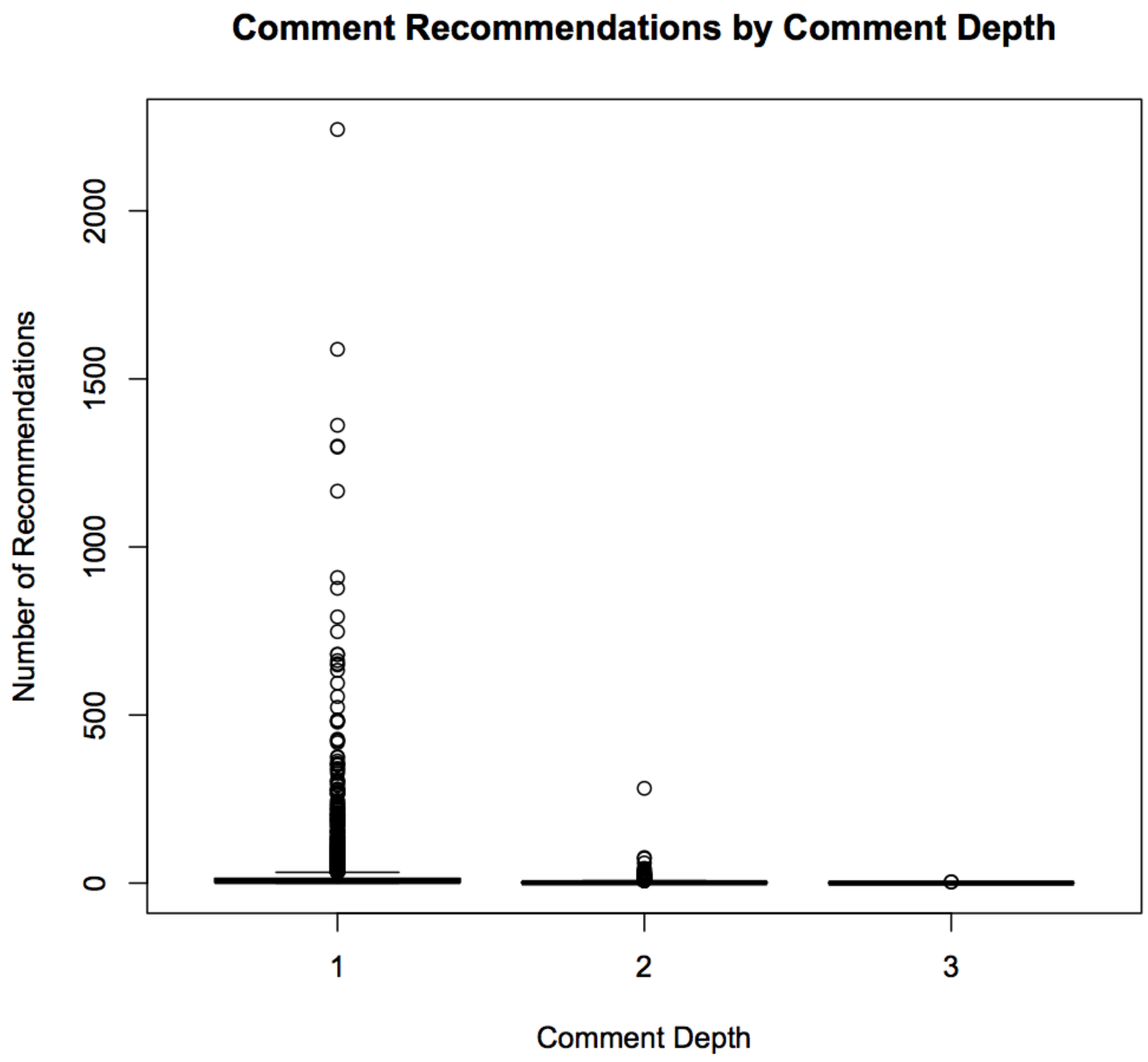
commentType and depth

Depth refers to the comment's level, where a level 2 comment is a reply to a level 1 comment, and a level 3 comment is a reply to a level 2 comment. Similarly, commentType categorizes each observation as a comment, user reply, and reporter reply. We can see in the plot below that they are very similar.



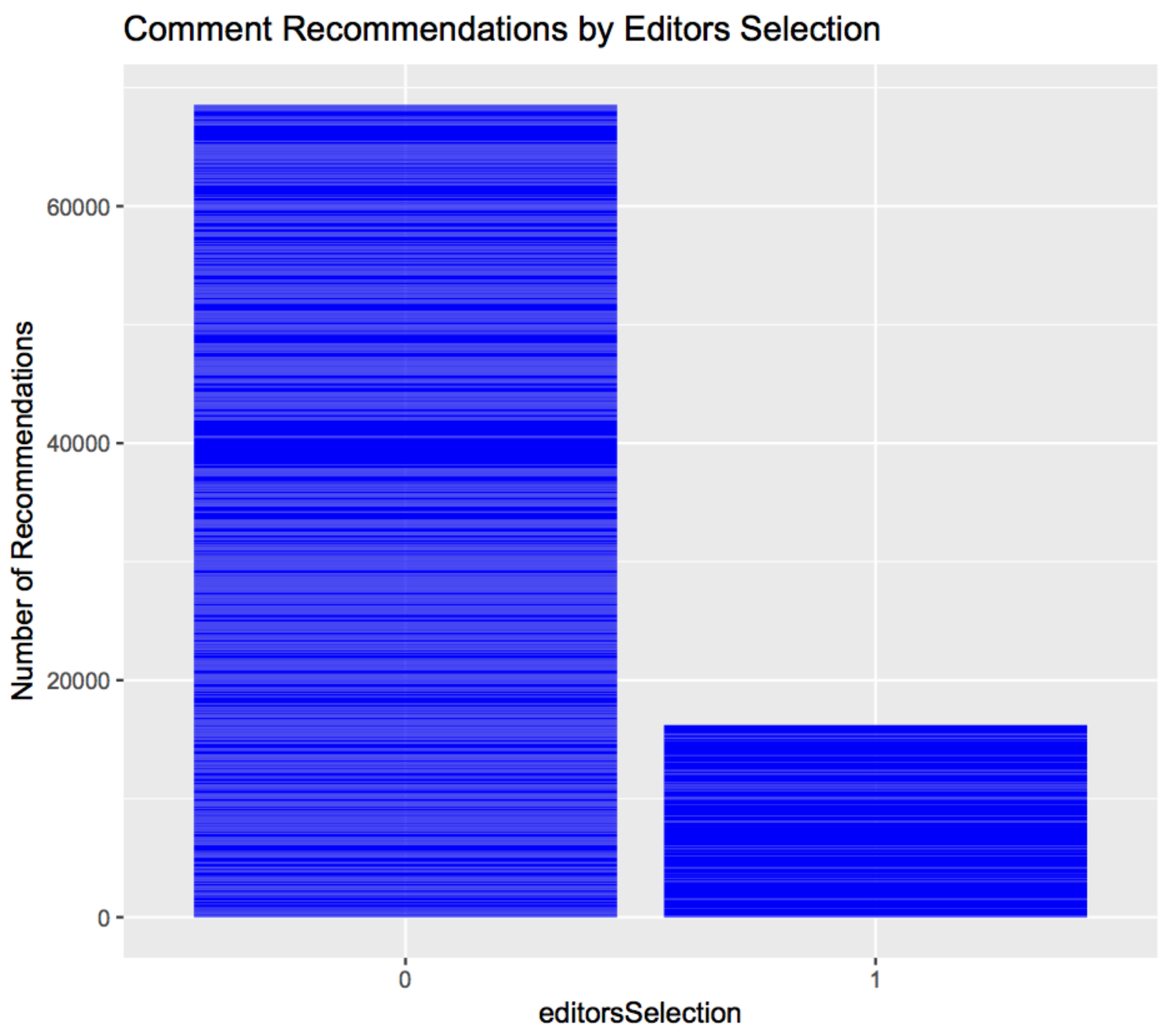


There also does appear to be some association between these variables and the number of recommendations, with higher-level comments getting more recommendations than replies.



editorsSelection

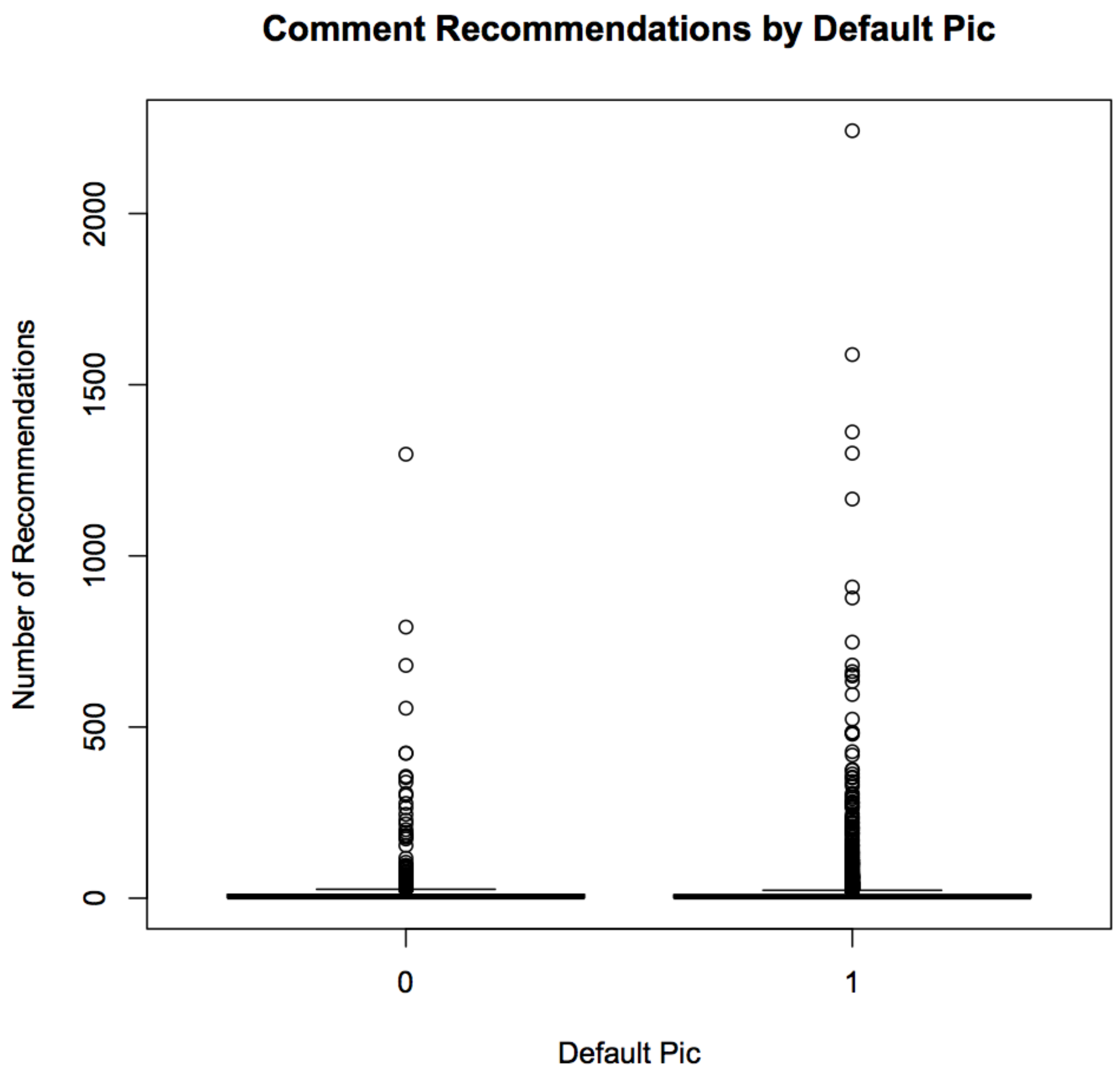
The editorsSelection feature takes values 0, 1, True, and False; we first re-coded this variable to have values 0 and 1 only. The plot below shows that it does seem to have a relationship with the number of recommendations:



picURL

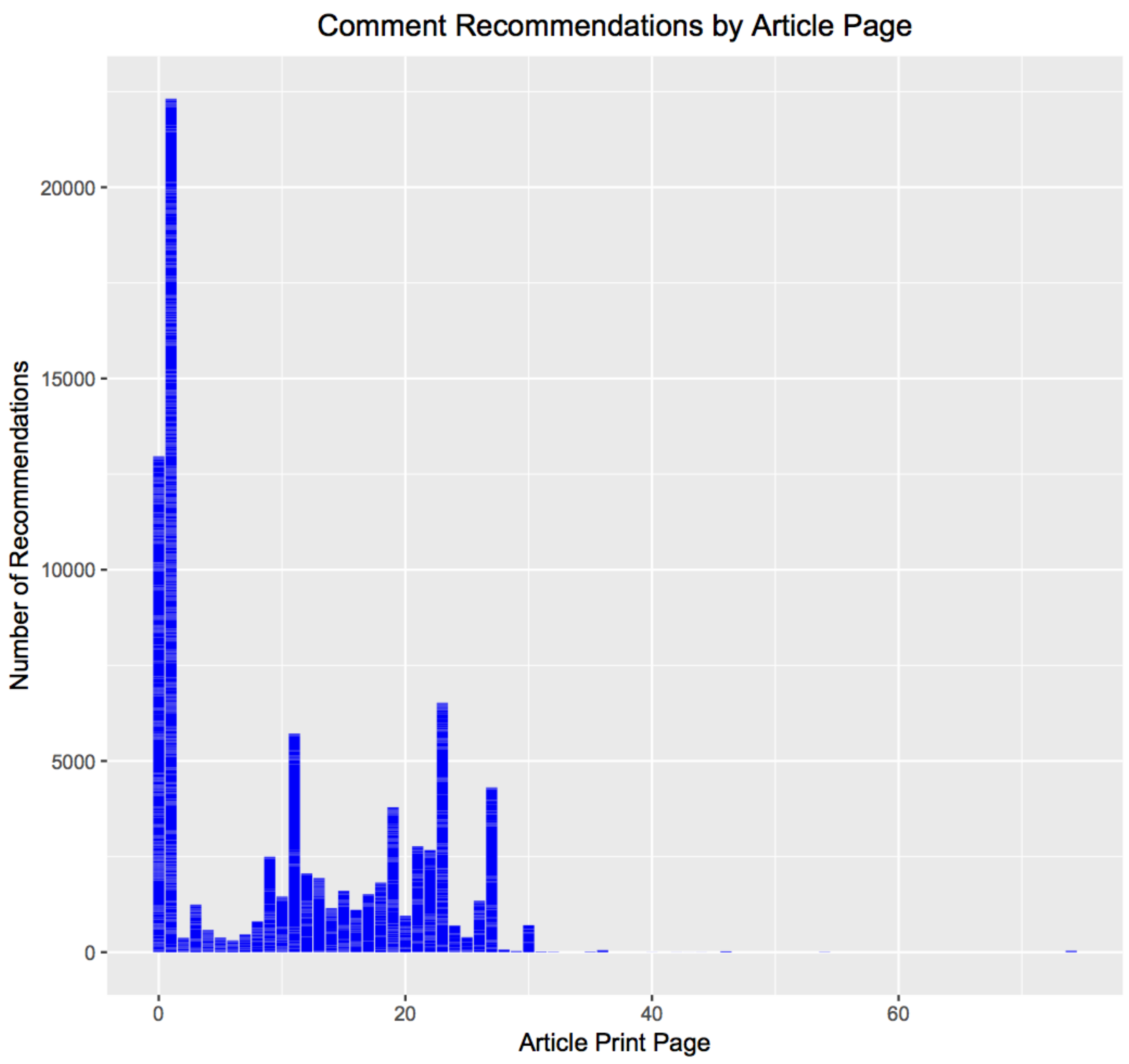
This feature shows the URL of the commenter's user pic. Most users have the default pic:
564322: https://graphics8.nytimes.com/images/apps/timespeople/none.png
9795: http://graphics8.nytimes.com/images/apps/timespeople/none.png

We used this to create a new variable, defaultPic, which is 1 if the user has the default pic and 0 otherwise. The plot below shows that there is some association with recommendations; users with default pics tend to have more recommendations than those with custom pics:



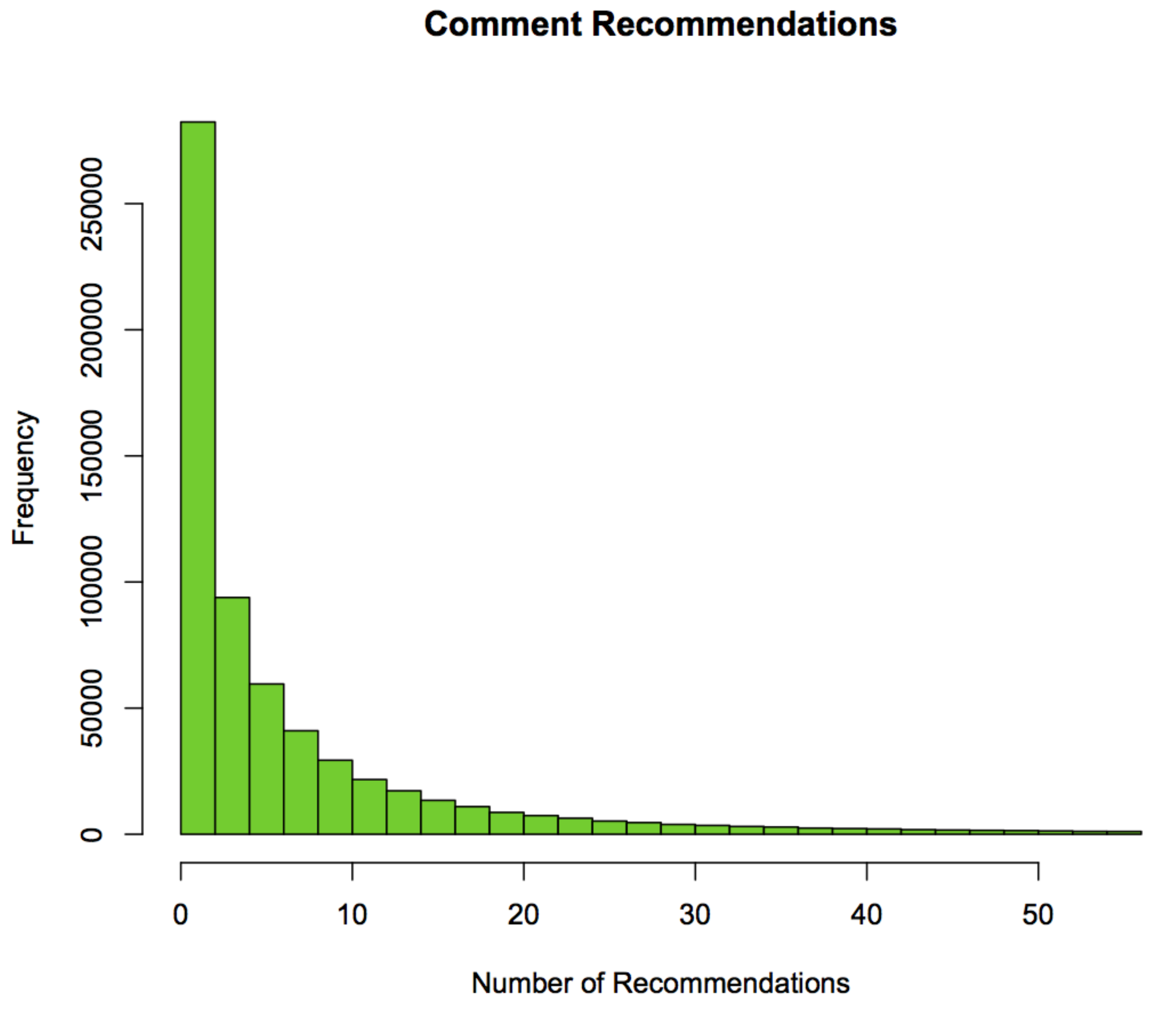
printPage

Articles printed on early pages of the paper also seem to have more recommendations than those in later pages. This makes sense, as we would expect popular articles or big news that makes the front page to correspondingly get more readers, comments, and recommendations.



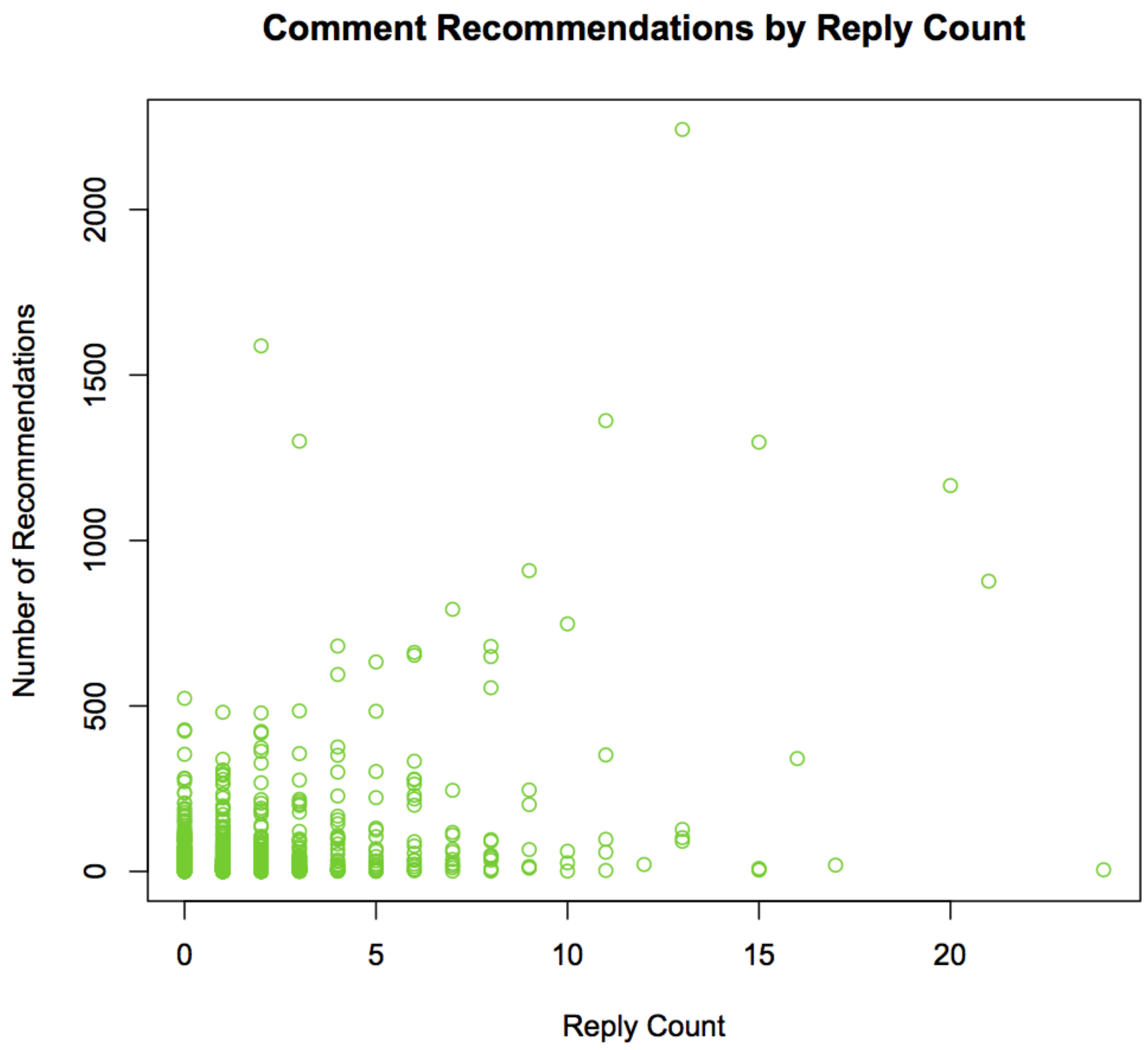
recommendations

The distribution of recommendations is heavily skewed, with a small number of comments receiving many hundreds or thousands of recommendations while the vast majority receive only a few recommendations. 99% of the comments have fewer than 306 recommendations, and 95% have fewer than 57 recommendations. The plot below shows the recommendation distribution for comments in the bottom 95% of recommendations:



replyCount

The number of replies seems to be associated with the number of recommendations, which makes sense as both in same way gauge the popularity of the comment. The plot below shows the relationship between these variables:

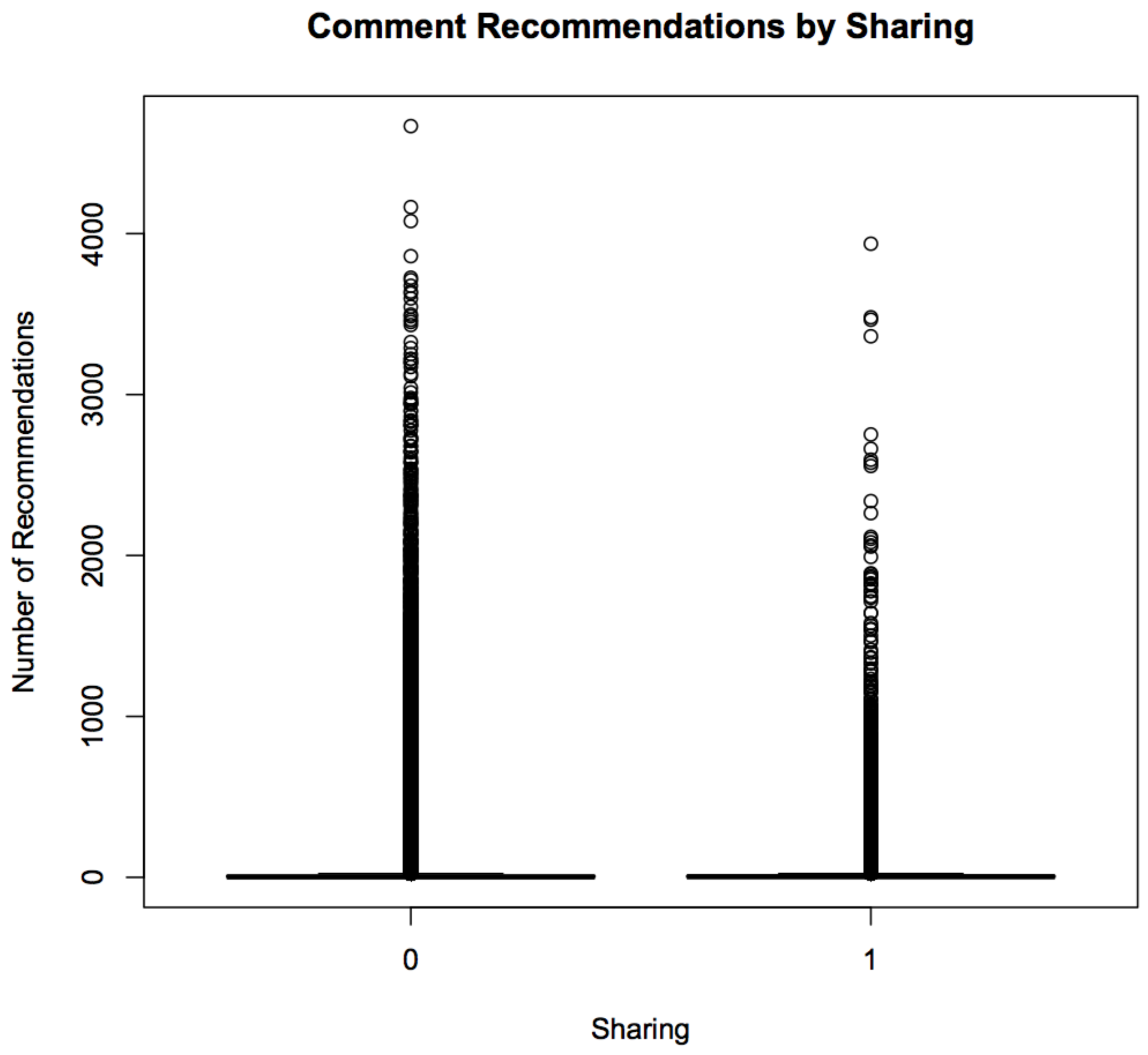


sectionName

Each comment is labeled with which of the 50 unique sections of the newspaper that it came from. Out of these, 254425 comments are from an unknown section, 149613 are NA, and 150326 are from the Politics section; the other sections have comparatively fewer comments. There are some clear differences between the number of recommendations for different sections. For example, the average number of recommendations for a comment in the Politics section is 21.4, higher than the average number of recommendations for a comment in an Unknown section (16.5, $p < 2.2e\text{-}16$).

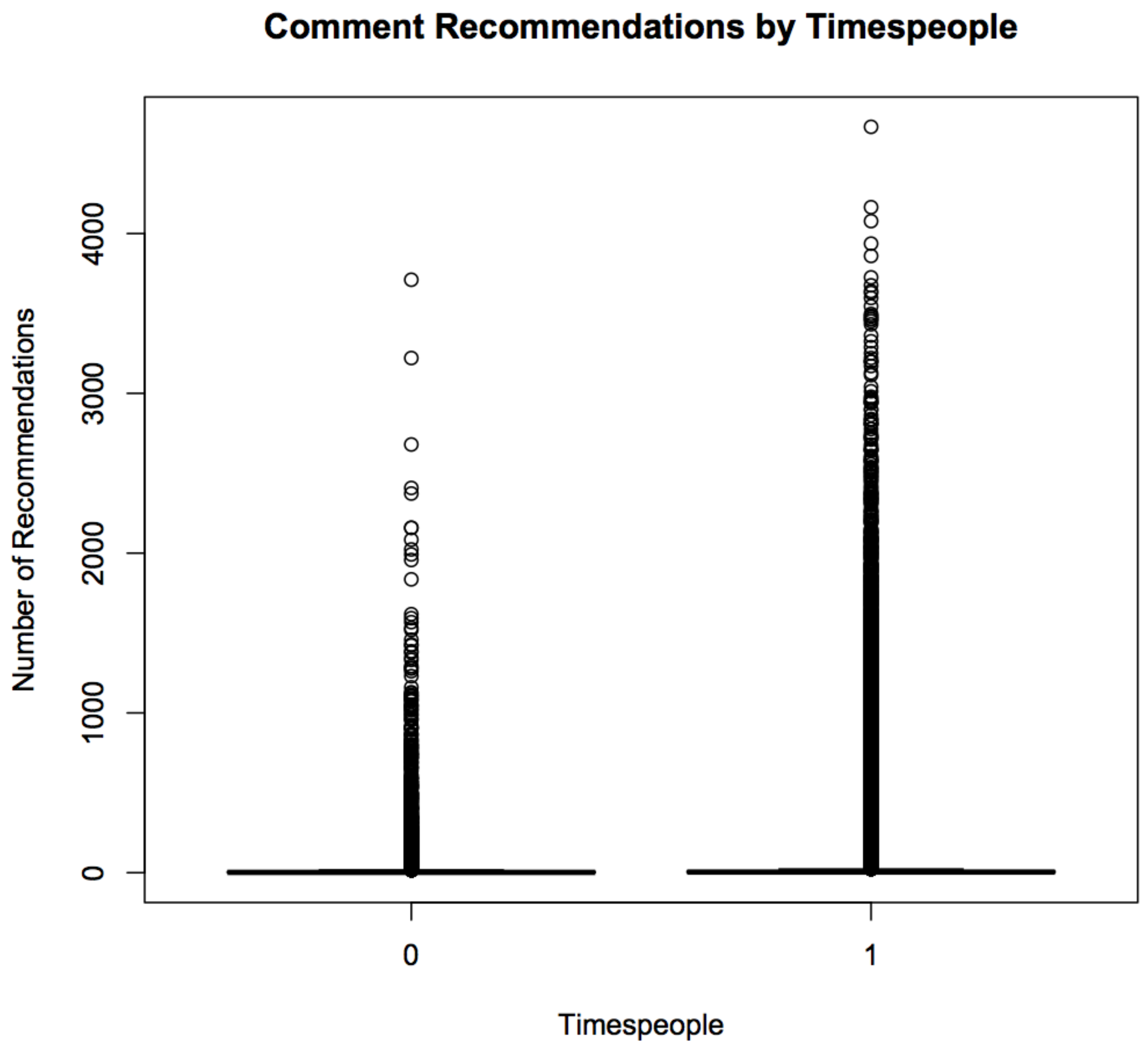
sharing

Sharing doesn't seem to be highly associated with recommendations, but it does seem that comments with sharing=0 may have more recommendations than comments with sharing=1:



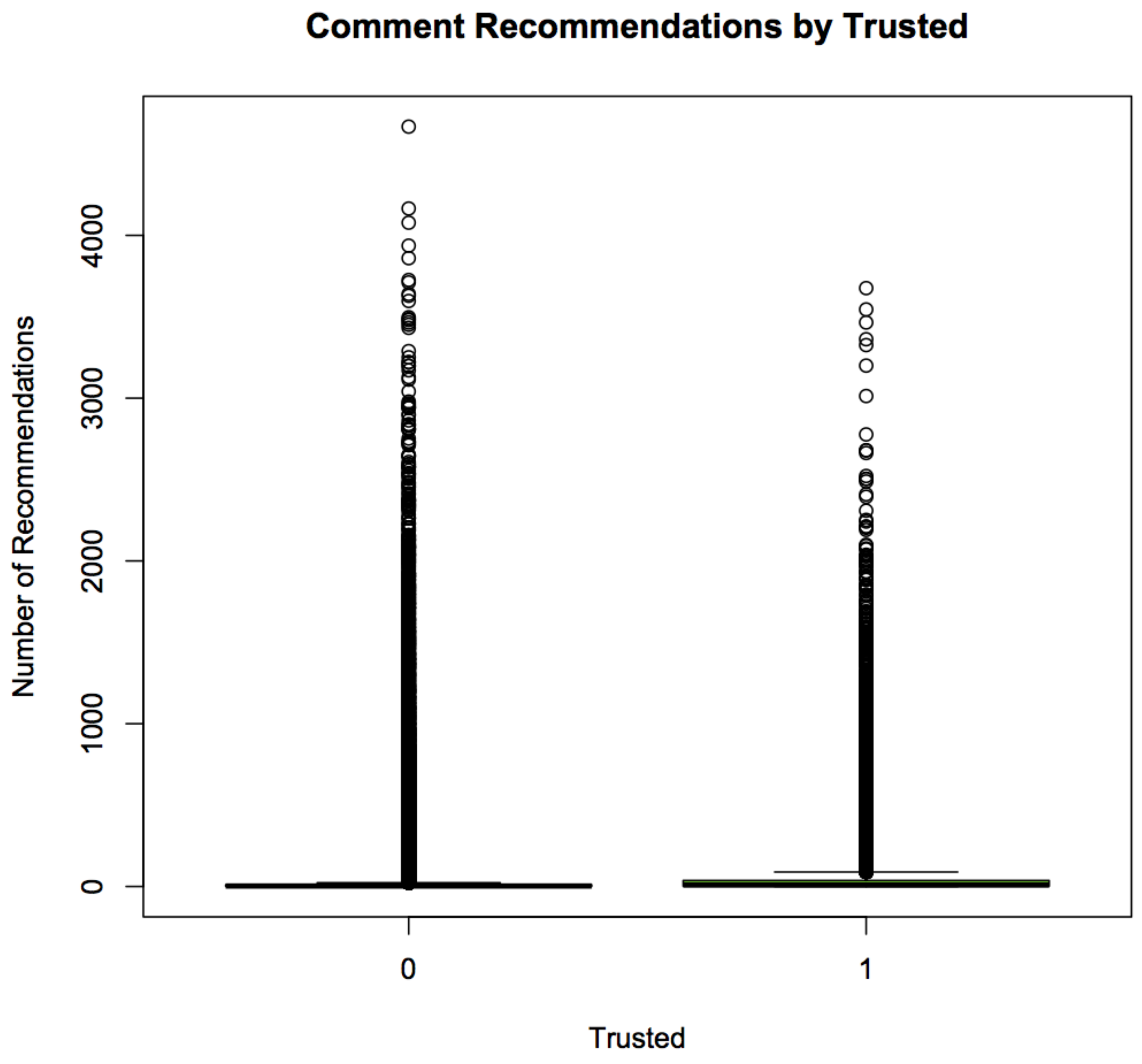
timespeople

Timespeople seems to have some slight association with recommendations, with comments where timespeople=1 having more recommendations:



trusted

Like sharing and timespeople, trusted is a 0/1 encoded variable. It has a stronger association with recommendations than the previous two variables; the average number of recommendations given to trusted=1 comments is higher than the average number of recommendations for trusted=0 comments.



typeOfMaterial

There are 14 different types of material, including News (372910), Op-Ed (240875), and Editorial (32046); the other material types all have less than 10000 observations. It appears that the average number of recommendations is different for different types of material. For example, comments from News articles have less recommendations on average than comments from Op-Ed articles (17.0 vs 20.1, $p < 2.2e-16$).

Article Data

The article dataset contains some of the same features as the comment dataset, such as articleWordCount, newDesk, printPage, and sectionName. It also includes several new features of interest. We can see a summary of the article data below:

```
'data.frame': 3445 obs. of 19 variables:
$ articleID : Factor w/ 3445 levels "5a4e778c7c459f29e79b2469",..: 905 904 903 902 901 900 899 898 897 896
```

. . .
$ articleWordCount: int 1322 1308 228 1114 777 1501 754 842 944 768 . . .
$ byline : Factor w/ 1486 levels “By A. ODYSSEUS PATRICK”,..: 1273 53 549 1371 169 1095 332 811 601 1079 . . .
$ documentType : Factor w/ 2 levels “article”,“blogpost”: 1 1 1 1 1 1 1 1 1 1 . . .
$ headline : Factor w/ 3123 levels “ ‘. . . And Being, Like, Really Smart’ ”,..: 2028 471 1547 1185 1115 1378 1301 2804 279 2804 . . .
$ keywords : Factor w/ 3035 levels “[‘#MeToo Movement’, ‘Academy Awards (Oscars)’, ‘Sex Crimes’, ‘Weinstein, Harvey’, ‘Sexual Harassment’, ‘Movies’, ‘Television’]”,..: 161 2759 322 2293 182 353 2811 1021 663 296 . . .
$ multimedia : int 68 68 0 61 68 68 68 65 68 66 . . .
$ newDesk : Factor w/ 42 levels “Arts&Leisure”,..: 35 40 16 7 22 9 22 31 16 2 . . .
$ printPage : int 5 17 16 24 0 1 0 0 22 11 . . .
$ pubDate : Factor w/ 3262 levels “2018-01-04T15:10:49Z”,..: 854 853 852 851 850 849 848 847 846 845 . . .
$ sectionName : Factor w/ 48 levels “Africa”,“Americas”,..: 46 36 46 16 46 1 46 46 46 9 . . .
$ snippet : Factor w/ 3406 levels “. . . Or solve Matthew Sewell’s pup-zle for a treat.”,..: 1548 3010 275 2700 1163 1808 2601 3119 1414 3261 . . .
$ source : Factor w/ 2 levels “International New York Times”,..: 2 2 2 2 2 2 2 2 2 2 . . .
$ typeOfMaterial : Factor w/ 14 levels “An Appraisal”,..: 9 9 9 6 12 9 12 9 9 9 . . .
$ webURL : Factor w/ 3445 levels “https://lens.blogs.nytimes.com/2018/01/04/a-race-to-photograph-the-citys-disappearing-di
894 899 877 883 882 902 885 890 876 870 . . .
$ topKeyword : Factor w/ 5 levels “Gun Control”,..: 5 4 2 4 2 2 4 2 2 2 . . .
$ bylineGender : Factor w/ 3 levels “female”,“male”,..: 1 2 1 3 2 3 2 1 2 1 . . .
$ snipSentiment : num -0.1732 -0.1179 -0.0258 -0.3556 -0.0589 . . .
$ snipWordCount : int 12 18 15 19 18 24 18 28 19 22 . . .

byline

The names of the article authors are given for each article. There are 1486 unique bylines, some with one author and some with two; 72 articles are also ‘By THE EDITORIAL BOARD’.

keywords

Each article is accompanied by a number of keywords. 217 articles have no keywords, and others have text containing many keywords separated by commas. There are 3035 unique keywords texts, although many keywords appear in multiple texts. In order to create new features for the keywords, we first began by pre-processing the keywords by splitting along the commas, removing unnecessary punctuation, and creating a list of all the unique keywords. There are 5238 unique keywords, with ‘Trump Donald J’, ‘United States Politics and Government’, ‘United States International Relations’, ‘New York City’, and ‘Politics and Government’ being the most popular five.

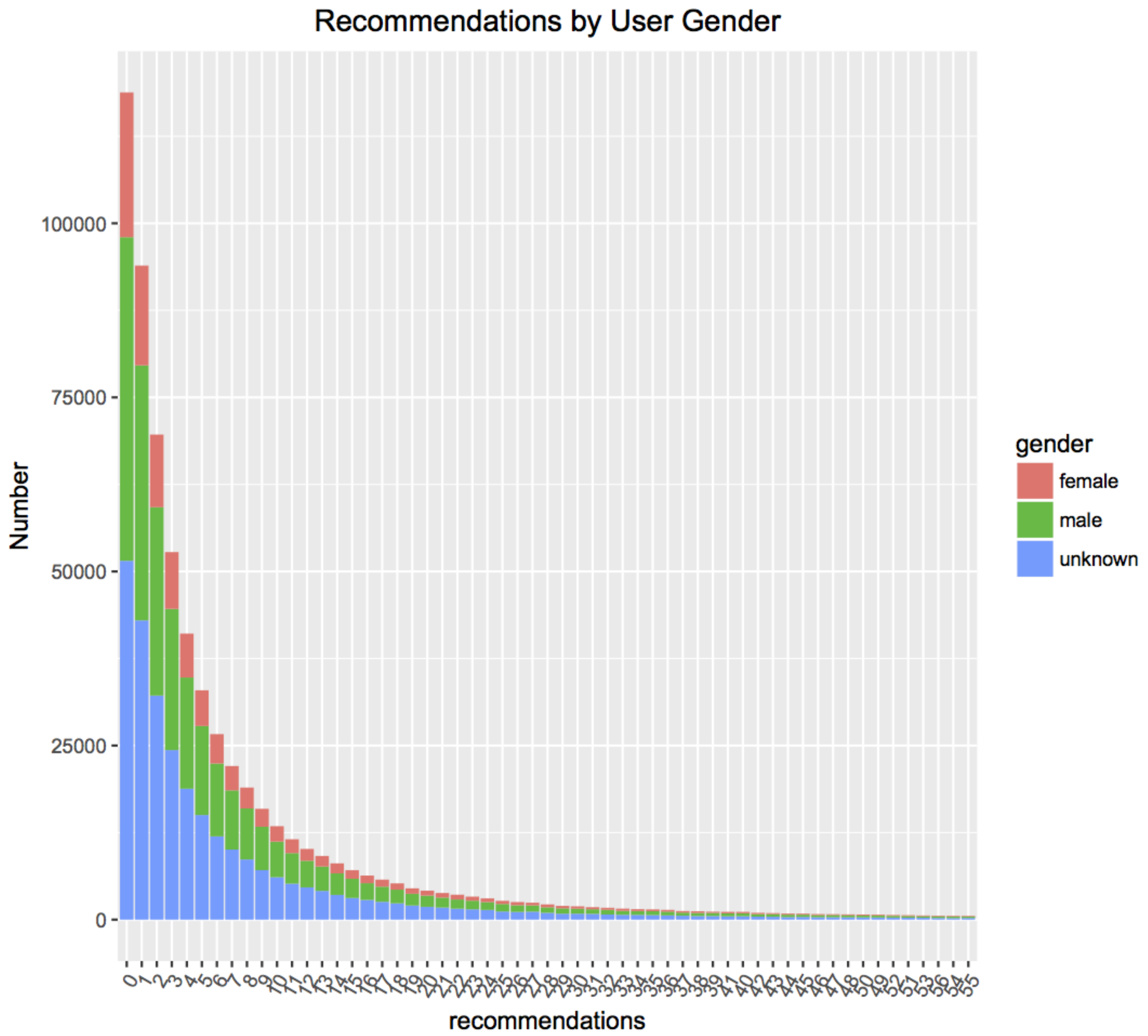
snippet

The article dataset also includes a single sentence of text from each article, which we analyzed in the section below to create new features.

Feature Creation

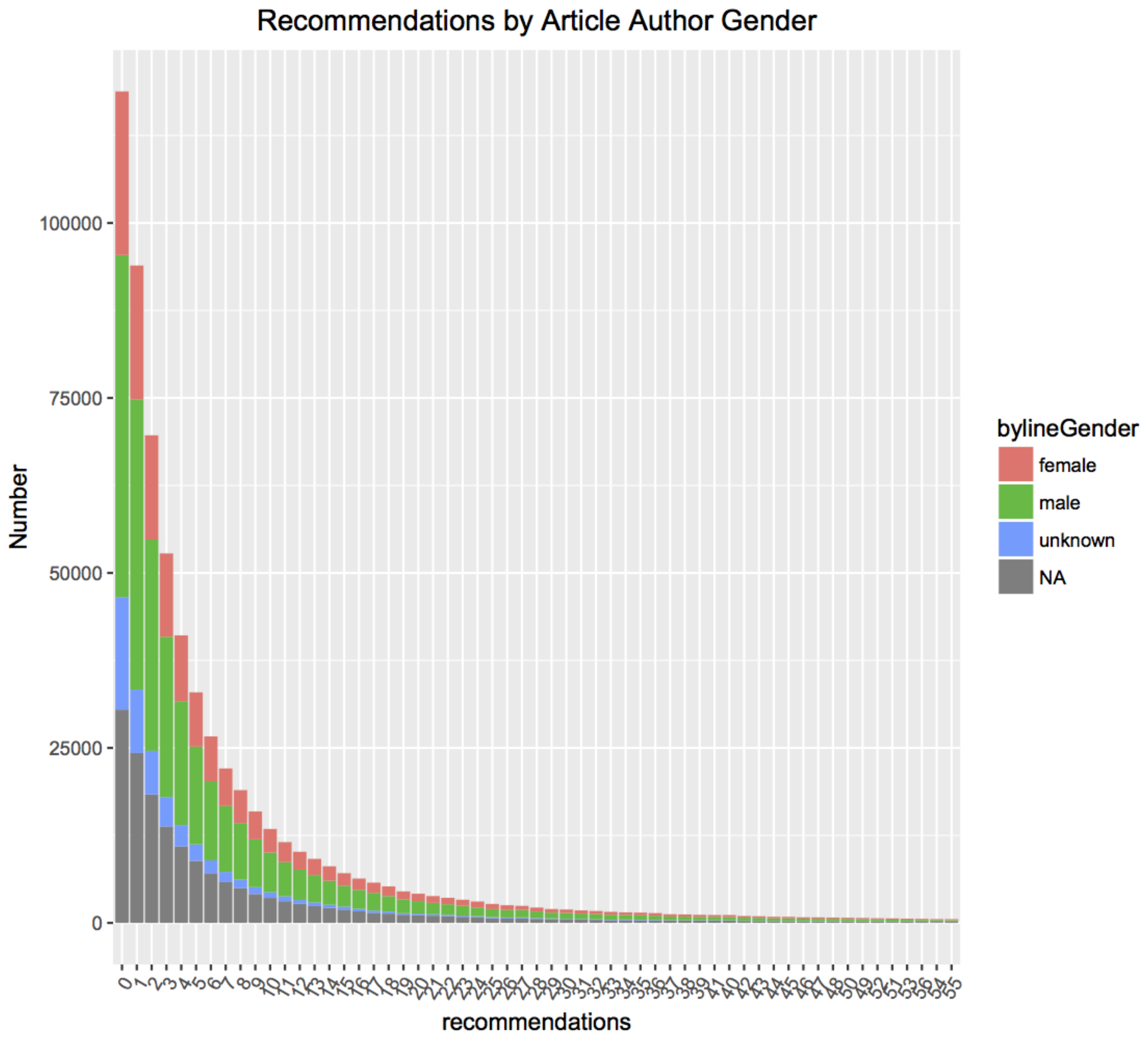
In addition to the new features mentioned above (approvalTime and defaultPic), we also created a number of other features that we felt might be useful in our models. First, we created a variable called topKeyword for each article. We sorted our list of all keywords by keyword popularity. The most popular keyword attached to each article was assigned to be its topKeyword. Finally, we made a topKeywordBin feature, which binned the top keywords into the five biggest categories: Trump Donald J, Politics and Government, Women and Girls, Gun Control, and NAs. The remaining keywords were placed into a new category 'Other'.

Next, we created features for the gender of the commenter's name and the article's author. For the commenter's name, we parsed the userDisplayName feature and took the first word of the text to be a first name. We then used the gender package to find the probability of the name being male or female. Probabilities were computed using U.S. Social Security Administration name data from 2012; names that weren't listed with the SSA were marked 'unknown'. A significant number of commenters used aliases such as 'Eyes Open' rather than their real first names, and were thus marked as having 'unknown' gender. The plot below shows the gender distribution for comments in the bottom 95% of recommendations (less than 57 recommendations).

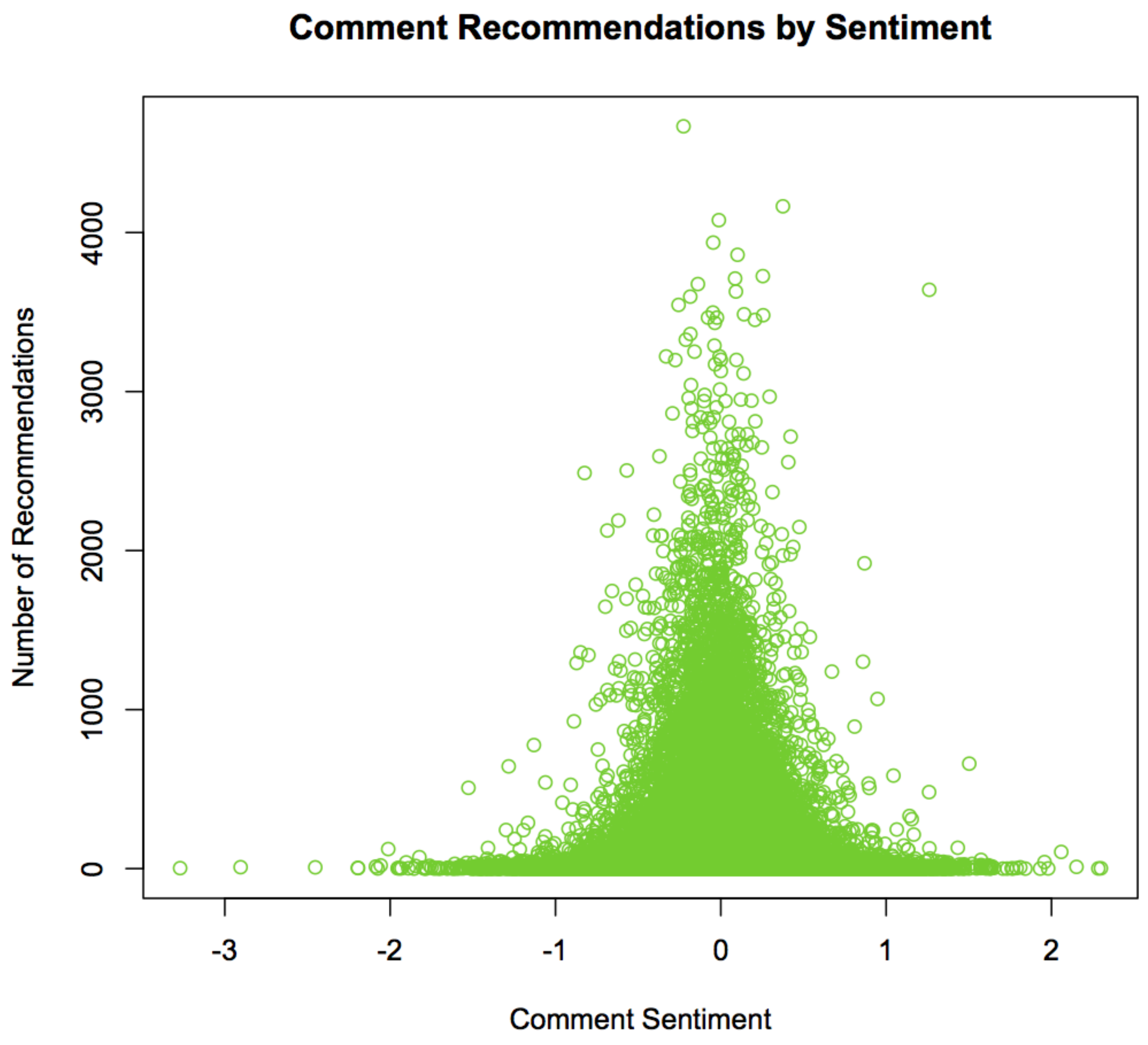


Although the proportions are similar, we can see that comments with no or very few recommendations are more likely to have unknown gender users, who have username aliases instead of SSA-recognized names for their user display name. The average number of recommendations for unknown user names is 17.6, significantly less than for male user names (18.2, $p < 0.004$) and female user names (19.7, $p < 4.7e\text{-}11$).

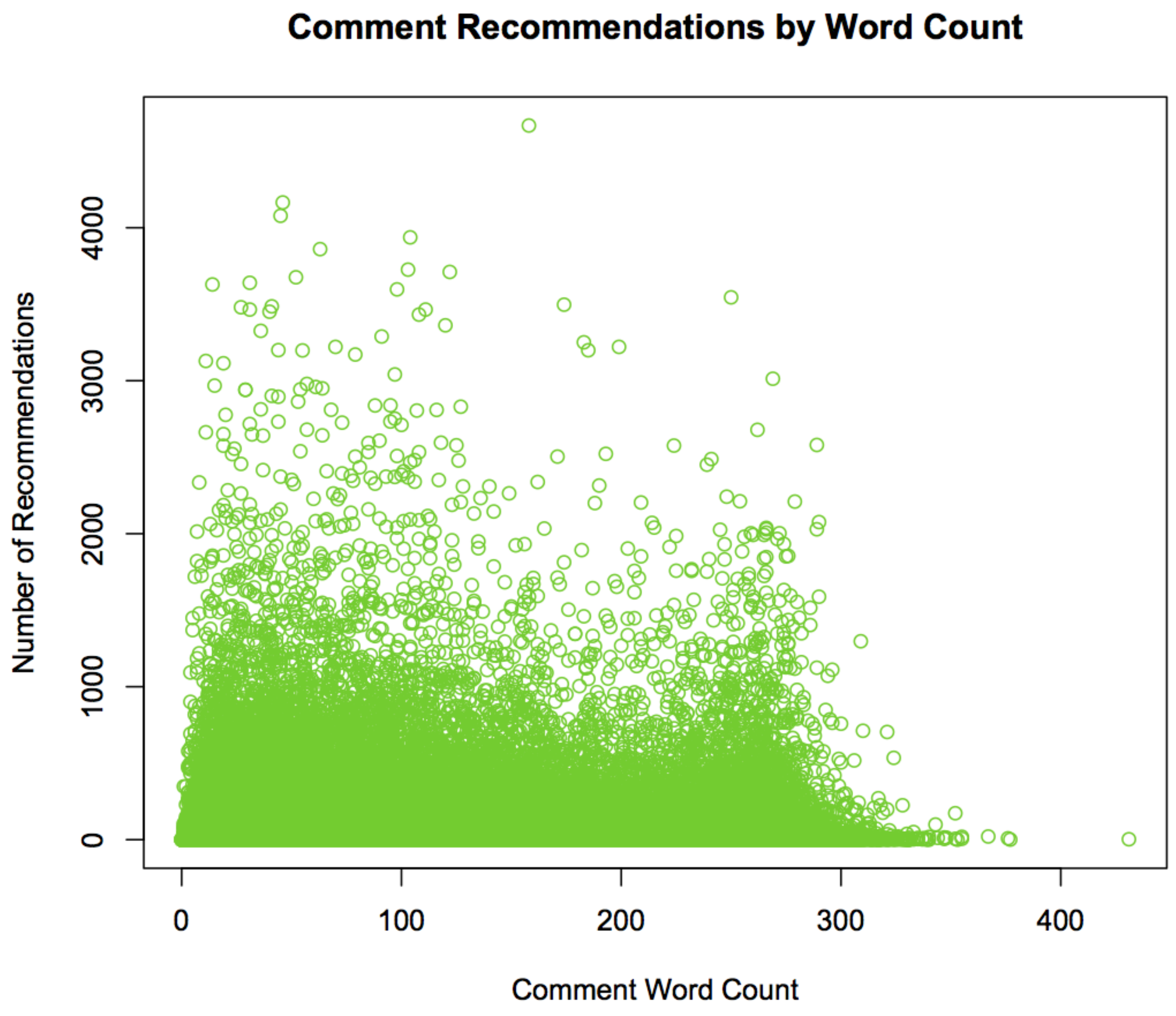
We also used the gender package to compute the probable gender of the article's author, bylineGender, as we felt it was possible that this would have an effect on recommendations as well. Similar to user gender, we found that unknown gender authors had less average recommendations than male authors (12.4 vs 18.7, $p < 2.2e\text{-}16$) and female authors (12.4 vs 19.7, $p < 2.2e\text{-}16$), although the effect was significantly stronger. The plot below shows the same trends:



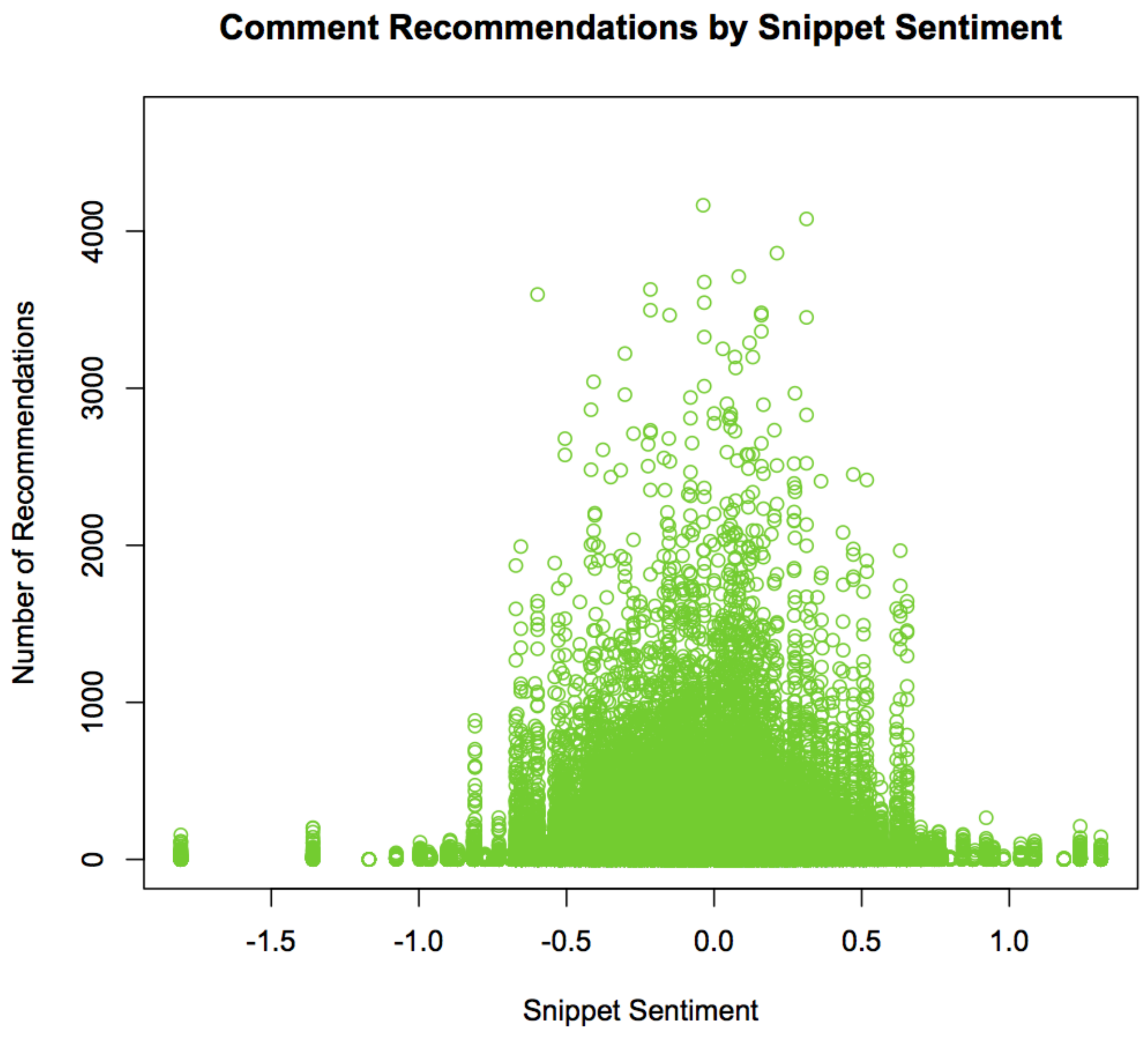
We also performed sentiment analysis using the sentimentr package for both the full body on the comments, and the snippets of the articles. For each length of text, we computed both the word count, commentWordCount and snipWordCount, and the numerical sentiment score, commentSentiment and snipSentiment. Negative sentiment values were given for negative comments and positive sentiment values for positive comments. We can see in the plot below that more neutral comments tended to get more recommendations.



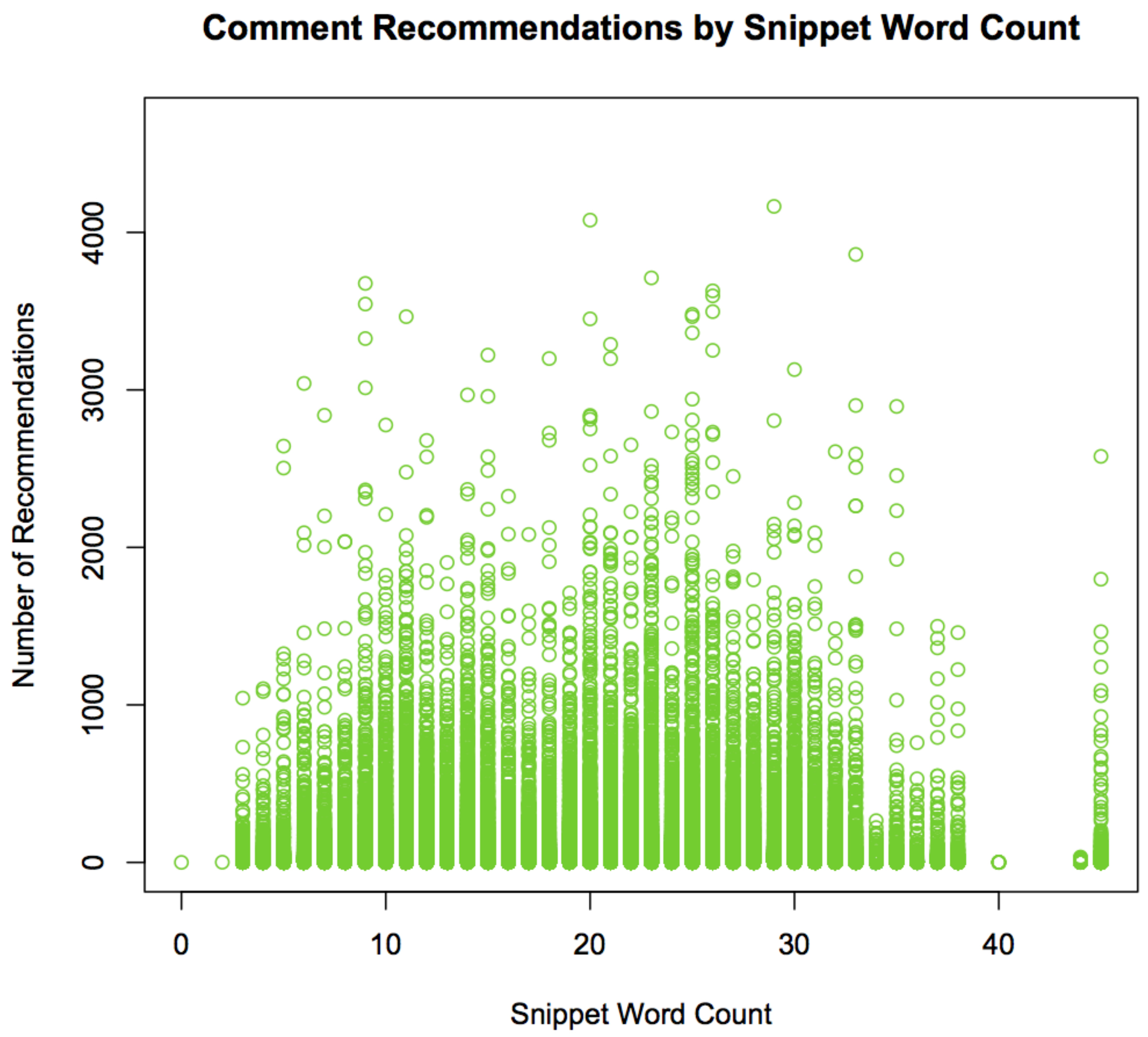
The patterns betweeen comment word count and comment recommendations are less clear, but it does seem that there is some positive relationship:



We saw some of the same patterns in the snippet sentiment analysis and word count. Articles with more neutral sentiments in their snippets tended to have comments with more recommendations:



Trends were less clear for the relationship between snippet word count and recommendations. We created this feature because of the possibility that it might interact with comment word count as well as recommendations; potentially a wordy comment in a wordy article might get more recommendations, and vice versa.



Finally, we created new variables by binning various levels for typeOfMaterial, newDesk, and sectionName. All three of these variables had a high number of factor levels, which the large majority of comments under around 4 factors. For typeOfMaterial, we kept the levels 'News', 'Op-Ed', and 'Editorial'. Any comments marked 'News Analysis' were also sorted into the 'News' category; the remainder were placed in the category 'Other'. For newDesk, we kept the levels 'Business', 'Editorial', 'National', 'OpEd', and 'Washington', and placed other levels into the new category 'Other'. Finally, for sectionName, we kept the levels 'Politics', 'Sunday Review', and 'Unknown', and placed other categories into 'Other'. For each of these variables, NA values were kept as NA rather than being placed into a different category.

Predictive Modeling

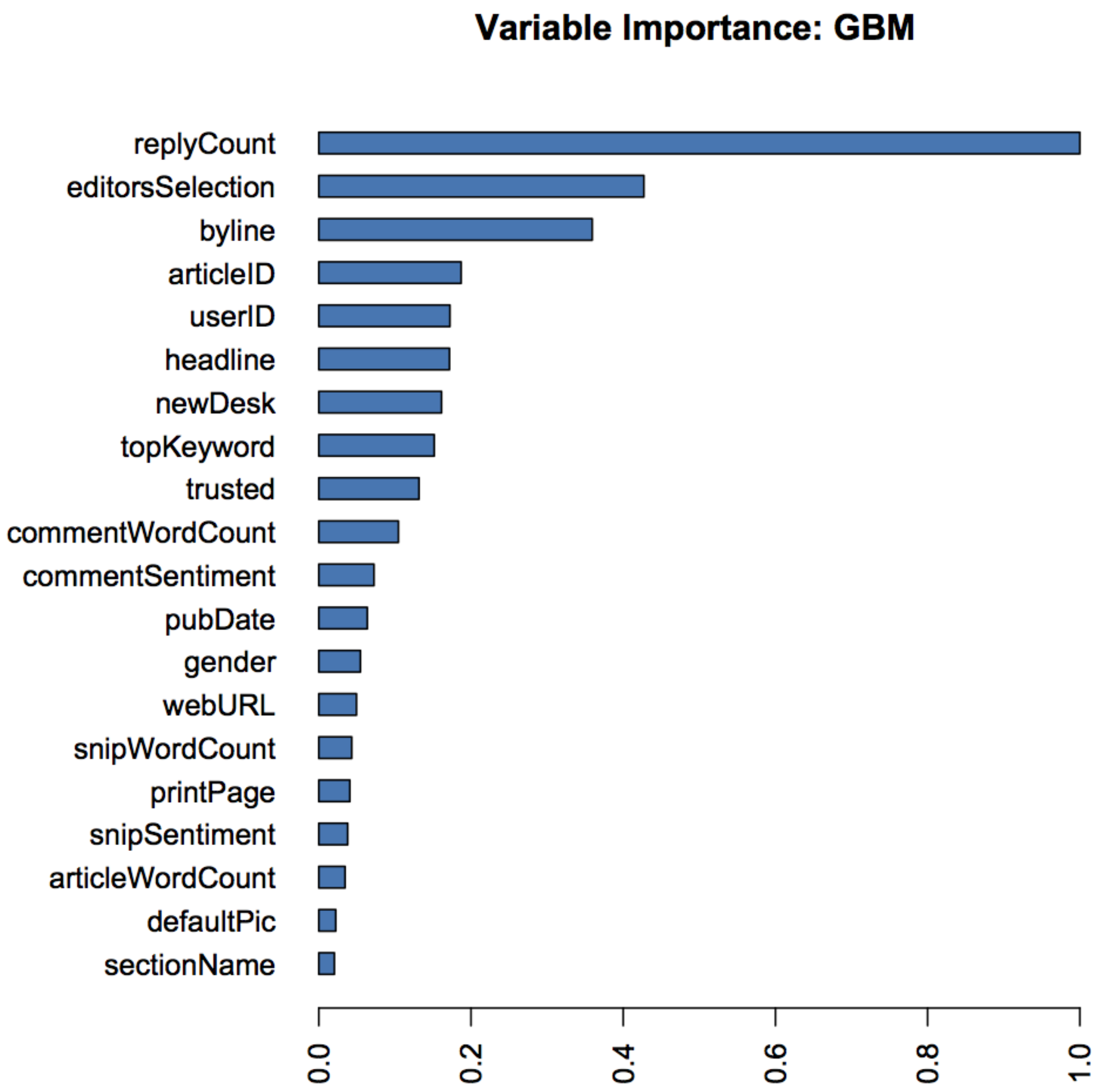
We developed a number of different models, and particularly focused on gradient-boosted machines and random forests, which gave the best results. The majority of models were built in H2O, although we also tried using the caret package. The training comments and training articles datasets were merged into a single data frame, which was then split into a training dataset with 75% of the data (499237 comments) and a validation dataset with 25% of the data (166159 comments). The validation dataset was used for model tuning and hyperparameter optimization, and to provide estimates of accuracy before uploading promising models to the Kaggle competition.

We also split the training dataset further into smaller datasets on several occasions; these datasets were used to quickly train new models that we wanted to try, giving an initial estimate of their performance to establish whether they were worth exploring further.

Gradient Boosted Machines

We began with gradient boosted machines in H2O. For our first models, we included most of the available features as predictors, only removing text variables such as commentBody, keywords, and snippet. We then began to remove features that were judged to be uninformative or unimportant, either based on our exploratory data analysis or based on their performance in variable importance plots. Features that were not used or hardly used in the models were removed, and we then re-trained a new model to see whether performance improved.

The variable importance plot below shows the top 20 most important variables from an initial GBM that retained several features that were later removed:

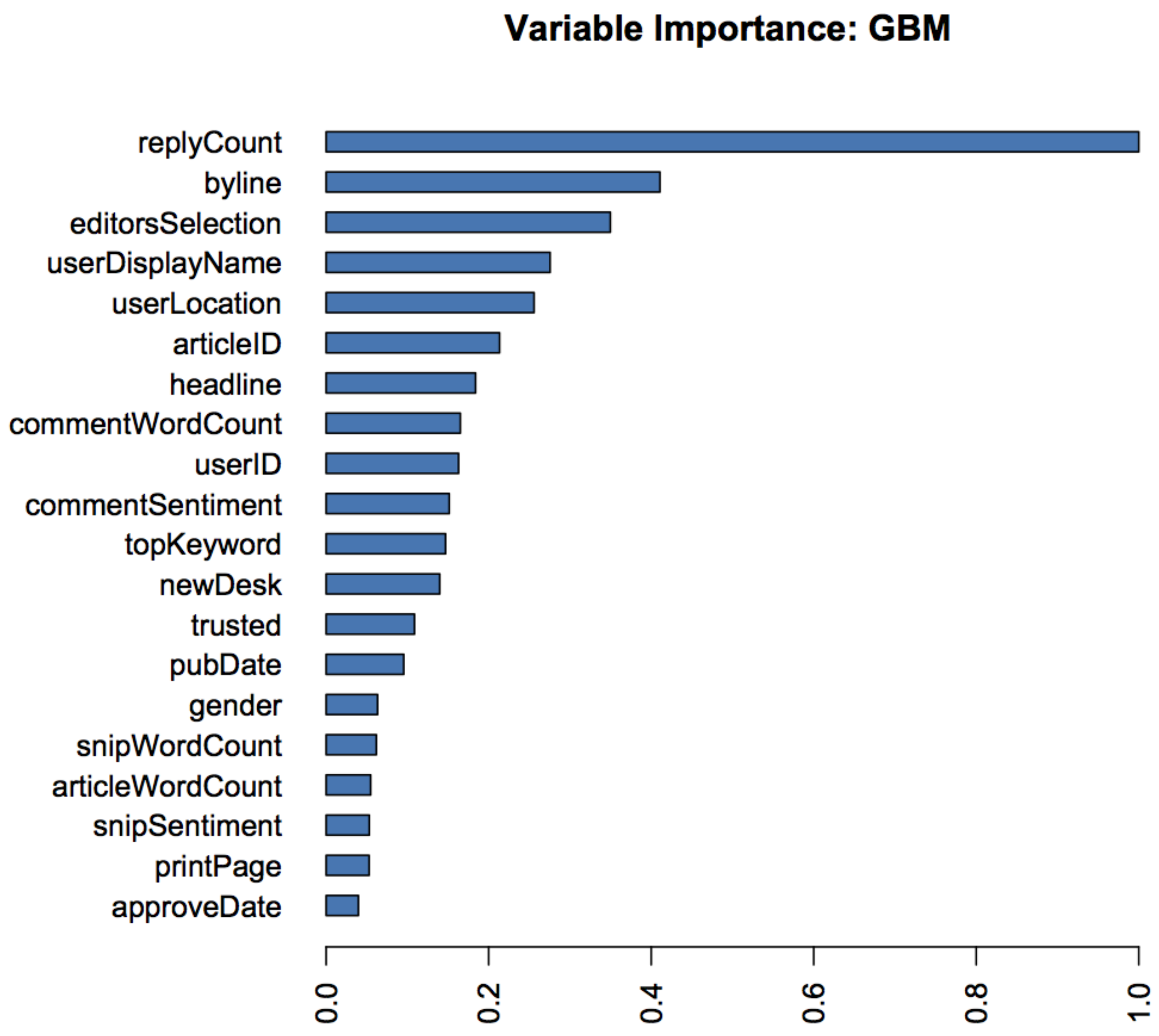


We can see that variables like userLocation and userDisplayName are given high importance, along with variables like permID which are nearly unique for each comment. We therefore removed these feature to improve performance. In contrast, the variables timespeople and inReplyTo seemed potentially promising in our initial analyses, but contributed very little to the model. We can see that our newly created features topKeyword, commentWordCount, commentSentiment, gender, snipWordCount, snipSentiment, and defaultPic are all in the top twenty.

After trying various combinations of features, and removing features with low variable importance, we found that our best models generally included most of the many available features. Even features such as articleID seemed to add something to the model, even if the performance improvement was very small; they were therefore retained.

After manually tuning with different choices of features and parameters, we found our best GBM model, which also produced our lowest MAE of 16.9 in the Kaggle competition, although it gave an improved result of 16.01 on the validation set. This GBM has 200 trees, a maximum depth of 13, learn rate of 0.05, nbins of

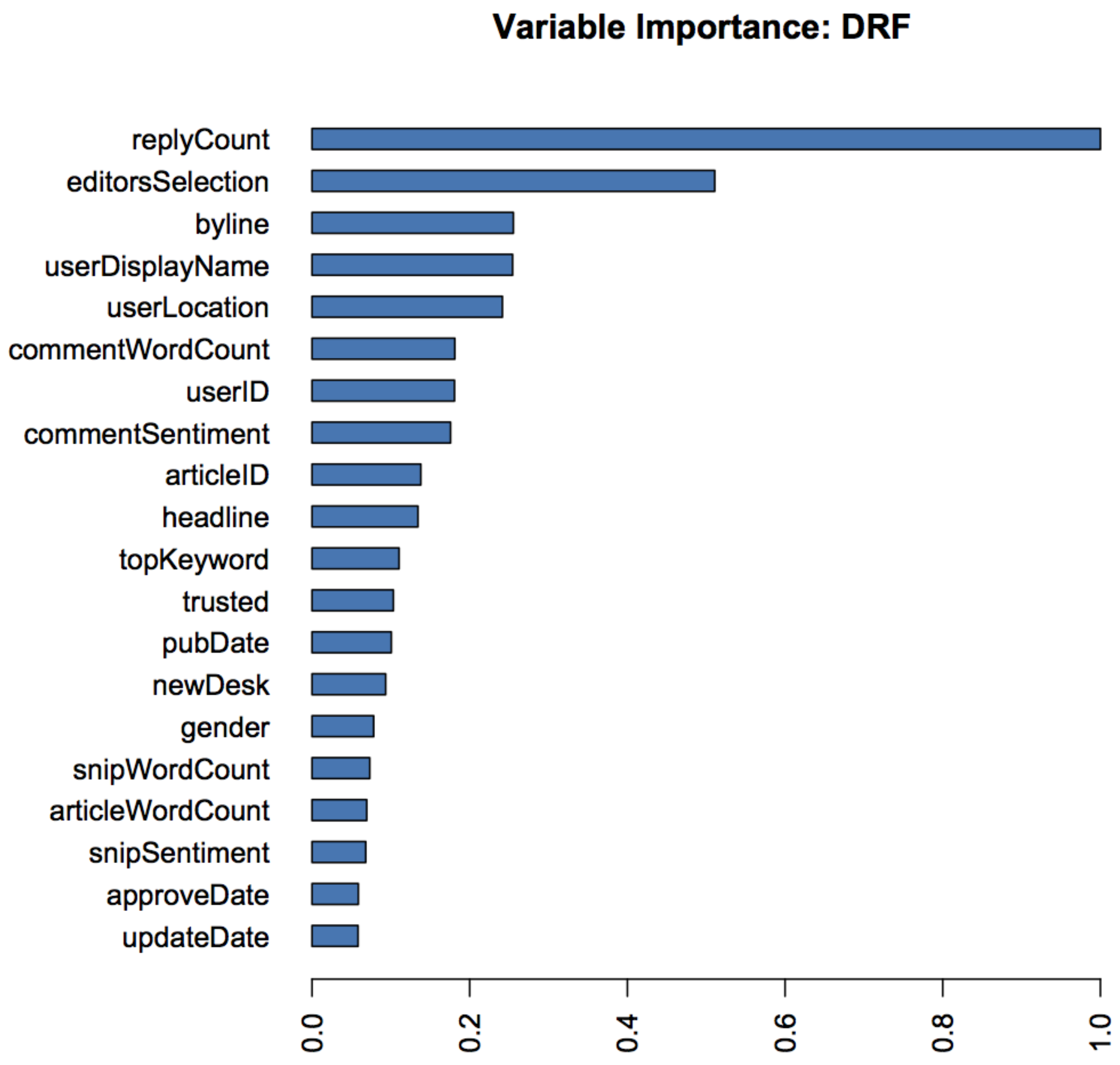
20, min_rows of 3, and a sample rate of 0.8. The plot below shows the variable importance for this GBM:



After manual tuning, we also ran a random grid search to attempt to find a better combination of parameters. However, the best models found with grid search performed essentially the same as the best manually tuned models. This is likely because we were only able to run 200 models without crashing H2O, due to our computer's memory limitations.

Random Forest

We followed a similar pattern with random forest models, although we focused less on these as their performance was generally worse than the GBMs while being somewhat slower. We tuned the parameters for several random forest models to find the best fit, and then ran a 200-model random grid search in H2O. The best random forest model did not perform as well as the best GBM. It had 50 trees, a maximum depth of 20, min_rows of 1, nbins of 20, and the default sample rate of 0.632. This random forest gave an MAE of 16.5 on the validation set. The plot below shows the importance of the variables in the random forest:



We also attempted to fit a random forest model in caret, but found that it took a significantly longer time to train. We did train a caret random forest with a small subset of 5000 comments from the dataset, but its performance was not as good as the H2O models.

Other Models

We trained a number of other models as well, including neural networks and GLMs. The neural networks were trained in H2O, but proved very slow for decreased performance. Additionally, deeplearning neural network models could not handle the same number of variables, so a reduced feature set was used when training them. The best neural network model gave an MAE of 18.2, significantly worse than both random forests and GBMs. We also ran several GLMs using the poisson and gaussian distributions, using H2O, caret, and the glm() function, but each had significantly worse performance on the validation data set, so these models were discarded.

Summary

Overall, we were able to achieve mean absolute error scores of around 16.0 at best using our predictive models, although this performance worsened to about 16.9 on the Kaggle competition test data. Our best models were gradient boosted machines in H2O, with random forests in H2O performing second best of the implementation methods tried. The variables commentWordCount, commentSentiment, and topKeyword were the most important of our created features to both the GBM and random forest models; however, the original features replyCount, editorsSelection, and byline were the most important overall. Our new features gender, snipWordCount, and snipSentiment were also helpful to various models. In contrast, the binned variables we created were among the least important to the models.

Code

```r
#############################################
# Loading Data & Packages
#############################################

library(sentimentr)
library(gender)
library(dplyr)
library(plyr)
library(h2o)
library(ModelMetrics)
library(caret)
library(pscl)
library(MASS)
library(readr)
library(magrittr)
library(tidyverse)

# Load data

train_comments <- read.csv("~/Desktop/412/final/train_comments.csv")
test_comments <- read.csv("~/Desktop/412/final/test_comments.csv")

train_articles <- read.csv("~/Desktop/412/final/train_articles.csv")
test_articles <- read.csv("~/Desktop/412/final/test_articles.csv")




#############################################
# Exploratory Data Analysis
#############################################

head(train_comments)
str(train_comments)
summary(train_comments)


# remove variables with constant values
train_comments <- train_comments[,-which(names(train_comments) %in% c('commentTitle', 'recommendedFlag'
test_comments <- test_comments[,-which(names(test_comments) %in% c('commentTitle', 'recommendedFlag', 'r


set.seed(1)
N <- nrow(train_comments)
train_samp <- train_comments[sample(1:N, 5000),]

# approveDate

plot(train_samp$approvalTime, train_samp$recommendations, xlab="Time to Comment Approval", ylab="Number

# articleID
length(unique(train_comments$articleID))
length(unique(test_comments$articleID))
length(which(test_comments$articleID %in% unique(train_comments$articleID)))

# articleWordCount
plot(train_samp$articleWordCount, train_samp$recommendations, xlab="Article Word Count", ylab="Number of


# commentID
length(unique(train_comments$commentID))


# commentSequence
cor(train_comments$commentSequence, train_comments$recommendations)
plot(train_samp$commentSequence, train_samp$recommendations, xlab="Comment Sequence", ylab="Number of Re


# commentType & depth
summary(train_comments$commentType)
par(mfrow=c(1,2))
barplot(c(497181, 47, 168168), names.arg = c("comment", "reportReply", "userReply"), xlab="Comment Type
barplot(c(497183, 2437, 165776), names.arg = c("1", "3", "2"), xlab="Comment Depth", main = "Comment Dep

plot(as.factor(train_samp$depth), train_samp$recommendations, xlab="Comment Depth", ylab="Number of Reco


# editorsSelection
for (i in 1:length(train_comments$editorsSelection)) {
  if (train_comments$editorsSelection[i] == "False") {
    train_comments$editorsSelection[i] <- 0
  } else {
    if (train_comments$editorsSelection[i] == "True") {
      train_comments$editorsSelection[i] <- 1
    }
  }
}

train_comments$editorsSelection <- factor(train_comments$editorsSelection)

set.seed(1)
N <- nrow(train_comments)
```

```
train_samp <- train_comments[sample(1:N, 5000),]

plot(train_samp$editorsSelection, train_samp$recommendations, xlab="Editors Selection", ylab="Number of

ggplot(train_samp, aes(x = editorsSelection, y = recommendations)) + geom_bar(stat = "identity", fill="


# inReplyTo
cor(train_comments$inReplyTo, train_comments$recommendations)
plot(train_samp$inReplyTo, train_samp$recommendations, xlab="In Reply To", ylab="Number of Recommendati


# newDesk
table(train_comments$newDesk)
plot(train_comments$newDesk, xlab="Article Desk", main = "Article Desks", col=3)


# picURL
defaultPic_train <- ifelse(train_comments$picURL == "https://graphics8.nytimes.com/images/apps/timespeop
train_comments$defaultPic <- as.factor(defaultPic_train)
train_comments$picURL <- NULL

defaultPic_test <- ifelse(test_comments$picURL == "https://graphics8.nytimes.com/images/apps/timespeopl
test_comments$defaultPic <- as.factor(defaultPic_test)
test_comments$picURL <- NULL

set.seed(1)
N <- nrow(train_dat)
train_samp <- train_dat[sample(1:N, 5000),]

cor(train_comments$defaultPic, train_comments$recommendations)
plot(train_samp$defaultPic, train_samp$recommendations, xlab="Default Pic", ylab="Number of Recommendat


# printPage
ggplot(train_samp, aes(x = printPage, y = recommendations)) + geom_bar(stat = "identity", fill="blue")
cor(train_comments$printPage, train_comments$recommendations)


# recommendations
quantile(train_comments$recommendations, 0.99) # 99% of comments have 306 or fewer recommendations
quantile(train_comments$recommendations, 0.95) # 99% of comments have 57 or fewer recommendations

hist(train_comments$recommendations[which(train_comments$recommendations < 57)], xlab="Number of Recomme


# replyCount
cor(train_comments$replyCount, train_comments$recommendations)

plot(train_samp$replyCount, train_samp$recommendations, xlab="Reply Count", ylab="Number of Recommendat


# sectionName
```

```r
length(unique(train_comments$sectionName))
summary(train_comments$sectionName)
mean(train_comments$recommendations[which(train_comments$sectionName == "Politics")]) # 21.43333
mean(train_comments$recommendations[which(train_comments$sectionName == "Unknown")]) # 16.45828
t.test(train_comments$recommendations[which(train_comments$sectionName == "Politics")], train_comments$r

# sharing
length(unique(train_comments$sharing))
cor(train_comments$sharing, train_comments$recommendations)
plot(as.factor(train_comments$sharing), train_comments$recommendations, xlab="Sharing", ylab="Number of



# timespeople
length(unique(train_comments$timespeople))
cor(train_comments$timespeople, train_comments$recommendations)
plot(as.factor(train_comments$timespeople), train_comments$recommendations, xlab="Timespeople", ylab="Nu

# trusted
length(unique(train_comments$trusted))
cor(train_comments$trusted, train_comments$recommendations)
plot(as.factor(train_comments$trusted), train_comments$recommendations, xlab="Trusted", ylab="Number of

# typeOfMaterial
length(unique(train_comments$typeOfMaterial))
summary(train_comments$typeOfMaterial)
mean(train_comments$recommendations[which(train_comments$typeOfMaterial == "Op-Ed")])
mean(train_comments$recommendations[which(train_comments$typeOfMaterial == "News")])
t.test(train_comments$recommendations[which(train_comments$typeOfMaterial == "News")], train_comments$re


# byline
length(unique(train_articles$byline))
summary(train_articles$byline)

# keywords
length(unique(train_articles$keywords))

keywords <- c()
for (i in 1:length(train_articles$keywords)) {
  kws <- as.vector(strsplit(as.character(train_articles$keywords[i]), "'")[[1]])
  kws <- gsub('[[:punct:] ]+',' ',kws)
  keywords <- c(keywords, kws[which(nchar(kws) > 3)])
}

keywords <- as.factor(keywords)
length(unique(keywords))
table(keywords)
keywords_tab[order(keywords_tab, decreasing=T)]
```

```r
#############################################
# Feature Creation
#############################################


####### KEYWORDS #########


# get 2 most popular keywords for each article
keywords <- c()
for (i in 1:length(train_articles$keywords)) {
  kws <- as.vector(strsplit(as.character(train_articles$keywords[i]), "'")[[1]])
  kws <- gsub('[[:punct:] ]+',' ',kws)
  keywords <- c(keywords, kws[which(nchar(kws) > 3)])
}

keywords <- as.factor(keywords)
keywords_tab <- table(keywords)
keywords_sort <- keywords_tab[order(keywords_tab, decreasing=T)]

tk <- function(x) keywords_sort[x][[1]]

keyword1 <- c()
keyword2 <- c()
for (i in 1:length(train_articles$keywords)) {
  kws <- as.vector(strsplit(as.character(train_articles$keywords[i]), "'")[[1]])
  kws <- gsub('[[:punct:] ]+',' ',kws)
  kws <- kws[which(nchar(kws) > 3)]

  if (length(kws) > 0) {
    counts <- sapply(kws, FUN = tk)
    keyword1[i] <- kws[which.max(counts)]
    if (length(kws) > 1) {
      keyword2[i] <- kws[which(counts == sort(counts, decreasing=T)[2])][1]
      } else {
      keyword2[i] <- "None"
      }
    } else {
    keyword1[i] <- "None"
    keyword2[i] <- "None"
    }
}

keyword1 <- as.factor(keyword1)
#keyword2 <- as.factor(keyword2)
train_articles$topKeyword <- keyword1
#train_articles$keyword2 <- keyword2


# get 2 most popular keywords for each article
keywords <- c()
for (i in 1:length(test_articles$keywords)) {
  kws <- as.vector(strsplit(as.character(test_articles$keywords[i]), "'")[[1]])
  kws <- gsub('[[:punct:] ]+',' ',kws)
  keywords <- c(keywords, kws[which(nchar(kws) > 3)])
}

keywords <- as.factor(keywords)
keywords_tab <- table(keywords)
keywords_sort <- keywords_tab[order(keywords_tab, decreasing=T)]

tk <- function(x) keywords_sort[x][[1]]

keyword1 <- c()
keyword2 <- c()
for (i in 1:length(test_articles$keywords)) {
  kws <- as.vector(strsplit(as.character(test_articles$keywords[i]), "'")[[1]])
  kws <- gsub('[[:punct:] ]+',' ',kws)
  kws <- kws[which(nchar(kws) > 3)]

  if (length(kws) > 0) {
    counts <- sapply(kws, FUN = tk)
    keyword1[i] <- kws[which.max(counts)]
    if (length(kws) > 1) {
      keyword2[i] <- kws[which(counts == sort(counts, decreasing=T)[2])][1]
      } else {
      keyword2[i] <- "None"
      }
    } else {
    keyword1[i] <- "None"
    keyword2[i] <- "None"
    }
}

keyword1 <- as.factor(keyword1)
#keyword2 <- as.factor(keyword2)
test_articles$topKeyword <- keyword1
#test_articles$keyword2 <- keyword2



####### TOP KEYWORD #########

# train_articles
keyw <- as.character(train_articles$topKeyword)
for (i in 1:length(keyw)) {
  if (keyw[i] != "Trump Donald J" & keyw[i] != "Politics and Government" & keyw[i] != "Women and Girls"
    if (keyw[i] == "United States Politics and Government") {
      keyw[i] <- "Politics and Government"
    } else {
        keyw[i] <- "Other"
    }
  }

}
keyw <- as.factor(keyw)
```

```
train_articles$topKeywordBin <- keyw


# test_articles
t_keyw <- as.character(test_articles$topKeyword)
for (i in 1:length(t_keyw)) {
  if (t_keyw[i] != "Trump Donald J" & t_keyw[i] != "Politics and Government" & t_keyw[i] != "Women and (
    if (t_keyw[i] == "United States Politics and Government") {
      t_keyw[i] <- "Politics and Government"
    } else {
        t_keyw[i] <- "Other"
    }
  }

}
t_keyw <- as.factor(t_keyw)

test_articles$topKeywordBin <- t_keyw



####### BYLINE GENDER #########

# train
udn <- as.character(train_articles$byline)


for (i in 1:length(udn)) {
  udn[i] <- strsplit(udn[i], " ")[[1]][2]
  udn[i] <- gsub('[[:punct:] ]+','',udn[i])
}

yrs <- rep(2012, length(udn))
df <- data_frame(first_names = udn, years = yrs)
results <- gender_df(df, name_col = "first_names", year_col="years", method="ssa")
df <- df %>% left_join(results, by=c("first_names" = "name"))

blgenders_train <- ifelse(df$proportion_female >= 0.5, "female", ifelse(df$proportion_male >= 0.5, "male
blgenders_train[which(is.na(blgenders_train))] <- 'unknown'

train_articles$bylineGender <- as.factor(blgenders_train)

# test
udn <- as.character(test_articles$byline)

for (i in 1:length(udn)) {
  udn[i] <- strsplit(udn[i], " ")[[1]][2]
  udn[i] <- gsub('[[:punct:] ]+','',udn[i])
}

yrs <- rep(2012, length(udn))
df <- data_frame(first_names = udn, years = yrs)
results <- gender_df(df, name_col = "first_names", year_col="years", method="ssa")
```

```
df <- df %>% left_join(results, by=c("first_names" = "name"))

blgenders_test <- ifelse(df$proportion_female >= 0.5, "female", ifelse(df$proportion_male >= 0.5, "male
blgenders_test[which(is.na(blgenders_test))] <- 'unknown'

test_articles$bylineGender <- as.factor(blgenders_test)

# analysis
temp <- join(train_comments, train_articles)

ggplot(temp[which(temp$recommendations < 57),], aes(x=reorder(recommendations, recommendations, functio

mean(temp$recommendations[which(temp$bylineGender == "male")]) # 18.66882
mean(temp$recommendations[which(temp$bylineGender == "female")]) # 19.74109
mean(temp$recommendations[which(temp$bylineGender == "unknown")]) # 12.35216

t.test(temp$recommendations[which(temp$bylineGender == "male")], temp$recommendations[which(temp$byline
t.test(temp$recommendations[which(temp$bylineGender == "male")], temp$recommendations[which(temp$byline
t.test(temp$recommendations[which(temp$bylineGender == "female")], temp$recommendations[which(temp$bylin




####### SNIPPET SENTIMENT #########

# train
snip <- as.character(train_articles$snippet)

snip_sentences <- get_sentences(snip)
snip_sentiments <- sentiment_by(snip_sentences)

train_articles$snipSentiment <- snip_sentiments$ave_sentiment
train_articles$snipWordCount <- snip_sentiments$word_count

# test
t_snip <- as.character(test_articles$snippet)

t_snip_sentences <- get_sentences(t_snip)
t_snip_sentiments <- sentiment_by(t_snip_sentences)

test_articles$snipSentiment <- t_snip_sentiments$ave_sentiment
test_articles$snipWordCount <- t_snip_sentiments$word_count





##### Merge data

train_dat <- join(train_comments, train_articles)
test_dat <- join(test_comments, test_articles)
```

```
# snip sentiment analysis
plot(train_dat$snipSentiment, train_dat$recommendations, xlab="Snippet Sentiment", ylab="Number of Recom

plot(train_dat$snipWordCount, train_dat$recommendations, xlab="Snippet Word Count", ylab="Number of Reco



####### COMMENT SENTIMENT #########

# train
comments <- as.character(train_dat$commentBody)

comments_sentences <- get_sentences(comments)
comments_sentiments <- sentiment_by(comments_sentences)

train_dat$commentWordCount <- comments_sentiments$word_count
train_dat$commentSentiment <- comments_sentiments$ave_sentiment
train_dat$commentBody <- NULL

# test
t_comments <- as.character(test_dat$commentBody)

t_comments_sentences <- get_sentences(t_comments)
t_comments_sentiments <- sentiment_by(t_comments_sentences)

test_dat$commentWordCount <- t_comments_sentiments$word_count
test_dat$commentSentiment <- t_comments_sentiments$ave_sentiment
test_dat$commentBody <- NULL

# analysis
cor(train_dat$commentSentiment, train_dat$recommendations)
plot(train_dat$commentSentiment, train_dat$recommendations, xlab="Comment Sentiment", ylab="Number of Re

cor(train_dat$commentWordCount, train_dat$recommendations)
plot(train_dat$commentWordCount, train_dat$recommendations, xlab="Comment Word Count", ylab="Number of N



####### USER DISPLAY NAME GENDER #########

udn <- as.character(train_dat$userDisplayName)

for (i in 1:length(udn)) {
  udn[i] <- strsplit(udn[i], " ")[[1]][1]
  udn[i] <- gsub('[[:punct:] ]+','',udn[i])
}

yrs <- rep(2012, length(udn))
df <- data_frame(first_names = udn, years = yrs)
results <- gender_df(df, name_col = "first_names", year_col="years", method="ssa")
df <- df %>% left_join(results, by=c("first_names" = "name"))

genders_train <- ifelse(df$proportion_female >= 0.5, "female", ifelse(df$proportion_male >= 0.5, "male"
genders_train[which(is.na(genders_train))] <- 'unknown'

train_dat$gender <- as.factor(genders_train)


udn <- as.character(test_dat$userDisplayName)

for (i in 1:length(udn)) {
  udn[i] <- strsplit(udn[i], " ")[[1]][1]
  udn[i] <- gsub('[[:punct:] ]+','',udn[i])
}

yrs <- rep(2012, length(udn))
df <- data_frame(first_names = udn, years = yrs)
results <- gender_df(df, name_col = "first_names", year_col="years", method="ssa")
df <- df %>% left_join(results, by=c("first_names" = "name"))

genders_test <- ifelse(df$proportion_female >= 0.5, "female", ifelse(df$proportion_male >= 0.5, "male",
genders_test[which(is.na(genders_test))] <- 'unknown'

test_dat$gender <- as.factor(genders_test)

# analysis
plot(train_dat$gender, train_dat$recommendations, xlab="User Gender", ylab="Number of Recommendations",

ggplot(train_dat[which(train_dat$recommendations < 57),], aes(x=reorder(recommendations, recommendation

mean(train_dat$recommendations[which(train_dat$gender == "male")]) # 18.22437
mean(train_dat$recommendations[which(train_dat$gender == "female")]) # 19.66927
mean(train_dat$recommendations[which(train_dat$gender == "unknown")]) # 17.56595

t.test(train_dat$recommendations[which(train_dat$gender == "male")], train_dat$recommendations[which(tr
t.test(train_dat$recommendations[which(train_dat$gender == "male")], train_dat$recommendations[which(tr
t.test(train_dat$recommendations[which(train_dat$gender == "female")], train_dat$recommendations[which(



####### TYPE OF MATERIAL #########

# train_dat
mat <- as.character(train_dat$typeOfMaterial)
for (i in 1:length(mat)) {
  if (mat[i] != "News" & mat[i] != "Op-Ed" & mat[i] != "Editorial") {
    if (mat[i] == "News Analysis") {
      mat[i] <- "News"
      } else {
        mat[i] <- "Other"
      }
    }
}
mat <- as.factor(mat)

train_dat$typeOfMaterialBin <- mat
# test_dat
t_mat <- as.character(test_dat$typeOfMaterial)
for (i in 1:length(t_mat)) {
  if (t_mat[i] != "News" & t_mat[i] != "Op-Ed" & t_mat[i] != "Editorial") {
    if (t_mat[i] == "News Analysis") {
      t_mat[i] <- "News"
      } else {
        t_mat[i] <- "Other"
      }
    }
}
t_mat <- as.factor(t_mat)

test_dat$typeOfMaterialBin <- t_mat



####### NEW DESK #########


# train_dat
ndesk <- as.character(train_dat$newDesk)
for (i in 1:length(ndesk)) {
  if (ndesk[i] != "Business" & ndesk[i] != "Editorial" & ndesk[i] != "National" & ndesk[i] != "OpEd" & 
    ndesk[i] <- "Other"
    }
}
ndesk <- as.factor(ndesk)

train_dat$newDeskBin <- ndesk

# test_dat
t_ndesk <- as.character(test_dat$newDesk)
for (i in 1:length(t_ndesk)) {
  if (t_ndesk[i] != "Business" & t_ndesk[i] != "Editorial" & t_ndesk[i] != "National" & t_ndesk[i] != "
    t_ndesk[i] <- "Other"
    }
}
t_ndesk <- as.factor(t_ndesk)

test_dat$newDeskBin <- t_ndesk


####### SECTION NAME #########

# train_dat
sname <- as.character(train_dat$sectionName)
for (i in 1:length(sname)) {
  if (sname[i] != "Politics" & sname[i] != "Sunday Review" & sname[i] != "Unknown" & !is.na(sname[i])) 
    sname[i] <- "Other"
    }
}
```

```r
sname <- as.factor(sname)

train_dat$sectionNameBin <- sname

# test_dat
t_sname <- as.character(test_dat$sectionName)
for (i in 1:length(t_sname)) {
  if (t_sname[i] != "Politics" & t_sname[i] != "Sunday Review" & t_sname[i] != "Unknown" & !is.na(t_snar
    t_sname[i] <- "Other"
    }
}
t_sname <- as.factor(t_sname)

test_dat$sectionNameBin <- t_sname







#############################################
# Data Splitting & Preparation
#############################################

temp <- train_dat
# remove features
train_dat <- train_dat[,-which(names(train_dat) %in% c('commentID', 'keywords', 'commentBody', 'snippet

# split data
set.seed(1)
N <- nrow(train_dat)
id_test <- sample(1:N, 0.25*N)
train <- train_dat[-id_test,]
test <- train_dat[id_test,]


# setting up for h2o
h2o.init(nthreads=-1)

htest_comments <- as.h2o(test_dat)
hdat <- as.h2o(train_dat)
hdat_split <- h2o.splitFrame(hdat, ratios = c(0.75))
htrain <- hdat_split[[1]]
htest <- hdat_split[[2]]


# caret pre-processing
trainX <- train_dat[,-which(names(train_dat) %in% c('recommendations'))]
trainPP <- preProcess(trainX, method=c("BoxCox", "center", "scale"))
trainTrans <- predict(trainPP, trainX)
trainTrans$recommendations <- train_dat$recommendations

testX <- test_dat
```

```r
testPP <- preProcess(testX, method=c("BoxCox", "center", "scale"))
testTrans <- predict(testPP, testX)




#############################################
# Modeling
#############################################


##### GRADIENT BOOSTED MACHINES ######


# H2O GBMs

Xnames <- names(htrain)[-which(names(htrain) %in% c('recommendations'))]

GBM_1 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 50, max_depth = 10, learn_rate = 0.1,
                 nbins = 20)

h2o.performance(GBM_1, htest) # RMSE 63.48551, MAE 16.64614
h2o.varimp_plot(GBM_1, 20)

Xnames <- names(htrain)[-which(names(htrain) %in% c('recommendations', 'permID', 'userDisplayName'))]

GBM_1 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 50, max_depth = 10, learn_rate = 0.1,
                 nbins = 20)

h2o.performance(GBM_1, htest) # MAE 16.73173
h2o.varimp_plot(GBM_1, 20)

Xnames <- names(htrain)[-which(names(htrain) %in% c('recommendations', 'permID'))]

GBM_1 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 50, max_depth = 10, learn_rate = 0.1,
                 nbins = 20)

h2o.performance(GBM_1, htest)
h2o.varimp_plot(GBM_1, 20)

GBM_1 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 50, max_depth = 10, learn_rate = 0.1,
                 nbins = 20)

h2o.performance(GBM_1, htest)
h2o.varimp_plot(GBM_1, 20)

# ntrees
GBM_1 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 10, learn_rate = 0.1, min_rows = 10,
                 nbins = 20, nbins_cats = 1024, sample_rate = 0.9, col_sample_rate = 1)

h2o.performance(GBM_1, htest)

GBM_1 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 250, max_depth = 9, learn_rate = 0.1, min_rows = 20,
                 nbins = 15, nbins_cats = 512, sample_rate = 1, col_sample_rate = 1)

h2o.performance(GBM_1, htest)

preds_gbm1 <- predict(GBM_1, htest_comments)
preds_gbm1 <- as.data.frame(preds_gbm1)

GBM_2 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 250, max_depth = 10, learn_rate = 0.1,
                 nbins = 20)

h2o.performance(GBM_2, htest)


GBM_2 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 500, max_depth = 10, learn_rate = 0.1,
                 nbins = 20)

h2o.performance(GBM_2, htest) # RMSE doesn't change!


# max_depth
GBM_2 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 15, learn_rate = 0.1,
                 nbins = 20)

h2o.performance(GBM_2, htest)

GBM_2 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 8, learn_rate = 0.1,
                 nbins = 20)

h2o.performance(GBM_2, htest)


GBM_2 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 9, learn_rate = 0.1,
                 nbins = 20)

h2o.performance(GBM_2, htest)


# learn_rate
GBM_3 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 9, learn_rate = 0.9,
                 nbins = 20)
```

```r
h2o.performance(GBM_3, htest) # 67.58965

GBM_3 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 9, learn_rate = 0.5,
                 nbins = 20)

h2o.performance(GBM_3, htest)

GBM_3 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 9, learn_rate = 0.2,
                 nbins = 20)

h2o.performance(GBM_3, htest)

GBM_3 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 9, learn_rate = 0.05,
                 nbins = 20)

h2o.performance(GBM_3, htest)

GBM_3 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 9, learn_rate = 0.07,
                 nbins = 20)

h2o.performance(GBM_3, htest)

GBM_3 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 9, learn_rate = 0.02,
                 nbins = 20)

h2o.performance(GBM_3, htest)


GBM_3 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 9, learn_rate = 0.06,
                 nbins = 20)

h2o.performance(GBM_3, htest)

GBM_3 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 9, learn_rate = 0.08,
                 nbins = 20)

h2o.performance(GBM_3, htest)

GBM_3 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 9, learn_rate = 0.09,
                 nbins = 20)

h2o.performance(GBM_3, htest)


## learning rate 0.08, ntrees 200, max depth 9 best so far
```

```r
# nbins
GBM_4 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 9, learn_rate = 0.08,
                 nbins = 10)

h2o.performance(GBM_4, htest)


GBM_4 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 9, learn_rate = 0.08,
                 nbins = 30)

h2o.performance(GBM_4, htest)

GBM_4 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 9, learn_rate = 0.08,
                 nbins = 15)

h2o.performance(GBM_4, htest)

GBM_4 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 9, learn_rate = 0.08,
                 nbins = 21)

h2o.performance(GBM_4, htest)

GBM_4 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 9, learn_rate = 0.08,
                 nbins = 19)

h2o.performance(GBM_4, htest)

# just stay at nbins = 20


GBM_5 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 9, learn_rate = 0.08,
                 nbins = 20, min_rows = 10)

h2o.performance(GBM_5, htest)


GBM_5 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 9, learn_rate = 0.08,
                 nbins = 20, min_rows = 100)

h2o.performance(GBM_5, htest)

GBM_5 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 9, learn_rate = 0.08,
                 nbins = 20, min_rows = 30)

h2o.performance(GBM_5, htest)
```

```r
GBM_5 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 9, learn_rate = 0.08,
                 nbins = 20, min_rows = 5)

h2o.performance(GBM_5, htest)

GBM_5 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 9, learn_rate = 0.08,
                 nbins = 20, min_rows = 3)

h2o.performance(GBM_5, htest) # good!

GBM_5 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 9, learn_rate = 0.08,
                 nbins = 20, min_rows = 1)

h2o.performance(GBM_5, htest)

GBM_5 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 9, learn_rate = 0.08,
                 nbins = 20, min_rows = 2)

h2o.performance(GBM_5, htest)


GBM_6 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = hdat,
                 ntrees = 200, max_depth = 9, learn_rate = 0.08,
                 nbins = 20, min_rows = 3)
preds_gbm6 <- predict(GBM_6, htest_comments)
preds_gbm6 <- as.data.frame(preds_gbm6)

Xnames <- names(htrain)[-which(names(htrain) %in% c('recommendations','source','documentType','permID'))

GBM_7 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = hdat,
                 ntrees = 200, max_depth = 9, learn_rate = 0.08,
                 nbins = 20, min_rows = 3)
preds_gbm7 <- predict(GBM_7, htest_comments)
preds_gbm7 <- as.data.frame(preds_gbm7)


GBM_8 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = hdat, # add genders, minus two var
                 ntrees = 200, max_depth = 9, learn_rate = 0.08,
                 nbins = 20, min_rows = 3)
preds_gbm8 <- predict(GBM_8, htest_comments)
preds_gbm8 <- as.data.frame(preds_gbm8)


GBM_9 <- h2o.gbm(x = Xnames, y = "recommendations", training_frame = hdat, # add genders, no minus, plu
                 ntrees = 200, max_depth = 9, learn_rate = 0.08,
                 nbins = 20, min_rows = 3)
preds_gbm9 <- predict(GBM_9, htest_comments)
preds_gbm9 <- as.data.frame(preds_gbm9)
```

```r
Xnames2 <- names(htrain)[-which(names(htrain) %in% c('recommendations','source','documentType','permID'

GBM_9 <- h2o.gbm(x = Xnames2, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 9, learn_rate = 0.05,
                 nbins = 20, min_rows = 3, sample_rate = 0.9)

h2o.performance(GBM_9, htest)
h2o.varimp(GBM_9)


Xnames3 <- names(htrain)[-which(names(htrain) %in% c('recommendations','source','documentType', 'parentI

GBM_10 <- h2o.gbm(x = Xnames3, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 9, learn_rate = 0.05,
                 nbins = 20, min_rows = 3, sample_rate = 0.9)

h2o.performance(GBM_10, htest)
h2o.varimp(GBM_10)


Xnames3 <- names(htrain)[-which(names(htrain) %in% c('recommendations','source','documentType', 'parentI

GBM_11 <- h2o.gbm(x = Xnames3, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 9, learn_rate = 0.05,
                 nbins = 20, min_rows = 3, sample_rate = 0.9)

h2o.performance(GBM_11, htest)
h2o.varimp(GBM_11)


Xnames4 <- names(htrain)[-which(names(htrain) %in% c('recommendations','source','documentType', 'articl


GBM_12 <- h2o.gbm(x = Xnames4, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 9, learn_rate = 0.08,
                 nbins = 20, min_rows = 3, sample_rate = 0.9)

h2o.performance(GBM_12, htest)
h2o.varimp(GBM_12)


Xnames5 <- names(htrain)[-which(names(htrain) %in% c('recommendations','source','documentType', 'parentI

GBM_13 <- h2o.gbm(x = Xnames5, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 9, learn_rate = 0.08,
                 nbins = 20, min_rows = 3, sample_rate = 0.9)

h2o.performance(GBM_13, htest)
h2o.varimp(GBM_13)
```

```
Xnames5 <- names(htrain)[-which(names(htrain) %in% c('recommendations','source','documentType', 'parentI

GBM_13 <- h2o.gbm(x = Xnames5, y = "recommendations", training_frame = htrain,
                  ntrees = 200, max_depth = 9, learn_rate = 0.08,
                  nbins = 20, min_rows = 3, sample_rate = 0.9)

h2o.performance(GBM_13, htest)
h2o.varimp(GBM_13)

Xnames5 <- names(htrain)[-which(names(htrain) %in% c('recommendations','source','documentType', 'parentU

GBM_13 <- h2o.gbm(x = Xnames5, y = "recommendations", training_frame = htrain,
                  ntrees = 200, max_depth = 9, learn_rate = 0.08,
                  nbins = 20, min_rows = 3, sample_rate = 0.9)

h2o.performance(GBM_13, htest)
h2o.varimp(GBM_13)


Xnames6 <- names(htrain)[-which(names(htrain) %in% c('recommendations','source','documentType', 'parentU


GBM_14 <- h2o.gbm(x = Xnames6, y = "recommendations", training_frame = htrain,
                  ntrees = 200, max_depth = 9, learn_rate = 0.05,
                  nbins = 20, min_rows = 3, sample_rate = 0.9)

h2o.performance(GBM_14, htest)
h2o.varimp(GBM_14)


Xnames7 <- names(htrain)[-which(names(htrain) %in% c('recommendations','source','documentType', 'parentI

GBM_15 <- h2o.gbm(x = Xnames7, y = "recommendations", training_frame = htrain,
                  ntrees = 200, max_depth = 9, learn_rate = 0.05,
                  nbins = 20, min_rows = 3, sample_rate = 0.9)

h2o.performance(GBM_15, htest)
h2o.varimp(GBM_15)


Xnames8 <- names(htrain)[-which(names(htrain) %in% c('recommendations','source','documentType', 'parentI

GBM_16 <- h2o.gbm(x = Xnames8, y = "recommendations", training_frame = htrain,
                  ntrees = 200, max_depth = 9, learn_rate = 0.05,
                  nbins = 20, min_rows = 3, sample_rate = 0.9)

h2o.performance(GBM_16, htest)
h2o.varimp(GBM_16)


Xnames9 <- names(htrain)[-which(names(htrain) %in% c('recommendations','source','documentType','depth')]

GBM_16 <- h2o.gbm(x = Xnames8, y = "recommendations", training_frame = htrain,
```

```r
                 ntrees = 300, max_depth = 9, learn_rate = 0.05,
                 nbins = 20, min_rows = 3, sample_rate = 0.9)

h2o.performance(GBM_16, htest)

# max depth
GBM_16 <- h2o.gbm(x = Xnames8, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 7, learn_rate = 0.05,
                 nbins = 20, min_rows = 3, sample_rate = 0.9)

h2o.performance(GBM_16, htest)

GBM_17 <- h2o.gbm(x = Xnames8, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 11, learn_rate = 0.05,
                 nbins = 20, min_rows = 3, sample_rate = 0.9)

h2o.performance(GBM_17, htest)


GBM_18 <- h2o.gbm(x = Xnames8, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 13, learn_rate = 0.05,
                 nbins = 20, min_rows = 3, sample_rate = 0.9)

h2o.performance(GBM_18, htest)

# learn rate
GBM_19 <- h2o.gbm(x = Xnames8, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 13, learn_rate = 0.1,
                 nbins = 20, min_rows = 3, sample_rate = 0.9)

h2o.performance(GBM_19, htest)

# sample rate
GBM_20 <- h2o.gbm(x = Xnames8, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 13, learn_rate = 0.05,
                 nbins = 20, min_rows = 3, sample_rate = 1)

h2o.performance(GBM_20, htest)


GBM_21 <- h2o.gbm(x = Xnames8, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 13, learn_rate = 0.05,
                 nbins = 20, min_rows = 3, sample_rate = 0.8)

h2o.performance(GBM_21, htest)

GBM_22 <- h2o.gbm(x = Xnames8, y = "recommendations", training_frame = htrain,
                 ntrees = 200, max_depth = 13, learn_rate = 0.05,
                 nbins = 20, min_rows = 3, sample_rate = 0.7)

h2o.performance(GBM_22, htest)

# min_rows
```

```r
GBM_23 <- h2o.gbm(x = Xnames8, y = "recommendations", training_frame = htrain,
                  ntrees = 200, max_depth = 13, learn_rate = 0.05,
                  nbins = 20, min_rows = 1, sample_rate = 0.8)

h2o.performance(GBM_23, htest)

GBM_24 <- h2o.gbm(x = Xnames8, y = "recommendations", training_frame = htrain,
                  ntrees = 200, max_depth = 13, learn_rate = 0.05,
                  nbins = 20, min_rows = 5, sample_rate = 0.8)

h2o.performance(GBM_24, htest)


GBM_25 <- h2o.gbm(x = Xnames8, y = "recommendations", training_frame = htrain,
                  ntrees = 200, max_depth = 13, learn_rate = 0.05,
                  nbins = 20, min_rows = 4, sample_rate = 0.8)

h2o.performance(GBM_25, htest)


GBM_26 <- h2o.gbm(x = Xnames8, y = "recommendations", training_frame = htrain,
                  ntrees = 200, max_depth = 13, learn_rate = 0.05,
                  nbins = 20, min_rows = 3, sample_rate = 0.8)

h2o.performance(GBM_26, htest) # 16.04835
h2o.varimp_plot(GBM_26, 20)


GBM_26 <- h2o.gbm(x = Xnames8, y = "recommendations", training_frame = hdat,
                  ntrees = 200, max_depth = 13, learn_rate = 0.05,
                  nbins = 20, min_rows = 3, sample_rate = 0.8)

preds_gbm26 <- predict(GBM_26, htest_comments)
preds_gbm26 <- as.data.frame(preds_gbm26)

Xnames <- Xnames8

# GBM grid search

hyper_params <- list(ntrees = 200,  ## early stopping
                     max_depth = 5:25,
                     min_rows = c(10,1:10,30, 100),
                     learn_rate = c(0.08,0.01,0.1, 0.5),
                     learn_rate_annealing = c(0.99,0.995,1,1),
                     sample_rate = c(seq(0.1, 0.9, .1), 1,1),
                     col_sample_rate = c(0.7,0.9, 1,1),
                     nbins = c(5,10, 20, 30,100),
                     nbins_cats = c(64,256,1024)
)

search_criteria <- list( strategy = "RandomDiscrete",
                        max_runtime_secs = 600000,
                        max_models = 2
)

HO1 <- h2o.grid(algorithm = "gbm", grid_id = "gg1",
                x = Xnames, y = "recommendations", training_frame = htrain,
                #validation_frame = htest,
                hyper_params = hyper_params,
                search_criteria = search_criteria,
                seed = 1)

mds_sort <- h2o.getGrid(grid_id = "gg1", sort_by = "mse", decreasing = TRUE)

md_best <- h2o.getModel(mds_sort@model_ids[[1]])

h2o.performance(md_best, htest)
preds_gbmgrid <- predict(md_best, htest_comments)
preds_gbmgrid <- as.data.frame(preds_gbmgrid)




##### RANDOM FOREST ######

# H2O Random Forest

RF_1 <- h2o.randomForest(x = Xnames8, y = "recommendations", training_frame = htrain,
                         ntrees = 50, max_depth = 20, min_rows = 1, nbins = 20,
                         sample_rate = 0.6320000291, col_sample_rate_per_tree = 1, mtries=-1,
                         seed = 1)

h2o.performance(RF_1, htest) # 16.55303
h2o.varimp(RF_1)
h2o.varimp_plot(RF_1, 20)

preds_rf1 <- predict(RF_1, htest_comments)
preds_rf1 <- as.data.frame(preds_rf1)


RF_2 <- h2o.randomForest(x = Xnames, y = "recommendations", training_frame = htrain,
                         ntrees = 50, max_depth = 10, min_rows = 1, nbins = 20,
                         sample_rate = 0.6320000291, col_sample_rate_per_tree = 1, mtries=-1,
                         seed = 1)

h2o.performance(RF_2, htest) # 64.92419


RF_2 <- h2o.randomForest(x = Xnames, y = "recommendations", training_frame = htrain,
                         ntrees = 50, max_depth = 25, min_rows = 1, nbins = 20,
                         sample_rate = 0.6320000291, col_sample_rate_per_tree = 1, mtries=-1,
                         seed = 1)

h2o.performance(RF_2, htest) # 64.92419

htrain_split <- h2o.splitFrame(htrain, ratios = 0.2)
```

```r
htrain_20 <- htrain_split[[1]]
RF_3 <- h2o.randomForest(x = Xnames3, y = "recommendations", training_frame = htrain_20,
                         ntrees = 500, max_depth = 18, min_rows = 1, nbins = 20,
                         sample_rate = 0.6320000291, col_sample_rate_per_tree = 1, mtries=-1,
                         seed = 1)

h2o.performance(RF_3, htest) # 16.36772
h2o.varimp(RF_1)


RF_3 <- h2o.randomForest(x = Xnames8, y = "recommendations", training_frame = hdat,
                         ntrees = 50, max_depth = 10, min_rows = 1, nbins = 20,
                         sample_rate = 0.6320000291, col_sample_rate_per_tree = 1, mtries=-1,
                         seed = 1, nfolds = 2, fold_assignment = "Modulo",
                         keep_cross_validation_predictions = TRUE)

preds_rf3 <- predict(RF_3, htest_comments)
preds_rf3 <- as.data.frame(preds_rf3)


# Random Forest grid search
hyper_params <- list(ntrees = seq(50,500, by = 25),
                     max_depth = c(1:30),
                     min_rows = c(1:30, 40, 50),
                     nbins = 2:30,
                     #mtries <- c(-1, 1:10),
                     sample_rate = c(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.6320000291, 0.7, 0.8, 0.9, 1),
                     col_sample_rate_per_tree = seq(0.05, 1, by = 0.05),
                     balance_classes = c(TRUE, FALSE)
)


search_criteria <- list( strategy = "RandomDiscrete",
                        max_runtime_secs = 600000,
                        max_models = 200
)


HO1 <- h2o.grid(algorithm = "randomForest", grid_id = "rfgrd",
                x = Xnames, y = "recommendations", training_frame = htrain,
                validation_frame = htest,
                hyper_params = hyper_params,
                search_criteria = search_criteria,
                stopping_tolerance = 1e-3, stopping_rounds = 2,
                seed = 1)

mds_sort <- h2o.getGrid(grid_id = "rfgrd", sort_by = "mse", decreasing = TRUE)

md_best <- h2o.getModel(mds_sort@model_ids[[1]])

h2o.auc(h2o.performance(md_best, hvalidation1)) #
h2o.auc(h2o.performance(md_best, htest1)) #
```

```r
# Caret Random Forest
set.seed(1)
N2 <- nrow(train)
train1 <- train[sample(1:N2, 2000),]

RF_10CV <- train(recommendations ~ commentWordCount + commentSentiment + trusted + depth + editorsSelect
                     data = train1,
                     method = "rf",
                     trControl = trainControl(method = "cv", number = 3, savePredictions = T), verbose=F

preds.rf <- predict(RF_10CV, test)
rmse(test$recommendations, preds.rf) # 88.71432



##### NEURAL NETWORKS ######


NN_1 <- h2o.deeplearning(x = Xnames, y = "recommendations", training_frame = htrain, hidden=c(20,20),
            epochs = 10, seed=1)

Xnamesnn <- c('replyCount', 'editorsSelection', 'bylineGender', 'gender', 'commentWordCount', 'commentS

NN_2 <- h2o.deeplearning(x = Xnamesnn, y = "recommendations", training_frame = hdat,
            epochs = 100, seed=1)
h2o.performance(NN_2, htest) # 18.23677

NN_3 <- h2o.deeplearning(x = Xnamesnn, y = "recommendations", training_frame = hdat,
            epochs = 20, seed=1)




##### GLM & OTHER MODELS ######

### H2O GLM
GLM_1 <- h2o.glm(x = Xnames, y = "recommendations", training_frame = htrain, lambda_search=T, family="p
h2o.performance(GLM_1, htest)

GLM_2 <- h2o.glm(x = Xnames8, y = "recommendations", training_frame = htrain)
h2o.performance(GLM_2, htest) # 22.56361

GLM_3 <- h2o.glm(x = Xnames8, y = "recommendations", training_frame = hdat, lambda_search = T)
h2o.performance(GLM_3, htest)


### Caret GLM
lim_train <- train_dat[,-which(names(train_dat) %in% colnames(train_dat)[colSums(is.na(train_dat)) > 0]
ltn <- nrow(lim_train)
ltt <- lim_train[sample(1:ltn, 50000),]

Xltrain <- ltt[,-which(names(ltt) == 'recommendations')]
XltrainPP <- preProcess(Xltrain, method=c("BoxCox", "center", "scale"))
```

```r
XltrainTrans <- predict(XltrainPP, Xltrain)

GLM_10CV <- train(XltrainTrans, ltt$recommendations,
                  method = "glm",
                  trControl = trainControl(method = "cv", number=5))

preds.glm <- predict(GLM_10CV, test)
rmse(test$recommendations, preds.glm)


### Other GLM Models
train1 <- train_comments[sample(1:nrow(train),5000),]
train1$recommendations <- as.factor(train1$recommendations)
train1$status <- NULL

fit.qp <- glm(recommendations ~ commentWordCount + commentSentiment, data = train, family = "quasipoiss
preds.qp <- predict(fit.qp, test)
rmse(test$recommendations, preds.qp) # 88.71432

fit.qp <- glm(recommendations ~ articleWordCount + commentType + depth + editorsSelection + inReplyTo +
preds.qp <- predict(fit.qp, test)
rmse(test$recommendations, preds.qp) # 88.57423

fit.nb <- glm.nb(recommendations ~ articleWordCount + commentType + depth + editorsSelection + inReplyT
preds.nb <- predict(fit.nb, test)
rmse(test$recommendations, preds.nb) # 88.38405

fit.hurdle.nb <- hurdle(recommendations ~ articleWordCount + commentType + depth + editorsSelection + i
preds.hurdle.nb <- predict(fit.hurdle.nb, test, type="response")
rmse(test$recommendations, preds.hurdle.nb)




#############################################
# Model Submission
#############################################


newsub <- cbind("commentID" = test_comments$commentID, "pred_recs" = preds_gbm26$predict)
write.csv(newsub, '~/Desktop/newsubgbm26.csv')

sub1 <- read_csv("~/Desktop/newsubgbm26.csv")
sub1$X1 <- NULL

sub1 %<>% mutate(commentID = as.double(commentID))
write_csv(sub1,'~/Desktop/newsubgbm26-s.csv')
```