Case Study 2

Sahil Jain, Jacques Sham, Mina Chan, Charles Sui

April 6, 2018

Loading Libraries

```
library(tree)
library(tidyverse)
```

```
## ── Attaching packages ────────
```

```
## ✔ ggplot2 2.2.1     ✔ purrr   0.2.4
## ✔ tibble  1.4.2     ✔ dplyr   0.7.4
## ✔ tidyr   0.8.0     ✔ stringr 1.3.0
## ✔ readr   1.1.1     ✔ forcats 0.3.0
```

```
## Warning: package 'ggplot2' was built under R version 3.3.2
```

```
## Warning: package 'readr' was built under R version 3.3.2
```

```
## Warning: package 'purrr' was built under R version 3.3.2
```

```
## Warning: package 'dplyr' was built under R version 3.3.2
```

```
## ── Conflicts ─────────────────
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
```

```
library(magrittr)
```

```
##
## Attaching package: 'magrittr'
```

```
## The following object is masked from 'package:purrr':
##
##     set_names
```

```
## The following object is masked from 'package:tidyr':
##
##     extract
```

```r
library(ISLR)
```

```
## Warning: package 'ISLR' was built under R version 3.3.2
```

```r
library(randomForest)
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
library(MASS)
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```r
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.3.2
```

```
## Loading required package: lattice
```

```
## Warning: package 'lattice' was built under R version 3.3.2
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
library(survival)
```

```
## Warning: package 'survival' was built under R version 3.3.2
```

```
##
## Attaching package: 'survival'
```

```
## The following object is masked from 'package:caret':
##
##     cluster
```

```r
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 3.3.2
```

```
## Loading required package: splines
```

```
## Loading required package: parallel
```

```
## Loaded gbm 2.1.3
```

```r
library(TSA)
```

```
## Loading required package: leaps
```

```
## Warning: package 'leaps' was built under R version 3.3.2
```

```
## Loading required package: locfit
```

```
## locfit 1.5-9.1    2013-03-22
```

```
## Loading required package: mgcv
```

```
## Warning: package 'mgcv' was built under R version 3.3.2
```

```
## Loading required package: nlme
```

```
## Warning: package 'nlme' was built under R version 3.3.2
```

```
##
## Attaching package: 'nlme'
```

```
## The following object is masked from 'package:dplyr':
##
##     collapse
```

```
## This is mgcv 1.8-22. For overview type 'help("mgcv-package")'.
```

```
## Loading required package: tseries
```

```
## Warning: package 'tseries' was built under R version 3.3.2
```

```
##
## Attaching package: 'TSA'
```

```
## The following object is masked from 'package:readr':
##
##     spec
```

```
## The following objects are masked from 'package:stats':
##
##     acf, arima
```

```
## The following object is masked from 'package:utils':
##
##     tar
```

```
library(ipred)
```

```
## Warning: package 'ipred' was built under R version 3.3.2
```

```
library(rpart)
library(TH.data)
```

```
## Warning: package 'TH.data' was built under R version 3.3.2
```

```
##
## Attaching package: 'TH.data'
```

```
## The following object is masked from 'package:MASS':
##
##     geyser
```

Loading training data

```
train <- read.csv("/Users/sahiljain/Desktop/Spring 2018/Statistical Learning/OnlineNe
wsPopularityTraining.csv")
test <- read.csv("/Users/sahiljain/Desktop/Spring 2018/Statistical Learning/OnlineNew
sPopularityTest.csv")
```

Omitting NA's

```
dataTrain <- na.omit(train)
dataTest <- na.omit(test)
```

Omitting extra variables

```
NewData_train <- dataTrain[,-c(1,2,61)]
NewData_test <- dataTest[,-c(1,2,61)]
```

Classification data variable.

```
train_tree <- NewData_train
test_tree <- NewData_test
```

```
train_tree$popular <- factor(train_tree$popular)
test_tree$popular <- factor(test_tree$popular)

data.train.class <- train_tree$popular
data.test.class <- test_tree$popular

set.seed(1)
data.train.upsample <- upSample(train_tree, data.train.class, T)$x
```

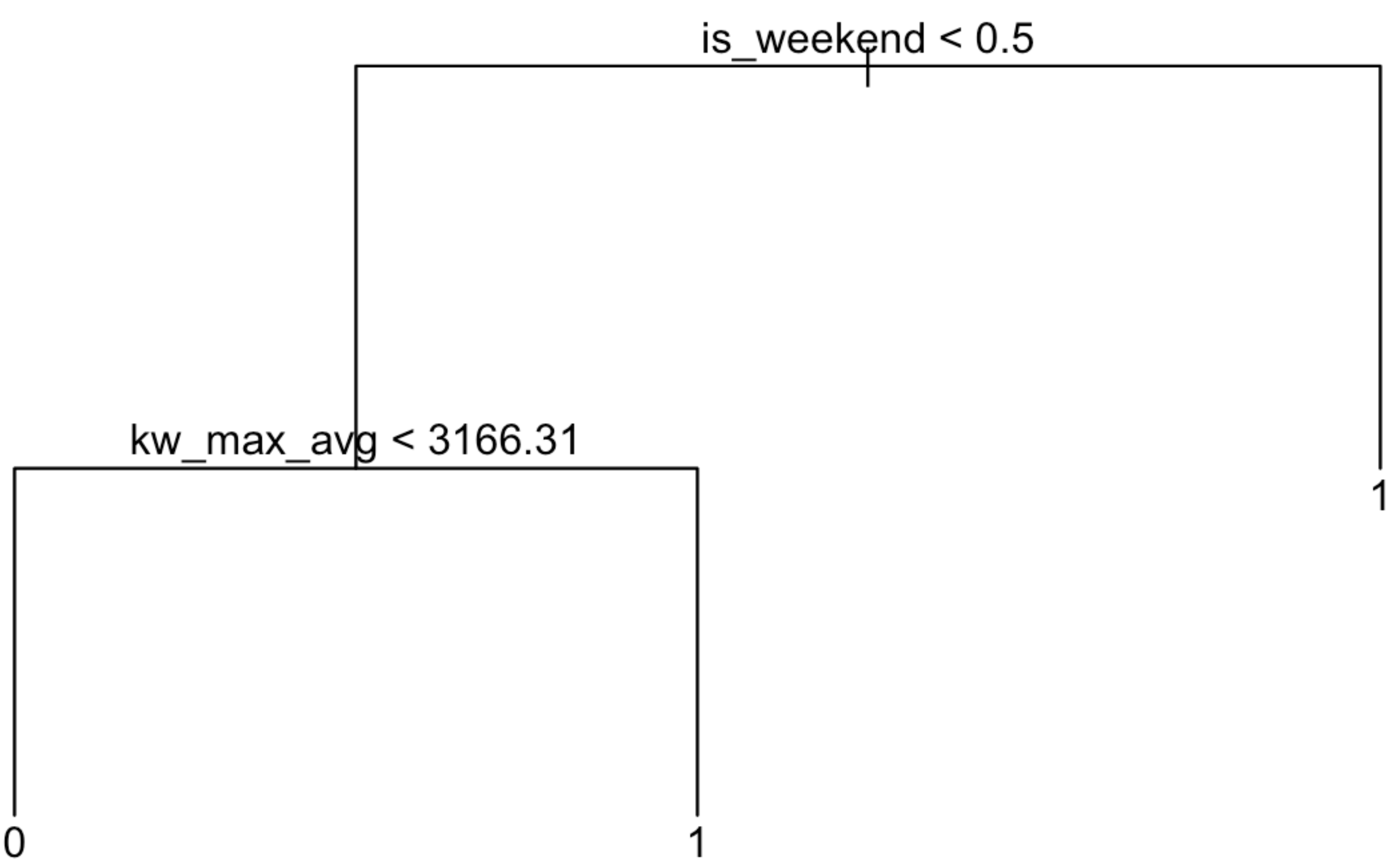
A. Build classifiers for this data set using the tree based methods that we've learned in class. In particular, build classifiers on the training data and assess the performance of each of the following methods on the test data (available on Canvas)

B. Classification tree

```
tree.train <- tree(popular ~ ., data = data.train.upsample)
summary(tree.train)
```

```
##
## Classification tree:
## tree(formula = popular ~ ., data = data.train.upsample)
## Variables actually used in tree construction:
## [1] "is_weekend" "kw_max_avg"
## Number of terminal nodes:  3
## Residual mean deviance:  1.341 = 8607 / 6417
## Misclassification error rate: 0.4265 = 2738 / 6420
```

```
plot(tree.train)
text(tree.train, pretty = 0)
```



Error Classification

```
tree.pred <- predict(tree.train, NewData_train, type = "class")
table(tree.pred, data.train.class)
```

```
##          data.train.class
## tree.pred    0    1
##         0  954  124
##         1 2256  699
```

```
mean(tree.pred == data.train.class)
```

```
## [1] 0.4098686
```

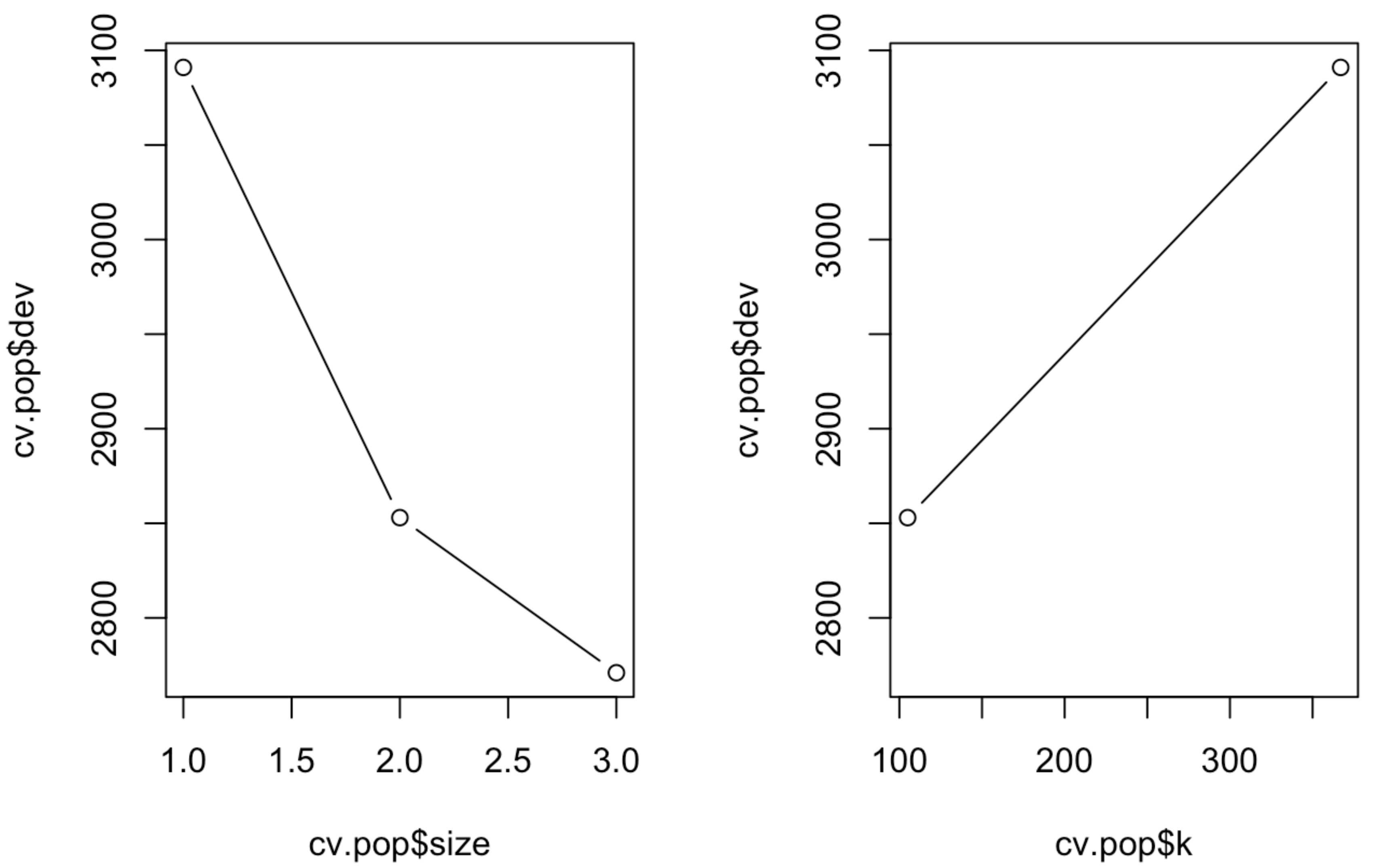
Cross Validation

```
cv.pop <- cv.tree(tree.train, FUN = prune.misclass)
cv.pop
```

```
## $size
## [1] 3 2 1
##
## $dev
## [1] 2771 2853 3091
##
## $k
## [1] -Inf  105  367
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

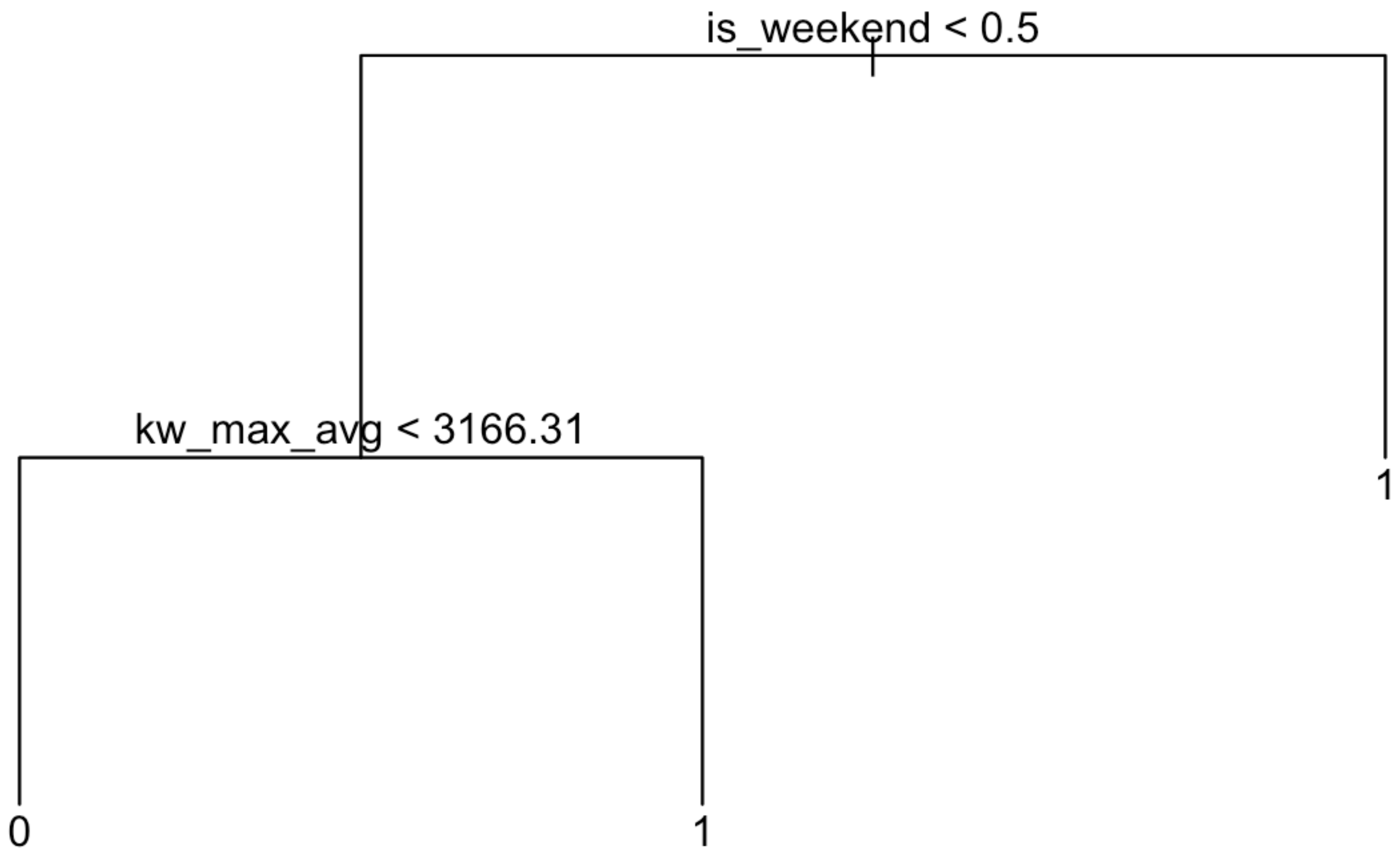
Plot

```
par(mfrow = c(1, 2))
plot(cv.pop$size, cv.pop$dev, type = "b")
plot(cv.pop$k, cv.pop$dev, type = "b")
```



Pruning the tree

```
par(mfrow = c(1,1))
prune.pop <- prune.misclass(tree.train, best = cv.pop$size[which.min(cv.pop$dev)])
plot(prune.pop)
text(prune.pop, pretty = 0)
```



Testing our model on held out test set

```
prune.pred <- predict(prune.pop, NewData_test, type = "class")
table(prune.pred, data.test.class)
```

```
##            data.test.class
## prune.pred    0    1
##           0  208   35
##           1 6077 1608
```

```
mean(prune.pred == data.test.class)
```

```
## [1] 0.2290616
```

So from tree classification we can see that our MSPE for training set was .40 where as our MSPE for our held out test set was .229 which is slightly better than training set. From this tree we can see that variables weekend and Kw_max_avg are very important variables.

II. Bagging

```
gbag <- bagging(popular ~., data = NewData_train, coob = TRUE)
print(gbag)
```

```
##
## Bagging regression trees with 25 bootstrap replications
##
## Call: bagging.data.frame(formula = popular ~ ., data = NewData_train,
##     coob = TRUE)
##
## Out-of-bag estimate of root mean squared error:  0.3966
```

Prediction on training set and measuring MSPE

```
yhat <- predict(gbag, newdata = NewData_train)
bagging_train_MSPE <- sqrt(mean((yhat - NewData_train$popular)^2))
bagging_train_MSPE
```

```
## [1] 0.3930793
```

Here, from our training set out MSPE is 0.3930 which is quite small, now we will see our MSPE and prediction on the held out test set.

Prediction and MSPE on held out test set

```
gbag_test <- bagging(popular ~., data = NewData_test, coob = TRUE)
print(gbag_test)
```

```
##
## Bagging regression trees with 25 bootstrap replications
##
## Call: bagging.data.frame(formula = popular ~ ., data = NewData_test,
##     coob = TRUE)
##
## Out-of-bag estimate of root mean squared error:  0.3952
```

```
yhat_test <- predict(gbag_test, newdata = NewData_test)
bagging_test_MPSE <- sqrt(mean((yhat_test - NewData_test$popular)^2))
bagging_test_MPSE
```

```
## [1] 0.3938981
```

We can see that out MSPE from our held out test set is 0.3938, which is slightly higher than our MSPE on training set.

III. Random Forest

We will be running our Random Forest over 1000 trees and all 58 predictors.

```
set.seed(1)
bag.popular <- randomForest(popular ~., data = NewData_train, mtry = 58, importance =
TRUE, ntrees = 1000)
```

```
## Warning in randomForest.default(m, y, ...): The response has five or fewer
## unique values. Are you sure you want to do regression?
```

```
bag.popular
```

```
##
## Call:
##  randomForest(formula = popular ~ ., data = NewData_train, mtry = 58,      importa
nce = TRUE, ntrees = 1000)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 58
##
##           Mean of squared residuals: 0.156999
##                     % Var explained: 3.34
```

Prediction on Training set and measuring MSPE

```
yhat.bag <- predict(bag.popular, newdata = NewData_train)
sqrt(mean((yhat.bag - NewData_train$popular)^2))
```

```
## [1] 0.1650788
```

As we can see our MSPE is incredibly small close to 0.16 predictors. Now we will investigate the performance on a held out test set.

```
set.seed(1)
rf.popular <- randomForest(popular ~ ., data = NewData_test, mtry = 58, importance =
TRUE)
```

```
## Warning in randomForest.default(m, y, ...): The response has five or fewer
## unique values. Are you sure you want to do regression?
```

```
yhat.rf <- predict(rf.popular, newdata = NewData_test)
sqrt(mean((yhat.rf - NewData_test$popular)^2))
```

```
## [1] 0.1625907
```

From our held out test set our MSPE turns out to be 0.1625 which in slightly less than our training set

We will now see the imprtance of each variable in the random forest.

Calculating the importance of each predictor

```
importance(rf.popular)
```

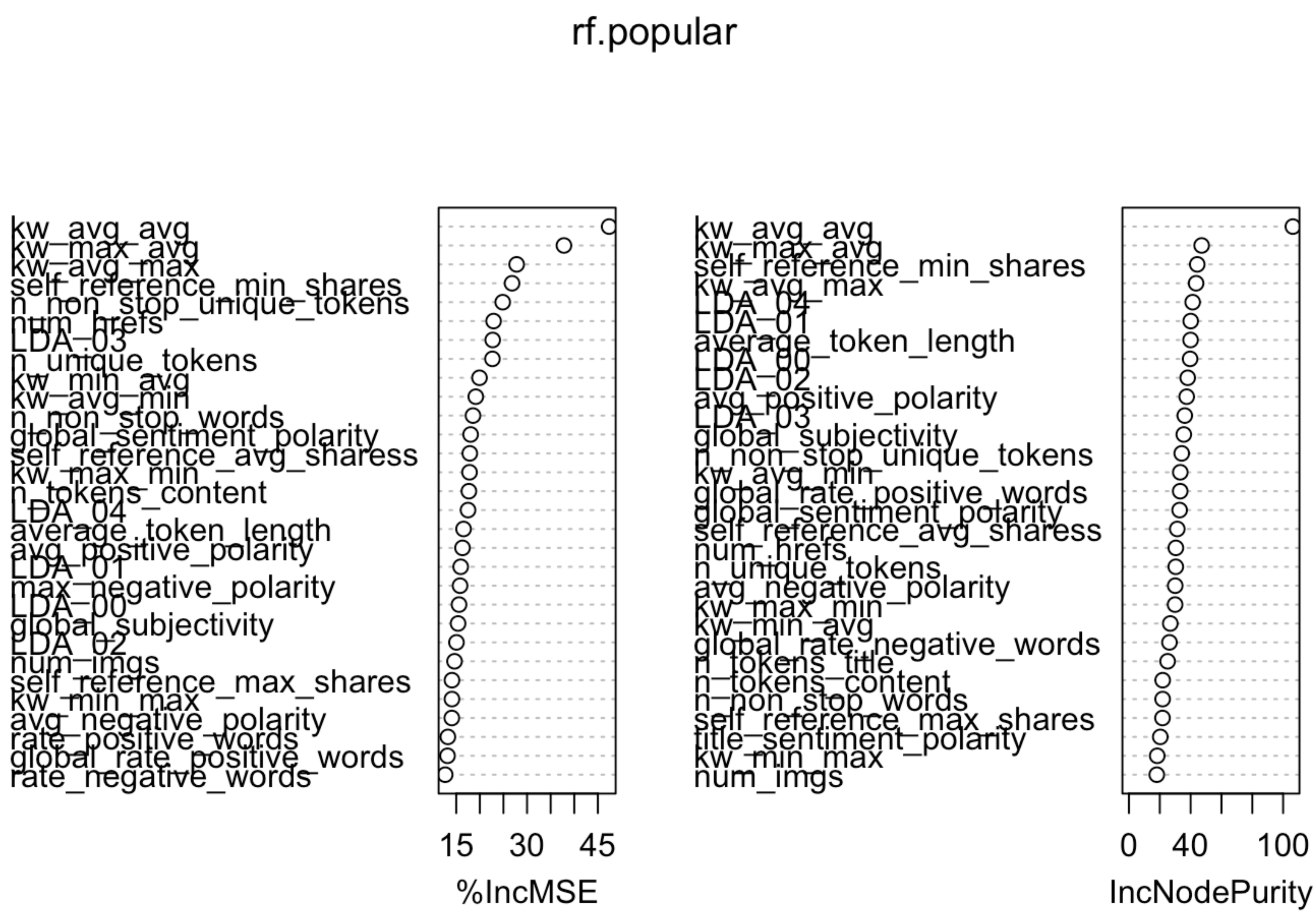
```
##                                  %IncMSE IncNodePurity
## n_tokens_title                 0.8602147     25.060645
## n_tokens_content              17.6770170     21.873313
## n_unique_tokens               22.6681528     30.352980
## n_non_stop_words              18.5452335     21.842255
## n_non_stop_unique_tokens      24.8469445     34.210009
## num_hrefs                     22.9088033     30.412205
## num_self_hrefs                 9.3187189     15.995567
## num_imgs                      14.6399749     18.056112
## num_videos                     8.8891014     10.634998
## average_token_length          16.5495374     39.955673
## num_keywords                   8.0228724      7.271354
## data_channel_is_lifestyle      5.0993747      2.563636
## data_channel_is_entertainment  5.2590186      2.008618
## data_channel_is_bus            5.8688217      1.539678
## data_channel_is_socmed         7.5375647      6.233736
## data_channel_is_tech           9.2848269      2.919197
## data_channel_is_world          7.1469859      1.758242
## kw_min_min                     8.7622271      4.396615
## kw_max_min                    17.8151328     29.909189
## kw_avg_min                    19.1386501     33.287439
## kw_min_max                    14.0940422     18.331017
## kw_max_max                    11.0985990      5.419983
## kw_avg_max                    27.7976037     43.548185
## kw_min_avg                    19.9228339     26.803473
## kw_max_avg                    37.8178077     47.149199
## kw_avg_avg                    47.3856446    106.325425
## self_reference_min_shares     26.8227033     44.293074
## self_reference_max_shares     14.0972570     21.757521
## self_reference_avg_sharess    17.8708453     31.330161
## weekday_is_monday             -0.6944324      2.981402
## weekday_is_tuesday            -1.3034491      2.837922
## weekday_is_wednesday          -2.5658597      3.011737
## weekday_is_thursday           -0.8668697      2.727200
## weekday_is_friday              0.4810165      3.240898
## weekday_is_saturday           12.4714334      5.598914
## weekday_is_sunday              3.3245842      2.680126
## is_weekend                     4.5108671      3.649530
## LDA_00                        15.5961781     39.643708
## LDA_01                        15.9418963     40.050983
## LDA_02                        15.0514284     38.097341
## LDA_03                        22.7227272     36.185016
```

```
## LDA_04                          17.4939095     41.315558
## global_subjectivity             15.3908404     35.561345
## global_sentiment_polarity       18.0440100     32.858338
## global_rate_positive_words      13.1530110     33.226666
## global_rate_negative_words      12.6276067     26.269466
## rate_positive_words             13.2300815     15.236322
## rate_negative_words             12.6632708     14.930929
## avg_positive_polarity           16.3296837     37.442335
## min_positive_polarity            8.9597893     12.153438
## max_positive_polarity            9.0881521      9.232422
## avg_negative_polarity           14.0646178     29.996031
## min_negative_polarity            8.8988700     13.048846
## max_negative_polarity           15.8111606     16.902372
## title_subjectivity               6.0342665     12.967628
## title_sentiment_polarity        12.2812183     20.323047
## abs_title_subjectivity           2.5834539     12.889704
## abs_title_sentiment_polarity     5.6473830     11.618776
```

Too much information, now we will plot the importance of each variable.

Plotting the importance measures of each variable.

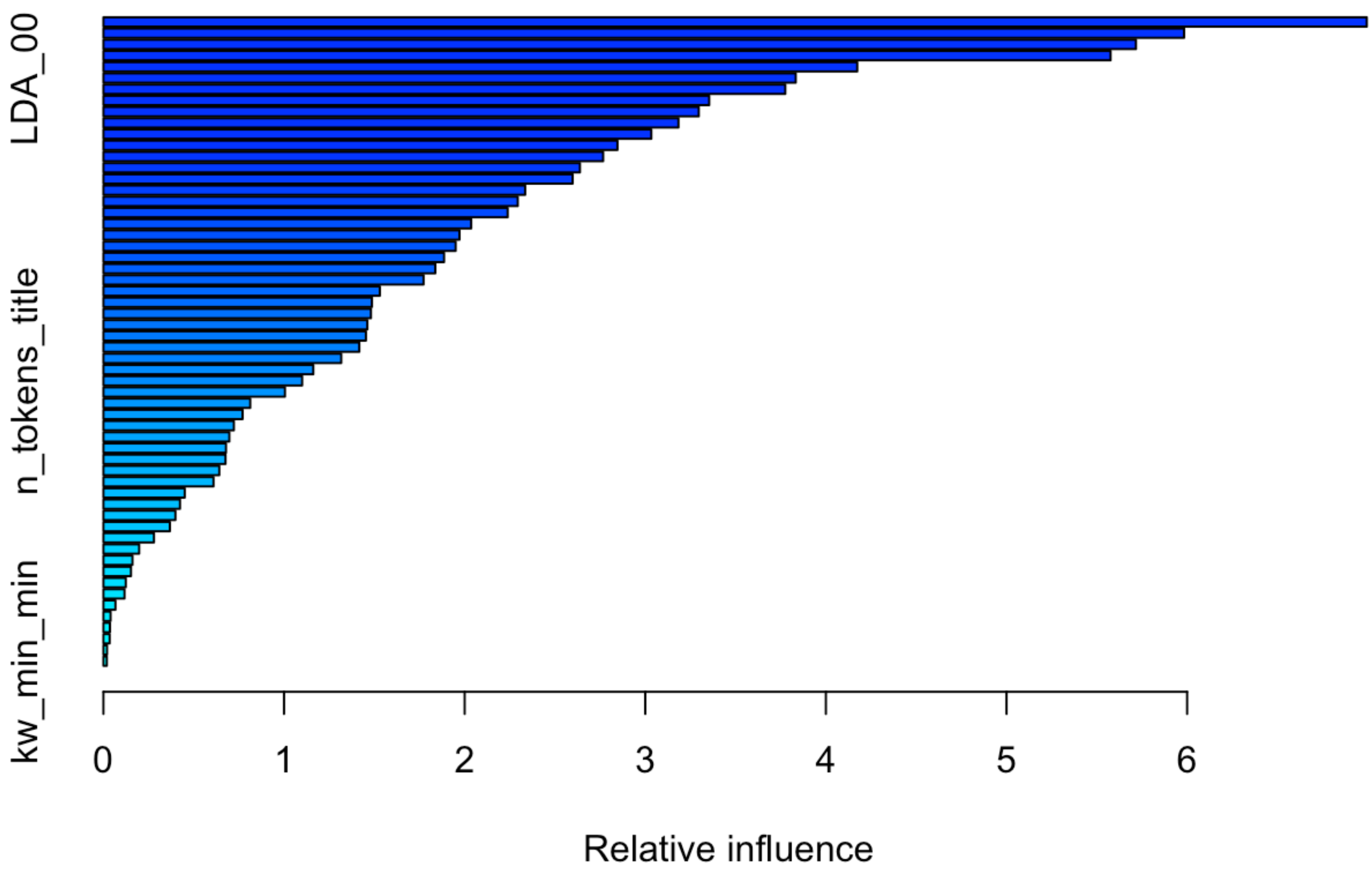
```
varImpPlot(rf.popular)
```



From the above result we can clearly see that variable kw_avg_avg is by far the most important variable in the random forest.

IV. Boosting - Boosting is very similar to randomForests

```
boost <- gbm(popular ~., data = NewData_train, distribution = 'gaussian', n.trees = 5
000, interaction.depth = 4)
summary(boost)
```

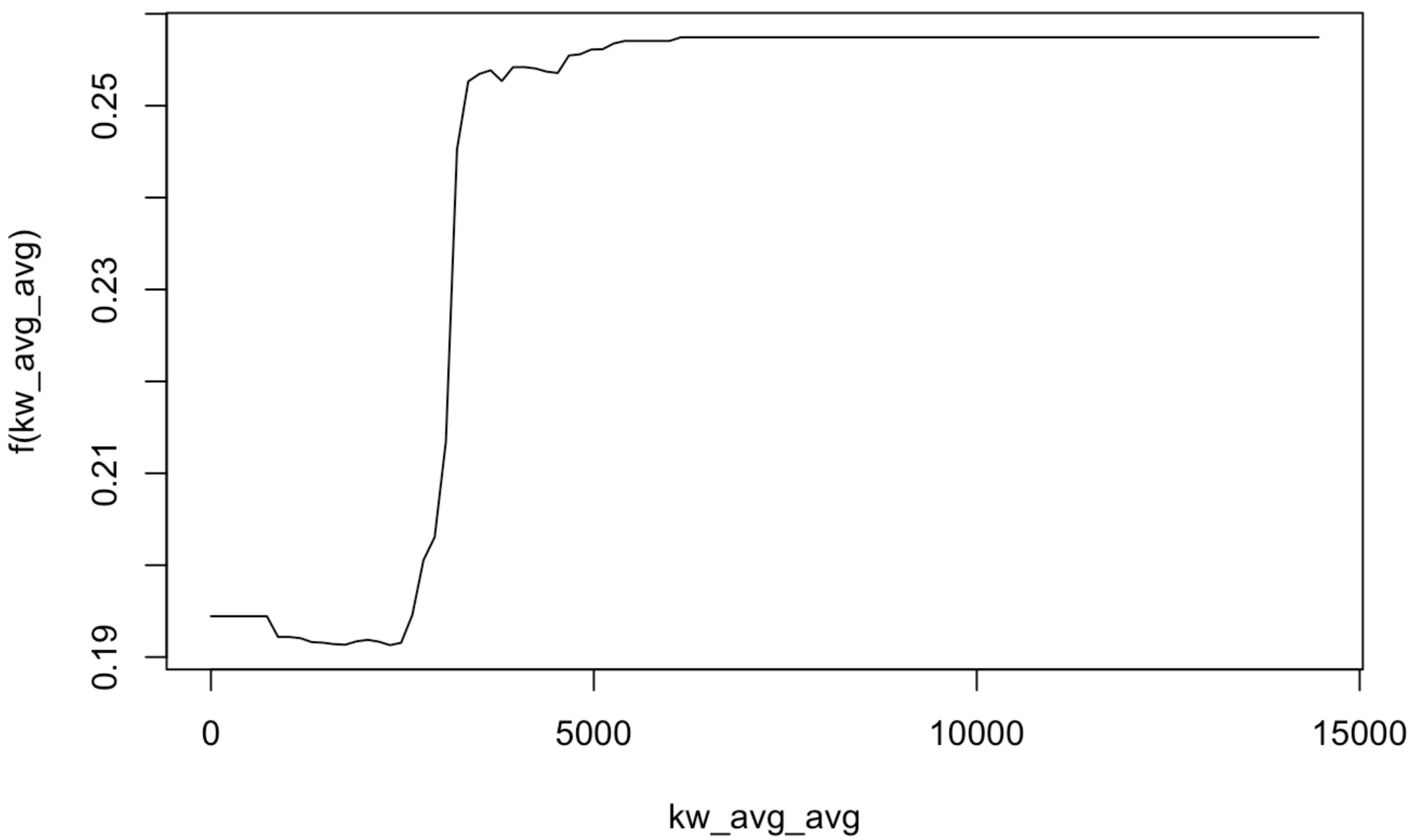



```
##                                                      var    rel.inf
## self_reference_min_shares          self_reference_min_shares 6.99466465
## kw_avg_avg                                        kw_avg_avg 5.98329282
## is_weekend                                        is_weekend 5.71670473
## kw_max_avg                                        kw_max_avg 5.57614629
## n_non_stop_unique_tokens            n_non_stop_unique_tokens 4.17363824
## LDA_00                                                LDA_00 3.83309395
## global_subjectivity                      global_subjectivity 3.77517903
## data_channel_is_socmed                data_channel_is_socmed 3.35393684
## avg_positive_polarity                  avg_positive_polarity 3.29622639
## avg_negative_polarity                  avg_negative_polarity 3.18382786
## global_rate_positive_words        global_rate_positive_words 3.03333063
## self_reference_avg_sharess        self_reference_avg_sharess 2.84602086
```

```
## n_tokens_content                                      n_tokens_content 2.76687587
## kw_min_avg                                                  kw_min_avg 2.63834135
## n_unique_tokens                                        n_unique_tokens 2.59781446
## kw_avg_max                                                  kw_avg_max 2.33501235
## average_token_length                              average_token_length 2.29412413
## weekday_is_saturday                                weekday_is_saturday 2.23814500
## LDA_03                                                          LDA_03 2.03606133
## kw_avg_min                                                  kw_avg_min 1.97126673
## LDA_01                                                          LDA_01 1.95017831
## LDA_02                                                          LDA_02 1.88610051
## n_non_stop_words                                      n_non_stop_words 1.83751239
## num_hrefs                                                    num_hrefs 1.77248561
## self_reference_max_shares                    self_reference_max_shares 1.53032484
## global_rate_negative_words                  global_rate_negative_words 1.48663648
## global_sentiment_polarity                    global_sentiment_polarity 1.48116860
## data_channel_is_tech                              data_channel_is_tech 1.46116194
## min_positive_polarity                            min_positive_polarity 1.45346115
## LDA_04                                                          LDA_04 1.41654305
## kw_min_max                                                  kw_min_max 1.31671379
## kw_max_min                                                  kw_max_min 1.16193347
## n_tokens_title                                          n_tokens_title 1.10028766
## title_sentiment_polarity                      title_sentiment_polarity 1.00504796
## num_self_hrefs                                          num_self_hrefs 0.81284935
## data_channel_is_entertainment data_channel_is_entertainment 0.77049314
## abs_title_sentiment_polarity   abs_title_sentiment_polarity 0.72235751
## rate_negative_words                                rate_negative_words 0.69718044
## abs_title_subjectivity                          abs_title_subjectivity 0.67865153
## max_negative_polarity                            max_negative_polarity 0.67602449
## min_negative_polarity                            min_negative_polarity 0.64157422
## rate_positive_words                                rate_positive_words 0.60949129
## max_positive_polarity                            max_positive_polarity 0.45027341
## title_subjectivity                                  title_subjectivity 0.42391633
## num_imgs                                                      num_imgs 0.39859165
## data_channel_is_bus                                data_channel_is_bus 0.36790641
## num_videos                                                  num_videos 0.27979399
## kw_max_max                                                  kw_max_max 0.19786394
## weekday_is_friday                                    weekday_is_friday 0.16094183
## num_keywords                                              num_keywords 0.15251847
## weekday_is_sunday                                    weekday_is_sunday 0.12381440
## data_channel_is_lifestyle                    data_channel_is_lifestyle 0.11677226
## weekday_is_wednesday                              weekday_is_wednesday 0.06662694
## data_channel_is_world                            data_channel_is_world 0.03964054
## weekday_is_thursday                                weekday_is_thursday 0.03612448
## weekday_is_tuesday                                  weekday_is_tuesday 0.03458143
## weekday_is_monday                                    weekday_is_monday 0.02003027
## kw_min_min                                                  kw_min_min 0.01872240
```

Clearly, we can see that kw_avg_avg is by far the most important variable. Now we can look at the plot of this variable.

```
plot(boost, i = 'kw_avg_avg')
```



Now we will predict our training set and measure training MSPE

```
boost.train <- predict(boost, newdata = NewData_train, n.trees = 5000)
sqrt(mean((boost.train - NewData_train$popular)^2))
```

```
## [1] 0.3714702
```

Using boosting we can clearly see that our MSPE is incredibly small and close to 0.37 predictors. Now we will investigate the performance on a held out test set.

Appling our boosted model to predit the test set

```
boost.pred <- predict(boost, NewData_test, n.trees = 5000)
sqrt(mean((boost.pred - NewData_test$popular)^2))
```

```
## [1] 0.3928828
```

From our held out test set our MSPE turns out to be 0.393 which in slightly higher than our training set

B. What is the MSPE for each of your fitted models? Compare and contrast between models and these ensemble-based models with the classification models that you fit in Homework 3. What are the advantages and Disadvantages of using these ensemble methods ?

MSPE for each fitted models for test and training set is :

Classification Tree : Training MSPE : .40 Test MSPE : .229

Bagging : Training MSPE : 0.3918575 Test MSPE : 0.3930174

Random Forests : Training MSPE : 0.1650788 Test MSPE : 0.1625907

Boosting : Training MSPE : 0.3714702 Test MSPE : 0.3928828

These ensemble methods are somewhat similar to methods like K-NN, LDA, QDA, Logistic regression and Naive Bayes Classifier. Just like these methods, ensemble based methods also take MSPE into account to predict the popularity of the website, however ensemble methods does not gives the accuracy of the model but only MSPE where as classification models gives us solid accuracy scores which tells a whole lot of story about the data. Advantages of these ensemle methods is this that it clearly tells the most important variable that is going to affect the popularity of a particular website which helps in only focusing on few important variables. However one of the biggest drawbacks of these ensemble based methods is this that they wont tell how accurate these models are. For instance, K-NN will tell about how accurate our model is in predicting the popularity of the website where as a classification tree will only look at important varibales and make decisions based solely on those facts.