trees

Mengqing Zhang

4/26/2020

Preparation

```
bosList <- read_csv("bosList.csv")
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   neighbourhood_cleansed = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

```
bjList <- read_csv("bjList.csv")
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   neighbourhood = col_character()
## )
## See spec(...) for full column specifications.
```

```
names(bosList)[33:42] <- c("host_resp_wt_a_day","host_resp_wt_a_few_hrs","host_resp
_wt_an_hr",
  "bed_type_Couch","bed_type_Futon","bed_type_Sofa","bed_type_Bed",
"room_type_Hotel_room","room_type_Private_room","room_type_Shared_room" )

names(bjList)[33:41] <- c("host_resp_wt_a_day","host_resp_wt_a_few_hrs","host_resp_
wt_an_hr",
  "bed_type_Couch","bed_type_Futon",
  "bed_type_Sofa","bed_type_Bed",
  "room_type_Private_room",
  "room_type_Shared_room" )
```

```
#Remove the first column X
bosList<- bosList[,-1]
bjList <- bjList[,-1]

bosList <- bosList %>% dplyr::select(host_listings_count,

accommodates,bathrooms,bedrooms,beds,cleaning_fee,
                                     guests_included,extra_people,

maximum_nights,availability_30,availability_90,
                                     availability_365,number_of_reviews,
                                     number_of_reviews_ltm,wifi_available,
                                     host_response_time_nodata,

host_resp_wt_a_few_hrs,cancellation_policy_strict,
                                     price)

bjList <- bjList %>% dplyr::select(host_listings_count,

accommodates,bathrooms,bedrooms,beds,minimum_nights,
                                   guests_included,
                                   availability_30,availability_90,
                                   availability_365,number_of_reviews,
                                   number_of_reviews_ltm,TV_available,
                                   wc_access,room_type_Shared_room,price
)

set.seed(68)
# This will split into train and test 75-25
bosList$train <- sample(c(0, 1), nrow(bosList), replace = TRUE, prob = c(.25, .75))
boslist_test <- bosList %>% filter(train == 0)%>% mutate_if(is.character, as.factor
)
boslist_train <- bosList %>% filter(train == 1)%>% mutate_if(is.character, as.facto
r)

bjList$train <- sample(c(0, 1), nrow(bjList), replace = TRUE, prob = c(.25, .75))
bjList_test <- bjList %>% filter(train == 0)%>% mutate_if(is.character, as.factor)
bjList_train <- bjList %>% filter(train == 1)%>% mutate_if(is.character, as.factor)

# #delete the neighborhood column
# boslist_train <- boslist_train[,-4]
# boslist_test <- boslist_test[,-4]
#
# bjList_train <- bjList_train[,-4]
# bjList_test <-  bjList_test[,-4]

#delete the last train column(0,1)
boslist_train <- boslist_train[,-ncol(boslist_train)]
boslist_test <- boslist_test[,-ncol(boslist_test)]

bjList_train <- bjList_train[,-ncol(bjList_train)]
```

```
bjList_test <-  bjList_test[,-ncol(bjList_test)]
```

Regression Tree–Boston

```
set.seed(68)
#Regression tree
fit.tree <- rpart(price~.,
                  boslist_train,
                  control = rpart.control(cp = 0.0001))
par(xpd = TRUE)

## Printcp will tell you what the cp of spliting into diffrent number layer and the
xerror and xstd of each cp.
printcp(fit.tree)
```

```
##
## Regression tree:
## rpart(formula = price ~ ., data = boslist_train, control = rpart.control(cp = 1e
-04))
##
## Variables actually used in tree construction:
##  [1] accommodates          availability_30       availability_365
##  [4] availability_90       bathrooms             bedrooms
##  [7] beds                  cleaning_fee          extra_people
## [10] guests_included       host_listings_count   maximum_nights
## [13] number_of_reviews     number_of_reviews_ltm
##
## Root node error: 261549359/2561 = 102128
##
## n= 2561
##
##            CP nsplit rel error xerror    xstd
## 1  0.04740875      0   1.00000 1.0006 0.52455
## 2  0.01637646      4   0.81037 1.0979 0.53254
## 3  0.01017419      5   0.79399 1.1370 0.53426
## 4  0.00865433      6   0.78381 1.1339 0.53426
## 5  0.00678101      7   0.77516 1.1282 0.53420
## 6  0.00480928      8   0.76838 1.1260 0.53629
## 7  0.00443587     11   0.75395 1.1268 0.53629
## 8  0.00235605     12   0.74952 1.1216 0.53627
## 9  0.00215684     14   0.74480 1.1153 0.53619
## 10 0.00208725     16   0.74049 1.1149 0.53619
## 11 0.00177847     19   0.73423 1.1100 0.53619
## 12 0.00152303     20   0.73245 1.1087 0.53618
## 13 0.00150011     21   0.73093 1.1081 0.53618
## 14 0.00133942     22   0.72943 1.1085 0.53618
## 15 0.00130661     23   0.72809 1.1085 0.53618
## 16 0.00096766     24   0.72678 1.1075 0.53618
```

```
## 17 0.00073091     25   0.72581 1.0986 0.53589
## 18 0.00068466     26   0.72508 1.0990 0.53589
## 19 0.00067875     31   0.72166 1.0992 0.53589
## 20 0.00059500     32   0.72098 1.0994 0.53589
## 21 0.00058039     34   0.71979 1.0995 0.53589
## 22 0.00057433     35   0.71921 1.1013 0.53711
## 23 0.00055054     38   0.71749 1.1011 0.53711
## 24 0.00052602     39   0.71694 1.1015 0.53711
## 25 0.00049045     40   0.71641 1.1011 0.53711
## 26 0.00047683     42   0.71543 1.1009 0.53711
## 27 0.00043334     43   0.71495 1.1011 0.53711
## 28 0.00043050     44   0.71452 1.1009 0.53749
## 29 0.00042955     45   0.71409 1.1009 0.53749
## 30 0.00042910     46   0.71366 1.1009 0.53749
## 31 0.00037147     47   0.71323 1.1008 0.53749
## 32 0.00031456     48   0.71286 1.1012 0.53749
## 33 0.00029768     49   0.71254 1.1009 0.53749
## 34 0.00027747     50   0.71225 1.1010 0.53749
## 35 0.00027244     51   0.71197 1.1008 0.53749
## 36 0.00027060     52   0.71170 1.1007 0.53749
## 37 0.00025443     54   0.71115 1.1005 0.53749
## 38 0.00025046     55   0.71090 1.1009 0.53790
## 39 0.00024971     56   0.71065 1.1009 0.53790
## 40 0.00023602     57   0.71040 1.1008 0.53790
## 41 0.00023567     58   0.71016 1.1011 0.53790
## 42 0.00021953     59   0.70993 1.1008 0.53790
## 43 0.00021412     62   0.70926 1.1011 0.53790
## 44 0.00021159     63   0.70905 1.1003 0.53790
## 45 0.00019558     64   0.70884 1.1000 0.53790
## 46 0.00019271     65   0.70864 1.1002 0.53790
## 47 0.00018788     66   0.70845 1.0997 0.53749
## 48 0.00017582     67   0.70826 1.0995 0.53749
## 49 0.00016874     68   0.70808 1.0991 0.53749
## 50 0.00016378     69   0.70792 1.0992 0.53749
## 51 0.00015906     70   0.70775 1.0991 0.53749
## 52 0.00015199     71   0.70759 1.0991 0.53749
## 53 0.00014750     72   0.70744 1.0993 0.53749
## 54 0.00014009     74   0.70715 1.0992 0.53749
## 55 0.00013589     75   0.70701 1.0988 0.53749
## 56 0.00013324     76   0.70687 1.0988 0.53749
## 57 0.00013246     77   0.70674 1.0989 0.53749
## 58 0.00012782     78   0.70660 1.0988 0.53749
## 59 0.00012724     79   0.70648 1.0987 0.53749
## 60 0.00012357     80   0.70635 1.0993 0.53799
## 61 0.00012187     81   0.70623 1.0993 0.53799
## 62 0.00011993     82   0.70610 1.0994 0.53799
## 63 0.00011872     85   0.70574 1.0994 0.53799
## 64 0.00011854     86   0.70563 1.0994 0.53799
## 65 0.00011588     87   0.70551 1.0994 0.53799
## 66 0.00011474     88   0.70539 1.0994 0.53799
## 67 0.00010915     89   0.70528 1.0994 0.53799
```

```
## 68 0.00010736     90   0.70517 1.0992 0.53799
## 69 0.00010294     91   0.70506 1.0993 0.53799
## 70 0.00010119     92   0.70496 1.0992 0.53799
## 71 0.00010091     93   0.70486 1.0992 0.53799
## 72 0.00010000     94   0.70475 1.0991 0.53799
```

```
## We can use the following method to choose the cp with the smallest xerror
fit.tree$cptable[which.min(fit.tree$cptable[,"xerror"]),"CP"]
```

```
## [1] 0.04740875
```

```
## Build the tree model with the cp which has smallest xerror
tree2 <- prune(fit.tree, cp= fit.tree$cptable[which.min(fit.tree$cptable[,"xerror"]
),"CP"])
## Make the visuallization of regreesion tree
rpart.plot(tree2)
```


```
## MSE of train for Boston
tree.pred.train = predict(tree2,boslist_train)
mean((tree.pred.train-boslist_train$price)^2)
```

```
## [1] 102127.8
```

```
## MSE of test for Boston
tree.pred.test = predict(tree2,boslist_test)
mean((tree.pred.test-boslist_test$price)^2)
```

```
## [1] 51023.49
```

Regression Tree–Beijing

```
#Regression tree
fit.tree <- rpart(price~.,
                  bjList_train,
                  control = rpart.control(cp = 0.0001))
## Printcp will tell you what the cp of spliting into diffrent number layer and the
xerror and xstd of each cp.
printcp(fit.tree)
```

```
##
## Regression tree:
## rpart(formula = price ~ ., data = bjList_train, control = rpart.control(cp = 1e-
04))
##
## Variables actually used in tree construction:
##  [1] accommodates          availability_30       availability_365
##  [4] availability_90       bathrooms             bedrooms
##  [7] beds                  guests_included       host_listings_count
## [10] minimum_nights        number_of_reviews     number_of_reviews_ltm
## [13] room_type_Shared_room TV_available
##
## Root node error: 1.7126e+10/11339 = 1510326
##
## n= 11339
##
##            CP nsplit rel error  xerror    xstd
## 1  0.05910445      0   1.00000 1.00021 0.38630
## 2  0.01825529      1   0.94090 0.94924 0.38588
## 3  0.01777481      2   0.92264 0.93084 0.38579
## 4  0.01675148      3   0.90487 0.98388 0.38619
## 5  0.01454429      4   0.88811 0.97752 0.38470
## 6  0.00597789      9   0.81418 0.97794 0.38477
## 7  0.00421139     10   0.80820 1.00271 0.38524
## 8  0.00355436     11   0.80399 0.99910 0.38431
## 9  0.00300284     12   0.80043 0.99603 0.38412
## 10 0.00282454     13   0.79743 0.99739 0.38406
## 11 0.00266045     14   0.79461 0.99735 0.38406
```

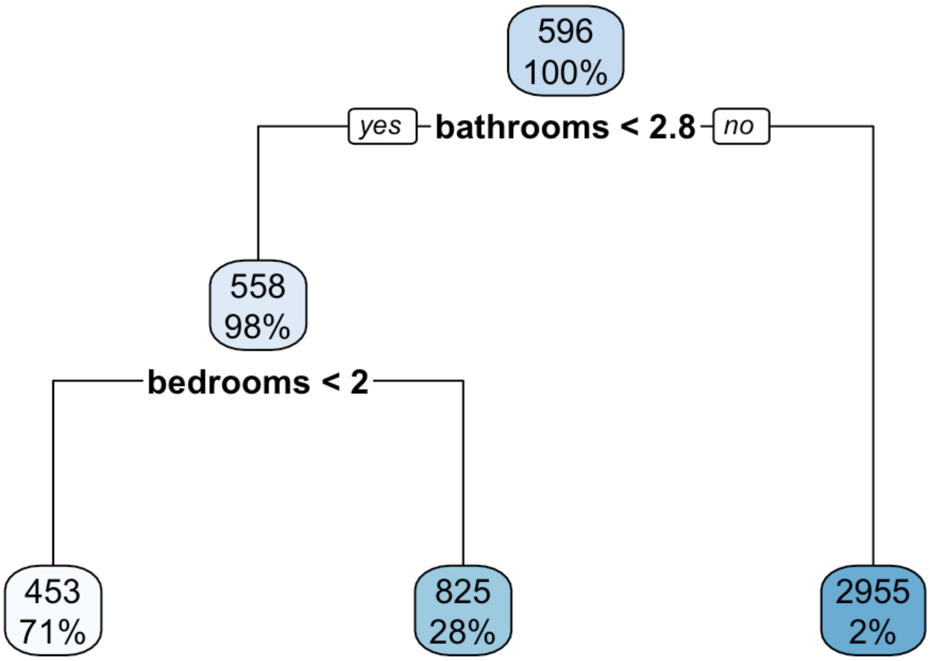
```
## 12 0.00258507     15   0.79195 1.00043 0.38407
## 13 0.00208874     16   0.78936 1.00013 0.38430
## 14 0.00134262     19   0.78309 1.00140 0.38444
## 15 0.00131161     20   0.78175 1.00251 0.38453
## 16 0.00114899     24   0.77651 1.00249 0.38446
## 17 0.00099309     28   0.77191 1.00296 0.38447
## 18 0.00093220     29   0.77092 1.00334 0.38447
## 19 0.00088688     30   0.76998 1.00309 0.38447
## 20 0.00081555     31   0.76910 1.00330 0.38448
## 21 0.00080145     32   0.76828 1.00353 0.38448
## 22 0.00075446     33   0.76748 1.00336 0.38448
## 23 0.00070218     34   0.76673 1.00352 0.38452
## 24 0.00066583     36   0.76532 1.00203 0.38452
## 25 0.00066127     37   0.76466 1.00241 0.38456
## 26 0.00063417     40   0.76267 1.00243 0.38456
## 27 0.00062281     41   0.76204 1.00256 0.38456
## 28 0.00052259     42   0.76142 1.00242 0.38456
## 29 0.00051859     43   0.76089 1.00307 0.38456
## 30 0.00049473     44   0.76037 1.00299 0.38455
## 31 0.00049396     46   0.75938 1.00357 0.38455
## 32 0.00045649     48   0.75840 1.00272 0.38415
## 33 0.00044375     49   0.75794 1.00279 0.38415
## 34 0.00040826     51   0.75705 1.00490 0.38516
## 35 0.00037973     53   0.75624 1.00498 0.38516
## 36 0.00036119     54   0.75586 1.00452 0.38543
## 37 0.00035905     55   0.75550 1.00512 0.38543
## 38 0.00035727     57   0.75478 1.00482 0.38543
## 39 0.00033033     58   0.75442 1.00486 0.38543
## 40 0.00032753     59   0.75409 1.00517 0.38543
## 41 0.00032219     61   0.75343 1.00495 0.38543
## 42 0.00031845     63   0.75279 1.00456 0.38516
## 43 0.00030435     65   0.75215 1.00435 0.38516
## 44 0.00030021     66   0.75185 1.00435 0.38516
## 45 0.00029904     68   0.75125 1.00446 0.38516
## 46 0.00029724     71   0.75035 1.00452 0.38516
## 47 0.00029234     72   0.75005 1.00454 0.38516
## 48 0.00027830     73   0.74976 1.00440 0.38516
## 49 0.00027172     75   0.74921 1.00438 0.38516
## 50 0.00025896     76   0.74893 1.00366 0.38515
## 51 0.00025374     77   0.74867 1.00335 0.38514
## 52 0.00024926     78   0.74842 1.00232 0.38513
## 53 0.00024303     79   0.74817 1.00232 0.38513
## 54 0.00024043     80   0.74793 1.00215 0.38513
## 55 0.00023573     81   0.74769 1.00224 0.38513
## 56 0.00021932     86   0.74651 1.00217 0.38513
## 57 0.00021313     87   0.74629 1.00166 0.38513
## 58 0.00020991     88   0.74608 1.00199 0.38513
## 59 0.00020304     90   0.74566 1.00187 0.38513
## 60 0.00019303     91   0.74545 1.00087 0.38503
## 61 0.00018119     94   0.74487 1.00076 0.38503
## 62 0.00017968     95   0.74469 1.00095 0.38503
```

```
## 63 0.00017557     99   0.74397 1.00098 0.38503
## 64 0.00016907    100   0.74380 1.00071 0.38502
## 65 0.00016836    102   0.74346 1.00068 0.38502
## 66 0.00016705    104   0.74312 1.00067 0.38502
## 67 0.00016307    106   0.74279 1.00074 0.38502
## 68 0.00016163    107   0.74263 1.00121 0.38502
## 69 0.00015853    108   0.74247 1.00102 0.38502
## 70 0.00014854    109   0.74231 1.00091 0.38502
## 71 0.00013946    115   0.74142 1.00061 0.38502
## 72 0.00013378    116   0.74128 1.00046 0.38501
## 73 0.00012851    118   0.74101 1.00072 0.38502
## 74 0.00012816    119   0.74088 1.00084 0.38502
## 75 0.00012334    120   0.74075 1.00106 0.38501
## 76 0.00011169    121   0.74063 0.99996 0.38489
## 77 0.00011162    122   0.74052 1.00010 0.38489
## 78 0.00011152    123   0.74041 1.00009 0.38489
## 79 0.00010025    124   0.74029 1.00030 0.38489
## 80 0.00010000    125   0.74019 1.00029 0.38489
```

```
## We can use the following method to choose the cp with the smallest xerror
fit.tree$cptable[which.min(fit.tree$cptable[,"xerror"]),"CP"]
```

```
## [1] 0.01777481
```

```
## Build the tree model with the cp which has smallest xerror
tree2 <- prune(fit.tree, cp= fit.tree$cptable[which.min(fit.tree$cptable[,"xerror"]
),"CP"])
## Make the visuallization of regreesion tree
rpart.plot(tree2)
```




```
## MSE of train for Beijing
tree.pred.train = predict(tree2,bjList_train)
mean((tree.pred.train-bjList_train$price)^2)
```

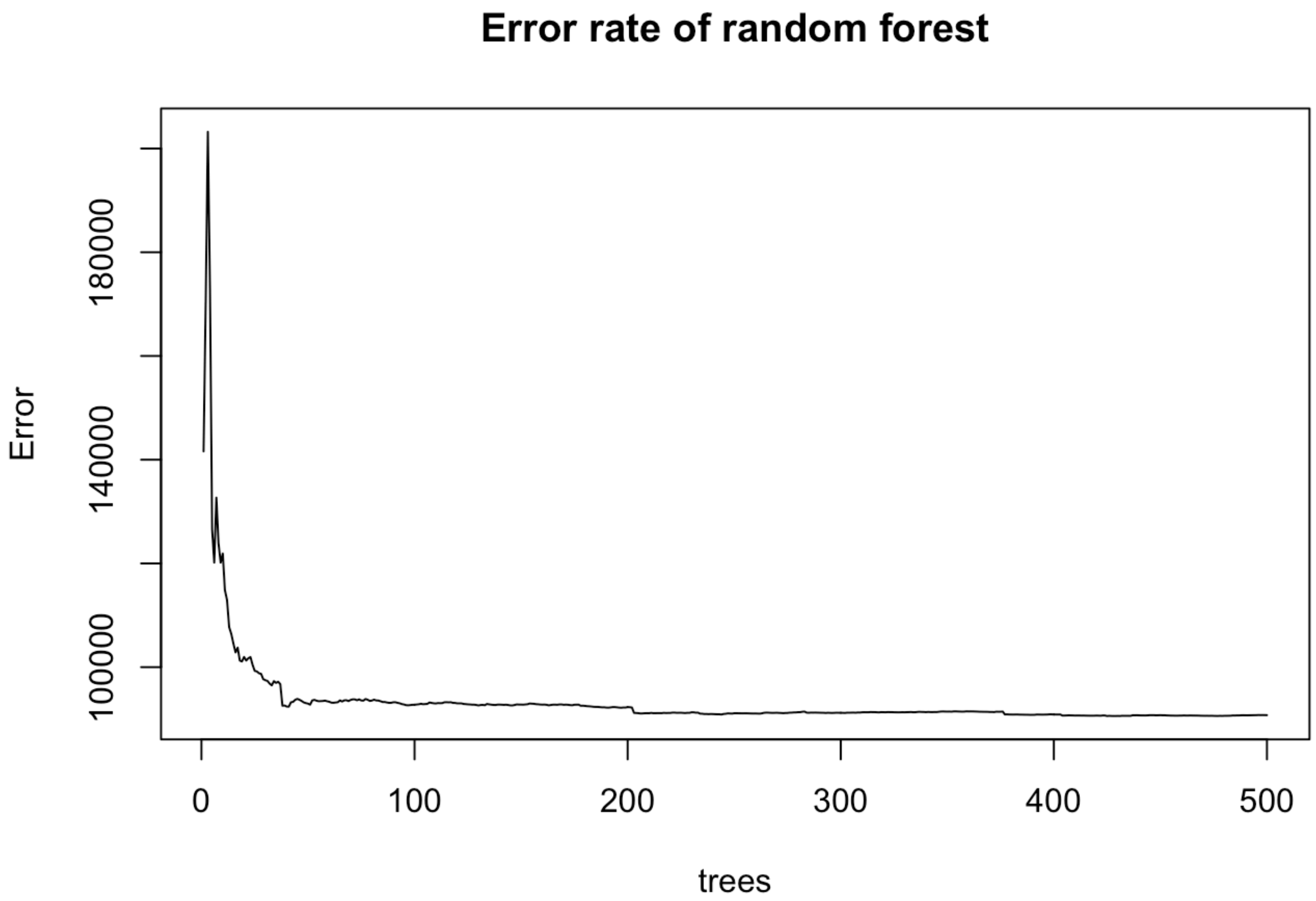
```
## [1] 1393488
```

```
## MSE of test for Beijing
tree.pred.test = predict(tree2,bjList_test)
mean((tree.pred.test-bjList_test$price)^2)
```

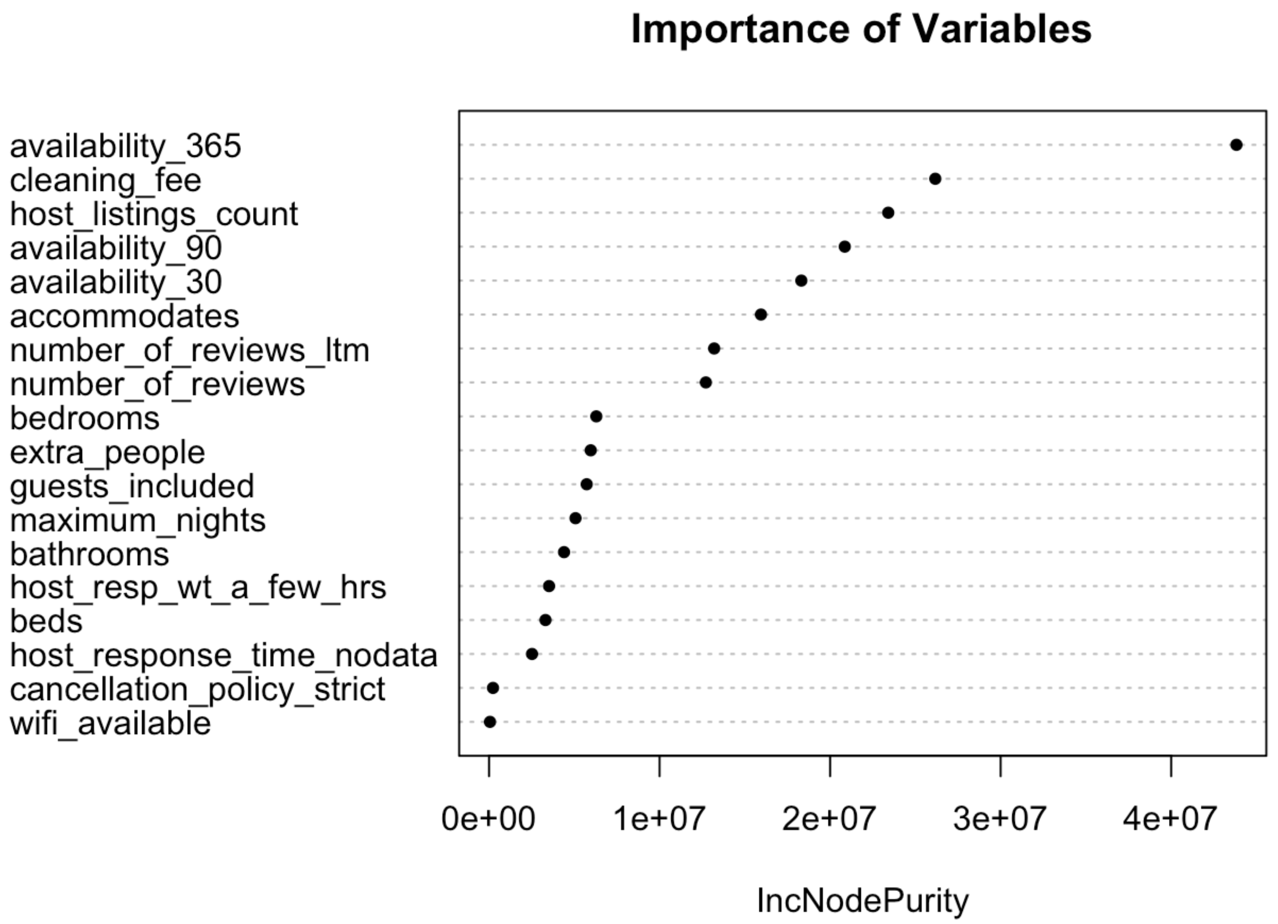
```
## [1] 3758002
```

Random Forest–Boston

```
##Random Forest
#decide ntree by the plot of error vs ntree
error_rf <- randomForest(price ~.,data=boslist_train)
plot(error_rf,main = "Error rate of random forest")
```




```
fit_rf <- randomForest(price~.,
                       boslist_train,
                       ntree=100,
                       do.trace=F)

varImpPlot(fit_rf,pch = 20, main = "Importance of Variables")
```




```
## MSE of train for Boston
yhat_rf <- predict(fit_rf, boslist_train)
train_mse_rf <- mean((yhat_rf - boslist_train$price) ^ 2)
print(train_mse_rf)
```

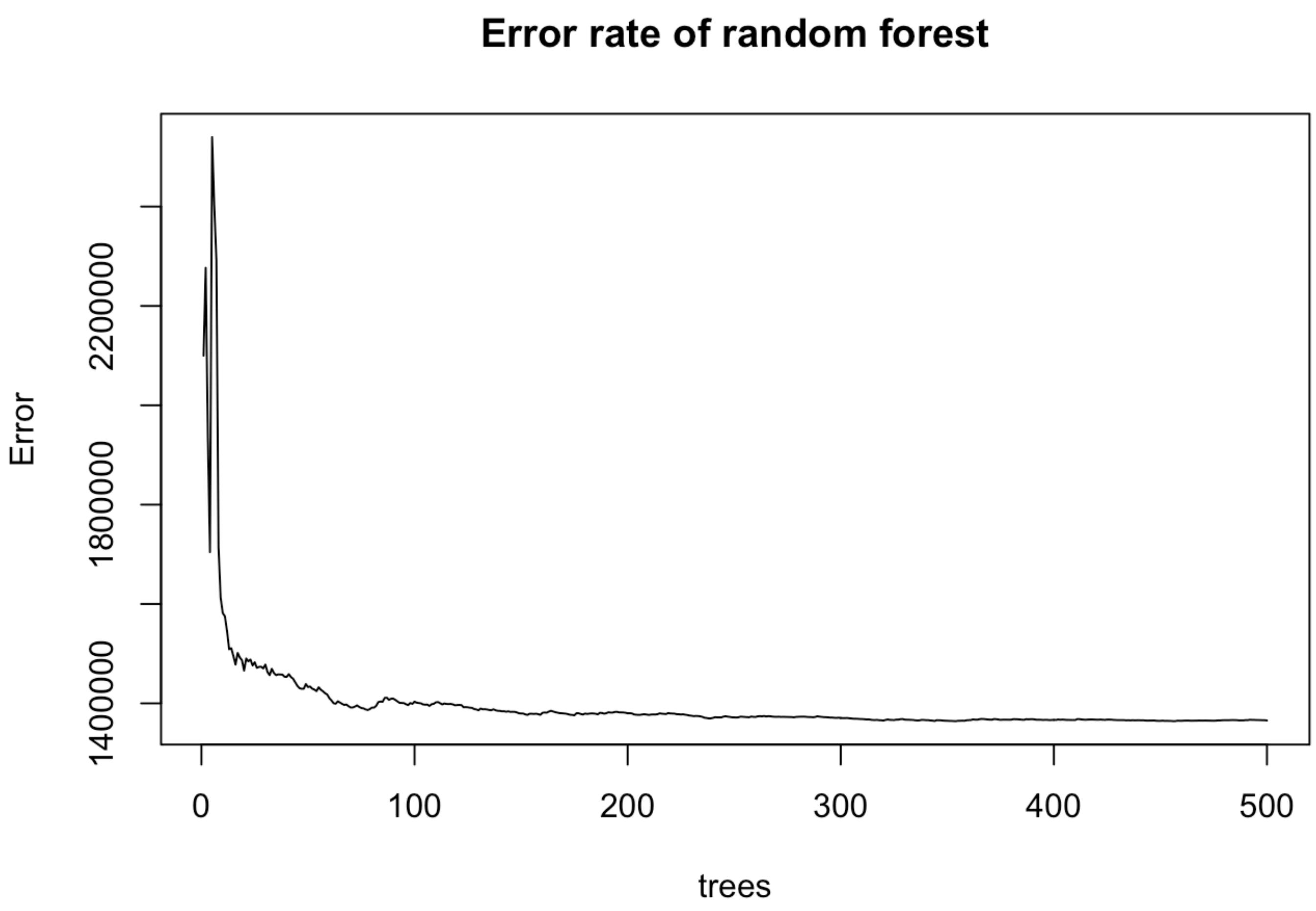
```
## [1] 26746.58
```

```
#levels(boslist_test$neighbourhood_cleansed) = levels(boslist_train$neighbourhood_c
leansed)

## MSE of Test for Boston
yhat_rf <- predict(fit_rf, boslist_test)
test_mse_rf <- mean((yhat_rf - boslist_test$price) ^ 2)
print(test_mse_rf)
```

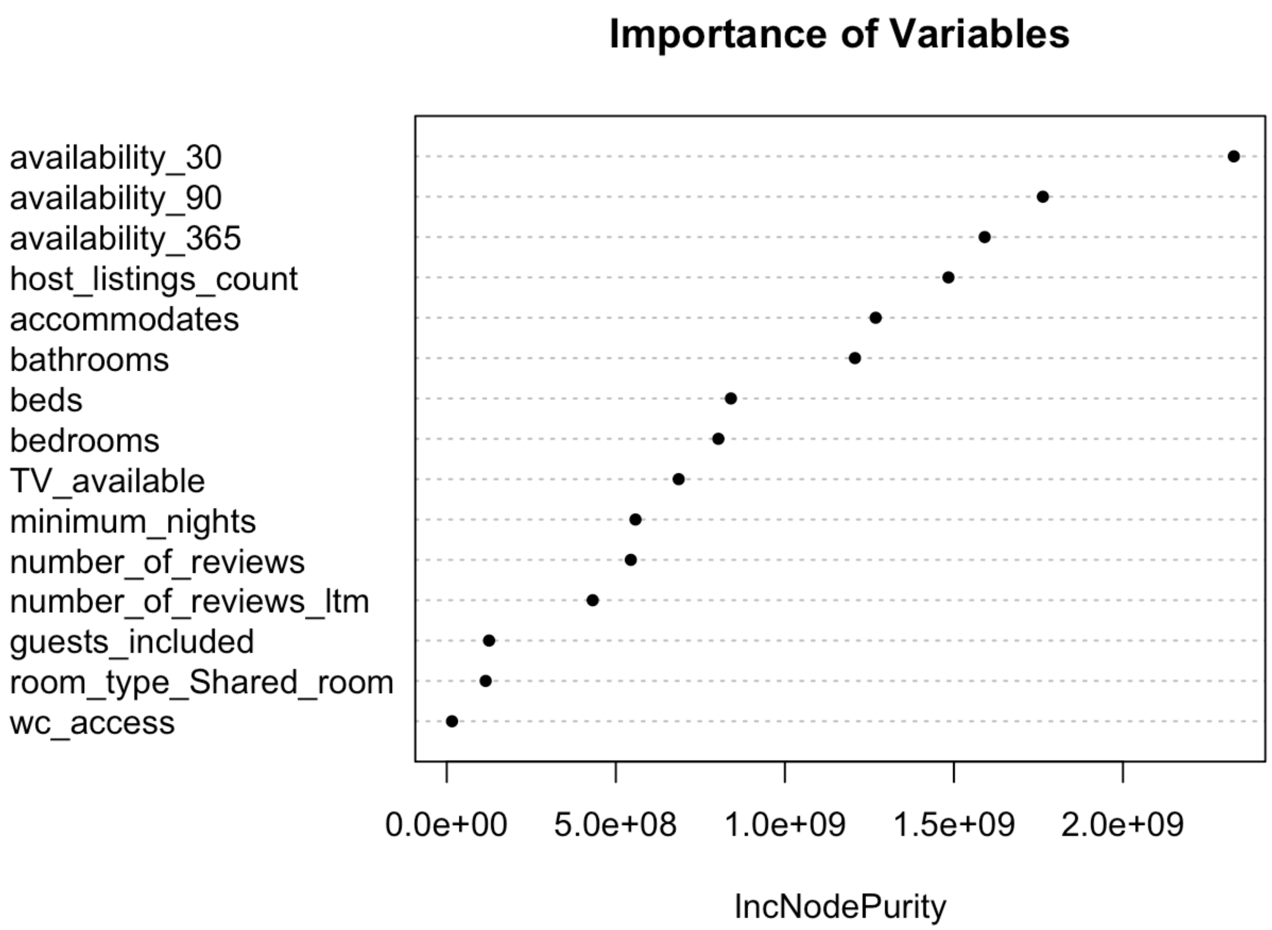
```
## [1] 41468.7
```

Random Forest–Beijing

```
##Random Forest
#decide ntree by the plot of error vs ntree
error_rf <- randomForest(price ~.,data=bjList_train)
plot(error_rf,main = "Error rate of random forest")
```




```
fit_rf <- randomForest(price~.,
                       bjList_train,
                       ntree=,
                       do.trace=F)

varImpPlot(fit_rf,pch = 20, main = "Importance of Variables")
```




```
## MSE of train for Beijing
yhat_rf <- predict(fit_rf, bjList_train)
train_mse_rf <- mean((yhat_rf - bjList_train$price) ^ 2)
print(train_mse_rf)
```

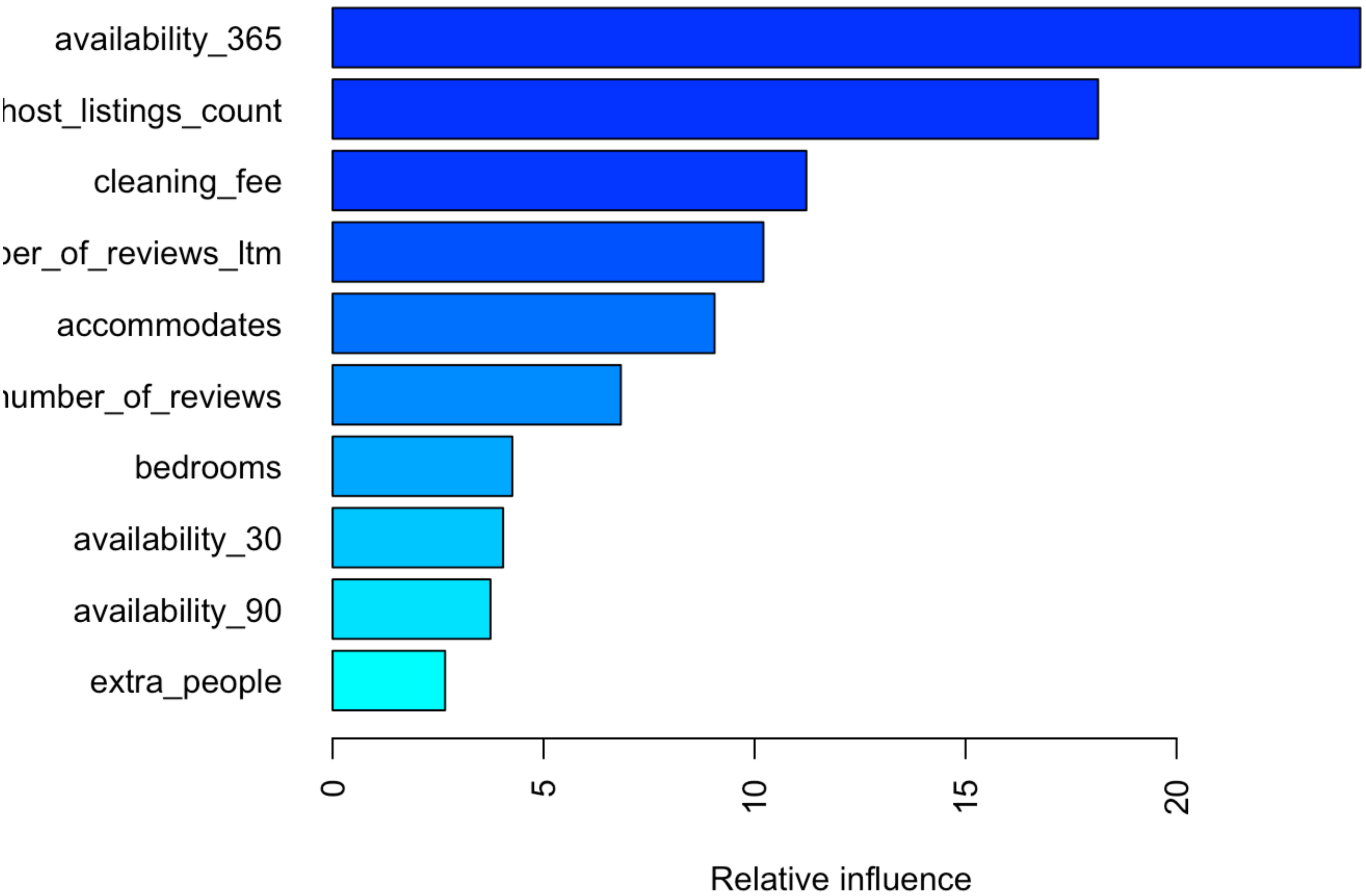
```
## [1] 423271.7
```

```
#levels(boslist_test$neighbourhood_cleansed) = levels(boslist_train$neighbourhood_c
leansed)

## MSE of Test for Beijing
yhat_rf <- predict(fit_rf, bjList_test)
test_mse_rf <- mean((yhat_rf - bjList_test$price) ^ 2)
print(test_mse_rf)
```

```
## [1] 3661734
```

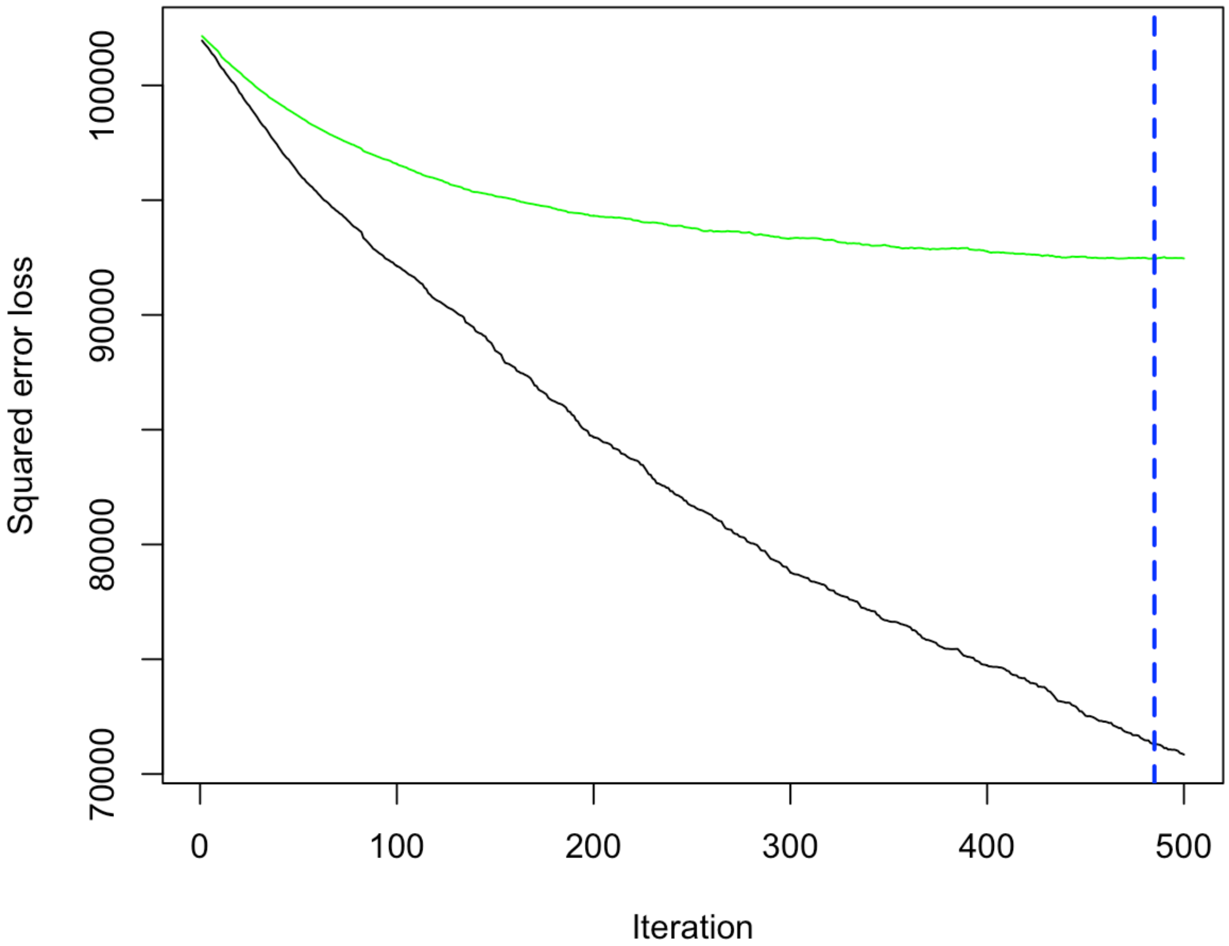
gradient boosting–Boston

```
Boston.boost=gbm(formula = price~., distribution = "gaussian", data = boslist_train
, n.trees = 500,interaction.depth = 15, shrinkage = 0.005,cv.folds = 5)

# A gradient boosted model with gaussian loss function.
# 10000 iterations were performed.
# There were 13 predictors of which 13 had non-zero influence.
par(mar = c(5, 8, 1, 1))
summary(
  Boston.boost,
  cBars = 10,
  method = relative.influence, # also can use permutation.test.gbm
  las = 2
)
```



```
##                                                        var      rel.inf
## availability_365                          availability_365 24.355936498
## host_listings_count                    host_listings_count 18.141879544
## cleaning_fee                                  cleaning_fee 11.227977415
## number_of_reviews_ltm                number_of_reviews_ltm 10.210150775
## accommodates                                  accommodates  9.052977348
## number_of_reviews                        number_of_reviews  6.832077647
## bedrooms                                          bedrooms  4.258651977
## availability_30                            availability_30  4.040906064
## availability_90                            availability_90  3.741571270
## extra_people                                  extra_people  2.665901501
## guests_included                            guests_included  1.805994028
## bathrooms                                        bathrooms  1.214525765
## beds                                                  beds  0.952384534
## maximum_nights                              maximum_nights  0.732995867
## host_response_time_nodata      host_response_time_nodata  0.548273361
## host_resp_wt_a_few_hrs            host_resp_wt_a_few_hrs  0.179287363
## cancellation_policy_strict cancellation_policy_strict  0.036979478
## wifi_available                              wifi_available  0.001529566
```

```
perf_gbm1 = gbm.perf(Boston.boost, method = "cv")
```



```
boostpre <- predict(
                    # the model from above
                    object = Boston.boost,
                    # the testing data
                    newdata = boslist_train,
                    # this is the number we calculated above
                    n.trees = perf_gbm1)
rmse_fit <- Metrics::rmse(actual = boslist_train$price,
                           predicted = boostpre)
## MSE of train for Boston
rmse_fit^2
```

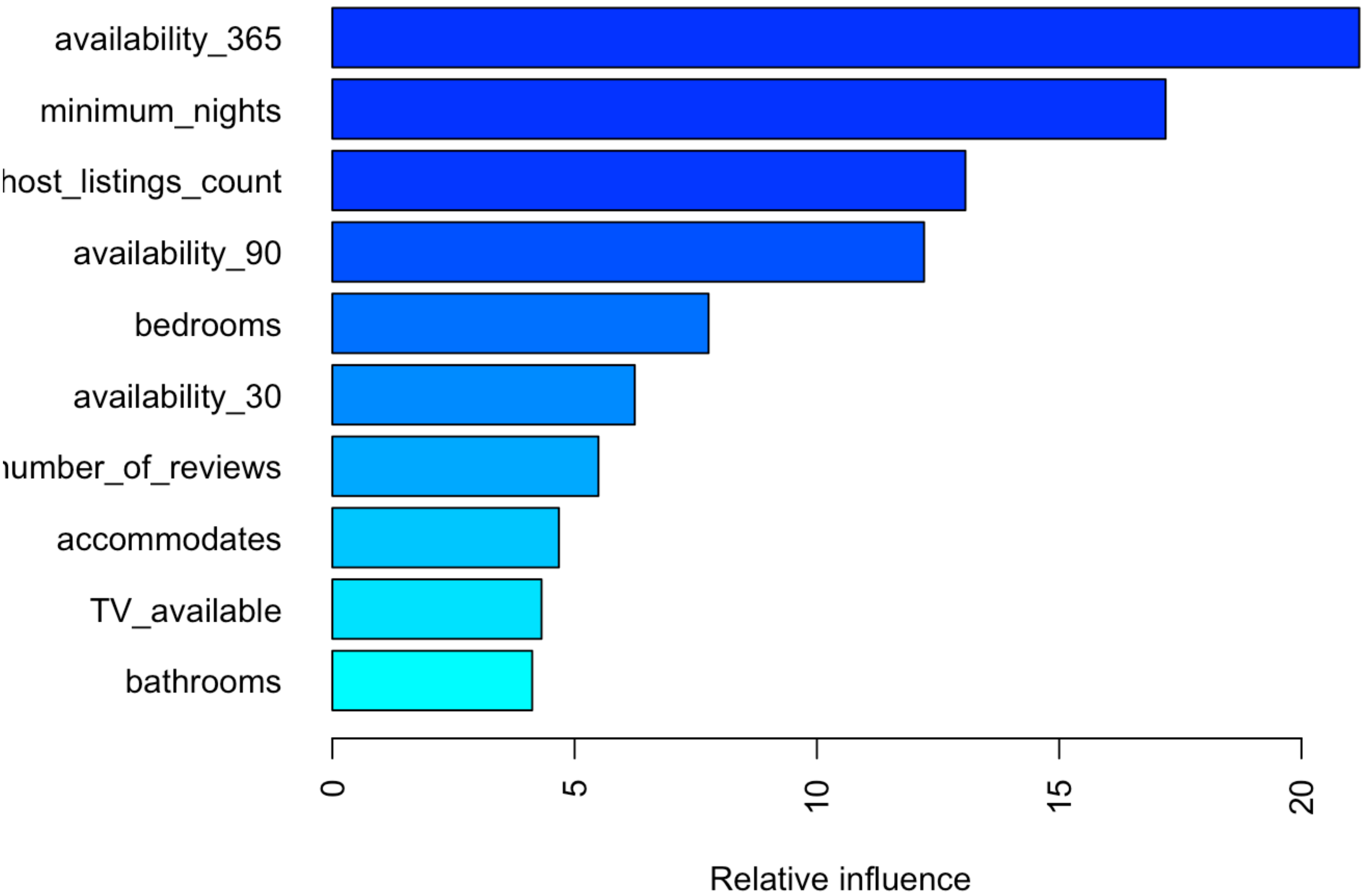
```
## [1] 71312.2
```

```
boostpre <- predict(
  # the model from above
  object = Boston.boost,
  # the testing data
  newdata = boslist_test,
  # this is the number we calculated above
  n.trees = perf_gbm1)
rmse_fit <- Metrics::rmse(actual = boslist_test$price,
                          predicted = boostpre)
## MSE of test for Boston
rmse_fit^2
```

```
## [1] 44115.97
```

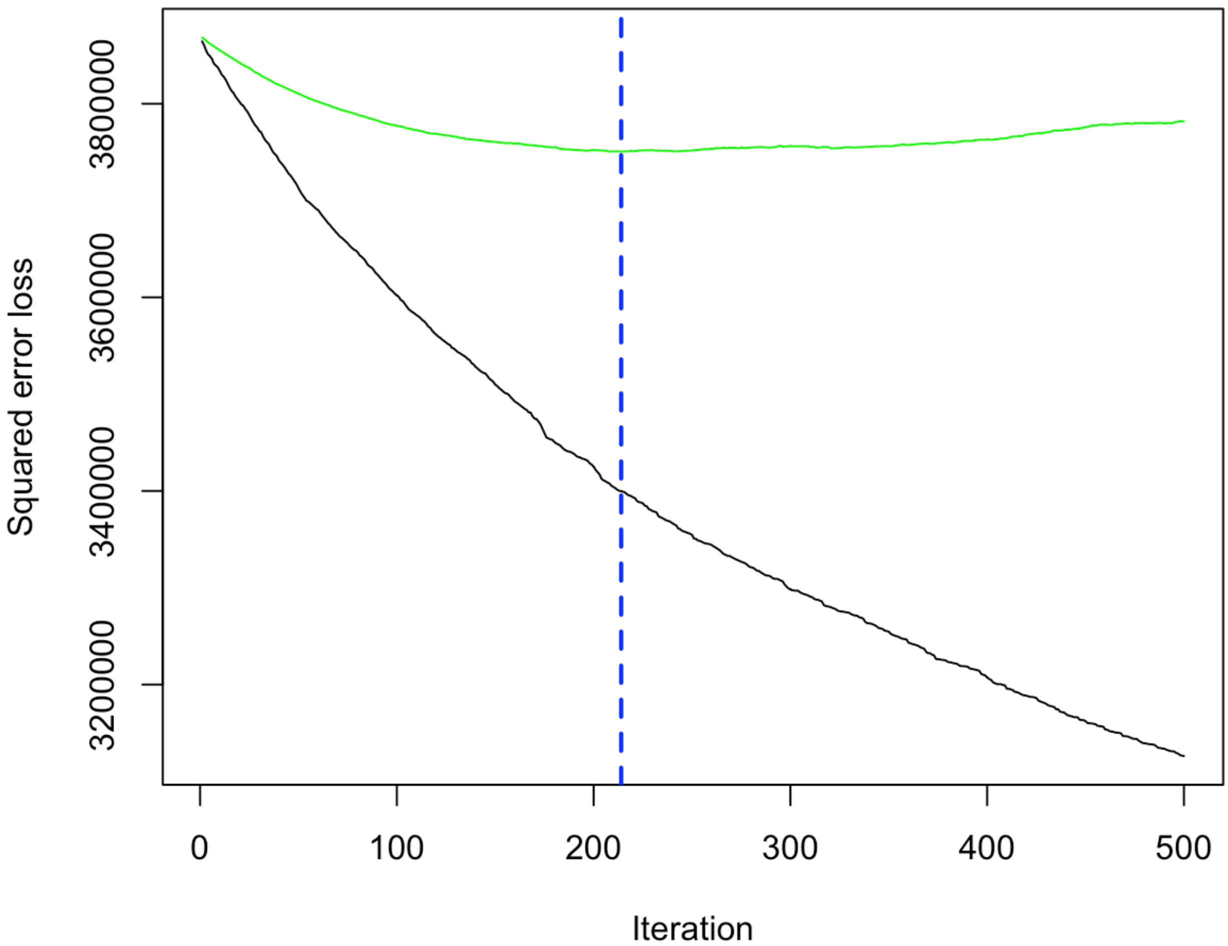
gradient boosting–Beijing

```
beijing.boost=gbm(formula = price~., distribution = "gaussian", data = bjList_test,
n.trees = 500,interaction.depth = 15, shrinkage = 0.005,cv.folds = 5)

# A gradient boosted model with gaussian loss function.
# 10000 iterations were performed.
# There were 13 predictors of which 13 had non-zero influence.
par(mar = c(5, 8, 1, 1))
summary(
  beijing.boost,
  cBars = 10,
  method = relative.influence, # also can use permutation.test.gbm
  las = 2
)
```




```
##                                              var    rel.inf
## availability_365               availability_365 21.1937833
## minimum_nights                   minimum_nights 17.1966362
## host_listings_count         host_listings_count 13.0653907
## availability_90                 availability_90 12.2120743
## bedrooms                               bedrooms  7.7645615
## availability_30                 availability_30  6.2414069
## number_of_reviews             number_of_reviews  5.4914477
## accommodates                       accommodates  4.6757798
## TV_available                       TV_available  4.3176242
## bathrooms                             bathrooms  4.1247151
## number_of_reviews_ltm number_of_reviews_ltm  2.3559674
## beds                                       beds  0.6539386
## wc_access                             wc_access  0.3611393
## room_type_Shared_room room_type_Shared_room  0.1902572
## guests_included                 guests_included  0.1552776
```

```
perf_gbm1 = gbm.perf(beijing.boost, method = "cv")
```




```
boostpre <- predict(
                    # the model from above
                    object = beijing.boost,
                    # the testing data
                    newdata = bjList_train,
                    # this is the number we calculated above
                    n.trees = perf_gbm1)
rmse_fit <- Metrics::rmse(actual = bjList_train$price,
                           predicted = boostpre)
## MSE of train for Beijing
rmse_fit^2
```

```
## [1] 1419802
```

```
boostpre <- predict(
  # the model from above
  object = beijing.boost,
  # the testing data
  newdata = bjList_test,
  # this is the number we calculated above
  n.trees = perf_gbm1)
rmse_fit <- Metrics::rmse(actual = bjList_test$price,
                          predicted = boostpre)
## MSE of test for Beijing
rmse_fit^2
```

```
## [1] 3399814
```