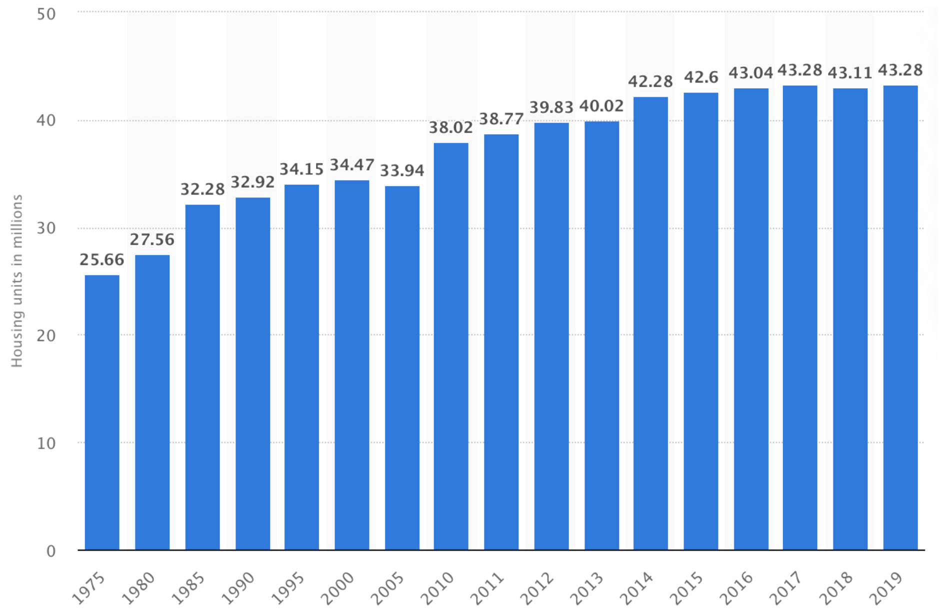
Report on Rental Listing Inquiries

Using three classification methods

Bing Chen
20133887
Stat 457
April 18, 2021

Introduction



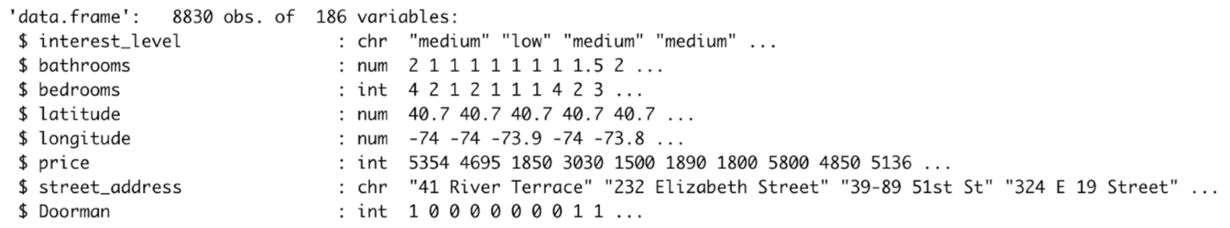
THIS GRAPH SHOWS THE NUMBER OF RENTERS FROM 1975 TO 2019 IN US.

According to statistical research published by the stastia research team, the number of renter-occupied homes in the U.S. 1975-2019, it is easy to observe approximately 43 million housing units occupied by renters in the United States 2019. It is a long-term upward swing since 1975, indicating that the demand for rental housing has increased. Since 2015, there has been a steady increase in the number of rental occupancies each year. According to data from the 2015 American Housing Survey, there are nearly 48.5 million rental units in the United States, 43.9 million of which are occupied. Due to technological developments, which support clear data analysis, renters now have more choices and methods in how they acquire their dream homes. In this report, I will apply technology and data science to financial forecasts. Based on the new listing's basic information and facilities, I will predict the number of inquiries received by the new listing. Doing so will help owners and agents better understand tenants' needs and preferences.

There will be three different methods of classification applying in this analysis, which consists of random forest (with ranger package), boosting tree (with gbm package), and finally, Support Vector Machine (SVM, with kernlab package).

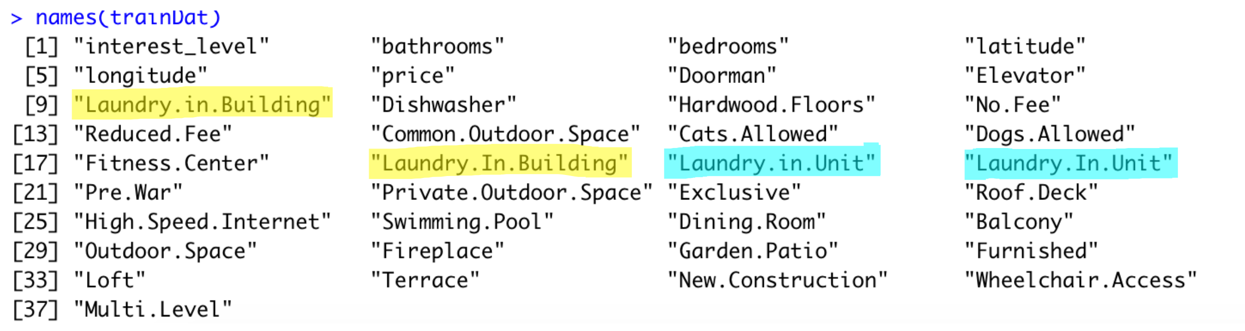
Description

I did some processing of the data set base on the code preprocessing based on the code provided by Professor Song. Firstly, we will introduce the dataset. There are 185 variables and a response (interest_level) and 8830 observations. The dimension of the train data set is 8830*186, and of test data set is 5888*186. The picture below is the structure of the first eight variables in the training dataset, and all other variables are indicator variables with integers 1(have) and 0(does not have).



```
'data.frame':	8830 obs. of  186 variables:
 $ interest_level              : chr  "medium" "low" "medium" "medium" ...
 $ bathrooms                   : num  2 1 1 1 1 1 1 1 1.5 2 ...
 $ bedrooms                    : int  4 2 1 2 1 1 1 4 2 3 ...
 $ latitude                    : num  40.7 40.7 40.7 40.7 40.7 ...
 $ longitude                   : num  -74 -74 -73.9 -74 -73.8 ...
 $ price                       : int  5354 4695 1850 3030 1500 1890 1800 5800 4850 5136 ...
 $ street_address              : chr  "41 River Terrace" "232 Elizabeth Street" "39-89 51st St" "324 E 19 Street" ...
 $ Doorman                     : int  1 0 0 0 0 0 0 0 1 1 ...
```

Professor Song deletes the variables that less than 200 houses can provide in preprocess. After the preprocessing, the training dataset has 36 variables.

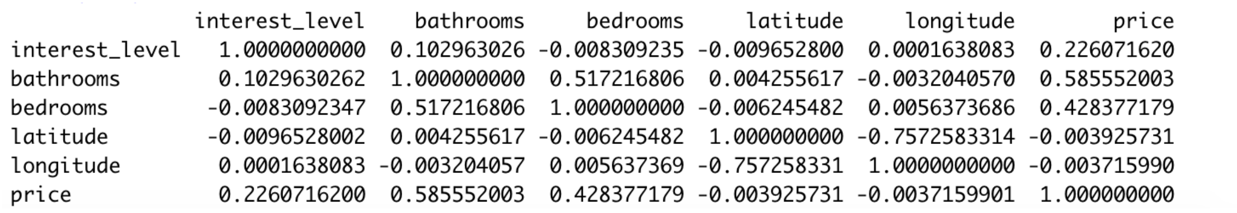


```
> names(trainDat)
 [1] "interest_level"      "bathrooms"             "bedrooms"          "latitude"
 [5] "longitude"           "price"                 "Doorman"           "Elevator"
 [9] "Laundry.in.Building" "Dishwasher"            "Hardwood.Floors"   "No.Fee"
[13] "Reduced.Fee"         "Common.Outdoor.Space"  "Cats.Allowed"      "Dogs.Allowed"
[17] "Fitness.Center"      "Laundry.In.Building"   "Laundry.in.Unit"   "Laundry.In.Unit"
[21] "Pre.War"             "Private.Outdoor.Space" "Exclusive"         "Roof.Deck"
[25] "High.Speed.Internet" "Swimming.Pool"         "Dining.Room"       "Balcony"
[29] "Outdoor.Space"       "Fireplace"             "Garden.Patio"      "Furnished"
[33] "Loft"                "Terrace"               "New.Construction"  "Wheelchair.Access"
[37] "Multi.Level"
```

I discovered a redundancy in the data, as there are two columns for “Laundry.in.Building" and two “Laundry.in.Units”, therefore the data is repetitive. By combining the same variables, there are two variables deleted in both the training and test dataset. Thus, there are 34 variables in the train dataset and test dataset.

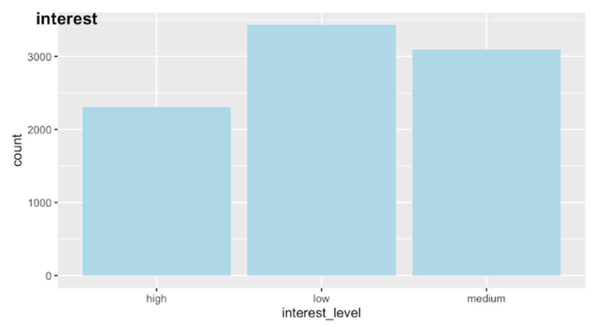
The basic data visualization and analysis:

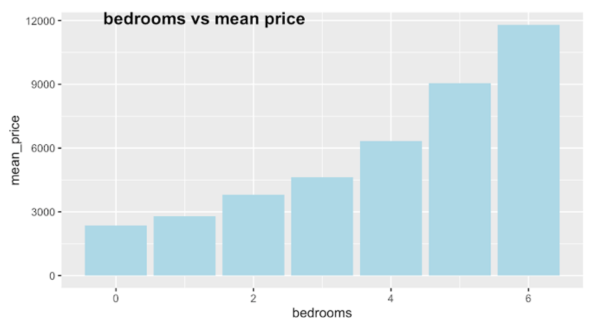
The table below is the correlation between response and the first six variables. As we can see, there is a positive correlation between price and level of interest(0.2260716200), as well as a positive correlation between price and both bathrooms and bedrooms. So, the increase in the number of bedrooms and bathrooms causes the price to increase.

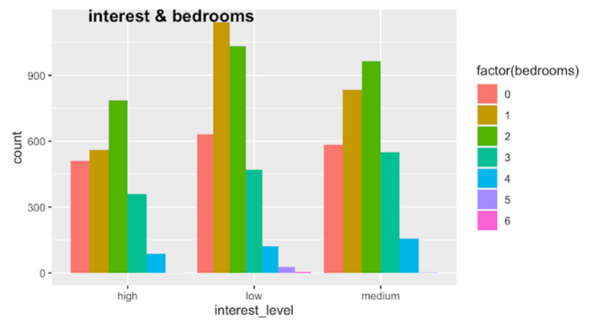


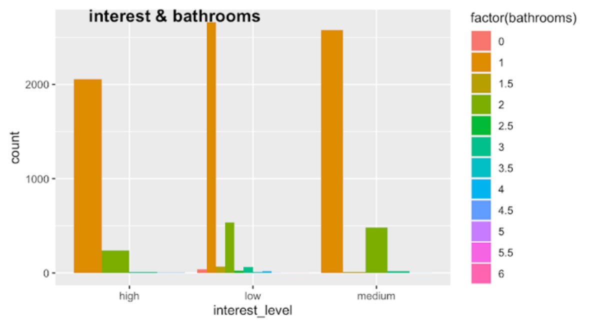
```
               interest_level    bathrooms     bedrooms     latitude     longitude        price
interest_level   1.0000000000  0.102963026 -0.008309235 -0.009652800  0.0001638083  0.226071620
bathrooms        0.1029630262  1.000000000  0.517216806  0.004255617 -0.0032040570  0.585552003
bedrooms        -0.0083092347  0.517216806  1.000000000 -0.006245482  0.0056373686  0.428377179
latitude        -0.0096528002  0.004255617 -0.006245482  1.000000000 -0.7572583314 -0.003925731
longitude        0.0001638083 -0.003204057  0.005637369 -0.757258331  1.0000000000 -0.003715990
price            0.2260716200  0.585552003  0.428377179 -0.003925731 -0.0037159901  1.000000000
```

Figure 2.1 is a graph table that includes the variables that are very important for people to consider. In the graph of interest, the number of each interest level is reasonable. There are no large gaps between low level and medium level, and between medium level and high level. From the graph of interest & bedroom, most apartments are one-bedroom or two bedrooms. Most people showed high or moderate interest in two bedrooms, demonstrating the greater demand is for two-bedroom apartments. In the "bedroom vs mean price" graph, it also shows the positive correlation between the number of bedrooms and means price, showing the same results as the correlation table. The last graph shows the rental units with one bathroom occupy 90 percent of the market.

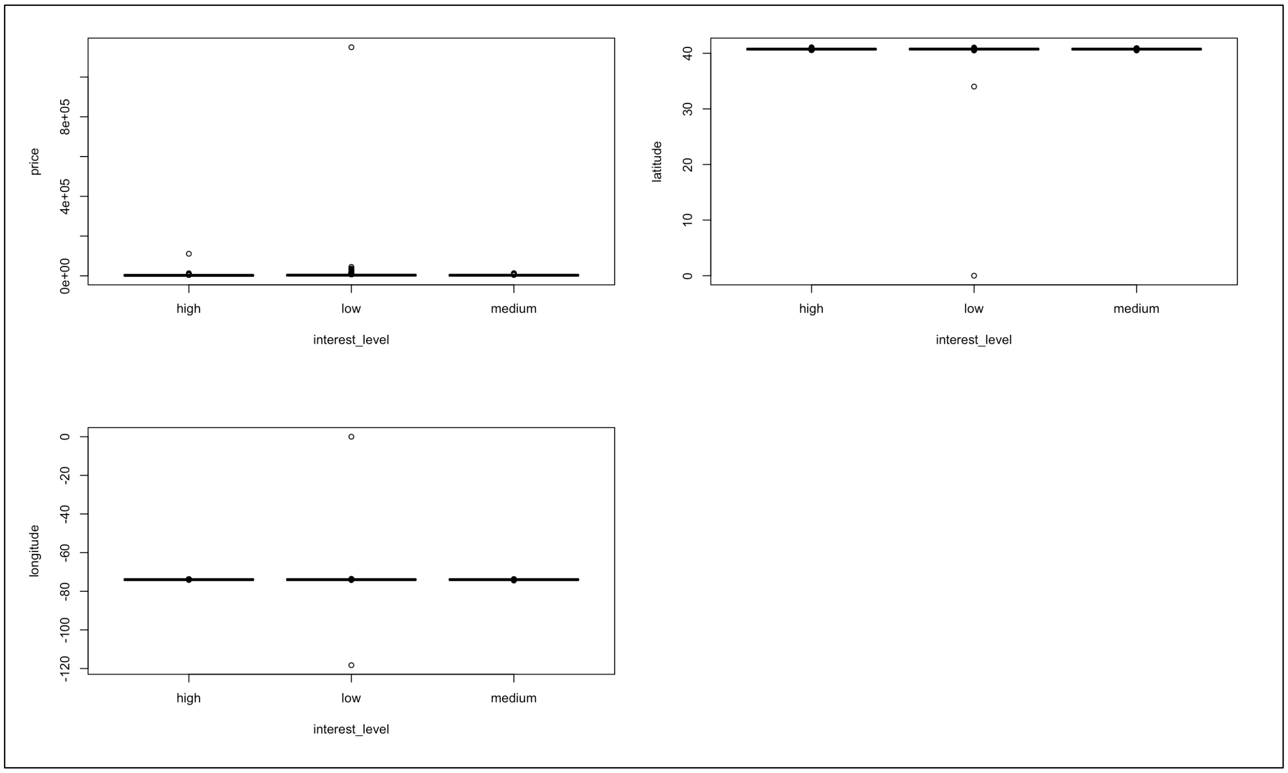




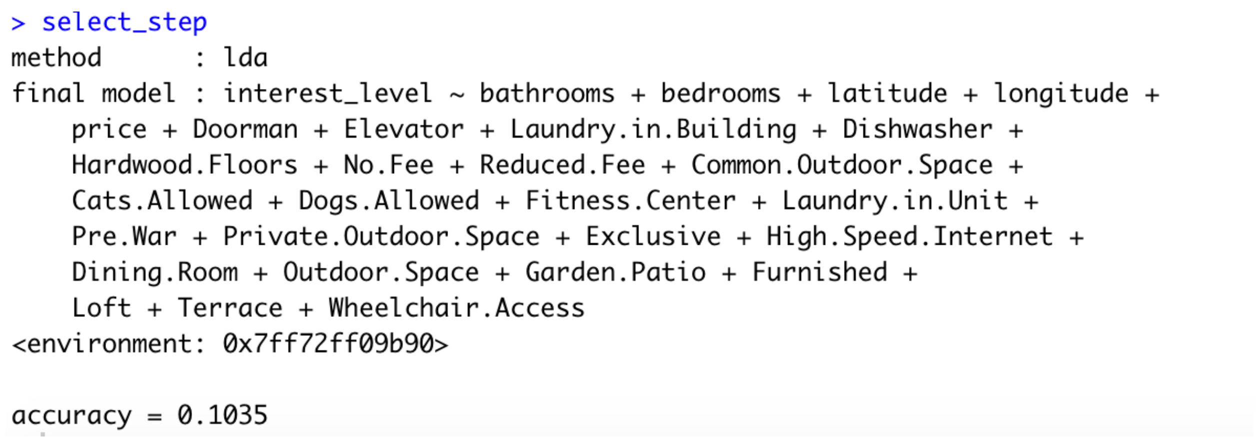




Next, we will see the outliers of the training dataset, which is prepared to do the feature selection and train the training dataset. By observing the graph table below, the boxplot of price vs interest_level shows that there are two outliers, one of the outliers is a house priced at close to one million dollars. In the second boxplot, latitude vs interest_level, there are two additional outliers. In the third boxplot, two outliers should be removed.



Feature selection:

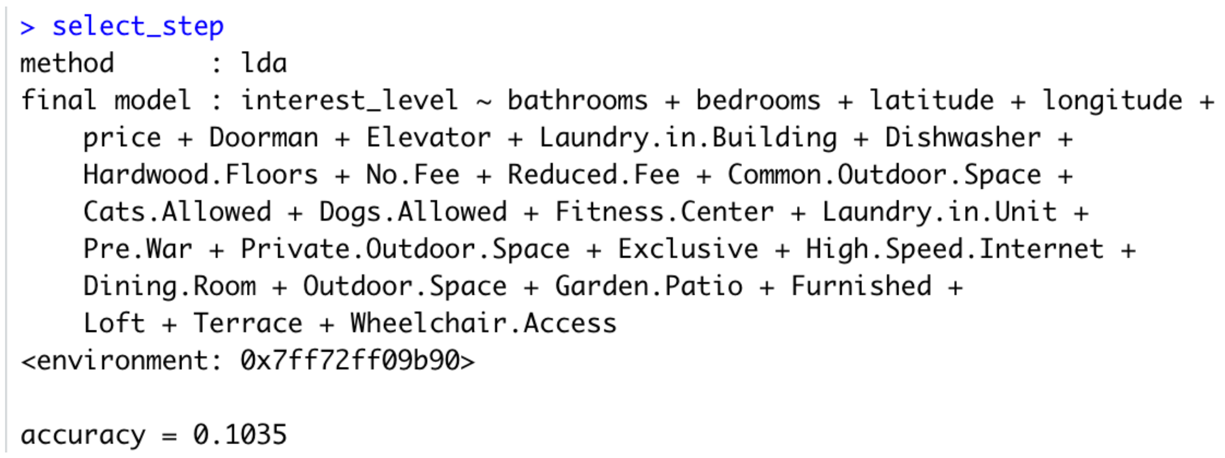


```
> select_step
method      : lda
final model : interest_level ~ bathrooms + bedrooms + latitude + longitude +
    price + Doorman + Elevator + Laundry.in.Building + Dishwasher +
    Hardwood.Floors + No.Fee + Reduced.Fee + Common.Outdoor.Space +
    Cats.Allowed + Dogs.Allowed + Fitness.Center + Laundry.in.Unit +
    Pre.War + Private.Outdoor.Space + Exclusive + High.Speed.Internet +
    Dining.Room + Outdoor.Space + Garden.Patio + Furnished +
    Loft + Terrace + Wheelchair.Access
<environment: 0x7ff72ff09b90>

accuracy = 0.1035
```

I chose the most important variables for analysis by using the stepclass and rpart functions.

The stepclass function is like the stepwise function but it is used for classification, and the variables are selected by estimated classification performance measure from ucpm, a function to calculate the Correctness Rate, the Accuracy, the Ability to Separate and the Confidence of a classification rule. Thus, there are 28 variables left and the result is in the picture below.



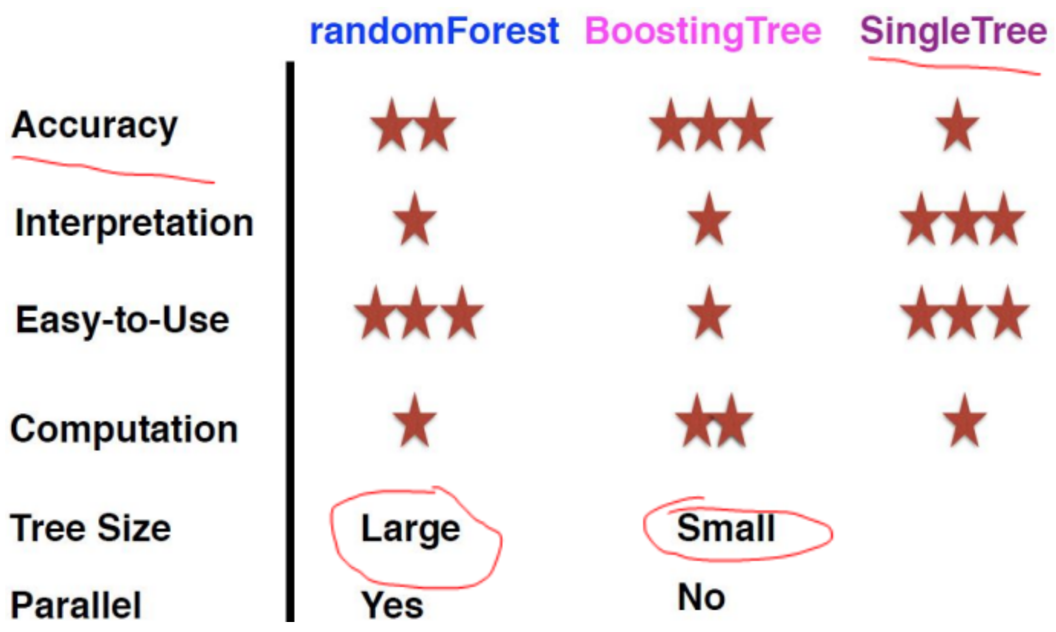
```
> select_step
method      : lda
final model : interest_level ~ bathrooms + bedrooms + latitude + longitude +
    price + Doorman + Elevator + Laundry.in.Building + Dishwasher +
    Hardwood.Floors + No.Fee + Reduced.Fee + Common.Outdoor.Space +
    Cats.Allowed + Dogs.Allowed + Fitness.Center + Laundry.in.Unit +
    Pre.War + Private.Outdoor.Space + Exclusive + High.Speed.Internet +
    Dining.Room + Outdoor.Space + Garden.Patio + Furnished +
    Loft + Terrace + Wheelchair.Access
<environment: 0x7ff72ff09b90>

accuracy = 0.1035
```

Method

I used tuning for all three of my classification methods, because parameter tuning can be beneficial by increasing the model accuracy, decreasing the time the model runs, and finally, decreasing the monetary spend on the model.

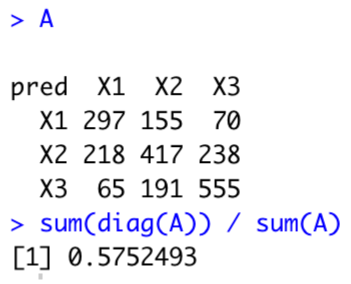
We have been introduced to several classification methods in this course, such as knn, randomforest, boosting, logistic, etc. Since the sample code is written by logistic, Random forest, boosting, and support vector machine(SVM) is mainly used to do classification for this project. Based on the above picture, I drew the conclusion based on the ratings that I had selected the best method that would suit my needs.



	randomForest	BoostingTree	SingleTree
Accuracy	★★	★★★	★
Interpretation	★	★	★★★
Easy-to-Use	★★★	★	★★★
Computation	★	★★	★
Tree Size	Large	Small	
Parallel	Yes	No	

SVM

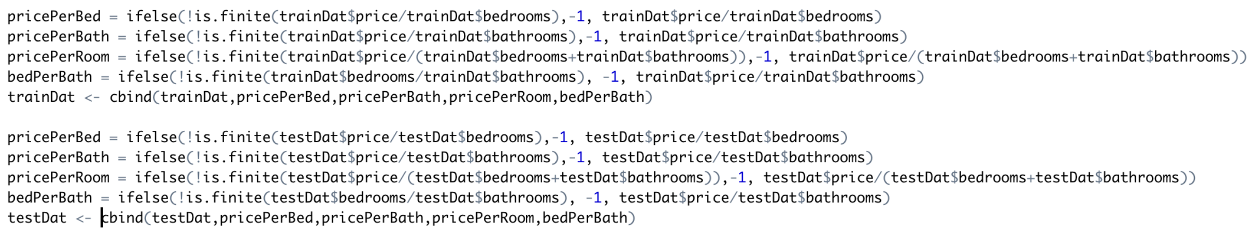
Support Vector Machine (SVM) is a very classical and efficient classification model. However, in the dataset, it provided the worst prediction result. The accuracy is 0.5752. The initial step is to set parameters, which are C and gamma. C is the penalty coefficient, namely the tolerance to error. The higher C means the less errors can be tolerated and easily overfitting. The smaller C is easier to underfit. If C is too large or too small, the generalization ability becomes poor and pruning tree with a suitable parameter. The smaller the gamma is, the more the support vectors are. The number of support vectors influences the speed of training and prediction. The best tuning parameters are C equal to 60 and gamma equal to 0.004



```
> A

pred  X1  X2  X3
  X1 297 155  70
  X2 218 417 238
  X3  65 191 555
> sum(diag(A)) / sum(A)
[1] 0.5752493
```

I also tried to add some meaningful variables into the both train and test data for all three models, Random Forest and SVM are improved. The new variables are price per bedroom, price per bathroom, and price per room and bedroom per bathroom. I have put the code for new variables below. Also, I add back the variable of address, and change it to factor first then to numeric. As the model change, the tuning parameter will also change. However, since running the tuning parameter of SVM takes 3 or 4 hours, I used C equal to 10 and gamma equal to 0.004. The score on Kaggle increased 0.01 for SVM method.



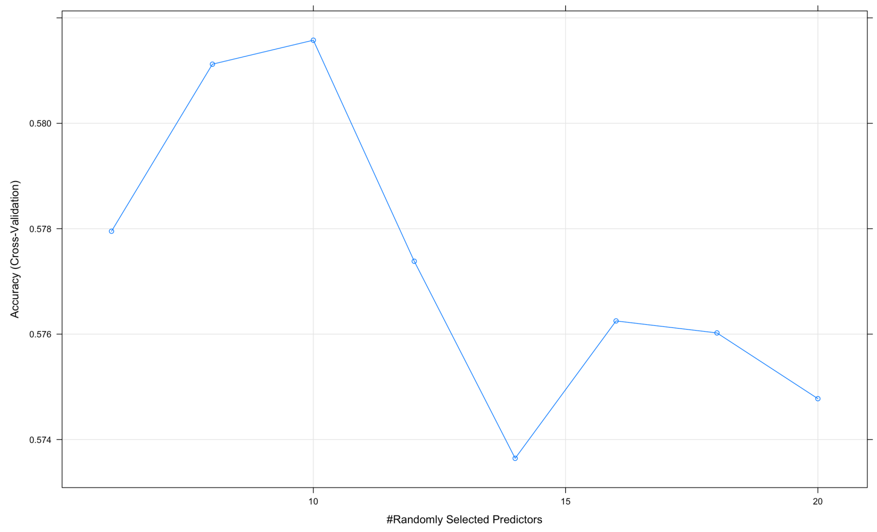
```
pricePerBed = ifelse(!is.finite(trainDat$price/trainDat$bedrooms),-1, trainDat$price/trainDat$bedrooms)
pricePerBath = ifelse(!is.finite(trainDat$price/trainDat$bathrooms),-1, trainDat$price/trainDat$bathrooms)
pricePerRoom = ifelse(!is.finite(trainDat$price/(trainDat$bedrooms+trainDat$bathrooms)),-1, trainDat$price/(trainDat$bedrooms+trainDat$bathrooms))
bedPerBath = ifelse(!is.finite(trainDat$bedrooms/trainDat$bathrooms), -1, trainDat$price/trainDat$bathrooms)
trainDat <- cbind(trainDat,pricePerBed,pricePerBath,pricePerRoom,bedPerBath)

pricePerBed = ifelse(!is.finite(testDat$price/testDat$bedrooms),-1, testDat$price/testDat$bedrooms)
pricePerBath = ifelse(!is.finite(testDat$price/testDat$bathrooms),-1, testDat$price/testDat$bathrooms)
pricePerRoom = ifelse(!is.finite(testDat$price/(testDat$bedrooms+testDat$bathrooms)),-1, testDat$price/(testDat$bedrooms+testDat$bathrooms))
bedPerBath = ifelse(!is.finite(testDat$bedrooms/testDat$bathrooms), -1, testDat$price/testDat$bathrooms)
testDat <- cbind(testDat,pricePerBed,pricePerBath,pricePerRoom,bedPerBath)
```

Random Forest

Decision tree is the basic component of Random Forest, and it is an intuitive model. We can think of the decision tree as a series of yes/no questions about the data, resulting in a category of predictions. Similar to the way humans classify: before we make a decision (in an ideal world), we would ask a series of questions about the available data. A Random Forest is a model composed of many decision trees. Rather than simply averaging the predictions of all the trees. Random Forests are achieved through the following two steps: bootstrap and Split-Variable Randomization. Bootstrap means each tree is bootstrapped with a different sample, making them slightly different and de-correlated, and in my random forest, I used cross-validation to randomly sampled. Split-Variable Randomization means randomly selecting m variables from the p variables at each split, and then picking the best split among them.

Since ranger is a fast implementation of a Random Forest algorithm and supports both classification and regression Random Forest. Besides the reasonable trainData created by the preprocessing step, there were two more parameters to care about for making the model for prediction. The number of trees was set to 500 which is the default. For mtry, which is the number of variables to possibly split at in each node. I use the caret package to find the best tuning parameter which is 10.



As I separated the training data set to 75 percent subtrain dataset and 25 percent subtest dataset, I got the overall accuracy is 0.5675 by using mtry equal to 10.

```
rf.cv.pre  X1  X2  X3
       X1 313 167  52
       X2 209 357 229
       X3  58 239 582
```

I also used new data set new variables to fit the Random Forest. Running the tuning parameter of random forest takes 3 or 4 hours even if I use the rangers. Hence, I used mtry equal to 10. The score on Kaggle increased 0.01 for Random Forest method.

Boosting

Boosting is the best model out of these three models and shows the best performance. First, the accuracy of this model is 0.5829556. Compared to the other models, Boosting has the highest score on kaggel, and is considered to be one of the best performing methods in statistical learning. Gradient Boosting is a boosting method which is Gradient Descent plus Boosting. Since it is a kind of boosting, it combines many weak learners (classification tree or regression tree) into a strong classifier. The main idea of Gradient Boosting is to set up gradient descent direction of the model loss function before setting up the model. Boosting the performance of a set of low variances, but high bias prediction rules by cleverly combining them.

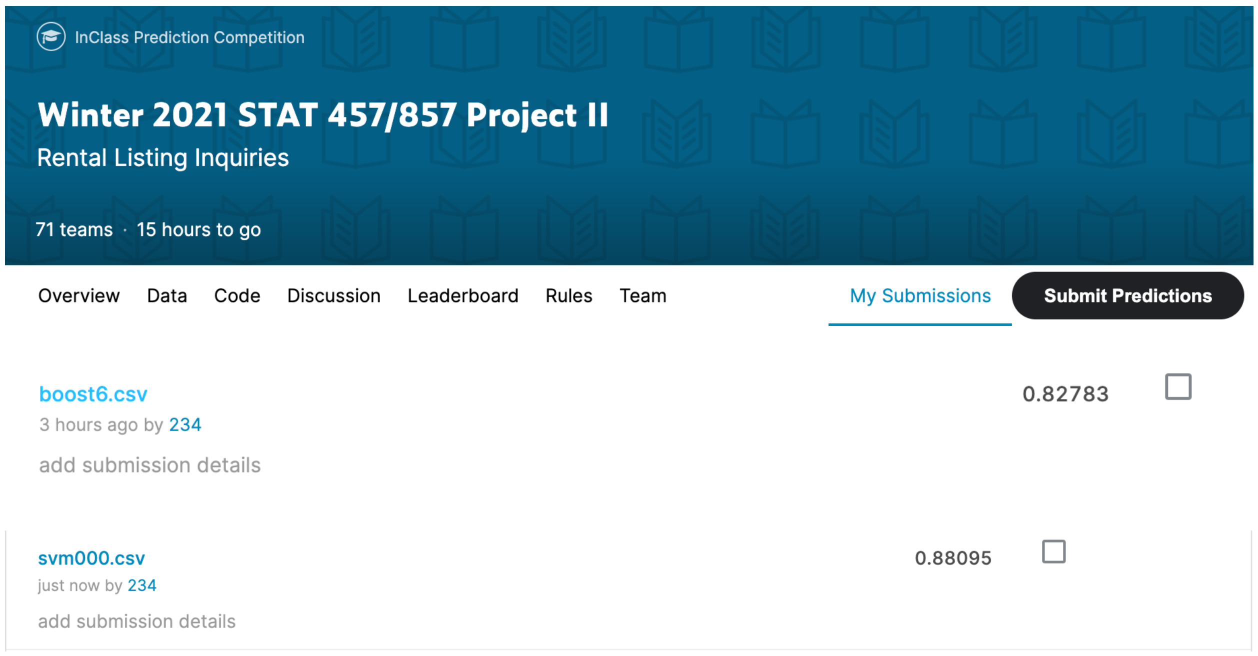
The package "caret" contains functions "train" with method 'gbm', which represents the algorithm of boosting, to set up the model with the best tuning parameter.

best tuning parameter is for the model were n.trees = 600, interaction.depth =4, shrinkage = 0.05 and n.minobsinnode = 10.

The prediction table of sub train and sub subtest.

```
gbm.pred   1   2   3
       1 307 136  40
       2 252 396 201
       3  54 237 583
```

Results (Score on Kaggle)



InClass Prediction Competition

Winter 2021 STAT 457/857 Project II

Rental Listing Inquiries

71 teams · 15 hours to go

boost6.csv
3 hours ago by 234
add submission details
0.82783

svm000.csv
just now by 234
add submission details
0.88095

rf000.csv
3 hours ago by 234
add submission details
0.87891 ☐

Discussion

In the project, the biggest problem was data processing, because The address of the properties is a significant factor in choosing a house. Since we were not allow to use “ggmap” to determine which rental house was most centrally located to the downtown area, or most convenient for living. Since this is not easy to determine by a string of address, I deleted this variable. Therefore, this is the biggest factor that force me to improve my model. My results support my conclusion which is that based on the the score on Kaggle, I can see that boosting gives the best score by the same data set and the same way of processing the data. In order to produce results which supported this, I considered multiple factors, and had to delete ones that would not assist me, which produced these unique set of results.

Reference

1. https://www.statista.com/statistics/187577/housing-units-occupied-by-renter-in-the-us-since-1975/
2. https://www.realpage.com/analytics/apartment-completions-spike-2020/