758T Airbnb Project Report
Instructor: Courtney Paulson
Group 4

1. Summary

Our project is going to solve a categorical prediction problem, which will help listing owners to know whether their house can get a high booking rate or not (the listing is popular or not). This information will assist listing owners to know which features are important or significant to get a high booking rate, then help them to decorate or remodel their house, change the price or other fees and other actions they should make. During this prediction process, we need to choose and transform the variables among 70 columns, select features and choose the best model and parameters to get highest accuracy on test data set, because the higher your accuracy, the more convincing your model are.

For our problem, it's a categorical prediction problem, so we tried all categorical prediction models to help us get the best result. After trying more than 6 models, we finally choose Xgboost model because of its high accuracy, then we can analyze the significant variables to give listing owners recommendations. In the model selection process, we have four steps to be done:

1) Analyze each column in data set, then figure out how we should clean or transform each column (fill nan value and garbled value, process typo error in text columns and so on).
2) Decide how to process text columns, for example, the amenities, we have two ways to process it: one is to transform it to dummy variables and another way is using a number to stand for how many amenities in the house. Then in our model selecting process, we will decide which processing method fits our different models well.
3) Try different models, adjust parameters and then compare the best accuracy of each model to choose the best one. For models without cross validation mechanism inside, we use cross validation average accuracy to evaluate its performance; for models with cross validation mechanism inside, we compare the accuracy directly. We tried several ensemble ways to check their performance is better or not, and finally we found the best model, Xgboost.
4) Based on the model, Xgboost, which gives us the best accuracy, we analyzed the features that Xgboost thinks are important for our prediction, then we give listing owners recommendations to help them get high booking rate.

After analyzing this data through Xgboost, we found latitude (importance rate 10.57%) and longitude (10.51%) are two features that have highest importance, which means the location is the most important feature in predicting high booking rate. The second important feature consist of two time variables: first_review (9.08%) and host_since (7.37%), which indicates the longer the house, the more likely it gets a high booking rate. Furthermore, availability, including availability_365(5.08%), availability_90(3.32%), availability_60(2.6%) and availability_30(2.6%), is an important feature as well. Additionally, to make a listing popular, the owner also needs to set a reasonable price (5.03%), cleaning fee (3.38%) and extra people fee (2.12%). Based on the above, our recommendations are as follows:

1) Location: Location is the most important factor. When a host decide to rent a listing on Airbnb, the first thing he should consider about is the location of the house (convenient traffic, near spotlights, near shopping mall or restaurants and so on.)
2) Time: If the host decides to rent listings on Airbnb, our recommendation is the sooner the better. The longer host history and the longer first review since indicate the more likely the listing is to be popular.
3) Availability: Availability, which stands for how many days the listing is available out of the next 30,60,90,365 days, is another important factor. If you want to gain a high booking rate, try to make your listing's availability as high as possible.
4) Price, cleaning fee and extra people fee: As hosts, if we try to charge above market price then renters will select more affordable alternatives. Setting a reasonable price is important for high booking rate as well.

The problem left during our analysis process is that we didn't do any reviews related text mining. However, some reviews may have great influence on improving our prediction model. For instance, listing description and interaction, we can assign labels to these two kind of reviews, then put more information to our model.

2. Data Processing: data cleaning

1) **Text variables processing:**

We won't deal with text mining during this analysis process, so we just delete text columns, including access, description, host_about, host_name, house_rules, interaction, name, neighborhood_overview, notes, space, street, summary, and transit.

2) **Categorical variables processing:**

- **Deleting repeated geographical columns**: We delete city, country_code, jurisdiction_names, market, neighbourhood, smart_location, state, host_location, host_neighbourhood and zip code. Because these variables serve the same function as city name, latitude and longitude. Latitude and longitude give us more accurate location information. City name is a clearer classification compared with jurisdiction_names, market, neighbourhood and so on. For geographical variables, we just keep these three.
- **Removing all columns with more than 99% missing values**: For experiences_offered and license, every value in this column is null, so we delete these two columns.
- **Replacing missing values by most frequent value**: We replace following columns: bed_type, cancellation_policy, host_has_profile_pic, host_identity_verified, host_is_superhost, instant_bookable, is_location_exact, require_guest_phone_verification, require_guest_profile_picture, requires_license, room_type, require_guest_phone_verification, require_guest_profile_picture, requires_license, room_type missing values by most frequent value.
- **Specific search**: There is only two missing values in city name, so we fill in these city names by searching missing values' corresponding latitude and longitude in the row.
- **Fill "No record" in columns with more than 50% missing values**: We fill "No record" in 'host_response_time' and 'is_business_travel_ready'
- **Unifying expressions in column**: For property_type, there are some duplicate expressions in the column, for example, bed and breakfast, bed & breakfast 'Bed & Breakfast', 'Bed and breakfast'. We unify these values by one same expression.
- **Create dummy variables for categorical variables**: after dealing with missing values and some wrong values, we create dummy variables for all categorical variables.

3) **Numerical variables processing:**

- **Removing all columns with more than 99% missing values**: For host_acceptance_rate, and square_feet, every value in this column is null, so we delete these two columns.
- **Standardize format** : 1)remove $ in extra_people, 2)delete % in host_response_rate
- **Fill missing values with mean value**: we fill accommodates, availability_30, availability_365, availability_60, availability_90, bathrooms, bedrooms, beds, extra_people, guests_included, host_listings_count, host_response_rate, host_total_listings_count, maximum_nights, minimum_nights and security_deposit.
- **Specific search**: For latitude and longitude, there is one missing value in the whole dataset, so we fill the missing value with its corresponding street's latitude and longitude.
- **Specific calculation**: for the missing values in weekly_price and monthly_price, we fill the missing value with price*7 and price*30.

4) **Date variables processing:**

- **convert date to 'how many days'**: For first_review and host_since these two variables, we convert them to how many days until 04/01/2018. Eg, we convert 03/01/2018 into 31days.

5) **Categorical list:**

- **Unifying expressions in column then getting dummy:** For amenities and host_verifications, firstly, we unify some unstandard expressions, eg, we only keep one between "firm matress" and "firm mattress", "free parking on street" and "free street parking". After all items are unified, we create dummy variables for each item in amenities and host_verifications.

3. Model

3.1. Model selection:

we split data into 80% and 20% as training and test data, and then we use cross validation onto the 80% data to train models. Afterwards we use test data to test the overall accuracy of our models.

3.1.1 Xgboost model

Among all algorithms, as an implementation of gradient boosted decision trees XGBoost is good at speed. We used this algorithm focusing on improving model's performance which is measured by prediction accuracy. Without feature selection, full features (310) were put into this model to get accuracy. One essential way to improve model's performance was to adjust parameter of XGBoost manually. Finally, we adjusted parameters like objective, max_depth, eta, gamma, colsample_bytree, min_child_weight, nrounds and get higher accuracy compared with default setting, which is from 84.6 to 85.2. (**For additional model details, see Appendix, Graph 1, pg. 9)**

XGBoost listed feature importance as below, we can tell that all these 238 features contribute to our prediction with importance from high to low. A higher value of this metric when compared to another feature implies it is more important for generating a prediction. Latitude, longitude and first_review were listed as top 3 which contribute to our prediction most. (**For additional model details, see Appendix, Graph 2, pg. 9)**

3.1.2 Gradient boost model

Like random forest as well as xgboost model, the gradient boosting is also an "ensemble" function based on TREEs, which is normally much more robust than KNN so that it can do a better job when you are trying to predict classification given a very large dataset. Therefore, we tried the gradient boosting model:

First, we didn't tune any parameters, just keeping defaults values without feature selection as well as cross validation to train a model on our training dataset to see if it could give a better accuracy.

```
> table(y_test,pred)
        pred
y_test     0     1
     0 14508   617
     1  3359  1512
```

It returned 80% accuracy score, which is much better than KNN even after tuning K and higher than our baseline 74.86%.

Therefore, we decided to focus more on training GBM by trying many alternatives: tuning parameters, feature selection as well as cross validation. We tried many combinations to trying to get the highest accuracy in training dataset:

- Feature selection: According to selected features we did later, we put them into the GBM model to see if the result could improve. When we picked them out and put them into GBM model, the accuracy did improve a little bit (from 84.54% to 84.89%), so we guessed it might because selected significant features mostly based on xgboost and random forests, which also belong to tree family, the distribution of features importance is similar.
- Tuning parameters: three main parameters: n.trees, interaction.depth and shrinkage and we got the highest accuracy on test data ended up with 85.1% when shrinkage=0.15, n.trees=1500 and interaction.depth=5.
- Cross validation: we used cross validation to train model further. The accuracy went down slightly to about 84.91%, but it makes sense. (**For additional model details, see Appendix, Graph 3 and 4, pg. 10-11)**

3.1.3 Random forest model

We know that random forest is also an "ensemble" function based on TREEs. We are now well aware of the overfitting problems with decision trees. But if we grow a whole lot of them and have them vote on the outcome, we can get passed this limitation. Each of these trees make their classification decisions based on different variables.

To keep the result really random, Random Forests do this in two ways:

- The first trick is to use bagging, for bootstrap aggregating. Bagging takes a randomized sample of the rows in your training set.
- The second source of randomness gets past this limitation though. Instead of looking at the entire pool of available variables, Random Forests take only a subset of them, typically the square root of the number available.

Hence, we did not use cross validation in random forest. When we are training our model, we want to keep our random forest as random as possible. The arguments that are mostly related with these features are: **ntree** and **mtry.**

after we tried different combinations of these arguments, we found out that with ntree=500 and with mtry = 16 will provide us 84% accuracy on our test data. This accuracy score is among the highest of all models. So we do recommend to use random forest to do the prediction.

3.1.4 Classification Tree

Decision tree is a type of supervised learning algorithm that is mostly used in classification problems. It works for both categorical and continuous input and output variables. In this project, we used classification tree to do prediction for binary prediction, whether one case is high_booking_rate(1) or not(0). For our classification tree model, full features were used, the value (class) obtained by terminal node in the training data is the mode of observations falling in that region.

One issue of classification tree is overfitting, it's pivotal to avoid this and we did with two ways, setting constraints on tree size and tree pruning. We set parameter constraints for our model (minsplit and Complexity Parameter) and did cross validation(5). We used xerror helping us choose where to prune the tree. Each row represents a different height of the tree. In general, more levels in the tree mean that it has lower classification error on the training. However, we have the risk of overfitting. This method helped us with avoiding such situation.

Finally, we got accuracy of this model for prediction, 74.38%, which doesn't perform well as other models like random forest, XGBoost and Gradient Boost. **(For additional model details, see Appendix, Graph 5 & 6, pg. 11)**

3.1.5 Logistic

In this Airbnb case, our problem is a traditional classification problem (high booking rate is either 1 or 0), so the first model came into our mind is logistic regression, which is also the first classification model we learnt during this semester. First, we put our partitioned training data with all features into our feature selection model, the first several lines of our logistic coefficients are in Appendix: **(For additional model details, see Appendix, Graph 7, pg. 12)**

As the output shows, some features such as availability_30, availability_60, cleaning_fee and extra_people have a negative relationship with high_booking_rate, meaning that the shorter the availability is, the less possible the room is going to get booked and the higher the cleaning fee is, the less possible the room is going to get booked. Conversely, availability_360 and guests_included have positive relationships with high_booking_rate, indicating the longer the availability is the more possible the room is going to get booked and the more guests included in the price the more possible the room is going to get booked. Later we used some feature selection methods, and these features with very low p-value in the logistic models were almost all selected in our final selected features. However, it should be noted that with either selected features or all features, our team discovered an accuracy of around 81% on our validation data.

```
> table(ab_test$high_booking_rate,log_class)
   log_class
        0     1
  0 13886  1240
  1  2624  2247
```

Later we used cross validation and acquired a 79% accuracy. Although this was higher than the validation baseline of 74%(majority vote), it was still lower than other models we tested, so ultimately, we do not recommend a logistic regression model for this problem.

3.1.6 KNN

Before I trained KNN model, I estimated the result would not be high since we know nearest neighbor methods can work poorly when the 'dimensionality is large' and we have 310 variables! Variables in KNN need to be scaled and converted into numerical datatype. And cross validation is recommended when training KNN.
First, I tried to use train function in caret because it can find the best K automatically. However, it didn't work because our dataset is so large and we have too many variables., Even after I selected features (use 238 significant variables) and also cross validation takes a lot computational power!

Then, I went back to use what we learned in class. When the K=5, which is the default k, I only got 73.2%. After tuning k several times and using selected features to reduce dimensions, I got the highest accuracy 78.72% when k=7. Although this was higher than the validation baseline of 74.86 %, it was much lower than other models we tested, and we should not consider it. (**For additional model details, see Appendix, Graph 8 & 9, pg. 12.)**

3.1.7 Adaboost

According to machine learning method, AdaBoost is one of the best used to boost the performance of decision trees on binary classification problems, so we decide to try Adaboost.

The most important parameters in Adaboost are base_estimator, n_estimators, and learning_rate. First, we try the default setting in Adaboost algorithm: base_estimator: default=DecisionTreeClassifier, n_estimators : default=50, learning_rate: default=1 then we get an accuracy on testing dataset: 81.41%. Then we try to adjust parameters to improve model performance. Firstly, base_estimator is the learning algorithm to use to train the weak models. This will almost always not needed to be changed because by far the most common learner to use with AdaBoost is a decision tree. Secondly, we set learning rate fixed and try to adjust n_estimators. After several attempts, we get a better result with n_estimator=500, and the accuracy equals to 82.01% . Then we set n_estimator=500 fixed and try to adjust learning_rate, Reducing the learning rate will mean the weights will be decreased to a small degree, forcing the model train slower (but sometimes resulting in better performance scores). After several attempts, we get a better result with learning_rate=0.3, and the accuracy equals to 82.35%. Keeping increasing n_estimator and decreasing learning_rate only have very very tiny increase in accuracy (**For additional model details, see Appendix, Graph 10, pg.12).** However, the accuracy is still lower than Xgboost, Gradient boost and Random forest, so we don't consider Adaboost in the later analysis.

3.2 Model optimization

3.2.1 Feature Selection

To try to find most relevant features to the predictive modeling problem we are working on to improve models' performance, we used feature selection, which helps us get better accuracy with requiring less data. Fewer attributes is desirable because it reduces the complexity of the model, and a simpler model is simpler to understand and explain.

We used several methods to do feature selection, domain knowledge, removing highly correlated features, selecting features with LASSO and getting feature importance based on Random Forest and XGBoost algorithms.

- **Domain Knowledge:** include jurisdiction_names, neighbourhood, host_name, name, neighborhood_overview.
- **Remove Redundant Features:** We used Caret package which is provided by R, it produced the findCorrelation which will analyze a correlation matrix of our data's attributes report on attributes that can be removed.

In total, 4 features are removed including accomdate, availability_60, availability_90 and host_total_listings_count. (**For additional model details, see Appendix, Graph 11, pg.13)**

- **Select Variables with LASSO:** We tried to find features which contribute most to the accuracy of the model while our model is being created, regularization algorithms like LASSO is commonly used. We did LASSO regression based on logistic algorithm, 20 features were selected and the other features with coefficients being 0 have been dropped out of the model. (**For additional model details, see Appendix, Graph 12, pg.13)**
- **Compute Feature Importance with Random Forest:** Not all data attributes are created equal. More is not always better when it comes to attributes or columns in dataset. In our dataset, 311 full features work with different importances with our model, methods that use ensembles of decision trees (like Random Forest or XGBoost) can compute the relative importance of each attribute. We can use information from feature selection methods to create filtered versions of our dataset and increase the accuracy of our models. We selected features with importance more than 0 and got 158 variables. **(For additional model details, see Appendix, Graph 13, pg.14)**
- **Compute Feature Importance with XGBoost:** A benefit of using ensembles of decision tree methods like gradient boosting is that they can automatically provide estimates of feature importance from a trained predictive model. Since XGBoost is very popular and can give us better result in explaining our dataset compared to other models, we estimated features importance calculated by using the Xgboost: Firstly, we trained XgBoost model on our entire datasetand then we sorted the importance values of all features. Then we selected variables whose values are greater than 0 and we got 238 significant variables. (**For additional model details, see Appendix, Graph 2, pg.9)**

Finally, we combined all features retrieved from these methods and got 236 features in total. These features were applied to models we selected in the first step but accuracy performance of all these models with current features are not as good as ones with full features.

3.2.2 Ensemble

From above, we tried different algorithms to do prediction and did models selection based on prediction accuracy. Under this, another way to produces more accurate solutions is to combine models with great performances with our dataset, which is ensemble method. In this case, we tried to combine multiple models which were created before to produce improved results.

- **Stacking:** We chose stacking, one of ensemble methods, to combine multiple models, XGBoost, AdaBoost, Random Forest and Gradient Boosting, all with great accuracy performance before. Our basic idea is to train machine learning algorithms with training dataset and then generate a new dataset, Random Forest, AdaBoost and Gradient Boosting were used as a first-level (base) and then used XGBoost model at the second-level to predict the output from the earlier first-level predictions. All algorithm used the same parameters generated from model selection part.

With random forest, ada bosst and gradient boost, we obtained our first-level predictions, which could be regarded as essentially building a new set of features to be used as training data for the next classifier. Then we used XGBoost and fit it to the first-level train and target data and use the learned model to predict the test data. The accuracy of this stacking method is 84.26, it didn't perform better than the single model, XGBoost, which performed best when predicting. (**For additional model details, see Appendix, Graph 14, pg. 14)**

- **3 in 1 model:** This idea first comes from an in-class game 'Who wants to be a Millionaire', during which we can vote the correct answer for PK, and we learned that this method will give the participant highest chance to get a correct answer among three help options. At first, we want to use a voting mechanism that combines several predictions from different models, then if most of the model predict 1, we predict it 1, otherwise predict it 0. After we learned Neural Network, we updated our idea, and we will pick 3 models which have highest accuracy on validation dataset. Then we use these three predictions as input to the fourth model to predict the results, thus, we use this Neural Network-like model to replace our voting idea. What we do in this model is four steps below:
  1) Split our data into two parts, one part is for choosing and training the first three models, another part is for choosing and training the fourth single model and checking the final accuracy of our 3 in1 model.

  2)Split the first part data into train and validation two parts, then use the accuracy on validation data to compare models and adjust parameters of models for higher accuracy. After having tried 11 models (Logistic, KNN, other tree-based models and so on), we found 3 models (RandomForest, Gradient boost and Xgboost) that have highest accuracy.

  3)Split the second part of data into train and validation, then use the three well trained models to predict the results and use these three predicted results as input to train the fourth model, finally use validation data to choose model and adjust parameters of the fourth model.

  4)Use all four trained models to work on our validation data to get the final prediction, and the final accuracy on validation data is 84.7% which is close to its performance on professor's test data set.

  (**For additional model details, see Attached File, 3in1.html)**

Appendix

Appendix (1)

Accuracy list

model	Best accuracy(%)
XGB	85.2
Gradient boost	84.91
3in1	84.54
Random Forest	83.9
adaboost	82.35
Logistic	79
KNN	78.72
Classification Tree	74.38

Appendix (2)

Group Member Roles:

1. Data Processing:

data cleaning (Qianli Cheng, Yujing Gao)

2. Model Selection:

1. Xgboost(He Su)
2. Gradient boost(Xin Gu)
3. Random forest(Qianli Cheng)
4. Adaboost(Yujing Gao)
5. Classification tree(He Su)
6. Logistic(Qianli Cheng)
7. Knn(Xin Gu)

3. Model Optimization:

i. Feature selection
- Delete highly correlated columns(He Su)
- Lasso(He Su)
- Random forest feature importance(He Su)
- Xgboost(Xin Gu)
- Gradient boost(Xin Gu)

ii. Ensemble
- Stacking(He Su)
- 3 in One(Guanyu Chen)

- ## Appendix (3) code

Please check out the attached code files. The following file names describe the tasks completed in each file.

1. Data Cleaning.html --- a html file which contains all code and results for cleaning both the training and testing datasets.
2. Classification Tree+XGBoost_Model.R--- An R script file containing code for building Classification Tree and XGboost modeling for the problem.
3. gradientboost+Knn_model.R-- An R script file containing code for building gradient boost and KNN models for the problem.
4. logistic and random forest.R-- An R script file containing code for building models for the problem.
5. Adaboost.html --- a Jupyter Notebook file of ada boost model
6. select_feaurel_grdientboost+xgboost.ipynb-- a Jupyter Notebook file which contains all code for features importance from gradient boost and xgboost
7. 3 in 1.html – Python code to train, tune and test the model.
8. Select Feature Random Forest.html-- a html file which contains codes for feature selection using Random Forest
9. Ensembling Stacking Method.html-- a html file which contains codes for stacking model
10. Select Features(LASSO+Correlation).R-- An R script file containing code for feature selection
11. Los Angeles.jpg & New York. Jpg – the distributions of houses with the high booking rate in two cities so that host can know the most popular location in that areas (we made after mock interview, just for fun!)

- **Appendix (4) Graph**

```
# Set our hyperparameters
param <- list(objective   = "binary:logistic",
              max_depth   = 4,
              eta         = 1,
              gammma      = 0,
              colsample_bytree = 1,
              min_child_weight = 6)

set.seed(1234)

# Pass in our hyperparameteres and train the model
system.time(xgb <- xgboost(params  = param,
                           data    = dtrain,
                           label   = train.label,
                           nrounds = 3000,
                           print_every_n = 100,
                           verbose = 1))
```

Graph 1

	Features	Importance
1	latitude	0.1057
2	longitude	0.105127
3	first_review	0.090807
4	host_since	0.073718
5	availability_365	0.050892
6	price	0.050318
7	cleaning_fee	0.033853
8	availability_90	0.033279
9	availability_60	0.026394
10	availability_30	0.026369
11	maximum_nights	0.023475
12	host_listings_count	0.022577
13	extra_people	0.021205
14	host_response_rate	0.016116
15	minimum_nights	0.015866
16	accommodates	0.012873
17	guests_included	0.011076
18	beds	0.008332
19	bedrooms	0.006536
20	bathrooms	0.006312
21	instant_bookable_t	0.005663
22	host_is_superhost_t	0.005289
23	washer	0.004341
24	cable tv	0.004166
25	first aid kit	0.004041
26	family/kid friendly	0.003992
27	facebook	0.003867
28	cancellation_policy_strict	0.003767
29	room_type_Private room	0.003742
30	fire extinguisher	0.003692

Graph 2(1)

31	host_response_time_with	0.003642
32	free parking on premises	0.003642
33	cancellation_policy_mode	0.003617
34	jumio	0.003592
35	tv	0.003443
36	is_location_exact_t	0.003393
37	24-hour check-in	0.003193
38	lock on bedroom door	0.003193
39	carbon monoxide detecto	0.003168
40	kba	0.003168
41	property_type_apartment	0.003118
42	safety card	0.003093
43	laptop friendly workspace	0.003068
44	hair dryer	0.003068
45	air conditioning	0.002944
46	work_email	0.002944
47	pets live on this property	0.002944
48	shampoo	0.002919
49	iron	0.002919
50	host_identity_verified_t	0.002844
51	property_type_house	0.002844
52	private entrance	0.002819
53	self check-in	0.002794
54	hangers	0.002669
55	pets allowed	0.002594
56	buzzer/wireless intercom	0.002445
57	is_business_travel_ready_	0.002445
58	breakfast	0.002445
59	lockbox	0.002395
60	indoor fireplace	0.002295
61	host_response_time_with	0.00227

...

Graph 2(2)

```
##train gbm model
##default :
GBM <- train(high_booking_rate ~., data=train,
             method="gbm")
# Create our prediction probabilities
preds<-predict(GBM,X_test)

#use cutoff=0.5 to classify it
pred<-ifelse(preds >=0.5, 1,0)

# Confusion matrix
table(y_test,pred)


# get the accuracy score
prediction <- ifelse(pred != test$high_booking_rate, "wrong", "correct")
round(sum(prediction == "correct")/(sum(prediction =="correct")+sum(prediction =="wrong")), digits = 2)
```

Graph 3

```
# Load the gradientboost package
library(gbm)
# Pass in our hyperparameteres and train the model
ctrl <- trainControl(method = "cv",
                     number = 5,
                     summaryFunction=twoClassSummary,
                     classProbs=TRUE,
                     allowParallel = TRUE)

grid <- expand.grid(interaction.depth=5,
                    n.trees=1500,
                    shrinkage=0.15,
                    n.minobsinnode = 20)

GBM <- train(high_booking_rate ~., data=train,
             method="gbm",
             trControl = ctrl,
             tuneGrid=grid,
             train.fraction = 0.5,
             tuneLength=3)
```

Graph 4

```
library(caret)
t.cont<-rpart.control(minsplit=2,cp=0.5,xval=5)
mtree  <- rpart(high_booking_rate~ .,data=airbnb_train,method = "class",control=t.cont)
best.cp<-mtree$cptable[which.min(mtree$cptable[,'xerror']),'CP']
pruned_mtree<-prune.rpart(mtree,cp=best.cp)
```

Graph 5

```
          Reference
Prediction     0     1
         0 14874  5122
         1     0     0

               Accuracy : 0.7438
                 95% CI : (0.7377, 0.7499)
    No Information Rate : 0.7438
    P-Value [Acc > NIR] : 0.5038

                  Kappa : 0
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 1.0000
            Specificity : 0.0000
         Pos Pred Value : 0.7438
         Neg Pred Value :    NaN
             Prevalence : 0.7438
         Detection Rate : 0.7438
   Detection Prevalence : 1.0000
      Balanced Accuracy : 0.5000

       'Positive' Class : 0
```

Graph 6

```
Call:
glm(formula = ab_train$high_booking_rate ~ ., family = "binomial",
    data = ab_train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.5689  -0.6936  -0.2848   0.4872   8.4904

Coefficients: (15 not defined because of singularities)
                                  Estimate Std. Error z value Pr(>|z|)
(Intercept)                     -8.800e+01  8.828e+02  -0.100 0.920596
accommodates                     1.135e-01  9.782e-03  11.605  < 2e-16 ***
availability_30                 -3.457e-02  2.577e-03 -13.416  < 2e-16 ***
availability_365                 7.418e-04  9.682e-05   7.661 1.84e-14 ***
availability_60                 -1.534e-03  2.544e-03  -0.603 0.546447
availability_90                  6.630e-03  1.316e-03   5.036 4.74e-07 ***
bathrooms                       -5.549e-03  2.298e-02  -0.241 0.809189
bedrooms                        -2.142e-01  1.966e-02 -10.894  < 2e-16 ***
beds                             4.562e-02  1.324e-02   3.446 0.000568 ***
cleaning_fee                    -9.088e-03  3.270e-04 -27.792  < 2e-16 ***
extra_people                    -3.223e-03  5.306e-04  -6.075 1.24e-09 ***
first_review                    -1.471e-04  2.949e-05  -4.986 6.17e-07 ***
guests_included                  8.774e-02  9.004e-03   9.744  < 2e-16 ***
host_listings_count             -7.889e-03  7.318e-04 -10.780  < 2e-16 ***
```

Graph 7

```
## other method: find the best K automatically  using caret pakage
ctrl <- trainControl(method = "repeatedcv", number = 5, repeats = 3)
knnFit <- train(X_train,y_train, method = "knn",trControl = ctrl,tuneLength = 5)
test_pred <- predict(knn_Fit, newdata = X_test)
```

Graph 8

```
##try k=10
#for training data
knn.pred_train10=knn(X_train,X_train,y_train,k=10)
train_acc10=sum(ifelse(knn.pred_train1==y_train,1,0))/nrow(ab_train)
#for testing data
knn.pred_test10=knn(X_train,X_test,y_train,k=10)
# accuracy on test
test_acc10=sum(ifelse(knn.pred_test7==y_test,1,0))/nrow(test)
test_acc10
#76.38%
```

Graph 9

```
%%time
from sklearn.ensemble import AdaBoostClassifier
ac = AdaBoostClassifier(learning_rate=0.3,n_estimators=500)
ac.fit(X_train, y_train)
y_pred3 = ac.predict(X_test)
```

```
CPU times: user 8min 7s, sys: 3.57 s, total: 8min 11s
Wall time: 8min 17s
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred3)
```

```
0.82351470294058815
```

Graph 10

```r
## This Part aims to select features:
##Calculate Correlation; Lasso; Stepwise Backward Elimination

## Remove Redundant Features
# load the data
data(Train_XY_AllNumeric_4.26.csv)

# ensure the results are repeatable
set.seed(11217)
# load the library
install.packages("mblench")
install.packages("caret")
library(mlbench)
library(caret)

# calculate correlation matrix
correlationMatrix <- cor(Train_XY_AllNumeric_4_26[,2:22])
# summarize the correlation matrix
print(correlationMatrix)
# find attributes that are highly corrected (ideally >0.75)
highlyCorrelated <- findCorrelation(correlationMatrix, cutoff=0.75)
# print indexes of highly correlated attributes
print(highlyCorrelated)
```

Graph 11

```
> variables
 [1] "(Intercept)"                              "bathrooms"
 [3] "bedrooms"                                 "cleaning_fee"
 [5] "host_listings_count"                      "host_since"
 [7] "host_total_listings_count"                "price"
 [9] "carbon monoxide detector"                 "hair dryer"
[11] "hangers"                                  "hot water"
[13] "iron"                                     "refrigerator"
[15] "self check-in"                            "shampoo"
[17] "washer"                                   "host_is_superhost_t"
[19] "host_response_time_within a few hours"    "host_response_time_within an hour"
[21] "instant_bookable_t"                       "requires_license_t"
```

Graph 12

```
In [7]: importances = importances.loc[importances.importance > 0]
        with pd.option_context('display.max_rows', None, 'display.max_columns', No
            print(importances)
```

feature	importance
availability_30	0.047
host_response_time_within an hour	0.044
availability_60	0.043
availability_90	0.037
minimum_nights	0.037
availability_365	0.036
longitude	0.035
first_review	0.034
host_since	0.034
price	0.033
latitude	0.033
cleaning_fee	0.033
host_response_rate	0.023
host_listings_count	0.018
host_is_superhost_t	0.017
instant_bookable_t	0.017
extra_people	0.017
host_total_listings_count	0.016
accommodates	0.015
maximum_nights	0.014
bedrooms	0.012
guests_included	0.011
hair dryer	0.011
beds	0.010
bathrooms	0.008
host_response_time_within a day	0.006
washer	0.006

Graph 13

```
In [17]: from sklearn.metrics import accuracy_score
         accuracy_score(y_test,predictions)

Out[17]: 0.84862729409411408
```

Graph 14

```
[4]: gbm = GradientBoostingClassifier(learning_rate=0.15, n_estimat
     gbm.fit(X, y)
     imp = pd.DataFrame(gbm.feature_importances_ ,columns = ['Impor
     imp = imp.sort_values(['Importance'], ascending = False)

     print(imp)
```

	Importance
first_review	6.133580e-02
latitude	5.626515e-02
longitude	5.023954e-02
host_since	4.474989e-02
availability_365	4.038267e-02
price	3.837835e-02
availability_90	3.404949e-02
availability_30	3.150157e-02
availability_60	3.122780e-02
cleaning_fee	2.778468e-02
maximum_nights	2.133580e-02
extra_people	1.707271e-02
host_total_listings_count	1.687634e-02
minimum_nights	1.642128e-02
host_listings_count	1.594683e-02
host_response_rate	1.475804e-02
accommodates	1.398749e-02
guests_included	1.230390e-02
bedrooms	1.017654e-02
beds	1.017438e-02
host_is_superhost_t	8.033379e-03
bathrooms	7.911869e-03
instant_bookable_t	6.860825e-03
host_response_time_within an hour	6.713320e-03
room_type_Private room	5.415619e-03
washer	5.357790e-03
lockbox	5.252365e-03

Graph 15