Report (Airbnb Rental Price Prediction)

Introduction

This project is basically the process of making a prediction for the price of Airbnb rentals in New York City. By providing a data set called "analysisData", which lists over 25,000 Airbnb rentals information including 90 variables like price, id, accommodates, bed type, etc., participants are required to explore these data, clean and arrange the useful data, construct efficient models, and apply models to the test data which is called "scoringData" to make the prediction for price. The final goal is to generate a dataset contains the id number of test data and prediction of price, and the accuracy is assessed by Root Mean Square Error (RMSE).

Exploring the Data

Before using the data provided, the action needed is to explore the data in order to remove the useless variables from data. To better understand the types and properties of the “analysisData”, I constructed a new data frame which shows type of data, whether it is a factor, number of levels, whether it is a missing value (NA), and the number of NAs for each variable.

```r
# Examing data -----------------------------------------------------------
colNames = colnames(analysisData)
colDataType = sapply(analysisData, class)
isColFactor = sapply(analysisData, is.factor)
numOfLevels = sapply(analysisData, function(x) length(levels(x)))
isNAincluded = sapply(analysisData, function(x) any(is.na(x)))
numOfNAs = sapply(analysisData, function(x) sum(is.na(x)))

colSummary = data.frame(colDataType, isColFactor, numOfLevels, isNAincluded,
numOfNAs)
str(colSummary)
```

```
## 'data.frame':    96 obs. of  5 variables:
##  $ colDataType : Factor w/ 4 levels "factor","integer",..: 2 1 4 1 1 1 1 1
1 1 ...
##  $ isColFactor : logi  FALSE TRUE FALSE TRUE TRUE TRUE ...
##  $ numOfLevels : int  0 29142 0 3 28770 27497 21254 28805 1 17634 ...
##  $ isNAincluded: logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ numOfNAs    : int  0 0 0 0 0 0 0 0 0 0 ...
```

Preparing the Data for Analysis

Before using the data from "analysisData", it is better to manually prepare the data. This will basically shrink the data set and increase the efficiency for R to run algorithms in in terms of the models that would be used for predicting. By looking at the features of the variables in original data sets, I found there are two variables that may be useful for prediction but was not presented in a measurable way, which are "amentities" and "host_verifications". Therefore, I change they to numeric format by counting the item numbers in each cell.

```
##                                    host_verifications
## 1 ['email', 'phone', 'facebook', 'google', 'reviews']
## 2                ['email', 'phone', 'reviews', 'kba']
## 3    ['email', 'phone', 'facebook', 'reviews', 'kba']
## 4           ['email', 'phone', 'facebook', 'reviews']
## 5                ['email', 'phone', 'reviews', 'kba']
## 6                ['email', 'phone', 'reviews', 'kba']
##   host_verifications_changed
## 1                          5
## 2                          4
## 3                          5
## 4                          4
## 5                          4
## 6                          4
```

The initial selection of data is to remove some useless variables from the data First of all, I removed the columns which is factor but only have 1 level, because this type of variable will not affect the result of prediction since there is only one category. Next, I removed the factorial variables which has more than 600 levels since many regression models cannot handle variable which contains too many categories. Then I removed variables that contains more than 12,000 missing values, because 12,000 is almost the half number of rows in train data set and they will decrease the accuracy of prediction.

```
# Remove useless columns
colSelect = colSummary[which(numOfLevels != 1 & numOfLevels < 600 & numOfNAs
< 12000),]
analysisDataSelected = analysisData_changed %>% select(rownames(colSelect))
scoringDataSelected = scoringData_changed %>%
select(setdiff(rownames(colSelect), 'price'))
```

The next step is to fill the missing values. Since in factorial variables, missing value are automatically regarded as a factor, there is no need to replace them. Now let's move attention to numerical variables. There are three numerical variables contains missing values, which respectively are "bed", "security_deposit" and "cleaning_fee". I replace the NAs in "bed" with median of the all the non-NA values in the same column. But for the "security_deposit" and "cleaning_fee", I replace the NAs with 0 since I assume the missing value means these specific rentals does not have security deposit and cleaning fee.

```
#replace missing values (NAs)
analysisDataSelected[is.na(analysisDataSelected$beds),"beds"] =
median(analysisDataSelected[,"beds"], na.rm = TRUE)
scoringDataSelected[is.na(scoringDataSelected$beds),"beds"] =
median(scoringDataSelected[,"beds"], na.rm = TRUE)

analysisDataSelected[is.na(analysisDataSelected$security_deposit),"security_d
eposit"] = 0
scoringDataSelected[is.na(scoringDataSelected$security_deposit),"security_dep
osit"] = 0
```

```
analysisDataSelected[is.na(analysisDataSelected$cleaning_fee),"cleaning_fee"]
= 0
scoringDataSelected[is.na(scoringDataSelected$cleaning_fee),"cleaning_fee"] =
0
```

Modeling Techniques

After the data was ideally cleaned, now it is the most essential part in the whole project, which is fitting data into models. Before using model, we need to determine which variables are going to be used as indicators for the model. Here I used Lasso as a feature selection method to list the reduced number of explanatory variables to describe a response variable, which is price. The advantage of Lasso is that it performs better than the other selection, performs variable selection to any number of variables, and it is relatively fast. Feature selection also can reduce overfitting of model. Following are the variables came from Lasso:

```
##Feature Selection, Lasso
library(glmnet)
x = model.matrix(price~., data = analysisDataSelected)
y = analysisDataSelected$price
set.seed(100)
cv.lasso = cv.glmnet(x,y,alpha=1,nfolds=10)
coef(cv.lasso)

coefs = coef(cv.lasso, s = 'lambda.min')
#coefs
index = which(coefs != 0)
variables = row.names(coefs)[index]
variables = variables[!(variables %in% '(Intercept)')]; variables

##  [1] "id"                                  "host_id"
##  [3] "host_response_time"                  "host_response_rate"
##  [5] "host_is_superhost"                   "host_neighbourhood"
##  [7] "host_listings_count"            "host_total_listings_count"
##  [9] "host_verifications"             "host_has_profile_pic"
## [11] "host_identity_verified"         "street"
## [13] "neighbourhood"                  "neighbourhood_cleansed"
## [15] "neighbourhood_group_cleansed"   "city"
## [17] "zipcode"                        "market"
## [19] "smart_location"                 "longitude"
## [21] "is_location_exact"              "property_type"
## [23] "room_type"                      "accommodates"
## [25] "bathrooms"                      "bedrooms"
## [27] "beds"                           "bed_type"
## [29] "amenities"                      "security_deposit"
## [31] "cleaning_fee"                   "guests_included"
## [33] "minimum_nights"                 "calendar_updated"
## [35] "availability_30"                "availability_90"
## [37] "availability_365"               "calendar_last_scraped"
## [39] "number_of_reviews"              "review_scores_rating"
## [41] "review_scores_accuracy"         "review_scores_cleanliness"
## [43] "review_scores_checkin"          "review_scores_communication"
## [45] "review_scores_location"         "review_scores_value"
## [47] "jurisdiction_names"             "is_business_travel_ready"
## [49] "cancellation_policy"            "require_guest_phone_verification"
## [51] "calculated_host_listings_count" "reviews_per_month"
```

Then I used random forest as a decision tree model to train the data instead of simple linear regression model, since it not only can be used for both classification and regression problems, but also reduces overfitting and reduce variance by using multiple trees, which would essentially increase the accuracy of results. Since only the categorical variables with less than 53 categories can be applied to random forest model, I manually removed the variables that don't satisfy the requirement of the model. Finally, the model was constructed below:

```r
library(randomForest)
#construct random forest model
```

```
forest = randomForest(price ~
host_response_time+host_is_superhost+host_listings_count+host_total_listings_
count+host_verifications+host_has_profile_pic+host_identity_verified+neighbou
rhood_group_cleansed+market+longitude+is_location_exact+property_type+room_ty
pe+accommodates+bathrooms+bedrooms+beds+bed_type+amenities+security_deposit+c
leaning_fee+guests_included+minimum_nights+availability_30+availability_90+av
ailability_365+calendar_last_scraped+number_of_reviews+review_scores_rating+r
eview_scores_accuracy+review_scores_cleanliness+review_scores_checkin+review_
scores_communication+review_scores_location+review_scores_value+jurisdiction_
names+is_business_travel_ready+cancellation_policy+require_guest_phone_verifi
cation+calculated_host_listings_count+reviews_per_month,
                      data = analysisDataSelected, ntree = 1000)
```

Results

After the random forest model is successfully trained, it is able to use the random forest model to predict the prices based on the test data. But there is still an issue, which is some categorical variables in test data contains the new factors that the train data does not have. So, I needed to set the levels of test data and make them equal to the levels of train data.

```
#set level of test data
for (k in colnames(analysisDataSelected)) {
  if (class(scoringDataSelected[[k]]) == "factor") {
    levels(scoringDataSelected[[k]]) = levels(analysisDataSelected[[k]])
  }
}
```

Finally, the prediction was made. The table constructed below shows the RMSE of the train data which is calculated by using the prediction based on the train data set, and the RMSE of the test data which is evaluated by Kaggle competition website. The reason why I calculated RMSE of the train data is that I always use the RMSE of the train as a reference to the RMSE provided by Kaggle in each attempt I tried for a new method.

```
##      rmse_train rmse_test(results from Kaggle)
## rmse   53.56432                       51.88588
```

Discussion

Looking backward to the whole process of doing this interesting competition, it is fun to have every attempt that tried to make the prediction as accurate I can. Especially when every time I tried a model or a new way to select variables, it is always appreciated to see the improvements. Following tables show the scores of each attempt I got and the method I used for each attempt. I want to highlight the 3 most significant changes that I have made when I got new improvements. In the beginning, I used a linear model, and I got the prediction result with RMSE around 65. And I have to say feature selection is the most useful especially lasso, which reduced the RMSE of my result based on the linear model to around 60. Then I tried the complicated model which is random forest based on the variables selected by Lasso, this method reduced the RMSE by 3. Besides, after comparing the results of my experiments of using the random forest, I found picking 1000 as the number of trees gave the best prediction.

```
##    public_score                 method
## 1      51.88588          random forest
## 2      51.96670          random forest
## 3      52.36257          random forest
## 4      53.48492          random forest
## 5      53.49934          random forest
## 6      53.63653          random forest
## 7      53.67660          random forest
## 8      53.86723          random forest
## 9      56.59990 lm + feature selection
## 10     57.15146 lm + feature selection
## 11     57.22065 lm + feature selection
## 12     58.07323 lm + feature selection
## 13     58.69859 lm + feature selection
## 14     58.69917 lm + feature selection
## 15     61.10157 lm + feature selection
## 16     61.15068 lm + feature selection
## 17     64.81387    simple linear model
## 18     77.15619    simple linear model
## 19     77.15637    simple linear model
## 20     77.33783    simple linear model
## 21     83.68948          original code
## 22     87.31540          original code
## 23     98.89310          original code
## 24    104.43520          original code
## 25    104.43897          original code
## 26    104.44127          original code
## 27    553.04760                  error
## 28  30298.53963                  error
## 29  53091.59959                  error
## 30  53544.27596                  error
## 31  55631.42631                  error
```