My Kaggle Project Report

Zhikun Chen

11/29/2021

Abstract

For this project, we are interested to predict the prices for Airbnb rentals using 90 variables on the property, host, and past reviews.

1 Initial exploration

1.1 Data

1.1.1 Variables

The data is provided by Kaggle and it consists of 90 variables. For the best performance and prediction accuracy of our model, we selected 15 variables, which contains variables to help fill up the missing data and independent variables that are in our model.

Here is the description of the variables that we used.

- **price**: Price for each Airbnb rental.
- **cleaning_fee**: Cleaning_fee applied after stay.
- **zipcode**: Zipcode of Airbnb rental, which has 209 levels. This gives us a fairly precise representation of location.

- **property_type**: Accounts up to 33 different property type of Airbnb.
- **guests_included**: The maximum number of guests that can be included in Airbnb.
- **accommodates**: Number of accommodates in the Airbnb. A higher number of accommodates usually mean a bigger house/apartment.
- **neighbourhood_cleansed**: Neighbourhood of which Airbnb is at. 218 levels in total.
- **review_scores_location**: This determines how the location is rated on Airbnb
- **room_type**: Rental room type.
- **minimum_nights**: Minimum nights required for stay.
- **review_scores_cleanliness**: A review of overall cleanliness of the rental.
- **maximum_nights**: Maximum nights of each stay.

1.1.2 Data Cleaning and Exporation

First of all, we combine the data and the scoring data into the same data frame and name it 'combinedData'. By doing so, we won't have any missing values when running the model on the prediction data set, since we do have a lot of categorical variables that are really hard to deal with when predicting prices based on those variables.

Then, we are going to take a look at the histogram of the zip code to see if there are any rare cases in the sample. Apparently, according to our histogram, there are 209 levels in total, and almost 90 levels have low frequency. So, we will lump those levels together and call it 'others'. So that the zip code variable has less noise.

Third, we will do the same to the 'property_type' variable. We cut it down to 35 levels.

Lastly, we will lump the 'neibourhood_cleansed' variable down to 150 levels for the better predicting performance.

After we are done with the existing variables, we turn our focus to making new dummy variables from strings, the variable we are extracting data from is called 'Amenities'. The reason why we are looking at amenities is because empirically, properties that have better or more amenities are always sold or rented at a higher price than the similar properties without the same amenities. So we would pick some amenities that we think is valuable and will be value adding to the final price prediction. In this case, we picked 8 key words, which are 'Coffee', 'Laptop', 'TV', 'Gym', 'Elevator', 'Kitchen', 'Wifi', and 'Garden'. We make 8 dummy variables out of 'Amenities' and then add them up together to form a new numerical variable called 'am' to represent the amount of key words that a property has.

We decide to make a correlation plot to assess to correlation between each numeric variables in the data set. We mainly want to find out the correlation between price and other independent variables, and the correlation between cleaning fees and other numeric variables that we will use later to fill out the missing values of cleaning fees for better model accuracy. We can tell from the plot that accommodates and cleaning_fee have the highest correlations to price than other variables. So, we would include these two variables in our model.

1.1.3 Missing Values

There are 41330 rows in total for 'cleaning_fee' variable with 5958 missing values. Cleaning fee is a really important factor for predicting prices. So, we have to fill out the missing data in order to get the better prediction. First, we have tried the medium imputation method. However, we cannot treat zip code as a numeric variable, because there is no linear relation between price and the number of zip code. So, decision tree is the best method for filling out the missing values of zip code since the method works well with the categorical variables. We use the package 'missForest' and selected multiple variables, which includes 'price, cleaning_fee, minimum_nights, review_scores_cleanliness, guests_included, extra_people, accommodates, room_type, maximum_nights, maximum_minimum_nights'. This gives us the lowest estimated error of about 0.00026.

```r
source("Kaggle Exploration .R")
source("Kaggle Feature Selection.R")
source("Kaggle Model Boosting.R")
source("rangerupdated.R")
```

```r
## INITIAL EXPLORATION STARTS HERE

## Read data
data <- read.csv("~/Downloads/rentlala2021/analysisData.csv", stringsAsFactors=TRUE)
dim(data)
scoringData = read.csv('~/Downloads/rentlala2021/scoringData.csv')
scoringData$zipcode <- as.character(scoringData$zipcode)
library(dplyr)

## Combining data to jointly deal with factor labels and missing data
combinedData <- bind_rows(data, scoringData)

## Visualize the distribution of variable 'zipcode'
hist(as.numeric(data$zipcode))

## Lump the levels of zipcode to 120
combinedData$zipcode <- substr(combinedData$zipcode, 1, 5)
combinedData$zipcode <- as.factor(combinedData$zipcode)
combinedData$zipcode <- forcats::fct_lump_n(combinedData$zipcode, 120)


combinedData$property_type <- as.factor(combinedData$property_type)
combinedData$property_type <- forcats::fct_lump_n(combinedData$property_type, 35)

## Reduce the levels of 'neighbourhood_cleansed' to 150 to prevent overfitting
combinedData$neighbourhood_cleansed <- as.factor(combinedData$neighbourhood_cleansed)
combinedData$neighbourhood_cleansed <- forcats::fct_lump_n(combinedData$neighbourhood_cleansed,

## Explore the 'Amenities' variable and make dummy variables out of it
combinedData <- combinedData %>% mutate(gar = case_when(
  grepl("Garden", amenities) == TRUE ~ 1,
  grepl("Garden", amenities) == FALSE ~ 0
))

combinedData <- combinedData %>% mutate(wifi = case_when(
  grepl("Wifi", amenities) == TRUE ~ 1,
  grepl("Wifi", amenities) == FALSE ~ 0
))

combinedData <- combinedData %>% mutate(kit = case_when(
  grepl("Kitchen", amenities) == TRUE ~ 1,
  grepl("Kitchen", amenities) == FALSE ~ 0
))
```

```
combinedData <- combinedData %>% mutate(el = case_when(
  grepl("Elevator", amenities) == TRUE ~ 1,
  grepl("Elevator", amenities) == FALSE ~ 0
))

combinedData <- combinedData %>% mutate(gym = case_when(
  grepl("Gym", amenities) == TRUE ~ 1,
  grepl("Gym", amenities) == FALSE ~ 0
))

combinedData <- combinedData %>% mutate(tv = case_when(
  grepl("TV", amenities) == TRUE ~ 1,
  grepl("TV", amenities) == FALSE ~ 0
))

combinedData <- combinedData %>% mutate(laptop = case_when(
  grepl("Laptop", amenities) == TRUE ~ 1,
  grepl("Laptop", amenities) == FALSE ~ 0
))

combinedData <- combinedData %>% mutate(coffee = case_when(
  grepl("Coffee", amenities) == TRUE ~ 1,
  grepl("Coffee", amenities) == FALSE ~ 0
))

combinedData <- combinedData %>% mutate(am = gar + wifi + kit + el + gym + tv + laptop + coffee)

## Visualize the correlations between numeric variables and price
data1 <- data
data1$price = as.numeric(data1$price)
corpredictors <- select_if(data1, is.numeric)
corpredictors <- corpredictors %>% select(price, review_scores_location, review_scores_accuracy,
                                          review_scores_cleanliness, review_scores_communication
                                          review_scores_value, extra_people, accommodates,
                                          cleaning_fee)
corPlot(corpredictors, cex = 0.8)

combinedData <- combinedData %>% select(price, cleaning_fee, zipcode, property_type,
                                        guests_included, extra_people, accommodates,
                                        neighbourhood_cleansed, review_scores_location,
                                        room_type, minimum_nights, review_scores_cleanliness,
                                        maximum_nights, maximum_minimum_nights, am)

## Assign the rows of 'combinedData' variable that do not have prices to scoringData and the rou
train <- combinedData[!is.na(combinedData$price),]
scoringData <- combinedData[is.na(combinedData$price),]
```

```
## INITIAL EXPLORATION ENDS HERE
```

2 Models and Feature Selection

For feature selection, we use LASSO, Best subset, Hybrid Stepwise to generate different combinations of variables for the model. After applying variables from these three feature selections and comparing the RMSEs of different models, we found out that LASSO is the best method for both of our random forest and boosting models. The variables output from LASSO are: 'neighbourhood_cleansed, property_type, room_type, accommodates, bedrooms, bathrooms, review_scores_cleanliness, review_scores_location, cleaning_fee'. After feature selection, we proceed to select the best model to help us predict the Airbnb prices. The models include linear regression model, random forest decision tree model, and boosting decision tree model.

For cv random forest decision tree model, The final values used for the model are mtry = 9, splitrule = variance and min.node.size = 5, with the RMSE of 73.89465.

In this case, boosting model with package 'xgboost' gives us the lowest test and train RMSE score, therefore, boosting model is our final model.

```
## MODELING AND FEATURE SELECTION STARTS HERE
## FEATURE SELECTION 1: LASSO, HYBRID SEARCH STARTS HERE

## Select 12 variables for LASSO
x <- model.matrix(price~host_response_rate + host_is_superhost +
                    neighbourhood_cleansed + property_type + room_type +
                    accommodates + bedrooms + bathrooms + beds +
                    review_scores_cleanliness + review_scores_location + cleaning_fee,
                  data=data)

## Dependent variable: Price
y = data$price
set.seed(1031)
cv_lasso = cv.glmnet(x = x,
                    y = y,
                    alpha = 1,
                    type.measure = 'mse')
lassoModel <- glmnet(x, y, alpha=1, standardize = TRUE)
cv.lasso <- cv.glmnet(x, y, alpha=1)
coef(cv.lasso, s = cv.lasso$lambda.1se) %>%
  round(4)
cv.lasso

## Output summary of our LASSO model, the final model contains host_response_rate, neighbourhood
summary(cv.lasso)

## FEATURE SELECTION 1: LASSO, HYBRID SEARCH ENDS HERE
```

```r
## MODEL 1 (Final Model): BOOSTING DECISION TREE STARTS HERE
set.seed(617)

## Set parameters for boosting model
tune_nrounds = xgb.cv(data=as.matrix(train_input),
                      label = train$price,
                      nrounds=250,
                      nfold = 5,
                      verbose = 0)

## Visualize the test rmse for different features
ggplot(data=tune_nrounds$evaluation_log, aes(x=iter, y=test_rmse_mean))+
  geom_point(size=0.4, color='sienna')+
  geom_line(size=0.1, alpha=0.1)+
  theme_bw()

which.min(tune_nrounds$evaluation_log$test_rmse_mean)

## Select the parameters that gives the lowest rmse for our final model
xgboost2= xgboost(data=as.matrix(train_input),
                  label = train$price,
                  nrounds=which.min(tune_nrounds$evaluation_log$test_rmse_mean),
                  verbose = 0)
pred = predict(xgboost2,
               newdata=as.matrix(test_input))

## MODEL 1: BOOSTING DECISION TREE ENDS HERE

## MODEL 2: RANDOM FOREST DECISION TREE STARTS HERE

formula <- "price ~ cleaning_fee + zipcode + property_type + guests_included + extra_people +
            accommodates+ neighbourhood_cleansed + review_scores_location + room_type"
trControl <- trainControl(method="cv",number=5)

## Test out different parameters, which are mtry, splitrule, and min.node.size
tuneGrid <- expand.grid(mtry=c(5,7,9),
                        splitrule = c('variance','extratrees','maxstat'),
                        min.node.size = c(5,15,30))
set.seed(617)
cvModel = train(as.formula(formula),
                data=train,
                method="ranger",
                num.trees = 1000,
                trControl=trControl,
                tuneGrid=tuneGrid,
                na.action = na.exclude)
cvModel
```

```
## Use best tuned parameters for our final model
cv_forest_ranger = ranger(as.formula(formula),
                          data=train,
                          num.trees = 1000,
                          mtry=cvModel$bestTune$mtry,
                          min.node.size = cvModel$bestTune$min.node.size,
                          splitrule = cvModel$bestTune$splitrule,
                          na.action = na.pass)
cv_forest_ranger
## MODEL 2: RANDOM FOREST DECISION TREE ENDS HERE

## MODEL 3: LINEAR REGRESSION STARTS HERE
lasso_model = lm(
  price ~ host_response_rate + neighbourhood_cleansed +
    property_type + room_type +
    accommodates + bedrooms + bathrooms +
    review_scores_cleanliness + review_scores_location + cleaning_fee,
  data
)

## MODEL 3: LINEAR REGRESSION ENDS HERE

## MODELING AND FEATURE SELECTION ENDS HERE
```

3 Model Comparison

Model from previous section	RMSE for test data holdout or CV	RMSE on Kaggle	notes
Model 1: Boosting	67.77079	72.55766	Final
Model 2: Ranger Random Forest	73.89465	75.38289	Used
Model 3: Linear Regression		82.71936	Used

4 Discussion

To understand the difference in performance in random forest model and boosting model, first we need to understand the difference in approaches these two models use to minimize the prediction error. For random forest model, since it uses fully grown decision trees, it reduces error by minimizing the variance, while boosting model is based on weak learners, which reduces error by reducing bias. This also explains why we see a major discrepancy for boosting model between the RMSE for CV and the RMSE on Kaggle (scoring data). Boosting model pays very high attention to training data, hence it is easier to over fit the model. While both of random forest and boosting model may be good at predicting in our case, bias might have more impact on our model than variance considering our existing cleaned data since boosting model gives a lower RMSE than random forest model does. There is still room for further cleaning and finding a better trade off between our bias and variance.

Another cause of over fitting may be because of the amount of categorical variables and their large number of levels. In our model, we have a categorical variable with 150 levels and a categorical variable with 120 levels. Although we have 41330 samples in our train data set, the distribution of price is left skewed and not evenly distributed, and it might still be too small for a 150 levels categorical variable to generalize, which can cause high variance in our model and leaves more room for over fitting.

5 Future directions

1) try more ways to fill out the missing data than just using random forest and compare the rmse to figure out the best approach.

2) reduce the levels of certain categorical variables such as zip code, neighbourhood_cleansed, and property type to find the equilibrium between bias and variance, thus minimizing the total error.

3) better utilize the character variables with texts, such as amenities, reviews, summary, etc. to predict the extreme prices. Because usually the houses with gardens that are located at some parts of long island are expensive.