Kaggle Report

Jiayi Huang

11/25/2021

Introduction

People share their rental information of their property from Airbnb. The goal of this project is to construct a predictive model to forecast the rental price of the properties based on certain features. The metric of the project is root mean squared error of the predictive model. My project selects minimum_nights, minimum_nights_avg_ntm, minimum_minimum_nights, zipcode, property_type, room_type, accommodates, bathrooms, bedrooms, cleaning_fee, guests_included, extra_people, availability_60, availability_365, instant_bookable, review_scores_cleanliness, review_scores_location, review_scores_rating, host_response_rate. I adopted Xgboost with nrounds = 182, max_depth = 6, eta= 0.161 to construct the model and achieved RMSE 61.64230 for scoring data set on private leader board.

Read Data

The “data” is training dataset to build the model. The "scoringData’ is validation dataset to test the accuracy of model. The data is inputed by read.csv data.

```
data = read.csv('/Users/jiayihuang/Documents/APAN5200/rentlala2021/analysisData.csv')
scoringData = read.csv('/Users/jiayihuang/Documents/APAN5200/rentlala2021/scoringDat
a.csv')
```

Impute Data

Missing data are common in real data sets; however, missing value will cause error in prediction. Many categories in data sets have “N/A” values. I applied is.na() function on the specific column to detect the missing value. These missing values should be filled with a rational value to successfully train the model and predict the price for scoring data set. In the real-life prediction, the row with missing or empty variables are deleted from data sets. However, Kaggle competition evaluate the prediction with the specific numbers of rows. Therefore, all the data in scoring data sets are imputed as well. To reduce the variance, variable with over 70% of missing values will be removed from data sets. I filled the rest of missing data with the median value of that column. The advantage of median imputation is to solve the skewness of data.

```
library(readr)
data$square_feet[is.na(data$square_feet)] <- median(data$square_feet, na.rm=TRUE)
data$bathrooms[is.na(data$bathrooms)] <- median(data$bathrooms, na.rm=TRUE)
data$bedrooms[is.na(data$bedrooms)] <- median(data$bedrooms, na.rm=TRUE)
data$beds[is.na(data$beds)] <- median(data$beds, na.rm=TRUE)
data$cleaning_fee[is.na(data$cleaning_fee)] <- median(data$cleaning_fee, na.rm=TRUE)
data$security_deposit[is.na(data$security_deposit)] <- mean(data$security_deposit, n
a.rm=TRUE)
```

Data Transformation

Many variables have wide range and complicated composition. For example, the zip code has many different types of inputs, including “10027”(5-digit string),"“(empty string),”NY 10014“(string includes characters and digits), and”11103-3233“(9-digit string). For consistency of model construction, all the inputs in each variable are transformed to the same type. The zip code column is transformed into a 5-digit integer by using regular expression. Since there are 209 unique zip codes, the zip codes occur smaller than 10 are collapsed into “00000” (other) to reduce the level of variables and model complexity. In addition, the Xgboost model is captious about the input parameters. Some string categorical variables is changed into integer and factor for prediction. For example, the variable property type includes 18 unique string values. Some of these values only appear once in the entire dataset. These strings are turned into numerical variables to fit into the model. Therefore, data transformation plays an important role in the data cleaning, which fits the data for train and prediction, remove redundant values, and categorized data. As mentioned in previous sections, all the same data transformation steps are applied to scoring data set for prediction.

```r
#zipcode
library(readr)
data$zipcode <- gsub("[^0-9]", "", data$zipcode, perl=TRUE)
data$zipcode = substr(data$zipcode, 1, 5)
zipcode_freq = data.frame(table(data$zipcode))
low_zipcode_freq = subset(zipcode_freq,Freq < 10)
data$zipcode = ifelse(data$zipcode %in% low_zipcode_freq$Var1, "00000", data$zipcode)
data$zipcode = ifelse(data$zipcode == "","00000", data$zipcode)
data$zipcode = parse_number(data$zipcode)
#accomodates
accommodates_freq = data.frame(table(data$accommodates))
low_accommodates_freq = subset(accommodates_freq,Freq < 100)
data$accommodates =  ifelse(data$accommodates %in% low_accommodates_freq$Var1,0,data
$accommodates)
### RoomType
data$room_type = ifelse(data$room_type == "Entire home/apt", 1, data$room_type)
data$room_type = ifelse(data$room_type == "Hotel room", 2, data$room_type)
data$room_type = ifelse(data$room_type == "Private room", 3, data$room_type)
data$room_type = ifelse(data$room_type == "Shared room",4, data$room_type)
### property type ----
proptype_freq = data.frame(table(data$property_type))
low_proptype_freq = subset(proptype_freq,Freq < 10)
data$property_type =  ifelse(data$property_type %in% low_proptype_freq$Var1,'other',d
ata$property_type)
data$property_type = ifelse(data$property_type == "Aparthotel",1,data$property_type)
data$property_type = ifelse(data$property_type == "Apartment",2,data$property_type)
data$property_type = ifelse(data$property_type == "Boutique hotel",3,data$property_ty
pe)
data$property_type = ifelse(data$property_type == "Bungalow",4,data$property_type)
data$property_type = ifelse(data$property_type == "Camper/RV",5,data$property_type)
data$property_type = ifelse(data$property_type == "Condominium",6,data$property_type)
data$property_type = ifelse(data$property_type == "Guest suite",7,data$property_type)
data$property_type = ifelse(data$property_type == "Guesthouse",8,data$property_type)
data$property_type = ifelse(data$property_type == "Hostel",9,data$property_type)
data$property_type = ifelse(data$property_type == "Hotel",10,data$property_type)
data$property_type = ifelse(data$property_type == "House",11,data$property_type)
data$property_type = ifelse(data$property_type == "Loft",12,data$property_type)
data$property_type = ifelse(data$property_type == "Other",13,data$property_type)
data$property_type = ifelse(data$property_type == "Resort",14,data$property_type)
data$property_type = ifelse(data$property_type == "Service apartment",15,data$propert
y_type)
data$property_type = ifelse(data$property_type == "Tiny house",16,data$property_type)
data$property_type = ifelse(data$property_type == "Townhouse",17,data$property_type)
data$property_type = ifelse(data$property_type == "Villa",18,data$property_type)
data$property_type = parse_number(data$property_type)

data$instant_bookable = ifelse(data$instant_bookable == "t",1,0)
data$host_response_rate=as.character(data$host_response_rate)
data$host_acceptance_rate=as.character(data$host_acceptance_rate)
data$host_response_rate=parse_number(data$host_response_rate)
data$host_response_rate=as.numeric(data$host_response_rate)
data$host_acceptance_rate=parse_number(data$host_acceptance_rate)
data$host_acceptance_rate=as.numeric(data$host_acceptance_rate)
```

Feature selection

All the descriptive columns are removed to reduce the prediction complexity, such as name, summary, descriptions, and so on. These variables are removed based on the understanding of model and relevance of price. The house rental price is based on location, amenities, housing brands, neighbors, and square foot of house (Angerer, 2016). For further feature selection, I have applied the Lasso Regression, which could force some coefficient equal to zero and make small modification to shrinkage penalty. The reason to apply feature selection is that predictions from simple (versus complex models) are more stable across samples. As number of predictors increases, the chance of finding correlations among a predictor or a set of predictors increases. There are some advantages of Lasso, such as perform better than stepwise selection and perform variable selection to any number of variables.

```
# library(glmnet);library(dplyr)
# x = model.matrix(price~host_since+minimum_nights_avg_ntm+minimum_nights+maximum_nig
hts+review_scores_rating+security_deposit+guests_included+cleaning_fee+bathrooms+bedr
ooms+zipcode+room_type+accommodates+review_scores_accuracy+beds,data=data)
# y = data$price
# set.seed(617)
# cv_lasso = cv.glmnet(x = x,
#                       y = y,
#                       alpha = 1,
#                       type.measure = 'mse')
# coef(cv_lasso, s = cv_lasso$lambda.1se) %>%
#   round(4)
```

Modeling Framework

As mentioned in the previous section, increase complexity in training data set will decrease the performance on test data set. To avoid the overfitting, the train data set is randomly separated by 70%. The 70% of data is training data, which is used for model construction; the 30% of data is test data, which is used for validation. The model performance on the test data the significant. Once the test RMSE is minimized under this situation, the model is applied to the entire data sets and predicted to the scoring data set.

```
set.seed(1731)
split = sample(x = 1:nrow(data),size = 0.7*nrow(data))
train = data[split,]
test = data[-split,]
```

Model

To predict a numerical outcome, a regression method is adopted to construct the model. I have applied the linear regression (as example), regression decision tree, random forest and Xgboost to train the model. The parameters are tuned to improve the prediction accuracy and reduce test RMSE. The linear regression is the oldest and most common predictive method. However, in this project, multiple predictors increase linear regressor's complexity, which leads to overfitting problem. Overfitting is seen when in-sample performance far exceeds out-of-sample performance. As model complexity increase, model perform better on train data sets, but accuracy on test data set is decrease and RMSE is increase. The general goal of a predictive model is to assess performance on the scoring data, instead of good performance on training data sets. The regression decision tree has the same overfitting problem when the model complexity increases with the large number of variables. The number of variables also extend the model construction length in time. Moreover, the decision tree cannot be used well for numerical outcomes. Random forest and Xgboost are two methods that perform better on large data and numerical outcomes. These two methods reduce overfitting in decision trees and helps to improve the accuracy. Especially, Xgboost consists of a number of hyper-parameters that can be tuned which is an integral of gradient boosting machines. The parameters are tuned to minimize the RMSE. By comparing the RMSE of random forest and Xgboost with the same variables, I decided to use Xgboost function to build my model for this project.

```r
# library(xgboost);library(caret)
# set.seed(1031)
# select_par_data <- data.frame(data$minimum_nights,
#                              data$minimum_nights_avg_ntm,
#                              data$minimum_minimum_nights,
#                              data$zipcode,
#                              data$property_type,
#                              data$room_type,
#                              data$accommodates,
#                              data$bathrooms,
#                              data$bedrooms,
#                              data$cleaning_fee,
#                              data$guests_included,
#                              data$extra_people,
#                              data$availability_60,
#                              data$availability_365,
#                              data$instant_bookable,
#                              data$review_scores_cleanliness,
#                              data$review_scores_location,
#                              data$review_scores_rating,
#                              data$host_response_rate)
# select_par_data_matrix = data.matrix(select_par_data)
# xgboost = xgboost(data = select_par_data_matrix, label= data$price,
#                  nrounds=182,
#                  verbose = 1,
#                  max_depth = 6,
#                  eta = 0.161)
```