Report on Predicting Rent in New York

Woon Kim

12/12/2021

The Data

```
data = read.csv('analysisData.csv')
scoringData = read.csv('scoringData.csv')

data = data %>% mutate(type = "analysis")
scoringData = scoringData %>% mutate(type = "scoring")
combinedData = bind_rows(data, scoringData)
```

1. Exploratory Data Analysis

Missing Values At first glance, it was apparent that a large part of the data was missing. Although regression trees and advanced tree methods can handle missing values well, in an effort to improve the predictive powers of the features, I explored methods to handle the missing values.

- Imputation:
  - Cleaning Fee: Entries without any values for "cleaning fee" may have been due to the fact that there were no cleaning fees. Hence a value of 0 was imputed for these missing entries.
  - Host_listings_count, reviews_per_month: Entries without any values for these variables were imputed with their median values.

```
# Imputation
combinedData$cleaning_fee[is.na(combinedData$cleaning_fee)] <- 0

combinedData$host_listings_count[is.na(combinedData$host_listings_count)] =
  median(combinedData$host_listings_count, na.rm=TRUE)

combinedData$reviews_per_month[is.na(combinedData$reviews_per_month)] =
  median(combinedData$reviews_per_month, na.rm=TRUE)
```

Messy Variables

Some entries of the "city" and the "zipcode" were recorded with special characters and were entered in both upper and lower cases, hence they would be recognized as different levels. In order to avoid any incorrect distinctions, all the special characters were removed and all the entries were converted to lower cases.
Furthermore, some variables were recognized as the wrong class. Two different functions were created to convert these features into the correct class:

1. to_logical_from_factor: This function converts any variables that are of type factor AND has the level 3 to be of type logical. Level 3 was selected instead of 2 because of blank entries in addition to the binary levels.
2. to_factor_from_numeric: This function converts any numerical variables with levels less than 10 to be of type factor. Variables such as ratings are often recorded from a range of 1-10 which is recognized as a numerical variable. However, the correct assignment of the class would be of Factor.

```r
# Zipcode
as.character(parse_number(combinedData$zipcode))
combinedData$zipcode <- gsub("[^\\d]", "", combinedData$zipcode, perl=TRUE)
combinedData$zipcode <- substr(combinedData$zipcode, 1, 5)

table(nchar(combinedData$zipcode))
combinedData$zipcode[nchar(combinedData$zipcode) < 5] <- NA_character_
combinedData$zipcode <- as.factor(combinedData$zipcode)

# City
# remove commas, dashes, newlines, parens and periods
combinedData$city <- gsub(",|-|\\n|\\)|\\(|/|\\.", " ",
                          tolower(combinedData$city))

# trim white space
combinedData$city <- stringr::str_trim(gsub("\\s+", " ", combinedData$city))

combinedData = combinedData %>% mutate_all(to_logical_from_factor)
combinedData = combinedData %>% mutate(across(where(is.character),as.factor))
combinedData = combinedData %>% mutate_all(to_factor_from_numeric)
```

Variables with low predictive power

Some factor variables were observed to have little variance or too much variance. Two different functions were created to remove these variables as they would not have much predictive power.

1. remove_fact_1lvl: This function removes any factor variables that only has one level.
2. remove_columns: This function removes any factor variables that have more than 1000 levels.

```r
#remove factor variables with only one level
combinedData = combinedData %>% mutate_all(remove_fact_1lvl)

#remove factor variables with more than 1000 levels
id = combinedData$id #so as to not remove the id column
combinedData = combinedData %>% mutate_all(remove_columns)
combinedData$id = id
```

2. Data Preparation

Following the exploratory data analysis, I conducted simple analyses on continuous variables in order to prepare for predictive modeling.

Outlier Handling

In an effort to handle the outliers while retaining as much information from the data, a function named "outlier_norm" was created to identify outliers and replace their values with either the 5th or the 95th percentile values.

```r
# Converts the outliers of all continuous variables into
#the 5th or 95th percentile values.
for (i in 2:ncol(cont_data)){
  #index 9 is the response variable
  if(i == 9){

  #The function only works for columns without any missing values
```

```
  }else if (sum(is.na(cont_data[,i]))==0){
    cont_data[,i] = outlier_norm(cont_data[,i])
  }
}
```

Bi-Variate Analysis: Multi-Colinearity

Bi-variate analysis was conducted to assess the correlations between variables. The function "remove_multi_cor" was created to identify variables that have correlations greater than 0.9 so that they can be inspected further.

In addition, a linear regression model was created so that an inspection on VIF can be done to identify the threat of multicolinearity.

Variables that were removed due to the threat of multi-colinearity are listed below:

- host_total_listings_count
- minimum_minimum_nights
- minimum_maximum_nights
- maximum_nights_avg_ntm
- minimum_nights_avg_ntm
- availability_60
- calculated_host_listings_count
- calculated_host_listings_count_private_rooms
- calculated_host_listings_count_entire_homes
- availability_90

3. Analysis Techniques Explored

With the data that has been prepared, Regression Tree, Simple Boosting model, and Boosting model with xgboost was conducted.

```
# Splitting the training and test set
set.seed(1031)
analysis = filter(combinedData, type == 'analysis')
split = createDataPartition(y = analysis$price, p = 0.75, list = F,groups = 10)

train = analysis[split,]
test = analysis[-split,]
```

Regression Tree

The benefit of the regression tree was that it was able to conduct the analysis with little runtime and without any further intervention with the data. Furthermore, by looking at its important variables, the regression tree model was a quick and easy way to conduct feature selection. One issue with this technique was that it did not produce very good prediction values.

```
# Make Valid Column Names
colnames(combinedData) <- make.names(colnames(combinedData))

tree = rpart(price~., data=train, method = 'anova')

pred_tree = predict(tree,newdata=train, type='vector')
pred_tree_test = predict(tree,newdata=test, type='vector')

tree_rmse_train = rmse(actual = train$price,
```

```
                 predicted = pred_tree)

tree_rmse_test = rmse(actual = test$price,
                 predicted = pred_tree_test)

tree$variable.importance

tree_rmse_train
tree_rmse_test
```

Boost gbm

The simple boosting model was conducted with the same variables as the regression tree. A clear strength of this technique compared to the regression tree is its performance in predicting the value. One weakness with the simple bossting model might be that it is prone to overfitting to the training data.

```
set.seed(617)
boost = gbm(price~.,
            data=train,
            distribution="gaussian",
            n.trees = 500,
            interaction.depth = 2,
            shrinkage = 0.01)

gbm_pred = predict(boost, n.trees=500)
gbm_pred_test = predict(boost, newdata = test, n.trees=500)

gbm_rmse_train = rmse(actual = train$price,
                predicted = gbm_pred)

gbm_rmse_test = rmse(actual = test$price,
                predicted = gbm_pred_test)

gbm_rmse_train
```

```
## [1] 70.20767
```

```
gbm_rmse_test
```

```
## [1] 71.52375
```

XGBOOST

Boosting with xgboost was considered as the final technique to predict the values. Unlike the other two techinques before, additional analyses was conducted on the categorical variables.

Encoding Categorical Variables To use the xgboost model in library(xgboost), all of the categorical variables are required to be dummy coded.

Removing near zero variance variables After dummy coding all of the categorical variables, the dataset was assessed for any variables with near zero variance. The function remove_nzv_columns removes any variables that have very small variance as this variable would not have any predictive power. This additional step was conducted only for xgboost because the categorical variables have been dummy coded.

```
#combine dummy coded categorical variables with
#continuous variable to complete the dataset
df_all_combined = cbind(cont_data, trsf)

#conduct near zero variance analysis to
#remove variables with low predictive power
combinedData = df_all_combined %>% mutate_all(remove_nzv_columns)
```

Performing XGBOOST xgboost model is finally conducted after converting the data into a matrix.

```
#Splitting the data to train and test set
data1 = filter(combinedData, type == 'analysis')

set.seed(1031)
split = createDataPartition(y = data1$price, p = 0.75, list = F,groups = 10)
train1 = data1[split,]
test1 = data1[-split,]

#Transforming the train and test set into Matrix
#without the response variable, type and id.

train2 = train1[,-c(which(names(train1) == 'price'),
                    which(names(train1) == 'type'),
                    which(names(train1) == 'id'))]

test2 = test1[,-c(which(names(test1) == 'price'),
                  which(names(test1) == 'type'),
                  which(names(test1) == 'id'))]

dtrain = xgb.DMatrix(label = train1$price, data = as.matrix(train2))
dtest = xgb.DMatrix(label = test1$price, data = as.matrix(test2))
```

One strength of xgboost over the simple boosting model is the cross-validation. Using the cross validation method, the model can determine the optimal hyper-parameters that would avoid overfitting. Difficulties using the xgboost was that all categorical variables had to be dummy coded, all the data had to be transformed into a matrix and also that xgboost model can be computationally heavy causing the runtime to be very long.

```
#default parameters
params1 = list(
  booster = "gbtree",
  eta=0.3,
  gamma=0,
  max_depth=6,
  min_child_weight=1,
  subsample=1,
  colsample_bytree=1)

set.seed(1031)
xgbcv = xgb.cv(
  label = train1$price,
  params = params1,
  data = dtrain,
  nrounds = 200,
  nfold = 5,
```

```
  stratified = T,
  early.stop.round = 10,
  maximize = F,
  verbose = 0)

which.min(xgbcv$evaluation_log$test_rmse_mean)
```

```
## [1] 50
```

```
#optimal nrounds = 50 round.
```

Model xgboost with the test set xgboost is conducted on the test set using the optimal nround value from cross-validation in order to avoid overfitting.

```
xgb1 <- xgb.train (
  params = params1,
  data = dtrain,
  nrounds = 50,
  watchlist = list(val=dtest,train=dtrain),
  early.stop.round = 10,
  maximize = F ,
  eval_metric = "rmse")
```

```
pred_xgboost_train = predict(xgb1, newdata = dtrain)
xgboost_rmse_train = rmse(actual = train$price,
                          predicted = pred_xgboost_train)

pred_xgboost_test = predict(xgb1, newdata=dtest)
xgboost_rmse_test = rmse(actual = test$price,
                         predicted = pred_xgboost_test)

xgboost_rmse_train
```

```
## [1] 49.86864
```

```
xgboost_rmse_test
```

```
## [1] 64.73401
```

4. Best Analysis Technique

Best analysis was chosen based on the rmse values each of the models produced. Both the rmse on train set and the test set were considered so as to avoid the model that had too much overfitting and also to adjust any necessary hyper-parameters.
Although not shown in this report, I have explored different hyper-parameters for the xgboost model. However, the default hyper-parameter seemed to have performed the best.

Summarizing Results

id	model	train RMSE	test RMSE
1	tree	78.411	80.360
2	boost - gbm	70.208	71.524
3	xgboost	49.869	64.734

5. Results

As xgboost clearly produced a set of prediction with the lowest rmse values, this technique was chosen to conduct the final prediciton using the entire dataset.

```
#RMSE evaluation
pred_xgboost = predict(xgb1, newdata = dtrain)
xgboost_rmse = rmse(actual = train1$price,
                    predicted = pred_xgboost)
xgboost_rmse
```

```
## [1] 52.17926
```

It is worth noting that much of the code presented in this report have been modified and improved since the Kaggle competition deadline. My submission to the competition resulted in an rmse of 63.1 which places me at 257/631.

Data wrangling processe has contributed the most to the performance of my model. Much of my effort was committed to conducting various data wrangling techniques in order to produce clean data with good predictive power.

One thing I would have done differently is, instead of removing variables entirely, transforming them to a format that would be useful. For example, changing dates into a numeric variable days since, or imputing the host verification variables into a factor variable with more interpretable levels.

```
pred = predict(xgb1, newdata=dtest)

submissionFile = data.frame(id = scoringData$id, price = pred)
write.csv(submissionFile, 'submission.csv',row.names = F)
```

6. Conclusion

One important takeaway from this project for me was the importance of exploratory data analysis and data wrangling techniques. Through the existing libraries, all the advanced predictive models were equally available for everyone participating in this competition. What set apart one's prediction over others' were on making the data better suited for the problem at hand through data wrangling practices.