Airbnb Case Report

By Peiyan Yu

Executive Summary

This report is commissioned by group10, using collected Airbnb data to identify different factors in determining a listing have a high booking rate or not. This information is designed to inform listing hosts on how to improve their booking rate.

Regarding prediction accuracy, after feature selection and model tuning, Random Forest provides us the highest accuracy of 84.4% on testing dataset, which suggests that the prediction provided for our customers are correct 84.4% of the time . As a result, random Forest is the best model for predicting the high booking rate of listings. In addition, attempt on logistic regression, classification tree and KNN has been made .

Results from logistic regression, KNN and importance score based on Random Forest offer a way to interpret predictors used to build the model. Although the hosts don't realize the importance of different factors, observation of the dataset identified several key potential characteristics that may explain different levels of booking rate: minimum nights, instant bookable or not, accommodates, etc. At the first glance, by visualization of listings distributed among the US, the Top three states having the most listing are California, New York and Washington DC respectively; On the contrary, Washington, Colorado and North Carolina have the least listings. And the scatter plot of availability_30, availability_60, availability_90, availability_365 indicate that, in general, availability_30 < availability_60 < availability_90 < availability_365;

According to logistic model, variables such as accommodates and cancellation policy are significant variables in predicting if it has a high booking rate for a listing. In general, an increase in cancellation_policymoderate or an increase in accommodates will result in an increase in the probability of listing has a high booking rate while an increase in cancellation_policysuper_strict_30 or in cleaning_fee would result in a decrease in probability of listing has a high booking rate. This gives listing holders an important signal for its listing campaign: listings without a strict cancellation policy are more welcomed than others with a strict policy; listings which can hold more guests to stay in and without a cleaning fee are more likely to attract customers.

Here are the recommendations for listing hosts to improve their booking rate:

- Prepare enough spaces to accommodate guests to stay in;
- Relieve the cancellation policy and decrease the cleaning fee to some extent if possible;
- Provide description of their listings as much as possible, which can help guests have a better insight into the listings they are going to live in;
- Provide 24-hour-check-in;
- Relieve the minimum night restriction;
- Prepare enough amenities such as coffee maker, air conditioning, hangers and shampoo can attract guest to book.

Data Pre-processing
We used both r and Python for data cleaning, identified and resolved the following issues:

Missing values

1. For price or fee data and other numeric data, we cleaned the form and filled NA with 0, mean or median depending on the context and domain knowledge.
2. For variables like bathrooms, which are highly correlated with bedrooms., we filled NA with the mode within the same number of bedrooms. Also like state column, we filled NA based on zip codes.
3. For categorical data, we filled NA with most common class or labeled them as Unknown class.
4. For first_review and host_since column, we transformed them into days of the timelapse. And we filled first_review NA as 0.

Text: For text variables like 'access', 'notes', 'house_rules', 'transit', 'interaction', 'description', we examined the content and it seems if the row is not NULL, the content is always positive. So firstly we transformed the rows that're not null to 1 and null rows to 0. Then we took it further to count the length of such text columns and fill the number in instead of 1.

Text that contains a list: For columns like amenities and host_verifications, we extracted the list of amenities and verification types, based on term frequency to eliminate amenities that appeared too little and transformed into a dummy matrix.

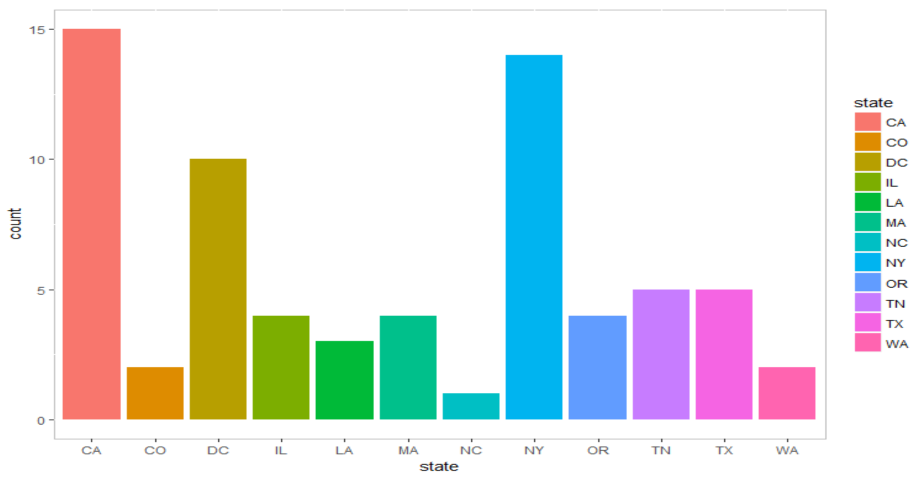
Wrong rows: We identified 11 rows whose data are misplaced. We eliminated these rows and marked the index.

Factors,dummies and numerics:
Based on the logic of different models, we transformed variables into factors,dummies or numerics. And in further model fitting process, we tried out all transformation methods.

Preliminary feature selection
Based on correlation table and duplicate columns, we eliminated several columns that contain identical information.

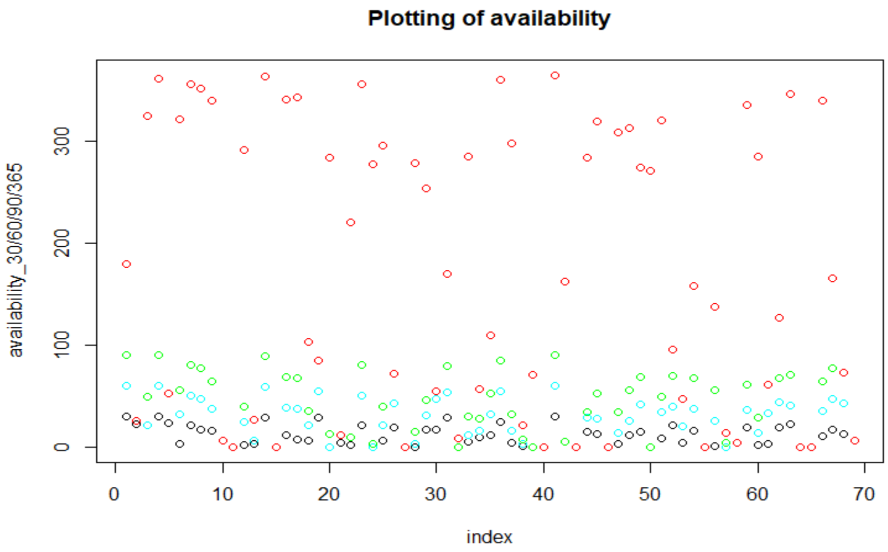
Preliminary Data Analysis

We prepare to get a general idea of our data set; therefore, we try to do some data visualization.
At this stage, since we have a dataset over 100,000 data, we sampled 0.1% of data to do visualization.
First, we want to know how the listings distributed among the US in the dataset, we used ggplot to find the frequency of each state:



Conclusion: Top three states which have the most listing are California, New York and Washington DC respectively; On the contrary, Washington, Colorado and North Carolina have the least listings.

Second, we want to find to general relationship among availability_30, availability_60, availability_90, availability_365:
We plot vailability_30 as black, availability_60 as cyan, availability_90 as green and availability_365 as red:



In general, we can tell that red points are higher than green point, green points are higher than cyan points and cyan points are higher than black points; Therefore, we can say that in this case, availability_30 < availability_60 < availability_90 < availability_365

1. Predictors relationship

In this step, we want to find the correlation of predictors:

Correlation matrix

	access	accommodates	availability_30	availability_365	availability_60	availability_90
access	1.00	0.04	-0.01	0.03	0.01	0.03
accommodates	0.04	1.00	0.04	0.13	0.04	0.05
availability_30	-0.01	0.04	1.00	0.53	0.93	0.86
availability_365	0.03	0.13	0.53	1.00	0.61	0.65
availability_60	0.01	0.04	0.93	0.61	1.00	0.97
availability_90	0.03	0.05	0.86	0.65	0.97	1.00
bathrooms	0.02	0.56	0.06	0.10	0.06	0.06
bedrooms	0.01	0.77	0.03	0.09	0.02	0.02
beds	0.04	0.83	0.05	0.12	0.05	0.05
cleaning_fee	0.03	0.58	0.05	0.14	0.05	0.06

Just a screenshot part of the correlation matrix, the correlation between availability_30 and availability_365 is 0.53, the correlation between availability_30 and availability_60 is 0.93, the correlation between availability_30 and availability_90 is 0.86. So far, we can say any two of these four variables thus have high correlation.

Another interesting correlation are between accommodates and bathrooms/bedrooms/beds/cleaning fee, which makes sense since the more persons a listing accommodates, the larger the listing should be, and therefore, the more bathrooms/bedrooms/beds it should have. Correspondingly, cleaning fee would be higher.

2. Variable selection

We use Recursive Feature Elimination (RFE) to find the significant variables: (Partial Output)
Variable Name Importance
1. minimum_nights 26.5786275873412

2. instant_bookable 22.2652056972072
3. availability_30 21.202058136947
4. availability_60 20.7826054935467
5. availability_90 19.2691108439627
6. host_is_superhost 17.9611551236132
7. availability_365 17.1300027224183
8. price 11.338949400977

We used random forest function and cross validation method to evaluate the outcome for every single iteration, to help us explore every possible combination of different variables subset. The higher the importance is, the more significant the variable is. And the outcome can tell us how many of the total data can perform as well as the whole dataset.

And then we compare predictors correlationship and variable selection, we choose variables which have importance over 4 without correlation. For example, we just choose availability_30 without availability_60, availability_90, availability_365, since availability_30 has highest importance.

Model Details

1. Logistic Model

In the very first beginning, since our dependent variable is binary with two levels, our team chose to use logistic model to take a glimpse of how those variables performs after feature selection. From our summary output, we can take a look at predictors marked as significant. Most of the predictors marked as significant in the output agree with the variables we selected by forward selection method. For example, we can see that amenities like shampoo, hair dryer and self-check-in are also marked as significant variables in our logistic model output. However, we notice that with this logistic model our accuracy is stick around 78% on our testing data, which is a little bit higher than our testing data baseline of 77%. It was much lower than other model we tested. Therefore, we do not recommend a logistic regression model for this classification model.

2. KNN

At first, we don't know much about the structure of the dataset and it's a classification problem. So KNN seems a good way to use. Since KNN will measure the distance of different instances, we converted all the variables to numeric and scale. Also we split validation dataset to do the selection of k. However, the model run very slow because this algorithm would compute distance to all training records. Later we reduced the dimensions according to feature selection and test it on our test dataset. The accuracy was around 78% and not high as we expect. Thus we went on exploring other models.

3. Classification Tree

At our early stage, since this is a classification problem, we also tried the classification tree. However, the result is not ideal. We create dummy variables for those variable which has more than 32 levels. Since it is a very large dataset, splitting method will not be a big issue here. The data is splitted into 70% train and 30% validation data. The test accuracy from classification tree is around 74.5%, which is below the baseline. The low accuracy suggests that a single tree doesn't perform well since it uses a greedy approach to induce trees --- locally optimal splits are induced at every node of the tree. A single tree can also easily lead to overfitting problems. We discard the single tree model and try some other tree-based models.

4. Random Forest

When we were testing classification tree to get predictions, we noticed that overfitting is one critical problem that causes prediction results getting worse. In this case, we learned that if there are enough trees in the forest, the classifier won't overfit the model. Also the types of our features are very complicated with numeric, ordered and unordered categorical variables. And these variables are under different scales. Thus, we decided to test the random forest algorithm.

-Features input
First we tried features with text columns being dummy variables and didn't include amenities and host_verification columns and reached the accuracy of 83.6%. This is great improvement compared to applying logistic and simple classification tree.
Then we further include in several amenities and host_verification dummy variables which have higher importance score. The accuracy was raised to 84.4%
Then we transformed several ordered categorical variables to dummy variables and feed in the model. However, this didn't improve our accuracy on validation dataset.
Regarding feature selection, Random Forest do not need to select out significant features. But there could be two aspects of problems: 1) possible overfitting 2) taking too much computational power
So we conducted rfe to calculate variable importance score and based on correlation table, we selected out 101 variables as final features.
-Adjusting training and validation split
We tried different splitting percentage to train the model. We discover that the split of 0.7 of training dataset and 0.3 of validation dataset would gave us the highest accuracy on validation dataset of 84% and the accuracy on testing data of 84.4%.
With this result we could say overfitting is not an issue here. With the highest accuracy on testing data, we chose this one as the best model for prediction.

3. Xgboost
Because of the complexity of the model and the amount of features, our team tried boosting algorithm using gbtree for booster.

-Features input
For this part, we tried variables with all categorical variables as dummy variables, which gave us the accuracy around 83.5% on validation dataset. Then we tried the model with features selected by random forest which contain several dummy variables generated from amenities and host_verification columns, and dummy variables with only two classes, and the other categorical variables as numeric variables generated from factors.

-Scaling
We tried scaling the variables but the accuracy didn't rise significantly. This makes sense because the booster is based on decision trees and we only have very few outliers.

-Adjusting training and validation split
We tried different splitting percentage to train the model. We discover that the split of 0.7 of training dataset and 0.3 of validation dataset would gave us the highest accuracy on validation dataset of 84.5% and the accuracy on testing data of 83.95%.
With this result we could say overfitting is not an issue here. With the highest accuracy on testing data, we chose this one as the best model for prediction.

-Tuning parameters
After tuning parameters, we get the following accuracy of 84.5% on validation dataset.

```
[1]     val-error@0.85:0.250075
[21]    val-error@0.85:0.171029
[41]    val-error@0.85:0.163561
[61]    val-error@0.85:0.159027
[81]    val-error@0.85:0.156359
[100]   val-error@0.85:0.154926
```

Submitting the model to test on testing dataset, we got an accuracy of 83.95%. Increasing the rounds of iterations, we were able to gain higher accuracy result on validation data, with the highest of 85.01%. Out accuracy on testing data is lower than the accuracy on validation dataset, so there could exist overfitting problems. So we didn't adopt boosting model.

Appendix

I. Team Member Roles

Ziyi: KNN, data cleaning
Lingling: Logistic, data cleaning
Shuaiyu: data visualization, feature selection
Xinyi: Classification tree, data cleaning
Peiyan: XGBoost, data cleaning
Everyone: random forest

II. Data Processing (R code)

```
#data clean
dollar.to.numeric<-function(prices){
  price.new=unlist(lapply(prices, function(p){
    price=0
    if(!is.na(p) & nchar(p)>0 & substr(p, 1, 1)=="$"){
      price=round(as.numeric(substr(p, 2, nchar(p))))
    }
    price
  }))
  price.new[which(is.na(price.new))]=0
  return(price.new)

# price, NAs introduced by coercion
price.new = dollar.to.numeric(train$price)
#property_type
table(train$property_type)
train$property_type[is.na(train$property_type)] <- 'Apartment'
# security deposit
security.deposit.new = dollar.to.numeric(train$security_deposit)
zipcode.new=unlist(lapply(train$zipcode, function(z){
  z2=0
  if(!is.na(z) & nchar(z)>4){
```

```
  z2=as.numeric(substr(z, 1,5))
 }
 z2
 }))
zipcode.new[which(is.na(zipcode.new))]=0

##missing values
for (i in 1:nrow(train)){
    if(is.na(train$latitude[i])){
        train$latitude[i] <- as.numeric(mean(train$latitude[train$city==train$city[i]], na.rm=TRUE))
    }
}
last.14.var.table=cbind(price.new,
                factor(train$property_type),
                factor(train$require_guest_phone_verification),
                factor(train$require_guest_profile_picture),
                factor(train$requires_license),
                factor(train$room_type),
                security.deposit.new,
                factor(train$smart_location),
                factor(train$state),
                zipcode.new)

colnames(last.14.var.table)=c("price","property.type", "phone.verification", "profile.picture", "license", "room.type",
"security.deposit", "smart.location", "state", "zipcode")
all.amenities = c()
for(i in 1:nrow(train)){
 amenities = train$amenities[i]
 amenities.new = strsplit(amenities, ",", fixed=TRUE)[[1]]
 all.amenities = c(all.amenities, unlist(lapply(amenities.new, function(a){x=strsplit(gsub("[^[:alnum:] ]", "", a), "
+")[[1]]; paste0(x, collapse = " ")})))

}

all.amenities.variables = unique(all.amenities)
all.amenities.variables = all.amenities.variables[which(unlist(lapply(all.amenities.variables,
function(a){nchar(a)}))>1)]

a.names=names(table(all.amenities))
a.values=as.numeric(table(all.amenities))
all.amenities.variables = a.names[which(a.values>(i*0.1) & a.values<(i*0.9))]
all.amenities.matrix = matrix(0, nrow(train), length(all.amenities.variables))
for(j in 1:nrow(train)){
 amenities = train$amenities[j]
 amenities.new = strsplit(amenities, ",", fixed=TRUE)[[1]]
 this.amenities = unlist(lapply(amenities.new, function(a){x=strsplit(gsub("[^[:alnum:] ]", "", a), " +")[[1]];
paste0(x, collapse = " ")}))
 all.amenities.dummy=rep(0, length(all.amenities.variables))
 all.amenities.dummy[which(all.amenities.variables %in% this.amenities)] = 1
 all.amenities.matrix[j,]=all.amenities.dummy
}

colnames(all.amenities.matrix) = all.amenities.variables
orig.nums=as.numeric(rownames(varImp(results)))
new.names=rownames(varImp(results))
for(k in 1:length(orig.nums)){
 v = orig.nums[k]
```

```
  if(!is.na(v)){
    new.names[k]=all.amenities.variables[v]
  }

}

var.imp = cbind(new.names, varImp(results)[,1])
colnames(var.imp) = c("variableName", "importance")
write.table(var.imp, "d:/research/Collaborations/Xinyi/varibleSelection.txt", quote=FALSE, sep=",", col.names = TRUE)
```

III. Feature Importance (R code)

```
##---------------------------------------------------
## feature elimination
##---------------------------------------------------

y=as.factor(y)
control <- rfeControl(functions=rfFuncs, method="cv", number=10)
        # run the RFE algorithm
        results <- rfe(x, y, sizes=c(1:20), rfeControl=control)
        # summarize the results
        #print(results)
        # list the chosen features
        predictors(results)
        # plot the results
        plot(results, type=c("g", "o"))


#results <- rfe(x[1:5000,], y[1:5000], sizes=c(1:20), rfeControl=control)
chosen.vars = predictors(results)[which(as.numeric(varImp(results)>3)==1)]
```

IV. Model fit and output (R code)

i. Logistic Model

```
#amenities to convert...
cols = colnames(trainx)[c(53:260)]
trainx[cols] <- lapply(trainx[cols], factor)

# convert to factors
trainx$city_name=as.factor(trainx$city_name)
trainx$host_has_profile_pic=as.factor(trainx$host_has_profile_pic)
trainx$host_identity_verified=as.factor(trainx$host_identity_verified)
trainx$host_is_superhost=as.factor(trainx$host_is_superhost)
trainx$instant_bookable=as.factor(trainx$instant_bookable)
trainx$is_business_travel_ready=as.factor(trainx$is_business_travel_ready)
trainx$is_location_exact=as.factor(trainx$is_location_exact)
trainx$property_type=as.factor(trainx$property_type)
trainx$requires_license=as.factor(trainx$requires_license)
trainx$room_type=as.factor(trainx$room_type)
trainx$state=as.factor(trainx$state)
trainx$cancellation_policy=as.factor(trainx$bed_type)
trainx$cancellation_policy=as.factor(trainx$cancellation_policy)
trainx$country=as.factor(trainx$country)
trainx$country_code=as.factor(trainx$country_code)
trainx$host_response_time=as.factor(trainx$host_response_time)
trainx$instant_bookable=as.factor(trainx$instant_bookable)
```

```
trainx$name=as.factor(trainx$name)
trainx$require_guest_phone_verification=as.factor(trainx$require_guest_phone_verification)
trainx$require_guest_profile_picture=as.factor(trainx$require_guest_profile_picture)
trainx$license=as.factor(trainx$license)

# drop columns - important score
train_all_with_dum <- subset(trainx, select = -c(54:55,57:62,64,66:68,70,72,73,76,77:82,84,86:88,91:94,96,98:104,107,109:112,114:119,121:130,132,133,135:142,145:147,149:153,155,156,159,161:170,172:176,179,180,182,183,185:188,190:191,195:211,213:222,224:226,228:240,247:249,251:253,255:258,260))


# split into train and test datasets
train_x = subset(train_all_with_dum,train_all_with_dum$label=='train')
test_x = subset(train_all_with_dum,train_all_with_dum$label=='test')

# combind train x and train y
train=cbind(train_x, trainy_clean_0503)
train<- subset(train,select = -c(label))
train$high_booking_rate=as.factor(train$high_booking_rate)

# select samples
set.seed(12345)
train.total = sample(nrow(train), .5*nrow(train))
sampleset = train[train.total,]
train.indicies = sample(nrow(sampleset), .9*nrow(sampleset))
train.samples = sampleset[train.indicies,]
test.samples = sampleset[-train.indicies,]

#logistic model--accuracy
fit2 <- glm(train.samples$high_booking_rate~.,data=train.samples,family="binomial")
summary(fit2)
log_preds <- predict(fit2,newdata=test.samples,type="response")
log_class <- ifelse(log_preds>.5,1,0)
```

(Partial Output)

```
                    Estimate Std. Error z value Pr(>|z|)
(Intercept)       -3.153e+13  4.783e+13  -0.659 0.509742
access            -6.758e-04  5.263e-04  -1.284 0.199121
accommodates       1.067e-01  1.442e-02   7.395 1.42e-13 ***
availability_30   -3.252e-02  3.894e-03  -8.351  < 2e-16 ***
availability_365   1.363e-03  1.440e-04   9.464  < 2e-16 ***
availability_60   -2.735e-03  3.858e-03  -0.709 0.478440
availability_90    1.106e-02  1.996e-03   5.538 3.07e-08 ***
bathrooms         -2.399e-02  3.409e-02  -0.704 0.481651
cleaning_fee      -7.095e-03  4.834e-04 -14.678  < 2e-16 ***
description        3.503e-03  5.360e-04   6.536 6.33e-11 ***
```

ii. KNN

```
# drop columns - dummy column
train_all_with_dum <- subset(trainx, select = -c(54:55,57:62,64,66:68,70,72,73,76,77:82,84,86:88,91:94,96,98:104,107,109:112,114:119,121:130,132,133,135:142,145:147,149:153,155,156,159,161:170,172:176,179,180,182,183,185:188,190:191,195:211,213:222,224:226,228:240,247:249,251:253,255:258,260))


# split into train and test datasets
```

```
train_x = subset(train_all_with_dum,train_all_with_dum$label=='train')
test_x = subset(train_all_with_dum,train_all_with_dum$label=='test')

# combind train x and train y
train=cbind(train_x, trainy_clean_0503)
train<- subset(train,select = -c(label))
test_x<-subset(test_x,select=-c(label))
#train$high_booking_rate=as.numeric(train$high_booking_rate)
data<-train
library(class)

data <- sapply(data, function(x)
{ if(is.character(x)) as.factor(x) else x })

data <- sapply(data, function(x)
{ if(is.factor(x)) as.numeric(x) else x })

#sapply(data,class)

data <- scale(data)
set.seed(12345)
test_instn = sample(nrow(data), 0.3*nrow(data))
data_test <- data[test_instn,]
data_rest <- data[-test_instn,]

valid_instn = sample(nrow(data_rest), 0.25*nrow(data_rest))
data_valid <- data_rest[valid_instn,]
data_train <- data_rest[-valid_instn,]

train.samples.Y=data_train$high_booking_rate
train.samples.X=subset(data_train,select=c(-high_booking_rate))

validation.x=subset(data_valid,select=c(-high_booking_rate))
validation.y=data_valid$high_booking_rate

test_X=subset(data_test,select=c(-high_booking_rate))
test_Y=data_test$high_booking_rate

#k=3
knn.pred_train3=knn(train.samples.X,train.samples.X,train.samples.Y,k=3)
knn.pred_valid3=knn(train.samples.X,validation.x,train.samples.Y,k=3)
valid_acc3=sum(ifelse(knn.pred_valid3 == validation.y,1,0))/nrow(data_valid)

#k=5
knn.pred_train5=knn(train.samples.X,train.samples.X,train.samples.Y,k=5)
knn.pred_valid5=knn(train.samples.X,validation.x,train.samples.Y,k=5)
valid_acc5=sum(ifelse(knn.pred_valid5 == validation.y,1,0))/nrow(data_valid)

#k=7
knn.pred_train10=knn(train.samples.X,train.samples.X,train.samples.Y,k=10)
knn.pred_valid10=knn(train.samples.X,validation.x,train.samples.Y,k=10)
valid_acc10=sum(ifelse(knn.pred_valid10 == validation.y,1,0))/nrow(data_valid)


accuracy<-c(valid_acc3,valid_acc5,valid_acc7)

accuracy
```

```
#choose k=5 for testing

knn.pred_train5=knn(train.samples.X,train.samples.X,train.samples.Y,k=5)
knn.pred_test5=knn(train.samples.X,test.X,train.samples.Y,k=5)
test_acc5=sum(ifelse(knn.pred_test5 == test.Y,1,0))/nrow(data_test)
test_acc5
```

iv. Classification Tree

```
set.seed(12345)
trainx$high_booking_rate=as.factor(trainx$high_booking_rate)
trainx$city_name=as.factor(trainx$city_name)
trainx$host_has_profile_pic=as.factor(trainx$host_has_profile_pic)
trainx$host_identity_verified=as.factor(trainx$host_identity_verified)
trainx$host_is_superhost=as.factor(trainx$host_is_superhost)
trainx$instant_bookable=as.factor(trainx$instant_bookable)
trainx$is_business_travel_ready=as.factor(trainx$is_business_travel_ready)
trainx$is_location_exact=as.factor(trainx$is_location_exact)
trainx$property_type=as.factor(trainx$property_type)
trainx$requires_license=as.factor(trainx$requires_license)
trainx$room_type=as.factor(trainx$room_type)
trainx$state=as.factor(trainx$state)

trainx$bed_type=as.factor(trainx$bed_type)
trainx$cancellation_policy=as.factor(trainx$cancellation_policy)
trainx$country=as.factor(trainx$country)
trainx$country_code=as.factor(trainx$country_code)
trainx$host_response_time=as.factor(trainx$host_response_time)
trainx$instant_bookable=as.factor(trainx$instant_bookable)
trainx$name=as.factor(trainx$name)
trainx$require_guest_phone_verification=as.factor(trainx$require_guest_phone_verification)
trainx$require_guest_profile_picture=as.factor(trainx$require_guest_profile_picture)
trainx$license=as.factor(trainx$license)
property=model.matrix(~property_type+0,data=trainx)
trainx=data.frame(trainx,property)

train_x = subset(trainx,trainx$label=='train')
test_x = subset(trainx,trainx$label=='test')
train_x<- subset(train_x,select = -c(label,property_type))
test_x = subset(test_x,select = -c(label,property_type))

trainx=cbind(train_x,trainy_clean_0503)

cols=c(53:305)
trainx[cols] <- lapply(trainx[cols], factor)


test_instn = sample(nrow(trainx), 0.3*nrow(trainx))
data_test <- trainx[test_instn,]
data_rest <- trainx[-test_instn,]

valid_instn = sample(nrow(data_rest), 0.25*nrow(data_rest))
data_valid <- data_rest[valid_instn,]
data_train <- data_rest[-valid_instn,]

sapply(trainx,class)
```

```
sapply(trainx,function(trainx) sum(is.na(trainx)))

library(tree)
booking.tree=tree(data_train$high_booking_rate~.,data_train)
summary(booking.tree)
plot(booking.tree)
text(booking.tree)

tree_probs <- predict(booking.tree,newdata=data_valid)[,2]
tree_pred_valid <- prediction(tree_probs,data_valid$high_booking_rate)

best = which.max(slot(performance(tree_pred_valid, measure = "acc"),"y.values")[[1]])
valid_acc = slot(performance(tree_pred_valid, measure = "acc"),"y.values")[[1]][best]

tree_test_probs = predict(booking.tree,newdata=data_test)[,2]
best = which.max(slot(performance(tree_pred_valid, measure = "acc"),"y.values")[[1]])
tree.cutoff = slot(performance(tree_pred_valid, measure = "acc"),"x.values")[[1]][best]
tree_test <- ifelse(tree_test_probs>0.75,1,0)
tree_accuracy = sum(ifelse(data_test$high_booking_rate==tree_test,1,0))/nrow(data_test)
tree_accuracy
```

v. Random Forest

```
cols = colnames(trainx)[c(53:260)]
trainx[cols] <- lapply(trainx[cols], factor)
# convert to factors
trainx$city_name=as.factor(trainx$city_name)
trainx$host_has_profile_pic=as.factor(trainx$host_has_profile_pic)
trainx$host_identity_verified=as.factor(trainx$host_identity_verified)
trainx$host_is_superhost=as.factor(trainx$host_is_superhost)
trainx$instant_bookable=as.factor(trainx$instant_bookable)
trainx$is_business_travel_ready=as.factor(trainx$is_business_travel_ready)
trainx$is_location_exact=as.factor(trainx$is_location_exact)
trainx$property_type=as.factor(trainx$property_type)
trainx$requires_license=as.factor(trainx$requires_license)
trainx$room_type=as.factor(trainx$room_type)
trainx$state=as.factor(trainx$state)

trainx$bed_type=as.factor(trainx$bed_type)
trainx$cancellation_policy=as.factor(trainx$cancellation_policy)
trainx$country=as.factor(trainx$country)
trainx$country_code=as.factor(trainx$country_code)
trainx$host_response_time=as.factor(trainx$host_response_time)
trainx$instant_bookable=as.factor(trainx$instant_bookable)
trainx$name=as.factor(trainx$name)
trainx$require_guest_phone_verification=as.factor(trainx$require_guest_phone_verification)
trainx$require_guest_profile_picture=as.factor(trainx$require_guest_profile_picture)
trainx$license=as.factor(trainx$license)

# drop columns - dummy column
train_all_with_dum <- subset(trainx, select = -
c(54:55,57:62,64,66:68,70,72,73,76,77:82,84,86:88,91:94,96,98:104,107,109:112,114:119,121:130,132,133,135:14
2,145:147,149:153,155,156,159,161:170,172:176,179,180,182,183,185:188,190:191,195:211,213:222,224:226,228:
240,247:249,251:253,255:258,260))


# split into train and test datasets
```

```
train_x = subset(train_all_with_dum,train_all_with_dum$label=='train')
test_x = subset(train_all_with_dum,train_all_with_dum$label=='test')

# combind train x and train y
train=cbind(train_x, trainy_clean_0503)
train<- subset(train,select = -c(label))
train$high_booking_rate=as.factor(train$high_booking_rate)

# check na and class
sapply(train,class)
sapply(train,function(train) sum(is.na(train)))

# select samples -total:100000,trian:70000,test:30000
set.seed(12345)
train.total = sample(nrow(train), 1*nrow(train))
sampleset = train[train.total,]
train.indicies = sample(nrow(sampleset), .7*nrow(sampleset))
train.samples = sampleset[train.indicies,]
test.samples = sampleset[-train.indicies,]

library(randomForest)
sapply(train.samples,class)
sapply(train.samples,function(train.samples) sum(is.na(train.samples)))
fit1 <- randomForest(train.samples$high_booking_rate~., data=train.samples)
rf_preds <- predict(fit1,newdata=test.samples,type="response")
test = table(test.samples$high_booking_rate,rf_preds)
test.accuracy=sum(diag(test))/sum(test)
test.accuracy

test_pred <- predict(fit1,newdata=test_x,type="response")


test_x[ ,(ncol(test_x)+1)] <- test_pred
write.csv(test_x, '~/Desktop/test0504.csv')
```

vi. Boosting

```
cols = colnames(trainx)[c(53:260)]
trainx[cols] <- lapply(trainx[cols], factor)
# convert to factors
trainx$city_name=as.factor(trainx$city_name)
trainx$host_has_profile_pic=as.factor(trainx$host_has_profile_pic)
trainx$host_identity_verified=as.factor(trainx$host_identity_verified)
trainx$host_is_superhost=as.factor(trainx$host_is_superhost)
trainx$instant_bookable=as.factor(trainx$instant_bookable)
trainx$is_business_travel_ready=as.factor(trainx$is_business_travel_ready)
trainx$is_location_exact=as.factor(trainx$is_location_exact)
trainx$property_type=as.factor(trainx$property_type)
trainx$requires_license=as.factor(trainx$requires_license)
trainx$room_type=as.factor(trainx$room_type)
trainx$state=as.factor(trainx$state)
trainx$bed_type=as.factor(trainx$bed_type)
trainx$cancellation_policy=as.factor(trainx$cancellation_policy)
trainx$country=as.factor(trainx$country)
trainx$country_code=as.factor(trainx$country_code)
trainx$host_response_time=as.factor(trainx$host_response_time)
trainx$instant_bookable=as.factor(trainx$instant_bookable)
```

```
trainx$name=as.factor(trainx$name)
trainx$require_guest_phone_verification=as.factor(trainx$require_guest_phone_verification)
trainx$require_guest_profile_picture=as.factor(trainx$require_guest_profile_picture)
trainx$license=as.factor(trainx$license)
# drop columns - dummy column
train_all_with_dum <- subset(trainx, select = -c(54:55,57:62,64,66:68,70,72,73,76,77:82,84,86:88,91:94,96,98:104,107,109:112,114:119,121:130,132,133,135:142,145:147,149:153,155,156,159,161:170,172:176,179,180,182,183,185:188,190:191,195:211,213:222,224:226,228:240,247:249,251:253,255:258,260))

# split into train and test datasets
train_x = subset(train_all_with_dum,train_all_with_dum$label=='train')
test_x = subset(train_all_with_dum,train_all_with_dum$label=='test')
# combind train x and train y
train=cbind(train_x, trainy_clean_0503)
train<- subset(train,select = -c(label))
test_x<-subset(test_x,select=-c(label))
train$high_booking_rate=as.numeric(train$high_booking_rate)
# check na and class
sapply(train,class)
sapply(train,function(train) sum(is.na(train)))
# select samples -total:100000,trian:70000,test:30000
set.seed(12345)
train.total = sample(nrow(train), 1*nrow(train))
sampleset = train[train.total,]
train.indicies = sample(nrow(sampleset), .7*nrow(sampleset))
train.samples = sampleset[train.indicies,]
train.samples.y=train.samples$high_booking_rate
train.samples.x=subset(train.samples,select=c(-high_booking_rate))
test.samples = sampleset[-train.indicies,]
test.samples.y=test.samples$high_booking_rate
test.samples.x=subset(test.samples,select=c(-high_booking_rate))
library(xgboost)
model_train <- xgb.DMatrix(data = data.matrix(train.samples.x), label =train.samples.y)
model_val <- xgb.DMatrix(data = data.matrix(test.samples.x), label = test.samples.y)
xgb_test <- xgb.DMatrix(data = data.matrix(test_x))

params <- list(objective = "binary:logistic",
        booster = "gbtree",
        eval_metric = "error@0.85",
        nthread = 7,
        eta = 0.10,
        max_depth = 30,
        gamma = 0.1,
        subsample = 0.9,
        colsample_bytree = 0.8,
        scale_pos_weight = 50,
        nrounds = 100)
## Build model
myxgb_model <- xgb.train(params, model_train,
                params$nrounds,
                list(val = model_val),
                print_every_n = 20)
test_pred <- predict(myxgb_model,newdata=xgb_test,type="response")
prediction<-as.numeric(test_pred>0.85)
write.csv(prediction, 'C:/Users/yupei/Desktop/project/prediction0508.csv')
```