Airbnb price prediction

Applied Analytics in Framework and Methods
Yufan Luo
Nov. 2018

Note: All parameters in this document are not necessarily the exact value of the real ones used in the competition.

Introduction

This is a competition about a list of over 25,000 Airbnb rentals in New York City (https://www.kaggle.com/c/airbnblala1). The goal of this competition is to predict the price for a rental using over 90 variables on the property, host, and past reviews.

During this month, I materialized the concepts and methods I have learned in class. I explored the data, dealt with anormalies, did necessary transformation, then tried to train different models on it. After that, I compared these methods and knew the pros and cons of them. Finally I got a high rank on the public leaderboard of the competition.

Data Processing and Exploratory Data Analysis

Load data and deal with NAs.

Load data.

```
data = read.csv('analysisData.csv',stringsAsFactors = F)
```

Check NAs.

```
num.NA = sort(sapply(data, function(x) { sum(is.na(x))} ))
num.NA[which(num.NA!=0)]
```

```
##             beds    cleaning_fee security_deposit     weekly_price
##               17            5702            11827            25307
##      square_feet   monthly_price    thumbnail_url       medium_url
##            28801           28901            29142            29142
##   xl_picture_url         license
##            29142           29142
```

Eliminate attributes with more than 80% NAs.

```r
remain.col = names(num.NA[which(num.NA<0.8*dim(data)[1])])
data.sub = data[,remain.col]
NA.new = sort(sapply(data.sub, function(x) { sum(is.na(x))} ))
NA.new[which(NA.new!=0)]
```

```
##            beds     cleaning_fee security_deposit
##              17             5702            11827
```

Fill remaining NAs with medians.

```r
data.sub$cleaning_fee[which(is.na(data.sub$cleaning_fee))] = median(data.sub$cleaning_fe
e,na.rm=T)
data.sub$beds[which(is.na(data.sub$beds))] = median(data.sub$beds,na.rm=T)
data.sub$security_deposit[which(is.na(data.sub$security_deposit))] = median(data.sub$sec
urity_deposit,na.rm=T)
```

Deal with categorical variables.

For attribotes that have a sense of distance, transfer them into different levels.

```r
data.sub$cancellation_policy[which(data.sub$cancellation_policy=='flexible')] = 1
data.sub$cancellation_policy[which(data.sub$cancellation_policy=='moderate')] = 2
data.sub$cancellation_policy[which(data.sub$cancellation_policy=='strict')] = 3
data.sub$cancellation_policy[which(data.sub$cancellation_policy=='super_strict_30')] = 4
data.sub$cancellation_policy[which(data.sub$cancellation_policy=='super_strict_60')] = 4
```

– Omitted –

For attribotes that have no sense of distance, do one-hot encoding.

I should have used **ifelse** to write more clearly :)

```
#one hot
data.sub$neighbourhood_Bronx=0
data.sub$neighbourhood_Bronx[which(data.sub$neighbourhood_group_cleansed=='Bronx')] = 1
data.sub$neighbourhood_Brooklyn=0
data.sub$neighbourhood_Brooklyn[which(data.sub$neighbourhood_group_cleansed=='Brooklyn'
)] = 1
data.sub$neighbourhood_Manhattan=0
data.sub$neighbourhood_Manhattan[which(data.sub$neighbourhood_group_cleansed=='Manhatta
n')] = 1
data.sub$neighbourhood_Queens=0
data.sub$neighbourhood_Queens[which(data.sub$neighbourhood_group_cleansed=='Queens')] =
1
data.sub$neighbourhood_Staten=0
data.sub$neighbourhood_Staten[which(data.sub$neighbourhood_group_cleansed=='Staten Islan
d')] = 1
```

– Omitted –

For amenity, I caught keywords of different amenities and did one-hot encoding

```
#amenity
data.sub$wifi = 0
data.sub$wifi[which((grepl("Wifi", data.sub$amenities)==T) | (grepl("Internet", data.sub
$amenities)==T))] = 1
data.sub$heat = 0
data.sub$heat[which(grepl("Heating", data.sub$amenities)==T)] = 1
data.sub$air = 0
data.sub$air[which(grepl("Air conditioning", data.sub$amenities)==T)] = 1
data.sub$kitcken = 0
data.sub$kitcken[which(grepl("Kitchen", data.sub$amenities)==T)] = 1
data.sub$shampoo = 0
data.sub$shampoo[which(grepl("Shampoo", data.sub$amenities)==T)] = 1
data.sub$essential = 0
data.sub$essential[which(grepl("Essentials", data.sub$amenities)==T)] = 1
data.sub$elevator = 0
data.sub$elevator[which(grepl("Elevator", data.sub$amenities)==T)] = 1
data.sub$tv = 0
data.sub$tv[which((grepl("TV", data.sub$amenities)==T) | (grepl("Cable TV", data.sub$ame
nities)==T))] = 1
data.sub$gym = 0
data.sub$gym[which(grepl("Gym", data.sub$amenities)==T)] = 1
data.sub$washer = 0
data.sub$washer[which(grepl("Washer", data.sub$amenities)==T)] = 1
data.sub$dryer = 0
data.sub$dryer[which(grepl("Dryer", data.sub$amenities)==T)] = 1
data.sub$fridge = 0
data.sub$fridge[which(grepl("Refrigerator", data.sub$amenities)==T)] = 1
data.sub$self_check = 0
data.sub$self_check[which(grepl("Self check-in", data.sub$amenities)==T)] = 1
data.sub$hair = 0
data.sub$hair[which(grepl("Hair dryer", data.sub$amenities)==T)] = 1
data.sub$smart = 0
data.sub$smart[which(grepl("Smart lock", data.sub$amenities)==T)] = 1
data.sub$aid = 0
data.sub$aid[which(grepl("First aid kit", data.sub$amenities)==T)] = 1
data.sub$hanger = 0
data.sub$hanger[which(grepl("Hangers", data.sub$amenities)==T)] = 1
data.sub$co = 0
data.sub$co[which(grepl("Carbon monoxide detector", data.sub$amenities)==T)] = 1
```

Select all useable attributes.

```
data.sub = data.sub[,c('price','cleaning_fee','beds','host_response_time',
                       'host_is_superhost','neighbourhood_Bronx','neighbourhood_Brookly
n',
                       'neighbourhood_Manhattan','neighbourhood_Queens','neighbourhood_S
taten',
                       'latitude','longitude','room_apt','room_private','room_shared',
                       'accommodates','bathrooms','bedrooms','guests_included','transit'
,'access',
                       'minimum_nights','calculated_host_listings_count','property_type'
,
                       'review_scores_rating','reviews_per_month','is_business_travel_re
ady',
                       'review_scores_cleanliness','review_scores_checkin','is_location_
exact',
                       'review_scores_location','review_scores_value','cancellation_poli
cy',
                       'availability_30','availability_60','availability_90','availabili
ty_365','bed_airbed',
                       'bed_couch','bed_futon','bed_sofa','bed_real','instant_bookable',
                       'wifi','heat','air','kitcken','shampoo','essential','tv','gym',
                       'washer','dryer','fridge','self_check','hair','smart','aid','hang
er','co')]
```

Exploratory Data Analysis

Do random forest on a subset of all attributes (because it's so time comsuming to include all records), see importance.

```
library(randomForest)
set.seed(1)
sample.data = sample(1:nrow(data.sub),0.3*nrow(data.sub))
data.try = data.sub[sample.data,]

rf = randomForest(price~.,data.try,importance=T)
importance(rf)
```

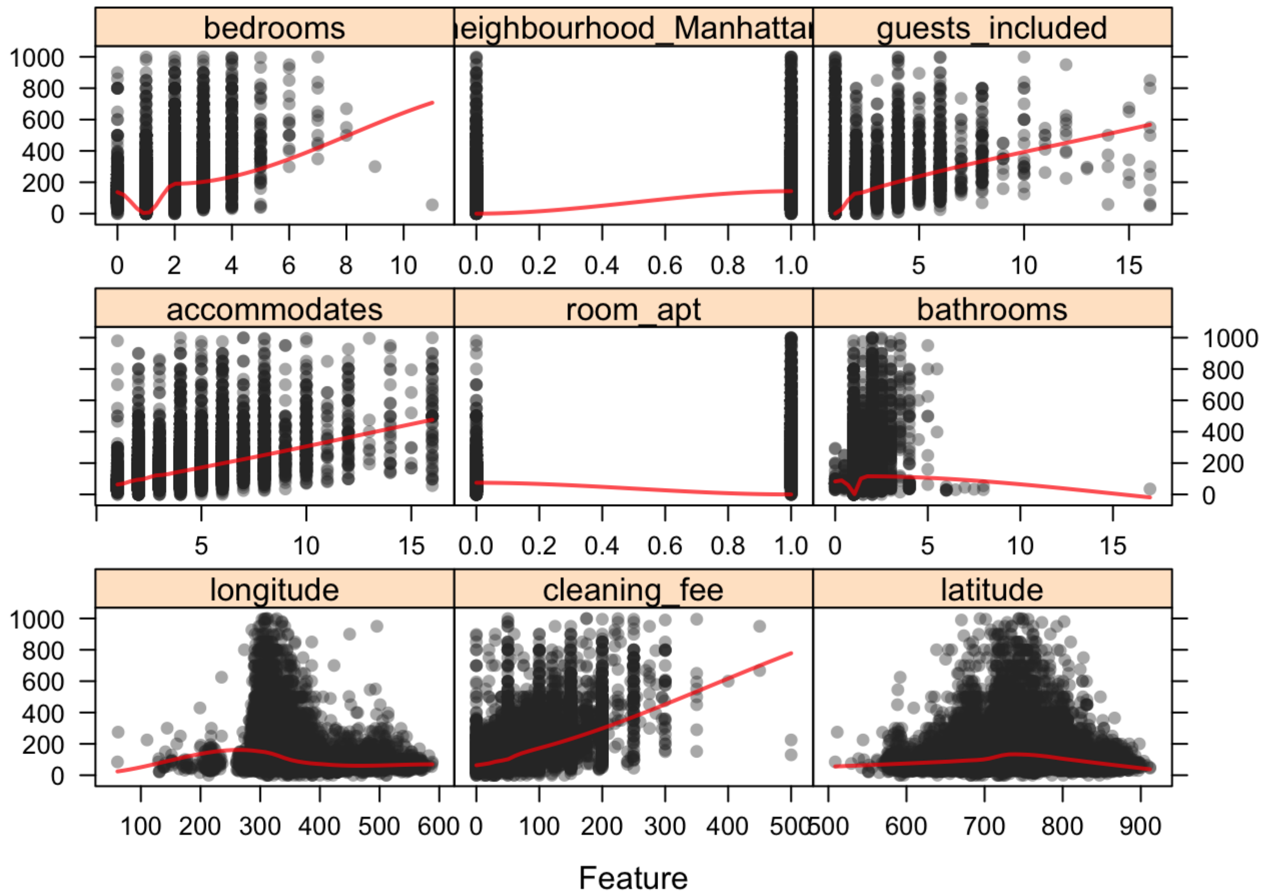
```
##                                 %IncMSE IncNodePurity
## cleaning_fee                 27.6664420  10553105.603
## beds                         18.1189347   2581369.922
## host_response_time           12.6273519    670676.476
## host_is_superhost             4.2796173    216426.016
## neighbourhood_Bronx           0.9213231      9598.034
## neighbourhood_Brooklyn       11.6026594    447463.052
## neighbourhood_Manhattan      25.9010458   2311424.786
## neighbourhood_Queens          8.6955992    126142.991
## neighbourhood_Staten          2.4668507     23367.985
## latitude                     40.8288898   4534751.536
## longitude                    56.5144212   8257354.560
## room_apt                     33.3237133  11765217.870
## room_private                 18.4193463   6988188.390
## room_shared                  11.3353821    333783.727
## accommodates                 27.2929195   6668051.785
## bathrooms                    32.0152236   8269373.133
## bedrooms                     26.1620113   8134318.576
## guests_included              15.9452839   1386828.583
## transit                       1.4053981    240579.022
## access                        3.2137113    234292.627
## minimum_nights               11.2991689   1275865.522
## calculated_host_listings_count 12.5915788    797417.745
## property_type                15.5736505    673715.220
## review_scores_rating         12.6137294   1457769.795
## reviews_per_month            14.8195643   2186952.222
## is_business_travel_ready     -1.0459319    165676.366
## review_scores_cleanliness     3.9264753    487418.862
## review_scores_checkin         2.0730552    407077.216
## is_location_exact             1.9518663    234270.710
## review_scores_location       12.3758778    741517.680
## review_scores_value           6.0833562    535975.856
## cancellation_policy           6.0809049    417960.983
## availability_30              18.7497447   1287502.734
## availability_60              22.3150786   1474819.613
## availability_90              25.7122595   1683590.862
## availability_365             20.2225013   1989933.094
## bed_airbed                    2.0459613     25188.159
## bed_couch                    -1.3733026      1207.816
## bed_futon                     1.6817964      8438.853
## bed_sofa                     -2.6645519      9215.523
## bed_real                      1.3888610     26596.397
## instant_bookable              6.0226291    265314.880
## wifi                          1.5728091     39948.745
## heat                         -0.2247077     81623.031
## air                           6.7817133    159998.331
## kitcken                       4.3387080    113883.460
## shampoo                       2.7251872    280988.264
## essential                     0.8785438    144945.983
## tv                            8.8618884    299470.330
## gym                           9.4172084    362432.225
## washer                        9.9447984    556172.091
## dryer                         9.0767780    573513.793
```

```
## fridge                          0.1765611    239884.419
## self_check                      3.7506057    236412.158
## hair                            3.0369997    208750.944
## smart                          -2.2168776     67048.974
## aid                             3.8374815    283599.081
## hanger                          3.6063135    207193.688
## co                              4.7815449    243998.597
```

Print scatter plots betweeen price and highly important attributes obtained before.

The red lines here are rough estimates.

```
library(mlbench)
library(lattice)
library(caret)
theme1 <- trellis.par.get()
theme1$plot.symbol$col = rgb(.2, .2, .2, .4)
theme1$plot.symbol$pch = 16
theme1$plot.line$col = rgb(1, 0, 0, .7)
theme1$plot.line$lwd <- 2
trellis.par.set(theme1)
featurePlot(x = data.sub[,c(12,2,11,16,13,17,18,8,19)],
            y = data.sub$price,
            plot = "scatter",
            type = c("p","smooth"),
            span = .5,
            layout = c(3,3))
```



Split training and test set

```
set.seed(1)
sample_row = sample(1:dim(data)[1],0.8*dim(data[1]))
train.sub = data.sub[sample_row,]
test.sub = data.sub[-sample_row,]
```

Train model

Linear Regression

```
linear.mod = lm(price~.,data=train.sub)
summary(linear.mod)
```

```
##
## Call:
## lm(formula = price ~ ., data = train.sub)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -513.59  -32.33   -4.50   23.40  871.97
##
## Coefficients: (3 not defined because of singularities)
##                                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)                     -0.015894  13.079159  -0.001 0.999030
## cleaning_fee                     0.483016   0.013900  34.750  < 2e-16 ***
## beds                            -4.616809   0.768909  -6.004 1.95e-09 ***
## host_response_time              -0.450897   0.129221  -3.489 0.000485 ***
## host_is_superhost                4.625015   1.252618   3.692 0.000223 ***
## neighbourhood_Bronx            132.697231   7.642328  17.363  < 2e-16 ***
## neighbourhood_Brooklyn         109.679557   6.310540  17.380  < 2e-16 ***
## neighbourhood_Manhattan        154.626882   6.379146  24.239  < 2e-16 ***
## neighbourhood_Queens           134.882308   6.933293  19.454  < 2e-16 ***
## neighbourhood_Staten                   NA         NA      NA       NA
## latitude                        -0.116902   0.013382  -8.736  < 2e-16 ***
## longitude                       -0.410909   0.016528 -24.862  < 2e-16 ***
## room_apt                        70.627416   3.227816  21.881  < 2e-16 ***
## room_private                    22.196031   3.081706   7.203 6.09e-13 ***
## room_shared                            NA         NA      NA       NA
## accommodates                    13.367553   0.496229  26.938  < 2e-16 ***
## bathrooms                       31.562690   1.190474  26.513  < 2e-16 ***
## bedrooms                        24.420796   0.900392  27.122  < 2e-16 ***
## guests_included                  1.765270   0.475089   3.716 0.000203 ***
## transit                         -1.689372   1.193169  -1.416 0.156827
## access                           0.023382   1.154874   0.020 0.983847
## minimum_nights                  -0.336733   0.046728  -7.206 5.93e-13 ***
## calculated_host_listings_count  -0.478732   0.212962  -2.248 0.024587 *
## property_type                  -12.257538   1.285712  -9.534  < 2e-16 ***
## review_scores_rating             0.678627   0.094242   7.201 6.16e-13 ***
## reviews_per_month               -2.733215   0.340760  -8.021 1.10e-15 ***
## is_business_travel_ready         5.066638   2.017680   2.511 0.012042 *
## review_scores_cleanliness        2.726692   0.580178   4.700 2.62e-06 ***
## review_scores_checkin           -3.578774   0.743517  -4.813 1.49e-06 ***
## is_location_exact               -0.122133   1.161810  -0.105 0.916279
## review_scores_location           8.981823   0.641491  14.001  < 2e-16 ***
## review_scores_value             -8.383240   0.773438 -10.839  < 2e-16 ***
## cancellation_policy             -1.964109   0.573527  -3.425 0.000617 ***
## availability_30                  0.495199   0.127426   3.886 0.000102 ***
## availability_60                 -0.013401   0.133925  -0.100 0.920295
## availability_90                  0.093050   0.069441   1.340 0.180262
## availability_365                 0.014513   0.004739   3.063 0.002196 **
## bed_airbed                       1.986801   5.874225   0.338 0.735198
## bed_couch                        6.263839   9.286853   0.674 0.500010
## bed_futon                       -2.258071   4.394132  -0.514 0.607338
## bed_sofa                        -7.811496   4.590986  -1.701 0.088865 .
## bed_real                               NA         NA      NA       NA
## instant_bookable                 0.496522   1.005800   0.494 0.621552
```

```
## wifi                          2.113316   3.349441   0.631 0.528082
## heat                         -0.770671   1.948324  -0.396 0.692436
## air                           4.101904   1.283601   3.196 0.001397 **
## kitcken                      -3.640013   1.772215  -2.054 0.039993 *
## shampoo                       5.797547   1.014886   5.713 1.13e-08 ***
## essential                    -1.579805   1.517733  -1.041 0.297934
## tv                            6.654107   0.976409   6.815 9.67e-12 ***
## gym                          25.578754   1.809747  14.134  < 2e-16 ***
## washer                        2.968060   3.037970   0.977 0.328585
## dryer                         8.200600   3.042474   2.695 0.007036 **
## fridge                       -2.393151   1.153396  -2.075 0.038009 *
## self_check                   -3.453858   1.334229  -2.589 0.009641 **
## hair                         -0.795430   1.079602  -0.737 0.461263
## smart                        -0.861765   3.810149  -0.226 0.821066
## aid                           1.019899   0.985249   1.035 0.300601
## hanger                       -3.389564   1.117325  -3.034 0.002419 **
## co                           -0.670325   0.981882  -0.683 0.494807
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 65.28 on 23256 degrees of freedom
## Multiple R-squared:  0.6117, Adjusted R-squared:  0.6108
## F-statistic: 654.2 on 56 and 23256 DF,  p-value: < 2.2e-16
```

```
train.pred = predict(linear.mod,train.sub)
test.pred = predict(linear.mod,test.sub)
train.error = sqrt(mean((train.pred-train.sub$price)^2))
train.error
```

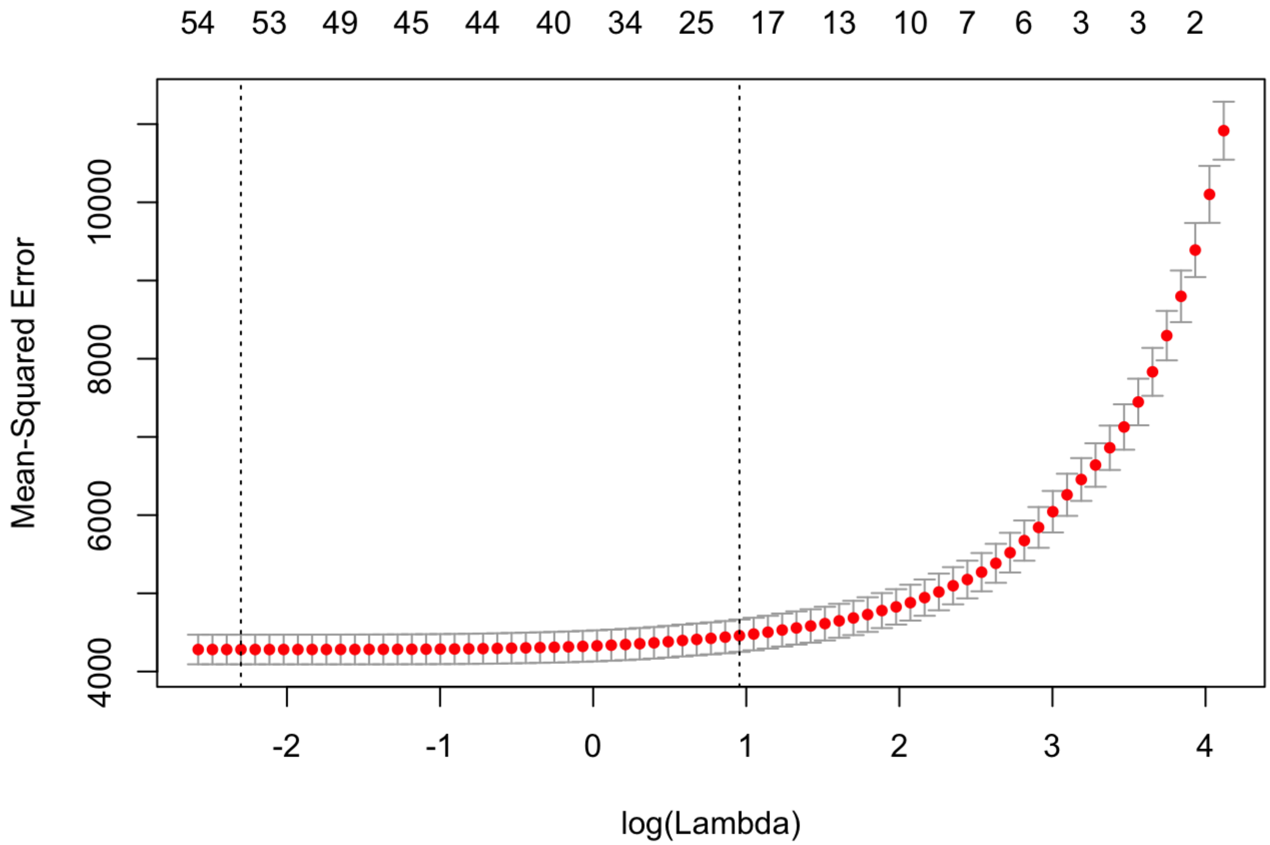
```
## [1] 65.20386
```

```
test.error = sqrt(mean((test.pred-test.sub$price)^2))
test.error
```

```
## [1] 62.45154
```

Lasso Regression

```
library(glmnet)
ind = train.sub[,-1]
ind <- model.matrix( ~., ind)
dep = train.sub[,1]
test.ind = test.sub[,-1]
test.dep = test.sub[,1]
test.ind = model.matrix(~.,test.ind)
cvfit <- cv.glmnet(ind, dep)
plot(cvfit, label = T)
```




```
cvfit$lambda.min
```

```
## [1] 0.1001886
```

```
cvfit$lambda.1se
```

```
## [1] 2.599918
```

```
coef(cvfit,s='lambda.1se')
```

```
## 61 x 1 sparse Matrix of class "dgCMatrix"
##                                          1
## (Intercept)                    14.52757534
## (Intercept)                     .
## cleaning_fee                    0.53381351
## beds                            .
## host_response_time              .
## host_is_superhost               .
## neighbourhood_Bronx             .
## neighbourhood_Brooklyn         -5.67415950
## neighbourhood_Manhattan        25.06750446
## neighbourhood_Queens            .
## neighbourhood_Staten          -53.57278272
## latitude                        .
## longitude                      -0.30086676
## room_apt                       45.97551392
## room_private                    .
## room_shared                    -5.15599225
## accommodates                   12.21224253
## bathrooms                      25.70450903
## bedrooms                       20.02576935
## guests_included                 .
## transit                         .
## access                          .
## minimum_nights                 -0.02087966
## calculated_host_listings_count  .
## property_type                  -3.98254661
## review_scores_rating            .
## reviews_per_month              -0.88203693
## is_business_travel_ready        .
## review_scores_cleanliness       .
## review_scores_checkin           .
## is_location_exact               .
## review_scores_location          6.80800628
## review_scores_value             .
## cancellation_policy             .
## availability_30                 0.50118405
## availability_60                 .
## availability_90                 .
## availability_365                .
## bed_airbed                      .
## bed_couch                       .
## bed_futon                       .
## bed_sofa                        .
## bed_real                        .
## instant_bookable                .
## wifi                            .
## heat                            .
## air                             0.58497301
## kitcken                         .
## shampoo                         0.27679191
## essential                       .
## tv                              5.10022660
```

```
## gym                               19.31950488
## washer                             .
## dryer                              8.20278720
## fridge                             .
## self_check                         .
## hair                               .
## smart                              .
## aid                                .
## hanger                             .
## co                                 .
```

```
pred.cv <- predict(cvfit,s=cvfit$lambda.1se, test.ind)
pred.cv <- as.numeric(pred.cv)
test.error = sqrt(mean((pred.cv-test.sub$price)^2))
test.error
```

```
## [1] 63.64972
```

Gradient Boosting Trees

Methods based on trees are robust to a large number of predictors. Thus I did not make selection on predictors.

```
library(gbm)
boost=gbm(price~.,data=train.sub,distribution="gaussian",interaction.depth = 10,
          n.minobsinnode=10,keep.data = TRUE,n.trees = 5000,shrinkage = 0.005,n.cores=8)
test.pred = predict(boost,test.sub,n.trees = 5000)
test.error = sqrt(mean((test.pred-test.sub$price)^2)); test.error
```

```
## [1] 52.10712
```

```
train.pred = predict(boost,train.sub,n.trees = 5000)
train.error = sqrt(mean((train.pred-train.sub$price)^2)); train.error
```

```
## [1] 45.38729
```

Random Forest

```
library(randomForest)
rf = randomForest(price~.,data=train.sub,ntree=1000)
test.pred = predict(rf,test.sub)
test.error = sqrt(mean((test.pred-test.sub$price)^2)); test.error
```

```
## [1] 53.15101
```

```
train.pred = predict(rf,train.sub)
train.error = sqrt(mean((train.pred-train.sub$price)^2)); train.error
```

```
## [1] 24.84847
```

Do the same transformation on scoring data.

– Code omitted –

Use Gradient Boosting Trees to make prediction on scoring data.

```
library(gbm)
boost=gbm(price~.,data=data.sub,distribution="gaussian",interaction.depth = 10,
          n.minobsinnode=10,keep.data = TRUE,n.trees = 5000,shrinkage = 0.005,n.cores=8)

train.pred = predict(boost,data.sub,n.trees = 5000)
sqrt(mean((train.pred-data.sub$price)^2))
```

```
## [1] 46.20756
```

```
test.pred = predict(boost,scoringData.sub,n.trees = 5000)

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

Result

After more than a month of effort and over 30 submissions, I eventually got RMSE of 51.84845, ranking 8th out of 362 competitors on the public leaderboard (Nov. 27).

Discussion

- I find that data processing is really important. At the very beginning, I ran models on the raw data set, but it turned out to be so bad. Then I tried to manipulate data. When I added more and more useful information to the models, the results got better and better.
- Theoretically, random forest should not overfit. However, maybe the number of trees is not big enough, I observe a significant overfitting under random forest.
- Gradient boosting gives the best result. Ensemble methods have better performance than single predictors.
- I have not come out with an idea to interpret texts such as access and transit. I tried to use Kmeans and NLTK to cluster the texts but it did not help.
- I used Leave-one-out cross validation rather than k-kold cross validation because k-fold is too time comsuming. One run of gradient boosting usually takes approximately 10 minutes. K-fold takes more than twice of that.

Citation

[1] 做了一点微小的工作19260817. Retrieved from https://www.kaggle.com/captainidiot/19260817 (https://www.kaggle.com/captainidiot/19260817) on Nov. 2, 2018.
[2] Package ‘gbm’. Retrieved from https://cran.r-project.org/web/packages/gbm/gbm.pdf (https://cran.r-project.org/web/packages/gbm/gbm.pdf) on Nov. 5, 2018.
[3] The caret Package. Retrieved from http://topepo.github.io/caret/index.html (http://topepo.github.io/caret/index.html) on Nov. 12, 2018.