How much for your Airbnb?

Zhijun Liu (User Name: janeliu)

2018/12/02

1 Introduction

This project is based on a data set consisting of 29142 homes and 96 variables on the Airbnb website. In order to make the prediction as accurate as possible. I tried both linear and non-linear models, single and ensemble models. And finally chose boosting model to do the prediction, which has the lowest RMSE and runs faster than other ensemble models, such as random forests model.

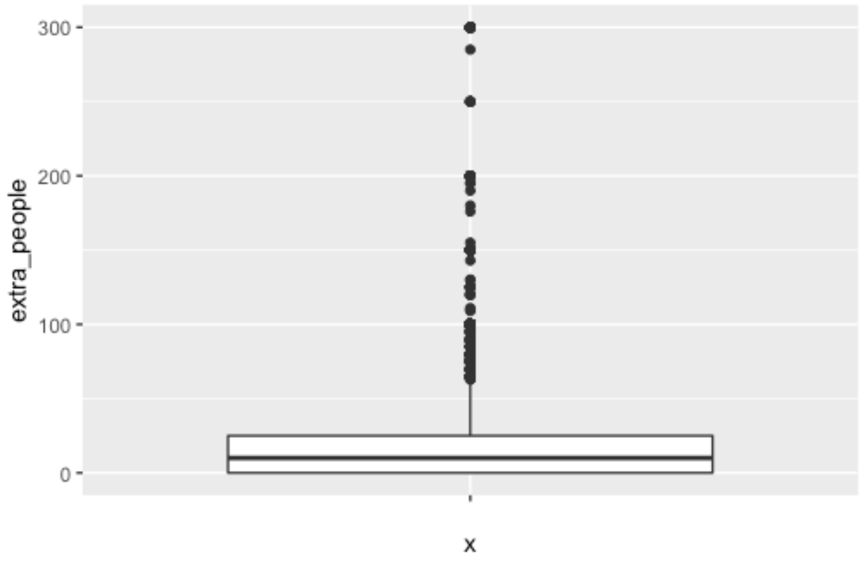
2 Exploring the data

After loading the data, I used str() and summary() to understand the structure of the whole dataset, and found this dataset has following features:

1) This data set has 29142 objects and 96 variables
2) Many unstructured data like text messages that are difficult to utilize
3) Some variables have too many missing values
4) Some variables have outliers that may affect the accuracy of prediction
5) Some categorical variables have too many levels, and using all of them can be meaningless

For example, the boxplot shows the outliers in the variable extra_people.

```
#library(ggplot2)
#ggplot(data_clean_none_text,aes(y=extra_people))+geom_boxplot()
```



3 Preparing the data for analysis

3.1 Feature selection

There are many data that I can hardly use such as urls and text message, so I manually selected all variables as follow that could be useful from the original dataset.

```
data_clean_none_text <-data[,c("id","host_since","host_location","host_respons
e_time","host_response_rate","host_is_superhost","host_neighbourhood","host_li
stings_count","host_total_listings_count","host_verifications","host_identity_
verified","street","neighbourhood","neighbourhood_cleansed","neighbourhood_gro
up_cleansed","city","zipcode","smart_location","latitude","longitude","is_loca
tion_exact","property_type","room_type","accommodates","bathrooms","bedrooms",
"beds","bed_type","amenities","price","cleaning_fee","guests_included","extra_
people","minimum_nights","maximum_nights","calendar_updated","availability_30",
"availability_60","availability_90","availability_365","calendar_last_scraped",
"number_of_reviews","first_review","last_review","review_scores_rating","revie
w_scores_accuracy","review_scores_cleanliness","review_scores_checkin","review
_scores_communication","review_scores_location","review_scores_value","instant
_bookable","is_business_travel_ready","cancellation_policy","require_guest_pro
file_picture","require_guest_phone_verification","calculated_host_listings_cou
nt","reviews_per_month")]
```

3.2 Transform some variables/values

I firstly replaced “N/A” with NA, so that it can be easy for me to deal with the missing values in the later analysis.

```
data_clean_none_text$host_response_time[data_clean_none_text$host_response_tim
e=="N/A"]<- NA
data_clean_none_text$host_response_rate[data_clean_none_text$host_response_rat
e=="N/A"]<- NA
data_clean_none_text$zipcode[data_clean_none_text$zipcode==""]<-NA
```

Then I created new variables to take place of a couple of original variables. For example, I created a new variable “review_range” to calculate how many days between “last_review” and “first_review”. In this case, the dates can be more meaningful for data analysis.

```
data_clean_none_text$review_range <- as.Date(data_clean_none_text$last_review,
format = '%Y-%m-%d') - as.Date(data_clean_none_text$first_review,format = '%Y-%
m-%d')
data_clean_none_text$review_range <- as.numeric(data_clean_none_text$review_ra
nge)
head(data_clean_none_text[,c("last_review","first_review","review_range")])
##   last_review first_review review_range
## 1  2017-08-03   2017-08-02            1
## 2  2018-01-20   2014-08-11         1258
## 3  2017-12-29   2016-10-02          453
## 4  2018-01-01   2017-07-09          176
## 5  2017-12-09   2016-04-12          606
## 6  2017-05-21   2016-04-02          414
```

In order to utilize some unstructured data, I defined a funtion to calculate number of elements in each text message, and turned “amenities” into “num_amenities” and turned “host_verifications” into “num_host_verifications”.

```
library(tidyverse, stringr, htmlwidgets)
# define a function to count the number
number_calculation <- function(vector){
  number = str_count(as.character(vector), "\\,")+1
  for (i in 1:length(vector)){
    if(str_detect(vector[i], "^\\{\\}$")){
      number[i] <- 0
    }
  }
  number
}
# amenities -> number of amenities
data_clean_none_text$num_amenities <- number_calculation(data_clean_none_text
$amenities)
# host_verifications -> number of host verifications
data_clean_none_text$num_host_verifications <- number_calculation(data_clean_n
one_text$host_verifications)
```

3.3 Level reduction

When we use a variable with too many levels and accuracy of the prediction could decrease. And for some tree models, variables with too many levels are not allowed. Consequently, I had to reduce levels of categorical variables that I needed. For example, I grouped some house types which only have 10 or less samples into “Others”.

```
data_clean_none_text$property_type[data_clean_none_text$property_type=="Yurt"]
<-"Other"
```

3.4 Change data type

In order to take the ultimate use of these data, I change some data types.

For host_response_rate, I turned it into character and only keep the number with the help of for loop, because I couldn't directly turn it into character or numeric type.

For the variable zipcode, I also used a for loop to change its data type. Mainly because that I could not directly turn it into numeric and turning zip codes into numeric data can still keep their geographic features.

```r
# host_response_rate
data_clean_none_text$host_response_rate2=c()
for (i in 1:length(data_clean_none_text$host_response_rate)){
  if(is.na(data_clean_none_text$host_response_rate[i])==FALSE){
    data_clean_none_text$host_response_rate2[i] <- as.character(data_clean_non
e_text$host_response_rate[i])
  }
}
data_clean_none_text$host_response_rate<-str_sub(data_clean_none_text$host_res
ponse_rate2,1,-2)
data_clean_none_text$host_response_rate<-as.numeric(data_clean_none_text$host_
response_rate)
# zipcode
data_clean_none_text$zipcode2=c()
for(i in 1:length(data_clean_none_text$zipcode)){
  if (is.na(data_clean_none_text$zipcode[i])==FALSE){
    data_clean_none_text$zipcode2[i]<-as.numeric(as.character(data_clean_none_
text$zipcode[i]))
  } else{data_clean_none_text$zipcode2[i]<-NA}
}
# calendar_updated
data_clean_none_text$calendar_updated<-as.numeric(data_clean_none_text$calenda
r_updated)
```

3.5 Impute missing value

For the numeric variables like cleaning_fee, beds, host_response_rate, calendar_updated and zipcode, I used their means to take place of missing values. And for categorical variables like host_reponse_time, I used the most frequent level to take place of the missing values.

```r
# cleaning_fee
data_clean_none_text$cleaning_fee[is.na(data_clean_none_text$cleaning_fee)]<-m
ean(data_clean_none_text$cleaning_fee,na.rm = TRUE)
# host_response_time
data_clean_none_text$host_response_time[is.na(data_clean_none_text$host_respon
se_time)]<-"within an hour"
```

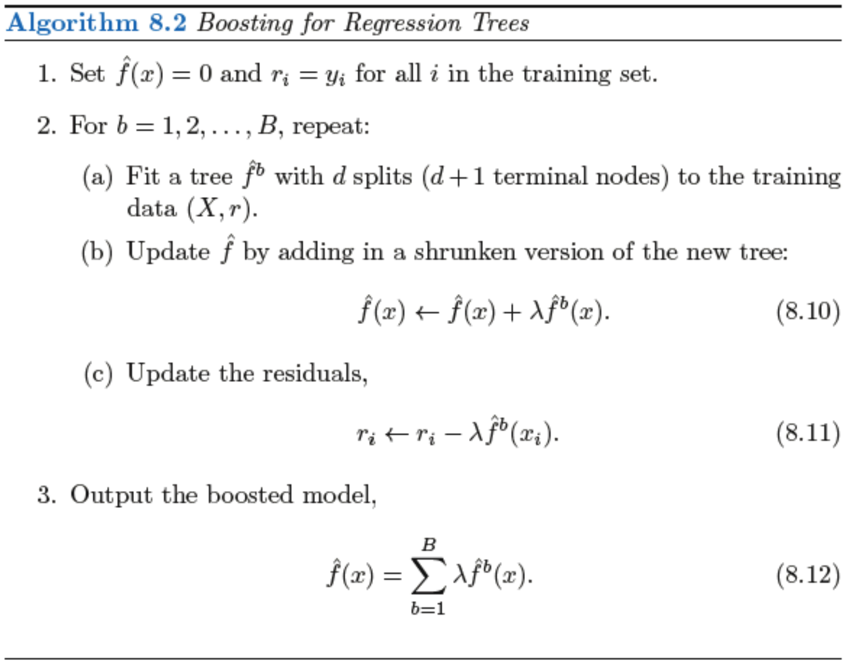
4 Modeling techniques

When constructing models, I chose linear regression with help of lasso and stepwise to select features at the very beginning. Later I added some interaction into the regression model, and the RMSE improved a little bit.

Then I tried tree with 10-fold cross validation, random forest and booting models, and finally chose boosting models which outperforms other methods and has the lowest RMSE.

4.1 Boosting

Boosting is a statistical method based on decision trees for both regression and classification. Different from bagging method, each tree uses information from previously grown trees in boosting model. The algorithm is cited as follow:



Algorithm 8.2 *Boosting for Regression Trees*

```
1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all $i$ in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
    (a) Fit a tree $\hat{f}^b$ with $d$ splits ($d+1$ terminal nodes) to the training data $(X, r)$.
    (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:
            $\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$.    (8.10)
    (c) Update the residuals,
            $r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$.    (8.11)
3. Output the boosted model,
            $\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$.    (8.12)
```

4.2 Feature selection

Firstly, I ran a random forest model and then used function importance() to select the features, but the result was not as expected. So I only kept the most important ones based on the IncNodePurity, and kept adding those variables I thought could be important one by one to see if the RMSE could be lower.

	IncNodePurity
host_response_time	1392870.0
host_response_rate	1532597.3
host_listings_count	1311186.9
host_total_listings_count	1343453.1
neighbourhood_group_cleansed	5969961.8
zipcode2	18763633.9

latitude	12128687.7	availability_365	5027794.7
longitude	19158237.2	number_of_reviews	3155666.0
property_type	4798374.0	review_scores_rating	3387936.2
room_type	46267260.5	review_scores_accuracy	1060236.2
accommodates	23148254.0	review_scores_cleanliness	1491585.7
bathrooms	21809217.9	review_scores_checkin	895799.0
bedrooms	30473077.3	review_scores_communication	718195.2
beds	8940212.4	review_scores_location	1852465.1
cleaning_fee	37145756.0	review_scores_value	1026461.1
guests_included	5268069.0	cancellation_policy	1287538.8
extra_people	3524070.9	calculated_host_listings_count	1516455.5
minimum_nights	2788484.7	reviews_per_month	5715083.3
calendar_updated	4003988.5	host_time	4939957.4
availability_30	3337611.9	review_range	3923333.7
availability_60	3752664.0	num_amenities	5267357.1
availability_90	4238217.4	num_host_verifications	2727956.0

4.3 Adjust parameters

Since this is a regression problem, I set distribution as gaussian distribution.

For the shrinkage parameter, if the shrinkage parameter is too small, more trees will be needed to guarantee the accuracy. So, I only compared 0.01 and 0.001. It turns out that 0.001 works better.

For a boosting method, a large number of trees can help to increase the accuracy to some degree, but too many trees can also lead to overfitting, which is unlike bagging and random forests. After trying 100000, 120000 and 150000 trees, I chose to set n.tree as 150000. When the number of trees equals 150000, the rmse improved not too much, so I stopped trying other larger numbers.

Besides I set the number of splits in each tree as 4, after trying both 3 and 4.

4.4 My boosting model

Consequently, my model is as follow:

```
# boost = gbm(price~ accommodates + cleaning_fee + neighbourhood_group_cleanse
d + room_type + bathrooms + longitude + bedrooms  + availability_60 + review_s
cores_location + latitude + num_amenities + reviews_per_month + minimum_nights
 + beds + review_scores_cleanliness + review_scores_value + review_scores_rati
ng + guests_included + review_scores_checkin + host_is_superhost  + num_host_v
erifications + review_range + host_time + extra_people + is_business_travel_re
ady + calculated_host_listings_count +  review_scores_communication + host_ide
ntity_verified + cancellation_policy + zipcode2 , data=data_clean_none_text, d
istribution = "gaussian",n.trees = 150000,interaction.depth = 4,shrinkage = 0.
001)
# predBoost = predict(boost,newdata=scoringData,n.trees = 150000)
```

5 Results

5.1 Linear regression

At the very beginning, I used stepwise method and lasso method to select features and construct the models, and the best result was around 64 (public score), which indicates that the relationships between price and independent variables are not totally linear. Then, I added some interaction into the model, and the result improved a little bit, but it was still over 60 (public score).

5.2 Tree models

Single decision tree worked badly in this situation, but tree with 10-fold cross validation and random forests models could lower the RMSE to approximately 55-56 (public score), but the boosting method could finally lower the RMSE to 52.89949 (public score) and 54.69546 (private score).

6 Discussion

6.1 Conclusion

After a month work, I have learnt a lot from this competition:

1) In this case, none linear models work better.
2) Ensemble methods, such as bagging and boosting, can do more accurate predictions compared with single models.
3) Though random forests can eliminate the problem of overfitting, it works much slower.
4) For a boosting model, a large number of trees can increase its accuracy but may also leads to overfitting, but it runs faster than random forests model.

6.2 Further analysis

In the future, I would like to try more different splits in each tree to figure out which one is better in this situation. Also, I would like to read more materials to figure out if there are any more efficient ways to do the feature selection for a boosting model.

7 Citations and reference

[1] James, G., Witten, D., Hastie, T., and Tibshirani, R. (2017). An Introduction to Statistical Learning with Applications in R. New York, NY: Springer.

[2] Wickham & Grolemund. R for Data Science (O'Reilly Media, Inc., 2016)