36-462 Final Report

Minyue Fan, Wanhe Zhao

May 4, 2019

1 Introduction

Ever since AirBnB was launched in 2008, it has taken the hospitality service industry by storm. Our dataset is based on AirBnB's property listings in Seattle with 5201 entries containing more than 20 parameters include both categorical, numerical variables and geographic locations. Our datasets have large more samples than total number of variables, and therefore is a n>p dataset. In this project, we are interested in:
1. Build a regressor to predict the price for each listing, so it can be used to help set price for new listings.
2. Build a classification algorithms to predict the review, being either undesirable or desirable for marketing purposes.

2 Exploration

2.1 Data Cleaning

We consider a listing with price zero or price around $5000 as invalid data. We also remove a listing with $price 999, which only offers pull out sofas and has abnormal host_listings_count 1276. We therefore removed four such data points from the dataset.

2.2 Univariate Visualization

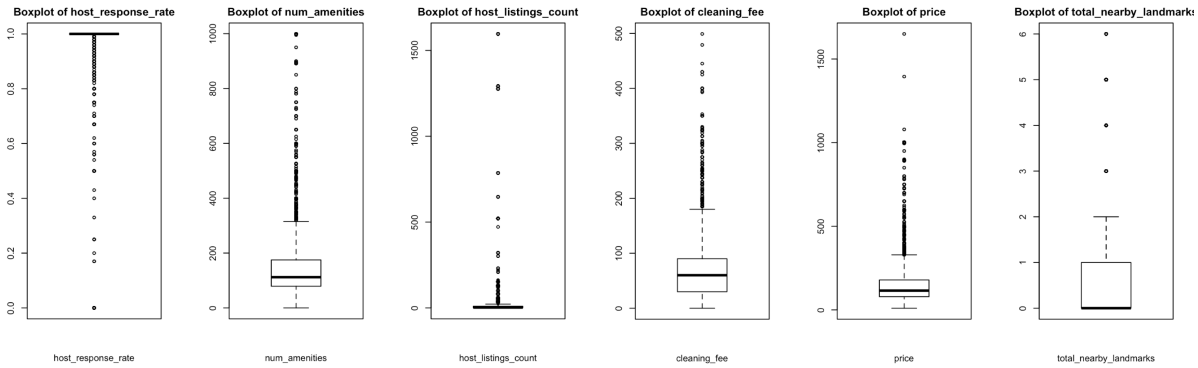

Figure 1: neighbourhood_group_cleansed and cancellation_policy vs. price boxplots

From the univariate boxplots, we find that most hosts provide about 100 kinds of amenities, while there are also host who offer more than that up to 1000. Most listing has no landmark within 1 mile. Most host has a small number of listing while some can go up to 1500. Cleaning fee and Price share a similar trend. Cleaning fee centered at around $80 and can go up to $500. Price is centered around $300 and can go up to larger than $1500. Cleaning fee might be an important predictor for price.

2.3 EDA on Price and Review datasets with Response

Neighbourhood_group_cleansed has more than 10 levels of neighborhoods. We will select some significant high and low price neighborhood. From the boxplots above, whether a listing in Downtown, Delridge, or Lake City, the price is different. Therefore, we encode Neighbourhood_group_cleansed to three factors as: In_Downtown", In_Delridge", In_Lake_City".
For cancellation_policy, we find that if a listing has super_strict_60 cancellation policy, then the price is relatively high and there is no outliers in this case. We did not re-categorized this variable given it only has six levels

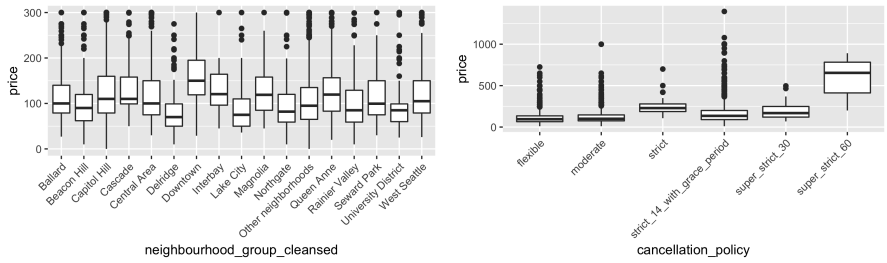

Figure 2: neighbourhood_group_cleansed and cancellation_policy vs. price boxplots

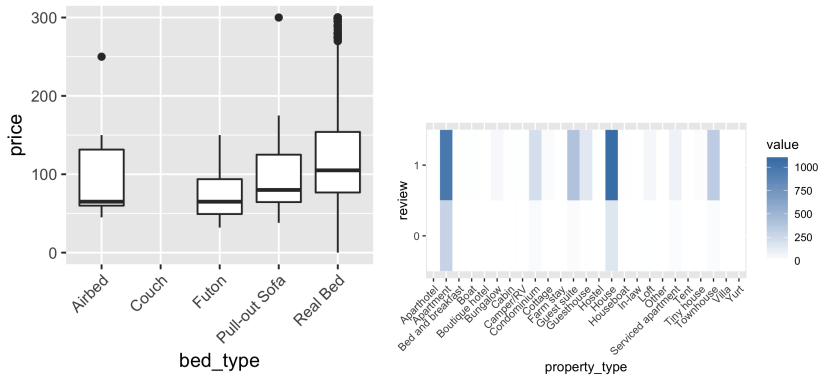

Figure 3: property type vs. review heatmap and bed_type vs.price boxplot

For bed_type, couch only appears once in training set, and this type of bed does not appear in our validation set. We change the bed_type for this particular listing to Real_Bed, considering that this listing has 4 beds in total and we assume that there is more than couch provided in the place with a price of 600$.
For property_type has more than 10 levels both in review and price. Only "Apartment", "Condominium", "Guesthouse", "House", "Townhouse", "Guest suite" are dominant categories, so we encode all other property types to "other". We did the same for price dataset.
We also exclude Maximum_night. Given the fact that maximum night has default value for hosts to choose when list their places. Therefore, this variable is not informative enough for our prediction models.

2.3.1 Unsupervised visualization

We could see that the numerical variables of data is linearly separable when projected to first and second principal axes.

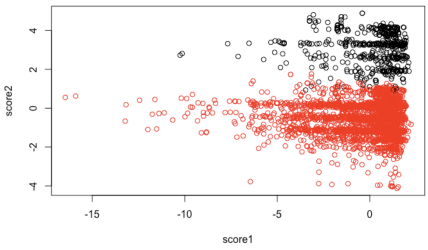

Figure 4: numerical variables in review with 57% total proportion of variance explained

3 Supervised analysis

3.0.1 Feature engineer

We change the geographic information, latitude-longitude, to number of nearby landmarks. We selected major popular landmarks in seatle: Museum of Pop Culture, Pike Place Market, Olympic Sculpture Park, Seattle Great Wheel, Seattle Art Museum and Kerry Park. Then, discretize a listing distance to the popular landmark, by thresholding it to 1 mile. Total nearby popular landmarks that are less than 1 mile is added as total_nearby_landmarks.We also change detailed text descriptions of amenities to total number of amenities in a listing, as num_amenities.

3.1 Model selections

Our datasets have mixed features of categorical and numerical data. Some models are not suitable for our data:
LDA, QDA and Gaussian Mixture Model assumes that variables to be multivairate normal distribution. This means our categorical variables are treated as predictors in a numerical way, which contains ordering info.
Naive bayes assumes the independence between variables. However, we our model suffers from multicollinearity, like price and cleaning fee. Naive bayes performs well on categorical features, but has strong assumption on numerical data. We have to normalized our numerical data to be normal distribution.
Also our datasets are relatively small compared to the natural datasets(geographical data, speech, sound or images), so **neural network** may not be useful.

3.2 Supervised analysis on predicting price

3.2.1 Baseline linear regression model

We chose linear regression to be our baseline model for predicting price(model 1). Linear regression initially has training mse 14288 and validation MSE 14574. MSE is huge in average, there could be a 200 dollar difference between predicted price and real price given most listings have price that fall between 50 and 200 dollars. We therefore proceeded to try different other models to decrease MSE.
We try to tune parameter, alpha, which add regularization with lasso, elastic net and ridge. Lasso Regression (model 2) that helps with varibale selection, gives a slightly better result: 14272 training MSE and 14567 validation MSE. From this slight improvement, we gain some insight into the underlying data that we may need to do variable selection on our datasets.

```
cv.glmnet(y=train$price, x=x_train, alpha = seq(0,1,by = 0.1)[i],family="gaussian")
```

3.2.2 random forest

Random forest works well on high dimensional data with mixed of categorical and numerical data. We first fit a random forest model (model 3) on all variables and had as training mse and as validation MSE. This is a good improvement and we wonder if we can do better with random forest. We called the importance function on the full model (model 3) and ranked each variables by their importance.

Cleaning fee is no doubt an important part of price. Since most people will make decisions based party

Table 1: Importance of Each Variable from a Full Random Forrest Model Top 5

Variable	Importance
host_listings_count	13779.6286367
cleaning_fee	5877.0395847
accommodates	4424.0278479
bedrooms	4276.5535172
total_nearby_landmarks	3170.9207826

size, it is also important how many people a host can accommodate. As we have shown in the exploration section, price in downtown has a much larger mean than other areas, it also makes sense for location to have large importance as in total_nearby_landmarks.

We then started from using the most important variable, host_listings _count, and keep adding variables according to the ranking to the random forest model to see if there will be a decrease in MSE. Our model 4 is a random forest only on host _listings _count. Model 5 is a random forest on host _listings _count and cleaning fee. We keep adding less important variables into the model until model 12, where we apply random forest on variables bedrooms, cleaning fee, accommodates, neighbourhood group cleansed, room type, bathrooms, total nearby landmarks, host listing count, cancellation policy, property type.

Table 2: Train and Validation MSE of Each Model

Model	Train MSE	Validation MSE	Model	Train MSE	Validation MSE
Model 1	14288	14574	model 7	3637.767	4740.841
Model 2	14272	14567	model 8	2288.821	4539.659
Model 3	9137.980	9519.055	model 9	2137.255	4320.539
Model 4	4407.255	5683.436	model 10	1964.689	4274.825
model 5	3542.333	5083.568	model 11	1693.940	4222.403
model 6	3568.773	4945.955	model 12	1611.018	4079.575

At this point, we are almost certain that adding more variables of importance into the model will decrease the MSE for both training and validation. Our final model contains continuous variables host listings count,cleaning fee, accommodates,bedrooms, total nearby landmarks, beds,number of amenities, guests included, bathrooms,categorical variables property type,room type,cancellation policy,the indicator variable host identity verified and the indicator variable In Downtown. This random forest model has 500 trees and number of variables tried at each split of 2. This model can explain 55.6% of the variance in training data and 49.87% in validation data.

Our next step is to tune the model with different parameters. We tried different values for number of variables randomly sampled as candidates at each split(mtry), number of trees to grow(ntree) and minimum node size of terminal nodes(nodesize).

From table 3 and table 4, we observe that when fixing mtry = 4 and varying ntree and nodesize, increasing nodesize increases MSE while increasing ntree decreases error rate. However, even though ntree = 500 always has a smaller training error, its validation error is much larger. It indicates that ntree = 500 overfits to training data. We therefore choose ntree = 400 to proceed to tune mtry. We also observe that with increasing nodesize, the MSE increases for both training and validation set. Because minimum nodesize restricts trees to be truncated at certain depth, it is likely that a tree with nodesize = 10 will be way too shallow to learn all the useful information and therefore is a bad fit.

Table 3: Model 12 with mtry = 4: Training Error

nodesize/ntree	ntree = 400	ntree = 500
2	1164.991	1097.594
4	1451.826	1418.813
6	1754.538	1742.118
8	2018.931	2014.189
10	2261.764	2243.728

Table 4: Model 12 with mtry = 4: Validation Error

nodesize/ntree	ntree = 400	ntree = 500
2	5703.669	6467.398
4	5820.621	6367.107
6	5796.359	6407.088
8	5806.210	6424.531
10	5823.071	6427.760

Table 5: Model 12 with ntree = 400 nodesize = 2

mtry	Training Error	Validation Error
2	1676.028	7286.296
3	1226.148	7248.164
4	1081.496	7287.982
5	1021.091	7272.601

Table 6: Model 12 with ntree = 400 nodesize = 4

mtry	Training Error	Validation Error
2	2113.044	6018.669
3	1595.269	5883.611
4	1459.095	5940.168
5	1372.490	5951.704

We observe that in general, training and validation error both decreases as mtry increases. Even though a nodesize = 4 tree has higher training error, its validation error is much lower than that with nodesize = 2. Therefore, we have to come to a conclusion to use a tree with ntree = 400, nodesize = 4 and mtry 5 as our random forest model while using host listings count,cleaning fee, accommodates,bedrooms, total nearby landmarks, guests included, bathrooms,property type,room type,cancellation policy,beds,num amenities, host identity verified as predictors.
Below is our final model for predicting price:

```
imp.variables <- c("price", "host_listings_count","cleaning_fee", "accommodates","bedrooms", "total_nea
train <- train %>% select(imp.variables)
rf_oob_comp <- randomForest(formula = price~., data = train,ntree = 400, nodesize = 4,  mtry = 4)
```

3.3 Supervised analysis on predicting review

This is a classification problem, and we have an imbalanced data set with base ratio 85.6%. In other words, most reviews have label 1.

3.3.1 baseline Logistic regression - linear mode)

Table 7: confusion matrix of lasso linear model(baseline)

	Reference	
prediction	0	1
0	155	88
1	425	3373
Accuracy : 0.8731	Sensitivity : 0.26724	Specificity : 0.97457

Our model is bad at predicting listings when review is 0. Therefore we want a model that adds weight on our class label 0 to improve the sensitivity. Even though logistic regression has good interpret ability, this may not separate our high dimensional data well, so we will try some non-linear methods.

3.3.2 Random forest on review

Here are some reasons we decide to use xgboost at the end:
•We tried to tune on the parameters, but there are not much available compared to xgboost. For exmaple, We use weight on class, but still rf do bad in class 0, and no apparent improvement with adjusting the

threshold. However, xgboost has much more parameters that deal with imbalance dataset
•Our data has lots categorical variables with different number of levels. As said in [3], "random forests are biased in favor of those attributes with more levels. Therefore, the variable importance scores from random forest are not quite reliable

3.3.3 xgboost

We choose Xgboost, a gradient boosting method. For the following reasons:
•As said in [3], "Boosting focuses step by step on difficult examples that gives a nice strategy to deal with unbalanced datasets by strengthening the impact of the positive class." This is shown in later analysis that xgboost did better in improving sensitivity•it iterative constructs on ensemble method of weak learners to ensure our model has high bias and low variance to avoid overfitting.
•Since our dataset has a balanced amount of both numerical and categorical data. To take into account both kinds of data, we choose boosting method takes in a sparse matrix (where we turn categorical variables to dummy variables).
•There is a scale_pos_weight tuning parameter to deal with imbalance class, so that we could increase sensitivity. So this classification problem now works on n=4041 and p=45 (originally 22) problem.

We initially tried out to play around some parameters, using xgb.train with validation set as watchlist. Even though the validation misclassification rate could be reach to 0.11 sometimes, the performance are quite unstable, and we suffer from overfitting. Boosting algorithm is easy to overfit with a noisy dataset, since at each sequence, it focuses on learning the mistake of the one before. We decide to reinspect our dataset and take out some outliers.

Then, we carry out a first round of grid search with all the variables. The performance is not good.

Variable selection There are noise variables in datasets. Theoretically, these do not give maximum gain split at all, so they would never be selected and do not influence our tree growth.
However, this is only perfectly correct for near infinite datasets.[1] In reality, to make our model more generalized, we need to have less dimensions in our data, that is eliminate the weak variables that may correlating with the response variable review by chance.

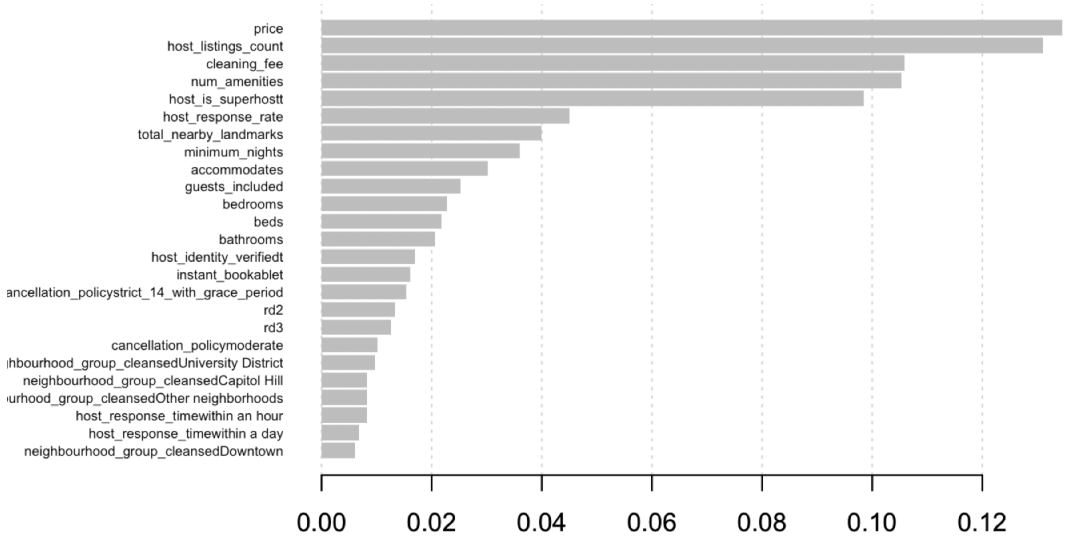

Figure 5: variable importance(top 25 displayed) with random variable introduced

We use our best tuned xgboost model with all the variables and plot the variable importance as in Fig.5. We added two random variable as a reference to help us distinguish the line between noise and useful variables. rd2 is a categorical rd with 3 levels, and rd3 is a random uniform distribution variable. We will select all the variables that above rd2.

Hyperparameter search
Now with selected variables, we rerun our grid search again to tune on new dataset. We follow the step from [2], and here we detailed our motivations in selecting each parameter, some idea credit to [2]:
•Step 0, we start a baseline xgboost model with stratified = TRUE and scale_pos_weight = 1 to take care of imbalance set.
•Step 1, I start from tuning max_depth and min_child_weight. We found min_child_weight =1 gives the best, which will take care of our imbalanced set leaf nodes that have smaller size groups. The performance for max_depth with 5 and 8 are almost the same. Since our model have already suffer from overfitting at the baseline model, so we pick a smaller depth value.
•Step 2, we tune Gamma, which is the loss reduction required to make a further partition. Since we didn't see an obvious differences between 0 to 0.5, we select Gamma 0 for making our model more conservative.
• Step 3: tune subsample[seq(0.6, 1, by=0.1)] and col_sample_bytree[seq(0.6, 1, by=0.1)]. This is for each time of boosting, how many n and p we want our model to learn. subsample=0.5 and colsample_bytree=0.5 gives the best validation error and AUC with smaller standard deviation.
•step 5: tune scale_pos_weight with c(1,2,3,4,5), and weight=1 gives the best.
•step 6: Last will be the eta(0.03, 0.08, 0.1, 0.3), where we tune the learning rate to avoid our overfitting. We end up eta=0.1, which is the proper rate that we could get low train and validation miscalssification rate.
`Below is our final xgboost model:`

```
xgb.cv(data =  dtrain_select, nrounds = 100,
    nfold = 5, showsd = TRUE,  metrics = list("error", "auc"),
    verbose = 0, objective = "binary:logistic", "max.depth" = 5, "min_child_weight" = 1,
    gamma = 0, save_period=1,
    "eta" = 0.1,  "scale_pos_weight" =  1,
    "subsample" = 0.5, "colsample_bytree" = 0.5,
    stratified = TRUE,early_stopping_rounds = 30,
    print_every_n = 1,  booster = "gbtree")
```

Final model explanation Most of the variables with high variable importance in our final model, are about host(etc. host_listings_count, host_is_superhost), price and the quality of the place. This makes sense by convention. When a traveler write review after staying at an airbnb place, one may evaluates more in terms of living conditions, whether he or she has a nice interaction with the host. People may forget which neighborhood the listing is located at, cancellation policy or whether it is instant bookable, factors are considered when selecting and booking a place.

4 Analysis of results

We have very different variables chosen for predicting price and review. For example, for price, location and accommodates are important. However, the most important factors for reviews are mostly related to host.

4.1 Analysis of results on predicting price

We performed a 5-hold cross-validation on data from price.csv and our model has validation MSE 3657.57. On average, each listing is predicted with a $20 error. This is a much better result than what we started with (14574 validation error with linear model). From the validation set, my model does poorly on listing

with high price larger than $150. This might because they are all good listings at great locations, but the owners offer different prices for different levels of lavish interior design.

Future Work Due to computational limitation, we did not explore models other than linear, lasso and random forest. If given time, we would like to explore more models. We would explore different ways to decode different variables. In the future, we might take all amenities into account as binary variables instead of counting them and see if that makes a difference. For example, a "pets allowed" policy is uncommon and will be preferred by family travelling with pets. In the future, we can also try different numbers for distance when determining nearby landmarks.

4.2 Analysis of results on predicting review

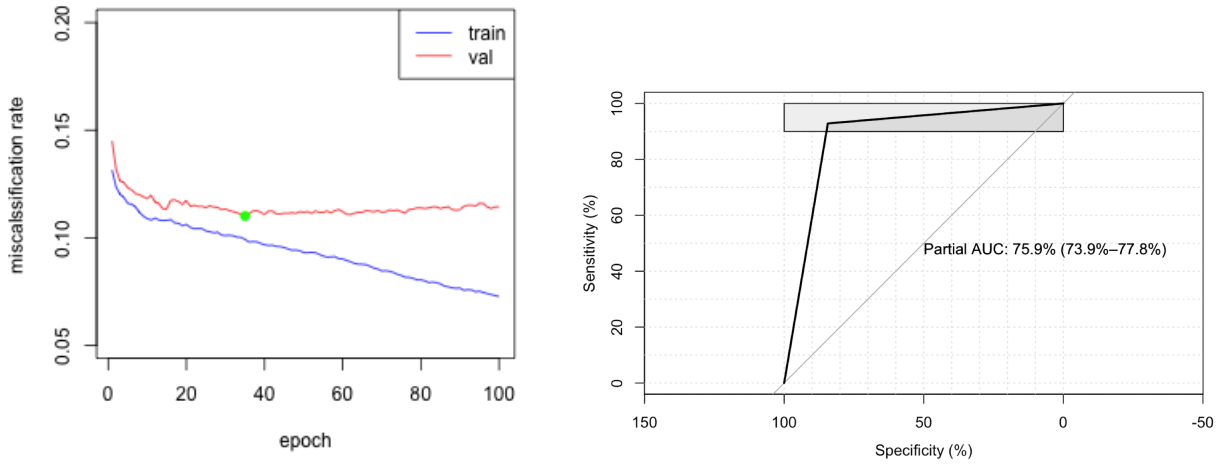

Figure 6: xgboost train and validation set performance) with random variable introduced, final model ROC curve(right)

The misclassification rate plot has much more stable lines than before, but still shows a sign of overfitting starting from round 36.

Table 8: confusion matrix of lasso linear model(baseline)

	Reference	
prediction	0	1
0	318	59
1	262	3402
Accuracy : 0.9206	Sensitivity : 0.54828	Specificity : 0.98295

From the confusion matrix and roc curve above, we could see that sensitivity improves from baseline 0.26724 to 0.548. This xgboost algorithm still does bad in predicting in the case of review=0. This may be that our xgboost only have 580 listings to learn from, compare to 3461 listings of review =1.

Future Work One last thing to note that host_listing_counts ranks pretty high in variable importance in both price and review models. We need to go back to reinspect the dataset, since there is no direct relationship between number of a host listings and price or review.
In the future, we could further tune our model or try out other models, so that we could prevent overfitting and add more penalty when model makes type 2 error. Or upsampling, get more listings with reveiw is 0, and train on balanced set.

References

[1]https://datascience.stackexchange.com/questions/17364/gradient-boosting-tree-the-more-variable-the-better
https://insightr.wordpress.com/2018/05/17/tuning-xgboost-in-r-part-i/
https://medium.com/@aravanshad/gradient-boosting-versus-random-forest-cfa3fa8f0d80