Duke University

STA
671

Assignment: Kaggle Competition Project

Authors:
Tiancheng Pan

December 8, 2021

1.Introduction

This report is a machine learning project looking to perform classification task predicting Airbnb Availability based on features such as number of bedrooms, number of bathrooms. I choose use the average accuracy of cross-validation for the train data set to compare the performance of each algorithm and parameters. All the code can be found as my github in the reference.[1]

2.Exploratory Analysis

For the original data set we have, there are 22 variables, where *Decision* is our Target, and *id* is just used as the index for each observation. Thus, there is no reason to include *id* in our Matrix **X**, and we have 20 parameters in total. Notice for both train and test data set, I all have the missing value. Thus, for train data set, we simply delete the columns with missing data since we have enough data and the missing is not a lot. However, for the test data, to finish all the prediction in Kaggle requirement, I replace the *na* by the most common variable in that column.

For 20 parameters we have, 10 of them are defined as 'Object' in python, which can be consider as the categorical variables. However, after I explored more of the data, the *Neighbourhood* is actually the categorical variable even it collect as numerical. And the *Price* can be transfer to the numerical variable by delete the dollar sign. Thus, finally I have 10 Non-numerical variable, and within them, 4 of them are Boolean variable, which are *Host_is_superhost*, *Host_has_profile_pic*, *Host_identity_verified*, and *Instant_bookable* . I change them from "T/F" to "1/0" to let it more readable by algorithm. And for the other 6 categorical variable. I create the dummy variable for each categories. Notice when creating dummy variable, it is possible that train data and test data has different categories. My approach here is combining the variable part of train and text first, creating the dummy variable for such big matrix, and delete the variable that shows the value for all the observation in train data set. Since, if it do not provide any information during the training of the model, it is useless to make prediction on such variable, even thought it may provide some difference in the observations of test.

In the end, I have the train data set with 5866 observations and 101 parameters. Notice even thought we have lots of variable, we also have relatively larger number of observations. Moreover, using PCA may let the interpretation to be harder since the parameters after PCA is much messy. Thus, I do not choose using PCA here. I also did the standardization of the train data set, since it contain lots of dummy variable and some algorithm like SVM or KNN do affect by the relative variance for each variables. A histogram for the number of observation with different categories can be found in the Python Notebook.

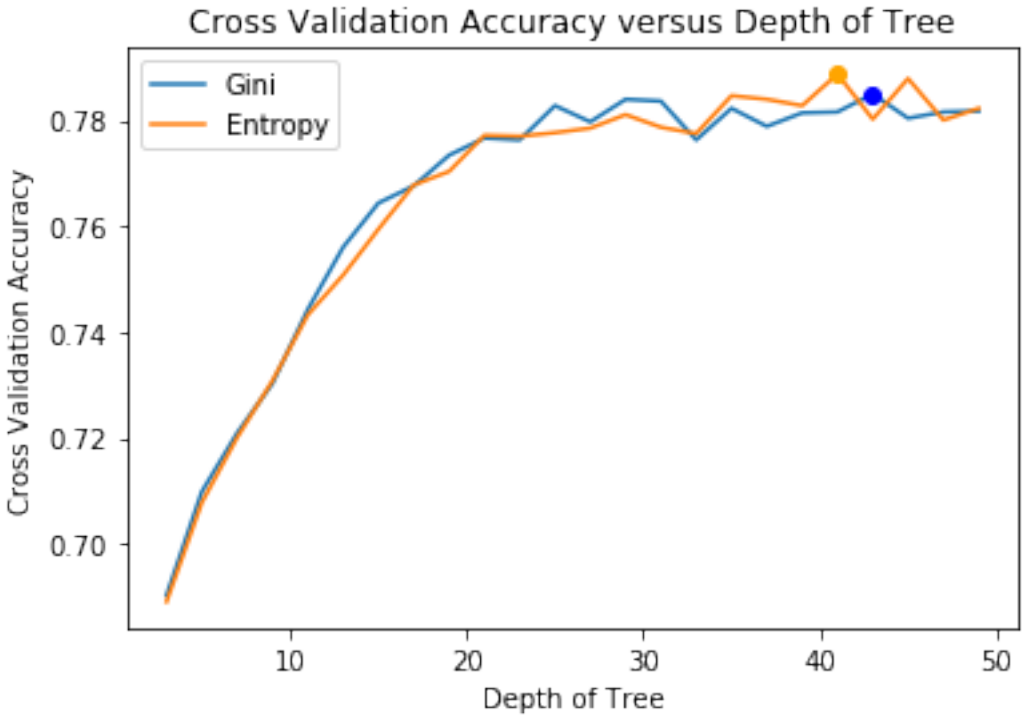
3. Method

Random Forest

The first Algorithm I choose is the random forest, since it is able to handle higher dimensionality with relative ease and is easy to interpret[2]. Additionally, random forest is generally a very powerful method able to learn non-linearity well. The specific method I used to implement random forest was *sklearn.ensemble.RandomForestClassifier*.

The *sklearn.ensemble* module provide a traditional version of random forest except one difference. it averages the probabilistic predictions of individual trees instead of the vote result of each tree. It use bagging, where it only use a subset of observations and variables to fit an individual tree based on either the Gini or Entropy criterion to determine node splits. The decision tree fitting itself is done by *DecisionTreeClassifier*, using the CART algorithm.

The hyper-parameter I focused is the maximum depth and the criterion, i.e. either the Gini or Entropy, of each tree. To compare the parameter, as I mentioned before I used the accuracy of cross validation. I set the range of maximum depth from 3 to 50 and the cross validation of the trees are shown as below



The accuracy keeps increasing from 3 to 25 and start the fluctuation after that. For this attempts, the highest cross validation accuracy get from Gini forest is 0.785 at depth 43 and the highest CV accuracy get from entropy forest is 0.789 at depth 41. The best model may change dual to the different split we have during cross validation and the final model, and after I tried several time, I think it is reasonable to choose any depths after 30 just based on CV accuracy. However, including too much variable may cause over-fitting and redundant for the model. Thus, finally, I choose Gini forest with depth 33 as my final forest model. The model build in 0.446 seconds. The prediction time it takes for test data set is 0.0400

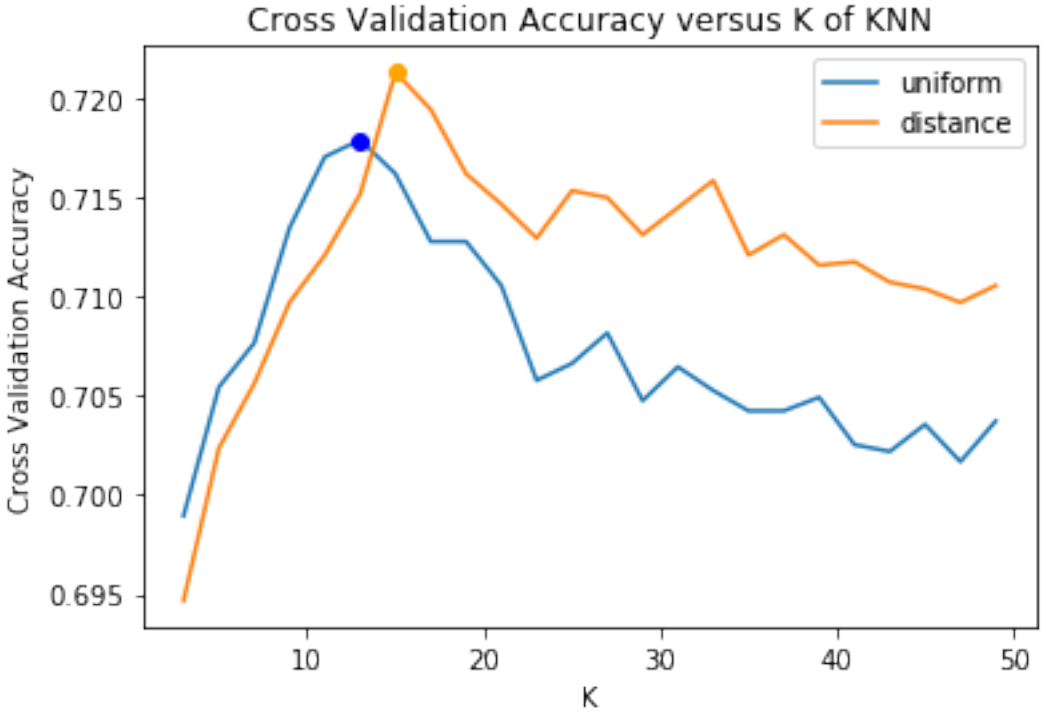
K-Nearest Neighborhood

The second algorithm I choose is KNN. There are lots of advantages of KNN algorithm. First, it is instance based learning, which means it does not learn anything in the training period. It stores the training data set and learns from in only at the time of making real time predictions[3]. Thus, it makes it much faster than other algorithms. Therefore, I want to compare the accuracy of it to other algorithms like random forest I mention before and SVM I will mention. I want

to illustrate whether the time I used during the training is worth to achieve a higher accuracy. Moreover, KNN algorithm is very easy to implement. It only has two parameter, the value of K and the distance function. I used the Euclidean distance here since most of the 101 parameters I have are dummy variable which can be consider as weighted equally after standardization. Thus, the Euclidean distance can represent the difference of each observation.

I used *sklearn.neighbors.KNeighborsClassifier* to run the KNN algorithm, which the prediction is given as the average of K nearest Neighborhood of each observations. Moreover, the weighted of each Neighborhood can be consider as equal or as the inversely proportion to the distance of the prediction point.

The hyper-parameter of KNN that I can focus on is K and the weighted approach after I constrain the distance function as Euclidean distance. I set the range of K from 3 to 50 and the cross validation of the KNN are shown as below:



The accuracy keeps increasing from 3 to 13 and immediately starts to decrease after that. For this attempt, The highest CV accuracy gets from uniform KNN is 0.718 at k= 13 The highest CV accuracy gets from Weight KNN is 0.721 at k= 15 The accuracy result may change dual to the different split we have during cross validation and the final model, however the best model is stuck on this too. Thus from the cross validation, the best performance model from Euclidean KNN is the Weighted KNN with K 15. And when I run the model with whole train data set. It only take 0.004850 second to "build" the model which is much faster than random forest. And the prediction time of the test data set it takes is 0.0356 which is almost same as the random forest algorithm.

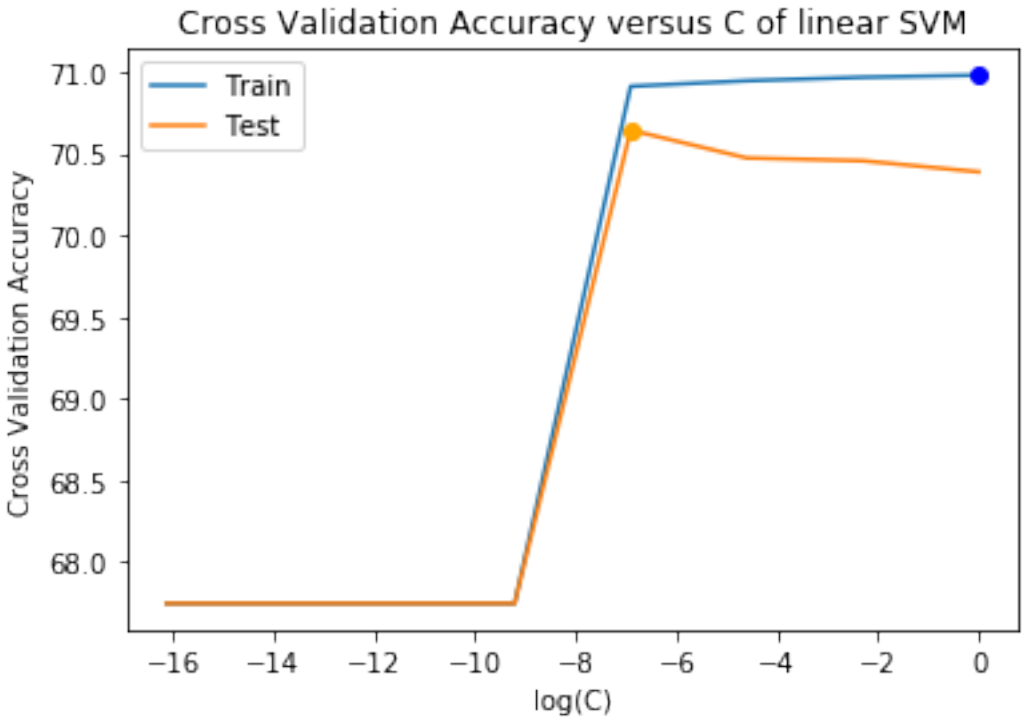
SVM

The final algorithm I tried to use is SVM. There is lot of reason why I choose SVM. It can

be used for the data that is not regularly distributed and have unknown distribution such as our example. With the help of Kernel, the algorithm can be used to high dimensional data. And the last but no the least, outliers have less in influence in SVM algorithm therefore there are less chance of skewing the result as outliers affect the mean of the data with the penalty strategy SVM have.

I use *libsvm* to achieve the SVM algorithm. Since there are lots of kernel type. I choose to use the linear SVM with several different C, quadratic SVM, and radical kernel SVM with several different σ^2 to select the best SVM algorithm. I also write the cross validation by myself since the *libsvm* is not an function from *sklearn*.

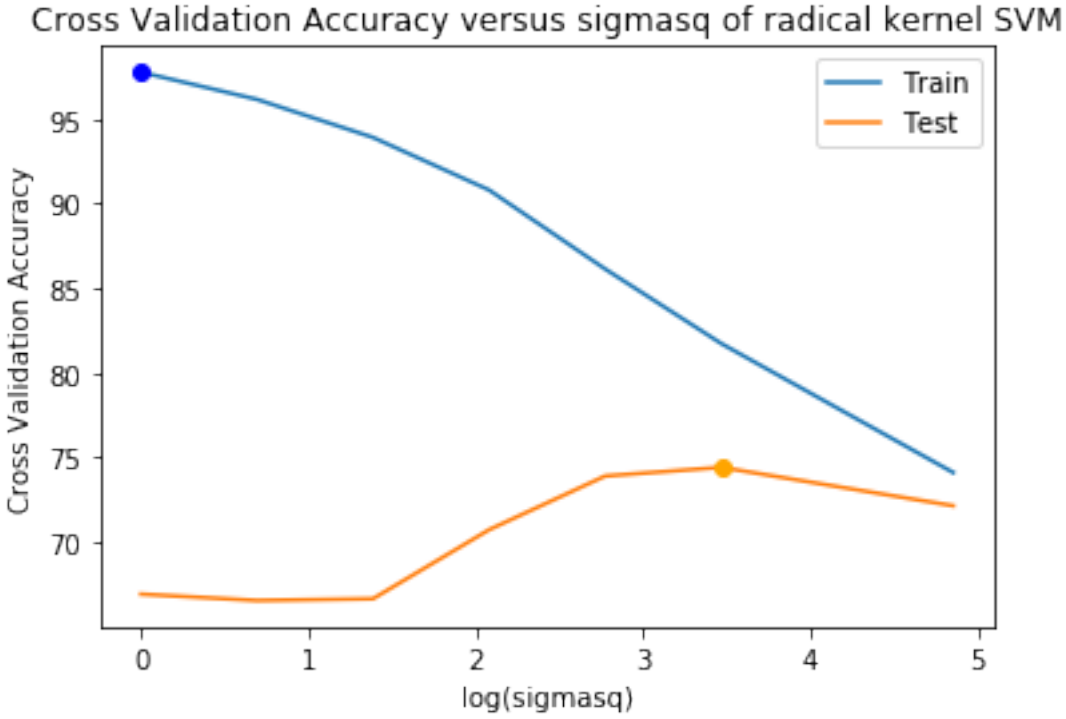
First, for the linear SVM, the strange thing is if I choose C over than 1, it will take much longer time to run the model. I have to a model with $C = 10$, and after one hour it still did not provide the result. Thus, to make the hyper-parameter selection I choose the focus on C less than or equal to 1, which are 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1} and 1. The result is below for both train and test data:



Obviously, highest CV accuracy get from linear SVM is 0.706 at $C = 0.001$ and after than even the CV accuracy for train set is still increasing the CV accuracy for test set start to decrease which indicate the over-fitting.

For the quadratic SVM, since there is not two much parameter I can choose, thus I run the quadratic SVM directly and get the CV accuracy as 0.720

For the radical kernel svm, I choose to focus on the σ^2 and range of the σ^2 I choose is $[1, 2, 4, 8, 16, 32, 128]$ and the result are shown as below:



The accuracy for the test set of cross validation increase at first and achieve the highest when $\sigma^2 = 32$ with the test accuracy 0.745, and start to decrease. Thus comparing to 3 SVM, the radical kernel SVM has the highest test accuracy from cross validation with $\sigma^2 = 32$ and it become my final model of SVM. To run the model with whole data set it takes 7.32 seconds, which takes the most of the time for all of my 3 algorithm.

4. Prediction

For all 3 algorithm I choose, random forest achieve the highest CV accuracy score about 0.78, and the highest Kaggle test score with 30 percent data, which is 0.31. Moreover, even I choose to standardize the data, it do not guarantee the model will perform better. However, Random forest is the only algorithm that do not affect by whether I standardize the data. Therefore, I choose Random forest as my final model. The importance of first five and last five variables are shown as below. Notice for variable that consider my my forest, the Price was really important and the *property_types* seems not that importance[4]. And, even thought I choose the maximum depth of each tree, since it is forest, it actually conclude all 101 parameters in my metrics.

	Variable	Importance
10	Price	0.138561
11	Number_of_reviews	0.129382
12	Review_scores_rating	0.094241
3	Accommodates	0.047325
7	Cooking	0.042827
...	...	...
95	Bathrooms_text_7 baths	0.000062
29	Property_type_Casa particular	0.000060
60	Property_type_Private room in hostel	0.000026
28	Property_type_Campsite	0.000019
62	Property_type_Private room in hut	0.000007

5. Future Improvement

After I finish this project, I do curious about some problem during my whole process. First, for the Forest algorithm, it can treat categorical parameters directly as one node. So, when I change the categorical parameters to dummy variable, whether the accuracy may be changed. I have tried the random forest on the categorical train data directly, but it is too hard to find the match depth to my dummy forest. Secondly, for the linear SVM, why an over 1 C parameter takes such longer time to run the whole algorithm. I will dig more on these two problem.

References

T. Pan, "Airbnb availability," 2021.

R. B. Man Singh, Soumya Kar Choudhury and O. S. U. Andres Manniste, "Airbnb new york city: Demystifying the superhost program,"

N. Kumar, "Advantages and disadvantages of knn algorithm in machine learning," 2019.

F. Revert, "Interpreting random forest and other black box models like xgboost," 2018.