Kaggle InClass Competition Report - Airbnb Prediction Track

Rajas Pandey
Kaggle username - Rajas Pandey

December 11, 2021

I have done the Kaggle Prediction challenge on the Airbnb dataset.

My predictions achieved a Misclassification error of 0.28005 on the Leaderboard evaluation dataset.

1 Exploratory Analysis

The first step after looking at the data file was to understand the columns and what they mean. I have attached my code with detailed comments for this analysis in the appendix. To start with, I tried to understand each feature and it's datatype by browsing through it on Excel. This gives me a sense of what sort of values are there in each column, what each column means and I can start to form hypothesis about the importance of various features for the classification problem of interest.

Around 63% of the training data has a positive Decision value.

1.1 NA analysis

I observed some columns had missing values, so my first thought was to understand what's going on there. The main question I wanted to answer was whether the missing values were missing at random (possibly due to a data pipeline issue) or were they missing structurally. To observe that, I plotted the a correlation heatmap of just the null values 1.

From here, I observed that most of the host identification attributes are missing when `Host_is_superhost` is missing. This means that all the 3 identifying attributes are linked and must be imputed together. As they are all 't'/'f' values, I impute them with False.

The feature with most number of attributes missing is Host_response_time 2. So I check how the Decision values are distributed in the missing data. The missing values have a significantly higher distribution of Decision 0 compared to the dataset. Thus, these values are critically important and should not be dropped. I hypothesize that the missing values in

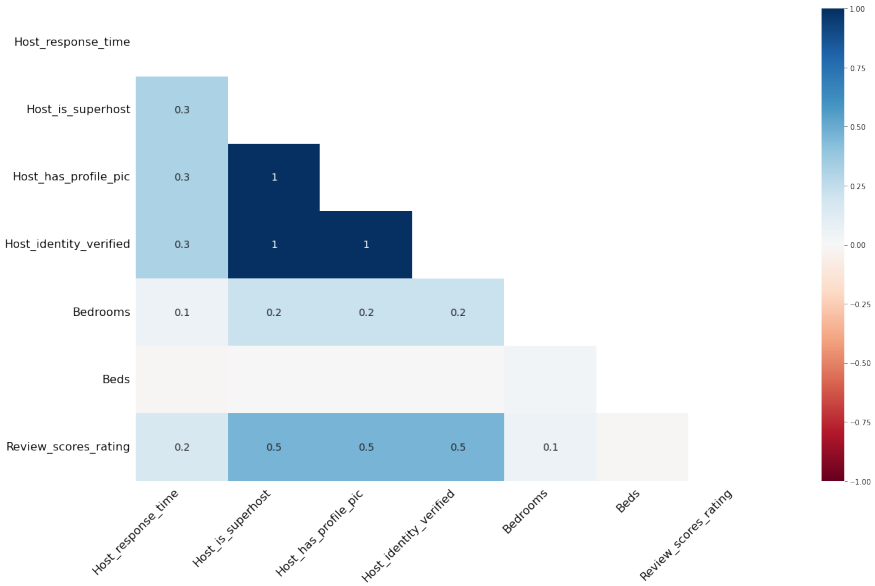


Figure 1: Which NA values occur together? Example, all missing values of `Host_has_profile_pic` occur when `Host_is_superhost` is also missing

this attribute are because of new host (at least a third of these hosts don't have their identities verified either, from 1). So the correct imputation will be to assign 0 to the missing values.

All this analysis used PandasMcKinney (2010) and MatplotlibHunter (2007) libraries. Similar analysis is also done for other features, which is present in the appendix.

1.2 Data Prerpocessing

The next logical step after NA analysis is to deal with the NA values. There are also some categorical features, which aren't readily read by ML models.

1.2.1 NA value imputation

`Host_response_time` - Fill NA with blank
The rationale is that when encoding the categorical variable, it will automatically assign a category to the blank value as well.

`Host_is_superhost` - Fill NA with false
`Host_ has_profile_pic` - Fill NA with false

Figure 2: What Decision values do rows with missing Host_response_time have?

`Host_has_identity_verified` - Fill NA with false
The rationale is that most of these values may be missing due to new host. So in that context, the NAs should ideally represent false.

`Bedrooms` - Fill NA with 0
`Beds` - Fill NA with 0
A similar rationale to identity attributes is hypothesised here. New houses or houses with 0 Beds or Bedrooms will have NA in these places. So imputed with 0 values there.

1.2.2 Encoding

Two types of encodings are used - LabelEncoding and One-Hot Encoding.

Label encoding is used when there is an ordinal relation between the categorical attributes. For example, in the `Host_response_time` feature, there is an inherent ordering in the time the host takes to respond. So that and all the other binary (including true false) variables are Label encoded. Label encoder assigns unique numeric values to each category and preserves ordinal relationship.

One hot encoding is used when there is no ordinal relationship between the categories. One hot encoding creates a new binary column for each category. This encoding has been used for categorical attributes like Neighbourhood, Month and Room_type where there is no ranking between the categories.

A challenge while using one-hot encoding is that there might be multi-collinearity between the new created features. This is avoided by dropping the first category. This implies that 0 values for all the other created attributes implies that it must be the first category.

1.2.3 Feature engineering

`Bathroom_text` feature had a string of values. Some of them had numeric values and string descriptors. An example of such values is "1.5 shared bathrooms" or "Private half-bath". I created 2 new columns - `Num_baths` and `Shared_bath` and split the information from the Bathroom Text column. The numerical part, including text numbers like "half" was stored as a float in the `Num_baths` column. `Shared_bath` is a binary column indicating whether it is a shared bath or not. This was done to better incorporate the information in a machine ready format

The Price column was also processed to use the float value of the price.

2 Methods

2.1 Models

I used 3 basic classifiers and a Stacked classifier for my final predictions.

2.1.1 Support Vector Machines

I used SVM because it is usually a good model for high-dimensional spaces. Though the computation costs are higher comparatively, the small size of the training data ensured that it was not too long. In the case of binary classification, there is the separating hyper plane intuition of SVM which is convenient to visualise and understand.

Further, I used the SVM with RBF kernel. RBF kernels are easy to tune and have a small number of Hyperparameters. Only Hyperparameters in RBF kernels are C and gamma, which I tuned using cross validation.

In addition, SVMs were recently covered in class and homeworks so I was confident of debugging with them.

2.1.2 Random Forest

Random Forests are based on decision trees and pick certain features randomly while creating a tree. Finally, it takes the vote of all the tree classifiers to determine the final class. As they are based on decision trees, Random Forests are really fast to train. They have several parameters like `max_features`, `min_samples_split` and `n_estimators` which can be tuned.

Usually, Random Forests tend to do well because of the randomisation at each level of the tree creation and the eventual voting. Ed Tam mentioned in his Discussion class that Random Forests are usually the second best algorithm for every classification task after the specific algorithm for that task.

2.1.3 Logistic Regression

Logistic Regression is again a useful algorithm for binary classification. It is also very easy to tune Logistic Regression hyper-parameters - there is just the regularisation penalty. I use l2 regularisation which has 1 hyper-parameter C and I tune it using cross validation.

Logistic regression also has a nice intuition for binary classification, in terms of the sigmoid value being closer to 0 or 1 (the value of the function being positive or negative).

2.1.4 Stacking

Finally, I got the best results for Stacking classifier. Stacking runs the estimators (logistic, random forest and svm) independently and then trains another classifier (in my case, a logistic regression) on the output of the estimators to predict the correct outcome.

This is similar to voting in random forest type algorithms. My goal was to reduce the bias in the algorithms, and Stacking is a way to combine the good parts of different algorithms.Unlike voting, in stacking, the final estimator learns the importance of each estimators output on the overall prediction, improving the quality of the result.

2.2 Training

2.2.1 SVM

I decided to use the RBF kernel for the reasons mentioned above. I was training on the Kaggle notebook using the CPU.

Before training SVM, it is very important for the data to be in the same scale. I used a StandardScaler which uses mean and variance to scale each observation. This is part of a pipeline, which executes every time SVM model is trained.

I ran a GridSerach over C and gamma parameters in the logspace (-4,4) based in Numpy Harris et al. (2020) with a step of 5. There were total 25 values of C and gamma which the algorithm iterated over. The algorithm I used was scikit-learn Pedregosa et al. (2011) SVC which is based on libsvm.

For each value, I used a 10-fold cross validation as a GridSearch parameter. I also leverage the inbuilt paralleization in gridsearch to use the max number of concurrent workers available. This significantly speeds up the training process for SVM which is usually long.

2.2.2 Random Forest

Here too I use gridsearch to estimate the hyperparameters using cross validation. The data is split up into 10 cross validation folds and the class weights are set to 'balanced_weights'. This setting of class weights is useful when there is imbalance among classes as is the case in our dataset.

Gridsearch in turn uses the scoring rule accuracy as we want to maximise the accuracy of our predictions.

2.2.3 Logistic Regression

LogisticRegressionCV is already implemented in sklearn and I used it directly. I used the 'l2' penalty and 'liblinear' solver which is considered to be good[1] for binary classification when there are cases of 1 vs all.

Here too, I run it over 10 cross validation folds. The only hyperparameter here is the C or regularization parameter. The best value for C is observed to be 10.

[1] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegressionCV.html

2.2.4 Stacking

While Stacking, I use the algorithms with their best parameters determined by cross validation. The final estimator in the scikit-learn Stacking implementation is Logistic Regression, which is used as is out of the box. Stacking classifier trains the fastest because there is no cross-validation used here.

The following table shows the training time comparisons of the various classifiers I have used. Logistic regression is the fastest to train, followed by Stacking which uses just the best cross validated parameters.

	Average training time (s)
Stacking	19.57
SVM	569.41
Random Forest	131.18
Logistic Regression	4.29

3 shows that Random Forest overfits on the training data and has a very high training accuracy.

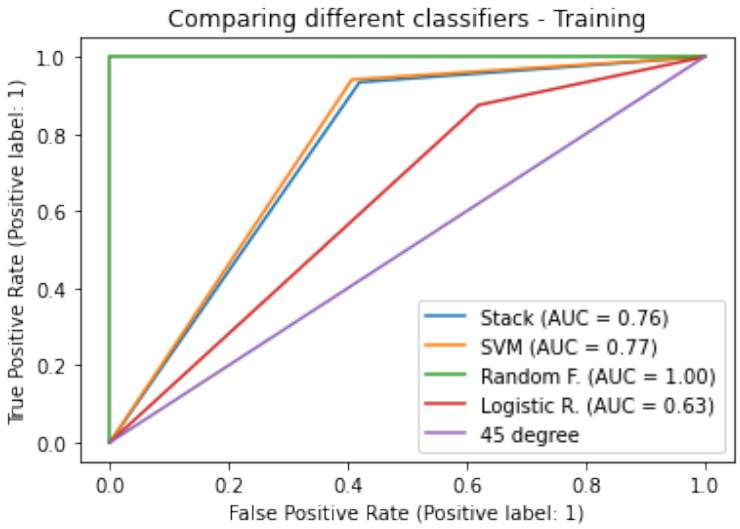


Figure 3: Comparing classifier training ROC curves

2.3 Hyper-parameter Selection

2.3.1 SVM

The two hyper-parameters under consideration were C and gamma in the RBF kernel. 4 shows the accuracy changing with C and Gamma. The best parameters were found to be C = 100 and gamma = 0.01.

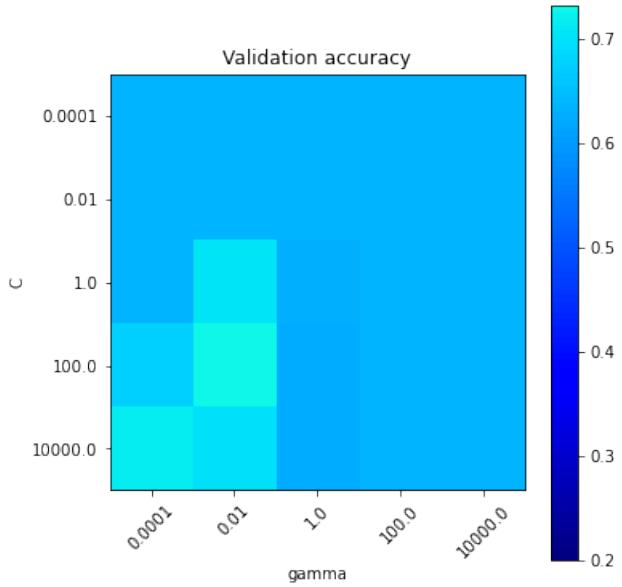


Figure 4: Validation set accuracy with C and Gamma

C is the penalty associated with the regularisation term and controls the error. Higher C usually implies more error. Gamma is a parameter of the RBF kernel and is used to control the curvature of the decision boundary. Lower gamma boundaries are more curved, while higher gamma boundaries are more spread[2].

2.3.2 Random Forest

The parameters used are n_estimators and max_features. We iterate over values of n_estimators from 100 to 500. This parameter represents the number of decision trees to be constructed during the training.

More trees often leads to more accuracy. However, the computation time also significantly increases with more trees. The best value was found to be 400 trees.

max_features parameter represents how many features should be considered in each iteration of the tree. I have iterated over 2 values - None and 'sqrt'. None means no restrictions on the number, it can theoretically consider all the features that it finds best. 'sqrt' parameter consideres only the square root of the number of features available at any time. Further implementation is in the appendix.

2.3.3 Logistic Regression

Only the C hyperparameter which is tuned by LogisticRegressionCV function on its own. It represents penalty and is similar to the C in SVM.

[2] https://medium.com/@myselfaman12345/c-and-gamma-in-svm-e6cee48626be

2.3.4 Stacking

No hyperparameters tuned here. Only the best parameters from all the previous algorithms are used to train this model.

3 Results - Prediction

The objective of the prediction task was to get the lowest Mean Absolute Error. In my case, that was achieved in the Stacking classifier using Logistic regression, SVM and Random Forest stacked together.

The following table shows the performance of these classifiers on the validation set -

	Mean absolute error
Stacking	0.2341
SVM	0.2428
Random Forest	0.1940
Logistic Regression	0.2933

We can further compare the classifiers using their ROC curves and AUC. This is presented in 5.

There is a quirky observation that although Random Forest seems to perform better on the validation set, it fails to do as well on the actual unseen data on Kaggle. This may be because training and validation sets are from the same distribution, so there are higher chances of Random forest learning and sort of overfitting on the training data.

I further investigate the observations on which the algorithms perform well. In particular, I look at the confusion matrices 6 and observe that Random Forest is predicting 0 labels much better than all the other algorithms. Perhaps this is why it was giving higher accuracy.

3.1 Fixing Mistakes

- Dropping features I couldn't engineer - There are features like `Property_type` and `Review_score_rating` which I have dropped from the analysis. Given more time, I would like to work on them further and try to include them in the analysis.
- `Property_type` had many property types and some of them seemed to be very similar to each other. I tried to study these property types in greater detail, combine the similar ones together and categorize them for the analysis. But together in train and test, there were just too many types and including them was leading to sparse data. So I decided to drop them for this analysis.
- `Review_score_rating` - This feature had missing values and I did not want to impute them with mean or 0 values. So I built a regression model which predicted the review

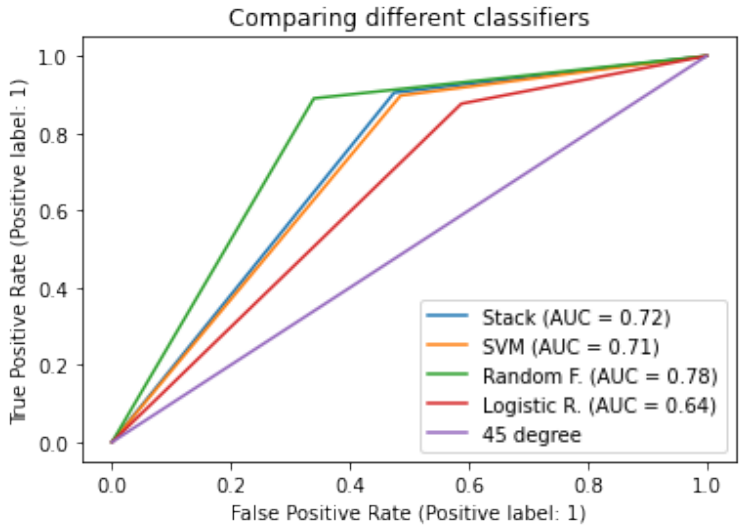
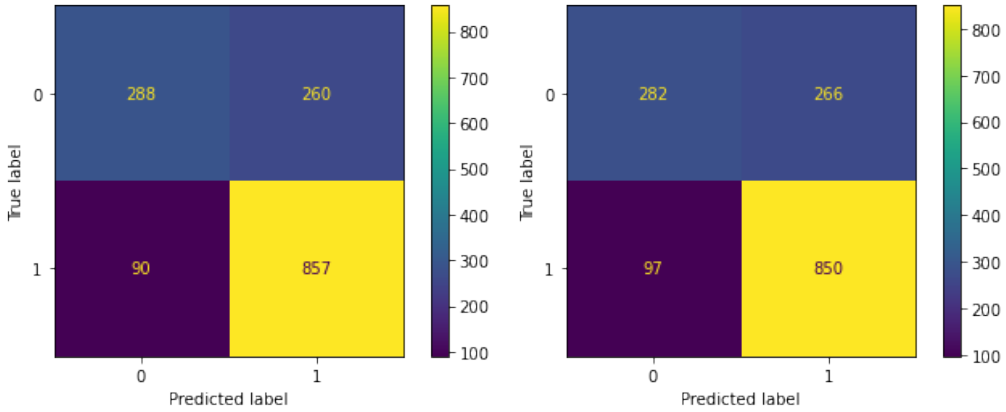
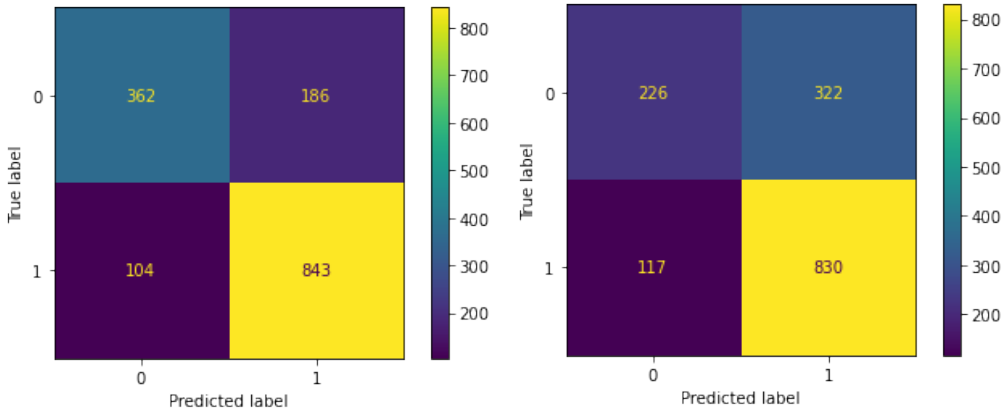


Figure 5: Comparison of ROC of different classifiers on the Validation set



Stacked classifier SVM classifier



Random Forest Classifier Logistic Regression Classifier

Figure 6: Confusion Matrices of different classifiers

scores based on other attributes. However, this did not particularly improve results while also taking up lot of computation resources. Intuitively, this meant that a house with similar facilites should have similar reviews, but this might not be true in practice. I decided to drop the feature for this analysis.

Citations

References

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90.

McKinney, W. (2010). Data structures for statistical computing in python. In van der Walt, S. and Millman, J., editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Code and Appendix

```
Put your code here.
```