NEW YORK UNIVERSITY

DS-GA-1003 MACHINE LEARNING

MAY 2017

Airbnb Price Prediction

Author:
Kailing WANG
Luyu JIN
Siyuan XIANG

Supervisor:
Kurt MILLER

1 Motivation and Problem

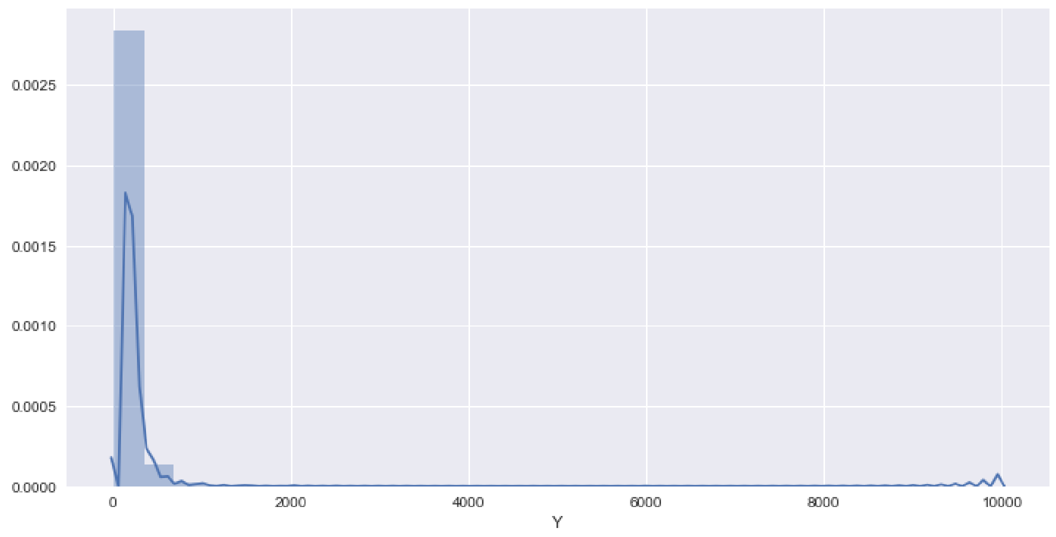
Airbnb is an online platform for people to lease or rent lodging. In this project, we predicted the optimal Airbnb price in New York City considering features such as room types, numbers of beds, cancellation policies and amenities. We plan to build a price prediction tool for Airbnb and its hosts and reach the ultimate goal that both the hosts and guests are satisfactory with the listed prices. This price prediction tool will provide extremely useful guidance for new hosts who have no idea of how to price the lodging they provided. With better predicted price, we hope that the Airbnb hosts can provide profitable accommodations, while the guests can afford the accommodations expenses.

2 Dataset Description and Analysis

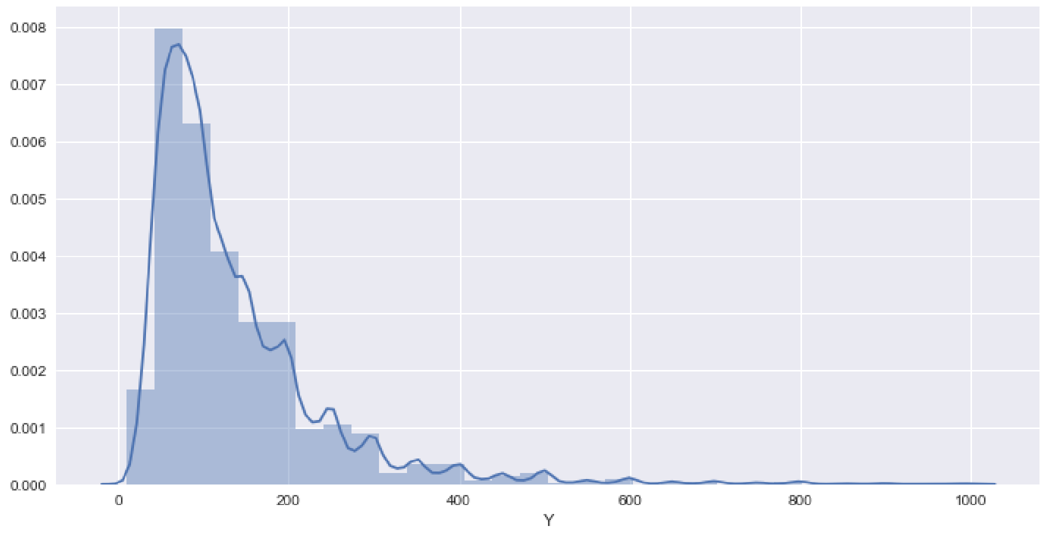
The Airbnb dataset in New York City contains 40, 227 records and 95 attributes in total. The whole set of features mainly include basic housing descriptions provided by the hosts, characteristics of the houses (e.g. number of beds, number of bedrooms, type of the house, location of the house), and the scores (e.g. cleanliness, location, description accuracy, communication) rated by users.

Among all, there are 36 features with more than 1, 000 missing values. Some features like `weekly_price` and `monthly_price` have more than 10, 000 missing values. We think that the occurrence of these missing values are normal because most hosts may only provide daily prices for the accommodation. In addition, the features relevant to rating scores such as `review_scores_rating`, `review_scores_cleanliness`, and `review_scores_accuracy` have around 10, 000 missing values, which implies that some housing information have been posted but still not be selected. Besides, we discovered some inconsistent entries, such as a house with no bedrooms or having an abnormally high price ($9, 999).

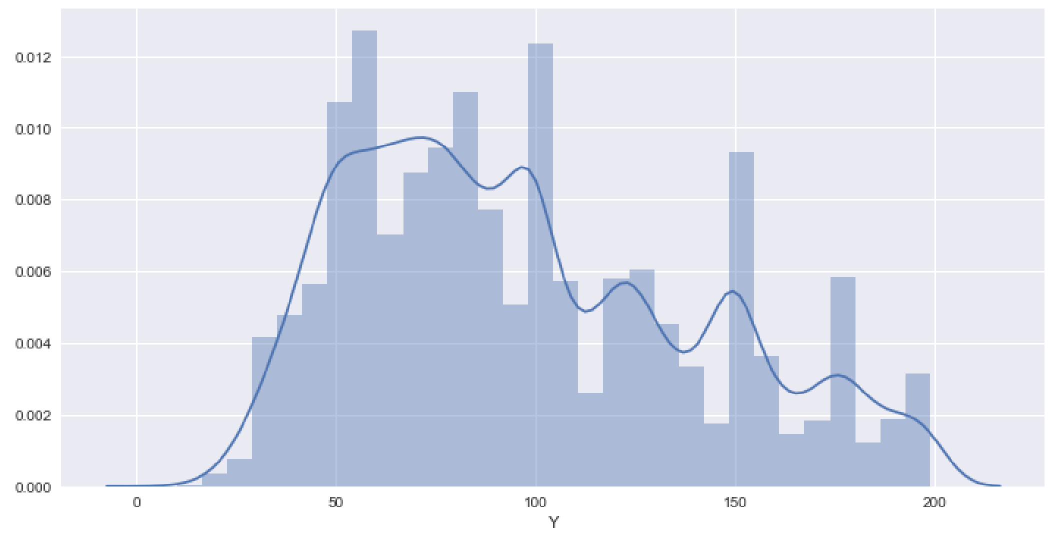
We used `price` as the target variable and generated three distributions.



The above plot shows the distribution of all the prices. We can see that almost all the prices are less than \$1, 000. Thus, we plotted the distribution of prices from \$0 to \$1, 000 as shown below. From this plot, we discovered that most prices are below \$200.



Finally, we ended up with the following plot with prices less than \$200.



In short, we concluded that most prices lies between \$0 and \$200. Besides, we assumed that the lodgings with high prices have distinct features, which our models can't easily capture. Therefore, when training our models, we focused on those with prices below \$200.

3 Feature Preprocessing

Among all the features with less than 1, 000 missing values, we first selected the following 10 attributes from the raw dataset with almost no missing values: `property_type`, `room_type`, `accommodates`, `bathrooms`, `bedrooms`, `beds`, `amenities`, `price`, `minimum_nights`, and `cancellation_policy`. Besides, we think that location information is important. Hence, among all the features about location, we selected `zipcode` which has least number of missing values.

Next, we preprocessed these attributes:

- Create dummies for the categorical variables (i.e. `property_type`, `room_type` and `cancellation_policy`)
- Delete invalid zip codes
- Convert the price format (e.g. convert a string `$1,200.00` to a float of 1200.0)
- Drop missing values in all columns except for `review_scores_rating`
- Drop inconsistent entries (i.e. `accommodates`, `bedrooms`, `beds`, or `price` with value of 0)

- Create feature vectors for `amenities`

4 Performance and Evaluation

We split the dataset into three parts: training data, validation data and testing data in proportion 0.56:0.14:0.3.

Mean Squared Error (MSE) was primarily used to train and tune the hyper-parameters of our models because it is convex and has second-order partial derivatives, the Hessian matrix, everywhere in its domain. Such properties are extremely important for our final model XGBoost because the optimization algorithm of XGBoost is based on these properties.

To compare the performance of different models, we used Median Absolute Error to measure the deviation in predicted price because of its robustness. Also, we compared the time performance of some selected models.

5 Model Selection

5.1 Baseline

We trained two baseline models with the following selected features: `room_type`, `accommodates`, `bathrooms`, `'bedrooms'`, and `'beds'`. These five features are used in the filters on the Airbnb's website, so we believe that they are the most essential features without further feature selection.

1. Our first baseline model is Ordinary Least Squares (OLS) model because it is the most basic regression model and trained fast.
2. The second baseline model is K-Nearest Neighbors (k-NN), which is basically finding several most similar listings and then using the average price of them as the recommended price.

5.2 Linear Models

We started with regularized linear models as well as Bayesian Ridge.

1. Ridge:
   Ridge regression is linear regression with ℓ_2 regularization.

2. Lasso:
   Lasso regression is linear regression with ℓ_1 regularization.

3. Elastic Net:
   Elastic net is linear regression with combined ℓ_1 and ℓ_2 regularization.

4. Bayesian Ridge:
   The prior for the parameter w has the following form:

   $$p(w|\lambda) = \mathcal{N}(w|\lambda^{-1}\boldsymbol{I_p})$$

   The output y is assumed to be Gaussian distributed around Xw:

   $$p(y|X, w, \alpha) = \mathcal{N}(y|Xw, \alpha)$$

   The advantage of the Bayesian version of Ridge Regression is that we can incorporate the use of a prior, or assumed knowledge about the current state of beliefs, and rationally update them.

Actually these linear models performed very similarly. We think it is because there are far more data instances (over $30,000$) than features (88). In this case, with abundant data, the variance is very low. Thus, regularization introduces bias while not helping much with the variance, that is to say, adding regularization is not effective for these models in terms of bias-variance tradeoff.

5.3 Cluster + Linear Models

Another idea is to take the advantage of unsupervised learning. We first used the k-means method to stratify the input data into clusters, according to the feature `zipcode`. The crux lies in this method is balancing bias and variance. Stratification helps reduce biases. However, for each cluster, we have fewer data to train a linear model, which increases variances. In this case, stratifying data into some clusters helps reduce the loss on test set.

We implemented two types of models using clusters: one is to directly train a model for each cluster and the other is to turn all clusters into dummy variables used as features for prediction.

With each of these two types, we fit the linear models mentioned above with the data in each cluster separately. As for prediction, when given a new data point, we will first decide which cluster it belongs to, then the corresponding linear model will be used to predict its target value.

5.4 Non-linear Models

Since linear models didn't perform well and dummy variables in our cleaned dataset are very likely to be non-linear features, we introduced models to better deal with non-linearity.

1. Kernel Based Models:
   (a) Kernel Ridge:
   Kernel ridge regression combines Ridge Regression with the kernel trick to deal with non-linear features.
   (b) Support Vector Regression (SVR):
   Support Vector Regression is another way to introduce non-linearity. To make the support vector machine works for regression, we need to introduce slack variables in both directions: too large and too small.
   The modified optimization problem is:

$$\begin{aligned} \min \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \\ s.t. \quad & y_i - \langle w, x_i\rangle - b \le \epsilon + \xi_i \\ & \langle w, x_i\rangle + b - y_i \le \epsilon + \xi_i^* \\ & \xi_i, \xi_i^* \ge 0 \end{aligned}$$

2. Tree Based Models:

   (a) Random Forest:
   A random forest is to fit a number of decision trees on various sub-samples of the dataset and use averaging to reduce variance and control over-fitting.

   (b) Gradient Boosted Regression Trees (GBRT):
   Gradient Boosting is another way of ensemble methods. By combining small trees additively, it reduces error mainly by reducing bias.

   (c) XGBoost:
   XGBoost is an implementation of gradient boosting designed for speed and performance that is dominative competitive machine learning. It also introduces features such as column sampling, GPU support and distributed computing.

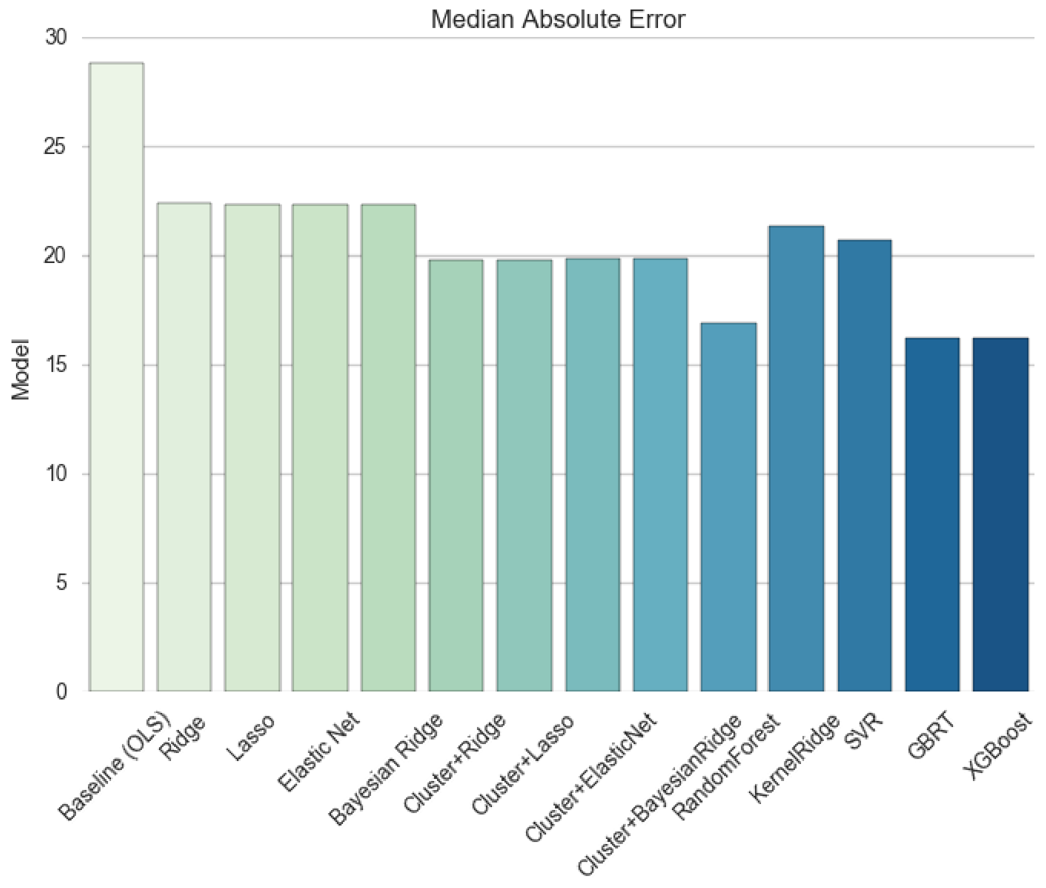
5.5 Hyper-parameter Tuning

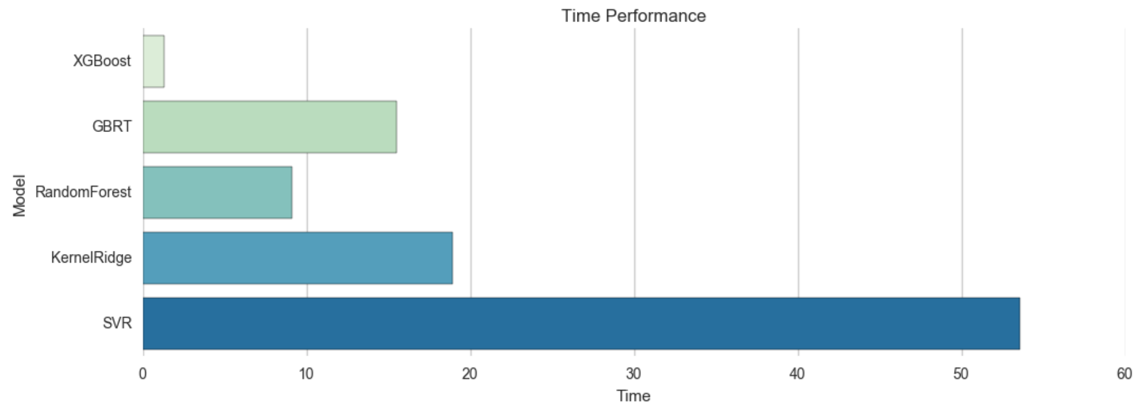
We tuned the hyper-parameters of our models by five-fold cross validation with Median Absolute Error as the evaluation metric. The following table presents the best combinations of hyper-parameters for each model we used. In this table, some cells is filled with 'N/A' because when we tuned the parameters of these model, we utilized scikit-learn's built-in cross-validation function for Ridge and Lasso Regression (RidgeCV, LassoCV), which returned a different best combinations of parameters for each iteration.

Model	Hyper-parameter
Ridge	N/A
Lasso	N/A
Elastic Net	alpha=0.001, l1-ratio=0.8
Bayesian Ridge	alpha1=10, alpha2=1e-7, lambda1=1e-11, lambda2=10
Cluster + Ridge	N/A
Cluster + Lasso	N/A
Cluster + Elastic Net	alpha=0.01, l1-ratio=0.7, n_clusters=4
Cluster + Bayesian Ridge	alpha1=1e-12, alpha2=10, lambda1=10, lambda2=1e-13, n_clusters=4
ClusterDum + Ridge	N/A
ClusterDum + Lasso	N/A
ClusterDum + Elastic Net	alpha=0.0001, l1-ratio=0.9, n_clusters=5
ClusterDum + Bayesian Ridge	alpha=0.0001, l1-ratio=0.9, n_clusters=5, lambda2=10, n_clusters=5
Random Forest	n_estimators=100, max_depth=13
Kernel Ridge	kernel='rbf', alpha=1, gamma=0.01
SVR	kernel='rbf', C=100, gamma=0.01
GBRT	loss='huber', n_estimators=50, max_depth=9
XGBoost	n_estimators=100, learning_rate=0.1, max_depth=8, colsample_bylevel=0.9, colsample_bytree=0.5

6 Results

Since the two baseline models perform similarly, we only included the OLS in the below plot.





According to the above two figures, XGBoost has the shortest runtime and performs the best. The two kernel based models (Kernel Ridge Regression and SVR) have relatively long computing time, but their performances don’t improve much compared to the linear models. With clusters, all linear models slightly improve.

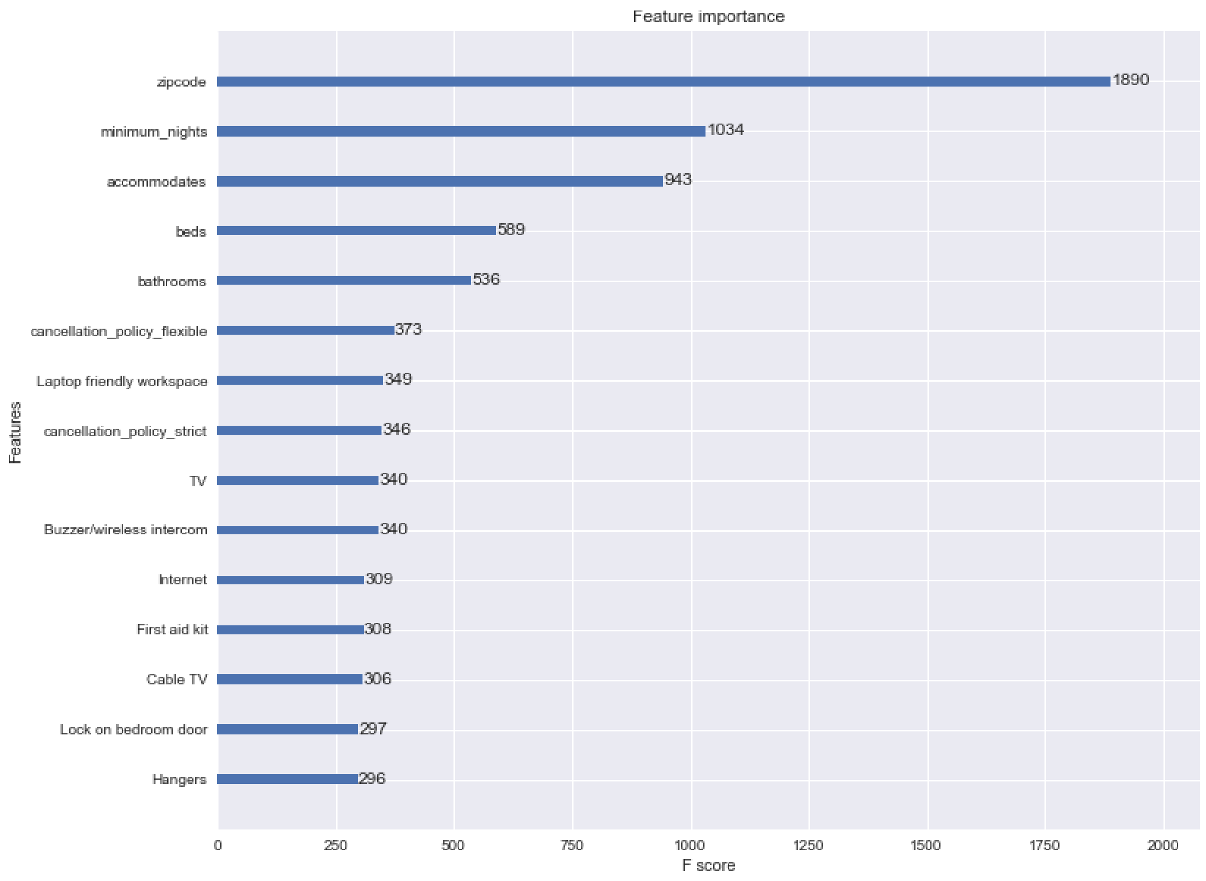
The below table presents the standard error of the median absolute loss associated with each model. It reveals that all the models are stable.

Model	Standard Error
Ridge	0.1237
Lasso	0.1200
Elastic Net	0.1017
Bayesian Ridge	0.1245
Cluster + Ridge	0.0908
Cluster + Lasso	0.0946
Cluster + Elastic Net	0.0735
Cluster + Bayesian Ridge	0.1010
Random Forest	0.1614
Kernel Ridge	0.1916
SVR	0.2052
GBRT	0.1736
XGBoost	0.1404

7 Error Analysis

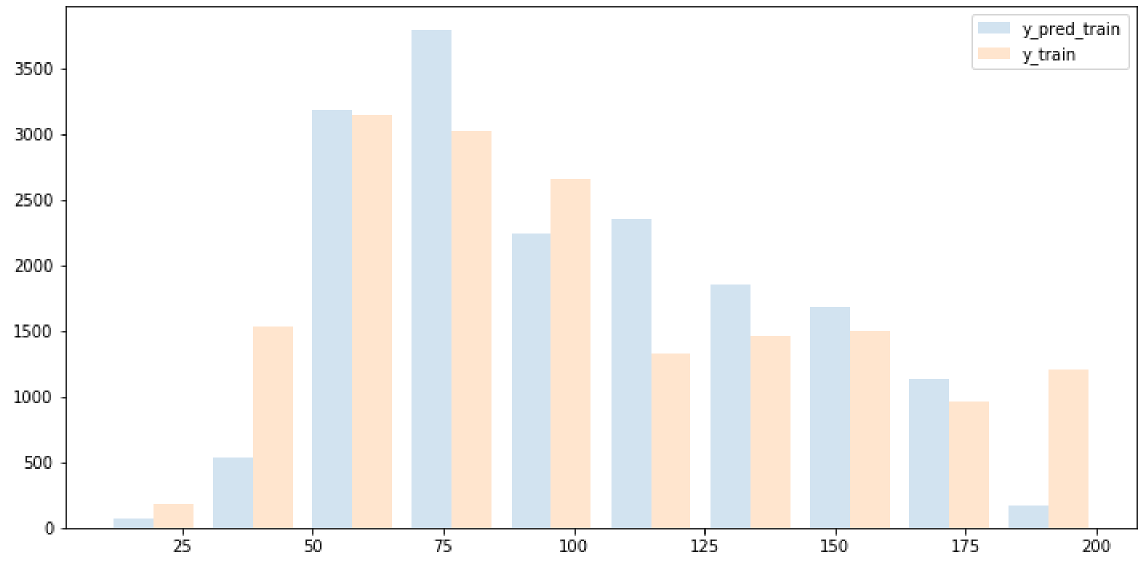
Based on the results in the previous section, we thought XGBoost performs the best, so our error analysis focused on this model.

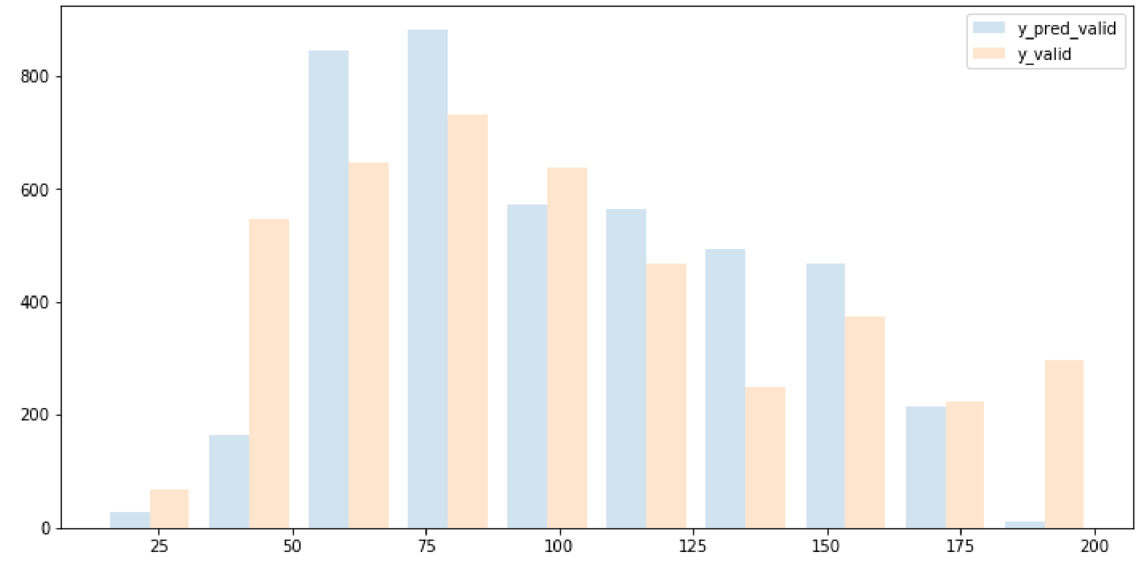
First, we checked the feature importance for interpretability. Below is the figure of top 15 feature importance:



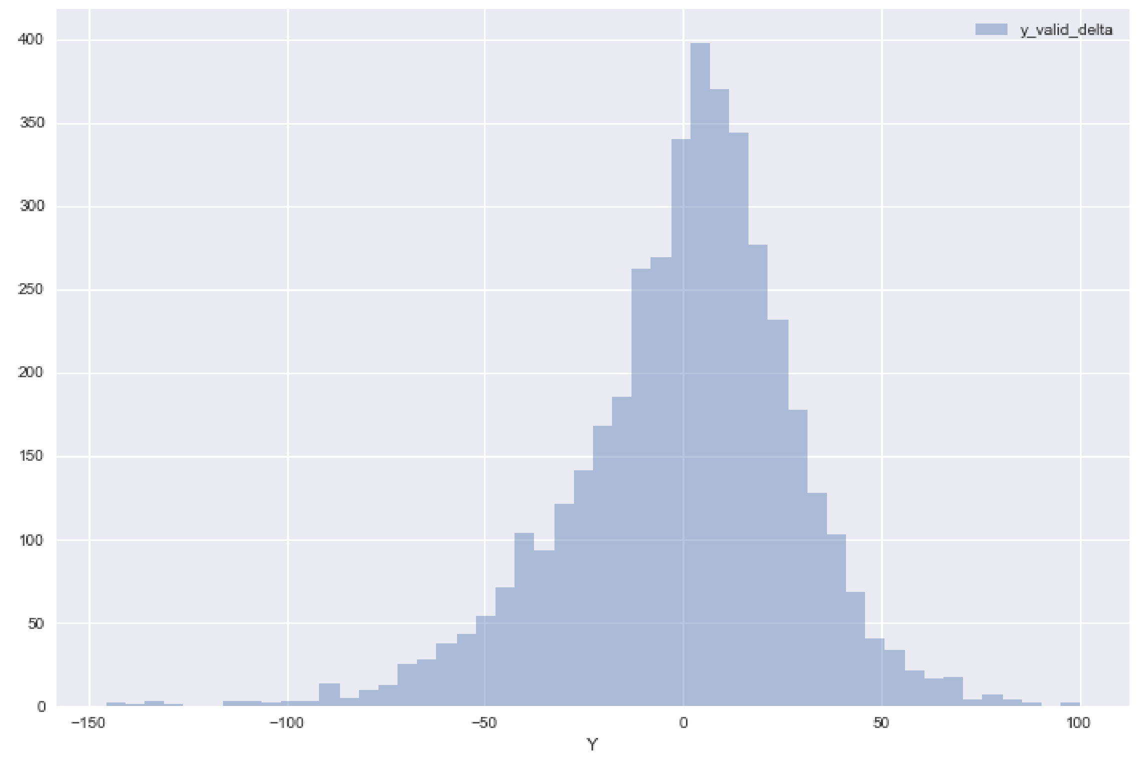
According to this plot, it makes sense that `zipcode` and `minimum_nights` have the highest F-score because intuitively they are essential information needed for every housing post. Besides, `accommodates`, `beds`, and `bathrooms` are among the top ranks, which also makes a lot sense because they are used as filtering information on the Airbnb website. Additionally, some features about amenities such as `TV`, `Internet`, and `Buzzer/wireless intercom` have relatively high F-score since these features describe the most common facilities in any housing. Therefore, the features selected by XGBoost are reasonable.

Next, we checked distributions of target price and predicted price in both training set and validation set.

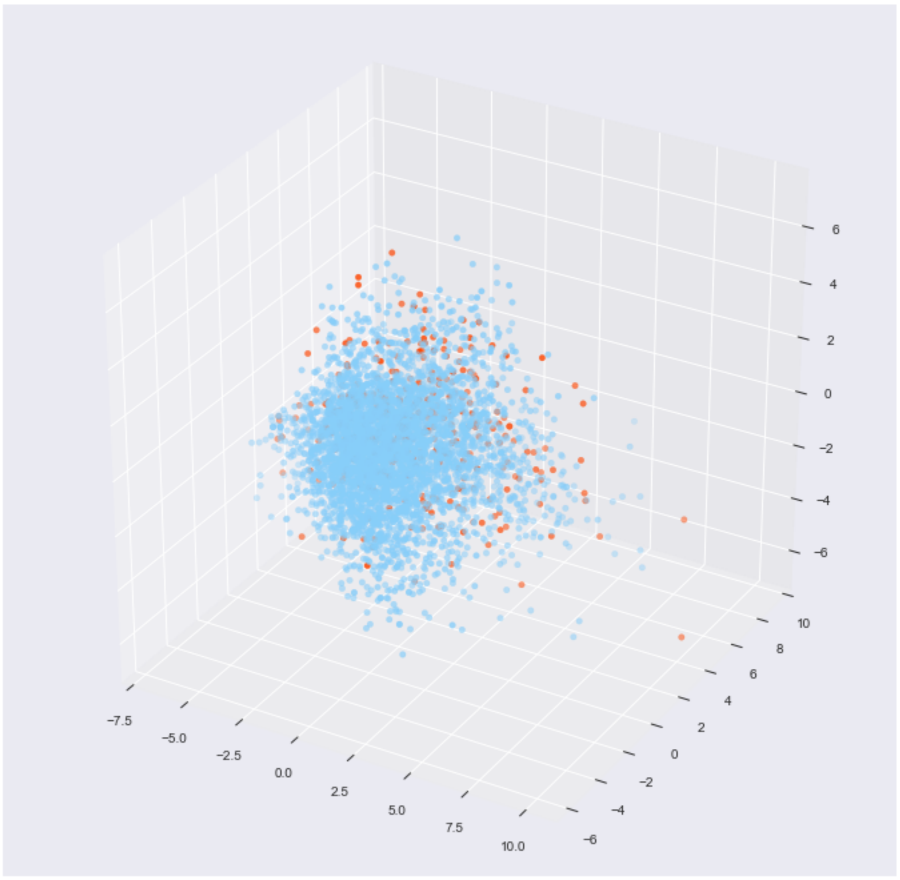




These two figures show that distributions in training set and validation set are similar, which implies that our model didn't suffer from overfitting.

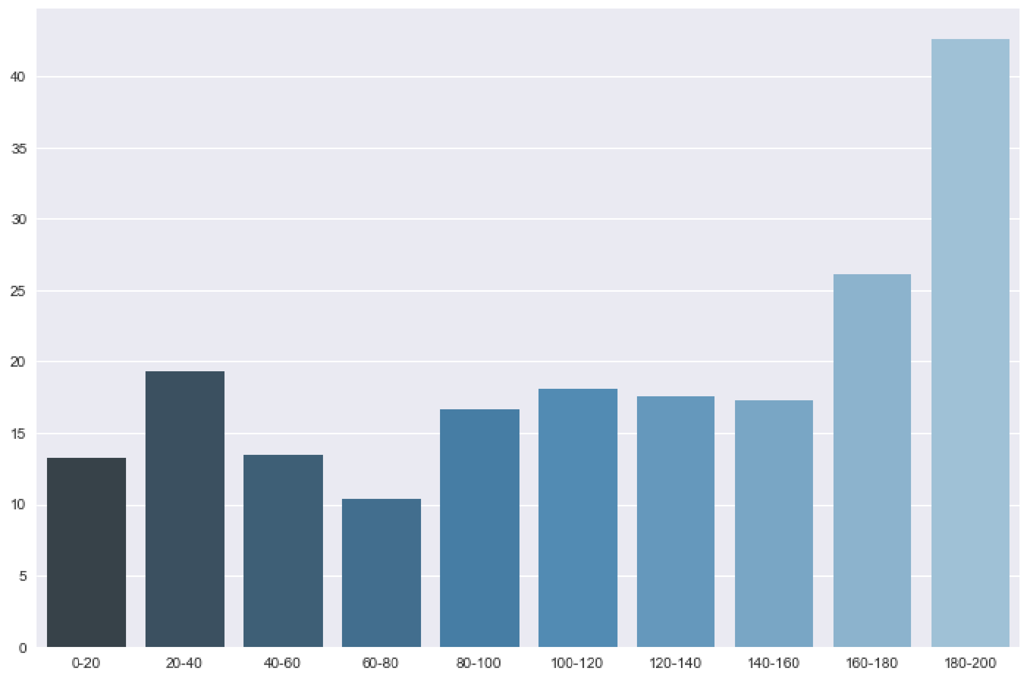


The above plot demonstrates that the distribution of error is symmetric about 0, so we confirmed that our model works not bad. However, we noticed that there are some anomalies far from their real prices. Hence, we filtered out the anomalies that have high absolute errors (greater than 70) and ended up with 104 anomalies in total. Later, we tried to discover a pattern of these anomalies. In order to plot them in a 3D figure, we utilized PCA and obtained the following plot.



From the above plot, it seems that the anomalies of predicted price are probably outliers of features, which are located in the outer layer of projected data instances.

Besides, we also found that most anomalies lie in the two ends of the price distribution, which means that our model tends to avoid extreme prices (low price and high price). To confirm this implication, we computed the median absolute loss for different price ranges:



As can be seen from the above plot, the highest two price ranges (160-180 and 180-200) have extraordinarily high error, implying that XGBoost doesn't accurately predict high prices.

8 Conclusion and Future Works

After employing different models, our findings suggest that XGboost performs the best in terms of both median absolute error and runtime. The second best model is GBRT, which achieves almost the same error but has longer runtime. Kernel Ridge Regression and SVR are not suggested because they cost long runtime but still attain relatively high error.

Plenty of topics can be explored further. For instance, we can incorporate seasonality or day of the week into our model to discover the relationship between prices and time patterns. We might also predict a price range rather than a single price. To further reduce runtime, we would like to attemp LightGBM.

References

Basak, Debasish, Srimanta Pal, and Dipak Chandra Patranabis. ”Support vector regression.” *Neural Information Processing-Letters and Reviews* 11.10 (2007): 203-224.

Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785-794). ACM.

Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.