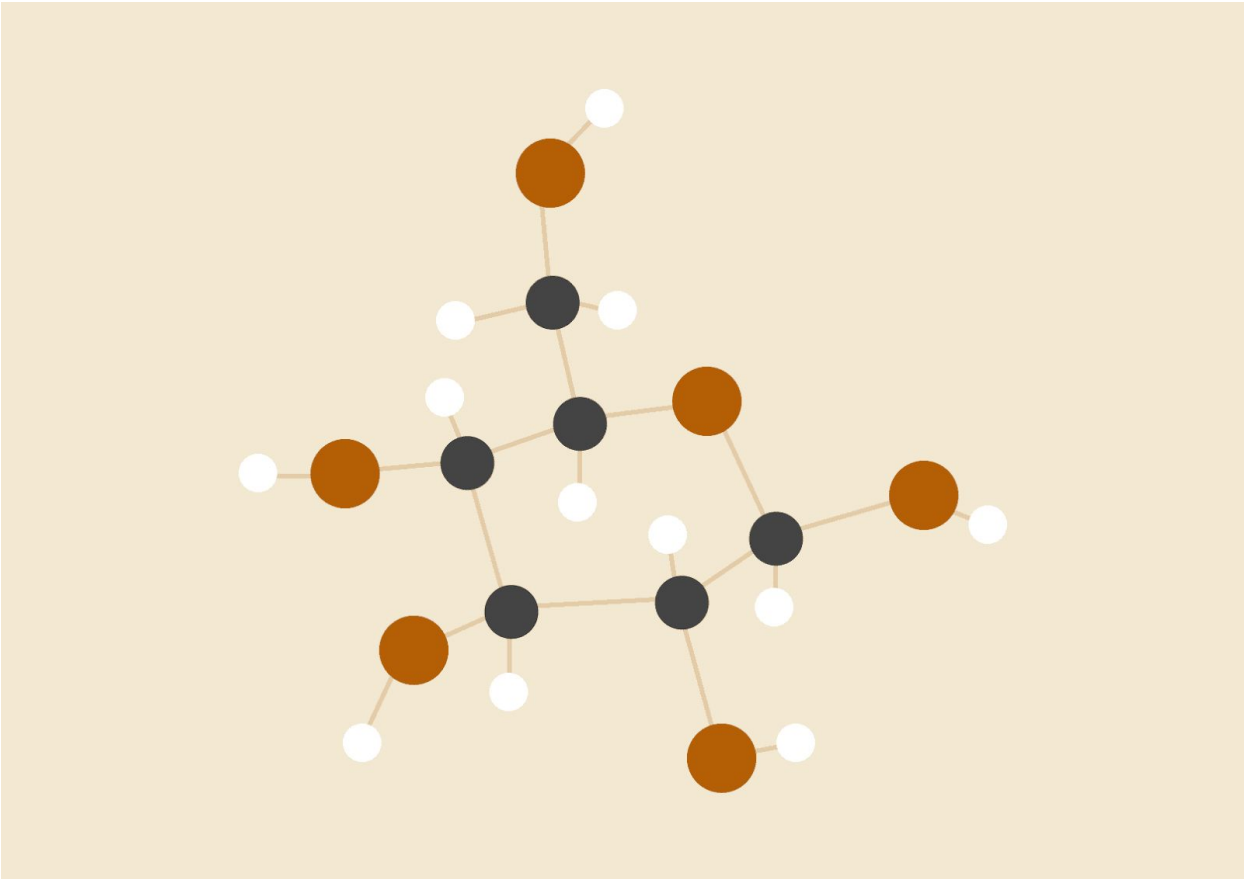
Predicting House Price



Dat Nguyen

12.18.2019

Source code:

https://datduyng.github.io/archives/cs440/cs440_project_predicting_house_price.pdf

Abstract

Recent progress in the discriminative model has been fuel in many applications. We experienced a use case of a Linear and logistic gradient-based learning technique. We compare linear model and Random Forest Regressor performance on New york Airbnb Dataset. Our best model achieves a MAE of 0.7 in the scale of the baseline 84 MAE score. Hyperparameters tuning was conducted to determine the optimal hyperparameters, data preprocessing was also utilized to improve the models' performance.

Introduction

The problem that we are going to solve here is given a set of features that describe a house in New york, our model must predict the house price. We use Scikit-learn Python library framework to help us train and analyze the data.

A house value is simply more than location and square footage. Like the features that make up a person, an educated party would want to know all aspects that give a house its value. We are going to take advantage of all of the feature variables available to use and use it to analyze and predict house prices. We are going to break everything into logical steps that allow us to ensure the cleanest, most realistic data for our model to make accurate predictions from.

Linear Model

Linear Regression is one of the most widely used models for both statistics and machine learning problem. In statistics, it is a way to model and explain a linear relationship between one or more independent variables to its corresponding dependent variable. In machine learning, it is a model used to predict a range of continuous value. It falls under the category of supervised learning as it required labeled data, where the model relies on learning from the training data. There are two approaches to linear regression problem, namely the closed-form solution, and the iterative optimization approach. The closed-form solution directly computes the model parameters that best-fit to the training set by compute the inverse matrix. However, the closed-form solution is computational expensive. For example, if there are N number of training data and C number of

features. The computation will take $O(C\text{^}2N)$

For matrix-inverse operation, which can be very expensive when the sample size is huge. Also it will be unstable if the matrix is ill-conditioned, which can be caused by having correlated features. The gradient descent approach is used when there are significant amount of data and computing the matrix inverse becomes infeasible. It is used to optimize the model weights by iteratively updating the weight and minimizing the cost function of the model on the training data. It also avoid the ill-conditioned problem that the close-form approach has.

Random Forest Regressor

Random forest is one of the most effective and simple non-linear model. The random forest model is a type of additive model that makes predictions by combining decisions from a sequence of base models. More formally we can write this class of models as

$$R(x) = T_1(x) + T_2(x) \dots T_n(x)$$

Where R(x) is the sum of all other simple model. Simple Model, T is often implement as "Decision Tree". This technique of combining multiple week learner call "model ensembling"

Data Summary

The dataset is about house price that can be obtained from Kaggle[1]. There are 38999 instances of house prices in NYC Airbnb Dataset. Each with 16 features to determine house price. There are 5 non-numeric feature in the dajtaset. The summary for the variables are listed in Figure 1. Each of the variables is listed below:

```
id                                48895 non-null int64
name                              48879 non-null object
host_id                           48895 non-null int64
host_name                         48874 non-null object
neighbourhood_group               48895 non-null object
neighbourhood                     48895 non-null object
latitude                          48895 non-null float64
longitude                         48895 non-null float64
room_type                         48895 non-null object
price                             48895 non-null int64
minimum_nights                    48895 non-null int64
number_of_reviews                 48895 non-null int64
last_review                       38843 non-null object
reviews_per_month                 38843 non-null float64
calculated_host_listings_count    48895 non-null int64
availability_365                  48895 non-null int64
```

Figure 1. NYC Airbnb Dataset features

Data Analysis

We performed multiple Data analysis techniques to determine the best feature to train our model. We first look at the correlation matrix among the numerical features. We didn't decide to drop any numerical in this step because of the small number of feature that the dataset holds. A Correlation matrix is shown in Figure 2 below.

Figure 2: Correlation matrix among feature

From the correlation matrix, we conclude that many feature hold redundancy information such as column id and host_id. This two column can create a high bias to our model because of their clarity. Longitude and Latitude also have the same correlation. This is expected because the longitude and latitude of these locations spread across only New York area. Number of reviews and Review per month are also highly correlated to each other. This indicates that these 2 features are similar and hold different unit. After this extraction, we now have minimum nights, “number_of_reviews”, “reviews_per_month”, “calculated_host_listing_count”, “avaibility_365” as numerical feature for our training set. Chosen object fields are “Neighborhood group” and “Room type”

Any rows with N/a or Nan numeric value will be dropped as this will cause the algorithm to have a nan loss.

Feature scaling

The statistics of the dataset is shown in Table 1. The features have different distributions

and different scales, which can be

difficult for the gradient descent algorithm to converge to a global minima, because the algorithm updates the weight by choosing the steepest gradient during each step Uneven scales can be detrimental, making the algorithm to converge with much longer time and potential landing in a suboptimal solution. In addition, gradient descent is sensitive to the scale of data. During the implementation, it can cause instability in computation, one apparent symptom is overflow or underflow during learning. It is known as the exploding-gradient and diminishing-gradient problem. Thus, in this study, it is essential to scale the features prior training, the features are normalized by converting them into zero mean and unit variance.

	neighbourhood_group	latitude	longitude	room_type	price	minimum_nights	number_of_reviews	calculated_host_listings_count
count	48895.000000	48895.000000	48895.000000	48895.000000	48895.000000	48895.000000	48895.000000	48895.000000
mean	1.675345	40.728949	-73.952170	0.504060	152.720687	7.029962	23.274466	7.143982
std	0.735816	0.054530	0.046157	0.545379	240.154170	20.510550	44.550582	32.952519
min	0.000000	40.499790	-74.244420	0.000000	0.000000	1.000000	0.000000	1.000000
25%	1.000000	40.690100	-73.983070	0.000000	69.000000	1.000000	1.000000	1.000000
50%	2.000000	40.723070	-73.955680	0.000000	106.000000	3.000000	5.000000	1.000000
75%	2.000000	40.763115	-73.936275	1.000000	175.000000	5.000000	24.000000	2.000000
max	4.000000	40.913060	-73.712990	2.000000	10000.000000	1250.000000	629.000000	327.000000

Table 1. Statistic summary of Airbnb dataset features.

Methods

Batch vs Stochastic gradient descent for Linear model

The batch gradient descent (BGD) and stochastic gradient descent (SGD) algorithm was implemented as an iterative optimization method. For this problem, iterative approach is more suitable than closed-form solution for the following reasons: (1) our dataset is significantly large; (2) small number of features; (3) there are non-linear relationship between input features and output target. Batch GD uses the whole batch of training data at every step, which makes it extremely slow when the training set is large. SGD on the other hand picks a random instance in the training set at every step and computes the gradients based only on that single instance. This makes it more suitable to implement as it is much faster since it has less data to manipulate at every iteration. When implementing the GD algorithm we need to consider the following two issues: (1) if the learning rate is set to too low, the algorithm will take a long time to reach the solution. (2) If the learning rate is set to too high, the algorithm will diverge, where the weight erratically moving all over the place instead landing itself to the global minima.

Lasso Regressor

Lasso regression addresses some of the problems of Ordinary Least Squares by imposing a penalty on the size of the coefficients with l1 regularization.

Ridge Regressor

Ridge regression addresses some of the problems of Ordinary Least Squares by imposing a penalty on the size of the coefficients with l2 regularization. L1 and L2 regularizations are termed added in the SGD update rule in Figure 3.

L1 regularization on least squares:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2 + \lambda \sum_{i=1}^{k} |w_i|$$

L2 regularization on least squares:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2 + \lambda \sum_{i=1}^{k} w_i^2$$

Figure 3. L1 and L2 regularization term

Random Forest Regressor

Random forest regressor is a simple non-linear model. Base on the correlation matrix in Figure 2, we conclude that the dataset is not linearly separable. The highest correlated score with price column is 0.057.

Results

Establishing a base line

Before diving in and implement compare all the model, we establish a base with a hope that our model will perform better. If a model cannot improve upon a baseline then it is a failure. The base line is implemented as follows.

Naive baseline

In [61]:
```
# Naive baseline is the median

import numpy as np


median_pred = y_df.median()

print(median_pred)

median_preds = [median_pred for _ in range(len(y_df))]
true = y_df


# Display the naive baseline metrics
mb_mae, mb_rmse = evaluate_predictions(median_preds, true)
print('Median Baseline  MAE: {:.4f}'.format(mb_mae))
print('Median Baseline RMSE: {:.4f}'.format(mb_rmse))
```

```
106.0
Median Baseline  MAE: 84.0945
Median Baseline RMSE: 244.6542
```

Model performances

Model	RMSE	MAE
Base line	244.65	84.0945
Linear Regression(OLS)	154.00	63.9448
Logistic Regression	155.63	62.7219
Lasso	156.9551	63.6070
Ridge	153.96	63.6774
Random Forest Regressor	3.4502	0.71

CONCLUSION

The gradient descent method is a relatively simple algorithm to implement. We have

shown that the NYC Airbnb dataset is not optimal as a linear regression problem. In all model, dropping redundant features improves our regression model. Data scaling is also essential for the stability of training a gradient descent model. In this study, the optimal model with optimal configuration is Random Forest Regressor with Maxdepth of 2.

REFERENCES

1. New York City Airbnb Open Data, https://www.kaggle.com/dgomonov/new-york-city-airbnb-open-data
2. De Cook, Dean. “Ames, Iowa: Alternative to the Boston Housing Data as an End of Semester Regression Project.” Journal of Statistics Education, vol. 19, no. 3, 2011.
3. Yu et al, “Real Estate Price Prediction with Regression and Classification”. 2016. available at http://cs229.stanford.edu/proj2016/report/WuYu_HousingPrice_report.pdf
4. Zhang et al, “Solving large scale linear prediction problems using stochastic gradient descent algorithms”. 2004. available at https://dl.acm.org/citation.cfm?id=1015332