p1_boston_housing

Joseph Kim

January 19, 2016

Contents

Chapter 1

Project 1: Predicting Boston Housing Prices

1.1 1) Statistical Analysis and Data Exploration

```
Number of data points (houses): 506
Number of features:  13
Minimum price: $5,000.00
Maximum price: $50,000.00
Mean price: $22,532.81
Median price: $21,200.00
Std dev: $9,188.01
```

1.2 2) Evaluating Model Performance

Performance Metric The two candidates that I considered for measuring the model performance of this dataset were Mean Squared Error and Mean Absolute Error mainly because these are ideal for measuring errors for target data containing continuous values since in this project, the target values are dollars. Of the two (MSE and MAE), I decided to go with MSE because it punishes the larger error values more severely, so that the resulting model will have a better fit. Originally, I wanted to go with root mean squared error (RMSE), which is the square root of MSE, since then the error values will be more "meaningful" in terms of being in the $ space, rather than the $\$^2$ space. However, sklearn didn't have RMSE as one of the builtin options, which makes sense since doing the extra square root computation doesn't change the fitted models. The other error metrics that were discussed in the lectures for this project, such as accuracy, precision, recall, and f1 score are better suited for classification problems, not regression problems.

Splitting data It's important to split the data into separate training and testing sets in order to provide a mechanism for assessing the model to make sure we're not over-fitting on training data. Optimizing the parameters and model complexity without a test set, and using the entire dataset to train will result in a model fitted well to the training data only, which means that the model will not generalize well. When trying to predict the housing prices of new data points, the overfitted model will not be very accurate.

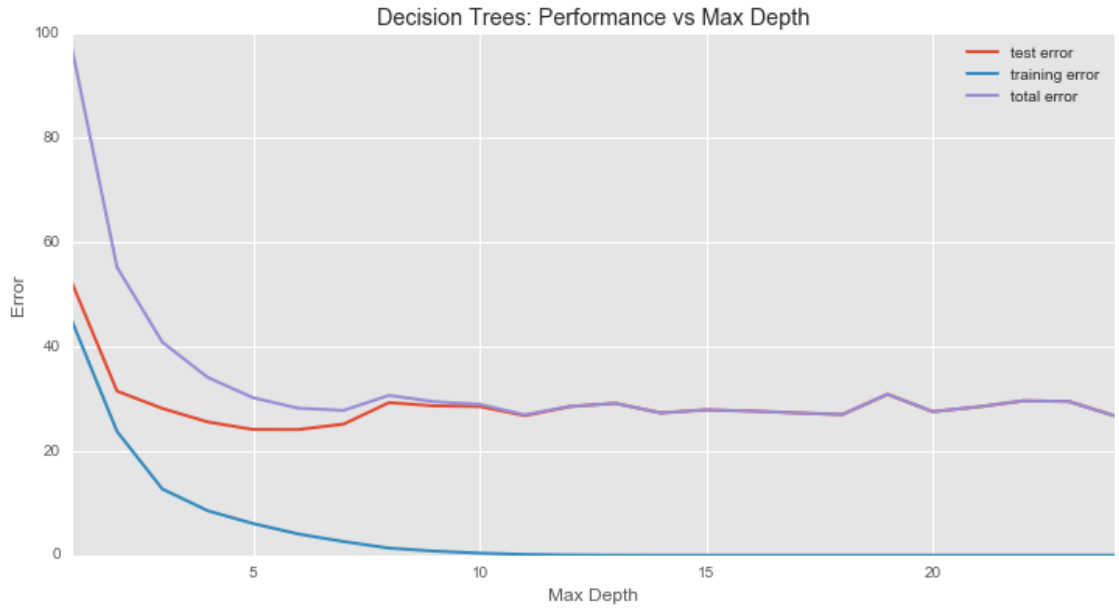
Grid Search Grid search is used to systematically find the combination of model parameters that will produce the best results. This method of fine-tuning the model by trying out different values for the parameters will result in a model that performs as optimally as possible using the evaluation metric of choice, so that the final model with the optimized parameters will predict outputs of new observations as accurately as possible. The `grid_search` module in Sklearn allows the users to quickly set up a grid search using only a couple of lines of code, so that there is minimal work on the part of the user.

Cross Validation CV is useful especially when you don't have a lot of training data since it allows the use of the full data set for training and testing by taking different combinations of the data. Although the training time is longer, cross validation will allow the user to see how a model will perform in general,

since testing on different combinations of the data simulates a new dataset. This is good to use with grid search since tuning the parameters of a model is likely to be affected by the training data, so testing the performance of a model with given parameters on multiple training and test sets, and averaging the error values gives you a good idea of the general performance, while also having the added advantage of averaging out extreme error values that may result from anomalous splits.

1.3 3) Analyzing Model Performance

Generally, in each of the plots above, as the size of the training data increases, the training error increases, while the test error decreases. At first, the test error drops quickly as the model sees more training data because the prameters become better tuned, but then the complexity of the model is too low to do any better with additional training data, so the rate of decrease becomes lower and lower until it almost plateaus. The increase in training error results from the bias error. When the model complexity is low (max_depth = 1,2,3 in the above plots), the two errors near-converge with each other at the bias error. With higher complexities (max_depth = 8, 9, 10 in the above plots), the training error is low, since the model overfits the training data with high-complexity models that are overfitting. The test error is high since the model does not generalize well, and there is a gap between test error and training error. Basically, the model with max_depth=1 suffers from high bias, while the model with max_depth=10 suffers from high variance.



As the model complexity increases, the test error first drops, then increases. This is because the model begins to overfit the training data at around `max_depth` of 5, so as the training error continues to decrease toward 0, the model does not do well in predicting the test data. The total error is the sum of these two error values, and you can see that the total error begin to converge with test error after `max_depth=5`, since total error increases as test error increases. Therefore in this data split, the model with `max_depth` of around 5 will best generalize the dataset, being complex enough to capture the patterns in the training data without overfitting.

1.4 4) Model Prediction

Running the grid search cross validation 100 times, the most common best max_depth value was 4. Setting the max_depth parameter to 4 results in the predicted price of $21,629.74. In the 100 randomized runs of GridSearchCV, 4 was the optimal max_depth value 57 times, while the next most common value was 5, which occurred 12 times.

The predicted price is a little less than the mean (~$22.5K), but is within 1 standard deviation (~$9K) of the mean.

Also, using `sklearn.neighbors.NearestNeighbors`, the average price for the top 10 data points who had the closest feature values to our query vector (nearest neighbors) was ~$21.5K, and the average price for the top 20 nearest neighbors was ~$18.2K.

Based on this result, the model seems to be reasonable, but without further analyses (i.e. feature engineering), this analysis alone is not enough to guarantee the client the best price for his or her house!