Predicting Housing Prices in Boston

Project 1, Udacity ML Engineer Nanodegree

Abstract

I used scikit-learn to create a regressive decision tree model, optimize it’s performance against mean squared error programmatically and used it to create predictions of housing prices based on a novel input.

Mike Cassell

Mike.Cassell@Rogers.com

Exploration and Statistical Analysis of the Dataset

The dataset had 506 house sale records. The range of prices was from a minimum of $5,000 to a maximum of $50,000. The mean price was $22,532.81 and the median was $21,199.99 with a standard deviation of $9,188.01.

We had a total of 13 features to work with plus the label (price) which were all left in the model (no explicit feature selection was employed).

Evaluating Model Performance

I selected the mean squared error as the measure to evaluate the performance of the model. This was selected since the modal was regressive (predicting a discrete value) and I'm more worried about approximating the correct prices closely instead of reducing the overall divergence from the test data (as would be the case in minimizing the mean absolute variance.) It should be noted as well that the regression criteria for continuous decision tress is listed as using mean squared error so it makes sense to use it in evaluation as well (as discussed in the forum.)

It's important to use data splitting in the creation and testing of a model to ensure that the risk of overfitting is minimized. By fitting the model on a partial set of the data, we can test for overfitting if the test data scores differ significantly from the training scores. If the model was built on the full set of data, it would be impossible to test the model on novel data.

In this model, I made use of scikit-learn's Test Train Split feature from cross validation. This is an efficient way to both shuffle the data (to ensure there is a random sample) and divide the dataset into a test and training set with the labels separated from the features.

Lastly to optimize the performance of the model, I made use of GridSearchCV to automatically tune the maximum depth (in splits) of the decision tree. GridSearch works by testing the range of values from 1-10 and selecting the model that performs the best on our training data and sets the final model to use the optimized parameter. This can be done recursively to choose the best combination of several parameters but in this case to ensure the visualizations of Performance vs. Training Size remain consistent, I left it at only 1 parameter being optimized this way.

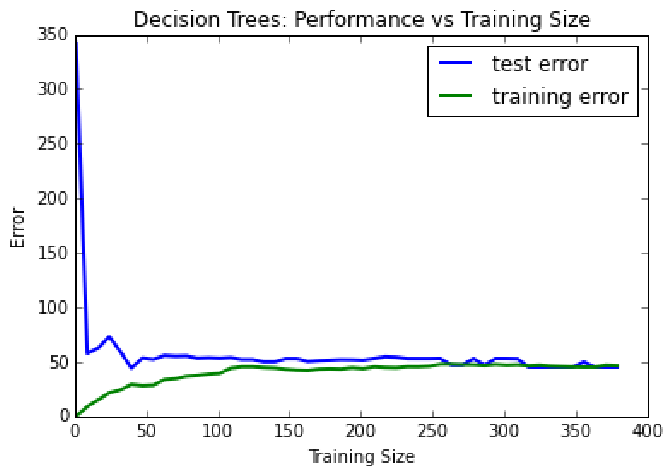
Analyzing Model Performance

Overall as the training size increases proportionally to the size of the total dataset, we see the training error rate rise and plateau while the test error rate falls from the poor early scores (where the bias was very high due to the very small training set) eventually reaching a plateau around the same level as the test data.

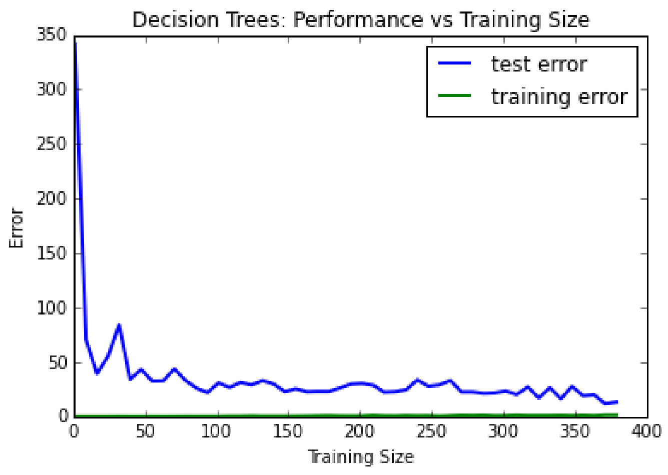
When we compare between the lowest and highest depth decision trees, we can see that the lowest shows very poor performance in both the test and training sets. This is due to a high degree of bias since the model is too simple to adequately model the data (only making two splits). The version with a depth of 10 is showing high variance with the testing score moving around significantly with small changes in the sample size. This is due to overfitting of the model.

Our grid search optimized level of a max depth of 5 shows much less bias while not introducing as much variance across the range of training sample sizes and is in agreement with the plot. In terms of complexity, the training error drops almost exponentially as additional levels of depth are added. The Test error rate is much more consistent but reaches a minimum at the 5th level of depth and so represents the best trade-off between bias and variance.

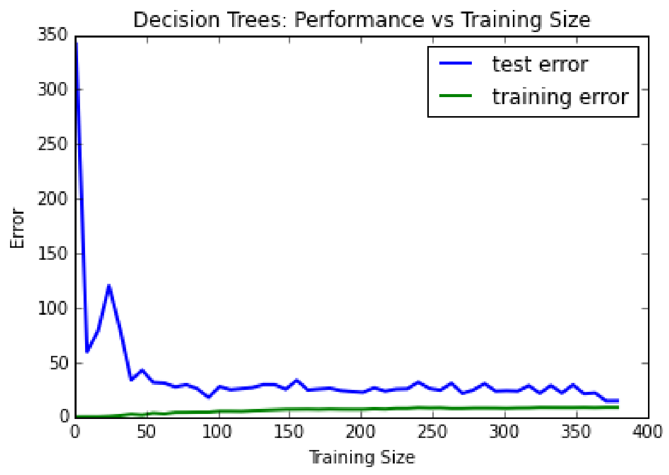
Decision Tree with a Maximum Depth of 2



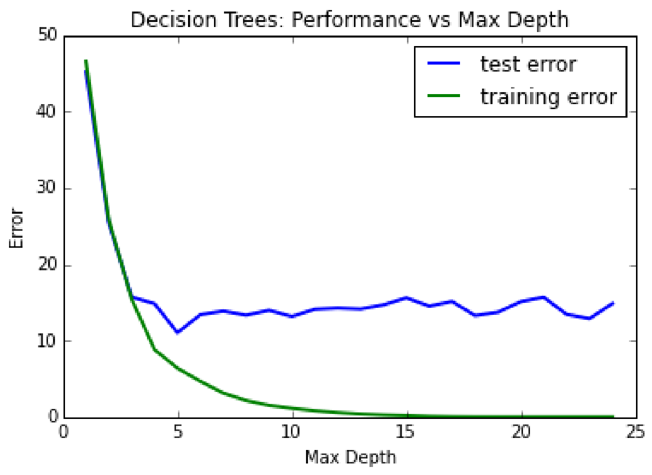
Decision Tree with a Maximum Depth of 10



Decision Tree with a Maximum Depth of 5



Model Complexity



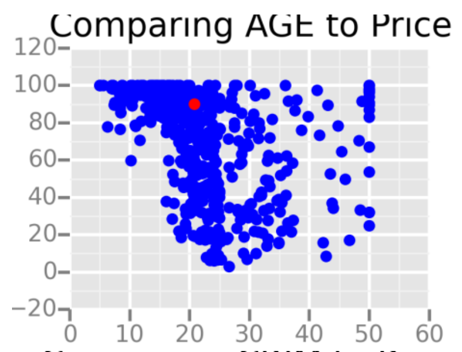
Model Predictions

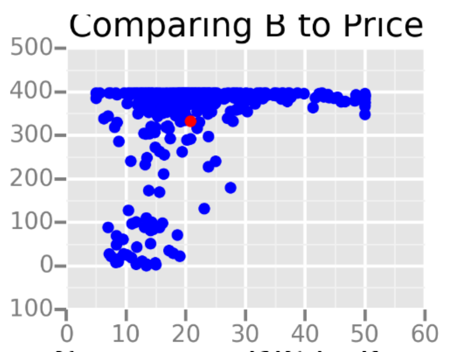
The test data point we were asked to predict was as listed below. The predicted value of the house was $20,798. This is very close to the mean and median values in the sample (well within a standard deviation.) Plotting each of the features confirms that where there are more suggestive visual trends between the features and pricing, it appears to be a good predictor (acknowledging this is a rough way to compare since we are not taking the weighting or decision structure into account).

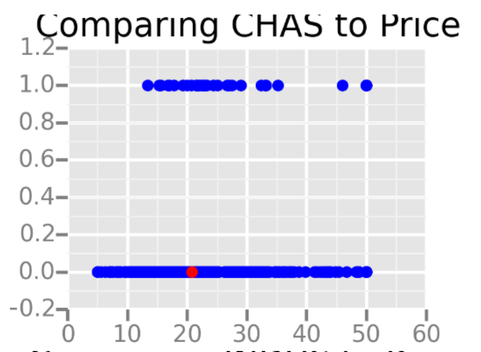
Prediction Property Values

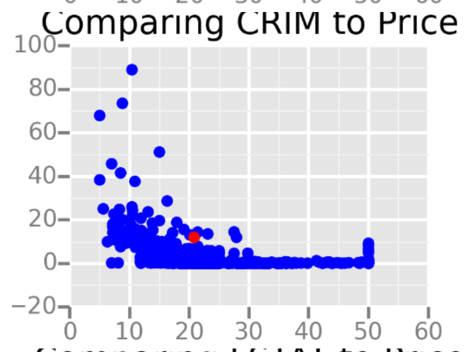
CRIM:	11.95	NOX:	0.659	RAD:	24
ZN:	0	RM:	5.609	TAX:	680
INDUS:	18.1	AGE:	90	PTRATIO:	20.2
CHAS:	0	DIS:	1.385	B:	332.09
LSTAT:	12.13				

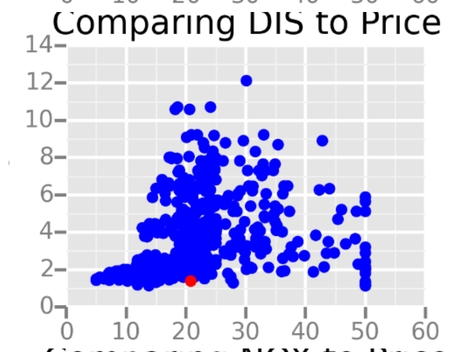
Plotting the Predicted Home against each feature's distribution with price individually (red point is the predicted):

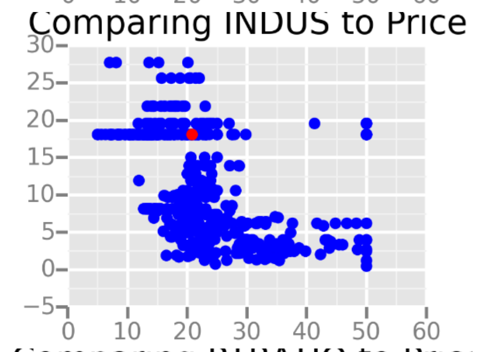


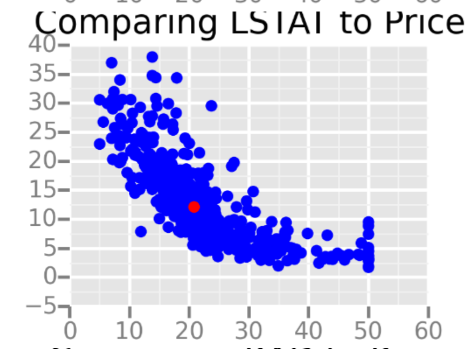


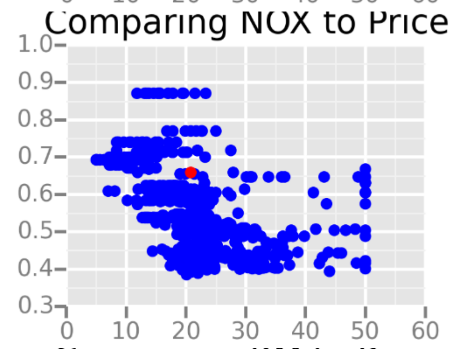


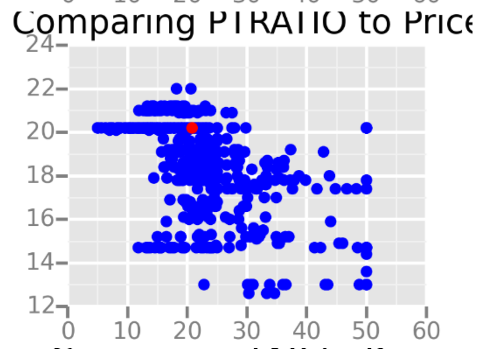


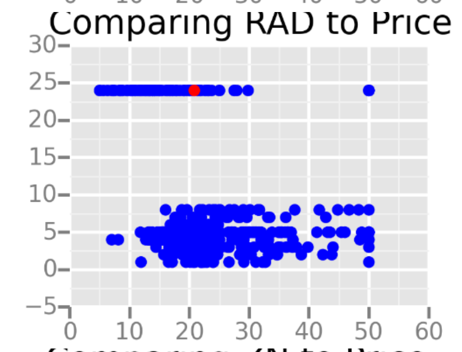


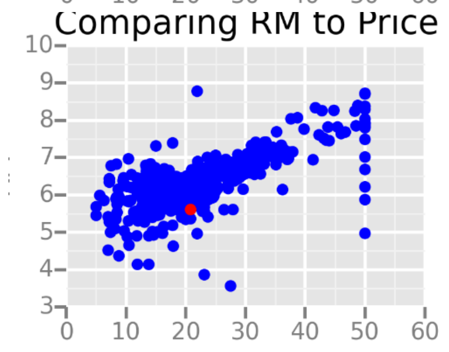


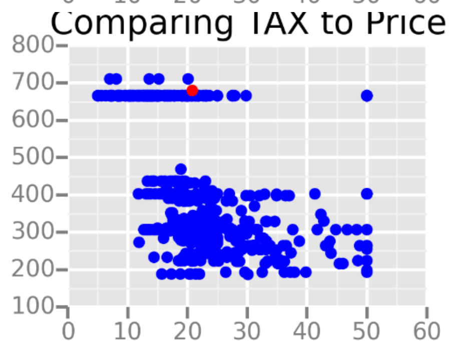


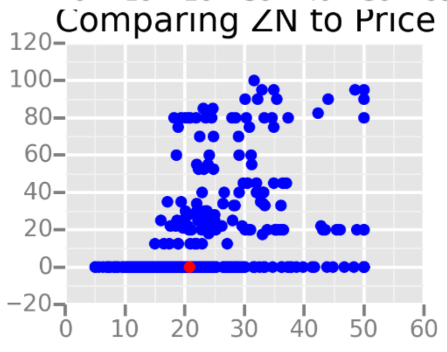












References:

Scikit-Learn’s documentation: http://scikit-learn.org/

Udacity Discussion Forumns