# Project 1: Boston Housing Prices

Patrick Martin

---

This project required synthesis of introductory modeling and evaluation material. Using the provided code skeleton, I modified four sections of the code to collect basic statistics from provided data, implement a performance metric, analyze the learning curves and model complexity, and tune a `DecisionTreeRegressor` algorithm.

## Statistical Analysis of Housing Data

The Boston housing data contained **506** entries that each have **13** features. Using `numpy` array functions, I determined the following statistical measures:

Minimum home price = **5.0**

Maximum home price = **50.0**

Mean home price = **22.5328**

Median home price = **21.2**

Standard deviation = **9.188**

## Model Evaluation

Home prices are based on continuous numerical values (or, drawn from $\mathbb{R}$). Consequently, predicting a new home price based on current home price data is a **regression** problem. Choosing an error-based metric fits the problem, since we want the prediction algorithm to place values as close as possible to comparable homes. In particular, I selected the **mean-squared error** metric to penalize incorrect predictions far more than an absolute error. The other scoring mechanisms discussed in class (e.g. precision, recall) make most sense for classification problems, not regression.

When you tune a machine learning algorithm, you have to keep in mind the tradeoff between error and complexity. If you overfit the algorithm, it will be too sensitive to outlier data; however, if you underfit the algorithm, it will have poor prediction performance. The tuning of the `DecisionTreeRegressor` used a 70/30 split of the provided housing data set using the simple function call to `train_test_split`. By pulling out 70% of the data randomly for training data, it will lessen the chance of bias, or under-fitting, into the predictor. Overall, we need to provide as much un-biased training data to the learning algorithm so that we get a better learning result.

The cross validation tool we used in this project was `GridSeachCV` that accepts a candidate algorithm, the parameters of interest, and a scoring function. By leveraging a known algorithm, the best `max_depth` of the `DecisionTreeRegressor` is computed with an exhaustive

search. My particular implementation uses the same `mean_squared_error` performance metric and passes in the `max_depth` parameter array. Performing an exhaustive cross validation operation prevents the algorithm from being tuned to a particular subset of the housing data set. Otherwise, the algorithm could possibly favor cluster of data to make an incorrect prediction. If `train_test_split` were the only method used to tune the algorithm, we would likely introduce more variance sensitivity in the model. The default `GridSeachCV` performs a 3-fold cross-validation process that diversifies the training data. Doing so allows for this particular data to be used three more times to train the model.

# Model Performance

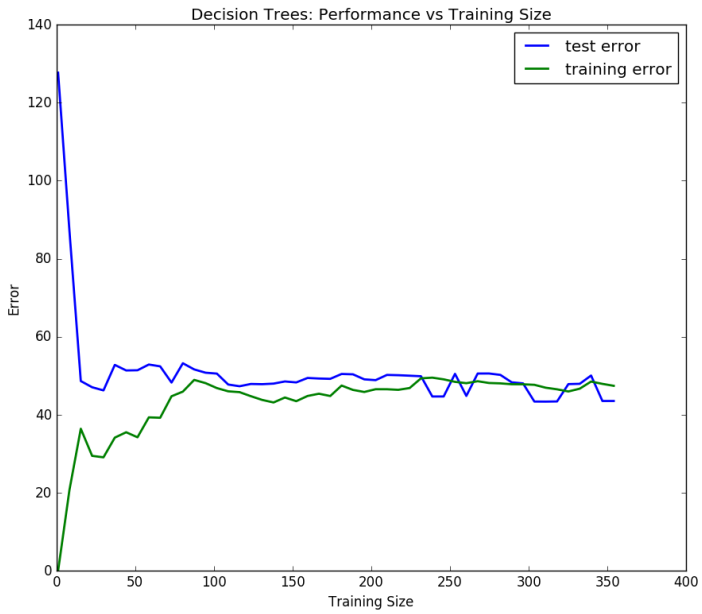


Figure 1: A plot of the training and test data error over size of training. This image is the initial run with the decision tree `depth = 1`.

The provided skeleton code created a loop to increase the maximum decision tree depth size and plot the error results. Figures 1 and 2 illustrate these curves for different `max_depth` values. By examining these figures (and the ones generated at run time) the **training error approaches a plateau** as the depth of the decision tree increases and as the amount training data is increased. With a lower depth, like Figure 1, this plateau results in a very large training error, but the error hovers around the test data error. The **testing error decreases** as the size of the training data set increases. **Although the testing and training error decrease, there still remains an gap between the results**, even at a high `max_depth` value.

Figure 1 shows an interesting case where there is **considerable bias** with `max_depth=1`. When the depth increases to 10, the training error drops significantly as the training data size in-


Figure 2: This image shows the error with a decision tree `depth = 10`.

creases. The test error also drops, but does not converge to the training error curve. These plots indicate that the algorithm is fully trained with at least 200 data points. This model still has variance, due to the difference between training data error and test data error. Figure 2 demonstrates that the model still suffers from **high variance** since the the test and training test error remain separated by about 20. One interesting trend is that the plots based on depths 5 through 10 do not increase the test error significantly. This observation indicates that higher depth does not always mean better performance for the added computational cost.

By plotting the test error against the depth of the the `DecisionTreeRegressor`, shown in Figure 3, I get confirmation of the trend seen in the learning curves. Although **training error approaches 0**, the test error continues to **remain between 10 and 20**. Given this plot, I would recommend a `DecisionTreeRegressor` model with `max_depth` $\in \{4, 5, 6\}$.

# Model Prediction

Using the `GridSearchCV` discussed before, I ran the program several times. On the whole, the tuned model resulting from this grid search based on mean-squared error had a `max_depth=4`, with the occasional value of `5` or `6`. One sample run of the main program iterated the model fit function five times and I saw the following output:

```
depth = [4, 6, 9, 4, 4]
predictions = [21.6297, 20.7659, 19.3272, 21.6297, 21.6297]
```

These results confirm that the grid search tunes to model to favor a **tree depth of 4**.

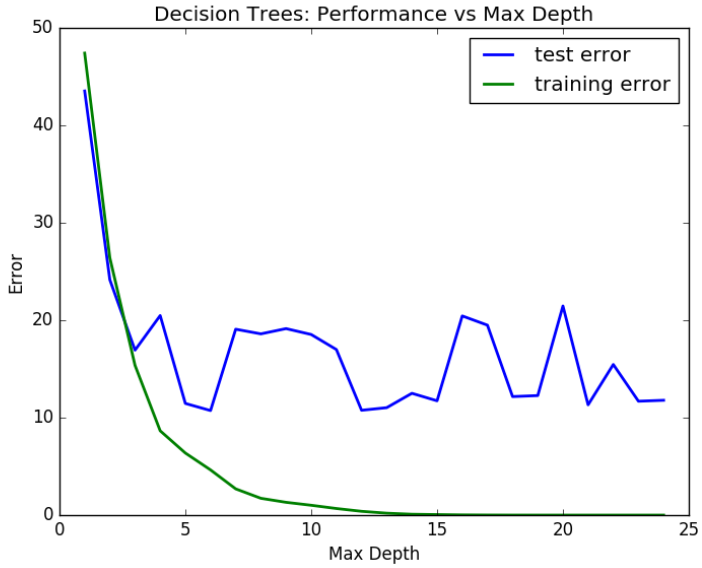


Figure 3: A graph of the model complexity as the depth of the decision tree increases.

The predictions from this tuned estimator appear to be near the mean and median home prices. In fact, they are easily within one standard deviation of the mean. The course so far has not discussed much about the output of these algorithms, other than that they are "predictors" based on prior data. How would we actually know at this point whether this prediction are reasonable other than seeing the **learning curves reducing** and that the **predicted value is near the mean**?