Predicting Boston Housing Prices

Joshua Cook

Abstract

Given a data set of 13 feature variables and a 14th target variable we use supervised learning techniques, specifically a decision tree regression, to build a regression model to predict the target variable given values for the feature variables. This is a training exercise toward becoming familiar with supervised learning techniques and the scikit-learn library.

This analysis is concerned with a dataset of housing values in the suburbs of Boston. It is a common data set used for education purposes. It is included with scikit-learn, but can also be found here.

Overview of the data

We begin our analysis by loading the data using scikit-learn's built-in mechanism for accessing this educational dataset.

```
# Importing a few necessary libraries
import numpy as np
import matplotlib.pyplot as pl
from sklearn import datasets
from sklearn.tree import DecisionTreeRegressor

# Make matplotlib show our plots inline (nicely formatted in the notebook)
%matplotlib inline

# Create our client's feature set for which we will be predicting a selling price
CLIENT_FEATURES = [[11.95, 0.00, 18.100, 0, 0.6590, 5.6090, 90.00, 1.385, 24, 680.0, 20.20,

# Load the Boston Housing dataset into the city_data variable
city_data = datasets.load_boston()

# Initialize the housing prices and housing features
housing_prices = city_data.target
housing_features = city_data.data
```

```
print "Boston Housing dataset loaded successfully!"
```

Per UCI's Machine Learning Repository, the following is the pertinent information regarding the data set.

- Data Set Characteristics: Multivariate
- Number of Instances: 506
- Area: N/A
- Attribute Characteristics: Categorical, Integer, Real
- Number of Attributes: 14
- Date Donated: 1993-07-07
- Associated Tasks: Regression
- Missing Values? No
- Number of Web Hits: 174083

Source

Origin: This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

Creator: Harrison, D. and Rubinfeld, D.L. 'Hedonic prices and the demand for clean air', J. Environ. Economics & Management, vol.5, 81-102, 1978.

Statistical Analysis and Data Exploration

```
housing_prices = city_data.target
housing_features = city_data.data

# Number of houses in the dataset
total_houses = housing_features.shape[0]

# Number of features in the dataset
total_features = housing_features.shape[1]

# Minimum housing value in the dataset
minimum_price = np.min(housing_prices)

# Maximum housing value in the dataset
maximum_price = np.max(housing_prices)

# Mean house value of the dataset
mean_price = np.mean(housing_prices)
```

```
# Median house value of the dataset
median_price = np.median(housing_prices)

# Standard deviation of housing values of the dataset
std_dev = np.std(housing_prices)

# Show the calculated statistics
print "Boston Housing dataset statistics (in $1000's):\n"
print "Total number of houses:", total_houses
print "Total number of features:", total_features
print "Minimum house price:", minimum_price
print "Maximum house price:", maximum_price
print "Mean house price: {0:.3f}".format(mean_price)
print "Median house price:", median_price
print "Standard deviation of house price: {0:.3f}".format(std_dev)
```

```
Boston Housing dataset statistics (in $1000's):

Total number of houses: 506
Total number of features: 13
Minimum house price: 5.0
Maximum house price: 50.0
Mean house price: 22.533
Median house price: 21.2
Standard deviation of house price: 9.188
```

Attribute Information:

1. `CRIM`: per capita crime rate by town
2. `ZN`: proportion of residential land zoned for lots over 25,000 sq.ft.
3. `INDUS`: proportion of non-retail business acres per town
4. `CHAS`: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
5. `NOX`: nitric oxides concentration (parts per 10 million)
6. `RM`: average number of rooms per dwelling
7. `AGE`: proportion of owner-occupied units built prior to 1940
8. `DIS`: weighted distances to five Boston employment centres
9. `RAD`: index of accessibility to radial highways
10. `TAX`: full-value property-tax rate per $10,000
11. `PTRATIO`: pupil-teacher ratio by town
12. `B`: 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
13. `LSTAT`: % lower status of the population
14. `MEDV`: Median value of owner-occupied homes in $1000's

Relevant Papers:

Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of Collinearity', Wiley, 1980. 244-261.

Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth International Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.

Question 1

As a reminder, you can view a description of the Boston Housing dataset here, where you can find the different features under **Attribute Information**. The `MEDV` attribute relates to the values stored in our `housing_prices` variable, so we do not consider that a feature of the data.

Of the features available for each data point, choose three that you feel are significant and give a brief description for each of what they measure.

Answer: I think that three features that will correlate closely with the median home-value are:

1. `CRIM`, which represents the per capita crime rate
2. `RM`, average number of rooms per dwelling
3. `AGE`, which represents the proportion of owner-occupied units built prior to 1940

Question 2

Using your client's feature set `CLIENT_FEATURES`, which values correspond with the features you've chosen above?

```
print CLIENT_FEATURES[0][0],CLIENT_FEATURES[0][5],CLIENT_FEATURES[0][6]
```

```
 11.95 5.609 90.0
```

For the dwelling the current client is looking at this corresponds to values of

1. `CRIM - 11.95`
2. `RM - 5.609`
3. `AGE - 90.0`

Evaluating Model Performance

In this second section of the project, you will begin to develop the tools necessary for a model to make a prediction. Being able to accurately evaluate each model's performance through the use of these tools helps to greatly reinforce the confidence in your predictions.

Developing a Regression Model

Selecting a Performance Metric

The essential nature of machine learning is that we develop a statistical model that is capable of optimizing with regard to the degree of error it generates given known data. We must begin our work by selecting an appropriate performance metric we will use to measure this error. Scikit-learn has several built-in tools for quantifying error. In this work, our target variable is continuous, making the prediction of the variable a regression rather than classification problem. Scikit-learn offers the following metrics for use in regression:

- `metrics.explained_variance_score(y_true, y_pred)`
  Explained variance regression score function
- `metrics.mean_absolute_error(y_true, y_pred)`
  Mean absolute error regression loss
- `metrics.mean_squared_error(y_true, y_pred[, ...])`
  Mean squared error regression loss
- `metrics.median_absolute_error(y_true, y_pred)`
  Median absolute error regression loss
- `metrics.r2_score(y_true, y_pred[, ...])`
  R^2 (coefficient of determination) regression score function

With little exposure to any of these, I chose to familiarize myself with what seem to be the least complex metrics, mean absolute error and mean squared error.

> Both the root mean square error (RMSE) and the mean absolute error (MAE) are regularly employed in model evaluation studies.[1]

[1] Chai, Tianfeng, and Roland R. Draxler. "Root mean square error (RMSE) or mean absolute error (MAE)?–Arguments against avoiding RMSE in the literature." Geoscientific Model Development 7.3 (2014): 1247-1250.

> The mean absolute error … is less sensitive to the occasional very large error because it does not square the errors in the calculation.[2]
>
> The root mean squared error is more sensitive than other measures to the occasional large error: the squaring process gives disproportionate weight to very large errors. If an occasional large error is not a problem in your decision situation (e.g., if the true cost of an error is roughly proportional to the size of the error, not the square of the error), then the MAE or MAPE may be a more relevant criterion.
>
> There is no absolute criterion for a "good" value of RMSE or MAE: it depends on the units in which the variable is measured and on the degree of forecasting accuracy, as measured in those units, which is sought in a particular application. Depending on the choice of units, the RMSE or MAE of your best model could be measured in zillions or one-zillionths. It makes no sense to say "the model is good (bad) because the root mean squared error is less (greater) than x", unless you are referring to a specific degree of accuracy that is relevant to your forecasting application.[3]
>
> If an occasional large error is not a problem in your decision situation (e.g., if the true cost of an error is roughly proportional to the size of the error, not the square of the error), then the MAE or MAPE may be a more relevant criterion.[4]

Based upon this, we will use the Mean Absolute Error as performance metric.

Training/Test dataset split

In order to perform our analysis, we split the data into two sets, a training set and a testing set. The training set will be used to the train the model. The testing set will be used to test the validity of the model being trained. If we did not reserve some of our data for testing then when we used data to test the validity, this data would have been used in the formation of the model. We would expect any data that had been used to form the model to fit the model well. With test data we can see how well data not used to form the model fits with the model that was formed.

```
In [3]: X_train, y_train, X_test, y_test = split_data(city_data)
```

[2] http://people.duke.edu/~rnau/compare.htm

[3] Hyndman, Rob J., and Anne B. Koehler. "Another look at measures of forecast accuracy." International journal of forecasting 22.4 (2006): 679-688.

[4] http://people.duke.edu/~rnau/compare.htm

Calibrating the model

For this work, we use a decision tree as our regressor. Decision trees are used in both classification and regression analyses, and is perfectly suited to supervised learning. The decision tree is used to train a model that can predict the value of a target variable. It works well for our regression in which we are seeking to predict the target value of median value of an owner owner-occupied home. The decision tree itself predicts which is the most relevant of the feature variables being passed to the model (at each decision node) and thus much of the work of the tree is largely out of our hands. One parameter over which we do have some control is the depth of the tree regression.

One of the major disadvantages of decision tree regression is a tendency to overfit, particularly with data sets with a large number of features. Given our 13-feature set, we certainly run this risk. The best way in which we can prevent overfitting is by setting a maximum depth for our decision tree. To find an ideal maximum depth, we perform a series of regressions using varying decision tree depths. For each regression, at a given decision tree depth, we plot the error for a given n (number of points in the dataset).

```
In [4]: max_depths = [1,2,3,4,5,6,7,8,9,10]

In [5]: for max_depth in max_depths:
            learning_curve(max_depth, X_train, y_train, X_test, y_test)
```

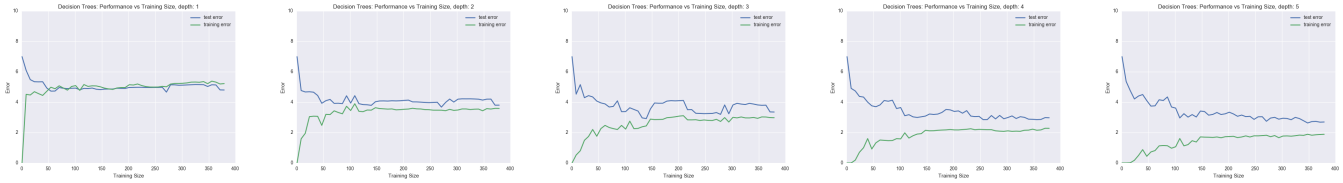


Figure 1: Decision Trees with maximum depths 1 - 5

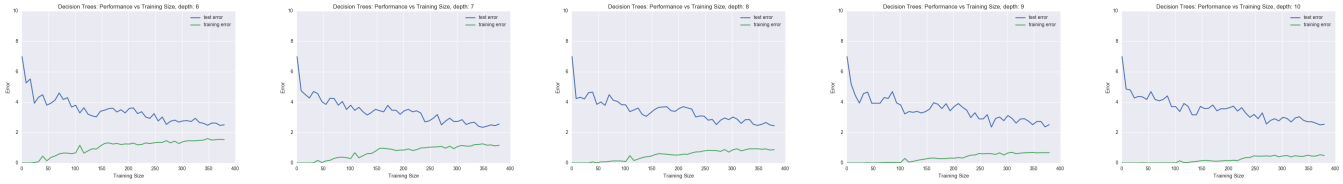


Figure 2: Decision Trees with maximum depths 1 - 5

Analyzing Training Data Performance

For a depth of 1 to 4, there is an initial rapid spike in error, after which the error settles down. This occurs around $n = 50$. Following this, the error is fairly linear with a slow increase in the size of training error. For a depth of greater than 5, we appear to lose this initial spike in the training error and the error more rapidly settles into a nearly linear increase. As the depth increases there appears to be a shallower and shallower slope for the linear increase. It is also of note that if we consider this linear increase the intercept of this slope is gradually decreasing. The intercept is about 4 for a depth of 1, hitting 0 for a depth of 6. To say this more simply, the error curve of the training data improves significantly as we increase the depth of our decision tree.

Analyzing Test Data Performance

All depths begins with a large spike and settle down by n=50. For depths of 1 to 4, the error remains largely flat as n increases. For a depth of greater than 5, the error decreases slowly and approximately linearly. The error appears to decrease at approximately the same rate.

Comparing Training and Test Data Performance

The plot for a depth of 1 is definitely underfit. The error remains flat for both test and training sets at around 5 as n increases. This would indicate a poor fit. The error for a depth of 10 would indicate overfitting in that the error for the training data is nearly flat at 0, but the error for the test data is nearly the same as that for a depth of greater than 5.

It is extremely important that while training data performance significantly increases with depth, test data performance does not. This is a sign that we may have overfitting as we increase the depth. I modified the accompanying Python script to force the `learning_curve_graph` function to plot the data on the same scale:

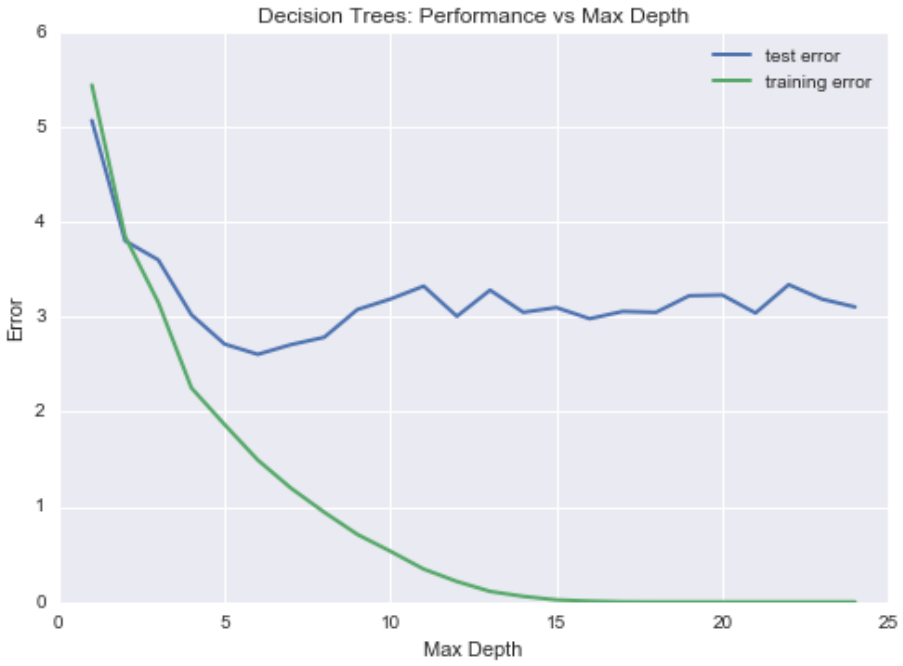
```
...
pl.ylim(0,10)
...
```

The results are remarkable. While perturbations within the test data are different, the overall shape of the error curve for plots of depth greater than four are nearly identical. The test data appears to have virtually no advantage from running the regression for a depth greater than four.

Analyzing Model Complexity

Given this observation it makes sense to explore error results for the whole set for varying levels of model complexity. For depths of 1 through 25, we perform the same regression. We use a decision tree to prepare a regression model for the given depth, then use our performance metric to find the error for that particular model. We perform this regression on both our training and test datasets.

```
In [6]: model_complexity(X_train, y_train, X_test, y_test)
```



the Bias-Variance Trade-Off

In analyzing our Model Complexity plot, we are essentially looking at the Bias-Variance Trade-Off[5]

While mean absolute error was used as a performance metric, consider the following. For mean square error, the expected test MSE is given by[6]

[5] James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning (p. 6). New York: springer. Chicago

[6] Ibid.

$$\mathbb{E}\left(y_0 - \hat{f}(x_0)\right)^2 = \text{Var}(\hat{f}(x_0)) + \left[\text{Bias}(\hat{f}(x_0))\right]^2 + \text{Var}(\epsilon)$$

With the Variance and the Bias both positive numbers and $\text{Var}(\epsilon)$ independent of the model, we seek to simultaneously minimize the Variance and the Bias. With lesser complexity, we have an oversimplified model which corresponds to underfitting or increased bias. With higher complexity, we have overfitting or high variance. The trade-off between the two corresponds to the U-shape of the plot of test error versus model complexity. The minimum of this plot represent the point where a given complexity simultaneously minimizes variance and bias.

It would appear that the test data error has a minimum near a depth of 6, while training error continues to decrease asymptotically as depth increases. Based upon this the error would decrease as depth increases until the depth of 6, after which it would increase, suffering from over fitting. This would signify that with a model depth of less than six we have an underfit model and with a model depth of higher than six we have an overfit model.

Tune and predict Model

```
In [7]: fit_predict_model(city_data)
Final Model:
GridSearchCV(cv=None, error_score='raise',
       estimator=DecisionTreeRegressor(criterion='mse', max_depth=None,
           max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False,
           random_state=None, splitter='best'),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'max_depth': (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)},
       pre_dispatch='2*n_jobs', refit=True, scoring=None, verbose=0)
House: [11.95, 0.0, 18.1, 0, 0.659, 5.609, 90.0, 1.385,
        24, 680.0, 20.2, 332.09, 12.13]
Prediction: [ 20.76598639]

In[8]: fit_predict_model(city_data)
Final Model:
GridSearchCV(cv=None, error_score='raise',
       estimator=DecisionTreeRegressor(criterion='mse', max_depth=None,
           max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False,
           random_state=None, splitter='best'),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'max_depth': (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)},
       pre_dispatch='2*n_jobs', refit=True, scoring=None, verbose=0)
House: [11.95, 0.0, 18.1, 0, 0.659, 5.609, 90.0, 1.385,
        24, 680.0, 20.2, 332.09, 12.13]
Prediction: [ 19.99746835]

In[9]: fit_predict_model(city_data)
Final Model:
GridSearchCV(cv=None, error_score='raise',
       estimator=DecisionTreeRegressor(criterion='mse', max_depth=None,
           max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False,
           random_state=None, splitter='best'),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'max_depth': (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)},
       pre_dispatch='2*n_jobs', refit=True, scoring=None, verbose=0)
House: [11.95, 0.0, 18.1, 0, 0.659, 5.609, 90.0, 1.385,
        24, 680.0, 20.2, 332.09, 12.13]
Prediction: [ 21.62974359]
```

```
In[10]: fit_predict_model(city_data)
Final Model:
GridSearchCV(cv=None, error_score='raise',
       estimator=DecisionTreeRegressor(criterion='mse', max_depth=None,
           max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False,
           random_state=None, splitter='best'),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'max_depth': (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)},
       pre_dispatch='2*n_jobs', refit=True, scoring=None, verbose=0)
House: [11.95, 0.0, 18.1, 0, 0.659, 5.609, 90.0, 1.385,
        24, 680.0, 20.2, 332.09, 12.13]
Prediction: [ 21.62974359]

In[11]: fit_predict_model(city_data)
Final Model:
GridSearchCV(cv=None, error_score='raise',
       estimator=DecisionTreeRegressor(criterion='mse', max_depth=None,
           max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False,
           random_state=None, splitter='best'),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'max_depth': (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)},
       pre_dispatch='2*n_jobs', refit=True, scoring=None, verbose=0)
House: [11.95, 0.0, 18.1, 0, 0.659, 5.609, 90.0, 1.385,
        24, 680.0, 20.2, 332.09, 12.13]
Prediction: [ 21.62974359]
```

Using Grid Search and Cross-Validation

Grid search takes a given model, a set of parameters, and a set of values for each of these parameters and compares the performance of the model against each permutation of the parameter values in sequence. The goal is to fine-tune the model against the parameters, finding the parameter against which the model performs best.

Cross validation is a process by which a training set of data is split into K sets. A train-test analysis is then performed against these sets reserving a different individual set as the test set in each pass. Using cross validation with grid search effectively multiplies the number of validation analyses performed against our model.

Analyzing Model Performance

4) Model Prediction

- *Model makes predicted housing price with detailed model parameters (max depth). Note due to the small randomization of the code it is recommended to run the program several times to identify the most common/reasoable price/model complexity.*
- *Compare prediction to earlier statistics and make a case if you think it is a valid model.*

The final prediction is run against this data point and returns the shown value.

```
House: [11.95, 0.0, 18.1, 0, 0.659, 5.609, 90.0, 1.385, 24, 680.0,
20.2, 332.09, 12.13]   Prediction: [ 21.62974359]
```

It is difficult to look at the original data set and speculate as the validity of this answer. As humans, we have little intuition for an 11-dimensional domain. That said, the process throuch which we generated this response was a reasonable one, involving both cross-validation and grid search. As such, I have a high degree of faith in this response.