boston_housing

September 9, 2017

1 Model Evaluation & Validation

1.1 Project: Predicting Boston Housing Prices

The Boston housing market is highly competitive, and you want to be the best real estate agent in the area. To compete with your peers, you decide to leverage a few basic machine learning concepts to assist you and a client with finding the best selling price for their home. Luckily, you've come across the Boston Housing dataset which contains aggregated data on various features for houses in Greater Boston communities, including the median value of homes for each of those areas. Your task is to build an optimal model based on a statistical analysis with the tools available. This model will then be used to estimate the best selling price for your clients' homes.

1.2 Getting Started

In this project, I will evaluate the performance and predictive power of a model that has been trained and tested on data collected from homes in suburbs of Boston, Massachusetts. A model trained on this data that is seen as a *good fit* could then be used to make certain predictions about a home — in particular, its monetary value. This model would prove to be invaluable for someone like a real estate agent who could make use of such information on a daily basis.

The dataset for this project originates from the UCI Machine Learning Repository. The Boston housing data was collected in 1978 and each of the 506 entries represent aggregated data about 14 features for homes from various suburbs in Boston, Massachusetts. For the purposes of this project, the following preprocessing steps have been made to the dataset: - 16 data points have an `'MEDV'` value of 50.0. These data points likely contain **missing or censored values** and have been removed. - 1 data point has an `'RM'` value of 8.78. This data point can be considered an **outlier** and has been removed. - The features `'RM'`, `'LSTAT'`, `'PTRATIO'`, and `'MEDV'` are essential. The remaining **non-relevant features** have been excluded. - The feature `'MEDV'` has been **multiplicatively scaled** to account for 35 years of market inflation.

Run the code cell below to load the Boston housing dataset, along with a few of the necessary Python libraries required for this project. You will know the dataset loaded successfully if the size of the dataset is reported.

```
In [4]: # Import libraries necessary for this project
        import numpy as np
        import pandas as pd
        from sklearn.cross_validation import ShuffleSplit
```

```
# Import supplementary visualizations code visuals.py
import visuals as vs

# Pretty display for notebooks
%matplotlib inline

# Load the Boston housing dataset
data = pd.read_csv('housing.csv')
prices = data['MEDV']
features = data.drop('MEDV', axis = 1)

# Success
print "Boston housing dataset has {} data points with {} variables each.".format(*data.s
```

```
Boston housing dataset has 489 data points with 4 variables each.
```

1.3 Data Exploration

In this first section of this project, I will make a cursory investigation about the Boston housing data and provide your observations. Familiarizing yourself with the data through an explorative process is a fundamental practice to help you better understand and justify your results.

Since the main goal of this project is to construct a working model which has the capability of predicting the value of houses, we will need to separate the dataset into **features** and the **target variable**. The **features**, `'RM'`, `'LSTAT'`, and `'PTRATIO'`, give us quantitative information about each data point. The **target variable**, `'MEDV'`, will be the variable we seek to predict. These are stored in `features` and `prices`, respectively.

1.3.1 Implementation: Calculate Statistics

For your very first coding implementation, you will calculate descriptive statistics about the Boston housing prices. Since `numpy` has already been imported for you, use this library to perform the necessary calculations. These statistics will be extremely important later on to analyze various prediction results from the constructed model.

In the code cell below, you will need to implement the following: - Calculate the minimum, maximum, mean, median, and standard deviation of `'MEDV'`, which is stored in `prices`. - Store each calculation in their respective variable.

In [5]:
```
# Minimum price of the data
minimum_price = np.min(prices)

# Maximum price of the data
maximum_price = np.max(prices)

# Mean price of the data
mean_price = np.mean(prices)

# Median price of the data
```

```
        median_price = np.median(prices)

        # Standard deviation of prices of the data
        std_price = np.std(prices)

        # Show the calculated statistics
        print "Statistics for Boston housing dataset:\n"
        print "Minimum price: ${:,.2f}".format(minimum_price)
        print "Maximum price: ${:,.2f}".format(maximum_price)
        print "Mean price: ${:,.2f}".format(mean_price)
        print "Median price ${:,.2f}".format(median_price)
        print "Standard deviation of prices: ${:,.2f}".format(std_price)
```

```
Statistics for Boston housing dataset:

Minimum price: $105,000.00
Maximum price: $1,024,800.00
Mean price: $454,342.94
Median price $438,900.00
Standard deviation of prices: $165,171.13
```

1.3.2 Feature Observation

As a reminder, we are using three features from the Boston housing dataset: `'RM'`, `'LSTAT'`, and `'PTRATIO'`. For each data point (neighborhood): - `'RM'` is the average number of rooms among homes in the neighborhood. - `'LSTAT'` is the percentage of homeowners in the neighborhood considered "lower class" (working poor). - `'PTRATIO'` is the ratio of students to teachers in primary and secondary schools in the neighborhood.

Answer: Based on my opinion, the prices of home depend on lots of condition, like location or the age of the house. For example, if a house only have 3 roome, but locate in a good neighbourhood. this house will be more worth than the house located in a bad neighbourhood, even though have 5 rooms in the house.

For the RM value, if a house have higher value of RM than another house, the house with higher value of RM would be more expensive than another house.

For the LSTAT value, I expect the house with very high LSTAT vaue would be more less worth than the house with very low LSTAT value. Because a house have lower poverty rate will be more expensive than a house have high poverty rate.

the location of house play a very import role in estimating the price. So if a house with higher PTRATIO, which mean there are some schoold located around the neighbourhood, the house in the schoold distirct would have more worth than the houses not located in school distric.

1.4 Developing a Model

In this second section of the project, you will develop the tools and techniques necessary for a model to make a prediction. Being able to make accurate evaluations of each model's performance

through the use of these tools and techniques helps to greatly reinforce the confidence in your predictions.

1.4.1 Implementation: Define a Performance Metric

It is difficult to measure the quality of a given model without quantifying its performance over training and testing. This is typically done using some type of performance metric, whether it is through calculating some type of error, the goodness of fit, or some other useful measurement. For this project, you will be calculating the *coefficient of determination*, R2, to quantify your model's performance. The coefficient of determination for a model is a useful statistic in regression analysis, as it often describes how "good" that model is at making predictions.

The values for R2 range from 0 to 1, which captures the percentage of squared correlation between the predicted and actual values of the **target variable**. A model with an R2 of 0 is no better than a model that always predicts the *mean* of the target variable, whereas a model with an R2 of 1 perfectly predicts the target variable. Any value between 0 and 1 indicates what percentage of the target variable, using this model, can be explained by the **features**. *A model can be given a negative R2 as well, which indicates that the model is* ***arbitrarily worse*** *than one that always predicts the mean of the target variable.*

For the `performance_metric` function in the code cell below, you will need to implement the following: - Use `r2_score` from `sklearn.metrics` to perform a performance calculation between `y_true` and `y_predict`. - Assign the performance score to the `score` variable.

```
In [6]: # Import 'r2_score'
        from sklearn.metrics import r2_score
        def performance_metric(y_true, y_predict):
            """ Calculates and returns the performance score between
                true and predicted values based on the metric chosen. """
            # Calculate the performance score between 'y_true' and 'y_predict'
            score = r2_score(y_true, y_predict)

            # Return the score
            return score
```

1.4.2 Goodness of Fit

Assume that a dataset contains five data points and a model made the following predictions for the target variable:

True Value	Prediction
3.0	2.5
-0.5	0.0
2.0	2.1
7.0	7.8
4.2	5.3

Run the code cell below to use the `performance_metric` function and calculate this model's coefficient of determination.

```
In [7]: # Calculate the performance of this model
        score = performance_metric([3, -0.5, 2, 7, 4.2], [2.5, 0.0, 2.1, 7.8, 5.3])
        print "Model has a coefficient of determination, R^2, of {:.3f}.".format(score)
```

```
Model has a coefficient of determination, R^2, of 0.923.
```

- Would you consider this model to have successfully captured the variation of the target variable?
- Why or why not?

** Hint: ** The R2 score is the proportion of the variance in the dependent variable that is predictable from the independent variable. In other words: * R2 score of 0 means that the dependent variable cannot be predicted from the independent variable. * R2 score of 1 means the dependent variable can be predicted from the independent variable. * R2 score between 0 and 1 indicates the extent to which the dependent variable is predictable. An * R2 score of 0.40 means that 40 percent of the variance in Y is predictable from X.

Answer: yes, this model is the good model, because the mean squared error for the linear regression model should be a lot smaller than the mean squard error for the simple model. I do belive this model can explain the variation of the target variable, taking into account those 5 data points. 92.3% of the variation of the dependent variable can be explained by the model.

1.4.3 Implementation: Shuffle and Split Data

Your next implementation requires that you take the Boston housing dataset and split the data into training and testing subsets. Typically, the data is also shuffled into a random order when creating the training and testing subsets to remove any bias in the ordering of the dataset.

For the code cell below, you will need to implement the following: - Use `train_test_split` from `sklearn.cross_validation` to shuffle and split the `features` and `prices` data into training and testing sets. - Split the data into 80% training and 20% testing. - Set the `random_state` for `train_test_split` to a value of your choice. This ensures results are consistent. - Assign the train and testing splits to `X_train`, `X_test`, `y_train`, and `y_test`.

```
In [8]: from sklearn.cross_validation import train_test_split
        # Import 'train_test_split'

        # Shuffle and split the data into training and testing subsets
        X_train, X_test, y_train, y_test = train_test_split(features, prices, test_size=0.2, ran

        # Success
        print "Training and testing split was successful."
```

```
Training and testing split was successful.
```

1.4.4 Training and Testing

- What is the benefit to splitting a dataset into some ratio of training and testing subsets for a learning algorithm?

Answer: To split the dataset into some ratio of training and testing subset can avoid the overfitting and underfitting. we can use the testing set to pick the best of these models. also the traing set should be over 50% of the dataset. If the ratio of training set to testing set is very small, it will cause the underfitting, othervise it will cause overfitting.

1.5 Analyzing Model Performance

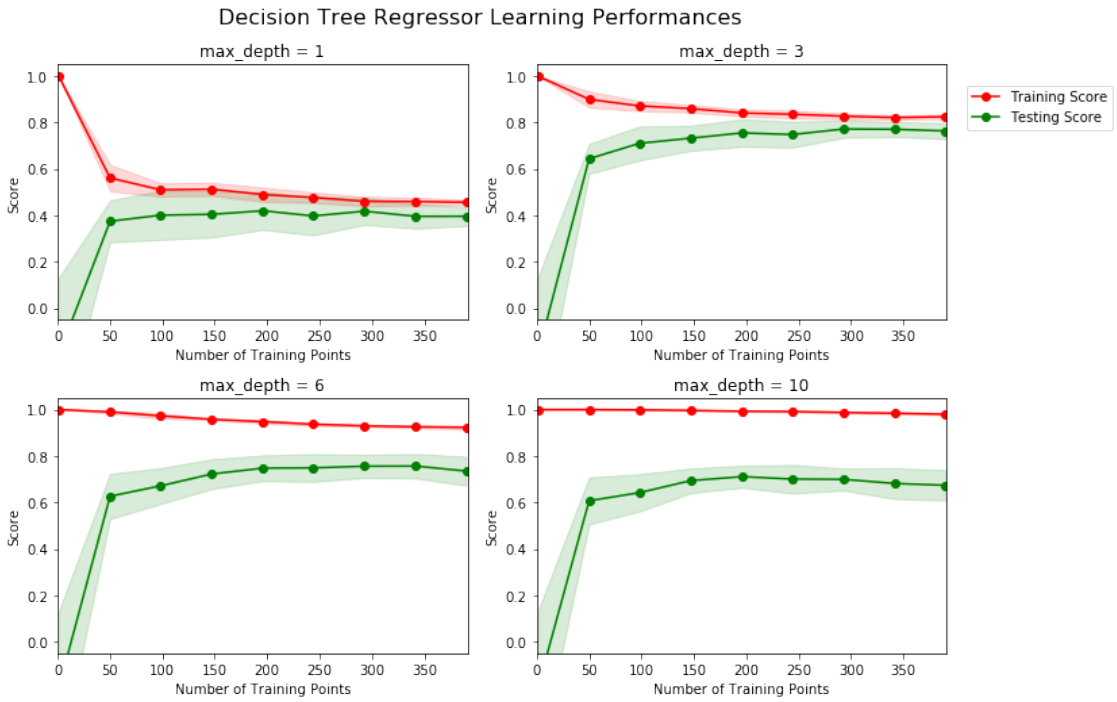
In this third section of the project, you'll take a look at several models' learning and testing performances on various subsets of training data. Additionally, you'll investigate one particular algorithm with an increasing `'max_depth'` parameter on the full training set to observe how model complexity affects performance. Graphing your model's performance based on varying criteria can be beneficial in the analysis process, such as visualizing behavior that may not have been apparent from the results alone.

1.5.1 Learning Curves

The following code cell produces four graphs for a decision tree model with different maximum depths. Each graph visualizes the learning curves of the model for both training and testing as the size of the training set is increased. Note that the shaded region of a learning curve denotes the uncertainty of that curve (measured as the standard deviation). The model is scored on both the training and testing sets using R2, the coefficient of determination.

Run the code cell below and use these graphs to answer the following question.

```
In [9]: # Produce learning curves for varying training set sizes and maximum depths
        vs.ModelLearning(features, prices)
```



1.5.2 Learning the Data

- Choose one of the graphs above and state the maximum depth for the model.
- What happens to the score of the training curve as more training points are added? What about the testing curve?
- Would having more training points benefit the model?

Hint: Are the learning curves converging to particular scores? Generally speaking, the more data you have, the better. But if your training and testing curves are converging with a score above your benchmark threshold, would this be necessary? Think about the pros and cons of adding more training points based on if the training and testing curves are converging.

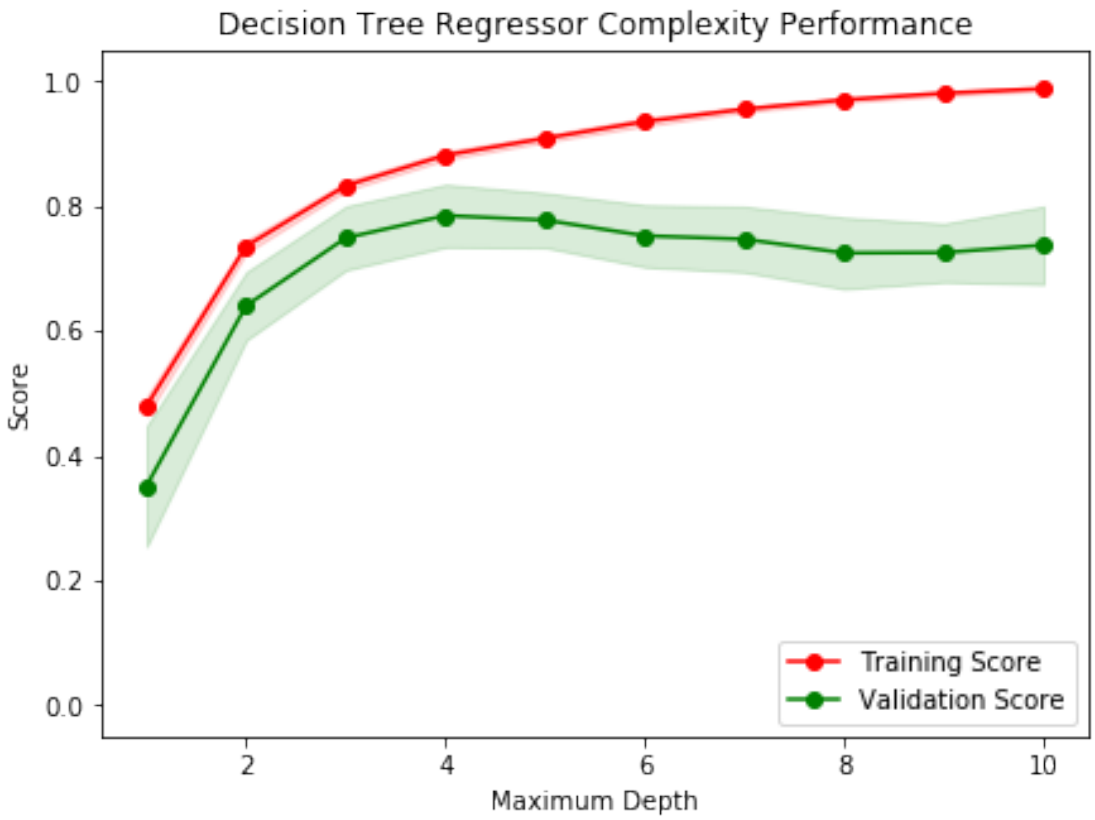
Answer: the max_depth=3 is the best fit model. there are a threshold for the training dataset. Testing dataset here is used to validate the model, not to evaluate the final performance. if the training point more than 50, the socre of training and testing will be hold on the line, and they will be converging to the particular scores.high variance is caused by an overfitting model and can actually be reduced by adding more that.

1.5.3 Complexity Curves

The following code cell produces a graph for a decision tree model that has been trained and validated on the training data using different maximum depths. The graph produces two complexity curves — one for training and one for validation. Similar to the **learning curves**, the shaded regions of both the complexity curves denote the uncertainty in those curves, and the model is scored on both the training and validation sets using the `performance_metric` function.

** Run the code cell below and use this graph to answer the following two questions Q5 and Q6. **

```
In [10]: vs.ModelComplexity(X_train, y_train)
```



1.5.4 Bias-Variance Tradeoff

- When the model is trained with a maximum depth of 1, does the model suffer from high bias or from high variance?
- How about when the model is trained with a maximum depth of 10? What visual cues in the graph justify your conclusions?

Hint: High bias is a sign of underfitting(model is not complex enough to pick up the nuances in the data) and high variance is a sign of overfitting(model is by-hearting the data and cannot generalize well). Think about which model(depth 1 or 10) aligns with which part of the tradeoff.

Answer: the model with a maximum depth of 1 does suffer the high bias,called underfitting. According to the above complexity curve graph, with the maximum depth of 1, the training score and the validation score are both very low and the training result is not good enough, which may cause the training model cannot fit the dataset nicely. the model with a maximum depth of 10 does suffer the high varience, called overfitting. with the higher maximum depth, the validation score is lower but the training score is way higher, which indicate the overfitting.

1.5.5 Best-Guess Optimal Model

- Which maximum depth do you think results in a model that best generalizes to unseen data?
- What intuition lead you to this answer?

** Hint: ** Look at the graph above Question 5 and see where the validation scores lie for the various depths that have been assigned to the model. Does it get better with increased depth? At what point do we get our best validation score without overcomplicating our model? And remember, Occams Razor states "Among competing hypotheses, the one with the fewest assumptions should be selected."

Answer: the model with the maximum depth 4 is the best generalized to the unseen data. Look at the learing curve above the question 5, when the maximum depth is 4, the model have the highest cross validation socre and the high training socre.

1.6 Evaluating Model Performance

In this final section of the project, you will construct a model and make a prediction on the client's feature set using an optimized model from `fit_model`.

1.6.1 Grid Search

- What is the grid search technique?
- How it can be applied to optimize a learning algorithm?

** Hint: ** When explaining the Grid Search technique, be sure to touch upon why it is used, what the 'grid' entails and what the end goal of this method is. To solidify your answer, you can also give an example of a parameter in a model that can be optimized using this approach.

Answer: the grid search is the technique of making a table to store all the posibilities and pick the best one. for example fot the support vector machines algorithm, the kernel can be linear or polynomial, and the degree can be 1,2,3,4. so we draw a table with the columns and row. columns contain different kernal and the row have different velue of the gamma. and then use the training set to train the bunch of linear and polynomial models with different value of gamma, then use the cross validation set to calculate the F1 score on all these models and then pick the one models with the higest F1 socre. and finally use the testing set to test the model is best fit.

1.6.2 Cross-Validation

- What is the k-fold cross-validation training technique?
- What benefit does this technique provide for grid search when optimizing a model?

Hint: When explaining the k-fold cross validation technique, be sure to touch upon what 'k' is, how the dataset is split into different parts for training and testing and the number of times it is run based on the 'k' value.

When thinking about how k-fold cross validation helps grid search, think about the main drawbacks of grid search which are hinged upon **using a particular subset of data for training or testing** and how k-fold cv could help alleviate that. You can refer to the docs for your answer.

Answer: K-fold validation break out the data into K buckets and then train model K times, each time using a different buckets as the testing set, finally get the average result for the final result. the grid search is based upon the particular subset of training and testing data set, however the K-fold validation can eliminate this drawback by divide the data set into lots of training sets and testing sets. each one can be testing set, and there are K-1 sets are the training set.So each one can be testing set or training set.

1.6.3 Implementation: Fitting a Model

Your final implementation requires that you bring everything together and train a model using the **decision tree algorithm**. To ensure that you are producing an optimized model, you will train the model using the grid search technique to optimize the `'max_depth'` parameter for the decision tree. The `'max_depth'` parameter can be thought of as how many questions the decision tree algorithm is allowed to ask about the data before making a prediction. Decision trees are part of a class of algorithms called *supervised learning algorithms*.

In addition, you will find your implementation is using `ShuffleSplit()` for an alternative form of cross-validation (see the `'cv_sets'` variable). While it is not the K-Fold cross-validation technique you describe in **Question 8**, this type of cross-validation technique is just as useful!. The `ShuffleSplit()` implementation below will create 10 (`'n_splits'`) shuffled sets, and for each shuffle, 20% (`'test_size'`) of the data will be used as the *validation set*. While you're working on your implementation, think about the contrasts and similarities it has to the K-fold cross-validation technique.

Please note that ShuffleSplit has different parameters in scikit-learn versions 0.17 and 0.18. For the `fit_model` function in the code cell below, you will need to implement the following: - Use `DecisionTreeRegressor` from `sklearn.tree` to create a decision tree regressor object. - Assign this object to the `'regressor'` variable. - Create a dictionary for `'max_depth'` with the values from 1 to 10, and assign this to the `'params'` variable. - Use `make_scorer` from `sklearn.metrics` to create a scoring function object. - Pass the `performance_metric` function as a parameter to the object. - Assign this scoring function to the `'scoring_fnc'` variable. - Use `GridSearchCV` from `sklearn.grid_search` to create a grid search object. - Pass the variables `'regressor'`, `'params'`, `'scoring_fnc'`, and `'cv_sets'` as parameters to the object. - Assign the `GridSearchCV` object to the `'grid'` variable.

```
In [12]: # TODO: Import 'make_scorer', 'DecisionTreeRegressor', and 'GridSearchCV'
         from sklearn.tree import DecisionTreeRegressor
         from sklearn.metrics import make_scorer
         from sklearn.grid_search import GridSearchCV

         def fit_model(X, y):
             """ Performs grid search over the 'max_depth' parameter for a
                 decision tree regressor trained on the input data [X, y]. """

             # Create cross-validation sets from the training data
             # sklearn version 0.18: ShuffleSplit(n_splits=10, test_size=0.1, train_size=None, r
             # sklearn versiin 0.17: ShuffleSplit(n, n_iter=10, test_size=0.1, train_size=None,
             cv_sets = ShuffleSplit(X.shape[0], n_iter = 10, test_size = 0.20, random_state = 0)

             # TODO: Create a decision tree regressor object
             regressor = DecisionTreeRegressor()

             # TODO: Create a dictionary for the parameter 'max_depth' with a range from 1 to 10
             params = {'max_depth':[1,2,3,4,5,6,7,8,9,10]}

             # TODO: Transform 'performance_metric' into a scoring function using 'make_scorer'
             scoring_fnc = make_scorer(performance_metric)
```

```
    # TODO: Create the grid search cv object --> GridSearchCV()
    # Make sure to include the right parameters in the object:
    # (estimator, param_grid, scoring, cv) which have values 'regressor', 'params', 'sc
    grid = GridSearchCV(regressor, params, scoring=scoring_fnc, cv=cv_sets)

    # Fit the grid search object to the data to compute the optimal model
    grid = grid.fit(X, y)

    # Return the optimal model after fitting the data
    return grid.best_estimator_
```

1.6.4 Making Predictions

Once a model has been trained on a given set of data, it can now be used to make predictions on new sets of input data. In the case of a *decision tree regressor*, the model has learned *what the best questions to ask about the input data are*, and can respond with a prediction for the **target variable**. You can use these predictions to gain information about data where the value of the target variable is unknown — such as data the model was not trained on.

1.6.5 Optimal Model

- What maximum depth does the optimal model have? How does this result compare to your guess in **Question 6**?

Run the code block below to fit the decision tree regressor to the training data and produce an optimal model.

```
In [13]: # Fit the training data to the model using grid search
         reg = fit_model(X_train, y_train)

         # Produce the value for 'max_depth'
         print "Parameter 'max_depth' is {} for the optimal model.".format(reg.get_params()['max
```

```
Parameter 'max_depth' is 4 for the optimal model.
```

** Hint: ** The answer comes from the output of the code snipped above.

Answer: previously I guess the maximum depth is 4, and the result is same with the result I guessed.

1.6.6 Predicting Selling Prices

Imagine that you were a real estate agent in the Boston area looking to use this model to help price homes owned by your clients that they wish to sell. You have collected the following information from three of your clients:

Feature	Client 1	Client 2	Client 3
Total number of rooms in home	5 rooms	4 rooms	8 rooms

Feature	Client 1	Client 2	Client 3
Neighborhood poverty level (as %)	17%	32%	3%
Student-teacher ratio of nearby schools	15-to-1	22-to-1	12-to-1

- What price would you recommend each client sell his/her home at?
- Do these prices seem reasonable given the values for the respective features?

Hint: Use the statistics you calculated in the **Data Exploration** section to help justify your response. Of the three clients, client 3 has has the biggest house, in the best public school neighborhood with the lowest poverty level; while client 2 has the smallest house, in a neighborhood with a relatively high poverty rate and not the best public schools.

Run the code block below to have your optimized model make predictions for each client's home.

```
In [14]: # Produce a matrix for client data
         client_data = [[5, 17, 15], # Client 1
                        [4, 32, 22], # Client 2
                        [8, 3, 12]]  # Client 3

         # Show predictions
         for i, price in enumerate(reg.predict(client_data)):
             print "Predicted selling price for Client {}'s home: ${:,.2f}".format(i+1, price)
```

```
Predicted selling price for Client 1's home: $411,096.00
Predicted selling price for Client 2's home: $219,961.54
Predicted selling price for Client 3's home: $955,500.00
```

Answer: based on the above prediction, the client 3's house are more expensive than the others two houses, because the client 3's house is the biggest house, and have the public school near the house, also with the lowest poverty level. However, the client 2's house have the lowest price because the smallest house, and the relatively high poverty rate near the house neighbourhood.So the client 3's house has the rich owner, client 2's house has the poor owner and the client 1's house has the average owner. Based on these three features, the prediction prices for these three houses are resonable.

1.6.7 Sensitivity

An optimal model is not necessarily a robust model. Sometimes, a model is either too complex or too simple to sufficiently generalize to new data. Sometimes, a model could use a learning algorithm that is not appropriate for the structure of the data given. Other times, the data itself could be too noisy or contain too few samples to allow a model to adequately capture the target variable — i.e., the model is underfitted.

Run the code cell below to run the `fit_model` function ten times with different training and testing sets to see how the prediction for a specific client changes with respect to the data it's trained on.

```
In [15]: vs.PredictTrials(features, prices, fit_model, client_data)
```

```
Trial 1: $391,183.33
Trial 2: $419,700.00
Trial 3: $415,800.00
Trial 4: $420,622.22
Trial 5: $413,334.78
Trial 6: $411,931.58
Trial 7: $399,663.16
Trial 8: $407,232.00
Trial 9: $351,577.61
Trial 10: $413,700.00

Range in prices: $69,044.61
```