Machine Learning Engineer Nanodegree

Model Evaluation & Validation

Project 1: Predicting Boston Housing Prices

Welcome to the first project of the Machine Learning Engineer Nanodegree! In this notebook, some template code has already been written. You will need to implement additional functionality to successfully answer all of the questions for this project. Unless it is requested, do not modify any of the code that has already been included. In this template code, there are four sections which you must complete to successfully produce a prediction with your model. Each section where you will write code is preceded by a **STEP X** header with comments describing what must be done. Please read the instructions carefully!

In addition to implementing code, there will be questions that you must answer that relate to the project and your implementation. Each section where you will answer a question is preceded by a **QUESTION X** header. Be sure that you have carefully read each question and provide thorough answers in the text boxes that begin with "**Answer:**". Your project submission will be evaluated based on your answers to each of the questions.

A description of the dataset can be found [here (https://archive.ics.uci.edu/ml/datasets/Housing)](https://archive.ics.uci.edu/ml/datasets/Housing), which is provided by the **UCI Machine Learning Repository**.

Getting Started

To familiarize yourself with an iPython Notebook, **try double clicking on this cell**. You will notice that the text changes so that all the formatting is removed. This allows you to make edits to the block of text you see here. This block of text (and mostly anything that's not code) is written using [Markdown (http://daringfireball.net/projects/markdown/syntax)](http://daringfireball.net/projects/markdown/syntax), which is a way to format text using headers, links, italics, and many other options! Whether you're editing a Markdown text block or a code block (like the one below), you can use the keyboard shortcut **Shift + Enter** or **Shift + Return** to execute the code or text block. In this case, it will show the formatted text.

Let's start by setting up some code we will need to get the rest of the project up and running. Use the keyboard shortcut mentioned above on the following code block to execute it. Alternatively, depending on your iPython Notebook program, you can press the **Play** button in the hotbar. You'll know the code block executes successfully if the message "*Boston Housing dataset loaded successfully!*" is printed.

```
In [1]:  # 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 no
         tebook)
         %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, 332.09, 12.13]]

         # 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!"
```

```
/Users/aleu/anaconda2/lib/python2.7/site-packages/matplotlib/font_
manager.py:273: UserWarning: Matplotlib is building the font cache
using fc-list. This may take a moment.
  warnings.warn('Matplotlib is building the font cache using fc-li
st. This may take a moment.')

Boston Housing dataset loaded successfully!
```

Statistical Analysis and Data Exploration

In this first section of the project, you will quickly investigate a few basic statistics about the dataset you are working with. In addition, you'll look at the client's feature set in CLIENT_FEATURES and see how this particular sample relates to the features of the dataset. Familiarizing yourself with the data through an explorative process is a fundamental practice to help you better understand your results.

Step 1

In the code block below, use the imported `numpy` library to calculate the requested statistics. You will need to replace each `None` you find with the appropriate `numpy` coding for the proper statistic to be printed. Be sure to execute the code block each time to test if your implementation is working successfully. The print statements will show the statistics you calculate!

```python
In [2]:  # Number of houses in the dataset
         total_houses = len(housing_features)

         # Number of features in the dataset
         total_features = len(housing_features[0])

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

         # Maximum housing value in the dataset
         maximum_price = np.amax(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
```

Question 1

As a reminder, you can view a description of the Boston Housing dataset here (https://archive.ics.uci.edu/ml/datasets/Housing), 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.

Remember, you can **double click the text box below** to add your answer!

Answer:

- CRIM measurse the crime rate per capital by the town.
- TAX measure the full property tax rate of the houses.
- RAD measures accessbility of the radial highway.

Question 2

Using your client's feature set `CLIENT_FEATURES`, which values correspond with the features you've chosen above?
Hint: Run the code block below to see the client's data.

```
In [3]: print CLIENT_FEATURES

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

Answer:

- CRIM : 11.95
- TAX : 680.0
- RAD : 24

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.

Step 2

In the code block below, you will need to implement code so that the `shuffle_split_data` function does the following:

- Randomly shuffle the input data `X` and target labels (housing values) `y`.
- Split the data into training and testing subsets, holding 30% of the data for testing.

If you use any functions not already acessible from the imported libraries above, remember to include your import statement below as well!
Ensure that you have executed the code block once you are done. You'll know the `shuffle_split_data` function is working if the statement "*Successfully shuffled and split the data!*" is printed.

```
In [4]:  # Put any import statements you need for this code block here
         from sklearn import cross_validation
         def shuffle_split_data(X, y):
             """ Shuffles and splits data into 70% training and 30% testing
         subsets,
                 then returns the training and testing subsets. """
             X_train, X_test, y_train, y_test = cross_validation.train_test_
         split(X, y, test_size =0.30, random_state=0)
             # Shuffle and split the data
             # Return the training and testing data subsets
             return X_train, y_train, X_test, y_test


         # Test shuffle_split_data
         try:
             X_train, y_train, X_test, y_test = shuffle_split_data(housing_f
         eatures, housing_prices)
             print "Successfully shuffled and split the data!"
         except:
             print "Something went wrong with shuffling and splitting the da
         ta."
```

Successfully shuffled and split the data!

Question 3

Why do we split the data into training and testing subsets for our model?

Answer:

- We split data into training and testing to have a group of training set to create regression ,and use the testing set to verify the result. By splitting the training set and testing set, we are able to verify our regression model's accuracy through our actual data.
- After creating a regression models with the training set, the testing set will be inserted into the model to output the prediction result of the prices. We then check the prediction result against it's actual result on the market to verify how accurate our regression model is.
- Using a testing subset to test the result over training subset also helps to protect the regression from over-fitting.

Step 3

In the code block below, you will need to implement code so that the `performance_metric` function does the following:

- Perform a total error calculation between the true values of the y labels `y_true` and the predicted values of the y labels `y_predict`.

You will need to first choose an appropriate performance metric for this problem. See the sklearn metrics documentation (http://scikit-learn.org/stable/modules/classes.html#sklearn-metrics-metrics) to view a list of available metric functions. **Hint:** Look at the question below to see a list of the metrics that were covered in the supporting course for this project.

Once you have determined which metric you will use, remember to include the necessary import statement as well!
Ensure that you have executed the code block once you are done. You'll know the `performance_metric` function is working if the statement "*Successfully performed a metric calculation!*" is printed.

```
In [5]: # Put any import statements you need for this code block here
        from sklearn.metrics import mean_squared_error
        def performance_metric(y_true, y_predict):
            """ Calculates and returns the total error between true and pre
        dicted values
                based on a performance metric chosen by the student. """

            error = mean_squared_error(y_true, y_predict)
            return error



        # Test performance_metric
        try:
            total_error = performance_metric(y_train, y_train)
            print "Successfully performed a metric calculation!"
        except:
            print "Something went wrong with performing a metric calculatio
        n."
```

```
Successfully performed a metric calculation!
```

Question 4

Which performance metric below did you find was most appropriate for predicting housing prices and analyzing the total error. Why?

- *Accuracy*
- *Precision*
- *Recall*
- *F1 Score*
- *Mean Squared Error (MSE)*
- *Mean Absolute Error (MAE)*

Answer: Mean squared error, because we wanted to find out the error of our model that is deviated away from the mean. Mean squred is used as it would be more impactful than the mean absolute error. With higher penality in the result comparing to mean absolute square, we get have a clearer visibility of the error result.

Mean aboslute error also provides the same result, but not as impactful in the length of deviation tahn Mean squared error.

Accuracy, precision, and recall are not used because we are not trying to find out how accurate or together the sample sets are, we are trying to determine the number of errors. F1 score is also used to test the accuracy of the data, which deviate from what we want to as inference.

Step 4 (Final Step)

In the code block below, you will need to implement code so that the `fit_model` function does the following:

- Create a scoring function using the same performance metric as in **Step 2**. See the sklearn make_scorer documentation (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.make_scorer.html).
- Build a GridSearchCV object using `regressor`, `parameters`, and `scoring_function`. See the sklearn documentation on GridSearchCV (http://scikit-learn.org/stable/modules/generated/sklearn.grid_search.GridSearchCV.html).

When building the scoring function and GridSearchCV object, *be sure that you read the parameters documentation thoroughly.* It is not always the case that a default parameter for a function is the appropriate setting for the problem you are working on.

Since you are using `sklearn` functions, remember to include the necessary import statements below as well!
Ensure that you have executed the code block once you are done. You'll know the `fit_model` function is working if the statement "*Successfully fit a model to the data!*" is printed.

In [6]:
```python
# Put any import statements you need for this code block
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import fbeta_score, make_scorer
from sklearn.grid_search import GridSearchCV
def fit_model(X, y):
    """ Tunes a decision tree regressor model using GridSearchCV on the input data X
        and target labels y and returns this optimal model. """

    # Create a decision tree regressor object
    regressor = DecisionTreeRegressor()
    # Set up the parameters we wish to tune
    parameters = {'max_depth':(1,2,3,4,5,6,7,8,9,10)}

    # Make an appropriate scoring function
    mse_scorer = make_scorer(performance_metric,  # Use our performance metric, which is just mean squared error(mse)
                             greater_is_better=False)

    # Make the GridSearchCV object
    reg = GridSearchCV(estimator=regressor,
                       param_grid=parameters,
                       scoring=mse_scorer)

    # Fit the learner to the data to obtain the optimal model with tuned parameters
    reg.fit(X, y)

    # Return the optimal model
    return reg.best_estimator_


# Test fit_model on entire dataset
try:
    reg = fit_model(housing_features, housing_prices)
    print "Successfully fit a model!"
except:
    print "Something went wrong with fitting a model."
```

```
Successfully fit a model!
```

Question 5

What is the grid search algorithm and when is it applicable?

Answer: Grid serach is using multiple of different types of parameter tunes, and cross validations to find the best performance for the fitted model. It is often used when the user woud like to learn the pattern of the data set. To optimize a machine learning model parameters, grid search can be used to evaluate user estimation to show an optimal number among them all.

How does grid search works?

- With a given list of parameters we believe to be the potential for the best model, the algorithm switchses different values of gamma and C to find out the best output. It then evaluate the result with cross validation. The output would be the best parameter.

Question 6

What is cross-validation, and how is it performed on a model? Why would cross-validation be helpful when using grid search?

Answer: Cross validation is checking the data set by patch by patch to different sets of test. Switching over different sets of testing data and training data among the whoel data set, and provide the best model fit at the end. Grid search and cross-validation works hand in hand, as grid search would be able to shuffle around the cross-validation matches to find the best suitable fit for the evaluated model.

Checkpoint!

You have now successfully completed your last code implementation section. Pat yourself on the back! All of your functions written above will be executed in the remaining sections below, and questions will be asked about various results for you to analyze. To prepare the **Analysis** and **Prediction** sections, you will need to intialize the two functions below. Remember, there's no need to implement any more code, so sit back and execute the code blocks! Some code comments are provided if you find yourself interested in the functionality.

In [7]:
```python
def learning_curves(X_train, y_train, X_test, y_test):
    """ Calculates the performance of several models with varying s
izes of training data.
        The learning and testing error rates for each model are the
n plotted. """

    print "Creating learning curve graphs for max_depths of 1, 3,
6, and 10. . ."

    # Create the figure window
    fig = pl.figure(figsize=(10,8))

    # We will vary the training set size so that we have 50 differe
nt sizes
    sizes = np.rint(np.linspace(1, len(X_train), 50)).astype(int)
    train_err = np.zeros(len(sizes))
    test_err = np.zeros(len(sizes))

    # Create four different models based on max_depth
    for k, depth in enumerate([1,3,6,10]):

        for i, s in enumerate(sizes):

            # Setup a decision tree regressor so that it learns a t
ree with max_depth = depth
            regressor = DecisionTreeRegressor(max_depth = depth)

            # Fit the learner to the training data
            regressor.fit(X_train[:s], y_train[:s])

            # Find the performance on the training set
            train_err[i] = performance_metric(y_train[:s], regresso
r.predict(X_train[:s]))

            # Find the performance on the testing set
            test_err[i] = performance_metric(y_test, regressor.pred
ict(X_test))

        # Subplot the learning curve graph
        ax = fig.add_subplot(2, 2, k+1)
        ax.plot(sizes, test_err, lw = 2, label = 'Testing Error')
        ax.plot(sizes, train_err, lw = 2, label = 'Training Error')
        ax.legend()
        ax.set_title('max_depth = %s'%(depth))
        ax.set_xlabel('Number of Data Points in Training Set')
        ax.set_ylabel('Total Error')
        ax.set_xlim([0, len(X_train)])

    # Visual aesthetics
    fig.suptitle('Decision Tree Regressor Learning Performances', f
ontsize=18, y=1.03)
    fig.tight_layout()
    fig.show()
```

```python
In [8]: def model_complexity(X_train, y_train, X_test, y_test):
    """ Calculates the performance of the model as model complexity
increases.
        The learning and testing errors rates are then plotted. """

    print "Creating a model complexity graph. . . "

    # We will vary the max_depth of a decision tree model from 1 to
14
    max_depth = np.arange(1, 14)
    train_err = np.zeros(len(max_depth))
    test_err = np.zeros(len(max_depth))

    for i, d in enumerate(max_depth):
        # Setup a Decision Tree Regressor so that it learns a tree
with depth d
        regressor = DecisionTreeRegressor(max_depth = d)

        # Fit the learner to the training data
        regressor.fit(X_train, y_train)

        # Find the performance on the training set
        train_err[i] = performance_metric(y_train, regressor.predic
t(X_train))

        # Find the performance on the testing set
        test_err[i] = performance_metric(y_test, regressor.predict
(X_test))

    # Plot the model complexity graph
    pl.figure(figsize=(7, 5))
    pl.title('Decision Tree Regressor Complexity Performance')
    pl.plot(max_depth, test_err, lw=2, label = 'Testing Error')
    pl.plot(max_depth, train_err, lw=2, label = 'Training Error')
    pl.legend()
    pl.xlabel('Maximum Depth')
    pl.ylabel('Total Error')
    pl.show()
```

Analyzing Model Performance

In this third section of the project, you'll take a look at several models' learning and testing error rates 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 learning and testing errors. 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.

In [9]: `learning_curves(X_train, y_train, X_test, y_test)`
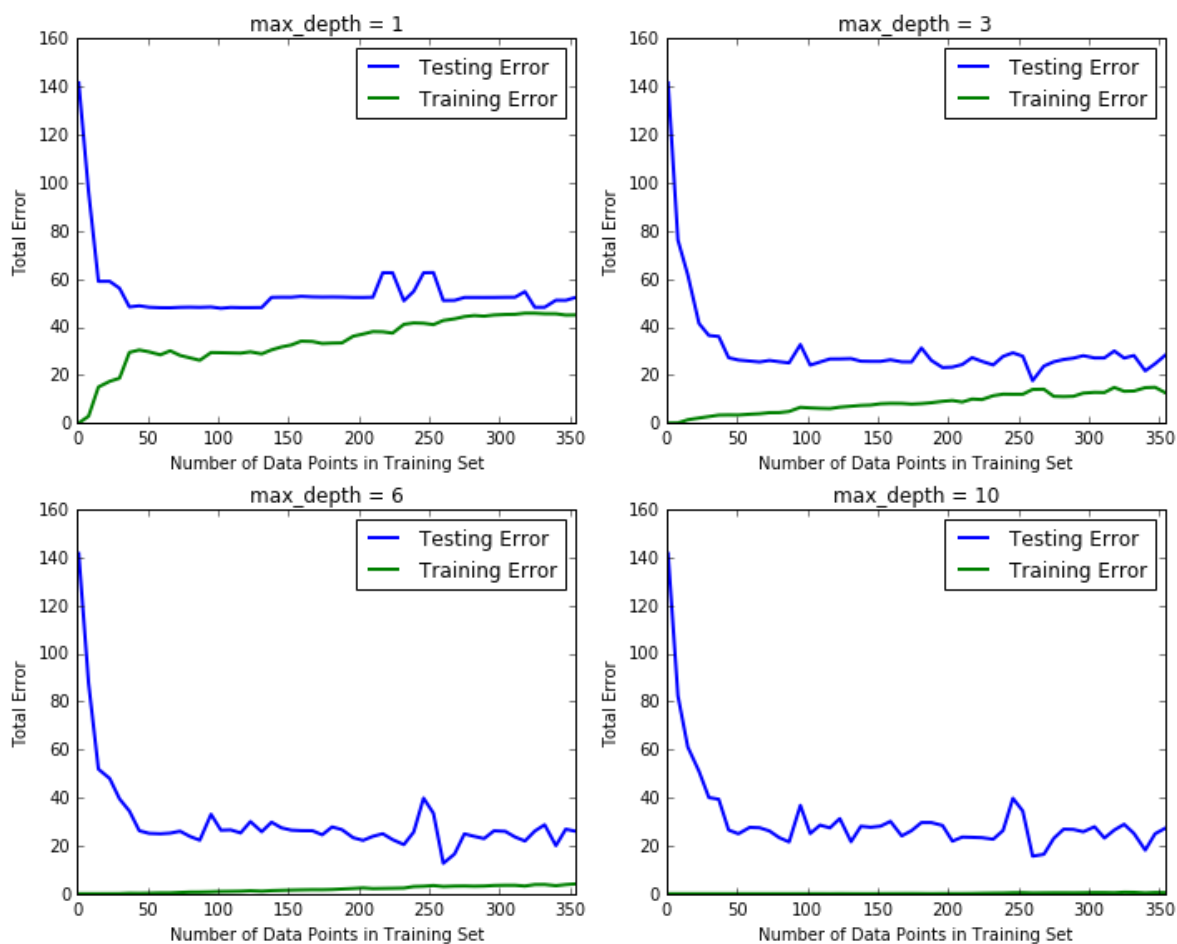
```
Creating learning curve graphs for max_depths of 1, 3, 6, and 10.
. .

/Users/aleu/anaconda2/lib/python2.7/site-packages/matplotlib/figur
e.py:397: UserWarning: matplotlib is currently using a non-GUI bac
kend, so cannot show the figure
  "matplotlib is currently using a non-GUI backend, "
```


Decision Tree Regressor Learning Performances

Question 7

Choose one of the learning curve graphs that are created above. What is the max depth for the chosen model? As the size of the training set increases, what happens to the training error? What happens to the testing error?

Answer: I have chosen max_depth = 1. As the size of the traning set increases, the training error increases. The testing error is consistent as the trainign set increases. Also looking at the max_depth 1, the fit is too generalizes to find out any trend in the data. Also by increasing the number of training did not improve the testing set determines the in-effectiveness of the model.
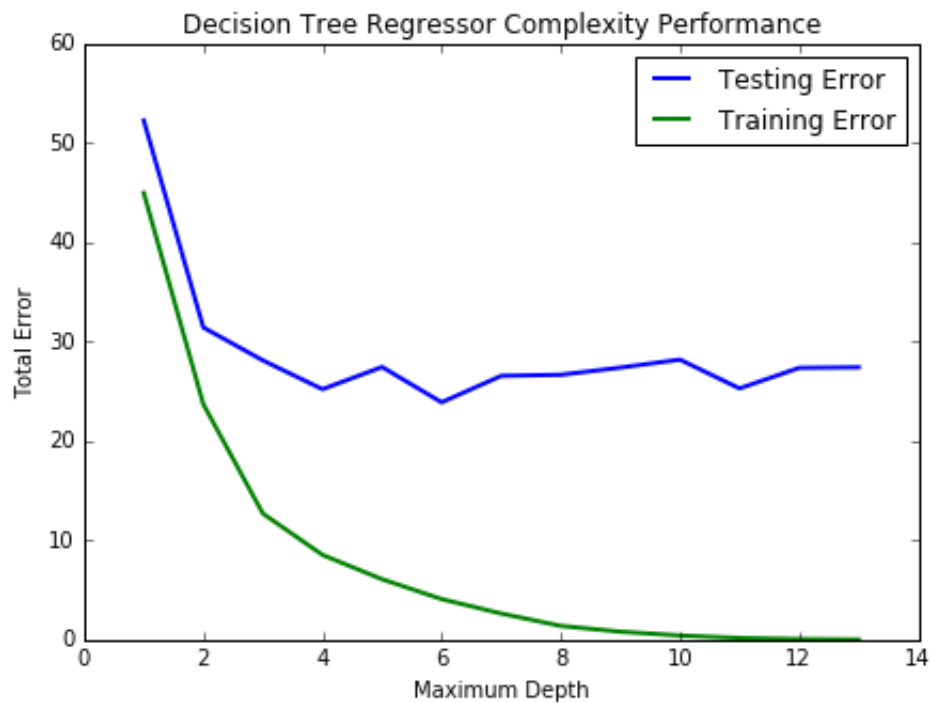
Question 8

Look at the learning curve graphs for the model with a max depth of 1 and a max depth of 10. When the model is using the full training set, does it suffer from high bias or high variance when the max depth is 1? What about when the max depth is 10?

Answer: When the max_depth is 1, it is suffering from a high variance since the error of the training set has been increases. When the max_depth is 10, it is suffering from a high bias result, as it is impossible for the training set to not have any error at all.

```
In [10]:  model_complexity(X_train, y_train, X_test, y_test)
```

Creating a model complexity graph. . .

Question 9

From the model complexity graph above, describe the training and testing errors as the max depth increases. Based on your interpretation of the graph, which max depth results in a model that best generalizes the dataset? Why?

Answer: As the max depth increases, the training error decreases and the testing error decreses until it hits the max depth of 6. This shows the max depth of 6 is best to generalizes the data set.

Based on this graph, I believe the bet max depth result would be 6, becaues the training set error is on the rate of decreasing with the testing error reaching to its most minimum part of the graph. The bump beteween 8 and 10 suggests by having too much in depth of the maximum is creaing some bias on the testing result.

Model Prediction

In this final section of the project, you will make a prediction on the client's feature set using an optimized model from `fit_model`. When applying grid search along with cross-validation to optimize your model, it would typically be performed and validated on a training set and subsequently evaluated on a **dedicated test set**. In this project, the optimization below is performed on the *entire dataset* (as opposed to the training set you made above) due to the many outliers in the data. Using the entire dataset for training provides for a less volatile prediction at the expense of not testing your model's performance.

To answer the following questions, it is recommended that you run the code blocks several times and use the median or mean value of the results.

Question 10

Using grid search on the entire dataset, what is the optimal `max_depth` parameter for your model? How does this result compare to your intial intuition?
Hint: Run the code block below to see the max depth produced by your optimized model.

```
In [11]:  print "Final model has an optimal max_depth parameter of", reg.get_
          params()['max_depth']

          Final model has an optimal max_depth parameter of 4
```

Answer:

The optimal max depth of my parameter is 4, which is not part of any of my predictions. The graphs seems to be leaning closer to the 5 than 4, which shows it could be caused by the random nature of computing and data splitting.

Question 11

With your parameter-tuned model, what is the best selling price for your client's home? How does this selling price compare to the basic statistics you calculated on the dataset?

Hint: Run the code block below to have your parameter-tuned model make a prediction on the client's home.

```
In [12]: sale_price = reg.predict(CLIENT_FEATURES)
         print "Predicted value of client's home: {0:.3f}".format(sale_price
         [0])
```

```
Predicted value of client's home: 21.630
```

Answer: The best seeling price is 21.630, which is higher than median, but slighter lower than mean. This selling price is reaching the norm of the the statistics with the variance of 9.188.

Question 12 (Final Question):   ¶

In a few sentences, discuss whether you would use this model or not to predict the selling price of future clients' homes in the Greater Boston area.

Answer:

I would use this model to predict the selling price of future clients' in the Great Boston area, as it gives more of a precise number than the preliminary statistical result. It also verifies the initial statistical variance. In addition, the grid search enables us to test the model with the entire data set. This strengthens our confidence to be a stronger portryal than the preliminary result.

```
In [ ]:
```