boston_housing

June 2, 2016

1 Machine Learning Engineer Nanodegree

1.1 Model Evaluation & Validation

1.2 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, which is provided by the **UCI Machine Learning Repository**.

2 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, 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 [57]: # 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, 33

# 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!"
```

```
Boston Housing dataset loaded successfully!
```

3 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.

3.1 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!

In [58]:
```
# Number of houses in the dataset
total_houses = len(housing_prices)

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

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

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

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

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

# Standard deviation of housing values of the dataset
std_dev = numpy.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
```

3.2 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.

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

Answer: The three features that are most significant are CRIM, RM and DIS. CRIM measures per capita crime rate per town, RM measures average number of rooms per dwelling, and DIS measures weighted distance to 5 Boston employment centers. CRIM could demonstrate the socioeconomics of the region and would inadvertently signifiy housing values. Number of rooms (RM) could be a loose measure of how large the house is, which correlates with housing value. Distance to employment centers (DIS) could increase value of the house for occupants.

3.3 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 [59]: 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: The corresponding client values for CRIM, RM and DIS in order are: 11.95, 5.609, and 1.385

4 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.

4.1 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 [69]: from numpy import array
         from sklearn import cross_validation
         import numpy as np
         total_houses = len(housing_prices)
         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 = []
             y_train=[]
             X_test=[]
             y_test=[]
             rs = cross_validation.ShuffleSplit(total_houses, n_iter=1,test_size=.3, random_state=0)
             for train, test in rs:

                 for val in train:
                     X_train.append(X[val])
                     y_train.append(y[val])

                 for val in test:
                     X_test.append(X[val])
                     y_test.append(y[val])
             # Return the training and testing data subsets
             np_X_train = array(X_train)
             np_y_train = array(y_train)
             np_X_test = array(X_test)
             np_y_test = array(y_test)
             return np_X_train, np_y_train, np_X_test, np_y_test
         # Test shuffle_split_data
         try:
             X_train, y_train, X_test, y_test = shuffle_split_data(housing_features, housing_prices)
             print "Successfully shuffled and split the data!"
         except:
             print "Something went wrong with shuffling and splitting the data."
```

```
Successfully shuffled and split the data!
```

4.2 Question 3

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

Answer: Splitting the data into training and testing subsets will allow the model to be tested for unseen data. Furthermore we can make evaluations on if the model {overfits or has high variance} or {if it is underfitted or has high bias}.

4.3 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 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 [71]: from sklearn.metrics import mean_squared_error
         # Put any import statements you need for this code block here

         def performance_metric(y_true, y_predict):
             """ Calculates and returns the total error between true and predicted 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! %.3f" % total_error
         except:
             print "Something went wrong with performing a metric calculation."
```

```
Successfully performed a metric calculation! 0.000
```

4.4 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:

4.5 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. - Build a GridSearchCV object using `regressor`, `parameters`, and `scoring_function`. See the sklearn documentation on GridSearchCV.

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 [72]: from sklearn import svm, grid_search, datasets
         from sklearn.tree import DecisionTreeRegressor
         from sklearn import datasets
         from sklearn.metrics import make_scorer
         from sklearn.metrics import mean_squared_error
         import numpy
         city_data = datasets.load_boston()

         housing_prices = city_data.target
         housing_features = city_data.data

         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
        scoring_function=make_scorer(mean_squared_error,greater_is_better=False)


        # Make the GridSearchCV object
        reg = grid_search.GridSearchCV(regressor, parameters, scoring_function)

        # 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!
```

4.6 Question 5

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

Answer: The grid search algorithm provides the machine learning engineer a exhaustive search over specified parameters, an estimator and scoring function. The return would be errors for each parameter input.

4.7 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 a method to reduce training and testing error. In k-fold cross validation, the dataset is spilt into k parts. Each part has a turn in being the test set, and all other parts are the training set. The resulting errors for all k parts are averaged together.

5 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 [74]: import numpy as np
```

```
    def learning_curves(X_train, y_train, X_test, y_test):
        """ Calculates the performance of several models with varying sizes of training data.
            The learning and testing error rates for each model are then 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 different 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 tree 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], regressor.predict(X_train[:s]))

                # Find the performance on the testing set
                test_err[i] = performance_metric(y_test, regressor.predict(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', fontsize=18, y=1.03)
        fig.tight_layout()
        fig.show()

In [75]: 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.predict(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()
```

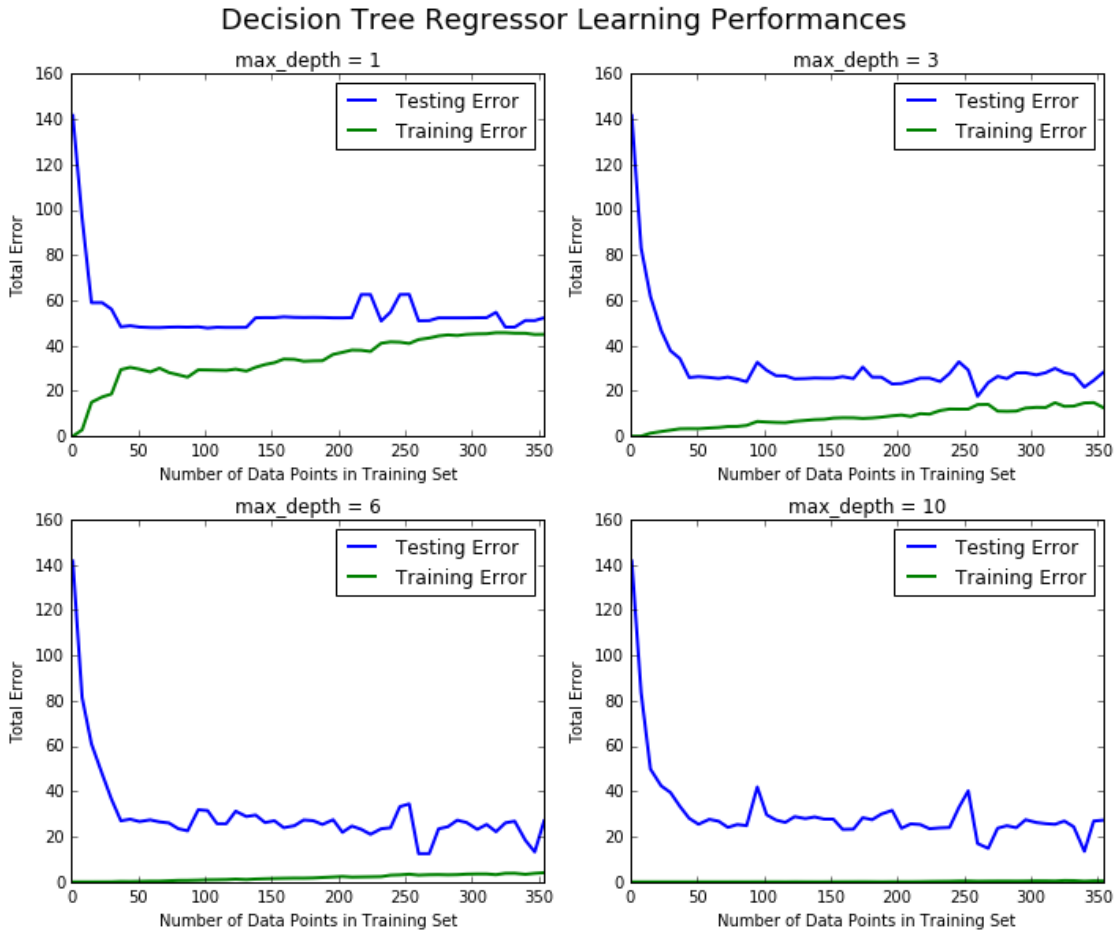
6 Analyzing Model Performance

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 [76]: learning_curves(X_train, y_train, X_test, y_test)

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

/usr/local/lib/python2.7/dist-packages/matplotlib/figure.py:397: UserWarning: matplotlib is currently us
  "matplotlib is currently using a non-GUI backend, "
```



6.1 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: For each one of the above graphs, the testing error generally decreases. The term generally is used due to fluctuations that causes the error to go either up or down. If one learning curve is to be chosen I would look at the third graph with max_depth = 6. As the size of the training set increases, the training error seems to increase with fluctuations once again. This is notable with all other graphs and least notable with the graph with max_depth = 10.

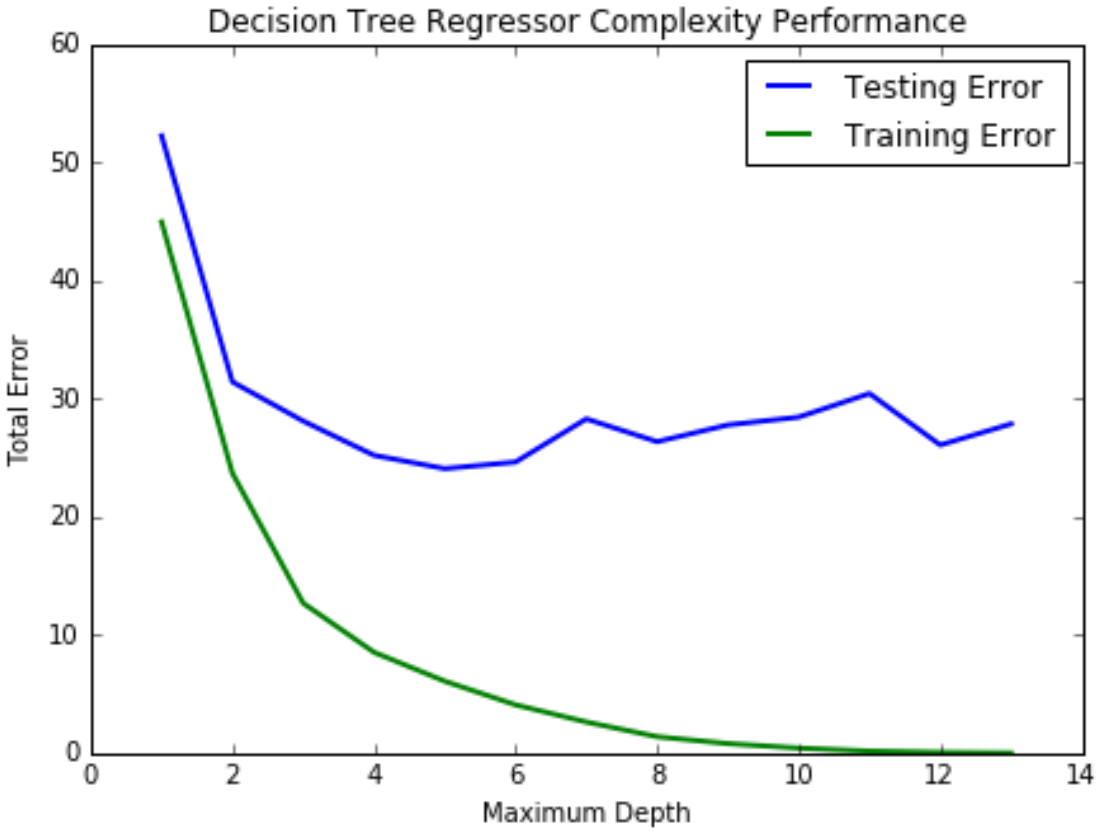
6.2 Question 8

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: Max depth 1 has a high bias which is easily seen by the higher error for the testing set. Max depth 10 has seems to have lower error for the testing set which would mean a correct fit. Observing the testing error curve more closely one would identify the curve stabilizing and not going up for max depth 10. This would mean it is not having an overfit or high variance problem and will be able to identify new data effectively.

```
In [77]: model_complexity(X_train, y_train, X_test, y_test)

Creating a model complexity graph. . .
```



6.3 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 max depth increases the training error decreases. While testing decreases and then slightly rises. The second rise in testing error could be accounted for high variance/overfitting. Based on this graph, I would choose a max depth of 5. This is the point before the testing error endures any rise after it's initial decrease. This would be an attempt to avoid overfitting while having the best regressor model possible.

7 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.

7.1 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 [78]: 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 parameter for the model is 4. It is approximately near 5 which was my hypothesis in the previous question. I assume the computer was able to identify 4 as the max_depth with the lowest testing error, where as my eyes can only approximate this solution.

7.2 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 [84]: import pandas as pd
         housing_pricespd = pd.DataFrame(data=housing_prices)
         sale_price = reg.predict(CLIENT_FEATURES)
         print "Predicted value of client's home: {0:.3f}".format(sale_price[0])
         print "pandas describe of housing_prices"
         housing_pricespd.describe()
```

```
Predicted value of client's home: 21.630
pandas describe of housing_prices
```

```
Out[84]:                 0
         count  506.000000
         mean    22.532806
         std      9.197104
         min      5.000000
         25%     17.025000
         50%     21.200000
         75%     25.000000
         max     50.000000
```

Answer: The best selling price according to the parameter-tuned model is 21.630. General stats about the dataset prices include the mean as 22.532, with std 9.19, and 50% being at 21.2. In comparison the clients house is slightly less than the mean of the dataset while being slightly more than the data at 50%.

7.3 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: Yes this model could be used to predict the selling price of future clients' homes in the Greater Boston area. There were many reasons that lead me to this decision. First we chose a model with max depth of 4 - this model has takes account into not overfitting while not having high bias. In addition we used gridsearchcv to tune for other best parameters.