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), 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), 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 [3]:
```
# 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!"
```

```
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!

In [109]:
```
# Number of houses in the dataset
n_houses, n_features = np.shape(housing_features)

total_houses = n_houses
#print total_houses

# Number of features in the dataset
total_features = n_features

# 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
```

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: 1.CRIM, Crime rate in an area can invoke psychological fear & change the perceived safety of the environment. Places with high crime rates, no matter how gentrified its surrounding areas are can have a detrimental impact on the value of the land. One anecdote is East Palo Alto, and West Oakland. Although they are in Silicon Valley, they have a reputation for being an unsafe place to live and thus have housing that is lower than the average value in the region. 3.INDUS, businesses that are non-retail have a chance to be manufacturing-related jobs. Although in the past, this could potentially be a suite factory, some of these businesses could still be an eyesore for housing. For example, an industrial warehouse could lure solicitation, theft, or illegal activity because of the open space it has & its somewhat difficult to maintain security. 7.AGE, buildings before 1940 held to different standards than to potential scrutiny post-1940s. The materials made from the house may be cheaper, or it'd be a budding concern whether the building has been rennovated to meet newer building safety codes.

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 [4]:
```
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, INDUS: 18.1, 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.

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 [137]:
```
# Put any import statements you need for this code block here
import sklearn
from sklearn.cross_validation import train_test_split #needed to ma
ke the name such
#X = housing_features
#y = 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 = housing_features
 #   y = housing_prices

    # Shuffle and split the data
    X_train, X_test, y_train, y_test= train_test_split(X ,y, test_s
ize = 0.3, random_state=0)
    # 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 X_train
    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: The training subset is what we have as our "gold standard" for our model. The testing subset is needed as as the actual data where go about applying our model to. This enables us to explore the estimated model properties.

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 [114]:
```
# 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: I chose the mean squared error (MSE) because it does a reliable job of checking the errors on average of the prediction and result. This comes in handy when we plan on adjusting the parameters of the train and test data. We can simply pick the lowest MSE because we want to point more weight to points AWAY from the mean.

Accuracy, F1 Score, Precision or recall would not be appropriate in this case because each of these only gives specific errors, when we're more concerned with total errors.

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)](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)](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 [115]:

```
# Put any import statements you need for this code block
from sklearn import metrics
from sklearn import grid_search
from sklearn.cross_validation import cross_val_score
from sklearn.tree import DecisionTreeRegressor

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() #Should work...
    print regressor
#    print regressor
    # 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
#    print fbeta_score
    scoring_function = metrics.make_scorer(performance_metric,great
er_is_better=False)
    # Make the GridSearchCV object
    reg = grid_search.GridSearchCV(regressor,parameters,scoring_fun
ction)
    # 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."
```

```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_feature
s=None,
           max_leaf_nodes=None, min_samples_leaf=1, min_samples_sp
lit=2,
           min_weight_fraction_leaf=0.0, random_state=None,
           splitter='best')
make_scorer(performance_metric, greater_is_better=False)
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_sp
lit=2,
           min_weight_fraction_leaf=0.0, random_state=None,
           splitter='best'),
       fit_params={}, iid=True, loss_func=None, n_jobs=1,
       param_grid={'max_depth': (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)},
       pre_dispatch='2*n_jobs', refit=True, score_func=None,
       scoring=make_scorer(performance_metric, greater_is_better=F
alse),
       verbose=0)
blah
PARAMS!!!
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_sp
lit=2,
           min_weight_fraction_leaf=0.0, random_state=None,
           splitter='best'),
       fit_params={}, iid=True, loss_func=None, n_jobs=1,
       param_grid={'max_depth': (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)},
       pre_dispatch='2*n_jobs', refit=True, score_func=None,
       scoring=make_scorer(performance_metric, greater_is_better=F
alse),
       verbose=0)
Successfully fit a model!
```

Question 5

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

Answer: The grid search algorithm searches estimator parameters by searching a parameter space for the best local extrema leading to the best cross-validation.

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 when you hold out a part of the data (test set) and use the training set experiment to compare to the supervised training algorithm. Cross-validation would be helpful in preventing us from overfitting the data.

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 [116]:
```
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()
```

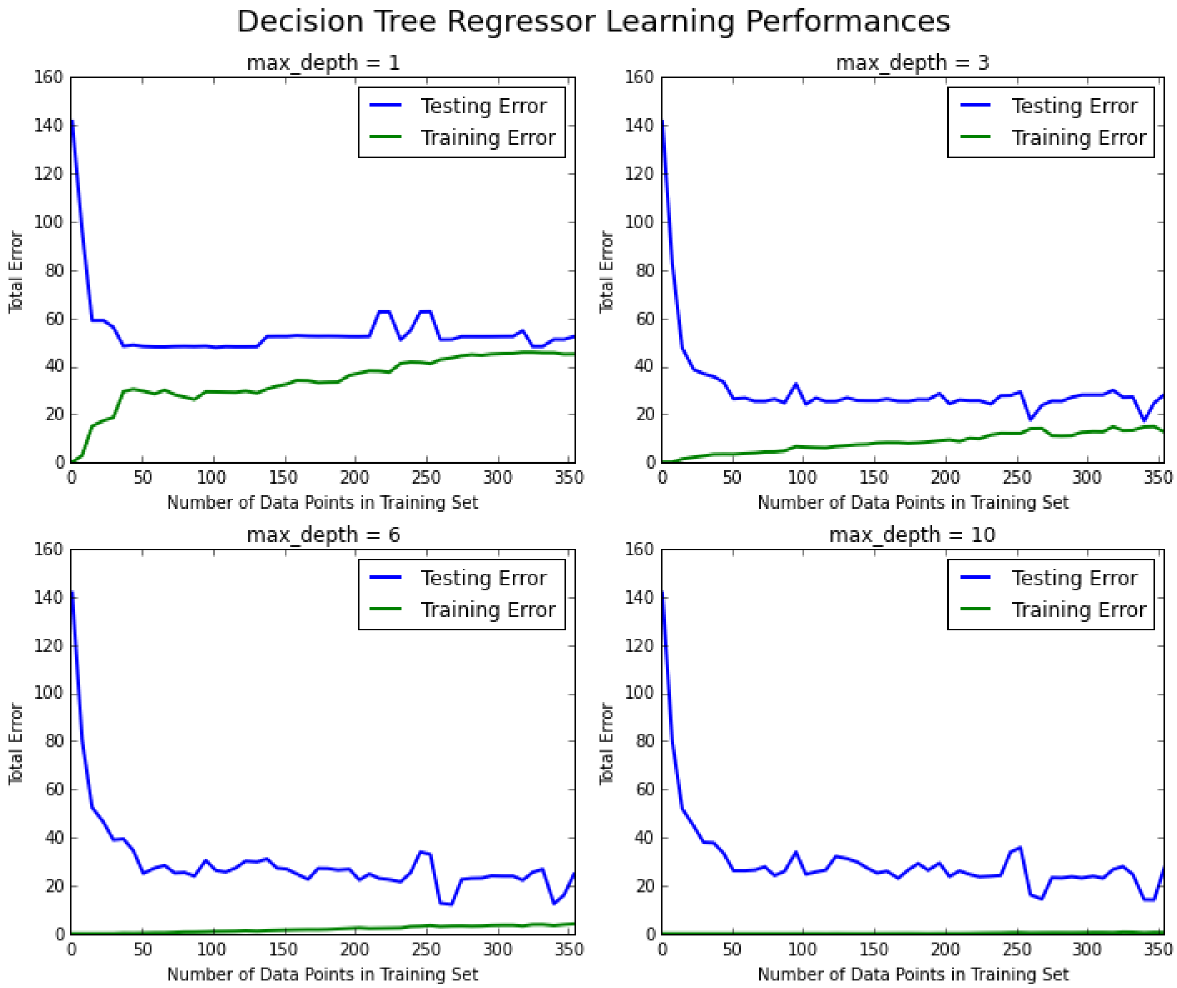
In [117]:
```
def model_complexity(X_train, y_train, X_test, y_test):
    """ Calculates the performance of the model as model complexity
increases.
        The learning and testing errors rates are then plotted. """

    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 [138]:
```
learning_curves(X_train, y_train, X_test, y_test)
```

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



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 chose the max_depth = 6 as my learning curve graph of choice as its not too shallow, not too deep, but just right. As the size of the data points in the training set increase, the training error decreases. One the max_depth reaches 10, there is no longer any training error.

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 suffers from underfitting and thus high bias. When you examine the training error, you'll notice that it's roughly incremental to 40 once you read 230 data points. The high training error would lead us to conclude more fitting is needed. When max_depth reaches 10 it suffers from overfitting and thus high variance. This is inspected by observing that the training level is at a perfect 0 on max_depth 0 and the error level at a lower level of max_depth of 6 is already at exceedingly low levels close to 0.

In [134]:
```
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, training error decreases. I would consider a max depth of 6 to have the best likelihood of generalizing the dataset. The testing area is the lowest at this point, and any higher and you get a spike in the increase of the testing error.

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 [140]:
```
print "Final model has an optimal max_depth parameter of", reg.get_params()['max_depth']
```

```
Final model has an optimal max_depth parameter of 9
```

Answer: This contradicts some of my intuition that the max_depth of 6 is the ideal parameter that gives the most optimal result. From inspection of the training/testing error, I was heavily factoring the test error. However, when we reach 8, the testing error decreases, and the training error significantly decreases. So the output of 9 is still an acceptable result. A max_depth of 10 runs the risk of over-fitting because the testing error starts to increase at this point.

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 [141]:
```
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: 19.327
```

Answer: 21.2 (median house value) +- 9.188 (STD). It's clear that since we have a house value less than the median, we're giving our client a really good offer. It's also a reasonable value as it lies within our standard deviation value.

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: In general, the client features are what you would expect would affect the selling price for a home. So long as the highest priority features are kept in place, the model should be a good predictive model of the housing prices.