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 [18]:
```
# 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 p
rice
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!
```

Exploring the feautres

In [2]:
```
import pandas as pd

housing_series = pd.DataFrame(housing_prices)
print "housing_series.describe"
print housing_series.describe()
print "\n"
```

```
housing_series.describe
                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
```

In [3]:
```
data = pd.DataFrame(housing_features,columns=["CRIM", "ZN", "INDUS", "CHAS",
"NOX", "RM", "AGE", "DIS", "RAD", "TAX", "PTRATIO", "B", "LSTAT"])

print "data.describe"
print data.describe()
print "\n"
data
```

```
data.describe
              CRIM          ZN       INDUS        CHAS         NOX
RM  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.0000
00
mean     3.593761   11.363636   11.136779    0.069170    0.554695    6.2846
34
std      8.596783   23.322453    6.860353    0.253994    0.115878    0.7026
17
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.5610
00
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.8855
00
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.2085
00
75%      3.647423   12.500000   18.100000    0.000000    0.624000    6.6235
00
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.7800
00

               AGE         DIS         RAD         TAX     PTRATIO
 B  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.0000
00
mean    68.574901    3.795043    9.549407  408.237154   18.455534  356.6740
32
std     28.148861    2.105710    8.707259  168.537116    2.164946   91.2948
64
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.3200
00
25%     45.025000    2.100175    4.000000  279.000000   17.400000  375.3775
00
50%     77.500000    3.207450    5.000000  330.000000   19.050000  391.4400
00
75%     94.075000    5.188425   24.000000  666.000000   20.200000  396.2250
00
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.9000
00

            LSTAT
count  506.000000
mean    12.653063
std      7.141062
min      1.730000
25%      6.950000
50%     11.360000
75%     16.955000
max     37.970000
```

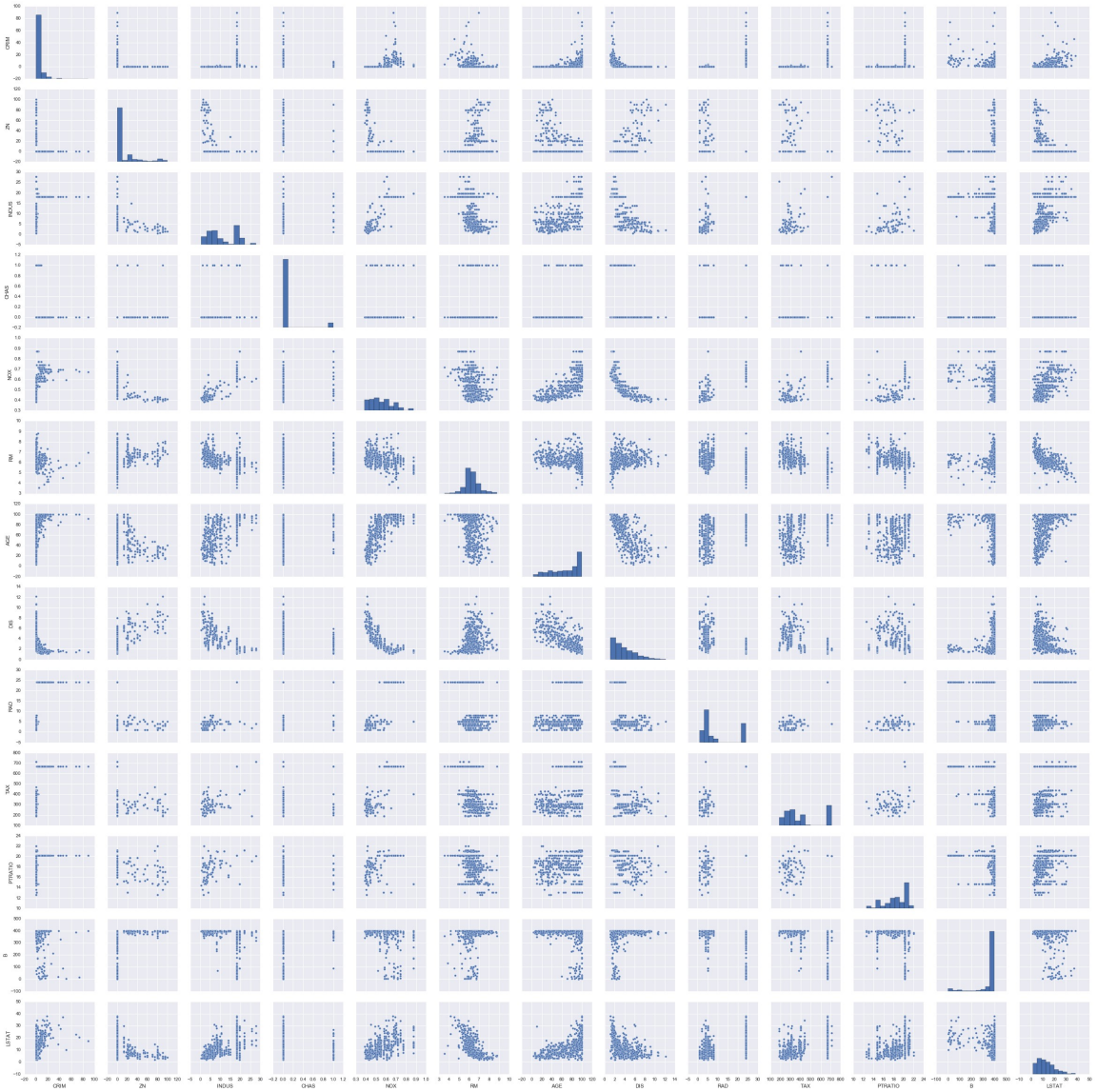
Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTA
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33
5	0.02985	0.0	2.18	0	0.458	6.430	58.7	6.0622	3	222	18.7	394.12	5.21
6	0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.60	12.43
7	0.14455	12.5	7.87	0	0.524	6.172	96.1	5.9505	5	311	15.2	396.90	19.15
8	0.21124	12.5	7.87	0	0.524	5.631	100.0	6.0821	5	311	15.2	386.63	29.93
9	0.17004	12.5	7.87	0	0.524	6.004	85.9	6.5921	5	311	15.2	386.71	17.10
10	0.22489	12.5	7.87	0	0.524	6.377	94.3	6.3467	5	311	15.2	392.52	20.45
11	0.11747	12.5	7.87	0	0.524	6.009	82.9	6.2267	5	311	15.2	396.90	13.27
12	0.09378	12.5	7.87	0	0.524	5.889	39.0	5.4509	5	311	15.2	390.50	15.71
13	0.62976	0.0	8.14	0	0.538	5.949	61.8	4.7075	4	307	21.0	396.90	8.26
14	0.63796	0.0	8.14	0	0.538	6.096	84.5	4.4619	4	307	21.0	380.02	10.26
15	0.62739	0.0	8.14	0	0.538	5.834	56.5	4.4986	4	307	21.0	395.62	8.47
16	1.05393	0.0	8.14	0	0.538	5.935	29.3	4.4986	4	307	21.0	386.85	6.58
17	0.78420	0.0	8.14	0	0.538	5.990	81.7	4.2579	4	307	21.0	386.75	14.67
18	0.80271	0.0	8.14	0	0.538	5.456	36.6	3.7965	4	307	21.0	288.99	11.69
19	0.72580	0.0	8.14	0	0.538	5.727	69.5	3.7965	4	307	21.0	390.95	11.28
20	1.25179	0.0	8.14	0	0.538	5.570	98.1	3.7979	4	307	21.0	376.57	21.02
21	0.85204	0.0	8.14	0	0.538	5.965	89.2	4.0123	4	307	21.0	392.53	13.83
22	1.23247	0.0	8.14	0	0.538	6.142	91.7	3.9769	4	307	21.0	396.90	18.72
23	0.98843	0.0	8.14	0	0.538	5.813	100.0	4.0952	4	307	21.0	394.54	19.88
24	0.75026	0.0	8.14	0	0.538	5.924	94.1	4.3996	4	307	21.0	394.33	16.30
25	0.84054	0.0	8.14	0	0.538	5.599	85.7	4.4546	4	307	21.0	303.42	16.51
26	0.67191	0.0	8.14	0	0.538	5.813	90.3	4.6820	4	307	21.0	376.88	14.81
27	0.95577	0.0	8.14	0	0.538	6.047	88.8	4.4534	4	307	21.0	306.38	17.28
28	0.77299	0.0	8.14	0	0.538	6.495	94.4	4.4547	4	307	21.0	387.94	12.80
29	1.00245	0.0	8.14	0	0.538	6.674	87.3	4.2390	4	307	21.0	380.23	11.98
...	...	...	...	...	...	...	...	...	...	...	...	...	...
476	4.87141	0.0	18.10	0	0.614	6.484	93.6	2.3053	24	666	20.2	396.21	18.68
477	15.02340	0.0	18.10	0	0.614	5.304	97.3	2.1007	24	666	20.2	349.48	24.91
478	10.23300	0.0	18.10	0	0.614	6.185	96.7	2.1705	24	666	20.2	379.70	18.03
479	14.33370	0.0	18.10	0	0.614	6.229	88.0	1.9512	24	666	20.2	383.32	13.11
480	5.82401	0.0	18.10	0	0.532	6.242	64.7	3.4242	24	666	20.2	396.90	10.74
481	5.70818	0.0	18.10	0	0.532	6.750	74.9	3.3317	24	666	20.2	393.07	7.74
482	5.73116	0.0	18.10	0	0.532	7.061	77.0	3.4106	24	666	20.2	395.28	7.01
483	2.81838	0.0	18.10	0	0.532	5.762	40.3	4.0983	24	666	20.2	392.92	10.42

In [4]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.pairplot(data)
```

Out[4]: <seaborn.axisgrid.PairGrid at 0x923f860>



In [5]:
```
data.corr()
```

Out[5]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS
CRIM	1.000000	-0.199458	0.404471	-0.055295	0.417521	-0.219940	0.350784	-0.377904
ZN	-0.199458	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.664408
INDUS	0.404471	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.708027
CHAS	-0.055295	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518	-0.099176
NOX	0.417521	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470	-0.769230
RM	-0.219940	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265	0.205246
AGE	0.350784	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000	-0.747881
DIS	-0.377904	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881	1.000000
RAD	0.622029	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022	-0.494588
TAX	0.579564	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456	-0.534432
PTRATIO	0.288250	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515	-0.232471
B	-0.377365	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534	0.291512
LSTAT	0.452220	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339	-0.496996

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 [19]:
```
# Number of houses in the dataset
total_houses = np.shape(housing_prices)[0]

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

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

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

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

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

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

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

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

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

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: LSAT(% lower status of the population), RM(average number of rooms per dwelling) , DIS (weighted distances to five Boston employment centres)

LSAT measures the population density, RM signifies the number of rooms and DIS is basically the distance between the house and the top 5 employement centers based on which the cost of the house majorly depends, though there are other features which have some impact, but these three I found to be more significant.

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 [20]:
```
print CLIENT_FEATURES
LSAT = CLIENT_FEATURES[0][12]
RM = CLIENT_FEATURES[0][5]
DIS = CLIENT_FEATURES[0][7]
print LSAT,RM,DIS
```

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

Answer: LSAT = 12.13 RM = 5.609 DIS = 1.385

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 if the `shuffle_split_data` function is working if the statement *"Successfully shuffled and split the data!"* is printed.

In [21]:
```
# Put any import statements you need for this code block here
from sklearn.cross_validation import train_test_split
def shuffle_split_data(X, y):
    """ Shuffles and splits data into 70% training and 30% testing subsets,
        then returns the training and testing subsets. """
    #Shuffle and Split the Data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)


    # 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_features, h
ousing_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!
```

Question 3

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

Answer: This is the thumb rule of any machine learning problem, so as to not introduce any bias while evaluating the model. Thus, the dataset on which training is done always put separately as of the dataset using which the model evaluation will be done. Consider if we evaluate on the same dataset we have trained then we are just fooling our evaluation metrics because the predict function will be predicting the same values what it has learnt. Thus, test dataset helps us understand that how our model will behave on the future dataset or in real life where we will be using the model.

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 if the `performance_metric` function is working if the statement *"Successfully performed a metric calculation!"* is printed.

In [22]:
```
# 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 predicted val
ues
        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 calculation."
```

```
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: The most appropriate metrics is Mean Squared Error for this regression problem rather than Accuracy, F1 Score, Precision and Recall because these metrics are basically suitable for Classification problem where we have the number of correct classified vs number of incorrect classified example. In case of regression problems Mean Square Error or mean Absolute Error can be used but the best one is Mean Sqaure Error because Squaring always gives a positive value, so the sum will not be zero and Squaring emphasizes larger differences - a feature that turns out to be both good and bad (think of the effect outliers have).

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 if the `fit_model` function is working if the statement *"Successfully fit a model to the data!"* is printed.

In [23]:
```
# Put any import statements you need for this code block
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import make_scorer
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=Fals
e)

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


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

    # Return the optimal model
    return reg


# 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: In any machine learning repressor or classifier there are parameters that decides the output of the model, thus the tuning of such parameters is very crucial in order to make a good model. Thus, it becomes an optimization problem where choosing the values of such parameters so as to get the best model. In terms of optimization problem, we can say that model evaluation metrics becomes the fitness function based on which we pick the best parameter values. For example, in Decision Trees max_depth, max_features, min_sample_splits etc are the parameters that we need to choose so that we get the best model thus we end up with many combinations of these parameters and we need to choose the best combination from this search space for which we can use grid search.

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: Though train and test split is the basic and simplest way to evaluate the model, but even this becomes a sampling problem to get the best sample from the whole dataset for testing and training, and there are very much likely chances that we might end up with the very biased testing sample, thus we prefer to use k-cross validation in which we iterate over k test data from the complete data and take the average score for evaluation metrics so as to get the better validation of the 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 [24]:
```
def learning_curves(X_train, y_train, X_test, y_test):
    """ Calculates the performance of several models with varying sizes of tr
aining 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.round(np.linspace(1, len(X_train), 50))
    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 m
ax_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 [25]:
```
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()
```

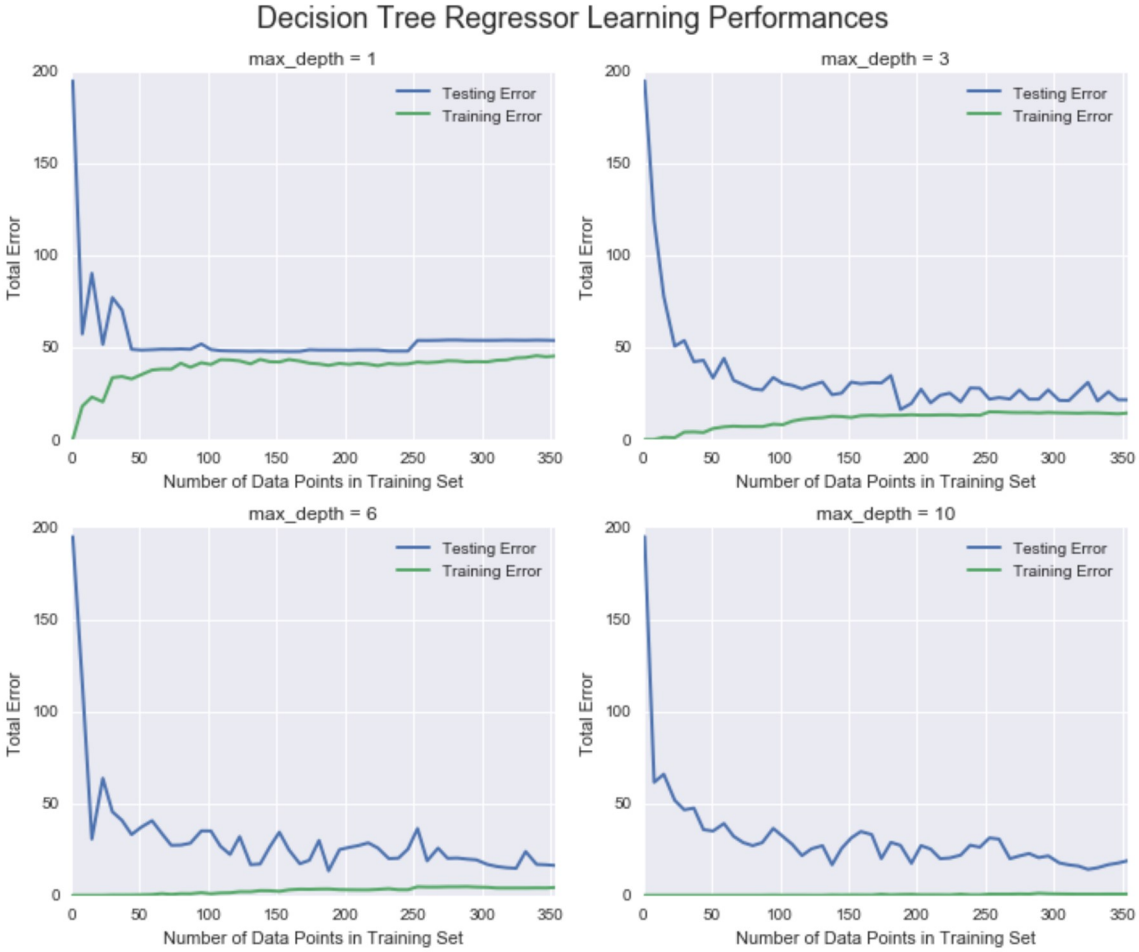
Analyzing Model Performance

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 [26]:
```
learning_curves(X_train, y_train, X_test, y_test)
```

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

C:\Anaconda2\lib\site-packages\ipykernel\__main__.py:24: DeprecationWarning
: using a non-integer number instead of an integer will result in an error
in the future
C:\Anaconda2\lib\site-packages\ipykernel\__main__.py:27: DeprecationWarning
: using a non-integer number instead of an integer will result in an error
in the future
```



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:

When max_depth = 3, with the increase in the number of samples the training error seems to get saturated after particular number which is also the case with testing error after number of data points > 50

The general trend that can be observed after going through all the pltos is that as the size of training dataset increases, the model generally gets better as compared to the small training dataset where the training error was considerable. As the training error itself if very much considerable thus the testing error makes no senese as it has to be poor also.

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 max_dept=1, the model model underwent the underfitting, where the training error is so large, that signifies that almost nothing has been learnt from the data and no pattern has been discovered from the data irrespective of the number of samples given from traning. This is because the model has learnt from the small dataset and only can perform well with that amount of data only, but could not do the justice with the testing dataset and thus perform really poor with testing data.Thus, we can observe high variance.

When max_dept=10, the model model underwent the overfitting, where the training error is went to almost 10 , though the test error was still significant thus model 10 can not generalize for future data. In this case the training error almost tends to zero, but the testing error is still considerable. Thus we can conclude that for this case the model is learnt is such a way that it can minimize the training error but can not generalize well and hence gives poor performance on testing data, which is not part of the training data, which is the case of high bias for training data.

In [27]:
```
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 complexity of the model increases the training error as well as the testing error both starts to decreas but we found a point where though training error was decreasing but testing error starts increasing thus we have reached a breakeven point.

Therefore, max depth = 4 could best generalize the data , even if we take 5 also, then the results will be somewhat similar, but 4 seems to be much suitable.

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`. *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 [28]:
```
print "Final model optimal parameters:", reg.best_params_
```

```
Final model optimal parameters: {'max_depth': 4}
```

Answer: The max_depth comes to be 4, thus either 4 or 5 seems to be one of the choice

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 [29]:
```
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: On the above mentioned feature set the predicted value comes to be 21.630. This is quite less that ehe average values i.e. 22.5 and but more than the first standard deviation, so the predicted price does not even an outlier as it also lies between the minimum and maximum range. Thus, it is fair to sell at this price.

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, I can use this model for the future clients' homes in the Greater Boston area.

Though I would also like to play with max_features and min_sample_split, but the current model also seems convincing.