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 [1]:

```
# Importing a few necessary libraries
import numpy as np
import matplotlib.pyplot as pl
from sklearn import datasets


# 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, 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
# 1*506, 506->sample size, housing_prices->label
housing_features = city_data.data
# 506*13


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 [2]:

```
# Number of houses in the dataset
total_houses =housing_prices.shape[0]

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

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

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

# 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: I think that these attributes are significant.

1.RM(average number of rooms per dwelling -> It measures a number of room in a house, and it can increase housing price evidently.

2.RAD(index of accessibility to radial highways) -> I think that most people are willing to buy a house if it is easily accesible to highways. Because they can go shopping or go to work more easily. Therefore, high index of accessibility to radial highways can increase housing price.

3.AGE(proportion of owner-occupied units built prior to 1940) ->I think that most people are unwilling to buy a house which is located near buildings built prior to 1940. Because it has a higher risk of being damaged by collapse of old builings. Also, old buildings can cause a bad effect on aesthetic feature of the area. Therefore, low portion of owner-occupied units prior to 1940 can increase housing price.

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
chosen_features = ['RM', 'AGE', 'RAD']
features = city_data.feature_names.tolist()
for feature in chosen_features:
    index = features.index(feature)
    print CLIENT_FEATURES[0][index]
```

```
[[11.95, 0.0, 18.1, 0, 0.659, 5.609, 90.0, 1.385, 24, 680.0, 20.2, 33
2.09, 12.13]]
5.609
90.0
24
```

Answer: 1.RM -> CLIENT_FEATURES[5]=5.609 2.AGE -> CLIENT_FEATURES[6]=90 3.RAD -> CLIENT_FEATURES[8]=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]:

```
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. """
    # Shuffle and split the data


    X_train, X_test, y_train, y_test=cross_validation.train_test_split(X,y,test_si
ze=0.3, random_state=42)

    # 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, housing_p
rices)
    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 4

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

Answer: It is required to check generalization error of prediction caused by overfitting.

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 [35]:

```
from sklearn.metrics import mean_squared_error

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!"
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: Our estimator is a regressor, so MSE and MAE are proper to predict and analyze. MSE is more sensitive to the extreme errors(for example, outliers) between actual values and predicted values because of the square term. That is, MSE provides the bigger error value than MAE does if there are outliers. Therefore, we can predict the level of outliers when the results of MSE and MAE are compared.

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 [70]:

```
# Put any import statements you need for this code block
from sklearn.metrics import make_scorer
from sklearn.tree import DecisionTreeRegressor
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)} # DecisionTreeRegressor
    # Candidataes from a grid of parameter values

    # Make an appropriate scoring function
    scoring_function = make_scorer(mean_squared_error, greater_is_better=False)

    # Make the GridSearchCV object
    reg = GridSearchCV(regressor, parameters, scoring=scoring_function, cv=20)

    # Fit the learner to the data to obtain the optimal model with tuned parameter
s
    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: The grid search is one of the hyperparameter optimization process, which is finding the hyperparameters (usually meausred with cross-validation) among a subset of the hyperparameter space of the learning model. This algorithm is helpful to minimize the error from independent data set(caused by overfitting). Thus, the grid search is applicable to show the better performance of the learning model by tuning and choosing the proper hyperparamters.

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: Partitioning the data into a training set, a validation set, and a test set is helpful to minimize the generalization error. However, it drastically reduce the number of samples. The way to avoid this problem is 'Cross-validation'. When doing 'cross-validation', the validation set is not required. Instead, it splits into the trainig set into k samller sets. The CV is also helpful to find out the parameters of the best estimator which can minimize the generalization error.

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]:

```
def learning_curves(X_train, y_train, X_test, y_test):
    """ Calculates the performance of several models with varying sizes of trainin
g 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))
    # Generating matrix having 50 0 elements
    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_de
pth = depth
            regressor = DecisionTreeRegressor(max_depth = depth)

            # Fit the learner to the training data
            # X_Train[:s]->split X_train into X_train[:s]
            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_tra
in[: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 [10]:

```
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

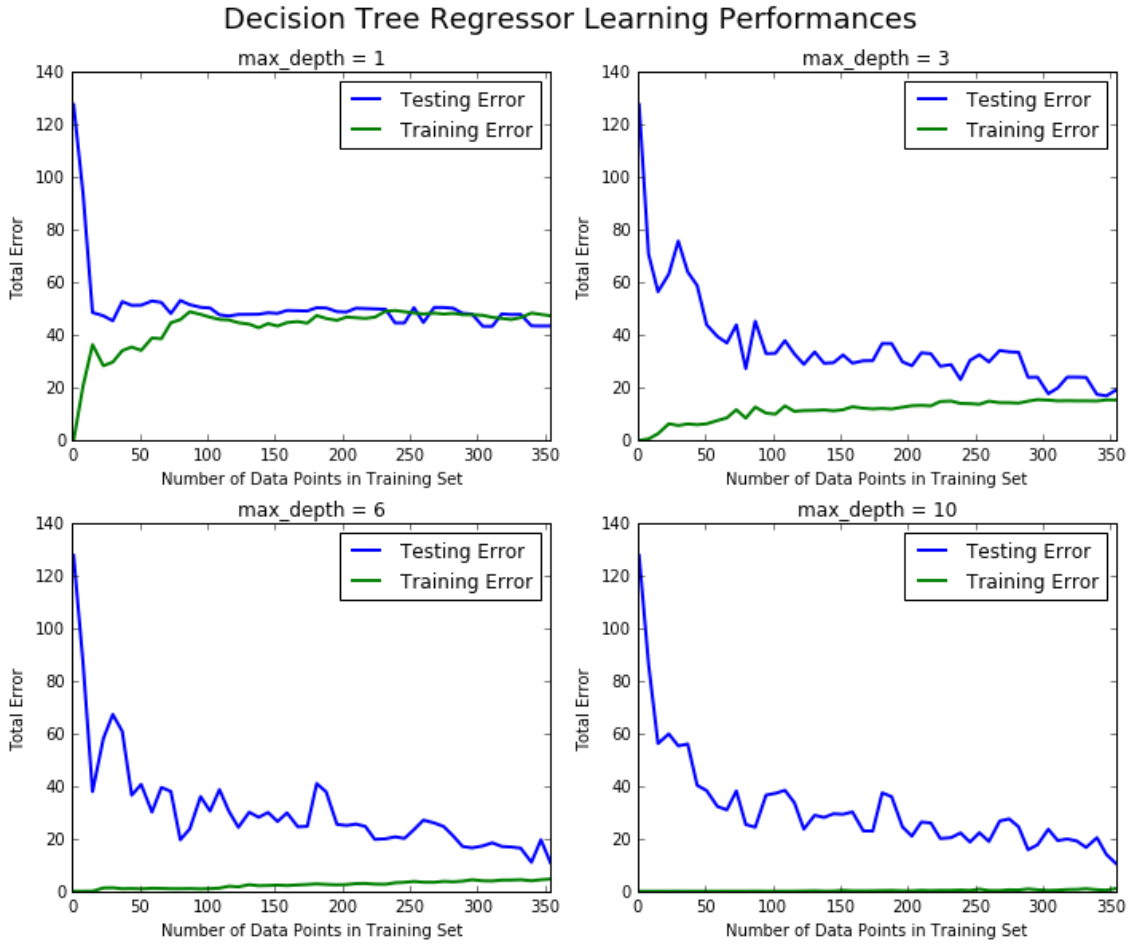
In this third section of the project, you'll take a look at several models' learning and testing error rates on various subsets of training data. Additionally, you'll investigate one particular algorithm with an increasing `max_depth` parameter on the full training set to observe how model complexity 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 [8]:

```
learning_curves(X_train, y_train, X_test, y_test)
```

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

E:\osm data\development_tool\Anaconda2\lib\site-packages\matplotlib\f
igure.py:397: UserWarning: matplotlib is currently using a non-GUI ba
ckend, so cannot show the figure
  "matplotlib is currently using a non-GUI backend, "
```



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 choose the learning curve with the max depth of 10. First, as the size of the training set increases, the training error stays around 0. This can be explained that because of the high max depth the learning model is fit to the trainig set closely. So, the training error does not increase even though the size of the training set increases. By just seeing this result, it may be thought that the learning model with the high max depth of 10 is the best model. However, when the test error is observed, this thouhgt should be reconsidered. As seen in the figure, the test error decreases approximately from 125 to 22 as the size of the trainig set increases. But, the value of the test error does not easily converge into that of the trainig error. This symptom is happend by overfitting. Thus, it can be conclueded that the highest depth is not always good because it can cause overfitting.

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 -> high bias (When the max_depth of 1 is used, the training error and the test error are almost same with the full training set. But the training error is considerablly big. It can be explained that the learning model is fit to the trainig set loosely(underfitting). This situation is usually called 'high bias'.

Max depth 10 -> high variance(When the max_depth of 10 is used, the training error stays around 0. But, the gap between the trainig error and test set does not converge easily. It can be explained that the learning model is fit to the trainig set tightly(overfitting). This situation is usually called 'high variance'.

In [11]:

```
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 model complexity(max_depth) increase, the training error decreases and is almost 0 at the max_depth of 13. This symptom can be explained that the learning model gets tightly fit to the training set with the higher max_depth. But, the high max_depth also leads to overfitting problem. That is, as the model complexity increases, the learning model is less sensitive to independet data sets(equivalently, the model makes the higher test error as the max_depth gets higher.). Therefore, choosing the proper max_depth value(not extremely low or high) is helpful to keep both of the training error and test error low. From the figure, it is known that the test error has the global minimum around the max_depth of 6, and the training error is also considerably low. According to this reason, I chose the max_depth of 6 to generalize the learning model.

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.

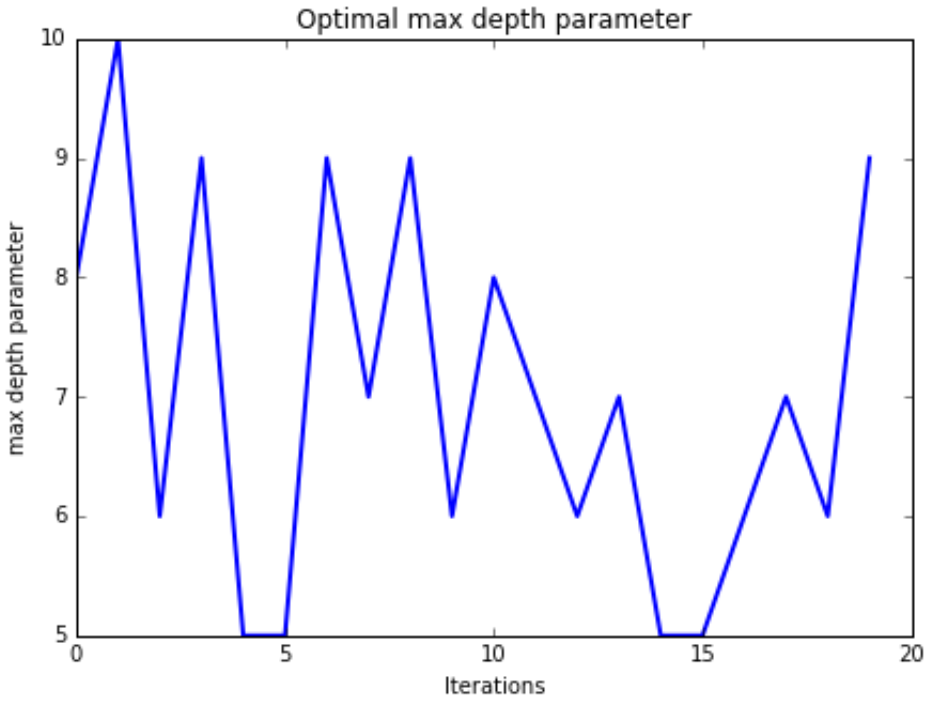
Question 10

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 [73]:

```
print "Final model has an optimal max_depth parameter of", reg.get_params()['max_d
epth']
iteration_for_max_depth= np.arange(0, 20)
max_depth_result=np.zeros(len(iteration_for_max_depth))
for i in iteration_for_max_depth:
    reg = fit_model(housing_features, housing_prices)
    max_depth_result[i]=reg.get_params()['max_depth']
print max_depth_result
pl.figure(figsize=(7, 5))
pl.title('Optimal max depth parameter')
pl.plot(iteration_for_max_depth, max_depth_result, lw=2)
pl.legend()
pl.xlabel('Iterations')
pl.ylabel('max depth parameter')
pl.show()
print "the mean of the max_depth paramters : ", max_depth_result.mean()
```

```
Final model has an optimal max_depth parameter of 7
[  8.  10.   6.   9.   5.   5.   9.   7.   9.   6.   8.   7.   6.
7.   5.
   5.   6.   7.   6.   9.]
```




```
the mean of the max_depth paramters :  7.0
```

Answer: By running the grid serach for 20 times, it is known that the average of the optimized parameters is 7. This result has the small difference with my initial intuition, but it could be acceptable.

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 [74]:

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

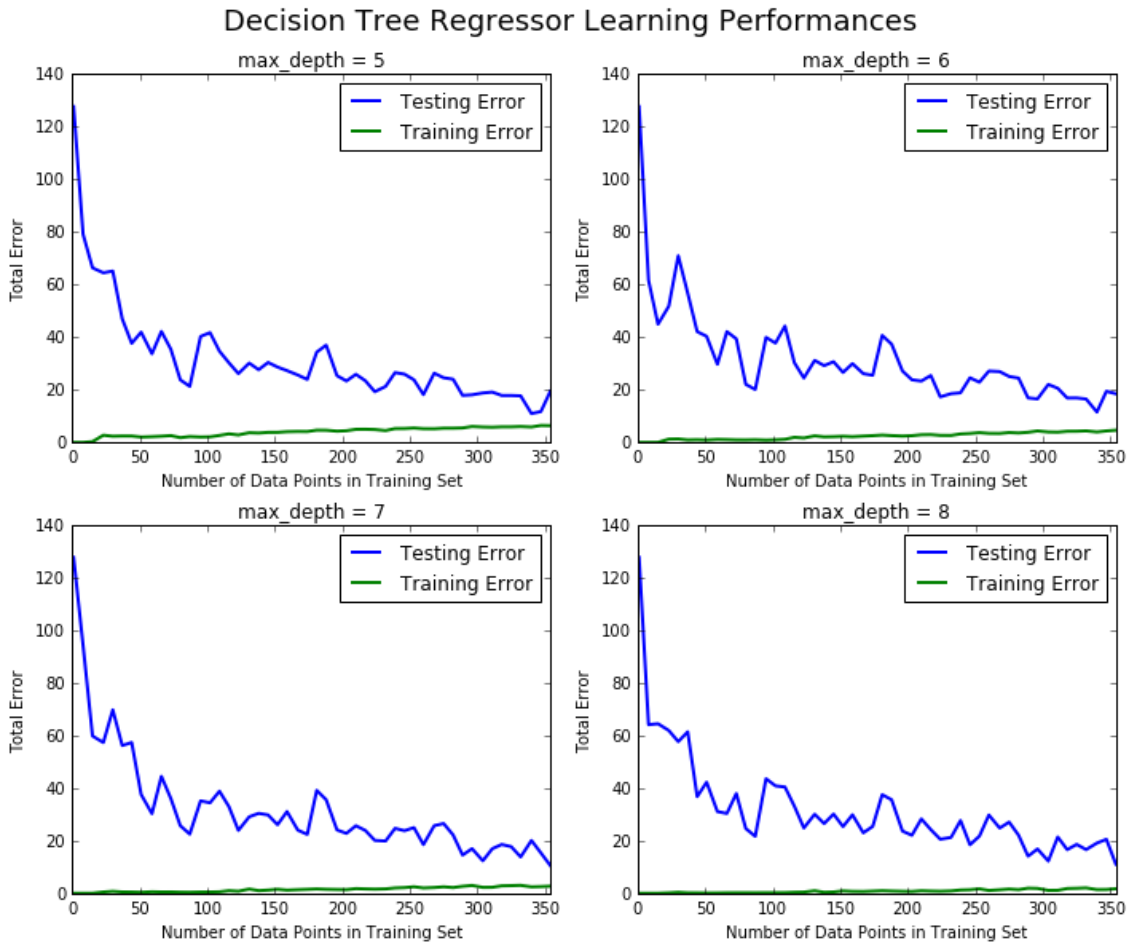
```
Predicted value of client's home: 19.327
```

Answer: The best selling price for the client's home, obtained from the learning model, is 19.327. First, this result is confidential because it is in a range of one standard deviation from the mean(where 68% of the data should be).

In [81]:

```
def learning_curves_after_grid_search(X_train, y_train, X_test, y_test):
    fig = pl.figure(figsize=(10,8))
    sizes = np.rint(np.linspace(1, len(X_train), 50)).astype(int)
    train_err = np.zeros(len(sizes))
    test_err = np.zeros(len(sizes))
    for k, depth in enumerate([5,6,7,8]):
        for i, s in enumerate(sizes):
            regressor = DecisionTreeRegressor(max_depth = depth)
            regressor.fit(X_train[:s], y_train[:s])
            train_err[i] = performance_metric(y_train[:s], regressor.predict(X_tra
in[:s]))
            test_err[i] = performance_metric(y_test, regressor.predict(X_test))
        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)])
    fig.suptitle('Decision Tree Regressor Learning Performances', fontsize=18, y=
1.03)
    fig.tight_layout()
    fig.show()

learning_curves_after_grid_search(X_train, y_train, X_test, y_test)
```



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. There are two reasons.

Firtst, this prediction provides the selling price around the average and inside the one standard deviation. Second, as seen in the above figures, it is observed that the learning model with the optimized paramter of 7 from the grid search is most generalized.

It would be better that could consider the following questions:

Would additional data points (or the inclusion of data per year) benefit the model?

-> From the above figure with the optimized parameter of 7, additional data points can be helpful to decrease the gap between the errors of the training set and test set. But this effect would be not that big.

Is there a possibility of outliers in the data that can drastically change predictive results?

-> I think a possibility of outliers can affect the predictive results slightly. Because I used the optimized paramters of 7 obtained from the grid search, and by this behavior the learning model is properly fit to the data set(not underfitting or not overfitting).

Does this dataset feature enough characteristics about homes to be considered robust?

-> I think this dataset features enough characteristics in that the learning model shows the low training and test errors.

Does performing grid search on the entire dataset affect your confidence in the model?

-> Yes. Performing grid search on the entire dataset can increase confidence in the model, because it can alleviate the overfitting problem.