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 [4]:
```
# 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
CLIENT_FEATURES = [[11.95, 0.00, 18.100, 0, 0.6590, 5.6090, 90.00, 1.385, 24

# 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 [5]:
```
# Number of houses in the dataset
total_houses = housing_features.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 = housing_prices.mean()

# Median house value of the dataset
median_price = np.median(housing_prices, axis=0)

# Standard deviation of housing values of the dataset
std_dev = np.std(housing_prices, axis=0)

# 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'm assuming that you're looking for domain knowledge here, not a numerical analysis like R2)

LSTAT, CRIM, PTRATIO

LSTAT is "% lower status of the population" This would tend to increase as home prices decrease.

RM is "average number of rooms per dwelling" This would tend to increase as home prices increase, assuming that multi-unit dwellings are counted as multiple dwellings

DIS is "weighted distances to five Boston employment centres" This would tend to decrease as home prices increase.

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 [6]:
```
print CLIENT_FEATURES
```

```
[[11.95, 0.0, 18.1, 0, 0.659, 5.609, 90.0, 1.385, 24, 680.0, 20.2, 332.0
9, 12.13]]
```

Answer:

Client's LSTAT = 12.13, well within 1 standard deviation below the mean, which would tend to increase the price

Client's RM is 5.609, just barely within 1 standard deviation below the mean, which would tend to decrease the price

Client's DIS is 1.385, just outside 1 standard deviation below the mean, which would tend to increase the price

All three measures are fairly close to the mean, so the price should be fairly close to the middle of the distribution if there are no large outliers.

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 [7]:
```
# 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,
    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:

Because if we use all of the data for training we will overfit, and the model won't generalize well. So we need to hold out a test set to score how well the model does with data that is has not seen yet.

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 [8]:
```
# 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 va
        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:

Accuracy, Precision, Recall, and the F1 Score are classification metrics and this is a regression problem, so they are not appropriate metrics.

Mean Squared Error and Mean Absolute Error are regression metrics. Mean Squared Error always returns a positive distance from the predicted value to the actual value, which is useful, and emphasizes large errors. For this problem we want to set a price that is fairly close and try to avoid large 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).
- 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 [9]:
```
# Put any import statements you need for this code block
from sklearn.metrics import make_scorer
from sklearn.grid_search import GridSearchCV

def fit_model(X, y):
    """ Tunes a decision tree regressor model using GridSearchCV on the inpu
        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(performance_metric, greater_is_better=Fal

    # Make the GridSearchCV object
    reg = GridSearchCV(regressor, parameters, scoring=scoring_function, cv=5

    # Fit the learner to the data to obtain the optimal model with tuned par
    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:

Grid search is used to select hyperparameters by testing a range of values to find the best performing values. Hyperparameters are things like the training rate, not the weights assigned to features. Grid search does an exhaustive search of the defined space, as opposed to a randomized or sampled search. This requires more time during training, especially with large datasets, but may often yield a better result.

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:

Even with a separate test set there is still a danger of overfitting with the test set when using a parameter tuning method like grid search. Information in the test set essentially leaks into the model during the tuning process. So to avoid overfitting we need to hold out a third data set that is not used for parameter tuning, called a validation set.

The problem is that we have now made the training set even smaller, which can often be a problem. So to avoid that we can use cross-validation, like k-fold cross-validation for example. In k-fold cross-validation the training set is divided into equal parts or "folds", one fold is held out as a validation set, and then the model parameters are tuned. This repeats with each of the folds held out in turn, and the results are averaged together. The test set is then used for final scoring of the model to assess how well it generalizes.

I increased the number of folds from the default 3 to 5 because at 3 the optimum model complexity did not seem to make the complexity graph and learning curve.

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 [10]:
```
def learning_curves(X_train, y_train, X_test, y_test):
    """ Calculates the performance of several models with varying sizes of t
        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
            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

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

        # 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=1
    fig.tight_layout()
    fig.show()
```

In [11]:
```
def model_complexity(X_train, y_train, X_test, y_test):
    """ Calculates the performance of the model as model complexity increase
        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 dept
        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()
```

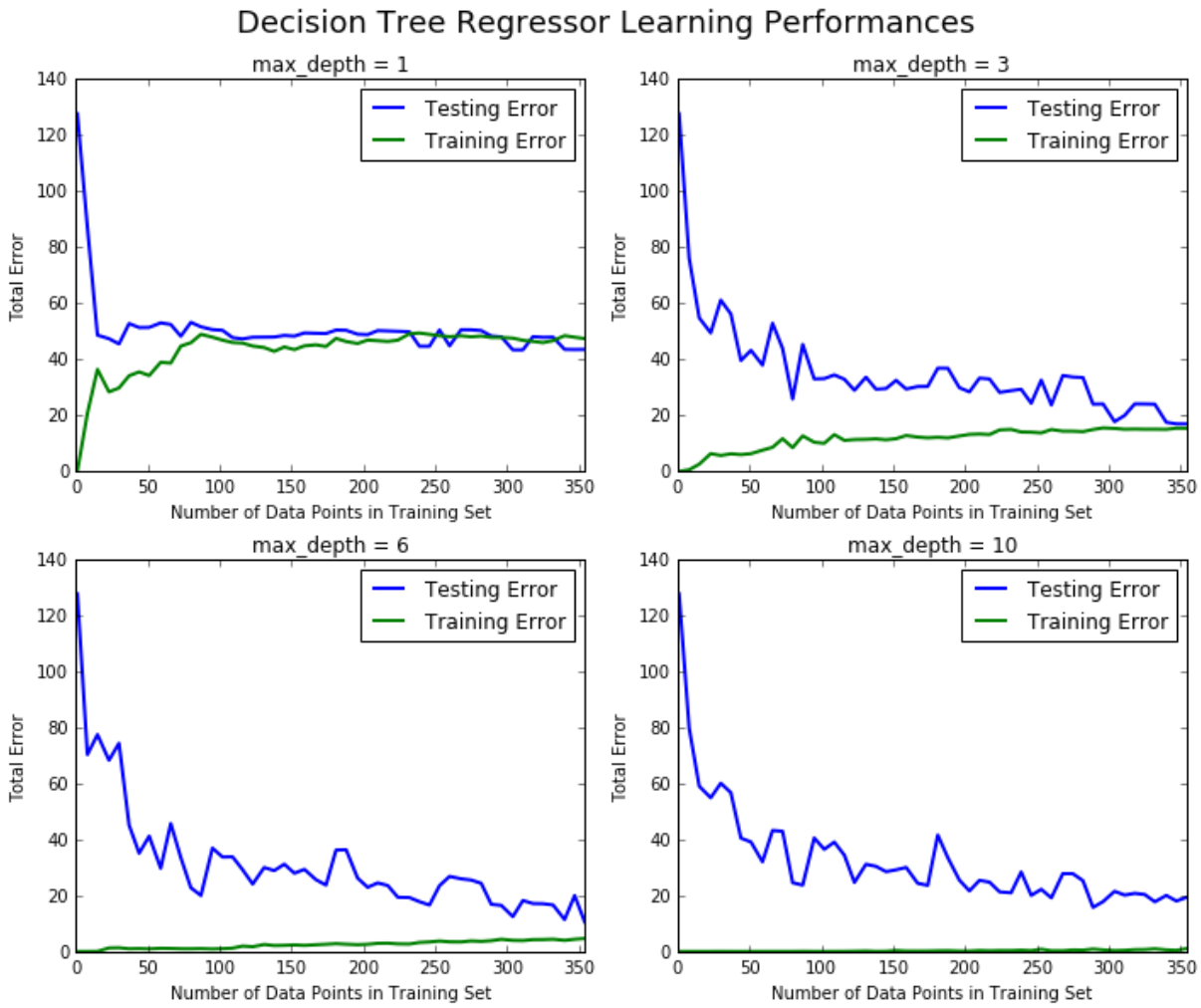
Analyzing Model Performance

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 [12]:
```
learning_curves(X_train, y_train, X_test, y_test)
```

```
J:\Python\Anaconda\lib\site-packages\ipykernel\__main__.py:24: Deprecatio
nWarning: using a non-integer number instead of an integer will result in
an error in the future
J:\Python\Anaconda\lib\site-packages\ipykernel\__main__.py:27: Deprecatio
nWarning: using a non-integer number instead of an integer will result in
an error in the future
J:\Python\Anaconda\lib\site-packages\matplotlib\figure.py:397: UserWarnin
g: matplotlib is currently using a non-GUI backend, so cannot show the fi
gure
  "matplotlib is currently using a non-GUI backend, "
```

```
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 would choose max_depth=6, because it seems to acheive the lowest testing error while training error continues to rise as the size of the training set increases. As the size of the training set increases the testing error falls, and the training and testing errors approach each other, which indicates that at mac_depth=6 we do not have a variance problem. The overall error seems to approach a fairly low level, which probably indicates that we don't have much of a bias problem either.

At max_depth=10 the training error maintains a very low level and does not rise as the size of the training set increases, which indicates overfitting and a variance problem. This is also indicated by the testing error remaining higher, and the testing and training error failing to converge after a minimum at about 290 datapoints of training data.

It would make sense to continue to explore values around max_depth=6 to find the optimum value.

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:

It clearly has a high bias problem when max depth is 1, because when using the full training set the training and testing errors both remain high while converging.

When max depth is 10 the training and testing errors are no longer converging and the training error is failing to rise as it should as the training set size increases, which indicates a variance problem and overfitting, and not a bias problem.

In [13]:
```
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 max depth and model complexity increases, training error falls to near zero. However as max depth and model complexity increases testing error reaches a baseline minimum at about max_depth=5, and further increases in depth do not reduce testing error.

Testing error reaches a minimum at about max_depth=5, while still maintaining a good level of training error with a minimum of model complexity. Some training error is typically to be expected, and too low a level of training error may indicate overfitting, which would result in a model that does not generalize well. So a max_depth=5 model is probably the best at minimizing model complexity and testing error, while generalizing well to unseen data.

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 [17]:
```
print "Final model optimal parameters:", reg.best_params_
```

```
Final model optimal parameters: {'max_depth': 9}
```

Answer:

The optimum is max_depth=5. This agrees with the learning curves and complexity graph, which makes sense.

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 [15]:
```
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 predicted value of the client's home is $19,327 (which in my real estate market is tiny). It's below the mean and the median home prices for Boston, but within one standard deviation of the mean.

It seems to indicate that the number of rooms being below the mean has lowered the price below the mean, which makes sense, while the neighbors and commute were not good enough to bring the price up enough to overcome the number of rooms.

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:

Well no, I wouldn't use this model because it was trained with data from 23 years ago.

However it does seem to do a reasonably good job of estimating the selling price for a house in the market that it was trained for, so I would consider using a similar model that was trained with current data. Since the market is constantly changing I might consider implementing more of an online learning approach to stay current as the market changes by adding new datapoints to the training set on an ongoing basis. Otherwise I will need to periodically retrain the model with new data.