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 noteboo
k)
%matplotlib inline

# Create our client's feature set for which we will be predicting a sell
ing price
CLIENT_FEATURES = [[11.95, 0.00, 18.100, 0, 0.6590, 5.6090, 90.00, 1.38
5, 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 [5]:

```
# Number of houses in the dataset
total_houses = np.size(housing_prices)

# Number of features in the dataset
total_features = int(housing_features.shape[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!

RM: average number of rooms per dwelling DIS: weighted distances to five Boston employment centres PTRATIO: pupil-teacher ratio by town

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, 33
2.09, 12.13]]
```

RM: 5.609

DIS: 1.385

PTRATIO: 20.2

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 [6]:

```
# 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 subse
ts,
        then returns the training and testing subsets. """
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size
= 0.7, random_state=42)
    # Shuffle and split the data
    X_train = X_train
    y_train = y_train
    X_test = X_test
    y_test = y_test

    # 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_featur
es, housing_prices)
    print "Successfully shuffled and split the data!"
except:
    print "Something went wrong with shuffling and splitting the data."
```

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

Question 4

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

Answer: We would like to have two data sets so that we can test the model we build on unseen data. While both training and testing subsets are expected to be samples of real-world data, we would like to make sure that we are not overfitting our model (i.e. our model must generalize to new data). If our model performs sufficiently well on testing data, we can be sure it will perform similarily to any further real world data as well.

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)](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 [7]:
```
# 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 predicte
d 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: I think Mean Squared Error is the best metric for analyzing error. Accuracy, Precision and Recall are not appropriate for regression type models as it is not obvious would be a correct prediction of a model. Mean Absolute Error is also an appropriate metric, however MSE is more commonly used due to ease of calculation.

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

In [8]:
```
# Put any import statements you need for this code block
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import mean_squared_error, 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
= False)

    # Make the GridSearchCV object
    reg = GridSearchCV(regressor,param_grid = parameters, scoring=scorin
g_function)

    # Fit the learner to the dataset to obtain the optimal model with tu
ned parameters
    reg.fit(X, y)

    # Return the optimal model
    return reg


# Test fit_model
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 algorithm iterates over a list of parameters to return the "optimal parameter" for the model (to be fitted for given data). Optimal parameter is defined as one which returns the best "score". Score can be a user defined function to evaluate the performance of a model.

Question 6

What is cross-validation, and how is it performed on a model? Why would cross-validation be helpful when using grid search?

Answer: Cross validation is a technique to repeatedly partitition (by resampling) the given dataset into training/testing subsets and using the average results of model performance on testing subsets as a measure of model performance on the given dataset. When we use grid-search, we need to significantly increase the size of our data as there is higher likelihood of seeing a model with low error due to random chance (since we are testing many fits for our model, we may accidently overfit). Since our data size is limited by real-world constraints, we can use cross-validation to "simulate" different data sets.

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 [9]:
```
def learning_curves(X_train, y_train, X_test, y_test):
    """ Calculates the performance of several models with varying sizes
of training data.
        The learning and testing error rates for each model are then plo
tted. """

    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 si
zes
    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 w
ith max_depth = depth
            regressor = DecisionTreeRegressor(max_depth = depth)

            # Fit the learner to the training data
            regressor.fit(X_train[:s], y_train[:s])

            # Find the performance on the training set
            train_err[i] = performance_metric(y_train[:s], regressor.pre
dict(X_train[:s]))

            # Find the performance on the testing set
            test_err[i] = performance_metric(y_test, regressor.predic
t(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', fontsi
ze=18, y=1.03)
    fig.tight_layout()
    fig.show()
```

In [13]:
```
def model_complexity(X_train, y_train, X_test, y_test):
    """ Calculates the performance of the model as model complexity incr
eases.
        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_t
rain))

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

    # 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()
```

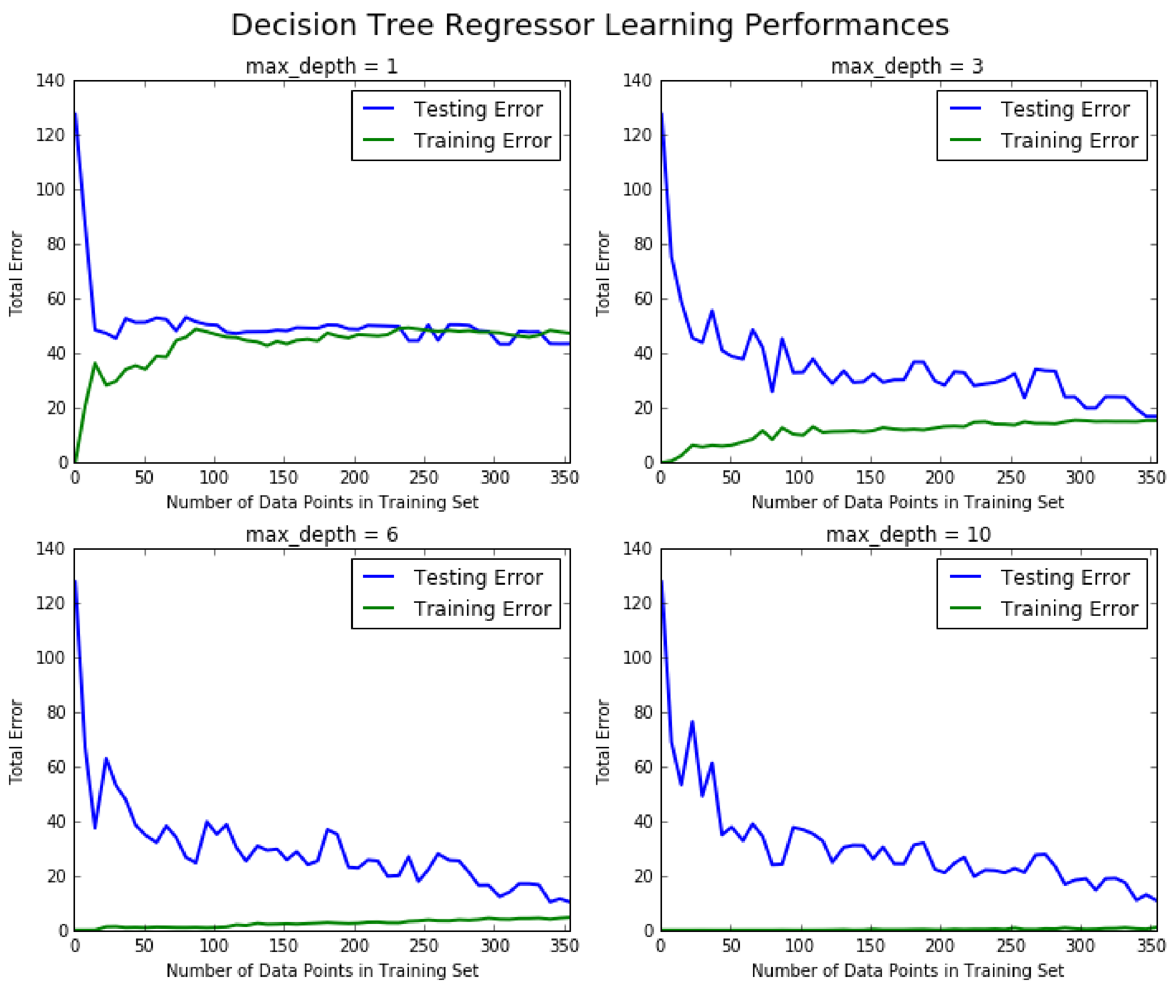
Analyzing Model Performance

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 [14]:
```
learning_curves(X_train, y_train, X_test, y_test)
```

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

C:\Users\systemcenter\Anaconda\lib\site-packages\ipykernel\__main__.p
y:24: DeprecationWarning: using a non-integer number instead of an inte
ger will result in an error in the future
C:\Users\systemcenter\Anaconda\lib\site-packages\ipykernel\__main__.p
y:27: DeprecationWarning: using a non-integer number instead of an inte
ger 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: Let's select max_depth = 6 learning curve. As the size of training set inceases, the training error goes up slowly. For say only 6 data points, the training model would be a perfect fit. As number of datapoints increase, the model can't perfectly fit all the points and the error goes up.

The testing error however decreases as the size of training set increases. As the model sees more data in training, it can better fit further real world examples (as represented by data in testing set).

This model is an example of a good bias-variance tradeoff model as the testing and training error are close to each AND testing error goes down as size of training data goes up.

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 the max depth is 1, the model suffers from high bias as it doesn't learn very well from increasing number of data points. This can be seen from testing error remaining constant with increasing number of points. When the max depth is 10, the model suffers from high variance as it doesn't generalize very well, this can be seen from Testing Error being much more than Training Error.

In [16]:
```
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 increases, training error goes down monotonically since we are allowing a more complex tree to be fitted to limited data. The testing error initially goes down (as we approach from an under-fitted model to appropriate model) but then starts increasing as the model goes from appropriate fit to over-fit.

I think max depth of 6 (or 7) best generalizes the dataset. This model has a good bias-variance trade-off (the testing error is low and is only slightly higher than training error). Based on the learning curves, as the size of the training set increases, the training error increase, however testing error decreases.At max_depth = 6, the testing error is lowest (along with max_depth = 7). While testing error is slightly higher than training error, the different is not very big. I'd select max_depth=6 for my parameter as the resulting tree is simpler than max_depth = 7 (and if two models perform approximately the same, we should pick simpler 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`. *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, 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 [10]:
```
print "Final model optimal parameters:", reg.best_params_
```

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

Answer: The optimal max_depth using grid search is 4. This is lower than my initial intuition but it seems reasonable. From the learning curves, any number for max_depth between 3 and 10 would be reasonable (i.e. there is a small variance error but not too high)

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 [11]:
```
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: My model predicts the value of client's home at $21,630. It is within 1 std deviation of both mean and median prices of the dataset.

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: No. I would not use this mode to predict the selling price of future clients' homes in Greater Boston area due to very high MSE of the model. The MSE of this model for testing dataset is almost 20 which is twice as high as the standard deviation of the original dataset. This seems to be suggest I might just as well predict the mean house price in my data set to be the selling price and still do reasonably well.