In [1]:

```
# Load libraries
import numpy as np
import pylab as pl
from sklearn import datasets
from sklearn.tree import DecisionTreeRegressor

def load_data():
    boston = datasets.load_boston()
    return boston

def explore_city_data(city_data):
    # Get the labels and features from the housing data
    housing_prices = city_data.target
    housing_features = city_data.data
    print 'Size of data is ' + str(len(housing_features))
    print 'Number of features is ' + str(len(city_data.feature_names))
    print 'Minimum price is ' + str(housing_prices.min())
    print 'Maximum price is ' + str(housing_prices.max())
    print 'mean price is ' + str(housing_prices.mean())
    print 'median price is ' + str(np.median(housing_prices))
    print 'standard deviation is ' + str(housing_prices.std())

def split_data(city_data):
    """Randomly shuffle the sample set. Divide it into 70 percent trai
ning and 30 percent testing data."""
    from sklearn import cross_validation
    X, y = city_data.data, city_data.target


    X_train, X_test, y_train, y_test = cross_validation.train_test_spl
it(
          X,y, test_size=0.3, random_state=0)


    return X_train, y_train, X_test, y_test

def performance_metric(label, prediction):
    """Calculate and return the appropriate error performance metric."
""

    # The following page has a table of scoring functions in sklearn:
    # http://scikit-learn.org/stable/modules/classes.html#sklearn-metr
ics-metrics
    from sklearn.metrics import mean_squared_error
    return  mean_squared_error(label, prediction)

def learning_curve(depth, X_train, y_train, X_test, y_test):
    """Calculate the performance of the model after a set of training
```

```
data."""

    # 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))

    print "Decision Tree with Max Depth: "
    print depth

    for i, s in enumerate(sizes):

        # Create and fit the decision tree regressor model
        regressor = DecisionTreeRegressor(max_depth=depth)
        regressor.fit(X_train[:s], y_train[:s])

        # Find the performance on the training and testing set
        train_err[i] = performance_metric(y_train[:s], regressor.predi
ct(X_train[:s]))
        test_err[i] = performance_metric(y_test, regressor.predict(X_t
est))


    # Plot learning curve graph
    learning_curve_graph(sizes, train_err, test_err)

def learning_curve_graph(sizes, train_err, test_err):
    """Plot training and test error as a function of the training size
."""

    pl.figure()
    pl.title('Decision Trees: Performance vs Training Size')
    pl.plot(sizes, test_err, lw=2, label = 'test error')
    pl.plot(sizes, train_err, lw=2, label = 'training error')
    pl.legend()
    pl.xlabel('Training Size')
    pl.ylabel('Error')
    pl.show()

def model_complexity(X_train, y_train, X_test, y_test):
    """Calculate the performance of the model as model complexity incr
eases."""

    print "Model Complexity: "

    # We will vary the depth of decision trees from 2 to 25
    max_depth = np.arange(1, 25)
    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
    model_complexity_graph(max_depth, train_err, test_err)

def model_complexity_graph(max_depth, train_err, test_err):
    """Plot training and test error as a function of the depth of the decision tree learn."""

    pl.figure()
    pl.title('Decision Trees: Performance vs Max Depth')
    pl.plot(max_depth, test_err, lw=2, label = 'test error')
    pl.plot(max_depth, train_err, lw=2, label = 'training error')
    pl.legend()
    pl.xlabel('Max Depth')
    pl.ylabel('Error')
    pl.show()

def fit_predict_model(city_data):
    from sklearn.tree import DecisionTreeRegressor
    """Find and tune the optimal model. Make a prediction on housing data."""

    # Get the features and labels from the Boston housing data
    X, y = city_data.data, city_data.target

    # Setup a Decision Tree Regressor
    regressor = DecisionTreeRegressor()

    parameters = {'max_depth':(1,2,3,4,5,6,7,8,9,10)}

    # 1. Find an appropriate performance metric. This should be the same as the
    # one used in your performance_metric procedure above:
    # http://scikit-learn.org/stable/modules/generated/sklearn.metrics
```

```
.make_scorer.html

    from sklearn.metrics import make_scorer
    from sklearn.metrics import mean_squared_error
    scorer = make_scorer(mean_squared_error, greater_is_better = False
)

    # 2. We will use grid search to fine tune the Decision Tree Regres
sor and
    # obtain the parameters that generate the best training performanc
e. Set up
    # the grid search object here.
    # http://scikit-learn.org/stable/modules/generated/sklearn.grid_se
arch.GridSearchCV.html#sklearn.grid_search.GridSearchCV
    from sklearn.grid_search import GridSearchCV
    grid = GridSearchCV(regressor, parameters, scoring = scorer)
    grid.fit(X,y)
    reg = grid.best_estimator_

    # Fit the learner to the training data to obtain the best paramete
r set
    print "Final Model: "
    print reg.fit(X, y)

    # Use the model to predict the output of a particular sample
    x = [11.95, 0.00, 18.100, 0, 0.6590, 5.6090, 90.00, 1.385, 24, 680
.0, 20.20, 332.09, 12.13]
    y = reg.predict(x)
    print "House: " + str(x)
    print "Prediction: " + str(y)

def main():
    """Analyze the Boston housing data. Evaluate and validate the
    performanance of a Decision Tree regressor on the housing data.
    Fine tune the model to make prediction on unseen data."""

    # Load data
    city_data = load_data()

    # Explore the data
    explore_city_data(city_data)

    # Training/Test dataset split
    X_train, y_train, X_test, y_test = split_data(city_data)

    # Learning Curve Graphs
    max_depths = [1,2,3,4,5,6,7,8,9,10]
    for max_depth in max_depths:
        learning_curve(max_depth, X_train, y_train, X_test, y_test)
    # Model Complexity Graph
    model_complexity(X_train, y_train, X_test, y_test)

    # Tune and predict Model
    fit_predict_model(city_data)


if __name__ == "__main__":
    main()
```

```
Size of data is 506
Number of features is 13
Minimum price is 5.0
Maximum price is 50.0
mean price is 22.5328063241
median price is 21.2
standard deviation is 9.18801154528
Decision Tree with Max Depth:
1
Decision Tree with Max Depth:
2
Decision Tree with Max Depth:
3
Decision Tree with Max Depth:
4
Decision Tree with Max Depth:
5
Decision Tree with Max Depth:
6
Decision Tree with Max Depth:
7
Decision Tree with Max Depth:
8
Decision Tree with Max Depth:
9
Decision Tree with Max Depth:
10
Model Complexity:
Final Model:
```

```
/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:58: Deprec
ationWarning: using a non-integer number instead of an integer will re
sult in an error in the future
/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:61: Deprec
ationWarning: using a non-integer number instead of an integer will re
sult in an error in the future
//anaconda/lib/python2.7/site-packages/sklearn/utils/validation.py:386
: DeprecationWarning: Passing 1d arrays as data is deprecated in 0.17
and willraise ValueError in 0.19. Reshape your data either using X.res
hape(-1, 1) if your data has a single feature or X.reshape(1, -1) if i
t contains a single sample.
  DeprecationWarning)
```

```
DecisionTreeRegressor(criterion='mse', max_depth=4, max_features=None,
           max_leaf_nodes=None, min_samples_leaf=1, min_samples_split=
2,
           min_weight_fraction_leaf=0.0, presort=False, random_state=N
one,
           splitter='best')
House: [11.95, 0.0, 18.1, 0, 0.659, 5.609, 90.0, 1.385, 24, 680.0, 20.
2, 332.09, 12.13]
Prediction: [ 21.62974359]
```

```
In [ ]:
```