P1: Predicting Boston Housing Prices

1) Statistical Analysis and Data Exploration

- Number of data points (houses)? : 506
- Number of features? : 13
- Minimum housing price: 5.0
- Maximum housing price : 50.0
- Mean Boston housing price: 22.5328063241
- Median Boston housing price: 21.2
- Standard deviation: 9.18801154528

2) Evaluating Model Performance

- Which measure of model performance is best to use for predicting Boston housing data and analyzing the errors? Why do you think this measurement most appropriate? Why might the other measurements not be appropriate here?
  Mean Squared Error (MSE) is the best loss function to evaluate the performance of the Boston housing model. The other loss functions are Mean Absolute Error and Median Absolute Error. MSE has the following properties which makes it better than Mean/Median Absolute Errors:
    - MSE is the average of the squared difference between the true training label and the model predicted training label. Calculating the gradient of the below loss function can be done easily which helps while computing the weights of the features associated with the training data. Mean/Median Absolute Errors do not take such forms.

    - By using MSE we end up being able to penalize large errors more heavily as opposed to

$$\text{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2.$$

      uniform weighting in case of Mean/Median Absolute Errors.

- Why is it important to split the Boston housing data into training and testing data? What happens if you do not do this?
  We require a separate training and test set to ensure that the Boston housing model can be evaluated impartially by using the test set only. If the test set were to be used for in the training, the final model would have been "fitted" over all the available data and hence impartial evaluation of the model cannot be performed without separate training and test data.

- What does grid search do and why might you want to use it?
  Grid search does an exhaustive search over a set of hyper-parameters(max_depth) which dictate model complexity which in our case is the set of 'max_depth' to find the best value of max_depth.

- Why is cross validation useful and why might we use it with grid search?
  Cross Validation(CV) is the process of validating the performance of the trained model and finding suitable hyper-parameters 'max_depth' on a separate dataset called the Cross Validation Dataset before it is used to make predictions on the Test dataset. Cross Validation is performed multiple times and the results averaged to find the best hyperparameters. Since there are 10 possible values for “max_depth” from which Grid Search has to choose the optimum one, we require a CVdataset on which GridSearch can operate to find and evaluate the best hyper-parameter 'max_depth' along with the trained model which does not underfit or overfit the training data and generalizes better to the unseen Test data.

3) Analyzing Model Performance

- Look at all learning curve graphs provided. What is the general trend of training and testing error as training size increases?
  As the training size increases, the training error increases at a very very low rate. The training error also decreases as when the model complexity is increasing.
  As the training size increases, the test error decreases rapidly at first and then gradually. The maximum test error also decreases as when the model complexity is increasing.

- Look at the learning curves for the decision tree regressor with max depth 1 and 10 (first and last learning curve graphs). When the model is fully trained does it suffer from either high bias/underfitting or high variance/overfitting?
  With max_depth =1, the model suffers from high bias/underfitting as apparent from the high error values from the training and test dataset.
  With max_depth = 10(when the model is fully trained), the model suffers from high variance/overfitting as apparent from the extremely low error during training and difference between train and test errors.

- Look at the model complexity graph. How do the training and test error relate to increasing model complexity? Based on this relationship, which model (max depth) best generalizes the dataset and why?
  The model with max_depth = **4** best generalizes the dataset. After that, the training error continues to decrease while the test error increases/fluctuating.

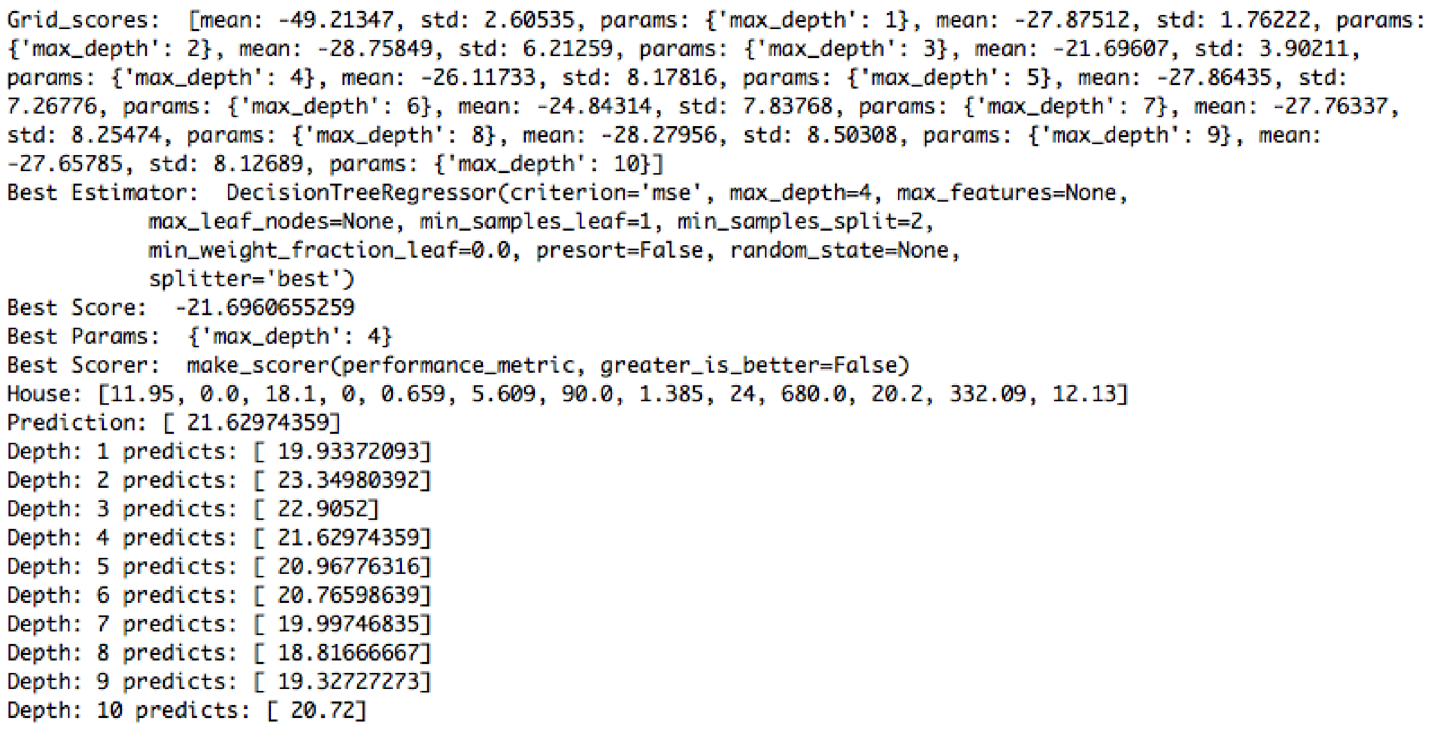
  As the model complexity increases, the training error decreases abruptly and then levels off to almost 0 indicating overfitting. The test error on the other hand decreases up-to the optimal model complexity point and after that it ends up fluctuating due to high variance.

4) Model Prediction

- Model makes predicted housing price with detailed model parameters (max depth) reported using grid search. Note due to the small randomization of the code it is recommended to run the program several times to identify the most common/reasonable price/model complexity.
- Compare prediction to earlier statistics and make a case if you think it is a valid model.
  The model makes a prediction of **21.629** by choosing a best max_depth of **4**. After that, the model starts overfitting for increasing complexity.

Given the following statistics from the raw “city_data” dataset, our model does a good job of predicting a value of **21.629** which is close to the mean and median of the housing prices.

- Mean Boston housing price: **22.5328063241**
- Median Boston housing price: **21.2**
- Standard deviation: **9.18801154528**



```
Grid_scores:  [mean: -49.21347, std: 2.60535, params: {'max_depth': 1}, mean: -27.87512, std: 1.76222, params:
{'max_depth': 2}, mean: -28.75849, std: 6.21259, params: {'max_depth': 3}, mean: -21.69607, std: 3.90211,
params: {'max_depth': 4}, mean: -26.11733, std: 8.17816, params: {'max_depth': 5}, mean: -27.86435, std:
7.26776, params: {'max_depth': 6}, mean: -24.84314, std: 7.83768, params: {'max_depth': 7}, mean: -27.76337,
std: 8.25474, params: {'max_depth': 8}, mean: -28.27956, std: 8.50308, params: {'max_depth': 9}, mean:
-27.65785, std: 8.12689, params: {'max_depth': 10}]
Best Estimator:  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=None,
           splitter='best')
Best Score:  -21.6960655259
Best Params:  {'max_depth': 4}
Best Scorer:  make_scorer(performance_metric, greater_is_better=False)
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]
Depth: 1 predicts: [ 19.93372093]
Depth: 2 predicts: [ 23.34980392]
Depth: 3 predicts: [ 22.9052]
Depth: 4 predicts: [ 21.62974359]
Depth: 5 predicts: [ 20.96776316]
Depth: 6 predicts: [ 20.76598639]
Depth: 7 predicts: [ 19.99746835]
Depth: 8 predicts: [ 18.81666667]
Depth: 9 predicts: [ 19.32727273]
Depth: 10 predicts: [ 20.72]
```