Predicting Boston Housing Prices

by Himanshu Panwar , Udacity Machine Learning Nanodegree , PROJECT 1

Report Structure

- Statistical Analysis and Data Exploration

1.Number of data points(houses)?
    no of data points : 506
2.Number of features?
    no of features : 13
3.Minimum and Maximum House Price?
    maximum price : 50
    minimum price : 5
4.Mean and Median Boston Housing Prices?
    mean of prices : 22.53
    median of prices : 21.2
5.Standard Deviation?
    standard deviation : 9.19

```
In [6]: '''Explore the city'''
        explore_boston(city_boston)

        no of houses : 506
        no of features : 13
        minimum price : 5.0
        maximum price : 50.0
        mean of prices : 22.5328063241
        median price : 21.2
        standard deviation of prices : 9.18801154528
```

-Evaluating Model Performance

1.Which measure of model performance is best to use for predicting Boston housing data and analyzing the errors?Why do you think this measurement is more appropriate?Why might the other measurement not be appropriate here?

I have checked two measures for model performance :
1. mean absolute error(MAE) and
2. root mean squared error(RMSE)
We use error metrics to get relative distance between predicted and actual labels, and the ones which does not return negative values that cancels out positive values.
Both of the metrics performance were equally well.
RMSE has some more properties: It gives relatively high weights on large errors, it punishes large errors than the low errors. This means MSE is most useful when large errors are undesirable .
Therefore in the code RMSE was used.

2.Why is important to split the Boston housing data into training and testing data?What happens if you do not use this?

We split the data to training and testing data , and train the training data and check the performance on testing data how well the model is performing by checking the errors between prediction and actual labels of testing data.
If we do not use it and train the entire data , it implies that the model has seen all the data, it will overfit and will perform bad on unseen data.

3.What does the grid search do and why you want to use it?

We use grid search to fine tune the Decision Tree Regressor and obtain the parameters that generate the best training performance. We define set of parameter values that we want to try on the model and then use cross validation to evaluate every possible combination of those values to choose the best between them.
We use grid search to efficiently search the optimal parameters to maximize the performance of the model.

4.Why is cross validation useful and why might we use it with grid search?

Cross-validation is a technique to evaluate predictive models by partitioning the original sample into a training set to train the model and a test set to evaluate it.

In k-fold cross-validation, the data is randomly partitioned into k equal size sets.

From the k sets, single set is used as validation for testing and others for training.

Then the cross validation is repeated k times, for each sets used exactly once for validation.

The k results then can be averaged to produce single estimation.

Advantage: All observations are used for both training and validation.

The k-fold is more reliable than train-test split as it reduces the variance associated with a single split.

Grid Search CV allows us to find the set of parameters to try on model and it automatically run cross-validation using each of those parameters keeping track of resulting score.

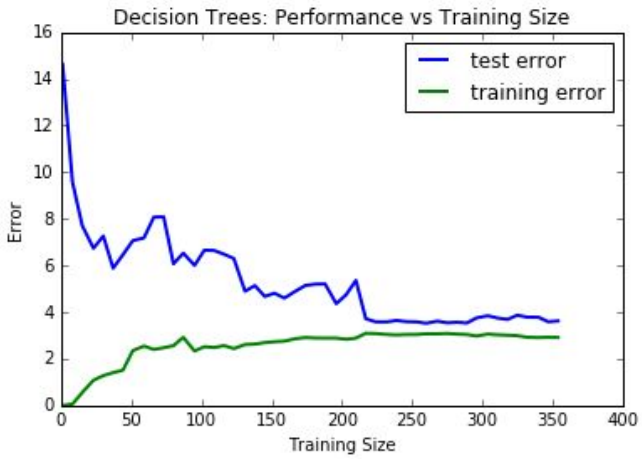
-Analyzing Model Performance

1.Look at the learning curve graphs provided. What is the general trend of training and testing errors as training size increases?

As per the learning curves, the training error increases as training size increases.

And the testing error curve decreases as in increase in training size.

```
Decision Tree with Max Depth:
4
```



2.Look at the learning curves for the decision tree regressor with max depth 1 and 10. When the model is fully trained does it suffer from either high bias/underfitting or high variance/overfitting?

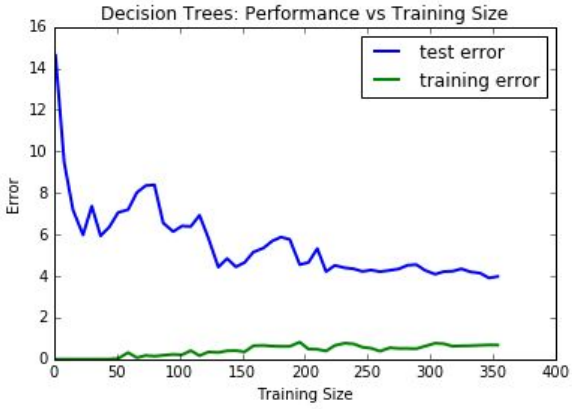
The learning curve with max depth 1 suffers from high bias/underfitting as it is not doing well with training set , the training and testing error are high .

And the graph with depth 10 is overfitting because there is large gap between the training and training error is almost 0.

Tree Depth : 1

Tree Depth : 10

```
Decision Tree with Max Depth:
10
```

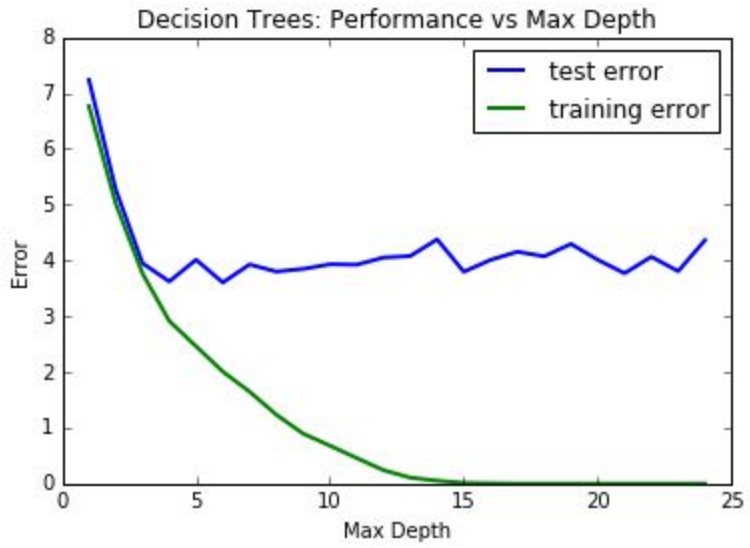


3.Look at the model complexity graph .How do the training and test error relate to increasing model complexity? Based on this relationship, which model(max_depth) best generalizes the data set and why?

As per the graph , the increase in the max depth leads to decrease in the training error to almost 0 and the testing error decreases and then around 4 constant .

Based on the graph , the depth with 4 best generalizes the data set , the minimum of test error is around 4 and training error is also not overfitting and underfitting .

Model Complexity:



As per use of :

```
print "Best model parameter:  " + str( reg.best_params_)
```

Results in : Best Parameters: {'max_depth': 4}

(Result in next picture below).

-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 feature set:
[11.95, 0.00, 18.100, 0, 0.6590, 5.6090, 90.00, 1.385, 24, 680.0, 20.20, 332.09, 12.13]
Prediction: [21.62974359] greater than nearest neighbors average.
The predicted value is near to the mean value.
Z-score: –0.098

```
In [17]: '''Tune and predict Model'''
         fit_predict_model(city_boston)

[Parallel(n_jobs=1)]: Done   1 jobs       | elapsed:    0.0s
[Parallel(n_jobs=1)]: Done  30 out of  30 | elapsed:    0.1s finished

Final Model:
Fitting 3 folds for each of 10 candidates, totalling 30 fits
GridSearchCV(cv=None, error_score='raise',
       estimator=DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
           max_leaf_nodes=None, min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, random_state=None,
           splitter='best'),
       fit_params={}, iid=True, loss_func=None, n_jobs=1,
       param_grid={'max_depth': (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)},
       pre_dispatch='2*n_jobs', refit=True, score_func=None,
       scoring=make_scorer(performance_metric, greater_is_better=False),
       verbose=True)
Best Parameters:  {'max_depth': 4}
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]
```

As per nearest neighbors average price:

```
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]
X=city_boston.data
from sklearn.neighbors import NearestNeighbors
def find_nearest_neighbor_indexes(x, X):  # x is your vector and X is the data set.
    neigh = NearestNeighbors( n_neighbors = 10 )
    neigh.fit( X)
    distance, indexes = neigh.kneighbors( x )
    return indexes
indexes = find_nearest_neighbor_indexes(x, X)
sum_prices = []
for i in indexes:
    sum_prices.append(city_boston.target[i])
neighbor_avg = np.mean(sum_prices)
print "Nearest Neighbors average: " +str(neighbor_avg)
```

```
Nearest Neighbors average: 21.52
```