Nadim Farhat 02/08/2016

Udacity

Machine Learning engineer Nano Degree

Acknowledgements :

I acknowledge that the feedback and reviews from Udacity teams were essential for the writing and completion of this project. especially many Thanks to Luca .

Nadim

Please note the code will be italic and blue and indented

```
pythonCode goes here
```

1. Objectives

The task of the project is to use statistical analysis tools to build the best model to predict the value of a given house. Predicting the house price for a house with a finite number of features will help clients in selling the house. Ideally the best predicted price of a house comes from the model best generalize the data .

2. Data

The data is provided by the sklearn library in python . The data summary and the data distribution are shown in Table 1 and Figure 1 respectively:

Samples total	506
Dimensionality	13
Features	real, positive
Targets	real 5. - 50.

Table 1: Boston Price Data summary

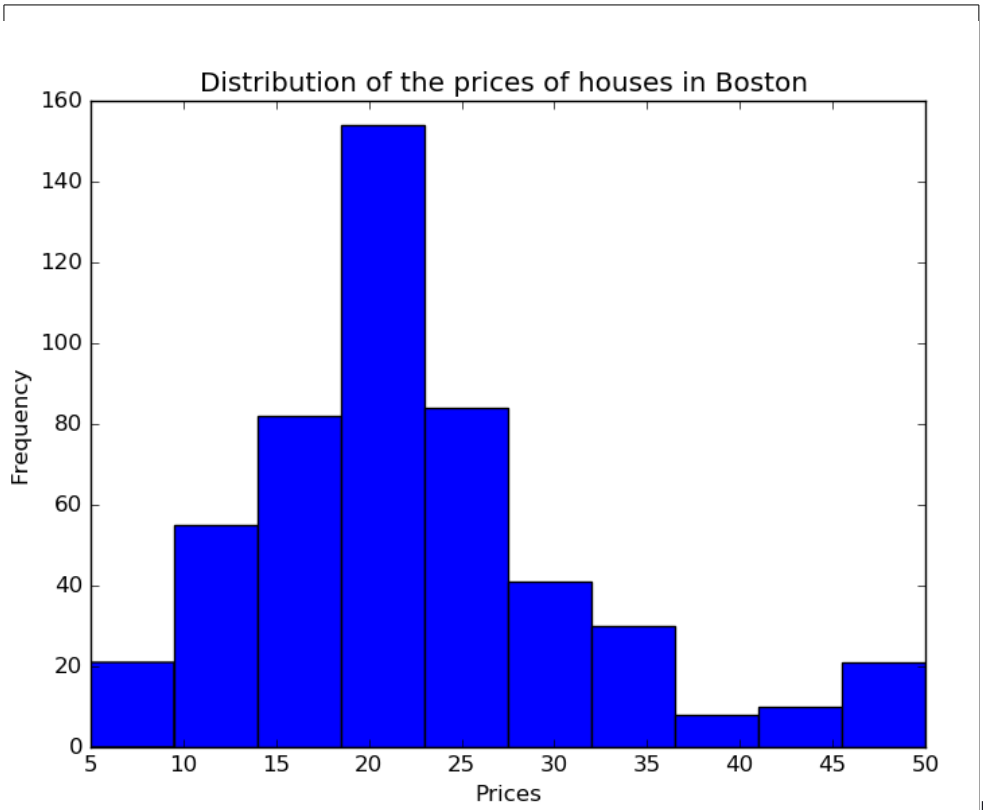

Figure 1: Distribute of the prices of houses in Boston

3. Methods

The project uses python as a programming languages with add-on libraries(*numpy , scipy , sklearn , matplotlib*) . Python developing time is short ,also the course instructors provided a template “boston_housing.py”, this template is available at the Udacity.com for download. The modified *boston_housing.py* satisfies the requirements of this project and it lives in the git directory

machinelearning/machinlearning_udacity/project-1

Terms Definitions :

Over fitting / High Variance : A model with a high variance or over fitting is a model that had a high performance on fitting the training data and is specific for that training data, therefore is not able to generalize to any new predictions.
Under fitting / high Bias : A model with a High bias or underfitting is a model disregards the input data .
Model Complexity : it is the trade-off between over-fitting and under-fitting where the model will have low variance and low bias.
Model Generalizing : is a model that able to predict any new input data with high performance.
Model Performance : is a metric that can be used to check how well a model generalize the data.

1. Statistical Analysis and Data Exploration

There are 506 houses in the total data set and each house has 13 features. The house prices range from 5 to 50 and with median and mean prices 23 and 21 respectively . The standard deviation from the mean is 9 . Table 2 summarizes the statistics.

Number of Houses	506
Number of Features	13
Maximum Price	50
Min Prices	5
Mean Price	22.53
Median Price	21.2
Standard Deviation of the price	9.2

Table 2: Boston Prices Statistics summary

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?.

The metrics library in sklearn provides performance metrics to assess the model performance. The regression part of the library provides 3 errors metrics . In the statistical analysis and data exploration section , the data shows that the mean is almost equal to the median prices(22.53 and 21.2) which suggests that the outliers effect on the model is small. Therefore the absolute median error is not applicable in this project. Compared to the mean absolute value the mean of squared error is the preferred metric for regression because it is augments any large error by square, therefore it might provide better insight to the model performance than the mean absolute value error .

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

In general splitting the data between training and testing data have two purposes :

1. Assessing the Bias : Once the model is trained , there is a need to check if the model is able to generalize to data. therefore testing the model with new test data shows if the model able to fit to the new data and not under fit (under fitting means the model stays the same regardless of the input).
2. Assessing the variance : to make sure that the model is not specific for particular data.

Decision trees keep creating nodes until it fits all the traning data therefore it has tendency to over fitt the data, If the data is not split between traning and testing ,

the decision trees may over fitt the model and this model can't be validated for bias and variance because training data can't be used for validation.

What does grid search do and why might you want to use it?

The grid search basically returns the optimized model for a given set of parameters using the performance metric as cost function (minimize in case of MSE) . Grid search helps to avoid manual search of the space for the best parameters . We can have multiple parameters and for each parameters we can have more than one value therefore the search space can be too big for human. It must be noted that make_scorer in sklearn takes a default boolean argument greater_is_better = True , this parameter must be set to False so the GridSearchCV can find the best parameter of a model of a minimal MSE .

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

In Machine learning Training data must be different than testing data because once we train our model we need to validate it with data that were not used to build its model. Therefore the data used for training can't be used for testing or prediction. Some supervised learning algorithm require a lot of data for training therefore it will be challenging if all this data is held for training only. Cross validation is a method to split the data between training and testing into folds or parts. The part that is used for training the model in one iteration is validated against the other testing parts. then the same data used in the previous iteration as training set will be used in an second iteration as part of the testing data set . We keep iterating this procedure over the number of parts or folds, the performance metrics from the iterations can be averaged together . thus the benefits of cross validation :

- Maximize Data usage.
- Assessing the over fitting and under-fitting.

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?

The trend of the testing error decrease as the the training size increase to approximately as size of 50 but for training sizes > 50 the error converges. The training error increase as the training size increase until in converges for values > 200 of training size. The error of training and test decrease as the tree depth increase . The precedent observations applies to all depth of the tree (1-10) , Figure 2, and Figure 3 shows the trend of the test error and training errors vs training size for tree depth of 1 and 10 respectively

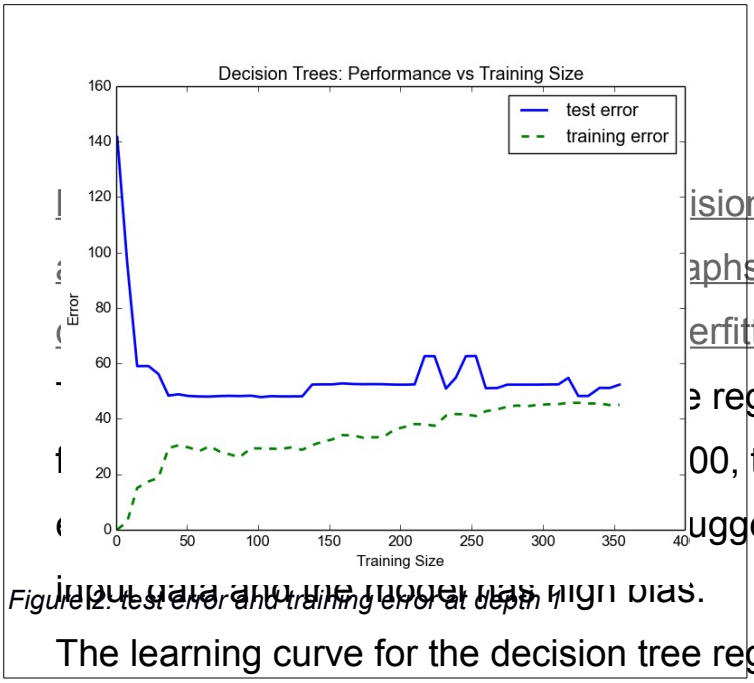

Figure 2: test error and training error at depth 1

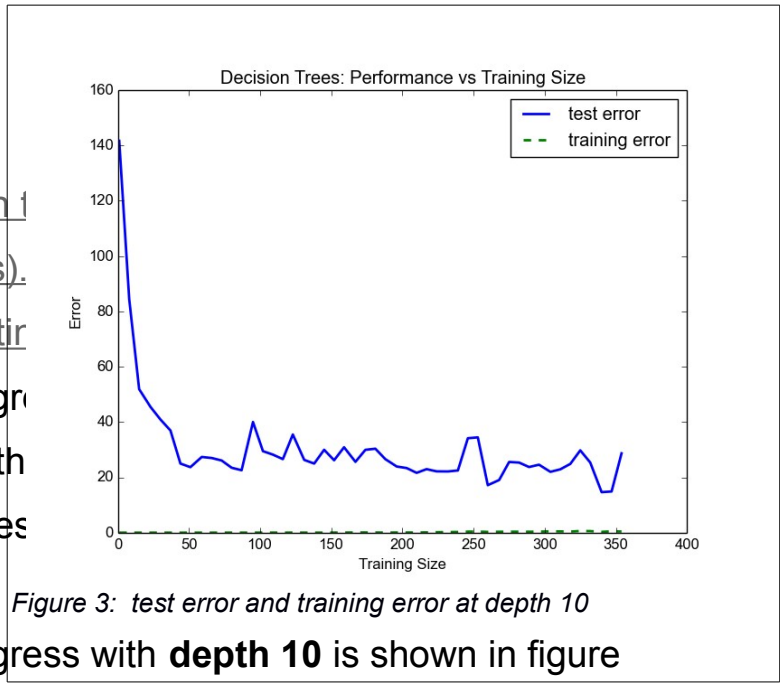

Figure 3: test error and training error at depth 10

The learning curve for the decision tree regress with **depth 10** is shown in figure 3 above , the error of the model in the training set is much smaller than then error of the model in the testing set , this suggests that the model is too specific to the training set and it is overfitting the data.

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?

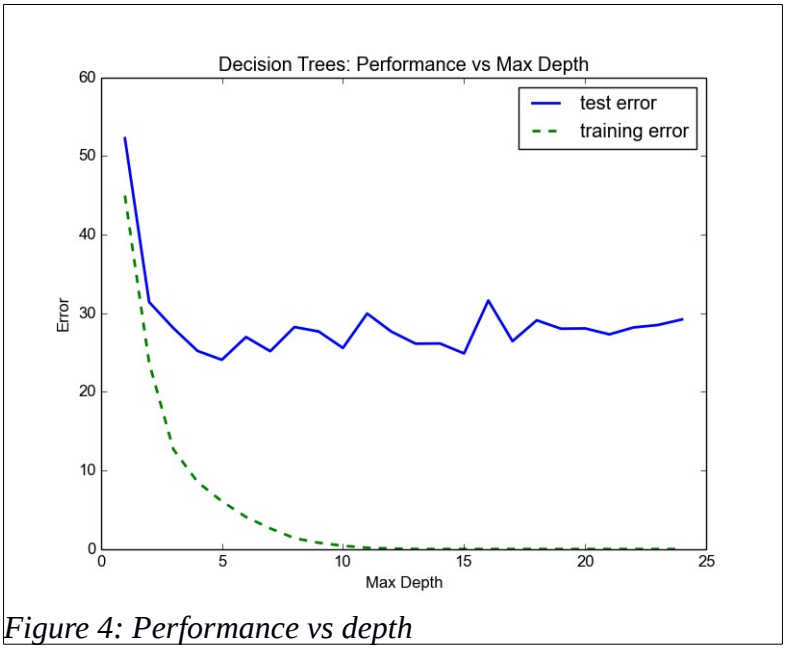

Figure 4: Performance vs depth

From the looks of the graph the training error keep decreasing with the depth increase which is expected from a decision tree as it tries to be more and more specific to the training set, also the test error have a sharp decline for depth between 0 and 5. After this sharp decline the test error seems to reach its minimum and it fluctuates at depth > 5 , meanwhile the training error keep decreasing which suggests an overfitt for the data. From the graph I can induce that a depth of 5 or 6 are optimal depths to build a model that generalized a data.

4. Model Prediction

For the provided house feature the model returns a prediction of 20.8 and a max depth of 6.

Prediction: [20.76598639]

Best model parameter: {'max_depth': 6}

After running the model for several times I found the predictions are within one standard deviation of the mean . The model is adequate for estimating house prices . Also the best r2(testing) picked from the graph of the learning curve was 0.8 which also confirms that the model is adequate to estimate prices, finally compared to the mean(21) and median(23) the prediction of the model is very close to both of these statistics.

Pro tips

(Suggested by Luca Udacity reviewer)

What does grid search do and why might you want to use it?

For data sets with a big number of parameters searching the full parameters space might be slow therefore RandomizedSearchCV can be faster by not search the whole parameters space instead it sample the parameters from the specified distribution

5. Model Prediction(complement/Alternative)

Get the average prices of all the neighbors for each feature associated with predicted price and compare it to each to the predicted price,

using K-neighbors unsuprvised learning algorithm .

Coding Appendix

STEP 1

- Please calculate the following values using the numpy library. The array housing_prices is basically the output (ytrue) therefore it is enough to get the length of the array of housing_prices :

  sizeOfData = len(housing_prices) simirarly we can use

  sizeOfData = np.shape(housing_prices)

Therefore the number of houses = 506

- Number of features , the housing_features is a 2D array where the number of rows represents the number of houses and the number of columns represent the number of features therefore

```
featuresArrayShape = np.shape(housing_features);
numberOfFeatures = featuresArrayShape[1];
```

- As for the other statstics we use the array of housing_prices and using numpy we run

```
minimumPrice = np.min(housing_prices);
maximumPrice = np.max(housing_prices);
meanPrice = np.mean(housing_prices);
medianPrice = np.median(housing_prices);
stdPrice = np.std(housing_prices);
```

STEP 2

```
r2 =  0
meanSquaredError = 0
r2  = metrics.r2_score(label,prediction)
#meanSquaredError = metrics.mean_squared_error(label,prediction)
#return meanSquaredError
return r2;
```

STEP 3

in This step we need to split our data between training and testing data and it was asked to have 30 precent of testing data.

first we need to import the import the cross_validation functionality thus we

import the cross_validation lib

from sklearn import cross_validation

and then we type the following command

X_train, X_test , y_train, y_test =

cross_validation.train_test_split(X,y,test_size= 0.3)

and we return the results in the same order

return X_train, X_test , y_train, y_test

STEP 4

I initialized the metric_scorer to use the same metric in step 3

#mse_Scorer = metrics.make_scorer(metrics.mean_squared_error);

r2_Scorer = metrics.make_scorer(metrics.r2_score);

arguments for the function GridSearchCV

reg =

GridSearchCV(regressor,parameters,mse_Scorer,greater_is_better=false);

Pro tips code

(Suggested by Luca Udacity reviewer)

```
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_data.target[i])
neighbor_avg = np.mean(sum_prices)
print "Nearest Neighbors average: " +str(neighbor_avg)
```