A

Internship project Report

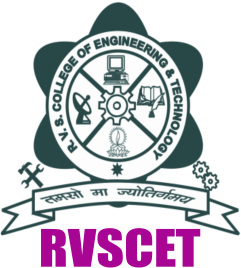
On

***"BOSTON HOUSE PRICE PREDICTION*”**

***BY***

**(NAVIN KUMAR)**

**University Roll No - (18013440040)**



**Department Of Computer Science & Engineering**

**R.V.S. College Of Engineering & Technology**

**Jamshedpur – 831012**

**2018 - 2022**

**ACKNOWLEDGEMENT**

I avail this opportunity to express our profound sense of gratitude to Asst. Prof. Manjeet Singh for his constant supervision, guidance and encouragement right from the beginning till the completion of the project.

Also I would like to give explicit thanks to the team of **Skyfi Labs** course of (**Machine Learning With Python**) for designing a compatible and easy way to understand the course and project.

We wish to thank all the faculty members and supporting staffs of Dept. of Computer Science and Engineering for their valuable suggestions, timely advice and encouragement.

**Abstract**

You want to be the best real estate agent out there. In order to compete with other agents in your area, you decide to use machine learning. You are going to use various statistical analysis tools to build the best model to predict the value of a given house. Your task is to find the best price your client can sell their house at. The best guess from a model is one that best generalizes the data.

For this assignment your client has a house with the following 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]. To get started, use the example scikit implementation. You will have to modify the code slightly to get the file up and running.

## Statistical Analysis and Data Exploration

Loading the dataset:

from sklearn import datasets

city\_data = datasets.load\_boston()

# Get the labels and features from the housing data

housing\_prices = city\_data.target

housing\_features = city\_data.data

Let us know begin with the exploration of data using numpy

* Number of data points (houses)?

number\_of\_houses = housing\_features.shape[0]

print "number of houses:",number\_of\_houses

number of houses: 506

* Number of features?

number\_of\_features = housing\_features.shape[1]

print "number of features:",number\_of\_features

number of features: 13

* Minimum and maximum housing prices?

max\_price = np.max(housing\_prices)

min\_price = np.min(housing\_prices)

print "max price of house:",max\_price

print "min price of house:",min\_price

max price of house: 50.0 min price of house: 5.0

* Mean and median Boston housing prices?

mean\_price = np.mean(housing\_prices)

median\_price = np.median(housing\_prices)

print "mean price of house:",mean\_price

print "median price of house:",median\_price

mean price of house: 22.5328063241 median price of house: 21.2

* Standard deviation?

standard\_deviation = np.std(housing\_prices)

print "standard deviation for prices of house:",standard\_deviation

standard deviation for prices of house: 9.18801154528

## 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?

In my previous iteration of the work I used r^2 but as the project demands a measure if the error, I believe using the mean squarred error would give us the best solution. The reason being MSE emphasises large error unlike absolute mean or median error. This is what is seen as ideal by most statisticians. However, another important aspect to see in the case of this project would be the measur that minimises the error the most. Therefore just the thought is not enough and we would need to see some kind of graphical proof for this.

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

One of the biggest problems that could occur is overfitting or underfitting. If we were to not split the dataset, it would imply that we have used all our data for training and there is a high chance that our algorithm becomes tailor made to the given dataset. Thus, there could be a huge dip in the performance of the algorihtm on the test dataset.

Some of the advantages that splitting the dataset gives us are as follows:

* It allows us to check if we have undefit or overfit the data.
* It provides a sense of validation for our model.
* It suggests the level of performance measure one would expect from the test data. Thus, we are better prepared for the unknown data.

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

Grid search helps in parameter tuning and the selecetion of appropriate model based on the parameters we wish to tune. While computing this, it also implemets cross validation folds, thus eliminating the risk of overfitting/underfitting. It also has the flexibility of making a customised scorer function.

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

The goal of cross validation is to define a dataset to "test" the model in the training phase (i.e., the validation dataset), in order to limit problems like overfitting, give an insight on how the model will generalize to an independent dataset (i.e., an unknown dataset, for instance from a real problem), etc.

Gird search performs 3-fold cross validation, hence we can easily validate the optimized parameter model generated by it. Cross validation ensures that we have a sufficiently good model and that we don't over or underfit the data in any way. Since, the gridsearch provides parameter optimization and uses crosss validation to provide the optimal parameters, we can be sure that there are no significant problems of underfitting or overfitting.

## 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 error reduces and the model begins to predict better. This can be seen form the learning curves. With the increase in max depth of the decision tree and training size, the training error is almost 0 whereas the testing error continues to reduce. Thus, there is improvement in the regression model as training size increases.

This however, is the general trend for higher depths. In the beginning, we see that both the training and the testing error flattens out. Thus, as the training size increases there are errors in the testing dataset that increases.

* 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?

Lets analyze the scenario for max depth being 1:

In this case, we are underfitting the dataset, both the training and test dataset has high errors and flattens out after a while. Thus, there is a high amount of bias when using max depth = 1. This is somewhat intuituive as we do not let the tree expand and we restrict the entropy or the learning value. Hence, with a depth levelof 1 we cannot the complexity of the dataset.

Lets analyze the scenario for max depth being 10:

Here, we are clearly overfitting. Using the same discussion above, since we allow increase in depth upto level 10, we are increasing the entropy levels. Thus, we see an error of 0 for training dataset. However, the testing data begins to flatten out with a few occassional spikes.

* 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?

I think a max\_depth between 5 generalises the dataset the best. It doesn't overfit as in the case of higher depths, we simply get a training error of 0. This means we are completly fitting the data and there is a high chance of overfitting. The model complxity graph verifies the above conclusion

## 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.

Final Model:

DecisionTreeRegressor(criterion='mse', max\_depth=5, max\_features=None,

max\_leaf\_nodes=None, min\_samples\_leaf=3, min\_samples\_split=1,

min\_weight\_fraction\_leaf=0.0, random\_state=None,

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: [ 20.96776316]

The final decision tree regressor uses the following optimized parameters:

* max\_depth:5
* min\_samples\_leaf = 3
* min\_samples\_split = 1
* Compare prediction to earlier statistics and make a case if you think it is a valid model.

The central tendency for the given dataset with respect to the mean and the median are as follows: mean price of house: 22.5328063241 median price of house: 21.2

This means that our current prediction for the given house is credible. Therefore we can say that we have a model that fits the data to an extent.

However, I would say that the model still doesn't completely describe the variance in the dataset. If we plot the graph of residuals, we get a cycle like graph. This means that a linear mode won't be able to generalise the data. We are underfitting the dataset.

**CODING**

*"""Load the Boston dataset and examine its target (label) distribution."""  
  
# Load libraries***import** numpy **as** np  
**import** pylab **as** pl  
**from** sklearn **import** datasets  
**from** sklearn.tree **import** DecisionTreeRegressor  
  
*################################  
### ADD EXTRA LIBRARIES HERE ###  
################################***from** sklearn.metrics **import** mean\_squared\_error,median\_absolute\_error,r2\_score,mean\_absolute\_error  
**from** sklearn **import** grid\_search  
**from** sklearn.cross\_validation **import** train\_test\_split  
  
**def** load\_data():  
 *"""Load the Boston dataset."""* boston = datasets.load\_boston()  
 **return** boston  
  
  
**def** explore\_city\_data(city\_data):  
 *"""Calculate the Boston housing statistics."""  
  
 # Get the labels and features from the housing data* housing\_prices = city\_data.target  
 housing\_features = city\_data.data  
  
 *###################################  
 ### Step 1. YOUR CODE GOES HERE ###  
 ###################################  
  
 # Please calculate the following values using the Numpy library  
 # Size of data (number of houses)?  
 # Number of features?  
 # Minimum price?  
 # Maximum price?  
 # Calculate mean price?  
 # Calculate median price?  
 # Calculate standard deviation?* number\_of\_houses = housing\_features.shape[0]  
 number\_of\_features = housing\_features.shape[1]  
 max\_price = np.max(housing\_prices)  
 min\_price = np.min(housing\_prices)  
 mean\_price = np.mean(housing\_prices)  
 median\_price = np.median(housing\_prices)  
 standard\_deviation = np.std(housing\_prices)  
  
 print **"number of houses:"**,number\_of\_houses  
 print **"number of features:"**,number\_of\_features  
 print **"max price of house:"**,max\_price  
 print **"min price of house:"**,min\_price  
 print **"mean price of house:"**,mean\_price  
 print **"median price of house:"**,median\_price  
 print **"standard deviation for prices of house:"**,standard\_deviation  
  
**def** performance\_metric(label, prediction):  
 *"""Calculate and return the appropriate error performance metric."""  
  
 ###################################  
 ### Step 2. YOUR CODE GOES HERE ###  
 ###################################  
  
 # http://scikit-learn.org/stable/modules/classes.html#sklearn-metrics-metrics  
 #return median\_absolute\_error(label, prediction)  
 #return r2\_score(label, prediction)  
 #return mean\_absolute\_error(label, prediction)* **return** mean\_squared\_error(label,prediction)  
 **pass  
  
  
def** split\_data(city\_data):  
 *"""Randomly shuffle the sample set. Divide it into 70 percent training and 30 percent testing data."""  
  
 # Get the features and labels from the Boston housing data* X, y = city\_data.data, city\_data.target  
  
 *###################################  
 ### Step 3. YOUR CODE GOES HERE ###  
 ###################################* X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 X, y, test\_size=0.30, train\_size=0.70, random\_state=42)  
 **return** X\_train, y\_train, X\_test, y\_test  
  
  
**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.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.predict(X\_train[:s]))  
   
 test\_err[i] = performance\_metric(y\_test, regressor.predict(X\_test))  
  
   
  
 pl.figure()  
 pl.plot(y\_train - regressor.predict(X\_train))  
 pl.savefig(**"residual\_plot.png"**)  
 *# Plot learning curve graph* learning\_curve\_graph(sizes, train\_err, test\_err, depth)  
  
  
**def** learning\_curve\_graph(sizes, train\_err, test\_err, depth):  
 *"""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()* pl.savefig(**"learning\_curve"**+**"\_"**+str(depth)+**".png"**)  
  
  
**def** model\_complexity(X\_train, y\_train, X\_test, y\_test):  
 *"""Calculate the performance of the model as model complexity increases."""* 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()* pl.savefig(**"model\_complexity.png"**)  
  
  
**def** fit\_predict\_model(city\_data):  
 *"""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),  
 **'min\_samples\_split'**: (1, 2, 3),  
 **'min\_samples\_leaf'**: (1, 2, 3)  
 }  
  
 *###################################  
 ### Step 4. YOUR CODE GOES HERE ###  
 ###################################  
  
 # 1. Find the best performance metric  
 # should be the same as your performance\_metric procedure  
 # http://scikit-learn.org/stable/modules/generated/sklearn.metrics.make\_scorer.html  
  
 # 2. Use gridearch to fine tune the Decision Tree Regressor and find the best model  
 # http://scikit-learn.org/stable/modules/generated/sklearn.grid\_search.GridSearchCV.html#sklearn.grid\_search.GridSearchCV* regressors = grid\_search.GridSearchCV(regressor, parameters, scoring=**'mean\_squared\_error'**)  
  
 regressors.fit(X,y)  
  
 *# pick the best* reg = regressors.best\_estimator\_  
  
 *# Fit the learner to the training data* 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()

**OUTPUT**

** **

**Learning Curve:1 Learning Curve:2**



**Model Complexity**

Conclusion

I here by conclude my project on the topic Boston House Price Prediction that is calculated by machine learning language using python.

I have provided with the used code and the visible output and the result is within satisfactory range there are a a good quite number of improvements on the project that are possible in this project like improving the user interface adding a few more modules also a few more features and we can easily predict the prices in much more easier and simpler form with the help of Machine Learning.