**Linear Regression**

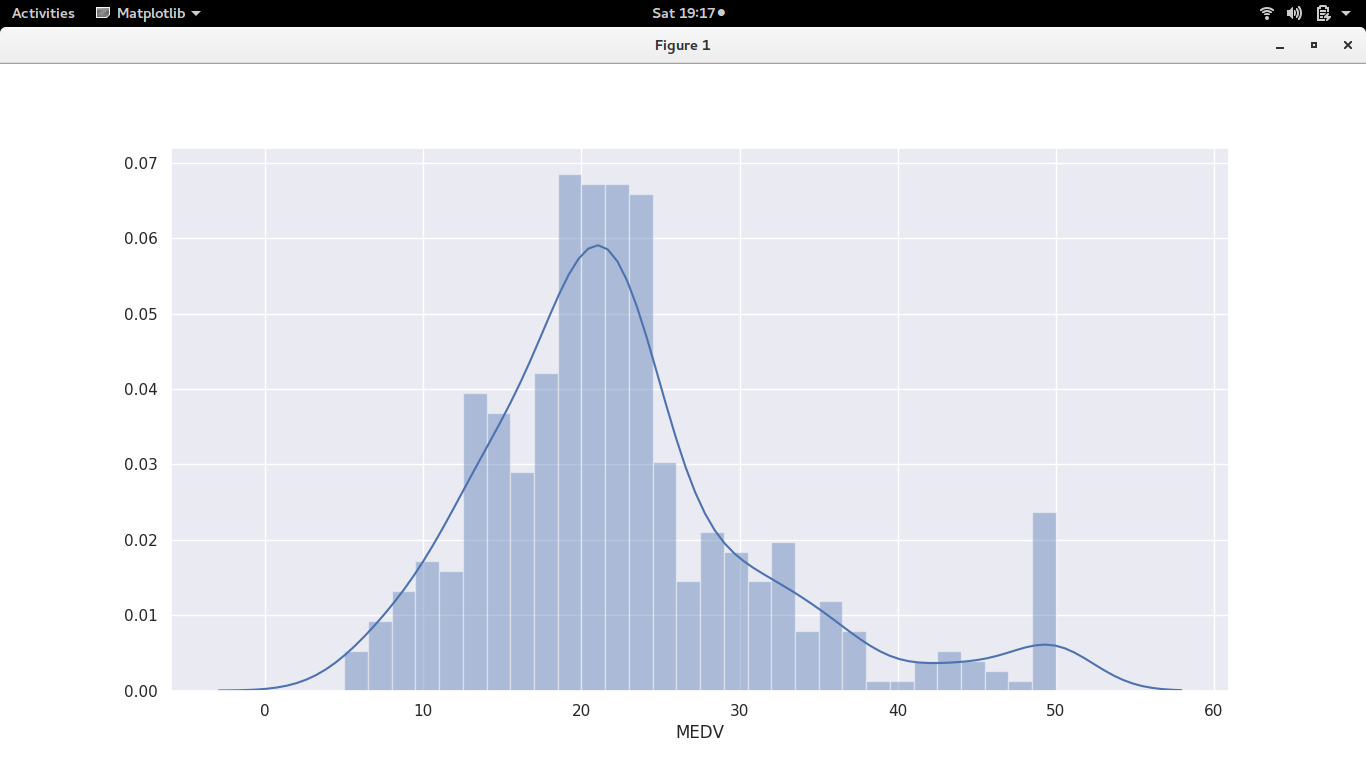
1. A supervised algorithm which can be used to distinguish between different labels. Simple one.
2. Libraries used: Numpy(for arrays), Matplotlib(for plotting), pandas, seaborn (for data manipulations, visualisations), sklearn(dataset and functions)
3. Worked on boston dataset(inputs🡪various features, outputs 🡪 Price)

**Code explanation**

* Lines 1-6 indicate all the libraries to be imported
* Line 7-10 indicate the loading of dataset from cloud and printing its key values
* Line 11-14 indicate the target value to be given to the output array
* Line 16-25 says about the most important features to be taken by using HEATMAPS. The frequency distribution is also plotted
* After choosing the corresponding features, those are plotted in line 26-37
* Line 40-46 is preparation of data in required format, dividing into train and test sets
* Line 47-53 uses sklearn function to train, validate and predict a new example
* RMSE(Root mean square error), and R2 score are observed in line 60-70 and the corresponding equation is printed in line 70
* Plots corresponding are plotted in line 71-105

1. #Import all libraries
2. import numpy as np
3. import matplotlib.pyplot as plt
4. import pandas as pd
5. import seaborn as sns
6. from sklearn.datasets import load\_boston
7. #load dataset and assign variable
8. boston\_dataset = load\_boston()
9. #Print what all are the attreibutes/features in dataset
10. print(boston\_dataset.keys())
11. boston = pd.DataFrame(boston\_dataset.data, columns=boston\_dataset.feature\_names)
12. #boston.head()
13. #target missing in dataset. So assign manually
14. boston['MEDV'] = boston\_dataset.target
15. #print(boston.isnull().sum())
16. #This is used to see corrlations and pick the most important ones. Visualise in HEATMAPS
17. sns.set(rc={'figure.figsize':(11.7,8.27)})
18. sns.distplot(boston['MEDV'], bins=30)
19. plt.show()
20. correlation\_matrix = boston.corr().round(2)
21. sns.heatmap(data=correlation\_matrix, annot=True)
22. #plt.show()
23. plt.figure(figsize=(20, 5))
24. features = ['LSTAT', 'RM']
25. target = boston['MEDV']
26. #plot the most important features
27. for i, col in enumerate(features):
28. plt.subplot(1, len(features) , i+1)
29. x = boston[col]
30. y = target
31. plt.scatter(x, y, marker='o')
32. plt.title(col)
33. plt.xlabel(col)
34. plt.ylabel('MEDV')
35. plt.show()
36. X = pd.DataFrame(np.c\_[boston['LSTAT'], boston['RM']], columns = ['LSTAT', 'RM'])
37. Y = boston['MEDV']
38. #print(X['LSTAT'])
39. #print(Y)
40. #split data as 80,20. Random state is similar to seed number generation in random number generator
41. from sklearn.model\_selection import train\_test\_split
42. X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size = 0.2, random\_state=5)
43. print("Train data shape is, X\_train " + str(X\_train.shape))
44. print("Test data shape, X\_test " +str(X\_test.shape))
45. print("Train output shape, Y\_train " + str(Y\_train.shape))
46. print("Test output output " +str(Y\_test.shape))
47. from sklearn.linear\_model import LinearRegression
48. from sklearn.metrics import mean\_squared\_error, r2\_score
49. #LinearRegression() is the function which has gradient descent, predicting and all the attributes
50. lin\_model = LinearRegression()
51. #Curve fitting
52. lin\_model.fit(X\_train, Y\_train)
53. y\_train\_predict = lin\_model.predict(X\_train)
54. #Root mean square error and R2 score
55. rmse = (np.sqrt(mean\_squared\_error(Y\_train, y\_train\_predict)))
56. r2 = r2\_score(Y\_train, y\_train\_predict)
57. print("The model performance for training set")
58. print("--------------------------------------")
59. print('RMSE is {}'.format(rmse))
60. print('R2 score is {}'.format(r2))
61. # model evaluation for testing set
62. y\_test\_predict = lin\_model.predict(X\_test)
63. rmse = (np.sqrt(mean\_squared\_error(Y\_test, y\_test\_predict)))
64. r2 = r2\_score(Y\_test, y\_test\_predict)
65. print("The model performance for testing set")
66. print("--------------------------------------")
67. print('RMSE is {}'.format(rmse))
68. print('R2 score is {}'.format(r2))
69. print("Coefficients are {}. How many features, those many coefficients. beta0 assumed as inherent constant".format(lin\_model.coef\_))
70. print("The equation will be beta0 + " + str(lin\_model.coef\_[0]) + "\*x + " +str(lin\_model.coef\_[1]) + "\*x^2")
71. # Plot outputs
72. plt.subplot(2,2,1)
73. plt.scatter(X\_train['LSTAT'], Y\_train, color='black')
74. plt.scatter(X\_train['LSTAT'], y\_train\_predict, color='blue')
75. plt.xticks(())
76. plt.yticks(())
77. plt.title('LSAT for train data')
78. plt.xlabel('LSAT feature')
79. plt.ylabel('MEDV feature')
80. plt.subplot(2,2,2)
81. plt.scatter(X\_test['LSTAT'], Y\_test, color='black')
82. plt.scatter(X\_test['LSTAT'], y\_test\_predict, color='blue')
83. plt.xticks(())
84. plt.yticks(())
85. plt.title('LSAT for test data')
86. plt.xlabel('LSAT feature')
87. plt.ylabel('MEDV feature')
88. #plt.show()
89. plt.subplot(2,2,3)
90. plt.scatter(X\_train['RM'], Y\_train, color='black')
91. plt.scatter(X\_train['RM'], y\_train\_predict, color='blue')
92. plt.xticks(())
93. plt.yticks(())
94. plt.title('RM for train data')
95. plt.xlabel('RM feature')
96. plt.ylabel('MEDV feature')
97. plt.subplot(2,2,4)
98. plt.scatter(X\_test['RM'], Y\_test, color='black')
99. plt.scatter(X\_test['RM'], y\_test\_predict, color='blue')
100. plt.xticks(())
101. plt.yticks(())
102. plt.title('RM for test data')
103. plt.xlabel('RM feature')
104. plt.ylabel('MEDV feature')
105. plt.show()

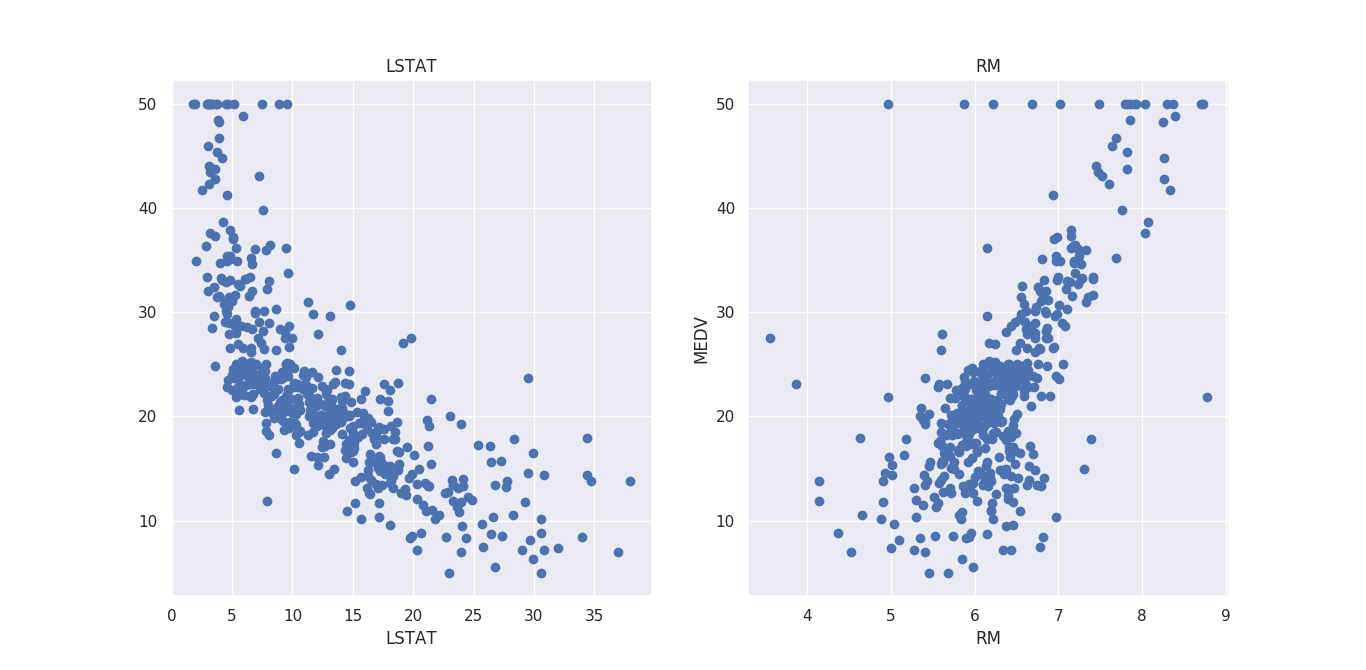
**Outputs**



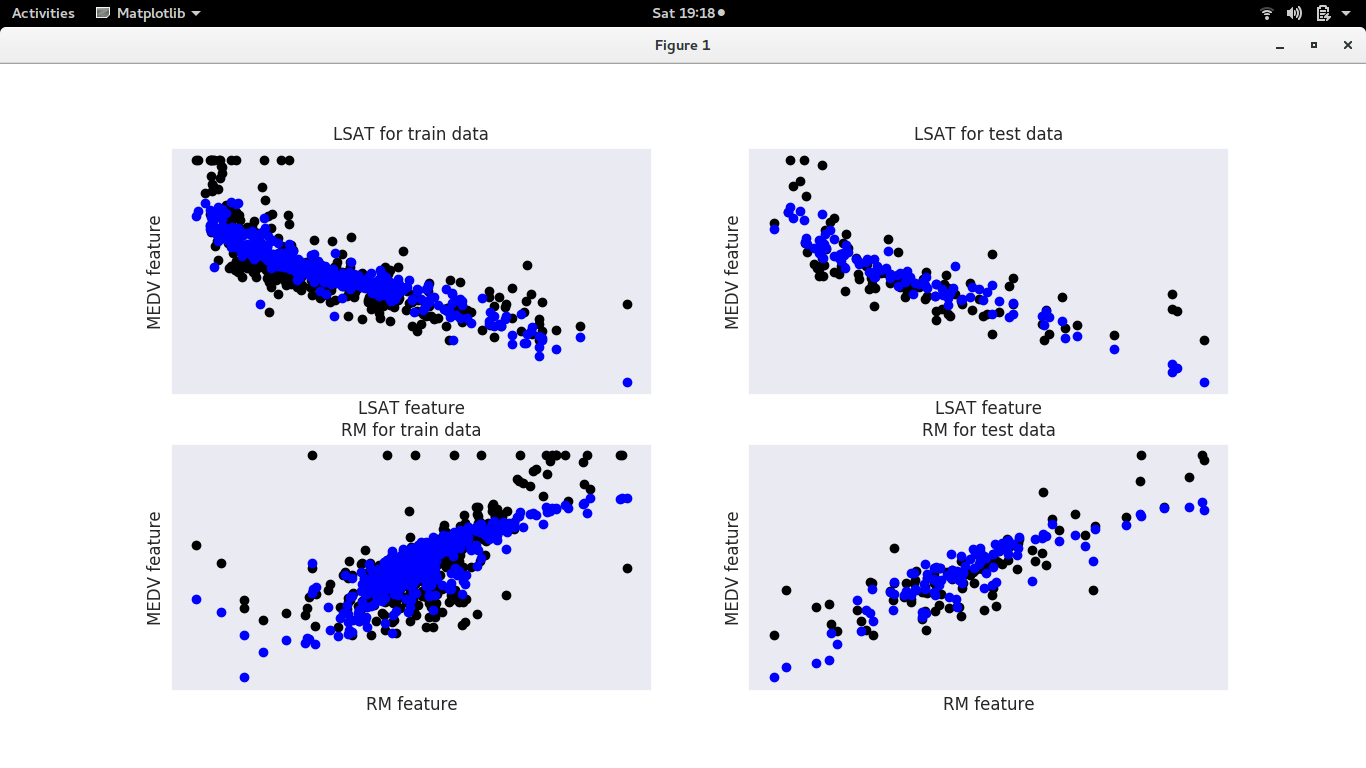
Data distribution plot



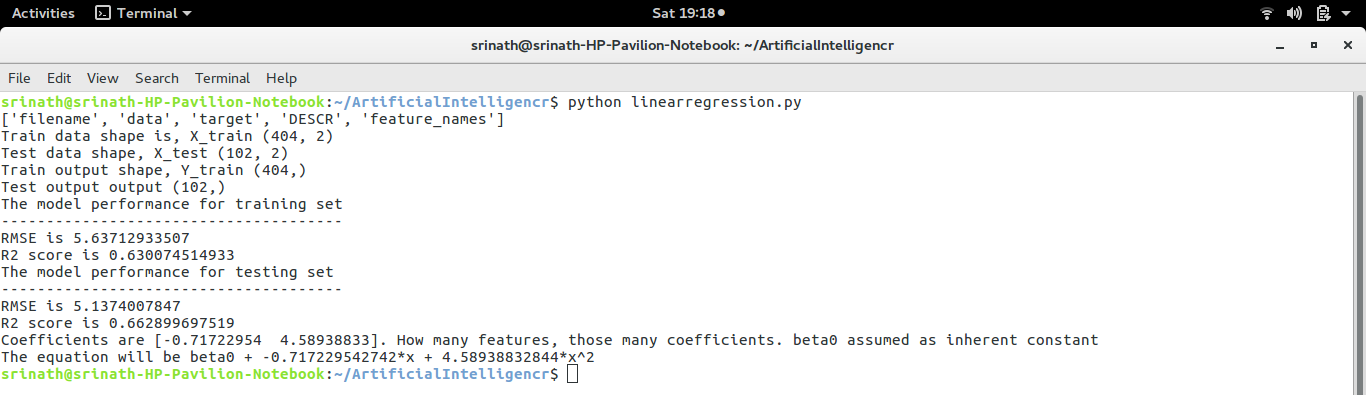
Heat map between different features. More positive indicates more correlated and more negative indicates more inversely correlated



Important features taken(LSTAT and RM)



Their corresponding data is visualized



Output in terminal

**Logistic regression**

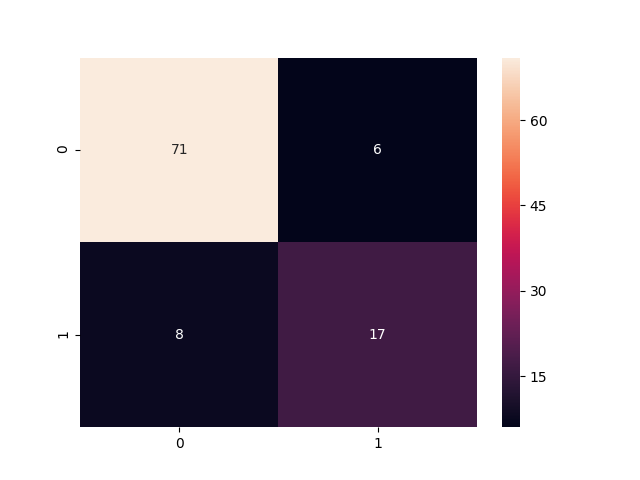
1. Another supervised learning algorithm. Used to distinguish different labels based on a non-linear function approximation
2. Libraries used: Numpy(for arrays), Matplotlib(for plotting), pandas, seaborn (for data manipulations, visualisations), sklearn(dataset and functions)
3. Worked on boston dataset(inputs🡪various features, outputs 🡪 Price)

**Code explanation**

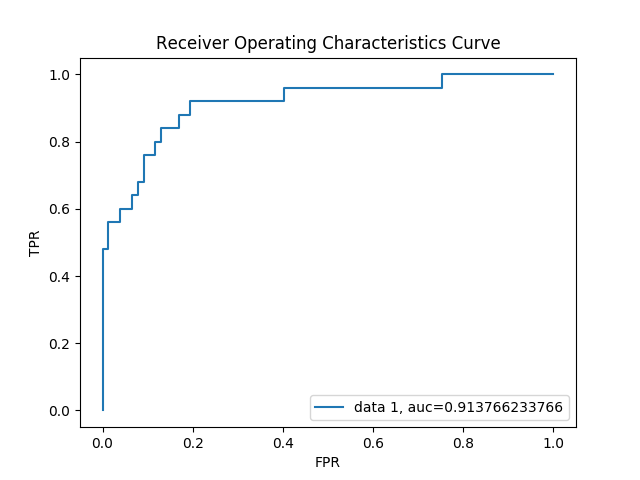
* Line 1-6 loading libraries
* Line 7-10 indicate the loading of dataset from cloud and printing its key values
* Line 11-14 indicate the target value to be given to the output array
* Line 14-16 takes the important features.
* Line 17-20 is because the logistic value is bounded. So, need to normalize and give binary values
* Line 21-31 to process data in required format
* Next fitting and predicting and plotting of confusion matrix in line 32-41
* Accuracy, precision and recall values printed in line 42-44
* Receiver operating characteristics plotted in line 45-53

1. #Load all the libraries
2. import numpy as np
3. import matplotlib.pyplot as plt
4. import pandas as pd
5. import seaborn as sns
6. from sklearn.datasets import load\_boston
7. #load dataset and assign a variable
8. boston\_dataset = load\_boston()
9. #Using Pandas dataframes to access data/columns
10. boston = pd.DataFrame(boston\_dataset.data, columns=boston\_dataset.feature\_names)
11. #boston.head(). Used to print first 5 rows
12. #Give the target variable. Not given in dataset
13. boston['MEDV'] = boston\_dataset.target
14. #Taken onlythe most important features. Determined from HEATMAPS in linearregression.py
15. X = pd.DataFrame(np.c\_[boston['LSTAT'], boston['RM']], columns = ['LSTAT', 'RM'])
16. Y = boston['MEDV']
17. Y = Y/25
18. for i in range (506):
19. if(Y[i] == 2):
20. Y[i] = 1
21. #split data as 80,20. Random state is similar to seed number generation in random number generator
22. from sklearn.model\_selection import train\_test\_split
23. X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size = 0.2, random\_state=5)
24. Y\_trainnew = Y\_train.astype(int)
25. Y\_testnew = Y\_test.astype(int)
26. # X\_trainnew = X\_train.astype(int)
27. # X\_testnew = X\_test.astype(int)
28. from sklearn.linear\_model import LogisticRegression
29. # instantiate the model (using the default parameters)
30. logreg = LogisticRegression()
31. #print("...." + str(type(X\_train)) + str(type(Y\_train)))
32. # fit the model with data
33. logreg.fit(X\_train,Y\_trainnew)
34. #
35. y\_pred=logreg.predict(X\_test)
36. #print(y\_pred, Y\_testnew)
37. from sklearn import metrics
38. cnf\_matrix = metrics.confusion\_matrix(Y\_testnew, y\_pred)
39. #print(cnf\_matrix.shape)
40. sns.heatmap(data=cnf\_matrix, annot=True)
41. plt.show()
42. print("Accuracy:",metrics.accuracy\_score(Y\_testnew, y\_pred))
43. print("Precision:",metrics.precision\_score(Y\_testnew, y\_pred))
44. print("Recall or True Positive Rate:",metrics.recall\_score(Y\_testnew, y\_pred))
45. y\_pred\_proba = logreg.predict\_proba(X\_test)[::,1]
46. fpr, tpr, \_ = metrics.roc\_curve(Y\_testnew, y\_pred\_proba)
47. auc = metrics.roc\_auc\_score(Y\_testnew, y\_pred\_proba)
48. plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
49. plt.legend(loc=4)
50. plt.xlabel('FPR')
51. plt.ylabel('TPR')
52. plt.title('Receiver Operating Characteristics Curve')
53. plt.show()

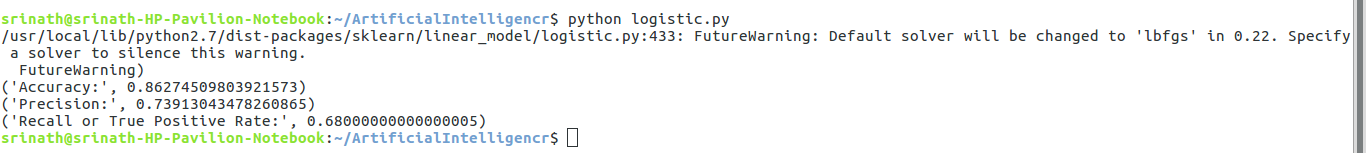
**Outputs**



Confusion matrix



ROC plot(between TPR and FPR)



Output in terminal

**Gradient descent**

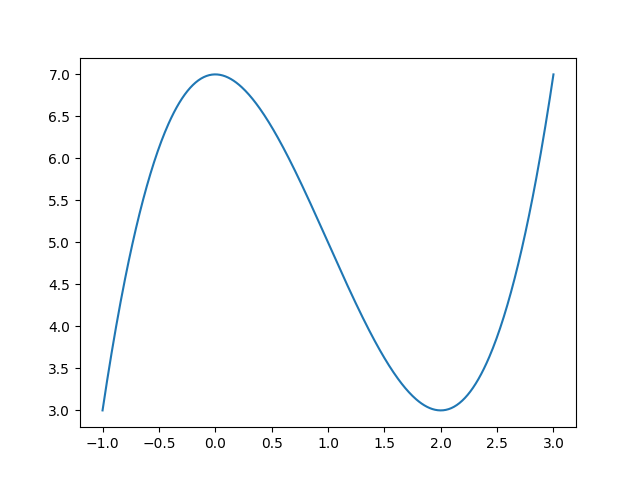
1. A technique used to converge/ get minimum error. Depends on the learning rate value choosen. Applied on cost function for error minimization
2. Numpy and matplotlib used

**Code Explanation**

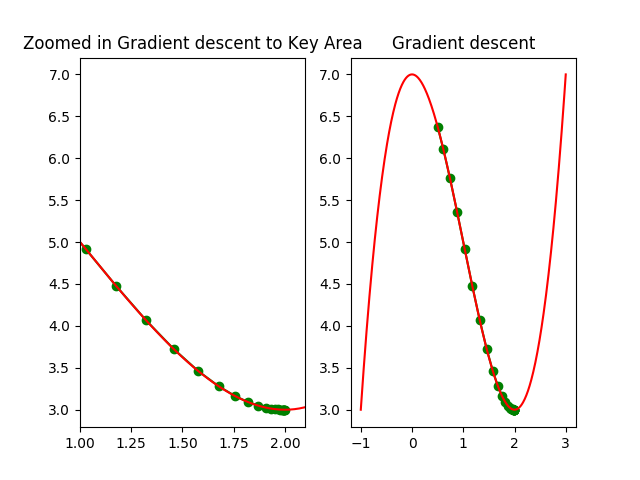
* Line 1-2 for importing libraries
* Line 3 is our function on which gradient descent is to be applied
* Line 4-8 used for plotting. linspace() is a function used for getting a vector
* Line 9-11 is a function for derivative
* Line 12-21 is a function for checking whether the step taken is less than a predefined value. If so, it stops iterating further. Else it continues to be in the loop. Called in line 34
* Line 22-33 for plotting and visualizing how this steps process occur

1. import numpy as np
2. import matplotlib.pyplot as plt
3. function = lambda x: (x \*\* 3)-(3 \*(x \*\* 2))+7
4. #Get 1000 evenly spaced numbers between -1 and 3 (arbitratil chosen to ensure steep curve)
5. x = np.linspace(-1,3,500)
6. #Plot the curve
7. plt.plot(x, function(x))
8. plt.show()
9. def deriv(x):
10. x\_deriv = 3\* (x\*\*2) - (6 \* (x))
11. return x\_deriv
12. def step(x\_new, x\_prev, precision, l\_r):
13. x\_list, y\_list = [x\_new], [function(x\_new)]
14. while abs(x\_new - x\_prev) > precision:
15. x\_prev = x\_new
16. d\_x = - deriv(x\_prev)
17. x\_new = x\_prev + (l\_r \* d\_x)
18. x\_list.append(x\_new)
19. y\_list.append(function(x\_new))
20. print ("Local minimum occurs at: "+ str(x\_new))
21. print ("Number of steps: " + str(len(x\_list)))
22. plt.subplot(1,2,2)
23. plt.scatter(x\_list,y\_list,c="g")
24. plt.plot(x\_list,y\_list,c="g")
25. plt.plot(x,function(x), c="r")
26. plt.title("Gradient descent")
27. plt.subplot(1,2,1)
28. plt.scatter(x\_list,y\_list,c="g")
29. plt.plot(x\_list,y\_list,c="g")
30. plt.plot(x,function(x), c="r")
31. plt.xlim([1.0,2.1])
32. plt.title("Zoomed in Gradient descent to Key Area")
33. plt.show()
34. step(0.5, 0, 0.001, 0.05)

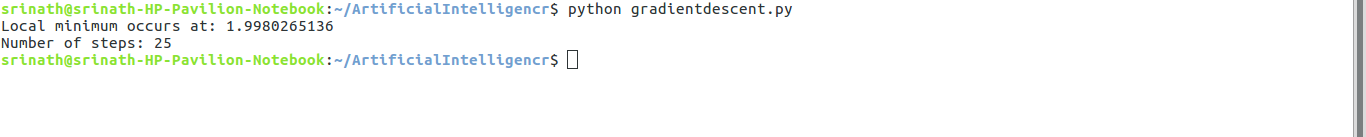
**Outputs**



Input graph



How gradient descent searches for minimum by going through steps



Output of terminal

**Perceptron**

1. A simple perceptron can be used only on “Linearly separable data” which means if we can draw a straight line dividing data, then only this perceptron can classify
2. Different gates are visualized
3. Numpy is used

**Code explanation**

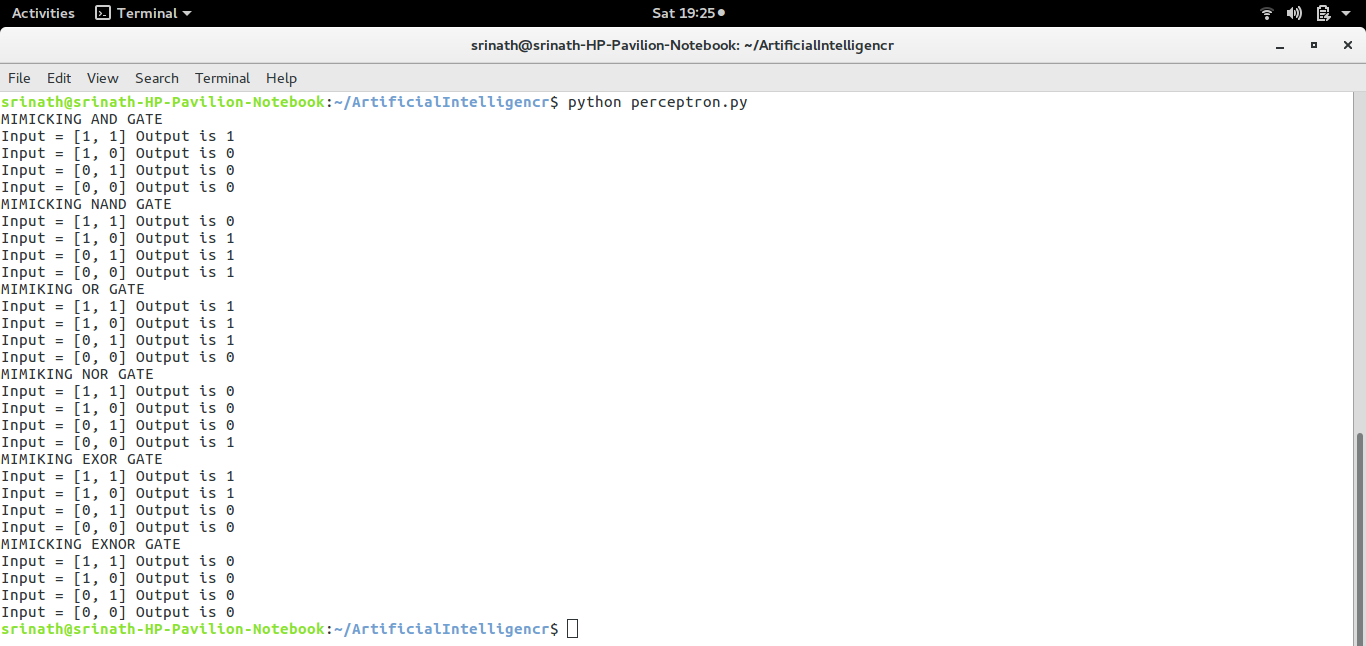
* Line 3 defines a function “Perceptron” where the threshold value, learning rate is defined and all the weights are initialized to 0
* Line 8 defines a function “Predict” which takes the inner product between the weights and inputs and checks if the value is greater than a defined threshold(0 here) and classifies accordingly
* Line 15 has a function “Train” for which the labelled value and predicted value is given as inputs. So, as long as it gets error, the weights get updated by formulas in line 18.a to 18.c

Bias weight is separate, so its updating is different from other weights

* Line 19-23 gives the standard inputs for 2 input gates
* AND, NAND, OR, NOR, EXOR, EXNOR are implemented and valuated in the next lines

1. import numpy as np
2. #from perceptron import Perceptron
3. class Perceptron(object):
4. def \_\_init\_\_(self, no\_of\_inputs, threshold=100, learning\_rate=0.01):
5. self.threshold = threshold
6. self.learning\_rate = learning\_rate
7. self.weights = np.zeros(no\_of\_inputs + 1)
8. def predict(self, inputs):
9. summation = np.dot(inputs, self.weights[1:]) + self.weights[0]
10. if summation > 0:
11. activation = 1
12. else:
13. activation = 0
14. return activation
15. def train(self, training\_inputs, labels):
16. for \_ in range(self.threshold):
    1. #Simply backpropagation algorithm
17. for inputs, label in zip(training\_inputs, labels):
18. #ZIP is used to group variables so that can be used together in for loop
    1. prediction = self.predict(inputs)
    2. self.weights[1:] += self.learning\_rate \* (label - prediction) \* inputs
    3. self.weights[0] += self.learning\_rate \* (label - prediction)
19. training\_inputs = []
20. training\_inputs.append(np.array([1, 1]))
21. training\_inputs.append(np.array([1, 0]))
22. training\_inputs.append(np.array([0, 1]))
23. training\_inputs.append(np.array([0, 0]))
24. labels = np.array([1, 0, 0, 0])
25. perceptron = Perceptron(2)
26. perceptron.train(training\_inputs, labels)
27. print("MIMICKING AND GATE")
28. print("Input = " + str([1,1]) + " Output is " + str(perceptron.predict([1,1])))
29. print("Input = " + str([1,0]) + " Output is " + str(perceptron.predict([1,0])))
30. print("Input = " + str([0,1]) + " Output is " + str(perceptron.predict([0,1])))
31. print("Input = " + str([0,0]) + " Output is " + str(perceptron.predict([0,0])))
32. labels = np.array([0, 1, 1, 1])
33. perceptron = Perceptron(2)
34. perceptron.train(training\_inputs, labels)
35. print("MIMICKING NAND GATE")
36. print("Input = " + str([1,1]) + " Output is " + str(perceptron.predict([1,1])))
37. print("Input = " + str([1,0]) + " Output is " + str(perceptron.predict([1,0])))
38. print("Input = " + str([0,1]) + " Output is " + str(perceptron.predict([0,1])))
39. print("Input = " + str([0,0]) + " Output is " + str(perceptron.predict([0,0])))
40. labels = np.array([1,1,1,0])
41. perceptron = Perceptron(2)
42. perceptron.train(training\_inputs, labels)
43. print("MIMIKING OR GATE")
44. print("Input = " + str([1,1]) + " Output is " + str(perceptron.predict([1,1])))
45. print("Input = " + str([1,0]) + " Output is " + str(perceptron.predict([1,0])))
46. print("Input = " + str([0,1]) + " Output is " + str(perceptron.predict([0,1])))
47. print("Input = " + str([0,0]) + " Output is " + str(perceptron.predict([0,0])))
48. labels = np.array([0,0,0,1])
49. perceptron = Perceptron(2)
50. perceptron.train(training\_inputs, labels)
51. print("MIMIKING NOR GATE")
52. print("Input = " + str([1,1]) + " Output is " + str(perceptron.predict([1,1])))
53. print("Input = " + str([1,0]) + " Output is " + str(perceptron.predict([1,0])))
54. print("Input = " + str([0,1]) + " Output is " + str(perceptron.predict([0,1])))
55. print("Input = " + str([0,0]) + " Output is " + str(perceptron.predict([0,0])))
56. labels = np.array([0,1,1,0])
57. perceptron = Perceptron(2)
58. perceptron.train(training\_inputs, labels)
59. print("MIMIKING EXOR GATE")
60. print("Input = " + str([1,1]) + " Output is " + str(perceptron.predict([1,1])))
61. print("Input = " + str([1,0]) + " Output is " + str(perceptron.predict([1,0])))
62. print("Input = " + str([0,1]) + " Output is " + str(perceptron.predict([0,1])))
63. print("Input = " + str([0,0]) + " Output is " + str(perceptron.predict([0,0])))
64. labels = np.array([1,0,0,1])
65. perceptron = Perceptron(2)
66. perceptron.train(training\_inputs, labels)
67. print("MIMICKING EXNOR GATE")
68. print("Input = " + str([1,1]) + " Output is " + str(perceptron.predict([1,1])))
69. print("Input = " + str([1,0]) + " Output is " + str(perceptron.predict([1,0])))
70. print("Input = " + str([0,1]) + " Output is " + str(perceptron.predict([0,1])))
71. print("Input = " + str([0,0]) + " Output is " + str(perceptron.predict([0,0])))

**Output**



Output in terminal. EXOR and EXNOR are not properly predicted as these are NON-SEPERABLE/ NON-LINEAR data

**KNN(K nearest neighbours)**

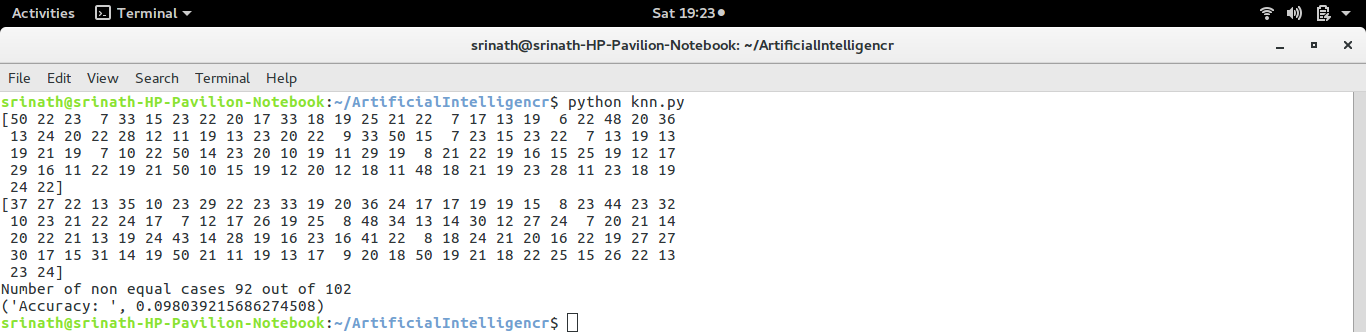
1. A clustering algorithm where the output depends on the value of the K choosen
2. Initially worked on boston dataset. But KNN is not suitable on that dataset
3. KNN is suitable to the datasets which has the datapoints as clusters. While boston doesn’t have any clusters between its datapoints.
4. So, IRIS dataset is used for KNN
5. Numpy, matplot, seaborn, pandas, sklearn libraries used

**Code explanation(For boston dataset)**

* Libraries imported in lines3-8
* Line 9-18. Important features are recognized(As in linear regression)
* Line 20-26. Data processing
* Line 27. Model fitting and predicting and then printing the accuracy values along with the

1. #KNN for this dataset wont give good results as a general cluster cannot be visualised. So, demonstrate this to differentiate between Linear
2. #regression and KNN and the importance of Algorithm vs dataset
3. #Load all the libraries
4. import numpy as np
5. import matplotlib.pyplot as plt
6. import pandas as pd
7. import seaborn as sns
8. from sklearn.datasets import load\_boston
9. #load dataset and assign a variable
10. boston\_dataset = load\_boston()
11. #Using Pandas dataframes to access data/columns
12. boston = pd.DataFrame(boston\_dataset.data, columns=boston\_dataset.feature\_names)
13. #boston.head(). Used to print first 5 rows
14. #Give the target variable. Not given in dataset
15. boston['MEDV'] = boston\_dataset.target
16. #Taken onlythe most important features. Determined from HEATMAPS in linearregression.py
17. X = pd.DataFrame(np.c\_[boston['LSTAT'], boston['RM']], columns = ['LSTAT', 'RM'])
18. Y = boston['MEDV']
19. #split data as 80,20. Random state is similar to seed number generation in random number generator
20. from sklearn.model\_selection import train\_test\_split
21. X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size = 0.2, random\_state=5)
22. from sklearn.neighbors import KNeighborsClassifier
23. model = KNeighborsClassifier(n\_neighbors=5)#5 is just arbitrary
24. Y\_trainnew = Y\_train.astype(int)
25. Y\_testnew = Y\_test.astype(int)
26. #print(Y\_trainnew[0:])
27. # Train the model using the training sets
28. model.fit(X\_train,Y\_trainnew)
29. #Predict Output
30. Y\_predicted= model.predict(X\_test)
31. print(Y\_predicted)
32. Y\_testnew = np.array(Y\_testnew)
33. print(Y\_testnew)
34. print("Number of non equal cases " + str(np.sum(Y\_predicted != Y\_testnew)) + " out of 102")
35. from sklearn import metrics
36. # Model Accuracy
37. print("Accuracy: ",metrics.accuracy\_score(Y\_testnew, Y\_predicted))

**Outputs**



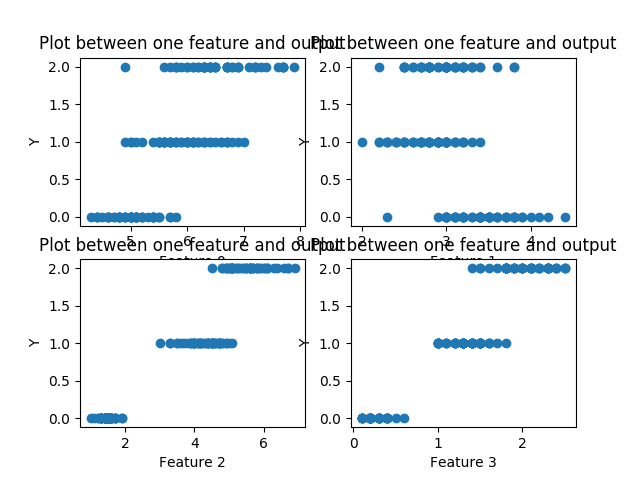
Output of terminal. Out of 102, 92 are wrongly predicted which makes KNN not a good fit for this dataset.

**Code explanation(for IRIS)**

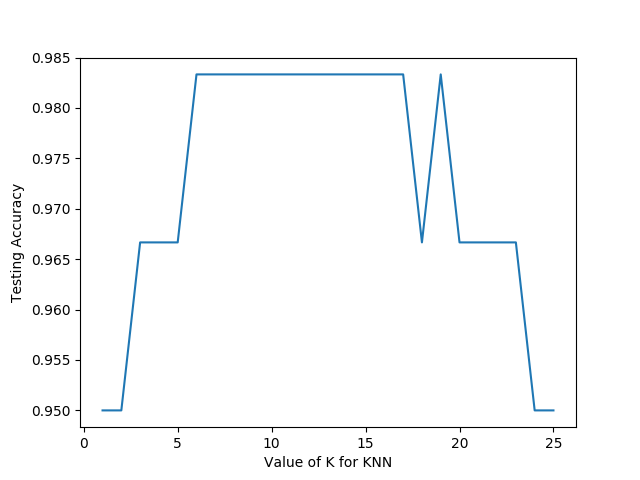
* Libraries loaded in line 1-4
* Features are observed, visualized in line 5-14. Checked how many are there in each class
* Line 15-23. For fitting and training
* Line 24-29. For checking all the shapes(error debugging)
* Line 30-50. Plotting different features and the outputs
* Line 51-67. To choose the best “K” value, a for loop is written iterating for K = 1 to 26 and for each K fitting the model and predicting, getting its accuracy metrics and all those are plotted. Best K is choosen accordingly
* Line 72-77 for checking a new test case by the uset

1. import numpy as np
2. from sklearn.datasets import load\_iris
3. import matplotlib.pyplot as plt
4. iris = load\_iris()
5. # create X (features) and y (response)
6. # X features are width and height of sepals and petals
7. #Y feature says which flower it is
8. X = iris.data
9. y = iris.target
10. #Use these to debug if errors come
11. #print(type(X))
12. #print(X[:,0])
13. #print(y.shape)
14. #print("Number of 0's " + str(np.sum(y == 0)) + " and no's of 1's are " + str(np.sum(y ==1)) + " no of 2's are " + str(np.sum(y == 2)))
15. from sklearn.neighbors import KNeighborsClassifier
16. from sklearn import metrics
17. #Below lines use complete set for training thus giving trivial solution/overfitting. So we need to divide into train/test data
18. #Or train with entire dataset and give user data. Right now it is commented
19. # knn = KNeighborsClassifier(n\_neighbors=5)
20. # knn.fit(X, y)
21. # y\_pred = knn.predict(X)
22. # print(metrics.accuracy\_score(y, y\_pred))
23. from sklearn.model\_selection import train\_test\_split
24. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4, random\_state=4)
25. #Use this if any errors regarding shapes
26. # print("Train X shape " + str(X\_train.shape))
27. # print("Train Y shape " + str(y\_train.shape))
28. # print("Test X shape " + str(X\_test.shape))
29. # print("Test Y shape " + str(y\_test.shape))
30. plt.subplot(2,2,1)
31. plt.scatter(X[:,0], y, marker = 'o')
32. plt.xlabel("Feature 0")
33. plt.ylabel("Y")
34. plt.title("Plot between one feature and output")
35. plt.subplot(2,2,2)
36. plt.scatter(X[:,1], y, marker = 'o')
37. plt.xlabel("Feature 1")
38. plt.ylabel("Y")
39. plt.title("Plot between one feature and output")
40. plt.subplot(2,2,3)
41. plt.scatter(X[:,2], y, marker = 'o')
42. plt.xlabel("Feature 2")
43. plt.ylabel("Y")
44. plt.title("Plot between one feature and output")
45. plt.subplot(2,2,4)
46. plt.scatter(X[:,3], y, marker = 'o')
47. plt.xlabel("Feature 3")
48. plt.ylabel("Y")
49. plt.title("Plot between one feature and output")
50. plt.show()
51. k\_range = range(1, 26)
52. # Create Python dictionary using [] or dict()
53. scores = []
54. #Use for loop to determine max accuracy for which K
55. for k in k\_range:
56. knn = KNeighborsClassifier(n\_neighbors=k)
57. knn.fit(X\_train, y\_train)
58. y\_pred = knn.predict(X\_test)
59. scores.append(metrics.accuracy\_score(y\_test, y\_pred))
60. print("Scores are ")
61. print(scores)
62. print("And the max index is there for ")
63. K\_max = scores.index(max(scores))
64. print(K\_max)
65. print("So, use that as K and now give some test inputs to check your clustering model")
66. knn = KNeighborsClassifier(n\_neighbors= K\_max)
67. knn.fit(X\_train, y\_train)
68. plt.plot(k\_range, scores)
69. plt.xlabel('Value of K for KNN')
70. plt.ylabel('Testing Accuracy')
71. plt.show()
72. print("An example input and output from train set to cross verify " + str(X\_train[0]) + " ---> " +str(y\_train[0]))
73. print("An example input and output from test set to cross verify " + str(X\_test[0]) + " ---> " +str(y\_test[0]))
74. input = np.array([[4.3,3.2,1.2,0.15],[6.2,2.7,5.45,1.9]])
75. input = input[:,np.newaxis]
76. print("User inputs are " + str(input[0]) + " " + str(input[1]))
77. print("For input 0 prediction is "+ str(knn.predict(input[0])) + " and for input 1 it is "+ str(knn.predict(input[1])))

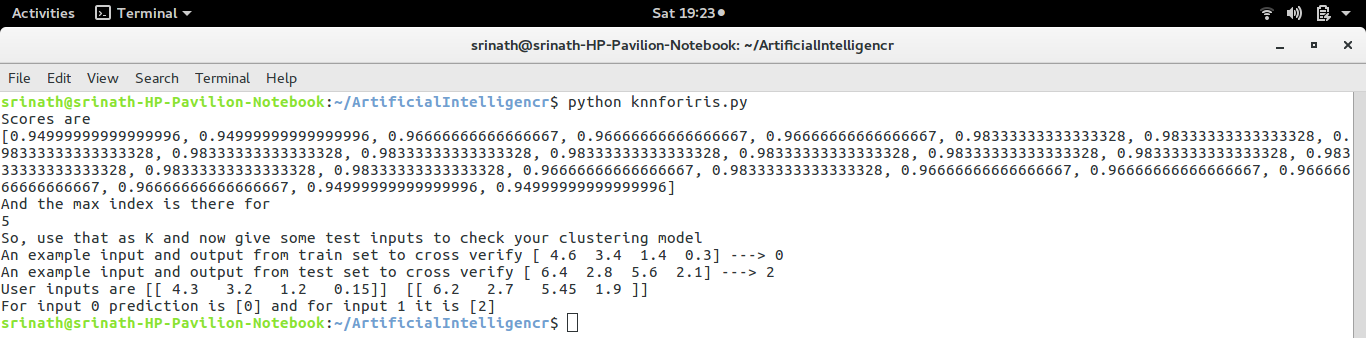
**Outputs**



Plot between different features and output. Here we can observe that the data has clusters. So, KNN can be used



Varying K and checking the accuracy. Choose the best K from this graph.



Output in terminal

**Naïve bayes**

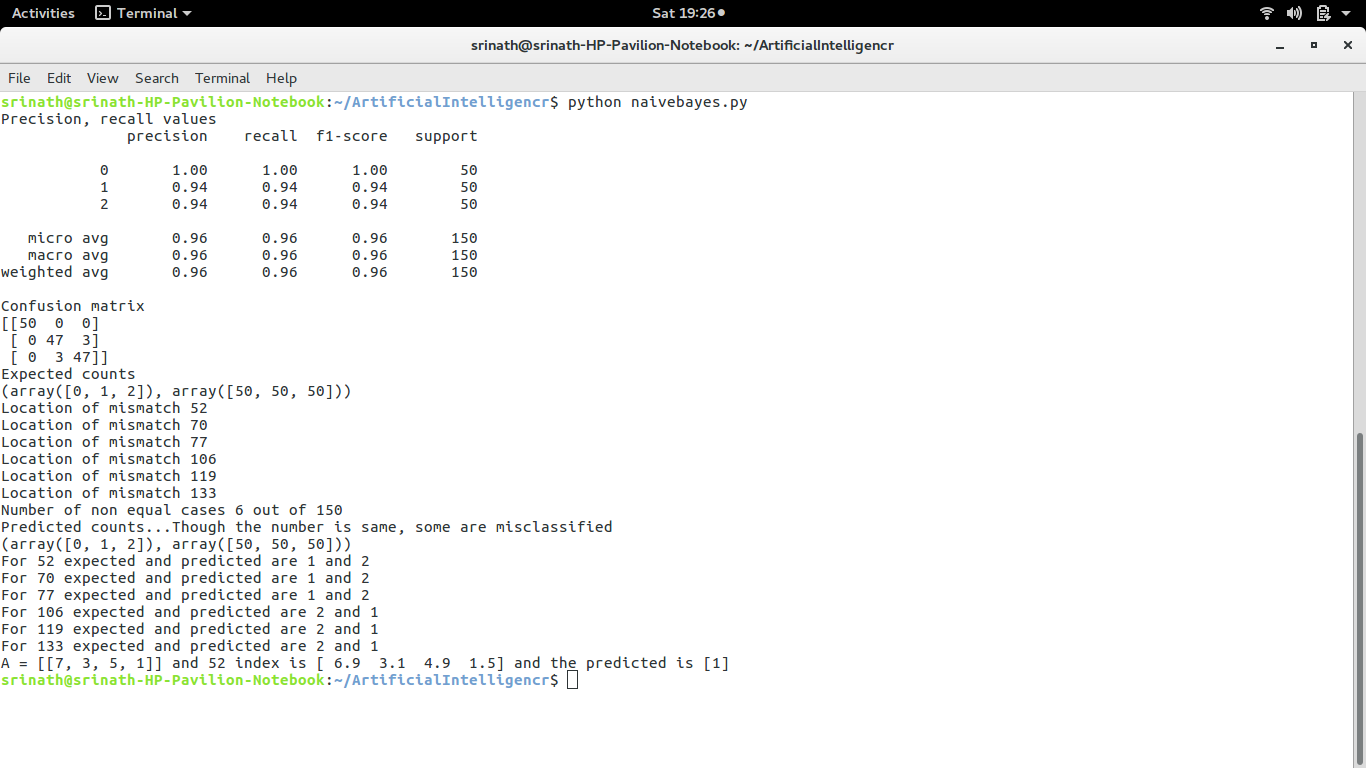
1. A probabilistic classification algorithm
2. Called naïve because of the assumptions that it takes i.e all the features are independent which is not ideal in real world
3. Numpy and sklearn used. Worked on IRIS dataset

**Code explanation**

* Line 1-4 loading libraries
* Line 6 loading dataset
* Line 7-11, fitting gaussian model and predicting the classes
* Line 12-18, printing various metrics and checking the classes.(Because some will be misclassified)
* That is if 50 examples are there for class 1 and the predicted number is also 50, it doesn’t guarantee that these 50 are the actual 50. Some may misclassify but the final count may remain the same. The rest of the lines debug this to know which are misclassified.
* Small changes in test input wrto the X present there is given to observe the change in predictions in line 32,33

1. import numpy as np
2. from sklearn import datasets
3. from sklearn import metrics
4. from sklearn.naive\_bayes import GaussianNB
5. # &amp;amp;nbsp;
6. dataset = datasets.load\_iris()
7. model = GaussianNB()
8. model.fit(dataset.data, dataset.target)
9. expected = dataset.target
10. #print(dataset.data)
11. predicted = model.predict(dataset.data)
12. print("Precision, recall values")
13. print(metrics.classification\_report(expected, predicted))
14. print("Confusion matrix")
15. print(metrics.confusion\_matrix(expected, predicted))
16. unique, counts = np.unique(expected, return\_counts = True)
17. print("Expected counts")
18. print(unique, counts)
19. matrix = (expected == predicted)
20. for i in range (150):
    1. if(matrix[i] == False):
       1. print("Location of mismatch " + str(i))
21. print("Number of non equal cases " + str(np.sum(predicted != expected)) + " out of 150")
22. unique\_predicted, counts\_predicted = np.unique(predicted, return\_counts = True)
23. print("Predicted counts...Though the number is same, some are misclassified")
24. print(unique\_predicted, counts\_predicted)
25. #print(dataset.data[52],dataset.data[70],dataset.data[77], dataset.data[106], dataset.data[119], dataset.data[133])
26. print("For 52 expected and predicted are " + str(expected[52]) + " and " + str(predicted[52]))
27. print("For 70 expected and predicted are " + str(expected[70]) + " and " + str(predicted[70]))
28. print("For 77 expected and predicted are " + str(expected[77]) + " and " + str(predicted[77]))
29. print("For 106 expected and predicted are " + str(expected[106]) + " and " + str(predicted[106]))
30. print("For 119 expected and predicted are " + str(expected[119]) + " and " + str(predicted[119]))
31. print("For 133 expected and predicted are " + str(expected[133]) + " and " + str(predicted[133]))
32. A = [[7,3,5,1]]
33. print("A = " + str(A) + " and 52 index is " + str(dataset.data[52]) + " and the predicted is " + str(model.predict(A)))

**Output**



Output of terminal

**Tree classifier**

1. A simple classification algorithm. Easily prone to changes in data.
2. Numpy, matplotlib, seaborn, pandas and sklearn used and worked on Boston dataset

**Code explanation**

* Line 1-6 for importing libraries
* Line 7-18 for visualizing data and picking the most important ones(check linear regression explanation)
* Line 18-20 for normalization of data(as tree classifier needs categorical data)
* Line 21-26 for data processing
* Line 27-32 for model fitting and converting to required format
* Line 33-36 for predicting and metrics
* Line 37-47 to save the output in the form of an image

1. #Load all the libraries
2. import numpy as np
3. import matplotlib.pyplot as plt
4. import pandas as pd
5. import seaborn as sns
6. from sklearn.datasets import load\_boston
7. #load dataset and assign a variable
8. boston\_dataset = load\_boston()
9. #Using Pandas dataframes to access data/columns
10. boston = pd.DataFrame(boston\_dataset.data, columns=boston\_dataset.feature\_names)
11. #boston.head(). Used to print first 5 rows
12. #Give the target variable. Not given in dataset
13. boston['MEDV'] = boston\_dataset.target
14. #Taken onlythe most important features. Determined from HEATMAPS in linearregression.py
15. columns = ['LSTAT', 'RM']
16. X = pd.DataFrame(np.c\_[boston['LSTAT'], boston['RM']], columns = ['LSTAT', 'RM'])
17. Y = boston['MEDV']
18. Y = Y/25
19. for i in range (506):
20. if(Y[i] == 2):
21. Y[i] = 1
22. #split data as 80,20. Random state is similar to seed number generation in random number generator
23. from sklearn.model\_selection import train\_test\_split
24. from sklearn.tree import DecisionTreeClassifier
25. from sklearn import metrics
26. X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size = 0.2, random\_state=5)
27. Y\_trainnew = Y\_train.astype(int)
28. Y\_testnew = Y\_test.astype(int)
29. # Create Decision Tree classifer object
30. clf = DecisionTreeClassifier(criterion="entropy", max\_depth=2)
31. # Train Decision Tree Classifer
32. clf = clf.fit(X\_train,Y\_trainnew)
33. #Predict the response for test dataset
34. y\_pred = clf.predict(X\_test)
35. print("Accuracy:",metrics.accuracy\_score(Y\_testnew, y\_pred))
36. print(np.sum(Y\_trainnew == 1))
37. from sklearn.tree import export\_graphviz
38. from sklearn.externals.six import StringIO
39. from IPython.display import Image
40. import pydotplus
41. dot\_data = StringIO()
42. export\_graphviz(clf, out\_file=dot\_data,
43. filled=True, rounded=True,
44. special\_characters=True,feature\_names = columns,class\_names=['0','1'])
45. graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue())
46. graph.write\_png('boston.png')
47. Image(graph.create\_png())

**Output**

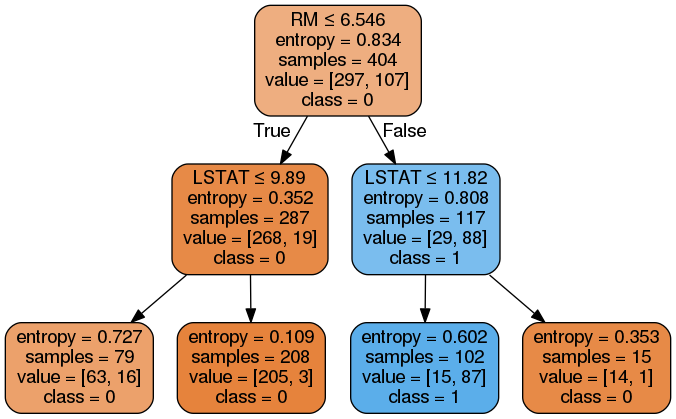
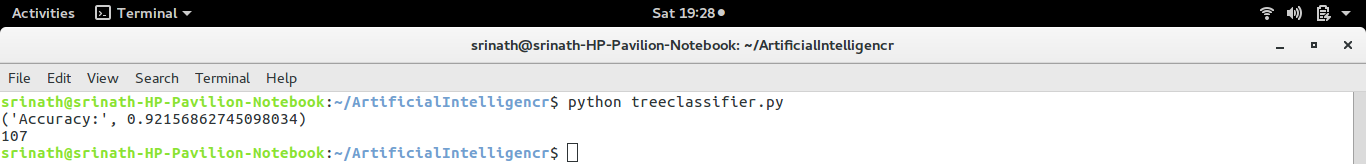


Image of the tree



Output of terminal

**SVM(Support vector machines)**

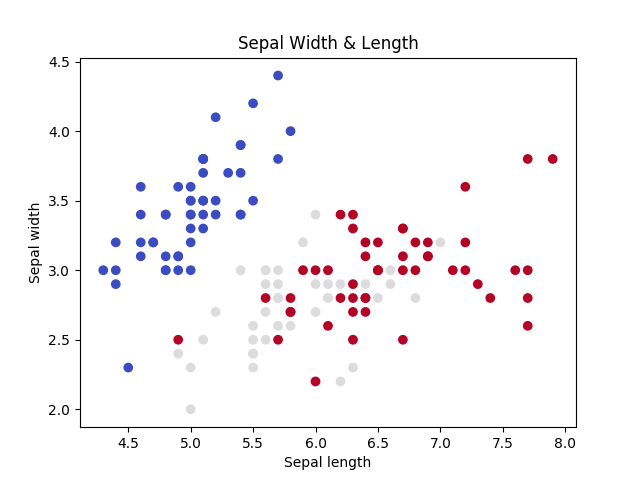
1. A clustering algorithm which can be used to divide data.
2. Based on the kernels/ functions used. Linear, RBF(radial basis function) are the famous kernels used
3. Worked on iris dataset and sklearn, numpy and matplotlib libraries used

**Code explanation**

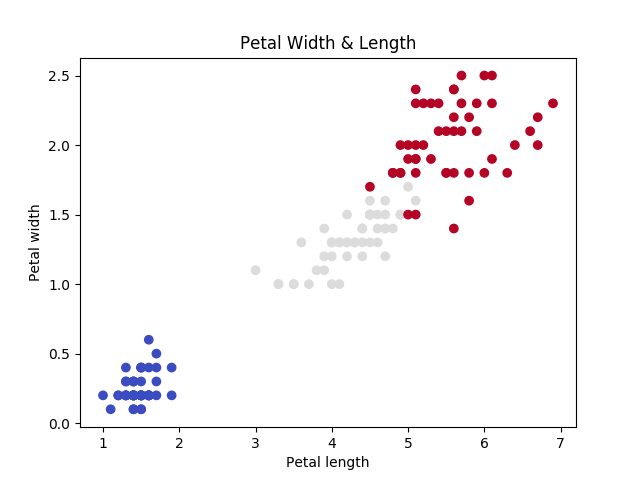
* Line 1-4 for importing libraries
* Line 5-6 for dataset visualization
* Line 9 and 11 are the functions which visualize the sepal data with the output and petal data with the output respectively
* Line 13-21 creates various kernels. Linear, polynomial and RBF are the ones
* Line 26-75 used to plot the data along with the kernels and to visualize. Two functions written for sepals and petals and meshes were created to observe all the graphs

1. from sklearn import datasets
2. from sklearn import svm
3. import numpy as np
4. import matplotlib.pyplot as plt
5. iris\_dataset = datasets.load\_iris()
6. # print("Iris data set Description :: ", iris\_dataset['DESCR'])
7. # print("Iris feature data :: ", iris\_dataset['data'])
8. # print("Iris target :: ", iris\_dataset['target'])
9. def visuvalize\_sepal\_data():
   1. iris = datasets.load\_iris()
   2. X = iris.data[:, :2] # we only take the first two features.
   3. y = iris.target
   4. plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)#Coolwarm is a class in diverging color in color map. Slow variations
   5. plt.xlabel('Sepal length')
   6. plt.ylabel('Sepal width')
   7. plt.title('Sepal Width & Length')
   8. plt.show()
10. visuvalize\_sepal\_data()
11. def visuvalize\_petal\_data():
    1. iris = datasets.load\_iris()
    2. X = iris.data[:, 2:] # we only take the last two features.
    3. y = iris.target
    4. plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)
    5. plt.xlabel('Petal length')
    6. plt.ylabel('Petal width')
    7. plt.title('Petal Width & Length')
    8. plt.show()
12. visuvalize\_petal\_data()
13. iris = datasets.load\_iris()
14. X = iris.data[:, :2] # we only take the Sepal two features.
15. y = iris.target
16. C = 1.0 # SVM regularization parameter
17. # SVC with linear kernel
18. svc = svm.SVC(kernel='linear', C=C).fit(X, y)
19. # LinearSVC (linear kernel)
20. lin\_svc = svm.LinearSVC(C=C).fit(X, y)
21. # SVC with RBF kernel
22. rbf\_svc = svm.SVC(kernel='rbf', gamma=0.7, C=C).fit(X, y)
23. # SVC with polynomial (degree 3) kernel
24. poly\_svc = svm.SVC(kernel='poly', degree=3, C=C).fit(X, y)
25. h = .02 # step size in the mesh
26. # create a mesh to plot in
27. #Minimum and maximum values of individual features. X and Y are features and Z is predicted one
28. x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1
29. y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1
30. #ARANGE CREATES AN ARRAY WITH START AND END AS MAX AND MIN. h is the step
31. #Meshgrid is used to create matrices used for visualising
32. xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),
    * 1. np.arange(y\_min, y\_max, h))
33. # title for the plots
34. titles = ['SVC with linear kernel',
    1. 'LinearSVC (linear kernel)',
    2. 'SVC with RBF kernel',
    3. 'SVC with polynomial (degree 3) kernel']
35. for i, clf in enumerate((svc, lin\_svc, rbf\_svc, poly\_svc)):
    1. # Plot the decision boundary. For that, we will assign a color to each
    2. # point in the mesh [x\_min, x\_max]x[y\_min, y\_max].
    3. plt.subplot(2, 2, i + 1)
    4. plt.subplots\_adjust(wspace=0.4, hspace=0.4)
    5. Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()])
    6. # Put the result into a color plot
    7. Z = Z.reshape(xx.shape)
    8. plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
    9. # Plot also the training points
    10. plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)
    11. plt.xlabel('Sepal length')
    12. plt.ylabel('Sepal width')
    13. plt.xlim(xx.min(), xx.max())
    14. plt.ylim(yy.min(), yy.max())
    15. plt.xticks(())
    16. plt.yticks(())
    17. plt.title(titles[i])
36. plt.show()
37. iris = datasets.load\_iris()
38. X = iris.data[:, 2:] # we only take the last two features.
39. y = iris.target
40. C = 1.0 # SVM regularization parameter
41. # SVC with linear kernel
42. svc = svm.SVC(kernel='linear', C=C).fit(X, y)
43. # LinearSVC (linear kernel)
44. lin\_svc = svm.LinearSVC(C=C).fit(X, y)
45. #print(lin\_svc.predict(X))
46. # SVC with RBF kernel
47. rbf\_svc = svm.SVC(kernel='rbf', gamma=0.7, C=C).fit(X, y)
48. # SVC with polynomial (degree 3) kernel
49. poly\_svc = svm.SVC(kernel='poly', degree=3, C=C).fit(X, y)
50. h = .02 # step size in the mesh
51. # create a mesh to plot in
52. x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1
53. y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1
54. xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),
    * 1. np.arange(y\_min, y\_max, h))
55. # title for the plots
56. titles = ['SVC with linear kernel',
    1. 'LinearSVC (linear kernel)',
    2. 'SVC with RBF kernel',
    3. 'SVC with polynomial (degree 3) kernel']
57. for i, clf in enumerate((svc, lin\_svc, rbf\_svc, poly\_svc)):
58. # Plot the decision boundary. For that, we will assign a color to each
59. # point in the mesh [x\_min, x\_max]x[y\_min, y\_max].
60. plt.subplot(2, 2, i + 1)
61. plt.subplots\_adjust(wspace=0.4, hspace=0.4)
62. Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()])
63. # Put the result into a color plot
64. Z = Z.reshape(xx.shape)
65. plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
66. # Plot also the training points
67. plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)
68. plt.xlabel('Petal length')
69. plt.ylabel('Petal width')
70. plt.xlim(xx.min(), xx.max())
71. plt.ylim(yy.min(), yy.max())
72. plt.xticks(())
73. plt.yticks(())
74. plt.title(titles[i])
75. plt.show()

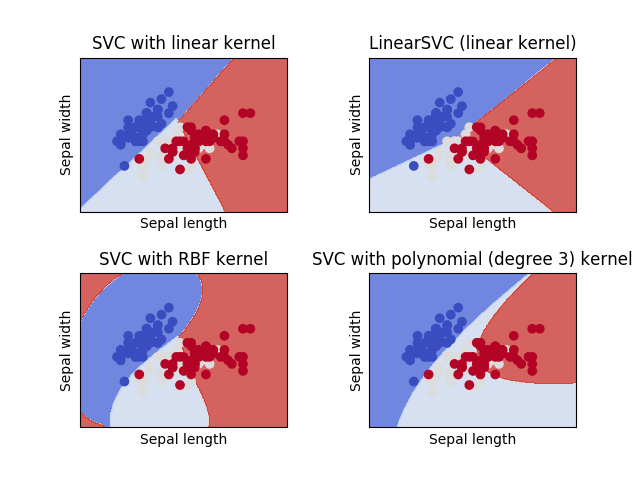
**Outputs**



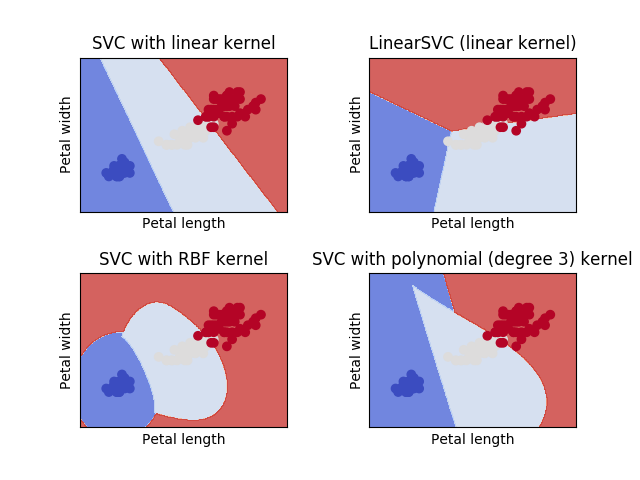
Sepal width and length dependency output. Different colors indicate different classes



Petal width and length dependency output. Different colors indicate different classes



SVM kernels linear, RBF and Polynomial for sepal



SVM kernels for linear, RBF and Polynomial for petal data.