

```

1 # In this code I have implemented logistic regression using batch gradient without
  feature scaling and found out the accuracy.
2 import numpy as np
3 import pandas as pd
4 import matplotlib.pyplot as plt
5
6 data=pd.read_excel(r'D:\Prakriti\2 VSCode Python\ML\Book1.xlsx')
7 data=np.array(data)
8 data=data[1:,:]
9 df=pd.DataFrame(data)
10
11 x=df.iloc[:,0:2]
12 y=df.iloc[:,2]
13
14 # filter out the applicants that got admitted
15 admitted=df.loc[y==1]
16 # print(admitted)
17 # filter out the applicants that din't get admission
18 not_admitted=df.loc[y==0]
19 plt.scatter(admitted.iloc[:, 0], admitted.iloc[:, 1], s=10, label='Admitted')
20 plt.scatter(not_admitted.iloc[:, 0], not_admitted.iloc[:, 1], s=10, label='Not
  Admitted')
21 plt.legend()
22 # plt.show()
23
24 one=np.ones((len(x),1))
25 x=np.append(one,x,axis=1)
26 y=np.array(y).reshape((len(y),1))
27
28 #splitting the data, 70% for training and 30% for testing
29 split_pct=int(0.7*len(x))
30 #print(split_pct)
31 train_x, test_x = x[:split_pct], x[split_pct:]
32 train_y, test_y= y[:split_pct], y[split_pct:]
33
34 # defining functions to use
35 def sigmoid(z):
36     return 1. / (1 + np.exp(-z))
37 def z(w, x):
38     return np.dot(x,w.T)
39 def hypothesis(w, x):
40     return sigmoid(z(w, x))
41 def costfunc(w,x,y):
42     one_case=-y*np.log(hypothesis(w,x))
43     zero_case=-(1-y)*np.log(1-hypothesis(w,x))
44     cost= one_case+zero_case
45     return (1/len(x))*sum(cost)
46 def gradient(theta, x, y):
47     # Computes the gradient of the cost function at the point theta
48     m = x.shape[0]
49     return (1 / m) * np.dot((hypothesis(theta,x) - y).T,x)
50 #Batch GDA
51 def batch_gradient_descent(X,Y,learning_rate,iterations):
52     cost_function = 0 # initialize our costfunc
53     m=X.shape[0]
54     theta=np.zeros(X.shape[1]).reshape(1,X.shape[1])
55     for i in range(0,iterations):
56         # prediction = Hypothesis
57         prediction =hypothesis(theta,X)

```

```

58         # print(prediction)
59         loss=prediction-Y
60
61         theta=theta-learning_rate* sum.gradient(theta,X,Y))
62     return theta
63
64 # Function to find ACCURACY
65 def accuracy(test_x,test_y,w,probab_threshold=0.5):
66     predicted_classes = (hypothesis(w,test_x) >= probab_threshold).astype(int)
67     predicted_classes = predicted_classes.flatten().astype(int)
68     test_y=test_y.flatten().astype(int)
69     # The following formula also gives the same result
70     # h=predicted_classes==test_y
71     # the sum() functn of an array with boolean values adds up only trues, i.e. 1
72     # In this case 27 are correct predictions out of 30
73     # accuracy = print((sum(h)/len(test_y))*100)
74     accuracy = np.mean(predicted_classes == test_y)
75     return accuracy * 100
76
77
78 # Finding parameters by Batch GDA
79 # Learning rate=0.01 , epochs=50000
80 w=batch_gradient_descent(train_x,train_y,0.01,50000)
81 # Finding accuracy
82 acc=accuracy(test_x,test_y,w)
83 print('The accuracy is: ', acc)
84 w=w.flatten()
85 # w=[-25.16131856, 0.20623159, 0.20147149]
86 print("The parameters are:", w)
87 # Taking two random values of X,
88 # to put in the equation of straight line to get the values of Y,
89 # Now having values of X and Y, plotting the graph.
90 x_values=np.linspace(30,100,2)
91 # to view X values
92 # print(x_values)
93 y_values = - (w[0] + np.dot(w[1], x_values)) / w[2]
94 # to view Y values
95 # print(y_values)
96 ''' I could have use this too...to randomly pick X values for plotting the Decision
Boundary
97 x_values = [np.min(x[:, 1] - 5), np.max(x[:, 2] + 5)]'''
98 plt.plot(x_values, y_values, label='Decision Boundary')
99 plt.show()
100
101
102 # Finding parameters by Stochastic GDA
103 # Learning rate=0.01 , epochs=50000
104 # w1=stochastic_gradient_descent(train_x,train_y,0.01,500)
105 # print(w1)
106
107
108
109

```

EXPLORER

ML

- __pycache__
- .vscode
- BatchGDA2.py
- BatchGDARegularization.py
- BatchGDAwithFS.py
- Book1.xlsx
- Cleveland.xlsx
- Hp.xlsx
- Linear Regression using Normal Equ...
- LogisticRegressionBGDA.py
- LogisticRegressionMIniBatchGD...
- LogisticRegressionStochasticGD...
- logisticregressionwithFS.py
- LRQ1.b.py
- LRQ2.py
- LWR.py
- MinibatchGDA.py
- rough.py
- rough2.py
- rough3.py
- rough4.py
- StochasticGDA.py
- tempCodeRunnerFile.py

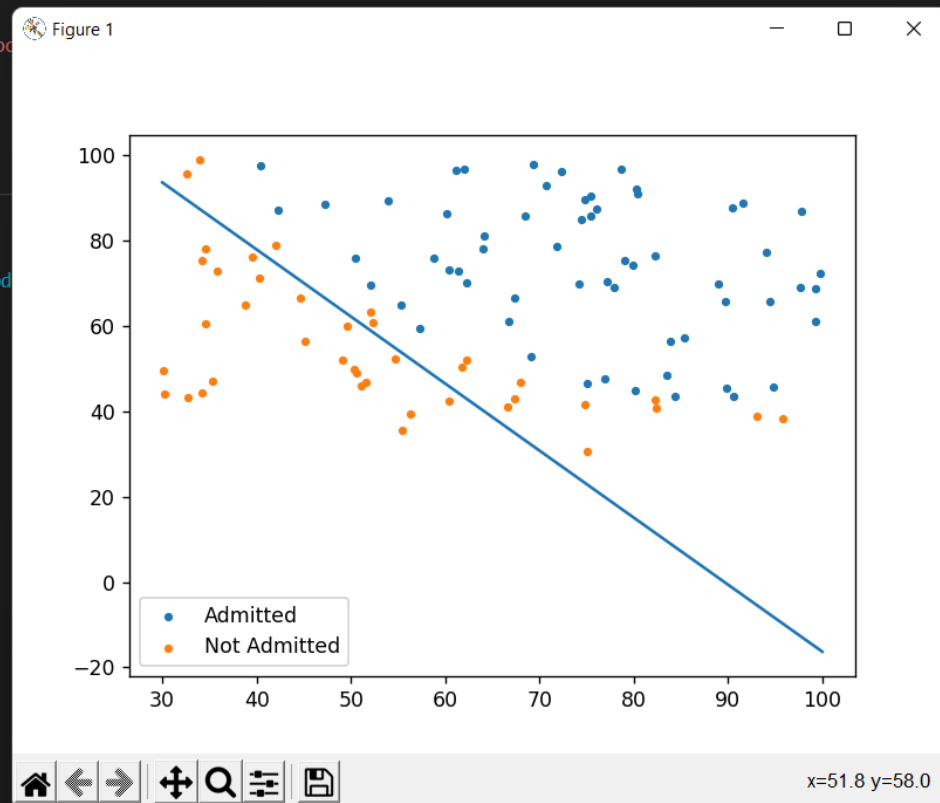
OUTLINE

TIMELINE

```
LogisticRegressionBGDA.py U X
LogisticRegressionBGDA.py > ...
1 # In this code I have implemented logistic regression using batch gradient without feature scaling and found out the accuracy.
2 import numpy as np
3 import pandas as pd
4 import matplotlib.pyplot as plt
5
6 data=pd.read_excel(r'D:\Prakriti\2 VSCode Python\ML\Book1.xlsx')
7 data=np.array(data)
8 data=data[1:,:]
9 df=pd.DataFrame(data)
10
11 x=df.iloc[:,0:2]
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
PS D:\Prakriti\2 VSCode Python\ML> python -u "d:\Prakriti\2 VSCode Python\ML\LogisticRegressionBGDA.py"
The accuracy is: 90.0
The parameters are: [-31.09844286  0.34704469  0.22116241]
```



```

1 # In this code I have implemented logistic regression using batch gradient with
  # feature scaling and found out the accuracy.
2 import numpy as np
3 import pandas as pd
4 import matplotlib.pyplot as plt
5
6 data=pd.read_excel(r'D:\Prakriti\2 VSCode Python\ML\Book1.xlsx')
7 data=np.array(data)
8 data=data[1:,:]
9 df=pd.DataFrame(data)
10
11 x=df.iloc[:,0:2]
12 y=df.iloc[:,2]
13
14 # filter out the applicants that got admitted
15 admitted=df.loc[y==1]
16 # print(admitted)
17 # filter out the applicants that didn't get admission
18 not_admitted=df.loc[y==0]
19 plt.scatter(admitted.iloc[:, 0], admitted.iloc[:, 1], s=10, label='Admitted')
20 plt.scatter(not_admitted.iloc[:, 0], not_admitted.iloc[:, 1], s=10, label='Not
  Admitted')
21 plt.legend()
22 # plt.show()
23
24 one=np.ones((len(x),1))
25 x=np.append(one,x,axis=1)
26 y=np.array(y).reshape((len(y),1))
27
28 # performing Feature scaling
29 # A=Marks in Exam1
30 # B=Marks in Exam2
31 A=x[:,1]
32 A_norm= (A-min(A))/(max(A)-min(A))
33 B=x[:,2]
34 B_norm=(B-min(A))/(max(B)-min(B))
35 X_norm=np.c_[x[:,0],A_norm,B_norm]
36
37 #splitting the data, 70% for training and 30% for testing
38 split_pct=int(0.7*len(X_norm))
39 #print(split_pct)
40 train_x, test_x = X_norm[:split_pct], X_norm[split_pct:]
41 train_y, test_y= y[:split_pct], y[split_pct:]
42
43 # defining functions to use
44 def sigmoid(z):
45     return 1. / (1 + np.exp(-z))
46 def z(w, x):
47     return np.dot(x,w.T)
48 def hypothesis(w, x):
49     return sigmoid(z(w, x))
50 def costfunc(w,x,y):
51     one_case=-y*np.log(hypothesis(w,x))
52     zero_case=-(1-y)*np.log(1-hypothesis(w,x))
53     cost= one_case+zero_case
54     return (1/len(x))*sum(cost)
55 def gradient(theta, x, y):
56     # Computes the gradient of the cost function at the point theta
57     m = x.shape[0]

```

```

58     return (1 / m) * np.dot((hypothesis(theta,x) - y).T,x)
59
60 def batch_gradient_descent(X,Y,learning_rate,iterations):
61     cost_function = 0 # initialize our costfunction
62     m=X.shape[0]
63     theta=np.zeros(X.shape[1]).reshape(1,X.shape[1])
64     for i in range(0,iterations):
65         # prediction = Hypothesis
66         prediction =hypothesis(theta,X)
67         # print(prediction)
68         loss=prediction-Y
69
70         theta=theta-learning_rate* sum(gradient(theta,X,Y))
71     return theta
72
73
74 # Function to find ACCURACY
75 def accuracy(test_x,test_y,w,probab_threshold=0.5):
76     predicted_classes = (hypothesis(w,test_x) >= probab_threshold).astype(int)
77     predicted_classes = predicted_classes.flatten().astype(int)
78     test_y=test_y.flatten().astype(int)
79     # The following formula also gives the same result
80     # h=predicted_classes==test_y
81     # the sum() functn of an array with boolean values adds up only trues, i.e. 1
82     # In this case 27 are correct predictions out of 30
83     # accuracy = print((sum(h)/len(test_y))*100)
84     accuracy = np.mean(predicted_classes == test_y)
85     return accuracy * 100
86
87
88 # Finding parameters by Batch GDA
89 # Learning rate=0.01 , epochs=50000
90 w=batch_gradient_descent(train_x,train_y,1,500000)
91 # Finding accuracy
92 acc=accuracy(test_x,test_y,w)
93 print('The accuracy is: ', acc)
94 w=w.flatten()
95 # w=[-25.16131856, 0.20623159, 0.20147149]
96 print("The parameters are:", w)
97 # Taking two random values of X,
98 # to put in the equation of straight line to get the values of Y,
99 # Now having values of X and Y, plotting the graph.
100 x_values=np.linspace(30,100,2)
101 # to view X values
102 # print(x_values)
103 y_values = - (w[0] + np.dot(w[1], x_values)) / w[2]
104 # to view Y values
105 # print(y_values)
106 ''' I could have use this too...to randomly pick X values for plotting the Decision
Boundary
107 x_values = [np.min(x[:, 1] - 5), np.max(x[:, 2] + 5)]'''
108 plt.plot(x_values, y_values, label='Decision Boundary')
109 plt.show()
110
111
112
113
114
115
116

```



```

1 # In this code I have implemented Logistic Regression using Stochastic without
  Feature scaling.
2 import numpy as np
3 import pandas as pd
4 import matplotlib.pyplot as plt
5
6 data=pd.read_excel(r'D:\Prakriti\2 VSCode Python\ML\Book1.xlsx')
7 data=np.array(data)
8 data=data[1:,:]
9 df=pd.DataFrame(data)
10
11 x=df.iloc[:,0:2]
12 y=df.iloc[:,2]
13
14 # filter out the applicants that got admitted
15 admitted=df.loc[y==1]
16 # print(admitted)
17 # filter out the applicants that din't get admission
18 not_admitted=df.loc[y==0]
19 plt.scatter(admitted.iloc[:, 0], admitted.iloc[:, 1], s=10, label='Admitted')
20 plt.scatter(not_admitted.iloc[:, 0], not_admitted.iloc[:, 1], s=10, label='Not
  Admitted')
21 plt.legend()
22 # plt.show()
23
24 one=np.ones((len(x),1))
25 x=np.append(one,x,axis=1)
26 y=np.array(y).reshape((len(y),1))
27
28
29 #splitting the data, 70% for training and 30% for testing
30 split_pct=int(0.7*len(x))
31 #print(split_pct)
32 train_x, test_x = x[:split_pct], x[split_pct:]
33 train_y, test_y= y[:split_pct], y[split_pct:]
34
35
36 # defining functions to use
37 def sigmoid(z):
38     return 1. / (1 + np.exp(-z))
39 def z(w, x):
40     return np.dot(x,w.T)
41 def hypothesis(w, x):
42     return sigmoid(z(w, x))
43 def costfunc(w,x,y):
44     one_case=-y*np.log(hypothesis(w,x))
45     zero_case=-(1-y)*np.log(1-hypothesis(w,x))
46     cost= one_case+zero_case
47     return (1/len(x))*sum(cost)
48 def gradient(theta, x, y):
49     # Computes the gradient of the cost function at the point theta
50     m = x.shape[0]
51     return (1 / m) * np.dot((hypothesis(theta,x) - y).T,x)
52
53
54 #Stochastic GDA
55 def stochastic_gradient_descent(x,y,learning_rate,iterations):
56     theta = np.zeros(x.shape[1])
57     for i in range(iterations):

```

```

58     random_index=np.random.randint(len(x),dtype='int')
59     xi = x[random_index,0:3].reshape(1,x.shape[1])
60     yi = y[random_index,0].reshape(1,1)
61
62     prediction = hypothesis(theta,xi)
63     loss=prediction-yi
64     gradient=np.dot(loss,xi)
65     # Updating the parameters i.e.theta here
66     theta = theta - learning_rate*gradient
67     return theta
68
69 # Function to find ACCURACY
70 def accuracy(test_x,test_y,w,probab_threshold=0.5):
71     predicted_classes = (hypothesis(w,test_x) >= probab_threshold).astype(int)
72     predicted_classes = predicted_classes.flatten().astype(int)
73     test_y=test_y.flatten().astype(int)
74     # The following formula also gives the same result
75     # h=predicted_classes==test_y
76     # the sum() functn of an array with boolean values adds up only trues, i.e. 1
77     # In this case 27 are correct predictions out of 30
78     # accuracy = print((sum(h)/len(test_y))*100)
79     accuracy = np.mean(predicted_classes == test_y)
80     return accuracy * 100
81
82
83 # Finding parameters by Stochastic GDA
84 # Learning rate=0.01 , epochs=50000
85 w=stochastic_gradient_descent(train_x,train_y,0.01,50000)
86 # Finding accuracy
87 acc=accuracy(test_x,test_y,w)
88 print('The accuracy is: ', acc)
89 w=w.flatten()
90 # w=[-25.16131856, 0.20623159, 0.20147149]
91 print("The parameters are:", w)
92 # Taking two random values of X,
93 # to put in the equation of straight line to get the values of Y,
94 # Now having values of X and Y, plotting the graph.
95 x_values=np.linspace(30,100,2)
96 # to view X values
97 # print(x_values)
98 y_values = - (w[0] + np.dot(w[1], x_values)) / w[2]
99 # to view Y values
100 # print(y_values)
101 ''' I could have use this too...to randomly pick X values for plotting the Decision
    Boundary
102 x_values = [np.min(x[:, 1] - 5), np.max(x[:, 2] + 5)]'''
103 plt.plot(x_values, y_values, label='Decision Boundary')
104 plt.show()
105
106
107
108
109
110

```


FileEditSelectionViewGoRunTerminalHelp

EXPLORER

ML

__pycache__

.vscode

BatchGDA2.py

BatchGDARegularization.py

BatchGDAwithFS.py

Book1.xlsx

Cleveland.xlsx

Hp.xlsx

Linear Regression using Normal Equ...

LogisticRegressionBGDA.py

LogisticRegressionStochasticGD...

logisticregressionwithFS.py

LRQ1.b.py

LRQ2.py

LWR.py

MinibatchGDA.py

rough.py

rough2.py

rough3.py

rough4.py

StochasticGDA.py

tempCodeRunnerFile.py

LogisticRegressionStochasticGDA.py

Book1.xlsx

LogisticRegressionStochasticGDA.py > ...

```
1 # In this code I have implemented Logistic Regression using Stochastic without Feature scaling.
2 import numpy as np
3 import pandas as pd
4 import matplotlib.pyplot as plt
5
6 data=pd.read_excel(r'D:\Prakriti\2 VSCode Python\ML\Book1.xlsx')
7 data=np.array(data)
8 data=data[1:,:]
9 df=pd.DataFrame(data)
10
11 x=df.iloc[:,0:2]
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

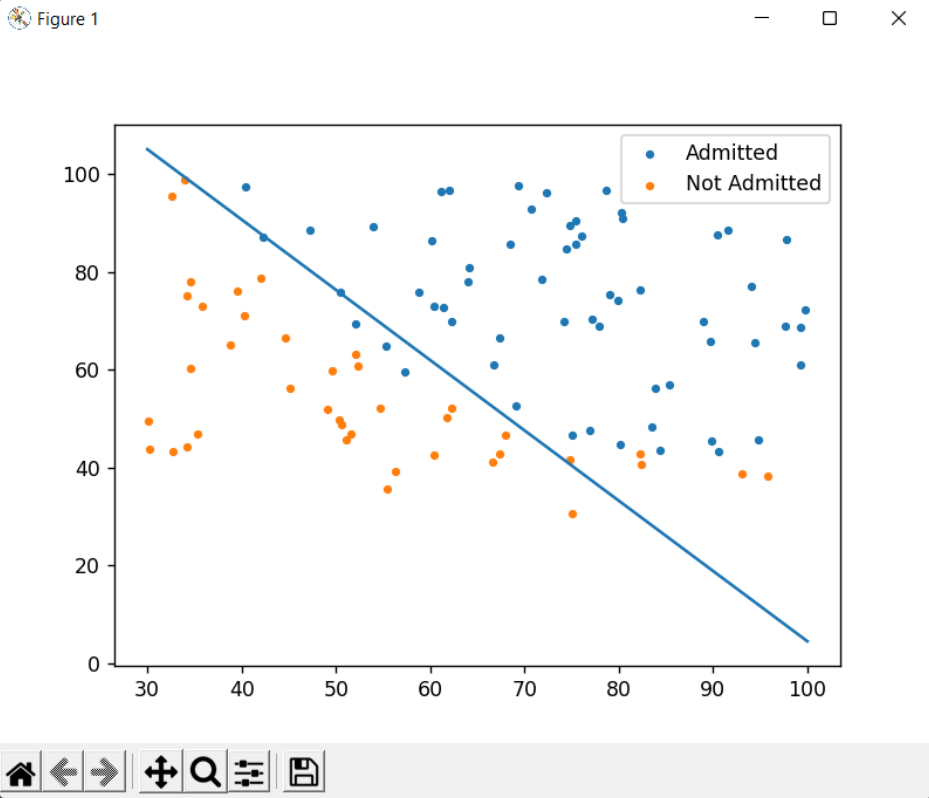
TERMINAL

JUPYTER

PS D:\Prakriti\2 VSCode Python\ML>

```
> python -u "d:\Prakriti\2 VSCode Python\ML\LogisticRegressionStochasticGDA.py"
The accuracy is: 86.66666666666667
The parameters are: [-35.9484987  0.3485414  0.24252846]
```

Figure 1



master*

0 0 0

Ln 1, Col 96 Spaces: 4 UTF-8 CRLF Python 3.10.5 64-bit

28°C Cloudy

Windows Taskbar Icons

22:49 09-07-2022

```

1 # In this code I have implemented Logistic Regression using Mini Batch without
  Feature scaling.
2 import numpy as np
3 import pandas as pd
4 import matplotlib.pyplot as plt
5
6 data=pd.read_excel(r'D:\Prakriti\2 VSCode Python\ML\Book1.xlsx')
7 data=np.array(data)
8 data=data[1:,:]
9 df=pd.DataFrame(data)
10
11 x=df.iloc[:,0:2]
12 y=df.iloc[:,2]
13
14 # filter out the applicants that got admitted
15 admitted=df.loc[y==1]
16 # print(admitted)
17 # filter out the applicants that din't get admission
18 not_admitted=df.loc[y==0]
19 plt.scatter(admitted.iloc[:, 0], admitted.iloc[:, 1], s=10, label='Admitted')
20 plt.scatter(not_admitted.iloc[:, 0], not_admitted.iloc[:, 1], s=10, label='Not
  Admitted')
21 plt.legend()
22 # plt.show()
23
24 one=np.ones((len(x),1))
25 x=np.append(one,x,axis=1)
26 y=np.array(y).reshape((len(y),1))
27
28 #splitting the data, 70% for training and 30% for testing
29 split_pct=int(0.7*len(x))
30 #print(split_pct)
31 train_x, test_x = x[:split_pct], x[split_pct:]
32 train_y, test_y= y[:split_pct], y[split_pct:]
33
34
35 # defining functions to use
36 def sigmoid(z):
37     return 1. / (1 + np.exp(-z))
38 def z(w, x):
39     return np.dot(x,w.T)
40 def hypothesis(w, x):
41     return sigmoid(z(w, x))
42 def costfunc(w,x,y):
43     one_case=-y*np.log(hypothesis(w,x))
44     zero_case=-(1-y)*np.log(1-hypothesis(w,x))
45     cost= one_case+zero_case
46     return (1/len(x))*sum(cost)
47 def gradient(theta, x, y):
48     # Computes the gradient of the cost function at the point theta
49     m = x.shape[0]
50     return (1 / m) * np.dot((hypothesis(theta,x) - y).T,x)
51
52
53 #Mini Batch GDA
54 def mini_batch_gradient_descent(x,y,iterations,learning_rate,batch_size=20):
55     theta = np.zeros(x.shape[1]).reshape(1,x.shape[1])
56     shuffled_indices = np.random.permutation(len(y))
57     x_shuffled = x[shuffled_indices]

```

```

58     y_shuffled = y[shuffled_indices]
59     for i in range(iterations):
60         xi = x_shuffled[i:i+batch_size]
61         yi = y_shuffled[i:i+batch_size]
62         prediction = hypothesis(theta,xi)
63         loss=prediction-yi
64         gradient=sum(np.dot(loss.T,xi))
65         # Updating the parameters i.e.theta here
66         theta =theta - (learning_rate * gradient)/batch_size
67
68     return theta
69
70 # Function to find ACCURACY
71 def accuracy(test_x,test_y,w,probab_threshold=0.5):
72     predicted_classes = (hypothesis(w,test_x) >= probab_threshold).astype(int)
73     predicted_classes = predicted_classes.flatten().astype(int)
74     test_y=test_y.flatten().astype(int)
75     # The following formula also gives the same result
76     # h=predicted_classes==test_y
77     # the sum() functn of an array with boolean values adds up only trues, i.e. 1
78     # In this case 27 are correct predictions out of 30
79     # accuracy = print((sum(h)/len(test_y))*100)
80     accuracy = np.mean(predicted_classes == test_y)
81     return accuracy * 100
82
83
84 # Finding parameters by Mini Batch GDA
85 # Learning rate=0.01 , epochs=50000
86 w=mini_batch_gradient_descent(train_x,train_y,25,0.0001)
87 # Finding accuracy
88 acc=accuracy(test_x,test_y,w)
89 print('The accuracy is: ', acc)
90 w=w.flatten()
91 # w=[-25.16131856, 0.20623159, 0.20147149]
92 print("The parameters are:", w)
93 # Taking two random values of X,
94 # to put in the equation of straight line to get the values of Y,
95 # Now having values of X and Y, plotting the graph.
96 x_values=np.linspace(30,100,2)
97 # to view X values
98 # print(x_values)
99 y_values = - (w[0] + np.dot(w[1], x_values)) / w[2]
100 # to view Y values
101 # print(y_values)
102 '''I could have use this too...to randomly pick X values for plotting the Decision
    Boundary
103 x_values = [np.min(x[:, 1] - 5), np.max(x[:, 2] + 5)] '''
104 plt.plot(x_values, y_values, label='Decision Boundary')
105 plt.show()
106
107
108
109
110
111
112

```

```
1 # In this code I have implemented Logistic Regression for multiclass Regression using
  sklearn library on Cleveland data
2 import pandas as pd
3 import numpy as np
4 from sklearn.linear_model import LogisticRegression
5
6 from warnings import simplefilter
7 # ignore all future warnings
8 simplefilter(action='ignore', category = FutureWarning)
9
10
11 df = pd.read_excel(r'D:\Prakriti\2 VSCode Python\ML\Cleveland.xlsx')
12
13
14 df.columns = ['age', 'sex', 'cp', 'trestbps', 'chol',
15               'fbs', 'restecg', 'thalach', 'exang',
16               'oldpeak', 'slope', 'ca', 'thal', 'target']
17 # print(df)
18 X = df.iloc[:, :-1].values
19 y = df.iloc[:, -1].values
20
21 from sklearn.model_selection import train_test_split
22 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
23               random_state = 0)
24
25 from sklearn.linear_model import LogisticRegression
26 classifier = LogisticRegression()
27 classifier.fit(X_train, y_train)
28
29 # Predicting the Test set results
30 y_pred = classifier.predict(X_test)
31
32 from sklearn.metrics import confusion_matrix
33 cm_test = confusion_matrix(y_pred, y_test)
34
35 y_pred_train = classifier.predict(X_train)
36 cm_train = confusion_matrix(y_pred_train, y_train)
37
38 print()
39 print('Accuracy for training set for Logistic Regression = {}'.format((cm_train[0][0]
40   + cm_train[1][1])/len(y_train)))
41 print('Accuracy for test set for Logistic Regression = {}'.format((cm_test[0][0] +
42   cm_test[1][1])/len(y_test)))
43
44
```

 LRQ2.py U X LRQ2.py > ...

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

Open file in editor (ctrl + click)

```
Open file in editor (ctrl + click) | C:\Users\hp\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\linear_model\logistic.py:444: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression
n_iter_i = _check_optimize_result(
[[31  9]
 [ 4 17]]
```

```
Accuracy for training set for Logistic Regression = 0.8512396694214877
Accuracy for test set for Logistic Regression = 0.7868852459016393
PS D:\Prakriti\2 VSCode Python\ML>
```

 $+, \vee, \wedge, \times$

- > Code
- > Code
- > Code
- > Code
- > Code
- ≥ Code
- ≥ Code
- ≥ Code
- ≥ Code
- ≥ Code