Lesson_10_Matplotlib_Logistic_Regression

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(42)
```

In [263]:

```
data = pd.read_csv("data_1.csv")
data.head()
```

Out[263]:

	p	q	label
0	0.78051	-0.063669	1
1	0.28774	0.291390	1
2	0.40714	0.178780	1
3	0.29230	0.421700	1
4	0.50922	0.352560	1

In [327]:

```
X = data.iloc[:,:2].values
y = data.iloc[:,2]
```

In [328]:

```
admitted = X[np.argwhere(y == 0)]
rejected = X[np.argwhere(y==1)]
```

In [329]:

```
def graph_prop(title,xlabel,ylabel,grid,xlimit_min,xlimit_max,ylimit_min,y_limit_max):
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.grid(grid)
    plt.xlim(xlimit_min,xlimit_max)
    plt.ylim(ylimit_min,y_limit_max)
```

In [330]:

```
def plt_scatter(data_list,color,s,edgecolor,label):
    plt.scatter([p[0][0] for p in data_list],
                 [p[0][1] for p in data_list],
                color = color, s = s, edgecolor = edgecolor,label=label)
    if label != 0:
        plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
```

In [331]:

```
def initial_plot():
    plt_scatter(admitted,'blue',15,'blue','Admitted')
    plt_scatter(rejected,'cyan',15,'k','Rejected')
    graph_prop('Admission','Test (GRE)','Grades (GPA)',True,-0.0,1.05,0.0,1.05)
```

In [332]:

```
def minIndexDefinition(e_l):
    min_misclassified = min(e_l)
    idx = [i for i,x in enumerate(e_l) if x == min_misclassified]
    print("To achive minimum it would be enough ", idx[0]+1, "epochs")
    return idx[0]
```

In [333]:

```
def best_solution(b_l,e_l):
    idx = minIndexDefinition(e_l)
    best_x = b_l[idx][0][0]
    best_y = b_l[idx][1][0]
    return best_x,best_y
```

In [334]:

```
def draw_graph(point,line_style,line_color):
        plt_scatter(admitted,'blue',15,'blue',0)
        plt_scatter(rejected,'cyan',15,'k',0)
        graph_prop('Admission','Test (GRE)','Grades (GPA)',True,-0.0,1.05,0.0,1.05)
        plt.plot((point[0],0),(0,point[1]),line_style,color=line_color)
```

In [335]:

```
def modelFunction(X,W,b):
    model = (np.matmul(X,W)+b)
    return model
```

In [336]:

```
def activationSigmoid(x):
    sigmoid = 1/(1+np.exp(-x))
    return sigmoid
```

In [337]:

```
def predictionSigmoid(X, W, b):
    score = modelFunction(X,W,b)
    prediction = activationSigmoid(score)
    return prediction
```

In [338]:

```
def error_vector(y, y_hat):
    error_vector = [-y[i]*np.log(y_hat[i]) - (1-y[i])*np.log(1-y_hat[i]) for i in 
range(len(y))]
    return error_vector
```

In [339]:

```
def errorSigmoid_entropy(y, y_hat):
    error_vector = error_vector(y, y_hat)
    entropy = sum(error_vector)/len(error_vector)
    return entropy
```

In [340]:

```
def sigmoid_prime(x):
    return sigmoid(x)*(1-sigmoid(x))
```

In [341]:

```
def evaluationFunction(y,y_hat):
    evaluation = [y[i] - y_hat[i]  for i in range(len(y))]
    return evaluation
```

In [403]:

```
# TODO: Fill in the code below to calculate the gradient of the error function.
# The result should be a list of three lists:
# The first list should contain the gradient (partial derivatives) with respect to w1
# The second list should contain the gradient (partial derivatives) with respect to w2
# The third list should contain the gradient (partial derivatives) with respect to b
```

```
def dErrors(X, W,evaluation):
    DErrorsDx = []
    for j in range(len(W)):
        DErrorsDx_i = [-X[i][0]*evaluation[i] for i in range(len(y))]
        DErrorsDx.append(DErrorsDx_i)
    DErrorsDb = [-(evaluation[i]) for i in range(len(y))]
    return DErrorsDx, DErrorsDb
```

In [404]:

```
def args_modification(X,W,b,learn_rate,evaluation,derivErrors):
    for i in range(len(W)):
        W[i] -= sum(derivErrors[0][0])*learn_rate
    b -= sum(derivErrors[1])*learn_rate
    return W, b
```

In [405]:

```
# TODO: Fill in the code below to implement the gradient descent step.
# The function should receive as inputs the data X, the labels y,
# the weights W (as an array), and the bias b.
# It should calculate the prediction, the gradients, and use them to
# update the weights and bias W, b. Then return W and b.
# The error e will be calculated and returned for you, for plotting purposes.
def gradientDescentStep(X, y, W, b, learn_rate = 0.01):
    y_hat = predictionSigmoid(X,W,b)
    evaluation = evaluationFunction(y,y_hat)
    entropy = sum(error_vector(y, y_hat))
    derivErrors = dErrors(X, W,evaluation)
    W, b = args_modification(X,W,b,learn_rate,evaluation,derivErrors)
    return W, b, entropy
```

In [406]:

```
def start_initialization(X):
    x_min, x_max = min(X.T[0]), max(X.T[0])
    y_min, y_max = min(X.T[1]), max(X.T[1])
    W = np.array(np.random.rand(2,1))
    b = np.random.rand(1)[0] + x_max
    return W,b
```

In [407]:

```
# This function runs the perceptron algorithm repeatedly on the dataset, and returns
# a few of the boundary lines obtained in the iterations, for plotting purposes. Feel
# free to play with the learn_rate and num_epochs, and see your results plotted below.
def trainLR(X, y, learn_rate, num_epochs):

    W, b = start_initialization(X)

    # These are the solution lines that get plotted below.
    boundary_lines = []
    error_list = []

    for i in range(num_epochs):
        # In each epoch, we apply the gradient descent step.
        W, b, error = gradientDescentStep(X, y, W, b, learn_rate)
        boundary_lines.append((-b/W[0], -b/W[1]))
        error_list.append(int(error))
    print("Minimal errors ",min(error_list))
    return boundary_lines, error_list
```

In [408]:

```
b_l, e_l = trainLR(X,y,learn_rate = 0.01, num_epochs = 95)
```

```
Minimal errors  32
```

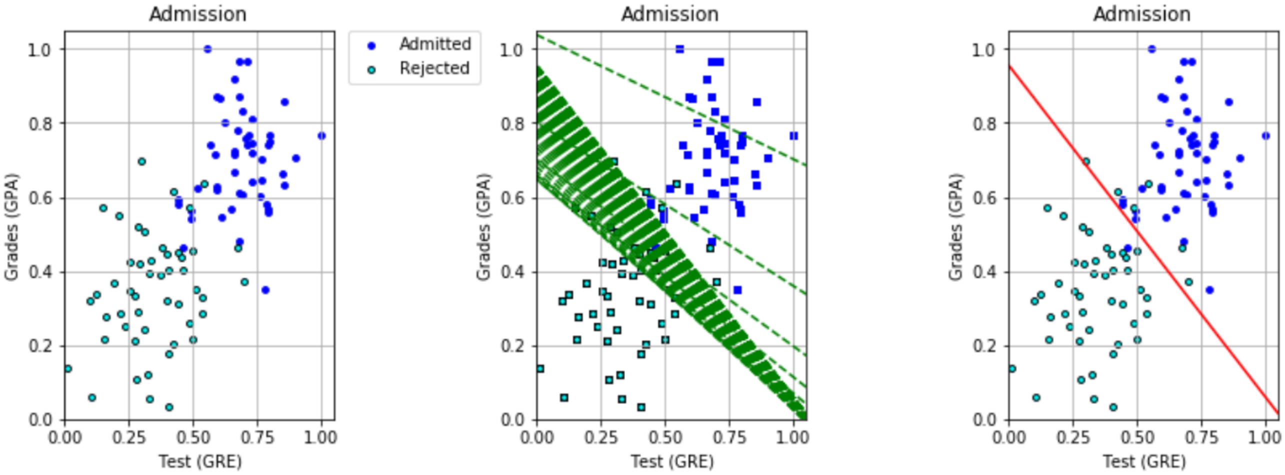
In [409]:

```
np.random.seed(42)
bestx, besty = best_solution(b_l,e_l)
```

```
To achive minimum it would be enough  93 epochs
```

In [410]:

```
fig1 = plt.figure()

ax1 = fig1.add_subplot(1,3,1)
initial_plot()

ax2 = fig1.add_subplot(1,3,2)
for i in range(len(b_l)):
    draw_graph(b_l[i],'k--','green')

ax3 = fig1.add_subplot(1,3,3)
draw_graph((bestx,besty),'k','red')

plt.subplots_adjust(top=0.92, bottom=0.08, left=0.20, right=1.95, hspace=2.95,
wspace=0.75)

plt.show()
```


Admission
Admitted
Rejected
Admission
Admission
Grades (GPA)
Test (GRE)