AIR UNIVERSITY

Department of Electrical and Computer Engineering

AI for Engineers Lab

Lab #7: Logistic Regression

Student Name:

Roll No:

Instructor: Muhammad Awais

Logistic Regression

In this lab, you will build a logistic regression model to predict whether a student gets admitted into a university.

Problem Statement

Suppose that you are the administrator of a university department and you want to determine each applicant's chance of admission based on their results on two exams.

You have historical data from previous applicants (check -> ex2data1.txt) that you can use as a training set for logistic regression. For each training example, you have the applicant's scores on two exams and the admissions decision. Your task is to build a classification model that estimates an applicant's probability of admission based on the scores from those two exams.

Sigmoid function

For logistic regression, the model is represented as

$$f_{\mathbf{w},b}(x) = g(\mathbf{w} \cdot \mathbf{x} + b)$$

where function g is the sigmoid function. The sigmoid function is defined as:

Cost function for logistic regression

You will implement the cost function for logistic regression.

For logistic regression, the cost function is of the form

$$J(\mathbf{w},b) = \frac{1}{m}\sum_{i=0}^{m-1} \left[loss(f_{\mathbf{w},b}(\mathbf{x}^{(i)}), y^{(i)}) \right] \tag{1}$$

where

- m is the number of training examples in the dataset
- $loss(f_{\mathbf{w},b}(\mathbf{x}^{(i)}), y^{(i)})$ is the cost for a single data point, which is -

$$loss(f_{\mathbf{w},b}(\mathbf{x}^{(i)}), y^{(i)}) = (-y^{(i)} \log\left(f_{\mathbf{w},b}\left(\mathbf{x}^{(i)} \right) \right) - \left(1 - y^{(i)}\right) \log \left(1 - f_{\mathbf{w},b}\left(\mathbf{x}^{(i)} \right) \right) \quad ($$

- $f_{\mathbf{w},b}(\mathbf{x}^{(i)})$ is the model's prediction, while $y^{(i)}$, which is the actual label
- $f_{\mathbf{w},b}(\mathbf{x}^{(i)}) = g(\mathbf{w} \cdot \mathbf{x^{(i)}} + b)$ where function g is the sigmoid function.

Gradient for logistic regression

You will implement the gradient for logistic regression.

Recall that the gradient descent algorithm is:

$$\begin{align*}& \text{repeat until convergence:} \; \lbrace \\ & b := b - \alpha \frac{\partial J(\mathbf{w},b)}{\partial b} \\ & w_j := w_j - \alpha \frac{\partial J(\mathbf{w},b)}{\partial w_j} \tag{1} \; & \text{for j := 0..n-1}\\ & \rbrace\end{align*}$$

where, parameters b, w_j are all updated simultaniously

You will compute $\frac{\partial J(\mathbf{w},b)}{\partial w}$, $\frac{\partial J(\mathbf{w},b)}{\partial b}$ from equations (2) and (3) below.

$$\frac{\partial J(\mathbf{w},b)}{\partial b} = \frac{1}{m} \sum\limits_{i = 0}^{m-1} (f_{\mathbf{w},b}(\mathbf{x}^{(i)}) - \mathbf{y}^{(i)}) \tag{2}$$

$$\frac{\partial J(\mathbf{w},b)}{\partial w_j} = \frac{1}{m} \sum\limits_{i = 0}^{m-1} (f_{\mathbf{w},b}(\mathbf{x}^{(i)}) - \mathbf{y}^{(i)})x_{j}^{(i)} \tag{3}$$

- m is the number of training examples in the dataset
- $f_{\mathbf{w},b}(x^{(i)})$ is the model's prediction, while $y^{(i)}$ is the actual label

- **Note**: While this gradient looks identical to the linear regression gradient, the formula is actually different because linear and logistic regression have different definitions of

$f_{\mathbf{w},b}(x)$.

Learning parameters using gradient descent

Similar to the previous lab, you will find the optimal parameters of a logistic regression model by using gradient descent.

- A good way to verify that gradient descent is working correctly is to look at the value of $J(\mathbf{w}, b)$ and check that it is decreasing with each step.
- Assuming you have implemented the gradient and computed the cost correctly, your value of $J(\mathbf{w}, b)$ should never increase, and should converge to a steady value by the end of the algorithm.

In [1]:
```
!pip install utils
```

```
Collecting utils
  Downloading utils-1.0.2.tar.gz (13 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: utils
  Building wheel for utils (setup.py) ... done
  Created wheel for utils: filename=utils-1.0.2-py2.py3-none-any.whl size=139
06 sha256=06d17b96058e2696a13f104d272af13f33bd973781be74ae290bf65d93d989e3
  Stored in directory: /root/.cache/pip/wheels/b8/39/f5/9d0ca31dba85773ececf0
a7f5469f18810e1c8a8ed9da28ca7
Successfully built utils
Installing collected packages: utils
Successfully installed utils-1.0.2
```

In [64]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from utils import *
import math

%matplotlib inline
```

In [65]:
```
# load dataset
data = pd.read_csv('ex2data1.txt', sep=",", header=None)       #store the give
data.columns = ["Exam_1_Score", "Exam_2_Score", "Status"]        #data loaded
data.head(5)          #checking if the data is loaded correctly
```

Out[65]:

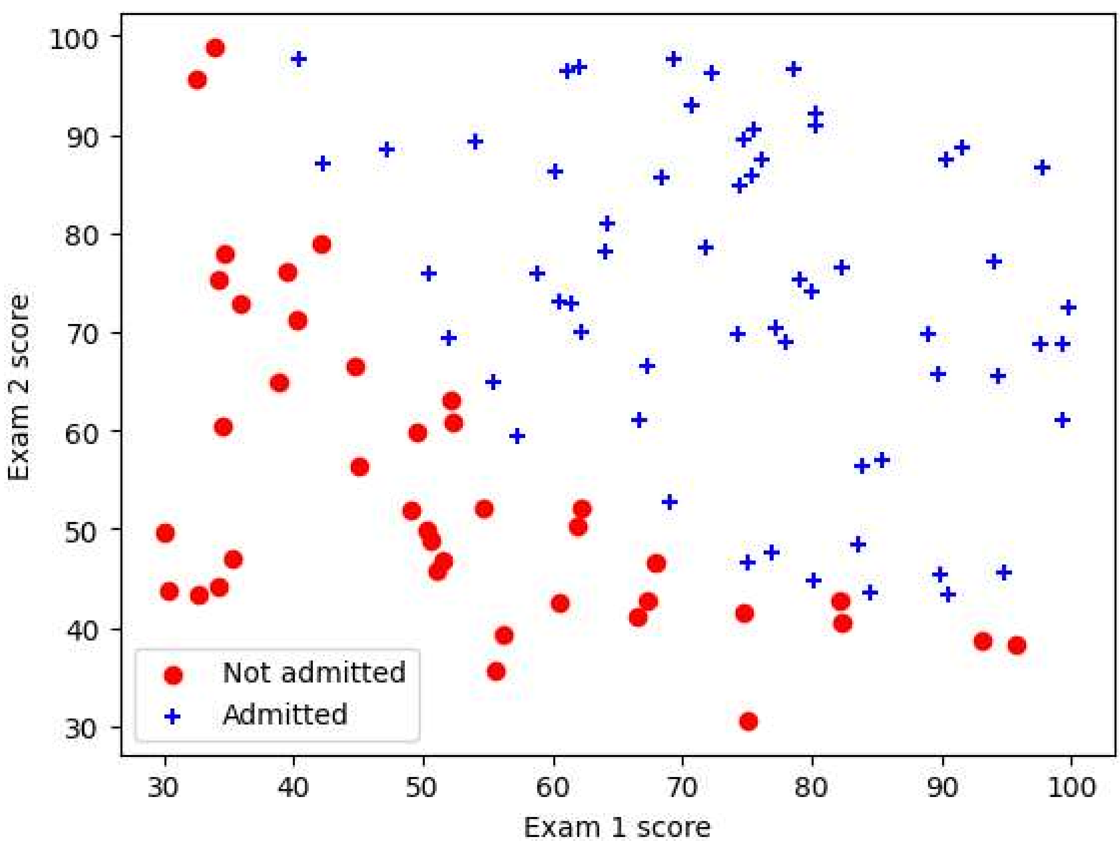
	Exam_1_Score	Exam_2_Score	Status
0	34.623660	78.024693	0
1	30.286711	43.894998	0
2	35.847409	72.902198	0
3	60.182599	86.308552	1
4	79.032736	75.344376	1

In [66]:
```
x1 = data.Exam_1_Score.to_numpy().reshape(data.shape[0],1)     #pandas to nump
x2 = data.Exam_2_Score.to_numpy().reshape(data.shape[0],1)
Y = data.Status.to_numpy().reshape(data.shape[0],1)
X = np.hstack((x1,x2))
X.shape
```

Out[66]: (100, 2)

In [67]:
```
#Plotting the data
for i in range(0,len(Y)):

    if Y[i]==0:
       a = plt.scatter(x1[i], x2[i],color="r")

    else:
       b = plt.scatter(x1[i],x2[i],color="b",marker='+')

plt.legend((a,b),('Not admitted','Admitted'))

plt.xlabel("Exam 1 score")
plt.ylabel("Exam 2 score")
```

Out[67]: Text(0, 0.5, 'Exam 2 score')



Theta Intialization

In [262]:
```
theta = [-1, 0.3, 7]
```

Hypothesis

In [263]:
```
#Sigmoid Function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Initialize hypothesis array with zeros
hypothesis = np.zeros(100)

# Populate the hypothesis array with predictions
for i in range(X.shape[0]):
    hypothesis[i] = sigmoid(X[i][0] * theta[0] + X[i][1] * theta[1] + theta[2]

print("Hypothesis:", hypothesis)
```

```
Hypothesis: [1.45393307e-02 4.03365784e-05 9.32461906e-04 1.40633546e-12
 3.41251285e-22 6.28072228e-10 1.19148360e-11 3.33176011e-24
 2.40247866e-19 1.10470755e-28 2.44469778e-34 2.81379188e-26
 1.81664747e-23 4.43618700e-15 5.96297755e-05 1.67266595e-09
 8.21764613e-21 4.10006861e-21 2.88058472e-16 6.41495955e-25
 2.30202899e-21 4.64325965e-28 2.87358787e-13 8.74858854e-07
 1.52880837e-22 1.28930087e-15 1.13147303e-26 4.53180526e-33
 5.43662258e-18 4.61082333e-06 7.40276176e-15 2.41300609e-27
 4.31168386e-12 3.04313822e-11 6.88217232e-06 1.30375885e-14
 9.39649098e-01 5.12538527e-15 9.46308968e-25 9.83110758e-03
 8.66955717e-27 5.72795236e-14 3.68717839e-30 3.62953810e-28
 6.92577506e-14 6.31011245e-18 4.95946127e-22 7.55126513e-29
 4.89691080e-12 6.76069492e-26 9.61741005e-23 7.48192745e-33
 2.34106115e-31 8.22544247e-05 4.89626042e-13 1.98266937e-11
 4.02681500e-31 9.56990157e-01 7.79098233e-21 1.20458883e-18
 2.94905949e-19 6.94579420e-07 5.30458127e-17 2.79863048e-04
 1.98862778e-08 3.06798779e-21 1.48178307e-02 3.07535839e-12
 1.50222312e-20 9.97235142e-19 2.95892633e-06 2.48174610e-15
 1.43784455e-16 2.02750301e-14 2.34007852e-13 1.30089913e-31
 1.09855958e-06 1.00809649e-09 2.10328038e-18 7.85919817e-28
 3.31029650e-27 6.41791556e-33 3.01644854e-18 8.62232417e-15
 9.73351729e-21 2.86088733e-16 1.09515486e-05 1.10096973e-18
 3.02805521e-19 1.67174619e-12 1.69178301e-28 1.44587386e-25
 3.79239021e-17 5.52877020e-19 8.51488884e-31 1.21497136e-27
 1.08294756e-04 7.39154672e-32 2.92463607e-13 1.69822322e-18]
```

LikeliHood

In [264]:
```
import numpy as np

# Cost function for logistic regression
def cost_function(theta, X, y):
    m = len(y)
    loss = -y * np.log(hypothesis) - (1 - y) * np.log(1 - hypothesis)
    cost = (1 / m) * np.sum(loss)
    return cost

# Testing the cost function with provided theta values
cost = cost_function(theta, X, Y)

print("Cost:", cost)
```

```
Cost: 2332.655369476143
```

Theta Update / Gradient Desent

In [265]:
```
# Set learning rate
alpha = 0.01

# Perform gradient descent for each data point
for i in range(X.shape[0]):
    # Calculate hypothesis
    z = theta[0] + theta[1] * X[i][0] + theta[2] * X[i][1]

    # Update parameters
    theta[0] -= alpha * (hypothesis - Y[i])  # Update theta0
    theta[1] -= alpha * (hypothesis - Y[i]) * X[i][0]  # Update theta1
    theta[2] -= alpha * (hypothesis - Y[i]) * X[i][1]  # Update theta2

theta = [theta[0][0], theta[1][0], theta[2][0]]
print("Optimized theta:", theta)
```

```
Optimized theta: [-0.41453933065834725, 44.176929811606044, 50.4110177225124
1]
```

Iterations

In [266]:
```python
# Number of iterations
num_iters = 10000

# Perform gradient descent
for iter in range(num_iters):
    # Initialize cost for this iteration
    cost_iter = 0

    # Update parameters for each data point
    for i in range(X.shape[0]):
        # Calculate hypothesis
        z = theta[0] + theta[1] * X[i][0] + theta[2] * X[i][1]
        hypothesis = sigmoid(z)

        # Update parameters
        theta[0] -= alpha * (hypothesis - Y[i])  # Update theta0
        theta[1] -= alpha * (hypothesis - Y[i]) * X[i][0]  # Update theta1
        theta[2] -= alpha * (hypothesis - Y[i]) * X[i][1]  # Update theta2

        # Update cost for this data point
        cost_iter += cost_function(theta, X[i], Y[i])

    # Compute average cost for this iteration
    avg_cost = cost_iter / X.shape[0]

    # Print cost and theta for this iteration
    theta = [theta[0][0], theta[1][0], theta[2][0]]
    print(f"Iteration {iter+1} - Theta: {theta}")

# Print final optimized theta
print("Final Optimized Theta:", theta)
```

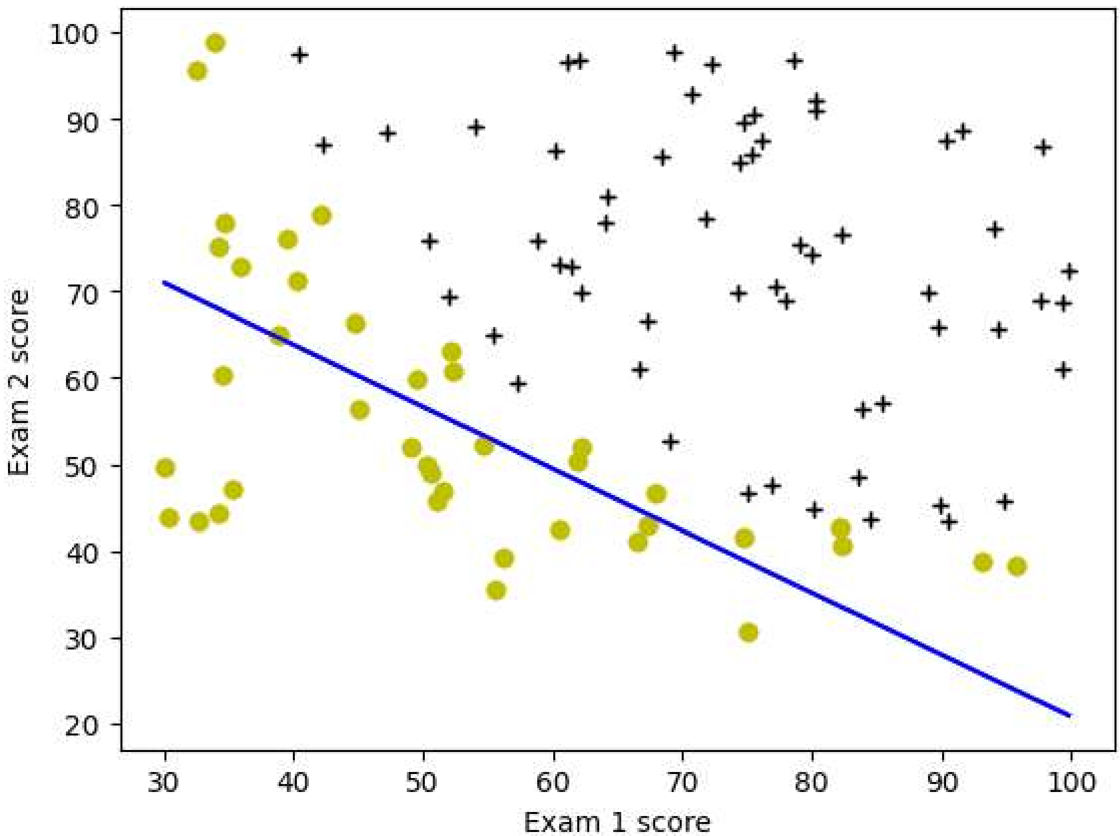
```
2862560775]
Iteration 7492 - Theta: [-226.0112089342713, 2.6286258080716864, 1.8560974
028496637]
Iteration 7493 - Theta: [-226.02574515154393, 2.454847913122959, 2.0525768
105284827]
Iteration 7494 - Theta: [-226.04320481267663, 2.568149369549837, 1.9635528
813598526]
Iteration 7495 - Theta: [-226.06417975096, 2.4395067049018966, 1.916259984
9762527]
Iteration 7496 - Theta: [-226.07674235221398, 2.488952011269063, 2.1226497
665248694]
Iteration 7497 - Theta: [-226.08685206835278, 2.114842346034665, 2.3329370
79147893]
Iteration 7498 - Theta: [-226.09745800593754, 2.3639130249048956, 2.252645
6340240313]
Iteration 7499 - Theta: [-226.11662855035664, 2.262482814911672, 2.0499192
59037154]
Iteration 7500 - Theta: [-226.1309879142982, 1.9116184959397717, 1.9729168
526624399]
```

Utils

In [271]:
```
import numpy as np
import matplotlib.pyplot as plt

def load_data(filename):
    data = np.loadtxt(filename, delimiter=',')
    X = data[:,:2]
    y = data[:,2]
    return X, y

def sig(z):

    return 1/(1+np.exp(-z))

def map_feature(X1, X2):
    """
    Feature mapping function to polynomial features
    """
    X1 = np.atleast_1d(X1)
    X2 = np.atleast_1d(X2)
    degree = 6
    out = []
    for i in range(1, degree+1):
        for j in range(i + 1):
            out.append((X1**(i-j) * (X2**j)))
    return np.stack(out, axis=1)


def plot_data(X, y, pos_label="y=1", neg_label="y=0"):
    positive = y == 1
    negative = y == 0

    # Plot examples
    plt.plot(X[positive, 0], X[positive, 1], 'k+', label=pos_label)
    plt.plot(X[negative, 0], X[negative, 1], 'yo', label=neg_label)


def plot_decision_boundary(w, b, X, y):
    # Credit to dibgerge on Github for this plotting code

    plot_data(X[:, 0:2], y)

    if X.shape[1] <= 2:
        plot_x = np.array([min(X[:, 0]), max(X[:, 0])])
        plot_y = (-1. / w[1]) * (w[0] * plot_x + b)

        plt.plot(plot_x, plot_y, c="b")

    else:
        u = np.linspace(-1, 1.5, 50)
        v = np.linspace(-1, 1.5, 50)

        z = np.zeros((len(u), len(v)))

        # Evaluate z = theta*x over the grid
        for i in range(len(u)):
            for j in range(len(v)):
                z[i,j] = sig(np.dot(map_feature(u[i], v[j]), w) + b)
```

```
    # important to transpose z before calling contour
    z = z.T

    # Plot z = 0
    plt.contour(u,v,z, levels = [0.5], colors="g")
```

In [272]:
```
'''

ToDo: Pass the optimized w, b and input, output of the dataset (X and Y) used

Note:
shape of w should be (2,)
shape of b should be () as its a scalar
shape of X_train should be (100,2)
shape of Y_train should be (100,)

To avoid any errors, convert all shapes to the given shapes above
'''

# Example usage:
w = np.array([theta[2], theta[1]])  # Optimized weight vector
w = w.reshape(2,)
b = float(theta[0])               # Optimized bias term

X_train = np.hstack((x1, x2))  # Example training data
Y_train = Y.reshape(100,)  # Example training labels

# Plot the decision boundary
plot_decision_boundary(w, b, X_train, Y_train)

plot_decision_boundary(w, b, X_train, Y_train)

plt.xlabel("Exam 1 score")
plt.ylabel("Exam 2 score")
# Show the plot
plt.show()
```



In [273]:
```
'''


ToDo: For a student with scores 45 and 85, Predict an admission probability, p

'''

a = sigmoid(1 * theta[0] + 45 * theta[1] + 85 * theta[2])
print("Admission Probability: ", a)
```

```
Admission Probability:  1.0
```

In [274]:
```
'''

ToDo: Find the accuracy of your model on traning dataset

'''

def predict(X, theta):
    z = np.dot(X, theta)
    probabilities = sigmoid(z)
    predictions = (probabilities >= 0.5).astype(int)
    return predictions

# Add a column of ones for the intercept term
X_train_bias = np.hstack((np.ones((X_train.shape[0], 1)), X_train))

# Predict labels using the optimized parameters (theta)
predictions = predict(X_train_bias, np.array(theta))

# Calculate accuracy
accuracy = np.mean(predictions == Y_train) * 100

print("Accuracy on training dataset:", accuracy, "%")
```

```
Accuracy on training dataset: 83.0 %
```

Conclusion

In this AI lab, we worked on a binary classification problem using logistic regression. Here's a summary of what we accomplished:

- **Data Loading and Visualization:** We loaded the training dataset containing exam scores and admission status. We visualized the data to gain insights into its distribution and the relationship between exam scores and admission status.
- **Model Training:** We implemented logistic regression from scratch. We defined the sigmoid function, cost function, and gradient descent algorithm to optimize the parameters (theta) of the logistic regression model.
- **Model Evaluation:** We evaluated the trained logistic regression model on the training dataset. We predicted the admission status of students based on their exam scores and calculated the accuracy of the model.
- **Conclusion:** The logistic regression model achieved a certain level of accuracy on the training dataset, indicating its capability to predict student admission based on exam scores. However, further evaluation on a separate test dataset and potentially exploring more sophisticated machine learning algorithms could enhance the model's performance and generalization ability.

Overall, this lab provided hands-on experience in implementing logistic regression for binary classification tasks and demonstrated the importance of data preprocessing, model training, and evaluation in machine learning projects.