L-Università ta' Malta

Logistic Regression

1. Use the following python code to load the training data that will be used in this experiment.

```
from sklearn.datasets import make_moons
import matplotlib.pyplot as plt
import numpy as np

X, y = make_moons(n_samples=100, noise=0.24)
```

2. Plot a scatter plot showing the different classes in different colours.
3. Write a function that computes the sigmoid function.

```
def sigmoid(z):
...
return s
```

4. Write a function loss that returns the cross entropy loss.

```
def loss(y, y_hat):
...
return l
```

5. Copy the following function that computes the gradients of both weights and biases.

```
def gradients(X, y, y_hat):

    # X --> Input.
    # y --> true/target value.
    # y_hat --> hypothesis/predictions.
    # w --> weights (parameter).
    # b --> bias (parameter).

    # m-> number of training examples.
    m = X.shape[0]

    # Gradient of loss w.r.t weights.
    dw = (1/m)*np.dot(X.T, (y_hat - y))
```

```
    # Gradient of loss w.r.t bias.
    db = (1/m)*np.sum((y_hat - y))

    return dw, db
```

6. Write a function that takes the raw data as input and returns the standardized data as output Xs.

```
def normalize(X):
    ...

    return Xs
```

7. Copy the following function that will be used to train the logistic regression using gradient descent.

```
def train(X, y, epochs, lr, bs=100):

    # X --> Input.
    # y --> true/target value.
    # bs --> Batch Size.
    # epochs --> Number of iterations.
    # lr --> Learning rate.

    # m-> number of training examples
    # n-> number of features
    m, n = X.shape

    # Initializing weights and bias to zeros.
    w = np.zeros((n,1))
    b = 0

    # Reshaping y.
    y = y.reshape(m,1)

    # Normalizing the inputs.
    x = normalize(X)

    # Empty list to store losses.
    losses = []

    # Training loop.
    for epoch in range(epochs):
        for i in range((m-1)//bs + 1):
```

```
            # Defining batches. SGD.
            start_i = i*bs
            end_i = start_i + bs
            xb = X[start_i:end_i]
            yb = y[start_i:end_i]

            # Calculating hypothesis/prediction.
            y_hat = sigmoid(np.dot(xb, w) + b)

            # Getting the gradients of loss w.r.t parameters.
            dw, db = gradients(xb, yb, y_hat)

            # Updating the parameters.
            w -= lr*dw
            b -= lr*db

        # Calculating loss and appending it in the list.
        l = loss(y, sigmoid(np.dot(X, w) + b))
        losses.append(l)

    # returning weights, bias and losses(List).
    return w, b, losses
```

8. Use the train function above to train the logistic regression. Print the derived weight and bias using 10000 epochs and a learning rate of 0.01.
9. Compute the accuracy of the model on the training data.