ECE4179 Assignment 1 Report

Full Name: Zhiyue Li ID: 28280016

Email Address: zlii0010@student.monash.edu

Q1.

No, it is not possible to train the data without changing bias weight. For each iteration, predicted y (y hat) equals sign {w. transpose * x + bias}. If predicted y does not equal to y, weight and bias are both required to update. If only weight is updated, then at next iteration, the predicted y will be affected again without changing the bias. After multiple iteration, the accuracy for model for fitting the dataset will decrease more and more. In the end, after the maximum iteration, weight will also have great difference from the real optimum weight.

Q2.

1.

```
def sigmoid(x):
    # this function should compute the sigmoid of x
    return 1/(1+np.exp(-x))
```

2. **write a function to compute the loss and gradient for the logistic model below**

```
def compute_grad_loss(X, y, theta):
    #this function will get X, a set of samples (each sample is a row in X),
    #the corresponding labels in the array y and the current parameter of the logistic model theta

    #use the sigmoid function to compute the loss and the gradient of samples with respect to theta

    #when computing the loss value, pay extra attention to the log function. log(0) can cause problems
    #to handle it

    loss = 0
    grad_vec = 0
    row,_ = X.shape
    ## Add one small value to void log(0)
    val = 1e-15
    loss =-(1/row)*np.sum(y*np.log(predict(X,theta)+val)+(1-y)*np.log(1-predict(X,theta)+val))

    grad_vec = (1/row)*np.dot(X.T,predict(X,theta)-y)

    return loss, grad_vec
```

3. **below you need to implement the gradient descent (GD) algorithm.**

```
# theta is the parameters of the logistic model

# To learn them, we randomly initilize them below
theta = np.random.randn(X_train.shape[1],1)

#this is the learning rate of the GD algorithm, you need to tune this and study its effects in your rep
lr = 0.01
print(theta)
# this is the maximum number of iterations of the GD algorithm.
# Since we use the GD, each iteration of the algorithm is equivalent to one epoch, hence the name
max_epoch = 70

loss = np.zeros(max_epoch) #keep track of the loss values for plotting
for epoch in range(max_epoch):
    # call the compute_grad_loss that you have implemented above to
    # measure the loss and the gradient
    loss[epoch], grad_vec = compute_grad_loss(X_train, y_train, theta)
    #update the theta parameter according to the GD here
    theta -= grad_vec*lr

#plt.plot(range(0,max_epoch),loss)
```

4.

```
def predict(X, theta):
    """
    this function should get X, an array of samples, and theta, the parameters
    of the logistic model and generate 0 or 1 as the label of each sample in X
    #the rule is that, if the sigmoid of x >= 0.5, we predict the label of x to be 1
    otherwise the label is 0
    """
    temp = np.dot(X,theta)
    val = np.where(sigmoid(temp)<0.5,0,1)
    return val
```

Q3.

1. Discuss the effect of the learning rate parameter.

ANS: If the learning rate is too small, the algorithm will have to go through loops more to get converged, which means it will take more time. However, if the learning rate is too large, it might jump across the global minimum and goes to another side, which might cause the algorithm to diverge.

2.

ANS: Apply the learning rate as 0.1 and a max epoch as 200, then the model has accuracy of 84.25%.

Evaluate your trained model using the code below

In [8]:
```
# now that you have trained your model, let's evaluate it

# first call teh predict function on your test data with the parameters obtained by DG
y_test_hat = predict(X_test,theta)
# make sure that the predictions are either 0 or 1 and the shape of y_test_hat
# matches that ofy_test

# the script below, if the dimensionality of the arrays is set correctly,
# will measure how many samples are correctly classified by your model
score = float(sum(y_test_hat == y_test))/ float(len(y_test))
print(score)
# this code will plot the data and see how the line fits the dataset
#plt.subplot(121)
#plt.scatter(X_test[:, 0], X_test[:, 1], marker='o', c=y_test[:,0], s=25, edgecolor='k')

#plt.subplot(122)
#plt.scatter(X_test_bias[:, 0], X_test_bias[:, 1], marker='o', c=y_test_bias[:,0], s=25, edgecolor='k')
#plt.plot(X_test,y_test_hat,label = 'Decision Boundary')
#plt.show()
```

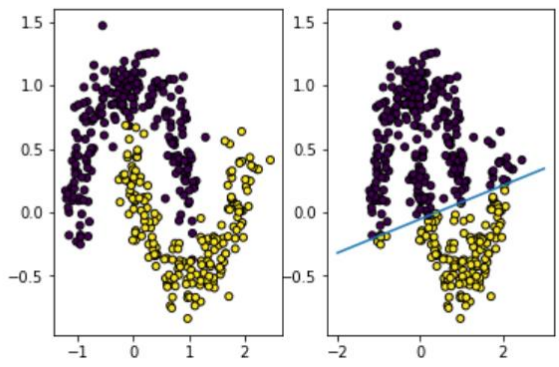
```
0.8425
```

3.

ANS: After adding the bias into the model, the accuracy has dropped to 82.55%, which is little lower than model without bias. It might be caused by the overfitting in the data set.

4.

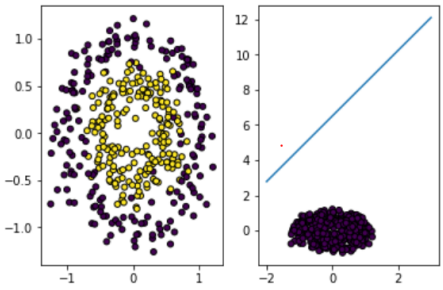
```
0.8225
```



Q4. 1)

```
plt.subplot(121)
plt.scatter(X_test[:, 0], X_test[:, 1], marker='o', c=y_test[:,0], s=25, edgecolor='k')
plt.subplot(122)
plt.scatter(X_test[:, 0], X_test[:, 1], marker='o', c=y_test_hat[:,0], s=25, edgecolor='k'
plt.plot(range(-2,4),(range(-2,4)*theta[0,0]+theta[2,0])/-theta[1,0])
plt.show()
```

```
0.47
```

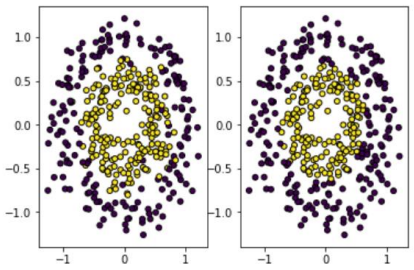


1) The model only has the accuracy of 47%

2.

```
plt.subplot(121)
plt.scatter(X_test_nonlinear[:, 0], X_test_nonlinear[:, 1], marke
plt.subplot(122)
plt.scatter(X_test_nonlinear[:, 0], X_test_nonlinear[:, 1], marke
#plt.plot(range(-2,4),(range(-2,4)*theta[0,0]+theta[2,0])/-theta[
plt.show()
```

```
0.9
```



ANS: After applying nonlinear features, the accuracy of the model has risen to 90%.