Assignment 1

Name:Navaneeth Shaji

Roll Number: 21CS30032

In [ ]:
```
# import all the necessary libraries here
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
```

In [ ]:
```
df = pd.read_excel('../../dataset/logistic-regression/Pumpkin_Seeds_Dataset.xlsx')
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

# creating a column of 1s inorder to represent the constant term in the linear cost function
column_of_ones = np.ones((X.shape[0],1))\

# normalising the data using skllearn library functions
st = StandardScaler()
X_norm = st.fit_transform(X)
# adding the column of 1s to the normalised data
X_norm = np.hstack((column_of_ones,X_norm))



# encoding categorical variables
label_encoder = LabelEncoder()

y = label_encoder.fit_transform(y)
```

In [ ]:
```
# splitting the  dataset into training , validation and testing wihtout normalising the data
X_train,X_test,y_train,y_test = train_test_split(X_norm,y,random_state=104,train_size=0.8)
X_train,X_validation,y_train,y_validation = train_test_split(X_train,y_train,random_state=104,train_size=5/8)
```

In [ ]:
```
# activation function
def sigmoid(x,theta) :
    x = np.dot(theta,x)
    g = 1/(1+np.exp(-x))
    return g
```

In [ ]:
```
# computes the value of the cost
def cost_func(x ,y,theta , sigmoid) :
    m = x.shape[0]

    cost_sum =0

    for i in range(m) :
        h = sigmoid(x[i],theta)
        cost = -(y[i]*np.log(h) + (1-y[i])*np.log(1-h))
        cost_sum = cost_sum + cost
    cost_sum = cost_sum / m

    return cost_sum
```

In [ ]:
```
# returns a numpy array having the values of the gradient of each feature
def find_gradient(x,y,theta,sigmoid)  :
    m = x.shape[0]
    n = x.shape[1]

    dl_d0 = np.zeros((n,))


    for j in range(n) :
        err_sum = 0
        for i in range(m) :
            err = sigmoid(x[i],theta) - y[i]
            err = err * x[i][j]
            err_sum = err_sum + err
        dl_d0[j] = err_sum /m

    return dl_d0
```

In [ ]:
```
# implements the gradient descent using find_gradient function and sigmoid function
def gradient_descent(x,y,alpha,iterations,find_gradient,sigmoid) :
    m = x.shape[0]
    n = x.shape[1]

    theta = np.ones(n)

    for i in range(iterations) :
        gradient = find_gradient(x,y,theta,sigmoid)
        theta = theta - alpha*gradient

    return theta
```

In [ ]:
```
# Here I am trying to do the logistic regression on 1000 iterations using three values for alpha


alpha = [0.1]
iterations_no = 1000
for i in alpha :
    theta = gradient_descent(X_train,y_train,i,iterations_no,find_gradient,sigmoid)
    print("Theta = ",theta)


    c = np.zeros((2,2))

    # making the confusion matrix

    m = X_train.shape[0]

    for i in range(m) :
        h = sigmoid(X_train[i],theta)
        if(h>=0.5 and y_train[i] == 1) :
            c[1][1] = c[1][1] + 1
        elif(h>=0.5 and y_train[i] == 0) :
            c[0][1] = c[0][1]+1
        elif (h<0.5 and y_train[i] == 1) :
            c[1][0] = c[1][0] + 1
        elif(h<0.5 and y_train[i] == 0) :
            c[0][0] = c[0][0] + 1
    print("Confusion matrix = ",c)

    #calculating mean


    # calculating accuracy
    acc = (c[0][0] + c[1][1]) /(c[0][0] + c[0][1] + c[1][0] + c[1][1])
    print("Accuracy = ",acc)

    # calculating precision
    pre = c[1][1] / (c[1][1] + c[0][1])
    print("Precision = " ,pre)

    #calculating recall
    rec = c[1][1] / (c[1][1] + c[1][0])
    print("Recall = " , rec)

    print("\n\n\n")
```

```
Theta =  [-0.0970886   0.03835434  0.21907652  0.56310666 -0.41463887  0.03173037
 -0.31828903  1.17413275  0.65368725  0.08231882  0.03016655  1.71005542
  0.40895983]
Confusion matrix =  [[589.  66.]
 [ 96. 499.]]
Accuracy =  0.8704
Precision =  0.8831858407079646
```

```
Recall =  0.838655462184874
```