นาย สัณหณัฐ พรมจรรย์ 63070501069

Quiz 6

1. *2.5 hours.* Modify "Logistic Regression Gradient Descent.ipynb" for Iris classification to use the cost function and Jacobian function and the "BFGS" optimization method in SciPy library to:

Link: https://colab.research.google.com/drive/1vkzSU-vKKglcro2_-pwxohRFZ8QD2uIS?usp=sharing

```
[50]  import numpy as np
      import matplotlib as plt
      import math
      from scipy.optimize import minimize
```

```
[51]  from sklearn.datasets import load_iris
      from sklearn.model_selection import train_test_split
      X, y = load_iris(return_X_y=True)
      print ("5 samples Iris data X = \n", X[:5][:])
      print ("All Iris data y = \n", y)
      #Use sklearn's library to split the data into training and testing sets with ratio 75% to 25%.
      X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.25, random_state=0)
      print("First 5 samples of X_train: \n", X_train[0:5, :])
      print("First 5 samples of y_train: \n", y_train[0:5])
```

```
5 samples Iris data X =
 [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
All Iris data y =
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
First 5 samples of X_train:
 [[5.9 3.  4.2 1.5]
 [5.8 2.6 4.  1.2]
 [6.8 3.  5.5 2.1]
 [4.7 3.2 1.3 0.2]
 [6.9 3.1 5.1 2.3]]
First 5 samples of y_train:
 [1 1 2 0 2]
```

[52]
```
#Explain if want to normalize data
a = np.array([[1, 2], [3, 4]])
print("matrix a: \n", a)
print("Mean a: \n", np.mean(a))
print("matrix per column a: \n",np.mean(a, axis=0))
print("matrix per row a: \n",np.mean(a, axis=1))
```

```
matrix a:
 [[1 2]
 [3 4]]
Mean a:
 2.5
matrix per column a:
 [2. 3.]
matrix per row a:
 [1.5 3.5]
```

[53]
```
#do this part only if you want to normalize the data
#YOU NEED NOT NORMALIZE FOR IT TO WORK
X_av = np.mean(X_train, axis=0)
X_sd = np.std(X_train, axis=0)
X_train = (X_train - X_av)/X_sd
X_test = (X_test - X_av)/X_sd
```

[54]
```
#input is a vector and returns a vector
def stable_sigmoid(x):
    sig = np.where(x < 0, np.exp(x)/(1 + np.exp(x)), 1/(1 + np.exp(-x)))
    return sig
```

[55]
```
def h(theta, X): #theta is a n x 1 vector
  #X is a mxn matrix
  #  [x0 x1 ... x_n-1]_1 where n-1 is size of feature vector (no. features)
  #  [x0 x1 ... x_n-1]_2 REMEMBER x0 is 1.
  #  ...
  #  [x0 x1 ... x_n-1]_m where m is number of data points
  #for matrix X_mxn, and vector theta_nx1 this will return a vector mx1
  #print("shape X, theta = ", X.shape, theta.shape)
  z = X @ theta # X is m x n, theta is n x 1. Result z is mx1.
  return stable_sigmoid(z) #returns vector mx1
```

[56]
```
def J(theta, X, y):
  #X[i] is 1 data point
  #y = X @ theta where X is m x n and theta is n x 1. y is m x 1.
  #y[i] is 0 or 1 only for data belongs to this class or not this class
  #m is number of data points.  n is number of features
  #first column of X contain all 1's
  #Assume we have J = (-1/m)SUM_{i=1..m}[y_i * log(h(theta, X[i])) +
  #                                      (1-y_i)*log(1-h(theta, X[i])) ]
  m = X.shape[0] #m - number of data points
  n = X.shape[1] #n - no. features + 1, because first column X is all 1 for intercepts.
  H = h(theta, X)
  #print ("In function E, m, n = ", m, n, "y.shape = ", y.shape, "h shape = ", T1.shape)
  cost = -(y.T @ np.log(H) + (1-y).T @ np.log(1-H))/m
  return cost #returns a scalar
```

[57]
```
def dJ_d_theta(theta, X, y):
    m = X.shape[0] #m - number of data points
    n = X.shape[1] #n - no. features + 1, because first column X is all 1 for intercepts.
    dJ_dt = X.T @ (h(theta, X) - y)/m # this is n x m * m x 1 to get n x 1
    return dJ_dt #an array n x 1
```

[58]
```
m = X_train.shape[0] #m - number of data points
n = X_train.shape[1] + 1 #n - number of features + 1; 1 is for column of 1's.
#start with random values of solution theta, array values betweeen -10 to 10:
theta0 = 20* np.random.rand(n) - 10 #theta0 is 1 x n array of random values
theta = theta0.T #convert to vector (n x 1)
```

[59]
```
#set yN to 0 if y is not equal to N, 1 if y is equanl to N.
y0_train = (y_train==0)*1.0 #yes/no of class 0
y1_train = (y_train==1)*1.0 #yes/no of class 1
y2_train = (y_train==2)*1.0 #yes/no of class 2
y0_test = (y_test ==0)*1.0
y1_test = (y_test ==1)*1.0
y2_test = (y_test ==2)*1.0
print("y_test = \n",y_test)
print("y1_test = \n", y1_test) #note that in y0_test only y with class 0 are set to 1, others to 0.\
```

```
y_test =
 [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
 1]
y1_test =
 [0. 1. 0. 0. 0. 0. 0. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 0. 0. 0. 1. 0. 0.
 0. 0. 0. 1. 1. 0. 0. 1. 0. 0. 0. 1. 0. 1.]
```

[65]
```
X_1_train = np.ones([m,n]) #create an array of all 1's
X_1_train[:, 1:] = X_train #leave column 1 as 1's, and replace all other data with X_train

X_1_test = np.ones([X_test.shape[0],X_test.shape[1] + 1]) #create an array of all 1's
X_1_test[:, 1:] = X_test #leave column 1 as 1's, and replace all other data with X_test
```

[66]
```
res0 = minimize(J, theta, args = (X_1_train, y0_train), method='BFGS', jac=dJ_d_theta, options={'disp': True})
theta_solved0 = res0.x
predict_y0_before =  h(theta_solved0, X_1_test) #values of probability
predict_y0 = (predict_y0_before > 0.5)*1.0 #threshold to 1 or 0
np.set_printoptions(precision=5,suppress=True) #print with 2 decimal places and suppress scientific notation
(predict_y0 == y0_test).sum()
correct0 = (((predict_y0 == y0_test).sum())/len(y0_test))*100
```

```
Optimization terminated successfully.
         Current function value: 0.000007
         Iterations: 29
         Function evaluations: 32
         Gradient evaluations: 32
```

[67]
```
res1 = minimize(J, theta, args = (X_1_train, y1_train), method='BFGS', jac=dJ_d_theta, options={'disp': True})
theta_solved1 = res1.x
predict_y1_before =  h(theta_solved1, X_1_test) #probability 0 - 1
predict_y1 = (predict_y1_before > 0.5)*1.0 #binary 0, 1
np.set_printoptions(precision=2,suppress=True) #print with 2 decimal places and suppress scientific notation
correct1 = (((predict_y1 == y1_test).sum())/len(y1_test))*100
```

```
Optimization terminated successfully.
         Current function value: 0.476675
         Iterations: 33
         Function evaluations: 37
         Gradient evaluations: 37
```

[68]
```
res2 = minimize(J, theta, args = (X_1_train, y2_train), method='BFGS', jac=dJ_d_theta, options={'disp': True})
theta_solved2 = res2.x
predict_y2_before =  h(theta_solved2, X_1_test)
predict_y2 = (predict_y2_before > 0.5)*1.0
np.set_printoptions(precision=1,suppress=True) #print with 2 decimal places and suppress scientific notation
correct2= (((predict_y2 == y2_test).sum())/len(y2_test))*100
```

```
Warning: Desired error not necessarily achieved due to precision loss.
         Current function value: nan
         Iterations: 31
         Function evaluations: 48
         Gradient evaluations: 48
<ipython-input-56-c0de80af3a5b>:13: RuntimeWarning: divide by zero encountered in log
  cost = -(y.T @ np.log(H) + (1-y).T @ np.log(1-H))/m
<ipython-input-56-c0de80af3a5b>:13: RuntimeWarning: invalid value encountered in matmul
  cost = -(y.T @ np.log(H) + (1-y).T @ np.log(1-H))/m
<ipython-input-56-c0de80af3a5b>:13: RuntimeWarning: divide by zero encountered in log
  cost = -(y.T @ np.log(H) + (1-y).T @ np.log(1-H))/m
<ipython-input-56-c0de80af3a5b>:13: RuntimeWarning: invalid value encountered in matmul
  cost = -(y.T @ np.log(H) + (1-y).T @ np.log(1-H))/m
```

a. 20 points. Report the total classification accuracy (score) for the Test data by finding the highest probability class as the output.

```
[69] max_class = []
     for i in range(predict_y0_before.shape[0]):
       max_val = max(predict_y0_before[i], predict_y1_before[i], predict_y2_before[i])
       if max_val == predict_y0_before[i]:
         max_class.append(0)
       elif max_val == predict_y1_before[i]:
         max_class.append(1)
       elif max_val == predict_y2_before[i]:
         max_class.append(2)
     max_class = np.array(max_class)
     print("Class with max probability: \n", max_class)
     print("Y test: \n", y_test)
     is_correct_match = (max_class == y_test)
     print("Is Correct Match: \n", is_correct_match)
     correct = ((is_correct_match.sum())/len(y_test))*100
     print("Correct percent for logistic class prediction: %5.1f%%"%correct)
```

```
Class with max probability:
 [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
 2]
Y test:
 [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
 1]
Is Correct Match:
 [ True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True False]
Correct percent for logistic class prediction:  97.4%
```

b. 5 points. Print the confusion matrix for the test data of the 3 classes found by your own implementation of the logistic regressor. You can use Sklearn's library; here's an example of how:

```
>>> from sklearn.metrics import confusion_matrix
>>> y_true = [2, 0, 2, 2, 0, 1]
>>> y_pred = [0, 0, 2, 2, 0, 2]
>>> confusion_matrix(y_true, y_pred)
array([[2, 0, 0],
       [0, 0, 1],
       [1, 0, 2]])
```

```
[70] from sklearn.metrics import confusion_matrix
     tn0, fp0, fn0, tp0 = confusion_matrix(y0_test,predict_y0).ravel()
     print('Confusion Matrix:')
     print('True Negative:', tn0)
     print('False Positive:', fp0)
     print('False Negative:', fn0)
     print('True Positive:', tp0)
```

```
Confusion Matrix:
True Negative: 25
False Positive: 0
False Negative: 0
True Positive: 13
```

```
[71] tn1, fp1, fn1, tp1 = confusion_matrix(y1_test,predict_y1).ravel()
     print('Confusion Matrix:')
     print('True Negative:', tn1)
     print('False Positive:', fp1)
     print('False Negative:', fn1)
     print('True Positive:', tp1)
```

```
Confusion Matrix:
True Negative: 20
False Positive: 2
False Negative: 10
True Positive: 6
```

```
[72] tn2, fp2, fn2, tp2 = confusion_matrix(y2_test,predict_y2).ravel()
     print('Confusion Matrix:')
     print('True Negative:', tn2)
     print('False Positive:', fp2)
     print('False Negative:', fn2)
     print('True Positive:', tp2)
```

```
Confusion Matrix:
True Negative: 28
False Positive: 1
False Negative: 0
True Positive: 9
```

c. 5 points. Print the confusion matrix for the test data found by sklearn's logistic regressor.

```
[73] from sklearn.linear_model import LogisticRegression
     logistic_regressor = LogisticRegression(max_iter = 500)
     # The default constructor is:
     # LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
     #       intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
     #       penalty='l2', random_state=None, solver='lbfgs', tol=0.0001,
     #       verbose=0, warm_start=False)
```

```
[74] #X is n rows of data points by 4 columns of features
     fit_class = logistic_regressor.fit(X_train, y_train)
     y_predict = fit_class.predict(X_test)
     print("Predicted output of all test data: \n", y_predict)
     print("Target output of all test data: \n", y_test)
     print("Correctly matched index of test class and predicted class: \n", y_test == y_predict)
     fit_score = 100*fit_class.score(X_test, y_test)
     print("Fitting score Percent: %5.1f%%" %fit_score)
```

```
Predicted output of all test data:
 [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
 2]
Target output of all test data:
 [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
 1]
Correctly matched index of test class and predicted class:
 [ True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True False]
Fitting score Percent:  97.4%
```

```
[75] confusion_matrix(y_test,y_predict)
```

```
array([[13,  0,  0],
       [ 0, 15,  1],
       [ 0,  0,  9]])
```

d. 10 points. For each mis-classified data sample, show the classification probability for class 0, 1, and 2 (we want to see that the probabilities are hovering around the 0.2 to 0.9 range for these data points).

```
[76] #Let's see what the prediction probability was for each class (0, 1, or 2) for the test cases
     y_test_prob = fit_class.predict_proba(X_test)
     print("Prediction probability for test data P[0], P[1], P[2]: \n", y_test_prob)
```

```
Prediction probability for test data P[0], P[1], P[2]:
 [[0.  0.  1. ]
 [0.  1.  0. ]
 [1.  0.  0. ]
 [0.  0.1 0.9]
 [1.  0.  0. ]
 [0.  0.  1. ]
 [1.  0.  0. ]
 [0.  0.7 0.3]
 [0.  0.7 0.3]
 [0.  0.9 0.1]
 [0.  0.4 0.6]
 [0.  0.8 0.2]
 [0.  0.9 0.1]
 [0.  0.7 0.3]
 [0.  0.7 0.2]
 [1.  0.  0. ]
 [0.  0.7 0.3]
 [0.  0.9 0.1]
 [0.9 0.1 0. ]
 [1.  0.  0. ]
 [0.  0.2 0.8]
 [0.  0.7 0.2]
 [1.  0.  0. ]
 [1.  0.  0. ]
 [0.  0.3 0.7]
 [1.  0.  0. ]
 [1.  0.  0. ]
 [0.  0.9 0.1]
 [0.1 0.9 0. ]
 [1.  0.  0. ]
 [0.  0.2 0.8]
 [0.1 0.7 0.2]
 [1.  0.  0. ]
 [0.  0.3 0.6]
 [0.  0.  1. ]
 [0.1 0.8 0.1]
 [1.  0.  0. ]
 [0.  0.4 0.6]]
```

2. *30 points*. 3 hours. Using the Kaggle open source dataset to predict Attrition (Will an employee continue to stay with our company?) based on several factors using Logistic Regression. Make sure you show the **confusion matrix.** In the case of a Yes/No classification like this problem it shows false positives and false negatives. The data is available in:

https://www.kaggle.com/pavansubhasht/ibm-hr-analytics-attrition-dataset

Link: https://colab.research.google.com/drive/1Pfn1nHM4k8db7WZ3vplLvQM2P5Kn6ela?usp=sharing

```
[55]  # Import the libraries and load the dataset
      import pandas as pd
      import numpy as np
      import matplotlib.pyplot as plt
      from sklearn import metrics
      from sklearn.model_selection import train_test_split
      from sklearn.linear_model import LogisticRegression
      from sklearn.metrics import confusion_matrix

      df = pd.read_csv("WA_Fn-UseC_-HR-Employee-Attrition.csv")
      print(df.head())

      # Converting the categorical variables to numerical using one-hot encoding
      categorical_cols = ['BusinessTravel', 'Department', 'EducationField', 'Gender', 'JobRole', 'MaritalStatus', 'Over18', 'OverTime']
      df = pd.get_dummies(df, columns=categorical_cols)
```

```
[56]  X = df.drop('Attrition', axis=1)
      y = df['Attrition']

      # Split the data into training and testing sets
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
   Age Attrition     BusinessTravel  DailyRate              Department  \
0   41       Yes      Travel_Rarely       1102                   Sales
1   49        No  Travel_Frequently        279  Research & Development
2   37       Yes      Travel_Rarely       1373  Research & Development
3   33        No  Travel_Frequently       1392  Research & Development
4   27        No      Travel_Rarely        591  Research & Development

   DistanceFromHome  Education EducationField  EmployeeCount  EmployeeNumber  \
0                 1          2  Life Sciences              1               1
1                 8          1  Life Sciences              1               2
2                 2          2          Other              1               4
3                 3          4  Life Sciences              1               5
4                 2          1        Medical              1               7

   ...  RelationshipSatisfaction StandardHours  StockOptionLevel  \
0  ...                         1            80                 0
1  ...                         4            80                 1
2  ...                         2            80                 0
3  ...                         3            80                 0
4  ...                         4            80                 1

   TotalWorkingYears  TrainingTimesLastYear WorkLifeBalance  YearsAtCompany  \
0                  8                      0               1               6
1                 10                      3               3              10
2                  7                      3               3               0
3                  8                      3               3               8
4                  6                      3               3               2

  YearsInCurrentRole  YearsSinceLastPromotion  YearsWithCurrManager
0                  4                        0                     5
1                  7                        1                     7
2                  0                        0                     0
3                  7                        3                     0
4                  2                        2                     2

[5 rows x 35 columns]
```

[57]
```
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

[58]
```
# Confusion matrix
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
print('Confusion Matrix:')
print('True Negative:', tn)
print('False Positive:', fp)
print('False Negative:', fn)
print('True Positive:', tp)

# Confusion matrix
cm = metrics.confusion_matrix(y_test, y_pred)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = cm, display_labels = [False, True])
cm_display.plot(cmap=plt.cm.Blues)
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.show()
```

```
Confusion Matrix:
True Negative: 255
False Positive: 0
False Negative: 39
True Positive: 0
```