Sydney Trout
Dr Li
CPSC 4430
14 March 2022

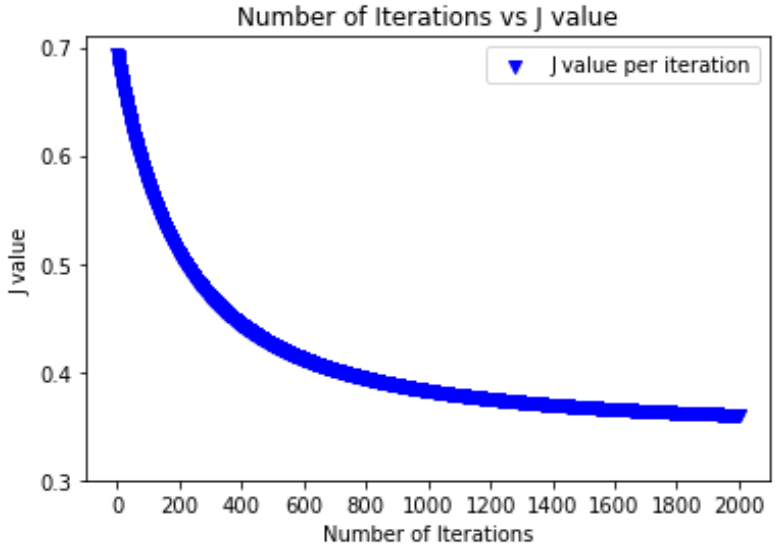
Logistic Regression Report

Model Description: This model uses logistic regression as a binary classifier. The two capacitor tests were changed into eight new features which allowed for the weights to be calculated using gradient descent on the training data, which ultimately provided an accuracy value of over 70% for the testing data.

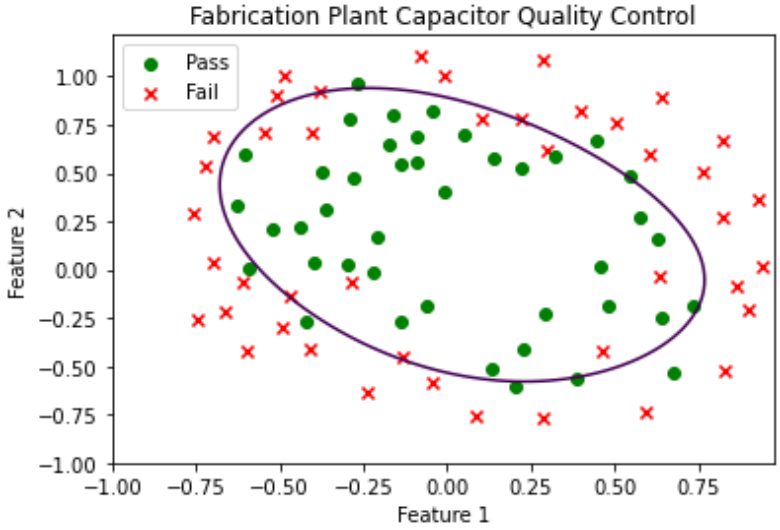
Initial Values:
Initial Weight 0 = 0.0
Initial Weight 1 = 0.0
Initial Weight 2 = 0.0
Initial Weight 3 = 0.0
Initial Weight 4 = 0.0
Initial Weight 5 = 0.0
Initial Weight 6 = 0.0
Initial Weight 7 = 0.0
Initial Weight 8 = 0.0
Learning Rate = 0.5
Initial J value = 0.6931471805599454

Final Values:
Learning Rate = 0.5
Iterations = 2000
Weight 0 = 3.6726850369579522
Weight 1 = 1.771454268667965
Weight 2 = -8.56886563881731
Weight 3 = 2.7583249314896743
Weight 4 = -5.523244106435219
Weight 5 = 0.6598208686128542
Weight 6 = -7.721164742349858
Weight 7 = -1.454726416348931
Weight 8 = -2.300918758798264
Final J value on training set: 0.36012776738506325

Plot of J (vertical axis) vs. number of iterations (horizontal axis):



Plot of hyperplane on the whole dataset:



Final J value on testing set: 0.3882163166462624

CONFUSION MATRIX ON TEST SET		Predicted Class	
		Negative	Positive
True Class	Negative	TN = 13	FP = 4
	Positive	FN = 1	TP = 15

The accuracy of the model on the testing data was 84.84%. The precision was 78.95%, the recall was 93.75%, and the F1 score was 85.71%.

CODE:

```
import numpy as np
import matplotlib.pyplot as plt
from math import exp
from numpy import arange
from numpy import meshgrid

#function that performs gradient descent to find the final weights
def logreg(xBin1, xBin2, xBin3, xBin4, xBin5, xBin6, xBin7, xBin8, yBin):
    #initial weights, learning rate, and number of iterations
    w0 = 0.0
    w1 = 0.0
    w2 = 0.0
    w3 = 0.0
    w4 = 0.0
    w5 = 0.0
    w6 = 0.0
    w7 = 0.0
    w8 = 0.0
    a = 0.5
    iterations = 2000
    J = np.zeros((iterations,1))
    for i in range(iterations):
        #calculates the J value and plots it
        sumJ = np.zeros((len(xBin1), 1))
        for j in range(len(xBin1)):
            sumh = (w0 + w1*xBin1[j,0] + w2*xBin2[j,0] + w3*xBin3[j,0] + w4*xBin4[j,0] +
w5*xBin5[j,0] + w6*xBin6[j,0] + w7*xBin7[j,0] + w8*xBin8[j,0])
            h = 1 / (1+exp(-sumh))
            sumJ[j,0] = (yBin[j,0]*np.log(h)) + ((1-yBin[j,0]) * np.log(1-h))
        J[i,0] = -(1/len(xBin1)) * np.sum(sumJ)
        plt.scatter(i,J[i,0], color = "blue", marker = "v", label = 'J value per iteration')
        sumw = np.zeros((len(xBin1), 1))
        sumw1 = np.zeros((len(xBin1), 1))
        sumw2 = np.zeros((len(xBin1), 1))
        sumw3 = np.zeros((len(xBin1), 1))
        sumw4 = np.zeros((len(xBin1), 1))
        sumw5 = np.zeros((len(xBin1), 1))
        sumw6 = np.zeros((len(xBin1), 1))
        sumw7 = np.zeros((len(xBin1), 1))
```

```
        sumw8 = np.zeros((len(xBin1), 1))
        for j in range(len(xBin1)):
            sumh = (w0 + w1*xBin1[j,0] + w2*xBin2[j,0] + w3*xBin3[j,0] + w4*xBin4[j,0] +
w5*xBin5[j,0] + w6*xBin6[j,0] + w7*xBin7[j,0] + w8*xBin8[j,0])
            h = 1 / (1+exp(-sumh))
            sumw[j,0] = (h - yBin[j,0])
            sumw1[j,0] = (h - yBin[j,0]) * xBin1[j,0]
            sumw2[j,0] = (h - yBin[j,0]) * xBin2[j,0]
            sumw3[j,0] = (h - yBin[j,0]) * xBin3[j,0]
            sumw4[j,0] = (h - yBin[j,0]) * xBin4[j,0]
            sumw5[j,0] = (h - yBin[j,0]) * xBin5[j,0]
            sumw6[j,0] = (h - yBin[j,0]) * xBin6[j,0]
            sumw7[j,0] = (h - yBin[j,0]) * xBin7[j,0]
            sumw8[j,0] = (h - yBin[j,0]) * xBin8[j,0]
        #calculates the new weights
        w0 = w0 - a * (1/len(xBin1)) * np.sum(sumw)
        w1 = w1 - a * (1/len(xBin1)) * np.sum(sumw1)
        w2 = w2 - a * (1/len(xBin1)) * np.sum(sumw2)
        w3 = w3 - a * (1/len(xBin1)) * np.sum(sumw3)
        w4 = w4 - a * (1/len(xBin1)) * np.sum(sumw4)
        w5 = w5 - a * (1/len(xBin1)) * np.sum(sumw5)
        w6 = w6 - a * (1/len(xBin1)) * np.sum(sumw6)
        w7 = w7 - a * (1/len(xBin1)) * np.sum(sumw7)
        w8 = w8 - a * (1/len(xBin1)) * np.sum(sumw8)
    #plots the J valuess vs the number of iterations and returns the final weights
    plt.xlabel("Number of Iterations")
    plt.ylabel("J value")
    plt.title("Number of Iterations vs J value")
    plt.legend(["J value per iteration"])
    x = [0, 200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]
    plt.xticks(x)
    y = [0.7, 0.6, 0.5, 0.4, 0.3]
    plt.yticks(y)
    plt.show()
    return w0, w1, w2, w3, w4, w5, w6, w7, w8

#retrieves the training file name and opens it
fileName = input("Enter the name of your training file: ")
dataFile = open(fileName, "r")
```

```
#reads in the number of rows and features
readData = dataFile.readline()
fileInfo = readData.split("\t")
rows = int(fileInfo[0])
features = int(fileInfo[1])

xBinTrain = np.zeros((rows, features))
xBinTrain1 = np.zeros((rows,1))
xBinTrain2 = np.zeros((rows,1))
yBinTrain = np.zeros((rows,1))

#reads the training data into bins
for i in range(rows):
    readData = dataFile.readline()
    data = readData.split("\t")
    xBinTrain[i,0] = float(data[0])
    xBinTrain[i,1] = float(data[1])
    xBinTrain1[i,0] = float(data[0])
    xBinTrain2[i,0] = float(data[1])
    yBinTrain[i,0] = float(data[2])

#creating new features
xBin1 = xBinTrain1
xBin2 = xBinTrain1 ** 2
xBin3 = xBinTrain2
xBin4 = xBinTrain1 * xBinTrain2
xBin5 = xBinTrain1 * (xBinTrain2 ** 2)
xBin6 = xBinTrain2 ** 2
xBin7 = (xBinTrain1 ** 2) * xBinTrain2
xBin8 = (xBinTrain1 ** 2) * (xBinTrain2 ** 2)

#writes new training features to text file
fout1 = open("trout_sydney_P3Train.txt", 'w')
fout1.write(str(rows)+"\t"+str(features**(2+1))+"\n")
for i in range(rows):
    fout1.write(str(xBin1[i,0])+"\t"+
str(xBin2[i,0])+"\t"+str(xBin3[i,0])+"\t"+str(xBin4[i,0])+"\t"+str(xBin5[i,0])+"\t"+str(xBin6[i,0]
)+"\t"+str(xBin7[i,0])+"\t"+str(xBin8[i,0])+"\t")
    fout1.write(str(yBinTrain[i,0])+"\n")
```

```
#retrieving the final weights using gradient descent
w0, w1, w2, w3, w4, w5, w6, w7, w8 = logreg(xBin1, xBin2, xBin3, xBin4, xBin5, xBin6,
xBin7, xBin8, yBinTrain)

#calculates the final J value of the testing data
sumJ = np.zeros((len(xBin1), 1))
for j in range(len(xBin1)):
   sumh = (w0 + w1*xBin1[j,0] + w2*xBin2[j,0] + w3*xBin3[j,0] + w4*xBin4[j,0] +
w5*xBin5[j,0] + w6*xBin6[j,0] + w7*xBin7[j,0] + w8*xBin8[j,0])
   h = 1 / (1+exp(-sumh))
   sumJ[j,0] = (yBinTrain[j,0]*np.log(h)) + ((1-yBinTrain[j,0]) * np.log(1-h))
J = -(1/len(xBin1)) * np.sum(sumJ)

#plots the decision boundary
delta = 0.025
x1 = arange(-1.0, 1.0, delta)
x2= arange(-1.0, 1.0, delta)
X,Y = meshgrid(x1,x2)
F = 0
G = (w0 + w1*X + w2*(X**2) + w3*Y + w4*X*Y + w5*(X**2)*Y + w6*(Y**2) +
w7*X*(Y**2) + w8*(X**2)*(Y**2))
plt.contour(X, Y, (F - G), [0])
for i in range(len(xBin1)):
   if(yBinTrain[i,0] == 1):
      s1 = plt.scatter(xBinTrain[i,0],xBinTrain[i,1], color = "green", marker = "o", label = "Pass")
   else:
      s2 = plt.scatter(xBinTrain[i,0],xBinTrain[i,1], color = "red", marker = "x", label = "Fail")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("Fabrication Plant Capacitor Quality Control")
plt.legend((s1,s2), ("Pass", "Fail"))
plt.show()

#retrieves the testing file name and opens it
fileNameT = input("Enter the name of your testing file: ")
dataFileT = open(fileNameT, "r")

#reads in the number of rows and features
readDataT = dataFileT.readline()
```

```
fileInfoT = readDataT.split("\t")
rowsT = int(fileInfoT[0])
featuresT = int(fileInfoT[1])


xBinTest = np.zeros((rowsT, featuresT))
xBinTest1 = np.zeros((rowsT,1))
xBinTest2 = np.zeros((rowsT,1))
yBinTest = np.zeros((rowsT,1))

for i in range(rowsT):
    readDataT = dataFileT.readline()
    dataT = readDataT.split("\t")
    xBinTest[i,0] = float(dataT[0])
    xBinTest[i,1] = float(dataT[1])
    xBinTest1[i,0] = float(dataT[0])
    xBinTest2[i,0] = float(dataT[1])
    yBinTest[i,0] = float(dataT[2])

#creates new features for the testing data
xBin1T = xBinTest1
xBin2T = xBinTest1 ** 2
xBin3T = xBinTest2
xBin4T = xBinTest1 * xBinTest2
xBin5T = xBinTest1 * (xBinTest2 ** 2)
xBin6T = xBinTest2 ** 2
xBin7T = (xBinTest1 ** 2) * xBinTest2
xBin8T = (xBinTest1 ** 2) * (xBinTest2 ** 2)

#writes new testing features to text file
fout2 = open("trout_sydney_P3Test.txt", 'w')
fout2.write(str(rowsT)+"\t"+str(featuresT**(2+1))+"\n")
for i in range(rowsT):
    fout2.write(str(xBin1T[i,0])+"\t"+
str(xBin2T[i,0])+"\t"+str(xBin3T[i,0])+"\t"+str(xBin4T[i,0])+"\t"+str(xBin5T[i,0])+"\t"+str(xBin
6T[i,0])+"\t"+str(xBin7T[i,0])+"\t"+str(xBin8T[i,0])+"\t")
    fout2.write(str(yBinTest[i,0])+"\n")

TP = 0
TN = 0
```

```
FP = 0
FN = 0

#calculates the final J value for the training and the number of TP, TN, FP, FN
sumJ = np.zeros((len(xBin1T), 1))
for j in range(len(xBin1T)):
   sumh = (w0 + w1*xBin1T[j,0] + w2*xBin2T[j,0] + w3*xBin3T[j,0] + w4*xBin4T[j,0] +
w5*xBin5T[j,0] + w6*xBin6T[j,0] + w7*xBin7T[j,0] + w8*xBin8T[j,0])
   h = 1 / (1+exp(-sumh))
   if(h >= 0.5 and yBinTest[j,0] == 1):
      TP = TP + 1
   elif(h >= 0.5 and yBinTest[j,0] == 0):
      FP = FP + 1
   elif(h < 0.5 and yBinTest[j,0] == 0):
      TN = TN + 1
   else:
      FN = FN + 1
   sumJ[j,0] = (yBinTest[j,0]*np.log(h)) + ((1-yBinTest[j,0]) * np.log(1-h))
JT = -(1/len(xBin1T)) * np.sum(sumJ)

#calculates the accuracy, precision, recall, and f1 score
accuracy = (TP + TN) / (TP + TN + FP + FN)
precision = TP / (TP + FP)
recall = TP / (TP + FN)
f1 = 2 * (1 / ((1/precision) + (1/recall)))

#prints all of the results
print("Final J value on training set:", J)
print("Final J value on testing set:", JT)
print("True positives:", TP)
print("True negatives:", TN)
print("False positives:", FP)
print("False negatives:", FN)
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1:", f1)
```