HW3_TiagoGoncalves

November 18, 2018

1. Load the height/weight data from the file heightWeightData.txt. The first column is the class label (1=male, 2=female), the second column is height, the third weight. Start by replacing the weight column by the product of height and weight.

For the Fisher's linear discriminant analysis as discussed in the class, send the python/matlab code and answers for the following questions:

a. What's the SB matrix?

b. What's the SW matrix?

c. What's the optimal 1d projection direction?

d. Project the data in the optimal 1d projection direction. Set the decision threshold as the middle point between the projected means. What's the misclassification error rate?

e. What's your height and weight? What's the model prediction for your case (male/female)?

In [93]:
```
#Imports
import numpy as np
import matplotlib.pyplot as plt
np.seterr(divide='ignore', invalid='ignore')

#Load Data
data = np.genfromtxt("heightWeightData.txt", delimiter=",")

#Weight is 3rd Column
np.set_printoptions(suppress=True)
new_data = np.zeros(data.shape)
for i in range(int(new_data.shape[0])):
    new_data[i, 0] = data[i, 0]
    new_data[i, 1] = data[i, 1]
    new_data[i, 2] = np.multiply(data[i, 1], data[i, 2])
```

In [94]:
```
#Implementing Fisher's Linear Discriminant Analysis
#Let's group data first
#Count males (=1) and females (=2)
nr_males = 0
nr_females = 0
```

```
for i in range(int(new_data.shape[0])):
    if new_data[i, 0] == 1:
        nr_males+=1
    elif new_data[i, 0] == 2:
        nr_females+=1
#print(nr_males, nr_females)
#Concatenate Class Sizes
class_sizes = np.array([nr_males, nr_females])

#Assign Classes
males = np.zeros([nr_males, new_data.shape[1]])
females = np.zeros([nr_females, new_data.shape[1]])
m_index = 0
f_index = 0
for index in range(int(new_data.shape[0])):
    if new_data[index, 0] == 1:
        males[m_index] = new_data[index]
        m_index+=1
    elif new_data[index, 0] == 2:
        females[f_index] = new_data[index]
        f_index+=1

#Calculate means vector for each class
#Drop Label Column
f_males = males[:, 1:]
f_females = females[:, 1:]
#Calculate mean vector for each class
mean_males = np.mean(a=f_males, axis=0)
mean_females = np.mean(a=f_females, axis=0)

print("Mean Vector for Males Class: \n", mean_males,"\nMean Vector for Females Class:
```

```
Mean Vector for Males Class:
 [  182.01013699 14552.85501781]
Mean Vector for Females Class:
 [ 165.28540146 9757.31728073]
```

a. What's the SB matrix?

In [95]:
```
#Calculate Overall Mean
overall_mean = np.mean(new_data[:, 1:], axis=0)
#print("Overall mean vector is: ", overall_mean)

#Let's Compute Between Class Scatter Matrix S_B
"According to the slides: S_B = (m2-m1)(m2-m1).T"
S_B = np.multiply((mean_females-mean_males), (mean_females-mean_males).T)
print("S_B Matrix is: ", S_B)
```

```
S_B Matrix is:  [     279.71677843 22997182.18774202]
```

b. What's the SW matrix?

```
In [96]: #Let's Compute Within Class Scatter Matrix S_W
         #According to Slides
         #Males Class
         scatter_male = sum(np.matmul((f_males-mean_males).T, ((f_males-mean_males).T).T))
         scatter_female = sum(np.matmul((f_females-mean_females).T, ((f_females-mean_females).T
         S_W = scatter_male+scatter_female
         print("S_W Matrix is: ", S_W)
```

```
S_W Matrix is:  [2.39983269e+06 1.07899323e+09]
```

c. What's the optimal 1d projection direction?

```
In [97]: #Optimal Projection or Matrix W
         W = (1/S_W)*(mean_females-mean_males)
         print("Optimal 1D Projection Direction is: ", W)
```

```
Optimal 1D Projection Direction is:  [-0.00000697 -0.00000444]
```

d. Project the data in the optimal 1d projection direction. Set the decision threshold as the middle point between the projected means. What's the misclassification error rate?

```
In [98]: #Calculate Threshold
         tot = 0
         class_means = np.array([mean_males, mean_females])
         for mean in class_means:
             tot += np.dot(W.T, mean)
             #print(tot)
         w0 = 0.5 * tot
         print("Calculated threshold is: ", w0)
```

```
Calculated threshold is:  -0.055232916501277526
```

```
In [99]: #Calculate Error
         #For each input project the point
         features = (new_data[:, 1:]).T
         labels = new_data[:,0]
         projected = np.dot(W.T, np.array(features))
         #projected
```

```
In [100]: #Assign Predictions
          predictions = []
          for item in projected:
```

```
        if item >= w0:
            predictions.append(2)
        else:
            predictions.append(1)
    #predictions
```

```
In [101]: #Check Classification
          errors = (labels != predictions)
          n_errors = sum(errors)

          error_rate = (n_errors/len(predictions) * 100)
          print("Error Rate is: ", error_rate, "%")
```

```
Error Rate is:  11.904761904761903 %
```

e. What's your height and weight? What's the model prediction for your case (male/female)?

```
In [102]: #My case
          my_height = 164
          my_weight = 65
          my_features = np.array([my_height, my_weight*my_height])
          my_ground_truth = "Male"

          #My Prediction
          my_projection = np.dot(W.T, my_features)
          if my_projection >= w0:
              my_pred = "Female"
          else:
              my_pred = "Male"

          print("In my case I was predicted as: ", my_pred, " which is ", my_ground_truth==my_
```

```
In my case I was predicted as:  Female  which is  False
```

```
In [103]: #Let's use Sklearn to see if our solution is correct
          #Using sklearn
          from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
          clf = LinearDiscriminantAnalysis()
          clf.fit(new_data[:, 1:], labels)
          LinearDiscriminantAnalysis(n_components=None, priors=None, shrinkage=None,
                        solver='eigen', store_covariance=False, tol=0.0001)
          print(clf.get_params())
          predictions = clf.predict(new_data[:, 1:])
          print(predictions)
          errors = sum(labels!=predictions)
          error_rate = (n_errors/len(predictions) * 100)
          print("Error Rate is: ", error_rate, "%")
          print("\nAs can be seen, our solution is right!")
```

```
{'n_components': None, 'priors': None, 'shrinkage': None, 'solver': 'svd', 'store_covariance':
[2. 2. 2. 2. 2. 2. 1. 2. 1. 2. 2. 1. 1. 2. 2. 2. 2. 2. 2. 1. 2. 2. 2. 1.
 1. 2. 2. 2. 2. 2. 1. 2. 2. 2. 1. 2. 2. 2. 1. 2. 2. 1. 2. 2. 1. 1. 2. 1.
 2. 2. 1. 2. 2. 1. 2. 1. 1. 2. 2. 2. 1. 2. 1. 2. 2. 2. 2. 1. 1. 2. 2. 1.
 2. 2. 2. 2. 1. 2. 1. 2. 2. 2. 2. 2. 2. 1. 2. 1. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 1. 1. 1. 1. 1. 2. 1. 1. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 1. 1. 1. 2.
 2. 2. 2. 2. 2. 1. 1. 1. 2. 1. 1. 1. 1. 2. 2. 2. 1. 2. 2. 2. 2. 1. 2. 2.
 1. 2. 2. 2. 2. 2. 1. 2. 1. 2. 2. 1. 2. 2. 2. 2. 1. 2. 2. 2. 2. 2. 2. 1.
 2. 1. 2. 2. 2. 1. 2. 2. 2. 2. 2. 2. 2. 1. 2. 2. 1. 2. 2. 2. 1. 2. 2. 1.
 2. 2. 2. 2. 1. 2. 2. 1. 2. 2. 1. 2. 1. 1. 1. 1. 2. 1.]
Error Rate is:  11.904761904761903 %

As can be seen, our solution is right!
```

2. Consider the Logistic Regression as discussed in the class. Assume now that the cost of erring an observation from class 1 is cost1 and the cost of erring observations from class 0 is cost0. How would you modify the goal function, gradient and hessian matrix (slides 11 and 12 in week 5)?

Change the code provided (or developed by you) in the class to receive as input the vector of costs. Test your code with the following script:

trainC1 = mvnrnd([21 21], [1 0; 0 1], 1000);
trainC0 = mvnrnd([23 23], [1 0; 0 1], 20);
testC1 = mvnrnd([21 21], [1 0; 0 1], 1000);
testC0 = mvnrnd([23 23], [1 0; 0 1], 1000);
NA = size(trainC1,1);
NB = size(trainC0,1);
traindata = [trainC1 ones(NA,1); trainC2 zeros(NB,1)]; %add class label in the last column
weights=logReg(traindata(:,1:end-1),traindata(:,end),[NB NA])
testC1 = [ones(size(testC1,1),1) testC1]; %add virtual feature for offset
testC0 = [ones(size(testC0,1),1) testC0]; %add virtual feature for offset
%FINISH the script to compute the recall, precision and F1 score in the test data

In this script the cost of erring in C1 is proportional to the elements in C0. Compute the precision, recall and F1 in the test data. Note: if you are unable to modify to account for costs, solve without costs.

In [104]:
```python
#Let's implement Logistic Regression according to the slides
#First define sigmoid function that will give us our hipothesis
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

#Define the log_likelihood
def log_likelihood(features,weights,labels):
    z = np.dot(features.T, weights)
    sigmoid_probs = sigmoid(z)
    #Cost 1 is proportional to the elements in C0
    cost1 = len(labels[labels==0])
```

```
    l_likell = np.sum((-np.log(sigmoid_probs)*cost1*labels) + ((-np.log(1-sigmoid_pro
    return l_likell

#Functions to predict probabilities and classes
def predict_proba(features, weights):
    z = np.dot(features, weights)
    proba = sigmoid(z)
    return proba

def predictions(features, weights, threshold):
    probs = predict_proba(features, weights)
    return probs >= threshold

#Define Gradient Function to be used in training phase; gradient descent!; Hessian wo
def gradient(features, labels, weights):
    z = np.dot(features, weights)
    sigmoid_probs = sigmoid(z)
    return np.dot(np.transpose(features), (sigmoid_probs - labels))
```

```
In [106]: def logReg(features, labels, learning_rate):
              # Initialize log_likelihood & parameters
              weights = np.zeros((features.shape[1], 1))
              l = log_likelihood(features, labels, weights)
              # Convergence Conditions
              max_iterations = 1000000
              for i in range(max_iterations):
                  g = gradient(features, labels, weights)
                  weights = weights - learning_rate*g
                  # Update the log-likelihood at each iteration
                  l_new = log_likelihood(features, labels, weights)
                  l = l_new
              return weights
```

```
In [107]: #Read Data
          trainC1 = np.random.multivariate_normal([21, 21], [[1, 0], [0, 1]], 1000);
          trainC0 = np.random.multivariate_normal([23, 23], [[1, 0], [0, 1]], 20);
          testC1 = np.random.multivariate_normal([21 , 21], [[1, 0], [0, 1]], 1000);
          testC0 = np.random.multivariate_normal([23, 23], [[1, 0], [0, 1]], 1000);

          #Build Train Data and add class label in the last column
          NA = int(trainC1.shape[0]);
          NB = int(trainC0.shape[0]);
          labels_C1 = np.ones([NA, 1])
          offset_C1 = np.ones((NA, 1))
          trainC1 = np.concatenate((offset_C1, trainC1, labels_C1), axis=1)
          offset_C0 = np.ones((NB, 1))
          labels_C0 = np.zeros([NB, 1])
          trainC0 = np.concatenate((offset_C0,trainC0, labels_C0), axis=1)
```

```
traindata = np.concatenate((trainC1, trainC0), axis=0)

#Compute Weights
weights=logReg(traindata[:, :3], traindata[:, 3:], learning_rate=0.001)
weights
```

Out[107]:
```
array([[350.37248684],
       [ -7.19467357],
       [ -8.18104244]])
```

In [108]:
```
#Test Data
#add virtual feature for offset
C1_virtualf = np.ones((int(testC1.shape[0]), 1))
testC1 = np.concatenate((C1_virtualf , testC1), axis=1);
C1test_labels = np.ones((int(testC1.shape[0]), 1))
testC1 = np.concatenate((testC1, C1test_labels), axis=1)
#add virtual feature for offset
C0_virtualf = np.ones((int(testC0.shape[0]), 1))
testC0 = np.concatenate((C0_virtualf, testC0), axis=1);
C0test_labels = np.zeros((int(testC0.shape[0]), 1))
testC0 = np.concatenate((testC0, C0test_labels), axis=1)
testdata = np.concatenate((testC1, testC0), axis=0)

#FINISH the script to compute the recall, precision and F1 score in the test data
```

In [109]:
```
from sklearn.metrics import confusion_matrix
#Predict on Test Data with the obtained weights
label_pred = predictions(testdata[:, :3], weights, 0.5)
label_pred = label_pred.astype(int)
labels = testdata[:, 3:]

tn, fp, fn, tp = confusion_matrix(labels, label_pred).ravel()
precision=tp/(tp+fp)
recall=tp/(tp+fn)
f1=2*((precision*recall)/(precision+recall))
print('Precision: ',precision)
print('Recall: ', recall)
print('F1: ',f1)
```

```
Precision:  0.7206946454413893
Recall:  0.996
F1:  0.8362720403022671
```

In [110]:
```
#Let's check with sklearn
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(random_state=0, solver='newton-cg', multi_class='ovr').fit(t
label_pred = clf.predict(testdata[:, :3])
tn, fp, fn, tp = confusion_matrix(testdata[:, 3:].ravel(), label_pred).ravel()
```

```
precision=tp/(tp+fp)
recall=tp/(tp+fn)
f1=2*((precision*recall)/(precision+recall))
print('Precision: ',precision)
print('Recall: ', recall)
print('F1: ',f1)
```

```
Precision:  0.6914008321775312
Recall:  0.997
F1:  0.8165438165438166
```

3. Several phenomena and concepts in real life applications are represented by angular data or, as is referred in the literature, directional data. Assume the directional variables are encoded as a periodic value in the range [0, 2]. Assume a two-class (y0 and y1), one dimensional classification task over a directional variable x, with equal a priori class probabilities.

a) If the class-conditional densities are defined as p(x|y0)= e2cos(x-1)/(2 2.2796) and p(x|y1)= e3cos(x+0.9)/(2 4.8808), what's the decision at x=0?

b) If the class-conditional densities are defined as p(x|y0)= e2cos(x-1)/(2 2.2796) and p(x|y1)= e3cos(x-1)/(2 4.8808), for what values of x is the prediction equal to y0?

c) Assume the more generic class-conditional densities defined as p(x|y0)= ek0cos(x- 0)/(2 I(k0)) and p(x|y1)= ek1cos(x-1)/(2 I(k1)). In these expressions, ki and i are constants and I(ki) is a constant that depends on ki. Show that the posterior probability p(y0|x) can be written as p(y0|x) = 1/(1+ew0+ w1sin(x-)), where w0, w1 and are parameters of the model (and depend on ki , i and I(ki)).

```
In [111]: #Imports
          from math import exp, cos, pi

          #Create functions
          #p(x|y0)= e2cos(x-1)/(2  2.2796)
          def p_x_y0(x):
              result = (exp(2*cos(x-1)))/(2*pi*2.2796)
              return result

          #p(x|y1)= e3cos(x+0.9)/(2  4.8808)
          def p_x_y1(x):
              result = (exp(3*cos(x+0.9)))/(2*pi*4.8808)
              return result
```

```
In [112]: #a)
          #Compute functions at x=0
          x0_y0 = p_x_y0(0)
          x0_y1 = p_x_y1(0)
          #print(x0_y0, x0_y1)
```

```
        #Decision at x=0 is equal to argmax(x0_y0, x0_y1), since a priori probabilities are
        if x0_y0 > x0_y1:
            decision = "y0"
        else:
            decision = "y1"

        print("At x=0, decision is: ", decision)
```

```
At x=0, decision is:  y1
```

In [113]:
```
#b
points = np.linspace(0, (2*pi), num=100)
#New p(x|y1)= e3cos(x-1)/(2  4.8808) funtion
def new_p_x_y1(x):
    result = (exp(3*cos(x-1)))/(2*p*4.8808)
    return result

#Compute values
x_y0 = []
for i in points:
    x_y0.append(p_x_y0(i))

x_y1 = []
for i in points:
    x_y1.append(p_x_y1(i))

results = []
for i in range(len(points)):
    if x_y0[i] > x_y1[i]:
        results.append(points[i])


print("The prediction of x is equal to y0 for the following points: \n")
for i in range(len(results)):
            print(results[i])
print("\nTotal number of points is: ", len(results))
```

```
The prediction of x is equal to y0 for the following points:

0.06346651825433926
0.12693303650867852
0.1903995547630178
0.25386607301735703
0.3173325912716963
0.3807991095260356
0.4442656277803748
0.5077321460347141
```

```
0.5711986642890533
0.6346651825433925
0.6981317007977318
0.7615982190520711
0.8250647373064104
0.8885312555607496
0.9519977738150889
1.0154642920694281
1.0789308103237674
1.1423973285781066
1.2058638468324459
1.269330365086785
1.3327968833411243
1.3962634015954636
1.4597299198498028
1.5231964381041423
1.5866629563584815
1.6501294746128208
1.71359599286716
1.7770625111214993
1.8405290293758385
1.9039955476301778
1.967462065884517
2.0309285841388562
2.0943951023931957
2.1578616206475347
2.221328138901874
2.284794657156213
2.3482611754105527
2.4117276936648917
2.475194211919231
2.53866073017357
2.6021272484279097
2.6655937666822487
2.729060284936588
2.792526803190927
2.8559933214452666
2.9194598396996057
2.982926357953945
3.0463928762082846
3.1098593944626236
3.173325912716963
3.236792430971302
3.3002589492256416
3.3637254674799806
3.42719198573432
3.490658503988659
```

```
Total number of points is:  55
```