

Data-Driven Classifiers

Neural Networks: ECE 5930

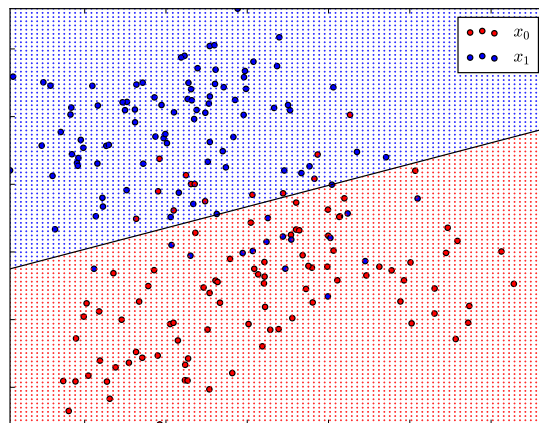


Figure: Linear Regression Classifier

Clint Ferrin
Utah State University
Fri Sep 28, 2017

Contents

1 Overview	1
2 Linear Regression	1
Problem 1:	1
Problem 2:	1
Problem 3:	2
3 Quadratic Regression	4
Problem 4:	4
Problem 5:	6
4 Linear and Quadratic Discriminant Analysis	7
Problem 6:	7
Problem 7:	7
5 Linear Logistic Regression	12
Problem 8:	12
Problem 9:	13
Problem 10:	14
Problem 11:	14
Problem 12:	15
6 k-nearest Neighbor Classifier	17
Problem 13:	17
7 Naive Bayes Classifier	20
Problem 14:	20
Problem 15:	21
8 Optimal Bayes Classifier	24
Problem 16:	24
9 Discussion	27

Problem 17:	27
-----------------------	----

List of Figures

1	Linear Classifier	2
2	Quadratic Regression Graph	4
3	Linear Discriminant Analysis	7
4	Quadratic Discriminant Function Classifier	8
5	Logistic Classifier	15
6	k=1 and k=5 Nearest Neighbor	17
7	k=15 Nearest Neighbor	18
8	Calculating Parzen Distribution	20
9	Altering the λ values in Parzen Distribution	20
10	Bayes 1D plot of X and Y	22
11	Naive Bayes Parzen Graph	22
12	Optimal Bayes Classifier	25

List of Tables

1	Nearest Neighbor Performance Comparison	17
2	Binary Classifier Performance Comparison	28

1 Overview

This Document answers 17 questions that walk through pattern recognition on binary static data. To view the problem set and description of several of these classifiers, visit [this link](https://drive.google.com/open?id=0B5NW7S3tXe5UTE0xSHJHNWxJbEE) or navigate to the following website: <https://drive.google.com/open?id=0B5NW7S3tXe5UTE0xSHJHNWxJbEE>

This document is not intended to be a comprehensive teaching document to describe each binary classifier, but rather aims to analyze some differences between a few classifiers as discussed in Section 9.

2 Linear Regression

Problem 1: Show that the β that minimizes $RSS(\beta)$ is $\beta = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$.

To prove the previous statement, we will multiply the polynomial out, and find where the derivative equals zero in order to minimize β .

$$\begin{aligned} (\mathbf{y} - \mathbf{X}\beta)^T (\mathbf{y} - \mathbf{X}\beta) &= \mathbf{y}^T \mathbf{y} - \mathbf{y}^T \mathbf{X}\beta - \mathbf{X}^T \beta^T \mathbf{y} + \mathbf{X}^T \beta^T \mathbf{X}\beta \\ &= \mathbf{X}^T \beta^T \mathbf{X}\beta - 2\mathbf{X}^T \beta^T \mathbf{y} + \mathbf{y}^T \mathbf{y} \end{aligned}$$

To find the minimized β we will now take the derivative and solve for β at zero.

$$\begin{aligned} \frac{d}{d\beta} \mathbf{X}^T \beta^T \mathbf{X}\beta - 2\mathbf{X}^T \beta^T \mathbf{y} + \mathbf{y}^T \mathbf{y} &= 0 \\ 2\mathbf{X}^T \mathbf{X}\beta - 2\mathbf{X}^T \mathbf{y} &= 0 \\ \mathbf{X}^T \mathbf{X}\beta &= \mathbf{X}^T \mathbf{y} \\ \beta &= (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \end{aligned} \tag{1}$$

Problem 2: Show that if the norm of $\|\mathbf{Y} - \mathbf{X}\hat{\mathbf{B}}\|^2$ is the Frobenius norm, then that the $\hat{\mathbf{B}}$ minimizing the same is determined by $\hat{\mathbf{B}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}$

Given that the Frobenius Norm for a matrix with real numbers is:

$$\sqrt{\text{Tr}(\mathbf{A}\mathbf{A}^T)}$$

Then the Frobenius Norm of the problem statement is:

$$\sqrt{\text{Tr}(\mathbf{Y} - \mathbf{X}\hat{\mathbf{B}})(\mathbf{Y} - \mathbf{X}\hat{\mathbf{B}})^T} = \sqrt{\text{Tr}(\mathbf{X}^T \hat{\mathbf{B}}^T \mathbf{X}\hat{\mathbf{B}} - 2\mathbf{X}^T \hat{\mathbf{B}}^T \mathbf{Y} + \mathbf{Y}^T \mathbf{Y})}$$

To find the $\hat{\mathbf{B}}$ minimizing the problem statement, we will take the deriving with respect to $\hat{\mathbf{B}}$

$$\begin{aligned} \frac{d}{d\hat{\mathbf{B}}} \left\| \sqrt{\text{Tr}(\mathbf{X}^T \hat{\mathbf{B}}^T \mathbf{X}\hat{\mathbf{B}} - 2\mathbf{X}^T \hat{\mathbf{B}}^T \mathbf{Y} + \mathbf{Y}^T \mathbf{Y})} \right\|^2 &= 0 \\ 2\mathbf{X}^T \mathbf{X}\hat{\mathbf{B}} - 2\mathbf{X}^T \mathbf{Y} &= 0 \\ \mathbf{X}^T \mathbf{X}\hat{\mathbf{B}} &= \mathbf{X}^T \mathbf{Y} \end{aligned}$$

$$\hat{\mathbf{B}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y} \tag{2}$$

Note that the trace could be removed from the equation because the result of the trace was zero, meaning that the sum of the trace was filled with all zeros.

Problem 3: Re-write the function `gendat2.m` into Python. Using the 100 points of training data in `classasgntrain1.dat`, write PYTHON code to train the coefficient matrix $\hat{\beta}$.

The program produced the desired results and the outcome can be seen in Figure 1. Note that the data is not completely linearly separable, and there were 29 errors on the wrong side of the line after it was drawn. The results of the outcome can be seen in Table 2.

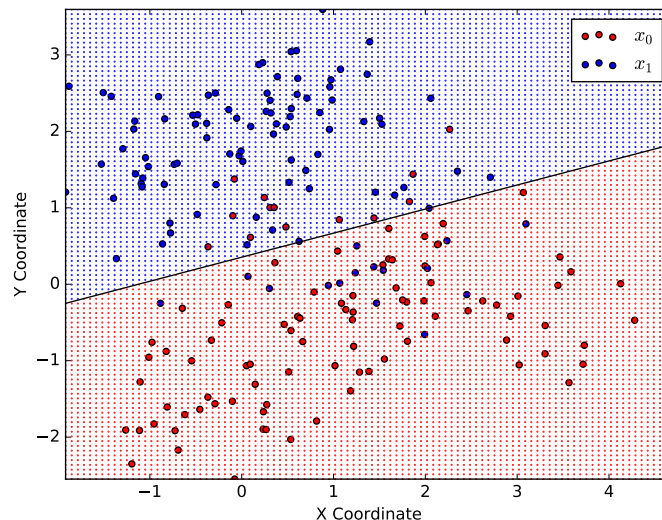


Figure 1: Linear Classifier

```

1 # Clint Ferrin
2 # Mon Sep 25, 2017
3 # Linear Classifier
4
5 import sys
6 import numpy as np
7 import matplotlib.pyplot as plt
8
9 def gendata2(class_type,N):
10     m0 = np.array(
11         [[-0.132,0.320,1.672,2.230,1.217,-0.819,3.629,0.8210,1.808, 0.1700],
12          [-0.711,-1.726,0.139,1.151,-0.373,-1.573,-0.243,-0.5220,-0.511,0.5330]])
13
14     m1 = np.array(
15         [[-1.169,0.813,-0.859,-0.608,-0.832,2.015,0.173,1.432,0.743,1.0328],
16          [ 2.065,2.441,0.247,1.806,1.286,0.928,1.923,0.1299,1.847,-0.052]])
17
18     x = np.array([[ ],[ ]])
19     for i in range(N):
20         idx = np.random.randint(10)
21         if class_type == 0:
22             m = m0[:,idx]
23         elif class_type == 1:
24             m = m1[:,idx]
25         else:
26             print("not a proper classifier")
27             return 0
28         x = np.c_[x, [[m[0]], [m[1]]] + np.random.randn(2,1)/np.sqrt(5)]
29     return x
30
31 def plotData(x0,x1):

```

```

32     fig = plt.figure() # make handle to save plot
33     plt.scatter(x0[0,:],x0[1,:],c='red',label='$x_0$')
34     plt.scatter(x1[0,:],x1[1,:],c='blue',label='$x_1$')
35     plt.xlabel('X Coordinate')
36     plt.ylabel('Y Coordinate')
37     plt.legend()
38
39
40 data = np.loadtxt("../data/classasgntrain1.dat",dtype=float)
41 x0 = data[:,0:2].T
42 x1 = data[:,2:4].T
43 data_tot = np.c_[x0,x1]
44
45 N0 = x0.shape[1]
46 N1 = x1.shape[1];
47 N = N0 + N1
48
49 # linear regression classifier
50 X = np.r_[np.c_[np.ones((N0,1)),x0.T],
51           np.c_[np.ones((N1,1)),x1.T]]
52
53 Y = np.r_[np.c_[np.ones((N0,1)),np.zeros((N0,1))],
54           np.c_[np.zeros((N1,1)),np.ones((N1,1))]]
55
56 # find parameter matrix
57 Bhat = np.dot(np.linalg.inv(np.dot(X.T,X)),np.dot(X.T,Y))
58
59 # find approximate response
60 Yhat = np.dot(X,Bhat)
61 Yhathard = Yhat > 0.5
62
63 num_err = sum(sum(abs(Yhathard - Y)))/2
64 print("Number of errors: %d"%(num_err))
65
66 Ntest0 = 10000;
67 Ntest1 = 10000;
68
69 err_rate_linregress_train = float(num_err) / N
70 print("Percent of errors: %.4f"%(err_rate_linregress_train))
71
72 # generate the test data for class 0
73 xtest0 = gendata2(0,Ntest0)
74 xtest1 = gendata2(1,Ntest1)
75 num_err = 0;
76
77 for i in range(Ntest0):
78     yhat = np.dot(np.r_[1,xtest0[:,i]],Bhat)
79     if yhat[1] > yhat[0]:
80         num_err = num_err + 1;
81
82 for i in range(Ntest1):
83     yhat = np.dot(np.r_[1,xtest1[:,i]],Bhat)
84     if yhat[1] < yhat[0]:
85         num_err = num_err + 1;
86
87 print("Number of errors: %d"%(num_err))
88 err_rate_linregress_test = float(num_err) / (Ntest0 + Ntest1);
89 print("Percent of errors: %.4f"%(err_rate_linregress_test))
90
91
92 # find max and min of sets
93 x_tot = np.r_[x0[0,:],x1[0,:]]
94 y_tot = np.r_[x0[1,:],x1[1,:]]
95 xlim = [np.min(x_tot),np.max(x_tot)]
96 ylim = [np.min(y_tot),np.max(y_tot)]
97
98 # find x,y coordinate of separating line
99 x_cor_lin = [xlim[0],xlim[1]]
100 y_cor_lin = [
101     (Bhat[0,0]-Bhat[0,1]+(Bhat[1,0]-Bhat[1,1])*xlim[0])
102     / (Bhat[2,1]-Bhat[2,0]),
103
104     (Bhat[0,0]-Bhat[0,1]+(Bhat[1,0]-Bhat[1,1])*xlim[1])

```

```

105         / (Bhat[2,1]-Bhat[2,0])
106     ]
107
108     # create colored graph above/below line
109     xpl = np.linspace(xlim[0],xlim[1], num=100)
110     ypl = np.linspace(ylim[0],ylim[1], num=100)
111
112     red_pts = np.array([],[])
113     green_pts= np.array([],[])
114
115     for x in xpl:
116         for y in ypl:
117             yhat = np.dot(np.r_[1,x,y],Bhat)
118             if yhat[1] > yhat[0]:
119                 green_pts = np.c_[green_pts,[x,y]]
120             else:
121                 red_pts = np.c_[red_pts,[x,y]]
122
123     plotData(x0,x1)
124     plt.plot(x_cor_lin,y_cor_lin,color='black')
125     plt.scatter(green_pts[0:],green_pts[1:],color='blue',s=0.25)
126     plt.scatter(red_pts[0:],red_pts[1:],color='red',s=0.25)
127     plt.xlim(xlim)
128     plt.ylim(ylim)
129     plt.show()

```

3 Quadratic Regression

Problem 4: For the data described in Problem 3, train the regression coefficient matrix \hat{B} . Determine the classification error rate on the training data and 10,000 points of test data (as before) and fill in the corresponding row of the results table. Plot the classification regions as before.

The program performed as expected, and the outcome graph can be seen in Figure 2. Note that due to the data that appears mostly linearly separable, the line does not curve much. The results of the program can be seen in Table 2.

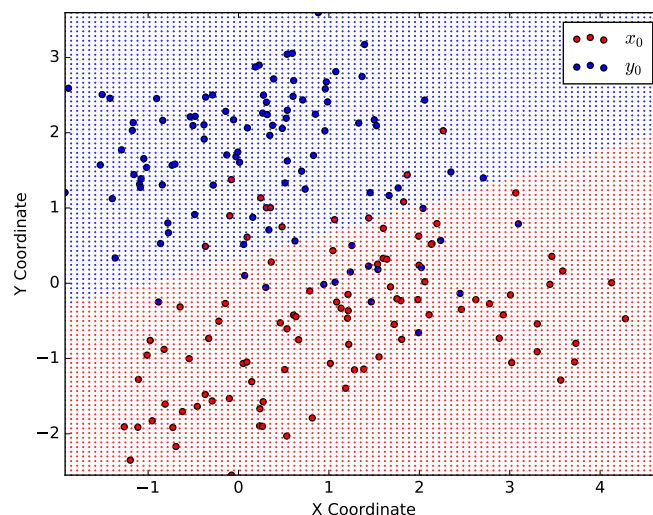


Figure 2: Quadratic Regression Graph

```

1 # Clint Ferrin
2 # Mon Sep 25, 2017

```

```

3 # Quadratic Classifier
4
5 import sys
6 import numpy as np
7 import matplotlib.pyplot as plt
8
9 def gendata2(class_type,N):
10     m0 = np.array(
11         [[-0.132,0.320,1.672,2.230,1.217,-0.819,3.629,0.8210,1.808, 0.1700],
12          [-0.711,-1.726,0.139,1.151,-0.373,-1.573,-0.243,-0.5220,-0.511,0.5330]])
13
14     m1 = np.array(
15         [[-1.169,0.813,-0.859,-0.608,-0.832,2.015,0.173,1.432,0.743,1.0328],
16          [ 2.065,2.441,0.247,1.806,1.286,0.928,1.923,0.1299,1.847,-0.052]])
17
18     x = np.array([[ ],[ ]])
19     for i in range(N):
20         idx = np.random.randint(10)
21         if class_type == 0:
22             m = m0[:,idx]
23         elif class_type == 1:
24             m = m1[:,idx]
25         else:
26             print("not a proper classifier")
27             return 0
28         x = np.c_[x, [[m[0]], [m[1]]] + np.random.randn(2,1)/np.sqrt(5)]
29     return x
30
31 data = np.loadtxt("../data/classasgntrain1.dat",dtype=float)
32 x0 = data[:,0:2].T
33 x1 = data[:,2:4].T
34 data_tot = np.c_[x0,x1]
35
36 fig = plt.figure() # make handle to save plot
37 plt.scatter(x0[0,:],x0[1,:],c='red',label='$x_0$')
38 plt.scatter(x1[0,:],x1[1,:],c='blue',label='$y_0$')
39 plt.xlabel('X Coordinate')
40 plt.ylabel('Y Coordinate')
41 plt.legend()
42
43 N0 = x0.shape[1]
44 N1 = x1.shape[1];
45 N = N0 + N1
46
47 # quadratic
48 X = np.c_[np.ones((N,1)),data_tot.T,data_tot[0].T*data_tot[0].T, data_tot[0]*data_tot[1],
49           data_tot[1]*data_tot[1]]
50
51 Y = np.r_[np.c_[np.ones((N0,1)),np.zeros((N0,1))],
52           np.c_[np.zeros((N1,1)),np.ones((N1,1))]]
53
54 # find parameter matrix
55 Bhat = np.linalg.lstsq(np.dot(X.T,X),np.dot(X.T,Y))[0]
56
57 # find approximate response
58 Yhat = np.dot(X,Bhat)
59 Yhathard = Yhat > 0.5
60
61 num_err = sum(sum(abs(Yhathard - Y)))/2
62
63 Ntest0 = 10000;
64 Ntest1 = 10000;
65
66 err_rate_linregress_train = float(num_err) / N
67
68 print(err_rate_linregress_train)
69 # generate the test data for class 0
70 xtest0 = gendata2(0,Ntest0)
71 xtest1 = gendata2(1,Ntest1)
72 num_err = 0;
73
74 for i in range(Ntest0):
75     yhat = np.dot(np.r_[1,xtest0[:,i],xtest0[0,i]*xtest0[0,i], xtest0[0,i]*xtest0[1,i],xtest0

```



```

    [1,i]*xtest0[1,i]],Bhat)
75     if yhat[1] > yhat[0]:
76         num_err = num_err + 1;
77
78 for i in range(Ntest1):
79     yhat = np.dot(np.r_[1,xtest1[:,i],xtest1[0,i]*xtest1[0,i], xtest1[0,i]*xtest1[1,i],xtest1
    [1,i]*xtest1[1,i]],Bhat)
80     if yhat[1] < yhat[0]:
81         num_err = num_err + 1;
82
83 print("Number of errors: %d"%(num_err))
84
85 err_rate_linregress_test = float(num_err) / (Ntest0 + Ntest1);
86 print(err_rate_linregress_test)
87
88
89 # find max and min of sets
90 x_tot = np.r_[x0[0,:],x1[0,:]]
91 y_tot = np.r_[x0[1,:],x1[1,:]]
92 xlim = [np.min(x_tot),np.max(x_tot)]
93 ylim = [np.min(y_tot),np.max(y_tot)]
94
95 # create colored graph above/below line
96 xpl = np.linspace(xlim[0],xlim[1], num=100)
97 ypl = np.linspace(ylim[0],ylim[1], num=100)
98
99 red_pts = np.array([[[]],[[]]])
100 green_pts= np.array([[[]],[[]]])
101
102 for x in xpl:
103     for y in ypl:
104         yhat = np.dot(np.r_[1,x,y,x*x,x*y,y*y],Bhat)
105         if yhat[1] > yhat[0]:
106             green_pts = np.c_[green_pts,[x,y]]
107         else:
108             red_pts = np.c_[red_pts,[x,y]]
109
110 plt.scatter(green_pts[0:],green_pts[1:],color='blue',s=0.25)
111 plt.scatter(red_pts[0:],red_pts[1:],color='red',s=0.25)
112 plt.xlim(xlim)
113 plt.ylim(ylim)
114 plt.show()

```

Problem 5: Show that $\log P(\text{class} = k | X = x) = \log \hat{\pi}_k - \frac{1}{2} \log |\hat{R}_k| - \frac{1}{2} (x - \hat{\mu}_k)^T \hat{R}_k^{-1} (x - \hat{\mu}_k)$ is true. In particular, make sure you understand what is meant by “up to a constant which does not depend on the class”

$$f_k(x) = \frac{1}{(2\pi)^{d/2} |\hat{R}_k|^{1/2}} \exp\left[-\frac{1}{2} (x - \hat{\mu}_k)^T \hat{R}_k^{-1} (x - \hat{\mu}_k)\right]$$

Using Bayes rule, we can produce the following form. Note: When using Bayes Rule, constants exuding the random variable can be eliminated without affecting the results:

$$\hat{\pi}_k |\hat{R}_k|^{-1/2} \exp\left[-\frac{1}{2} (x - \hat{\mu}_k)^T \hat{R}_k^{-1} (x - \hat{\mu}_k)\right]$$

Now taking the log of the equation gives us:

$$\log \hat{\pi}_k - \frac{1}{2} \log |\hat{R}_k| - \frac{1}{2} (x - \hat{\mu}_k)^T \hat{R}_k^{-1} (x - \hat{\mu}_k) \quad (3)$$

4 Linear and Quadratic Discriminant Analysis

Problem 6: For the data set described in problem 3, build a LDA classifier. That is, train sample means for each class and population co-variance, and classify based on the linear discriminant functions in $\delta_k^l = \mathbf{x}^T \hat{\mathbf{R}}^{-1} \hat{\boldsymbol{\mu}}_k - \frac{1}{2} \hat{\boldsymbol{\mu}}_k^T \hat{\mathbf{R}}^{-1} \hat{\boldsymbol{\mu}}_k + \log \pi_k$. Characterize the error rate on the training data and on 10,000 points of test data. Plot the classification regions as before.

The graph seen in Figure 3 shows the effectiveness of the linear discriminant analysis. The error rates and results for the LDA can be seen in Table 2, and the code for the classifier is seen following Figure 4.

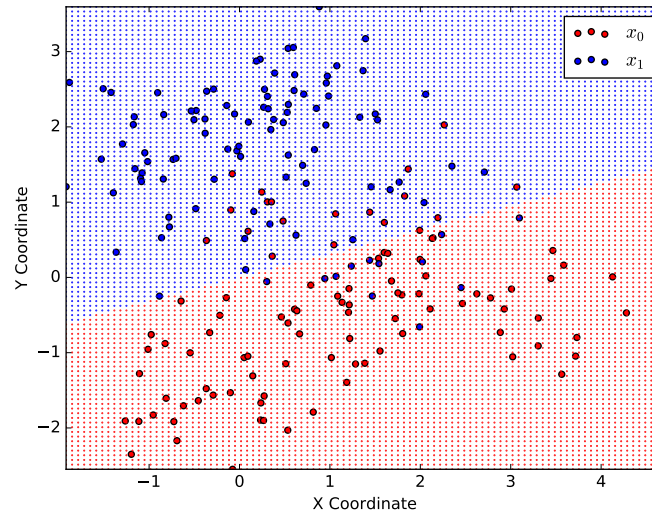


Figure 3: Linear Discriminant Analysis

Problem 7: For the data set described in problem 3, build a QDA classifier. In this case, you will also need to build the class co-variance matrices. Classify based on the quadratic discriminant functions in the equation $\log \hat{\pi}_k - \frac{1}{2} \log |\hat{\mathbf{R}}_k| - \frac{1}{2} (\mathbf{x} - \hat{\boldsymbol{\mu}}_k)^T \hat{\mathbf{R}}_k^{-1} (\mathbf{x} - \hat{\boldsymbol{\mu}}_k)$. Characterize the error rate on the training data and on 10,000 points of test data. Plot the classification regions as before. Compare the decision boundaries between QDA and quadratic regression.

The plotted data for Problem 7 can be seen in Figure 4. The co-variance matrix can be seen in the second code block below, and it is referenced to as `Rhat` in the code. The decision boundaries between the LDA and QDA are significantly different; the LDA has a linear shape almost exactly the same as the Linear Regression, whereas the QDA has a steep curve towards the `class1` data as seen in Figure 4. This curve can allow the classifier to be more sensitive to nonlinearities.

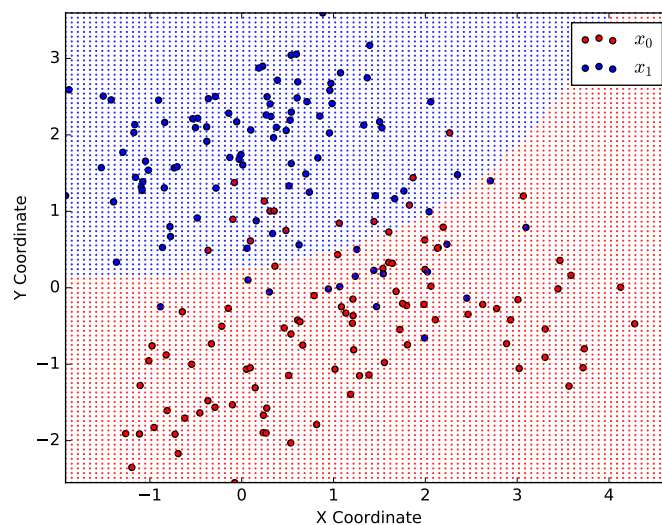


Figure 4: Quadratic Discriminant Function Classifier

```

1 # Clint Ferrin
2 # Mon Sep 25, 2017
3 # Linear Discriminant Analysis
4
5 import sys
6 import numpy as np
7 import matplotlib.pyplot as plt
8
9 def gendata2(class_type,N):
10     m0 = np.array(
11         [[-0.132,0.320,1.672,2.230,1.217,-0.819,3.629,0.8210,1.808, 0.1700],
12          [-0.711,-1.726,0.139,1.151,-0.373,-1.573,-0.243,-0.5220,-0.511,0.5330]])
13
14     m1 = np.array(
15         [[-1.169,0.813,-0.859,-0.608,-0.832,2.015,0.173,1.432,0.743,1.0328],
16          [ 2.065,2.441,0.247,1.806,1.286,0.928,1.923,0.1299,1.847,-0.052]])
17
18     x = np.array([[ ],[ ]])
19     for i in range(N):
20         idx = np.random.randint(10)
21         if class_type == 0:
22             m = m0[:,idx]
23         elif class_type == 1:
24             m = m1[:,idx]
25         else:
26             print("not a proper classifier")
27             return 0
28         x = np.c_[x, [[m[0]], [m[1]]] + np.random.randn(2,1)/np.sqrt(5)]
29     return x
30
31 def getRhat(x0,x1):
32     N0 = x0.shape[1]
33     N1 = x1.shape[1]
34     N = N0 + N1
35
36     mu = np.array([[np.mean(x0[0,:]),np.mean(x0[1,:])],
37                   [np.mean(x1[0,:]),np.mean(x1[1,:])]])
38
39     Rhat = np.empty([2,2])
40     for i in range(N0):
41         Rhat = Rhat + np.outer(x0[:,i]-mu[0],x0[:,i]+mu[0])
42
43     for i in range(N1):
44         Rhat = Rhat + np.outer(x1[:,i]-mu[1],x1[:,i]+mu[1])
45

```

```

46     return Rhat / (N-2)
47
48 def calcDel(data, Rhat, mu, N0, N):
49     return np.dot(np.dot(data, np.linalg.inv(Rhat)), mu)
50     - 0.5 * np.dot(np.dot(mu, np.linalg.inv(Rhat)), mu)
51     + np.log(N0) - np.log(N)
52
53 def getDel(x0, x1, Rhat):
54     N0 = x0.shape[1]
55     N1 = x1.shape[1]
56     N = N0 + N1
57     data_tot = np.c_[x0, x1]
58     num_errors = 0
59     mu = [[np.mean(x0[0, :]), np.mean(x0[1, :])],
60           [np.mean(x1[0, :]), np.mean(x1[1, :])]]
61
62     del_l = np.array([[[]], [[]]])
63
64     for i in range(N):
65         del_l = np.c_[del_l, np.array(
66             [calcDel(data_tot[:, i], Rhat, mu[0], N0, N),
67              calcDel(data_tot[:, i], Rhat, mu[1], N0, N) ]) .T ]
68
69     for i in range(N0):
70         if del_l[0, i] < del_l[1, i]:
71             num_errors = num_errors + 1
72
73     for i in range(N1):
74         if del_l[0, N0 + i] > del_l[1, N0 + i]:
75             num_errors = num_errors + 1
76
77     # return an array of 2 values for every point. Larger=class
78     return del_l, num_errors
79
80
81 data = np.loadtxt("../data/classasgntrain1.dat", dtype=float)
82 x0 = data[:, 0:2].T
83 x1 = data[:, 2:4].T
84
85 N0 = x0.shape[1]
86 N1 = x1.shape[1]
87 N = N0 + N1
88
89 mu = np.array([[np.mean(x0[0, :]), np.mean(x0[1, :])],
90               [np.mean(x1[0, :]), np.mean(x1[1, :])]])
91
92 fig = plt.figure() # make handle to save plot
93 plt.scatter(x0[0, :], x0[1, :], c='red', label='$x_0$')
94 plt.scatter(x1[0, :], x1[1, :], c='blue', label='$x_1$')
95 plt.xlabel('X Coordinate')
96 plt.ylabel('Y Coordinate')
97 plt.legend()
98
99 # find parameter matrix
100 Rhat = getRhat(x0, x1)
101
102 del_l, num_err = getDel(x0, x1, Rhat)
103 print("Number of Errors: %d" % (num_err))
104 print("Percent errors: %.4f" % (float(num_err) / N))
105 Ntest0 = 10000;
106 Ntest1 = 10000;
107
108 # generate the test data for class 0
109 xtest0 = gendata2(0, Ntest0)
110 xtest1 = gendata2(1, Ntest1)
111
112 del_l, num_err = getDel(xtest0, xtest1, Rhat)
113
114 np.savetxt('output.out', del_l)
115
116 print("Number of Errors: %d" % (num_err))
117 print("Percent errors: %.4f" % (float(num_err) / (Ntest0 + Ntest1)))
118

```

```

119 # find max and min of sets
120 x_tot = np.r_[x0[0,:],x1[0,:]]
121 y_tot = np.r_[x0[1,:],x1[1,:]]
122 xlim = [np.min(x_tot),np.max(x_tot)]
123 ylim = [np.min(y_tot),np.max(y_tot)]
124
125 # create colored graph above/below line
126 xpl = np.linspace(xlim[0],xlim[1], num=100)
127 ypl = np.linspace(ylim[0],ylim[1], num=100)
128
129 red_pts = np.array([],[])
130 green_pts= np.array([],[])
131
132 for x in xpl:
133     for y in ypl:
134         del_l = np.array(
135             [calcDel([x,y],Rhat,mu[0],N0,N),
136              calcDel([x,y],Rhat,mu[1],N1,N)])
137
138         if del_l[0]<del_l[1]:
139             green_pts = np.c_[green_pts,[x,y]]
140         else:
141             red_pts = np.c_[red_pts,[x,y]]
142
143 plt.scatter(green_pts[0:],green_pts[1:],color='blue',s=0.25)
144 plt.scatter(red_pts[0:],red_pts[1:],color='red',s=0.25)
145 plt.xlim(xlim)
146 plt.ylim(ylim)
147 plt.show()

```

```

1 # Clint Ferrin
2 # Mon Sep 25, 2017
3 # Quadratic Discriminant Analysis
4 import sys
5 import numpy as np
6 import matplotlib.pyplot as plt
7
8 def gendata2(class_type,N):
9     m0 = np.array(
10         [[-0.132,0.320,1.672,2.230,1.217,-0.819,3.629,0.8210,1.808, 0.1700],
11          [-0.711,-1.726,0.139,1.151,-0.373,-1.573,-0.243,-0.5220,-0.511,0.5330]])
12
13     m1 = np.array(
14         [[-1.169,0.813,-0.859,-0.608,-0.832,2.015,0.173,1.432,0.743,1.0328],
15          [ 2.065,2.441,0.247,1.806,1.286,0.928,1.923,0.1299,1.847,-0.052]])
16
17     x = np.array([],[])
18     for i in range(N):
19         idx = np.random.randint(10)
20         if class_type == 0:
21             m = m0[:,idx]
22         elif class_type == 1:
23             m = m1[:,idx]
24         else:
25             print("not a proper classifier")
26             return 0
27         x = np.c_[x, [m[0]], [m[1]]] + np.random.randn(2,1)/np.sqrt(5)]
28     return x
29
30 def getRhat(x0,x1):
31     N0 = x0.shape[1]
32     N1 = x1.shape[1]
33     N = N0 + N1
34
35     mu = np.array([ [np.mean(x0[0,:]),np.mean(x0[1,:])],
36                    [np.mean(x1[0,:]),np.mean(x1[1,:])] ])
37
38     Rhat = [np.empty([2,2]),np.empty([2,2])]
39     for i in range(N0):
40         Rhat[0] = Rhat[0] + np.outer(x0[:,i]-mu[0],x0[:,i]+mu[0])
41
42     for i in range(N1):
43         Rhat[1] = Rhat[1] + np.outer(x1[:,i]-mu[1],x1[:,i]+mu[1])

```

```

44     return Rhat[0]/(N0-1), Rhat[1]/(N1-1)
45
46 def calcDelQDA(data, Rhat, mu, N0, N):
47     return (np.log(N0)-np.log(N))-0.5*np.log(np.linalg.norm(Rhat))-0.5*np.dot(np.dot((data-mu)
48         .T, np.linalg.inv(Rhat)), data-mu)
49
50 def getDel(x0, x1, Rhat):
51     N0 = x0.shape[1]
52     N1 = x1.shape[1]
53     N = N0 + N1
54
55     num_errors = 0
56
57     mu = [[np.mean(x0[0,:]), np.mean(x0[1,:])],
58         [np.mean(x1[0,:]), np.mean(x1[1,:])]]
59
60     del_l = np.array([[0], [0]])
61     for i in range(N0):
62         del_l = np.c_[del_l, np.array([
63             calcDelQDA(x0[:,i], Rhat[0], mu[0], N0, N),
64             calcDelQDA(x0[:,i], Rhat[1], mu[1], N0, N)]) .T ]
65
66     for i in range(N1):
67         del_l = np.c_[del_l, np.array([
68             calcDelQDA(x1[:,i], Rhat[0], mu[0], N0, N),
69             calcDelQDA(x1[:,i], Rhat[1], mu[1], N0, N)]) .T ]
70
71     for i in range(N0):
72         if del_l[0,i]<del_l[1,i]:
73             num_errors=num_errors+1
74
75     for i in range(N1):
76         if del_l[0,N0+i]>del_l[1,N0+i]:
77             num_errors=num_errors+1
78
79     # return an array of 2 values for every point. Larger=class
80     return del_l, num_errors
81
82 # def returnBound(x, Rhat, mu):
83
84
85 data = np.loadtxt("../data/classasgntrain1.dat", dtype=float)
86 x0 = data[:,0:2].T
87 x1 = data[:,2:4].T
88
89 N0 = x0.shape[1]
90 N1 = x1.shape[1]
91 N = N0 + N1
92
93 mu = np.array([[np.mean(x0[0,:]), np.mean(x0[1,:])],
94     [np.mean(x1[0,:]), np.mean(x1[1,:])]])
95
96
97 fig = plt.figure() # make handle to save plot
98 plt.scatter(x0[0,:], x0[1,:], c='red', label='$x_0$')
99 plt.scatter(x1[0,:], x1[1,:], c='blue', label='$x_1$')
100 plt.xlabel('X Coordinate')
101 plt.ylabel('Y Coordinate')
102 plt.legend()
103
104 # find parameter matrix
105 Rhat = getRhat(x0, x1)
106 print(Rhat)
107 del_l, num_err = getDel(x0, x1, Rhat)
108
109 Ntest0 = 10000;
110 Ntest1 = 10000;
111
112
113 print(float(num_err)/N)
114 # generate the test data for class 0
115 xtest0 = gendata2(0, Ntest0)

```

```

116 xtest1 = gendata2(1,Ntest1)
117
118 del_l,num_err = getDel(xtest0,xtest1,Rhat)
119 print(num_err)
120
121 print("Percent errors: %.4f"%(float(num_err)/(Ntest0 + Ntest1)))
122
123 # find max and min of sets
124 x_tot = np.r_[x0[0,:],x1[0,:]]
125 y_tot = np.r_[x0[1,:],x1[1,:]]
126 xlim = [np.min(x_tot),np.max(x_tot)]
127 ylim = [np.min(y_tot),np.max(y_tot)]
128
129 # create colored graph above/below line
130 xpl = np.linspace(xlim[0],xlim[1], num=100)
131 ypl = np.linspace(ylim[0],ylim[1], num=100)
132
133 red_pts = np.array([],[])
134 green_pts= np.array([],[])
135
136 for x in xpl:
137     for y in ypl:
138         del_l = np.array(
139             [calcDelQDA([x,y],Rhat[0],mu[0],N0,N),
140              calcDelQDA([x,y],Rhat[1],mu[1],N1,N)])
141
142         if del_l[0]<del_l[1]:
143             green_pts = np.c_[green_pts,[x,y]]
144         else:
145             red_pts = np.c_[red_pts,[x,y]]
146
147 plt.scatter(green_pts[0,:],green_pts[1:],color='blue',s=0.25)
148 plt.scatter(red_pts[0,:],red_pts[1:],color='red',s=0.25)
149 plt.xlim(xlim)
150 plt.ylim(ylim)
151 plt.show()

```

5 Linear Logistic Regression

Problem 8: Using the probability model $P(Y = 0|X = \mathbf{x}) = \frac{1}{1+\exp[-\beta^T \mathbf{x}]}$, show that $l(\beta)$ can be written as

$$l(\beta) = \sum_{i=1}^N y_i \beta^T \mathbf{x}_i - \log(1 + e^{\beta^T \mathbf{x}_i})$$

The following note was necessary to remove a negative sign from the leading term of the result.

Note:

$$\frac{e^{\beta^T \mathbf{x}_i}}{1 + e^{\beta^T \mathbf{x}_i}} = \frac{1}{1 + e^{-\beta^T \mathbf{x}_i}}$$

We begin with the equation:

$$\begin{aligned}
l(\beta) &= \sum_{i=1}^N y_i \log p(\mathbf{x}_i; \beta) + (1 - y_i) \log(1 - p(\mathbf{x}_i; \beta)) \\
&= \sum_{i=1}^N y_i \log\left(\frac{1}{1 + e^{-\beta^T \mathbf{x}_i}}\right) + (1 - y_i) \log\left(1 - \frac{1}{1 + e^{-\beta^T \mathbf{x}_i}}\right) \\
&= \sum_{i=1}^N -y_i \log(1 + e^{-\beta^T \mathbf{x}_i}) + (1 - y_i) \log\left(\frac{1 + e^{-\beta^T \mathbf{x}_i}}{1 + e^{-\beta^T \mathbf{x}_i}} - \frac{1}{1 + e^{-\beta^T \mathbf{x}_i}}\right) \\
&= \sum_{i=1}^N -y_i \log(1 + e^{-\beta^T \mathbf{x}_i}) + (1 - y_i) \log\left(\frac{e^{-\beta^T \mathbf{x}_i}}{1 + e^{-\beta^T \mathbf{x}_i}}\right) \\
&= \sum_{i=1}^N -y_i \log(1 + e^{-\beta^T \mathbf{x}_i}) + (1 - y_i)(\log(e^{-\beta^T \mathbf{x}_i}) - \log(1 + e^{-\beta^T \mathbf{x}_i})) \\
&= \sum_{i=1}^N -y_i \log(1 + e^{-\beta^T \mathbf{x}_i}) + (1 - y_i)(-\beta^T \mathbf{x}_i - \log(1 + e^{-\beta^T \mathbf{x}_i})) \\
&= \sum_{i=1}^N -y_i \log(1 + e^{-\beta^T \mathbf{x}_i}) - \beta^T \mathbf{x}_i - \log(1 + e^{-\beta^T \mathbf{x}_i}) + y_i \beta^T \mathbf{x}_i + y_i \log(1 + e^{-\beta^T \mathbf{x}_i}) \\
&= \sum_{i=1}^N y_i \beta^T \mathbf{x}_i - \beta^T \mathbf{x}_i - \log(1 + e^{-\beta^T \mathbf{x}_i}) \\
&= \sum_{i=1}^N y_i \beta^T \mathbf{x}_i - (\beta^T \mathbf{x}_i + \log(1 + e^{-\beta^T \mathbf{x}_i})) \\
&= \sum_{i=1}^N y_i \beta^T \mathbf{x}_i - (\log(e^{\beta^T \mathbf{x}_i}) + \log(1 + e^{-\beta^T \mathbf{x}_i})) \\
l(\beta) &= \sum_{i=1}^N y_i \beta^T \mathbf{x}_i - \log(1 + e^{\beta^T \mathbf{x}_i}) \tag{4}
\end{aligned}$$

Problem 9: Show that

$$\frac{\partial l(\beta)}{\partial \beta} = \sum_{i=1}^N \mathbf{x}_i (y_i - p(\mathbf{x}_i; \beta))$$

Starting with the equation from the previous problem.

$$\begin{aligned}
&\frac{\partial}{\partial \beta} \sum_{i=1}^N y_i \beta^T \mathbf{x}_i - \log(1 + e^{\beta^T \mathbf{x}_i}) \\
&\quad \sum_{i=1}^N y_i \mathbf{x}_i - \frac{\partial}{\partial \beta} \log(1 + e^{\beta^T \mathbf{x}_i}) \\
&\quad \sum_{i=1}^N y_i \mathbf{x}_i - \frac{\mathbf{x}_i e^{\beta^T \mathbf{x}_i}}{1 + e^{\beta^T \mathbf{x}_i}} \\
&\quad \sum_{i=1}^N y_i \mathbf{x}_i - \frac{\mathbf{x}_i e^{\beta^T \mathbf{x}_i}}{1 + e^{\beta^T \mathbf{x}_i}} \\
&\quad \sum_{i=1}^N y_i \mathbf{x}_i - \mathbf{x}_i p(\mathbf{x}_i; \beta)
\end{aligned}$$

$$\sum_{i=1}^N \mathbf{x}_i (y_i - p(\mathbf{x}_i; \beta)) \quad (5)$$

Problem 10: Show that

$$\frac{\partial^2 l(\beta)}{\partial \beta \partial \beta^T} = \sum_{i=1}^N \mathbf{x}_i \mathbf{x}_i^T p(\mathbf{x}_i; \beta) (1 - p(\mathbf{x}_i; \beta))$$

Beginning with the first derivative from the previous problem:

$$\begin{aligned} \frac{\partial}{\partial \beta} \sum_{i=1}^N \mathbf{x}_i (y_i - p(\mathbf{x}_i; \beta)) \\ \sum_{i=1}^N -\frac{\partial}{\partial \beta} \mathbf{x}_i p(\mathbf{x}_i; \beta) \\ \sum_{i=1}^N -\mathbf{x}_i \frac{\partial}{\partial \beta} \frac{1}{1 + e^{\beta^T \mathbf{x}_i}} \\ \sum_{i=1}^N \frac{-\mathbf{x}_i e^{\beta^T \mathbf{x}_i} \mathbf{x}_i^T}{(1 + e^{\beta^T \mathbf{x}_i})^2} \\ \sum_{i=1}^N \frac{-\mathbf{x}_i e^{\beta^T \mathbf{x}_i} \mathbf{x}_i^T}{(1 + e^{\beta^T \mathbf{x}_i})^2} \end{aligned}$$

Note:

$$\begin{aligned} 1 - p(\mathbf{x}_i; \beta) &= \frac{1 + e^{\beta^T \mathbf{x}_i}}{1 + e^{\beta^T \mathbf{x}_i}} - \frac{1}{1 + e^{\beta^T \mathbf{x}_i}} \\ &= \frac{e^{\beta^T \mathbf{x}_i}}{1 + e^{\beta^T \mathbf{x}_i}} \end{aligned}$$

So,

$$\begin{aligned} \sum_{i=1}^N \frac{-\mathbf{x}_i e^{\beta^T \mathbf{x}_i} \mathbf{x}_i^T}{(1 + e^{\beta^T \mathbf{x}_i})^2} &= \frac{-\mathbf{x}_i^T \mathbf{x}_i (1 - p(\mathbf{x}_i; \beta))}{1 + e^{\beta^T \mathbf{x}_i}} \\ &= -\mathbf{x}_i^T \mathbf{x}_i p(\mathbf{x}_i; \beta) (1 - p(\mathbf{x}_i; \beta)) \end{aligned} \quad (6)$$

Problem 11: Show that the Newton-Raphson update step can be written:

$$\hat{\beta}^{[m+1]} = (\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{W} (\mathbf{X} \hat{\beta}^{[m]} + \mathbf{W}^{-1} (\mathbf{y} - \mathbf{p}^{[m]}))$$

Using the definition of the Newton-Raphson update and the results from Problems 9-10:

$$\begin{aligned} \hat{\beta}^{[m+1]} &= \hat{\beta}^{[m]} + (\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^T (\mathbf{y} - \mathbf{p}^{[m]}) \\ &= (\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{W} \mathbf{X} \hat{\beta}^{[m]} + (\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{W} \mathbf{W}^{-1} (\mathbf{y} - \mathbf{p}^{[m]}) \\ \hat{\beta}^{[m+1]} &= (\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{W} (\mathbf{X} \hat{\beta}^{[m]} + \mathbf{W}^{-1} (\mathbf{y} - \mathbf{p}^{[m]})) \end{aligned} \quad (7)$$

Problem 12: For the data set described in problem 3, program the logistic regression classifier. That is, program the IRLS algorithm to determine β from the training data, then use it to compute the log-likelihood ratio. Use this for classification of the training data and 10,000 points of test data. Plot the classification regions as before. Record the probability of classification error for test and training data on the table.

β was determined using the IRLS algorithm that was optimized to eliminate the need for the \mathbf{W} matrix because it is all zeros except for the trace. The algorithm that I used can be found at [this link](#), or it can be seen in my code below. It takes about 10 iterations of β to have an accurate classifier. The classification regions can be seen in Figure 5, and the probability errors for the test and training data are in Table 2.

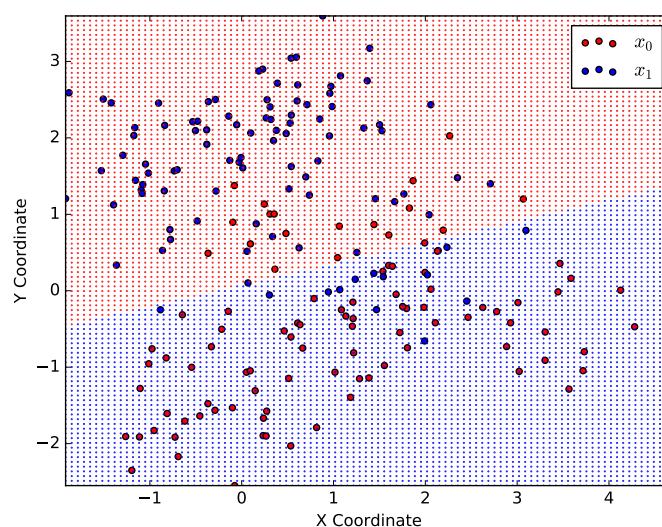


Figure 5: Logistic Classifier

```

1 # Clint Ferrin
2 # Mon Sep 25, 2017
3 # Logistic Regression
4
5 import sys
6 import numpy as np
7 import matplotlib.pyplot as plt
8
9 class data_frame:
10     def __init__(self, data):
11         self.x0 = data[:,0:2].T
12         self.x1 = data[:,2:4].T
13         self.xtot = np.c_[self.x0,self.x1]
14         self.N0 = self.x0.shape[1]
15         self.N1 = self.x1.shape[1]
16         self.N = self.N0 + self.N1
17         self.xlim = [np.min(self.xtot[0,:]),np.max(self.xtot[0,:])]
18         self.ylim = [np.min(self.xtot[1,:]),np.max(self.xtot[1,:])]
19
20 def plot_data(data):
21     fig = plt.figure() # make handle to save plot
22     plt.scatter(data.x0[0,:],data.x0[1:],c='red',label='$x_0$')
23     plt.scatter(data.x1[0,:],data.x1[1:],c='blue',label='$x_1$')
24     plt.xlabel('X Coordinate')
25     plt.ylabel('Y Coordinate')
26     plt.legend()
27
28 def get_phat(data,X,beta):

```

```

29 phat = np.zeros([data.N,1])
30 for i in range(data.N):
31     phat[i] = (np.power(np.e,np.dot(beta.T,X[i,:]).T))/(1 + np.power(np.e,np.dot(beta.T,X[
32         i,:]).T))
33     return phat
34 def gendata2(class_type,N):
35     m0 = np.array(
36         [[-0.132,0.320,1.672,2.230,1.217,-0.819,3.629,0.8210,1.808, 0.1700],
37         [-0.711,-1.726,0.139,1.151,-0.373,-1.573,-0.243,-0.5220,-0.511,0.5330]])
38
39     m1 = np.array(
40         [[-1.169,0.813,-0.859,-0.608,-0.832,2.015,0.173,1.432,0.743,1.0328],
41         [ 2.065,2.441,0.247,1.806,1.286,0.928,1.923,0.1299,1.847,-0.052]])
42
43     x = np.array([[ ],[ ]])
44     for i in range(N):
45         idx = np.random.randint(10)
46         if class_type == 0:
47             m = m0[:,idx]
48         elif class_type == 1:
49             m = m1[:,idx]
50         else:
51             print("not a proper classifier")
52             return 0
53         x = np.c_[x, [[m[0]], [m[1]]] + np.random.randn(2,1)/np.sqrt(5)]
54     return x
55
56 data = np.loadtxt("../data/classasgntrain1.dat",dtype=float)
57 data = data_frame(data)
58 y = np.r_[np.ones([data.N0,1]),np.zeros([data.N1,1])]
59 X = np.r_[np.ones([1,data.xtot.shape[1]]), data.xtot].T
60 beta = np.zeros([3,1])
61
62 for i in range(10):
63     phat = get_phat(data,X,beta)
64     Xhat = X*phat
65     beta = beta + np.dot(np.dot(np.linalg.inv(np.dot(X.T,Xhat)),X.T),(y-phat))
66
67 phat_hard = phat > 0.5
68 num_err = (sum(abs(phat_hard - y)))
69 print("Percent of errors: %.4f"%(float(num_err)/data.N))
70
71 Ntest0 = 10000;
72 Ntest1 = 10000;
73
74 xtest0 = gendata2(0,Ntest0)
75 xtest1 = gendata2(1,Ntest1)
76
77 for i in range(Ntest0):
78     prob = np.dot(beta.T,np.r_[1,xtest0[:,i]])
79     if prob < 0.5:
80         num_err = num_err + 1;
81
82 for i in range(Ntest1):
83     prob = np.dot(beta.T,np.r_[1,xtest1[:,i]])
84     if prob > 0.5:
85         num_err = num_err + 1;
86
87 print("Number of errors: %d"%(num_err))
88 err_rate_linregress_test = float(num_err) / (Ntest0 + Ntest1);
89 print("Percent of errors: %.3f"%(err_rate_linregress_test))
90
91
92 # create colored graph above/below line
93 xpl = np.linspace(data.xlim[0],data.xlim[1], num=100)
94 ypl = np.linspace(data.ylim[0],data.ylim[1], num=100)
95
96 red_pts = np.array([[ ],[ ]])
97 green_pts= np.array([[ ],[ ]])
98
99 for x in xpl:
100     for y in ypl:

```

```

101     prob = np.dot(beta.T,np.r_[1,x,y])
102     if prob > 0.5:
103         green_pts = np.c_[green_pts,[x,y]]
104     else:
105         red_pts = np.c_[red_pts,[x,y]]
106
107 plot_data(data)
108 plt.scatter(green_pts[0,:],green_pts[1:],color='blue',s=0.25)
109 plt.scatter(red_pts[0,:],red_pts[1:],color='red',s=0.25)
110 plt.xlim(data.xlim)
111 plt.ylim(data.ylim)
112 plt.show()

```

6 k-nearest Neighbor Classifier

Problem 13: For the data set described in problem 3, program a k-nearest neighbor function. Make it so that you can change the value of k . Use your k-nearest neighbor function for classification of the training data and 10,000 points of test data for $k = 1$, $k = 5$, and $k = 15$. Comment on the probability of error on the training data when $k = 1$. Plot the classification regions. Record the probability of classification error for test and training data on the table.

As seen in the following code, the k value can be easily manipulated to give the corresponding results seen in Figure 6 through Figure 7.

Table 1 shows how the nearest neighbor program reacts when $k=1$. The program overfits to the training data and reports “0 Errors,” but later has the highest error rate on the test data.

The classification regions can be seen in Figure 6 and Figure 7.

Method	Run-time	Errors in %	
		Training	Test
1-Nearest Neighbor	35.02s	00.0	21.83
5-Nearest Neighbor	37.92s	12.0	20.29
15-Nearest Neighbor	36.47s	16.0	19.25

Table 1: Nearest Neighbor Performance Comparison

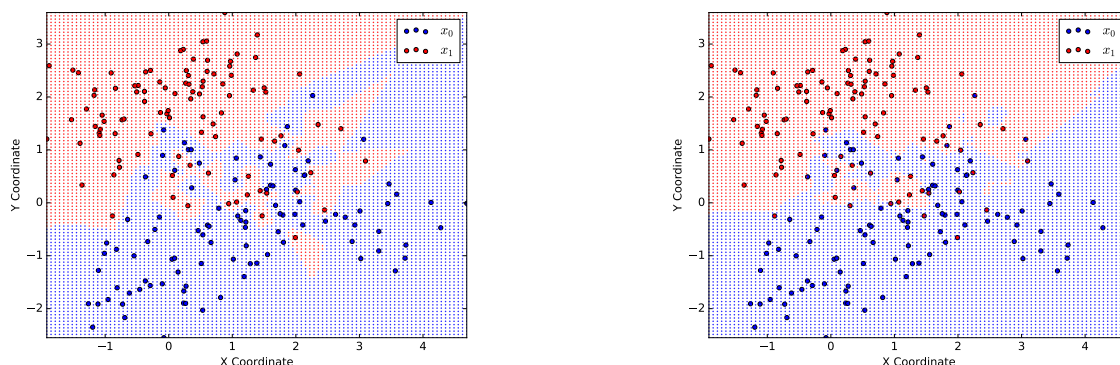


Figure 6: Left: $k=1$ Nearest Neighbor. Right: $k=5$ Nearest Neighbor

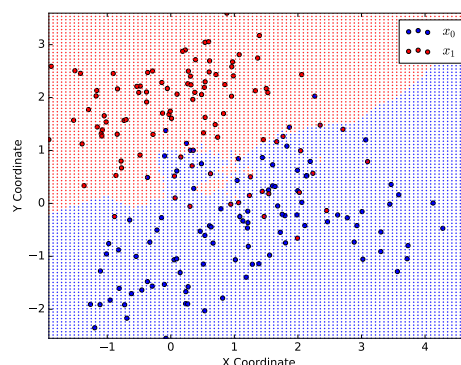


Figure 7: k=15 Nearest Neighbor

```

1 # Clint Ferrin
2 # Mon Sep 25, 2017
3 # K Nearest Neighbor
4
5 import sys
6 import numpy as np
7 import matplotlib.pyplot as plt
8
9 def gendata2(class_type,N):
10     m0 = np.array(
11         [[-0.132,0.320,1.672,2.230,1.217,-0.819,3.629,0.8210,1.808, 0.1700],
12          [-0.711,-1.726,0.139,1.151,-0.373,-1.573,-0.243,-0.5220,-0.511,0.5330]])
13
14     m1 = np.array(
15         [[-1.169,0.813,-0.859,-0.608,-0.832,2.015,0.173,1.432,0.743,1.0328],
16          [ 2.065,2.441,0.247,1.806,1.286,0.928,1.923,0.1299,1.847,-0.052]])
17
18     x = np.array([[ ],[ ]])
19     for i in range(N):
20         idx = np.random.randint(10)
21         if class_type == 0:
22             m = m0[:,idx]
23         elif class_type == 1:
24             m = m1[:,idx]
25         else:
26             print("not a proper classifier")
27             return 0
28         x = np.c_[x, [[m[0]], [m[1]]] + np.random.randn(2,1)/np.sqrt(5)]
29     return x
30
31 class data_frame:
32     def __init__(self, data):
33         self.x0 = data[:,0:2]
34         self.x1 = data[:,2:4]
35         self.xtot = np.r_[self.x0,self.x1]
36         self.N0 = self.x0.shape[0]
37         self.N1 = self.x1.shape[0]
38         self.N = self.N0 + self.N1
39         self.xlim = [np.min(self.xtot[:,0]),np.max(self.xtot[:,0])]
40         self.ylim = [np.min(self.xtot[:,1]),np.max(self.xtot[:,1])]
41
42     def plot_data(data):
43         fig = plt.figure() # make handle to save plot
44         plt.scatter(data.x0[:,0],data.x0[:,1],c='blue',label='$x_0$')
45         plt.scatter(data.x1[:,0],data.x1[:,1],c='red',label='$x_1$')
46         plt.xlabel('X Coordinate')
47         plt.ylabel('Y Coordinate')
48         plt.legend()
49
50     def get_distance_matrix(X,point):
51         dist_mat = np.empty([X.shape[0],2])
52         for i in range(X.shape[0]):

```

```

53     dist_mat[i] = [np.linalg.norm(point - X[i,0:2]),X[i,2]]
54     return dist_mat
55
56 def find_class(dist_mat,k):
57     neighbors = dist_mat[0:k]
58     for i in range(k,X.shape[0]):
59         if np.max(neighbors[:,0]) > dist_mat[i,0]:
60             neighbors[np.argmax(neighbors[:,0])] = dist_mat[i]
61     prob = sum(neighbors[:,1])/k
62     if prob > 0.5:
63         class_type = 1
64     else:
65         class_type = 0
66     return class_type
67
68 data = np.loadtxt("../data/classasgntrain1.dat",dtype=float)
69 data = data_frame(data)
70 k = 5
71 y = np.r_[np.zeros([data.N0,1]),np.ones([data.N1,1])]
72 X = np.c_[data.xtot,y]
73 yhat = np.empty([data.N,1])
74
75 for i in range(data.N):
76     dist_mat = get_distance_matrix(X,data.xtot[i])
77     yhat[i] = find_class(dist_mat,k)
78
79 num_err = (sum(abs(yhat - y)))
80 print("Percent of errors: %.4f"%(float(num_err)/data.N))
81
82 Ntest0 = 10000;
83 Ntest1 = 10000;
84
85 xtest0 = gendata2(0,Ntest0)
86 xtest1 = gendata2(1,Ntest1)
87 num_err = 0
88
89 for i in range(Ntest0):
90     dist_mat = get_distance_matrix(X,xtest0[:,i])
91     class_type = find_class(dist_mat,k)
92     if class_type == 1:
93         num_err = num_err + 1
94
95 for i in range(Ntest1):
96     dist_mat = get_distance_matrix(X,xtest1[:,i])
97     class_type = find_class(dist_mat,k)
98     if class_type == 0:
99         num_err = num_err + 1
100
101 print("Number of errors: %d"%(num_err))
102 err_rate_linregress_test = float(num_err) / (Ntest0 + Ntest1);
103 print("Percent of errors: %.4f"%(err_rate_linregress_test))
104
105 # create colored graph above/below line
106 xpl = np.linspace(data.xlim[0],data.xlim[1], num=100)
107 ypl = np.linspace(data.ylim[0],data.ylim[1], num=100)
108
109 red_pts = np.array([],[])
110 blue_pts= np.array([],[])
111
112 for x in xpl:
113     for y in ypl:
114         dist_mat = get_distance_matrix(X,[x,y])
115         class_type = find_class(dist_mat,k)
116         if class_type == 0:
117             blue_pts = np.c_[blue_pts,[x,y]]
118         else:
119             red_pts = np.c_[red_pts,[x,y]]
120
121 plot_data(data)
122 plt.scatter(blue_pts[0,:],blue_pts[1,:],color='blue',s=0.25)
123 plt.scatter(red_pts[0,:],red_pts[1,:],color='red',s=0.25)
124 plt.xlim(data.xlim)
125 plt.ylim(data.ylim)

```

```
126 plt.show()
```

7 Naive Bayes Classifier

Problem 14: The true density of the data is a Gaussian mixture represented by $f_X(x) = p_0N(x, \mu_0, \sigma_0^2) + p_1N(x, \mu_1, \sigma_1^2)$. Using this equation, with given values of μ and σ , make a plot of the true density, the sample empirical distribution, the kernel functions, and the estimated density. Try different values of λ .

As seen in Figure 8, the Parzen distribution was calculated using an empirical distribution and adding the corresponding Gaussian distributions. When $\lambda = 0.8$ was chosen, the best results can be seen. If λ is too small, the data begins to look corrupt, and if it was much larger than $\lambda = 0.8$, it became very elongated as seen in Figure 9.

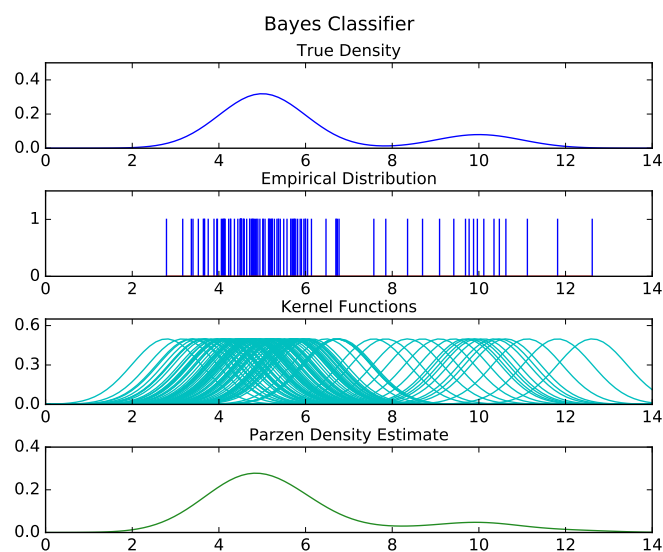


Figure 8: Calculating Parzen Distribution

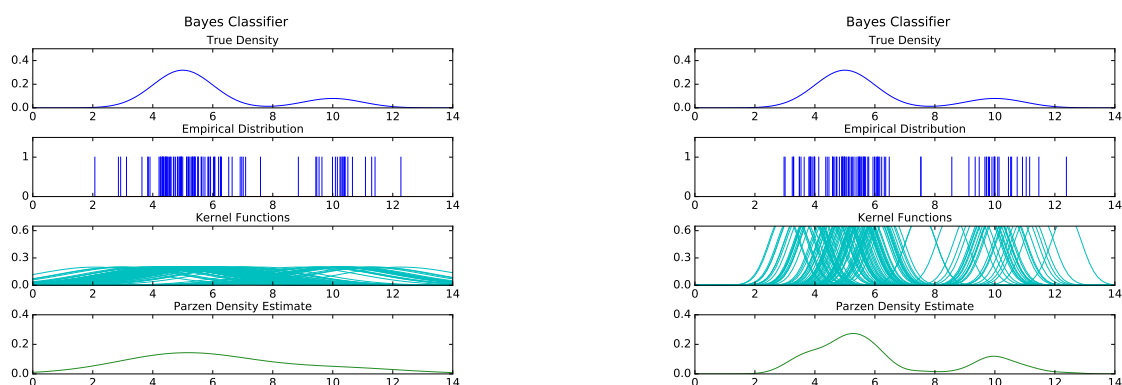


Figure 9: Left: $\lambda = 2$. Right: $\lambda = 0.5$

```
1 # Clint Ferrin
2 # Mon Sep 25, 2017
```

```

3 # Parzen Density Estimate
4
5 import numpy as np
6 import matplotlib.pyplot as plt
7 import matplotlib.mlab as mlab
8
9 p_0 = 0.8
10 p_1 = 0.2
11 mu_0 = 5
12 mu_1 = 10
13 sig_0 = 1
14 sig_1 = 1
15
16 lam = .5
17
18 f, ax = plt.subplots(4,1)
19 f.subplots_adjust(hspace=.5)
20 f.suptitle("Bayes Classifier", fontsize=14)
21
22 ax[0].set_title("True Density", fontsize=12)
23 ax[0].set_ylim(0, 0.5)
24 ax[0].yaxis.set_ticks(np.arange(0,0.6,0.2))
25 ax[1].set_title("Empirical Distribution", fontsize=12)
26 ax[1].set_ylim(0, 1.5)
27 ax[1].yaxis.set_ticks(np.arange(0,2,1))
28 ax[2].set_title("Kernel Functions", fontsize=12)
29 ax[2].set_ylim(0, .65)
30 ax[2].yaxis.set_ticks(np.arange(0,.64,.3))
31 ax[3].set_title("Parzen Density Estimate", fontsize=12)
32 ax[3].set_ylim(0, 0.35)
33 ax[3].yaxis.set_ticks(np.arange(0,0.6,0.2))
34 for i in range(4):
35     ax[i].set_xlim(0, 14)
36
37 pts = 100
38 x = np.linspace(0,14, pts)
39 pdf = p_0*mlab.normpdf(x, mu_0, sig_0)+p_1*mlab.normpdf(x, mu_1, sig_1)
40 ax[0].plot(x,pdf)
41 cdf = np.cumsum(pdf)/sum(pdf)
42
43
44
45 emperical = np.interp(np.random.rand(pts,1),cdf,x)
46 emperical = np.sort(emperical,axis=None)
47
48 ax[1].stem(emperical,np.ones(pts),'b',markerfmt=' ')
49 ax[1].set_xlim(0,14)
50
51 kernel = np.empty([emperical.shape[0],pts])
52 for i in range(emperical.shape[0]):
53     kernel[i] = mlab.normpdf(x, emperical[i], lam)
54     ax[2].plot(x,kernel[i],color='c')
55
56 parzen = np.empty(100)
57 for i in range(pts):
58     parzen[i] = sum(kernel[:,i])/pts
59
60 ax[3].plot(x,parzen,color='forestgreen')
61
62 plt.show()

```

Problem 15: Using the training data in `classassgntrain1.dat` to estimate the densities, apply the Naive Bayes estimator to our data set. Plot the classification regions. Record the probability of classification error for test and training data on the table.

The data with the classification regions are plotted in Figure 11 and the code is typed out below the graph. The probability of classification error for test and training data are recorded on Table 2.

To estimate the density functions, I simply found the probability that an x value would be chosen in

class0 and a y value in class0. Their pdf were calculated using the Parzen technique, and their distribution can be seen in Figure 10.

Because the Bayes method assumes the probability is independent, the two can simply be multiplied together to create a probability that a given point is in class 0 or 1. Figure 11 shows how the division line ended up, and the classification error can be seen in Table 2.

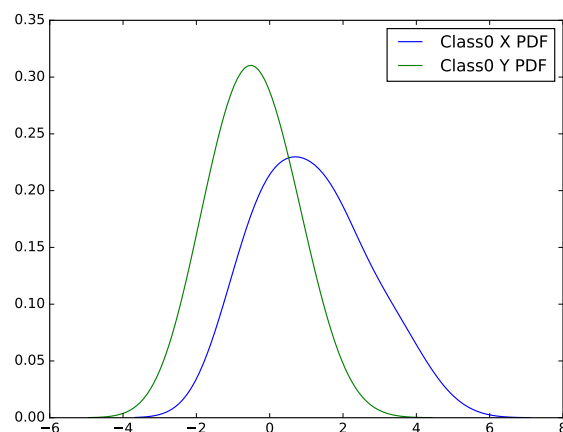


Figure 10: Bayes 1D plot of X and Y

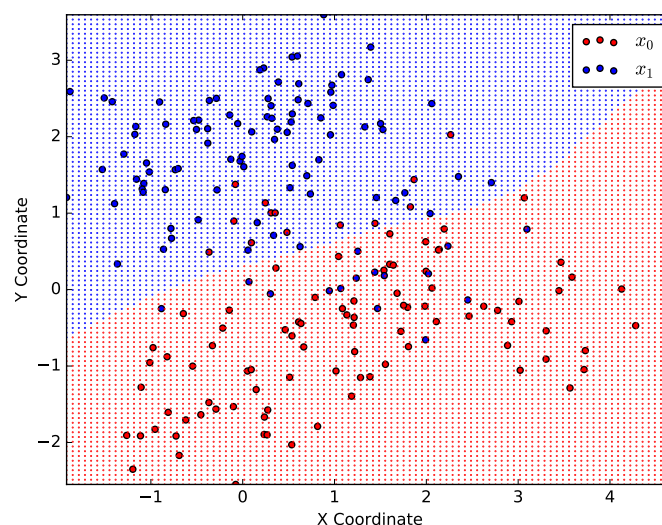


Figure 11: Naive Bayes Parzen Graph

```

1 # Clint Ferrin
2 # Mon Sep 25, 2017
3 # Bayes Naive Classifier
4
5 import numpy as np
6 import matplotlib.pyplot as plt
7 import matplotlib.mlab as mlab
8
9 class data_frame:
10     def __init__(self, data0, data1):
11         self.x0 = data0

```

```

12     self.x1 = data1
13     self.xtot = np.c_[self.x0,self.x1]
14     self.N0 = self.x0.shape[1]
15     self.N1 = self.x1.shape[1]
16     self.N = self.N0 + self.N1
17     self.xlim = [np.min(self.xtot[0,:]),np.max(self.xtot[0,:])]
18     self.ylim = [np.min(self.xtot[1,:]),np.max(self.xtot[1,:])]
19
20 def gendata2(class_type,N):
21     m0 = np.array(
22         [[-0.132,0.320,1.672,2.230,1.217,-0.819,3.629,0.8210,1.808, 0.1700],
23          [-0.711,-1.726,0.139,1.151,-0.373,-1.573,-0.243,-0.5220,-0.511,0.5330]])
24
25     m1 = np.array(
26         [[-1.169,0.813,-0.859,-0.608,-0.832,2.015,0.173,1.432,0.743,1.0328],
27          [ 2.065,2.441,0.247,1.806,1.286,0.928,1.923,0.1299,1.847,-0.052]])
28
29     x = np.array([[ ],[ ]])
30     for i in range(N):
31         idx = np.random.randint(10)
32         if class_type == 0:
33             m = m0[:,idx]
34         elif class_type == 1:
35             m = m1[:,idx]
36         else:
37             print("not a proper classifier")
38             return 0
39         x = np.c_[x, [[m[0]], [m[1]]] + np.random.randn(2,1)/np.sqrt(5)]
40     return x
41
42 def gen_test_df(num0,num1):
43
44     return data_frame
45
46 def get_parzen(data,pts,lam):
47     x = np.linspace(min(data)-3*lam,max(data)+3*lam, pts)
48     kernel = np.empty([data.size,pts])
49     for i in range(data.size):
50         kernel[i] = mlab.normpdf(x, data[i], lam)
51         # plt.plot(x,kernel[i],color='b')
52
53     parzen = np.empty(data.size)
54     for i in range(data.size):
55         parzen[i] = sum(kernel[:,i])/pts
56
57     return x, parzen
58
59 def class_parzen(data0,pts,lam):
60     x0,parzen_x0 = get_parzen(data0[0,:],pts[0],lam)
61     y0,parzen_y0= get_parzen(data0[1,:],pts[1],lam)
62     return [x0,y0],[parzen_x0,parzen_y0]
63
64 def prob2d(point,linspace0,parzen0):
65     prob_x = np.interp(point[0],linspace0[0],parzen0[0])
66     prob_y = np.interp(point[1],linspace0[1],parzen0[1])
67     # print("prob x: %f, prob y: %f"%(prob_x, prob_y))
68     return prob_x*prob_y
69
70 def run_bayes_test(data_tot,linspace,parzen):
71     y = np.r_[np.zeros([data_tot.shape[1],1]),np.ones([data_tot.shape[1],1])]
72     y_hat = np.zeros([data_tot.shape[1],1])
73
74     for i in range(data_tot.shape[1]):
75         prob0 = prob2d(data_tot[:,i],linspace[0],parzen[0])
76         prob1 = prob2d(data_tot[:,i],linspace[1],parzen[0])
77         if prob1 > prob0:
78             y_hat[i] = 1
79
80     return y_hat
81
82 def plot_data(x0,x1):
83     fig = plt.figure() # make handle to save plot
84     plt.scatter(x0[0,:],x0[1:],c='red',label='$x_0$')

```

```

85 plt.scatter(x1[0,:],x1[1,:],c='blue',label='$x_1$')
86 plt.xlabel('X Coordinate')
87 plt.ylabel('Y Coordinate')
88 plt.legend()
89
90
91 data = np.loadtxt("../data/classasgntrain1.dat",dtype=float)
92 x0 = data[:,0:2].T
93 x1 = data[:,2:4].T
94 data = data_frame(x0,x1)
95
96 pts = [data.N0,data.N1]
97 lam = 0.8
98
99 linspace0,parzen0 = class_parzen(data.x0,pts,lam)
100 linspace1,parzen1 = class_parzen(data.x1,pts,lam)
101
102 plt.plot(linspace0[0],parzen0[0],label='Class0 X PDF')
103 plt.plot(linspace0[1],parzen0[1],label='Class0 Y PDF')
104 plt.legend()
105
106 print(np.array(parzen0).shape)
107
108 linspace = np.array([linspace0,linspace1])
109 parzen = np.array([parzen0,parzen1])
110
111 y = np.r_[np.zeros([data.N1,1]),np.ones([data.N0,1])]
112 y_hat = run_bayes_test(data.xtot,linspace,parzen)
113
114 num_err = sum(abs(y_hat - y))
115 print("Percent of errors: %.4f"%(float(num_err)/data.N))
116
117
118 xtest0 = gendata2(0,10000)
119 xtest1 = gendata2(1,10000)
120 test_data = data_frame(xtest0,xtest1)
121 y = np.r_[np.zeros([test_data.N1,1]),np.ones([test_data.N0,1])]
122
123 y_hat = run_bayes_test(test_data.xtot,linspace,parzen)
124
125 num_err = sum(abs(y_hat - y))
126 print("Percent of errors: %.4f"%(float(num_err)/test_data.N))
127
128 xpl = np.linspace(data.xlim[0],data.xlim[1], num=100)
129 ypl = np.linspace(data.ylim[0],data.ylim[1], num=100)
130
131 red_pts = np.array([],[])
132 blue_pts= np.array([],[])
133 for x in xpl:
134     for y in ypl:
135         prob0 = prob2d([x,y],linspace[0],parzen[0])
136         prob1 = prob2d([x,y],linspace[1],parzen[0])
137         if prob1 > prob0:
138             blue_pts = np.c_[blue_pts,[x,y]]
139         else:
140             red_pts = np.c_[red_pts,[x,y]]
141
142 plot_data(x0,x1)
143 plt.scatter(blue_pts[0,:],blue_pts[1,:],color='blue',s=0.25)
144 plt.scatter(red_pts[0,:],red_pts[1,:],color='red',s=0.25)
145 plt.xlim(data.xlim)
146 plt.ylim(data.ylim)
147 plt.show()

```

8 Optimal Bayes Classifier

Problem 16: For the data set described in problem 3, determine the Bayes error rate on the training data and 10,000 points of test data. Record the probability of classification error for test and training

data on the table. Plot the classification regions.

The classification regions for the Optimal Bayes Classifier can be seen in Figure 12. It performed the best on the test data above all other data sets because it had the actual data model embedded into the classifier. See Table 2 for specific details.

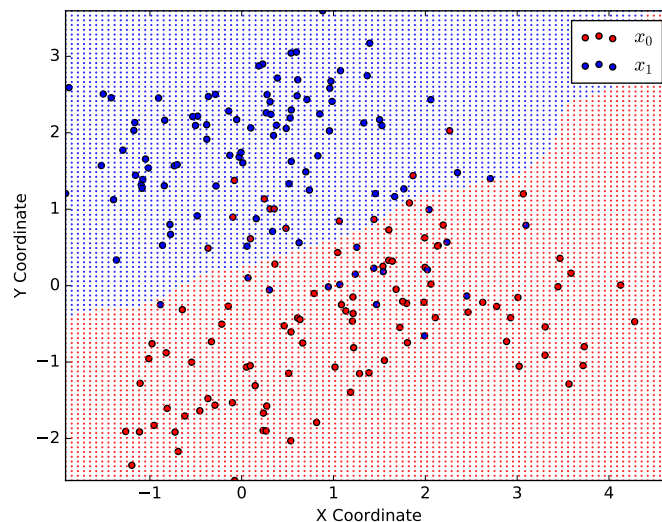


Figure 12: Optimal Bayes Classifier

```

1 # Clint Ferrin
2 # Mon Sep 25, 2017
3 # Bayes Optimal Classifier
4
5 import numpy as np
6 import matplotlib.pyplot as plt
7 import matplotlib.mlab as mlab
8
9 class data_frame:
10     def __init__(self, data0, data1):
11         self.x0 = data0
12         self.x1 = data1
13         self.xtot = np.c_[self.x0, self.x1]
14         self.N0 = self.x0.shape[1]
15         self.N1 = self.x1.shape[1]
16         self.N = self.N0 + self.N1
17         self.xlim = [np.min(self.xtot[0, :]), np.max(self.xtot[0, :])]
18         self.ylim = [np.min(self.xtot[1, :]), np.max(self.xtot[1, :])]
19
20 def gendata2(class_type, N):
21     m0 = np.array(
22         [[-0.132, 0.320, 1.672, 2.230, 1.217, -0.819, 3.629, 0.8210, 1.808, 0.1700],
23          [-0.711, -1.726, 0.139, 1.151, -0.373, -1.573, -0.243, -0.5220, -0.511, 0.5330]])
24
25     m1 = np.array(
26         [[-1.169, 0.813, -0.859, -0.608, -0.832, 2.015, 0.173, 1.432, 0.743, 1.0328],
27          [ 2.065, 2.441, 0.247, 1.806, 1.286, 0.928, 1.923, 0.1299, 1.847, -0.052]])
28
29     x = np.array([[[]], [[]]])
30     for i in range(N):
31         idx = np.random.randint(10)
32         if class_type == 0:
33             m = m0[:, idx]
34         elif class_type == 1:
35             m = m1[:, idx]
36         else:
37             print("not a proper classifier")

```

```

38         return 0
39         x = np.c_[x, [[m[0]], [m[1]]] + np.random.randn(2,1)/np.sqrt(5)]
40     return x
41
42 def get_parzen(data,pts,lam):
43     x = np.linspace(min(data)-3*lam,max(data)+3*lam, pts)
44     kernel = np.empty([data.size,pts])
45     for i in range(data.size):
46         kernel[i] = mlab.normpdf(x, data[i], lam)
47         # plt.plot(x,kernel[i],color='b')
48
49     parzen = np.empty(data.size)
50     for i in range(data.size):
51         parzen[i] = sum(kernel[:,i])/pts
52
53     return x, parzen
54
55 def class_parzen(data0,pts,lam):
56     x0,parzen_x0 = get_parzen(data0[0,:],pts[0],lam)
57     y0,parzen_y0= get_parzen(data0[1,:],pts[1],lam)
58     return [x0,y0],[parzen_x0,parzen_y0]
59
60 def prob2d(point,linspace0,parzen0):
61     prob_x = np.interp(point[0],linspace0[0],parzen0[0])
62     prob_y = np.interp(point[1],linspace0[1],parzen0[1])
63     # print("prob x: %f, prob y: %f"%(prob_x, prob_y))
64     return prob_x*prob_y
65
66 def run_bayes_test(data_tot,linspace,parzen):
67     y = np.r_[np.zeros([data_tot.shape[1],1]),np.ones([data_tot.shape[1],1])]
68     y_hat = np.zeros([data_tot.shape[1],1])
69
70     for i in range(data_tot.shape[1]):
71         prob0 = prob2d(data_tot[:,i],linspace[0],parzen[0])
72         prob1 = prob2d(data_tot[:,i],linspace[1],parzen[0])
73         if prob1 > prob0:
74             y_hat[i] = 1
75
76     return y_hat
77
78 def plotData(data):
79     fig = plt.figure() # make handle to save plot
80     plt.scatter(data.x0[0,:],data.x0[1:],c='red',label='$x_0$')
81     plt.scatter(data.x1[0,:],data.x1[1:],c='blue',label='$x_1$')
82     plt.xlabel('X Coordinate')
83     plt.ylabel('Y Coordinate')
84     plt.legend()
85
86 data = np.loadtxt("../data/classasgntrain1.dat",dtype=float)
87 x0 = data[:,0:2].T
88 x1 = data[:,2:4].T
89 data = data_frame(x0,x1)
90
91 m0 = np.array(
92     [[-0.132,0.320,1.672,2.230,1.217,-0.819,3.629,0.8210,1.808, 0.1700],
93      [-0.711,-1.726,0.139,1.151,-0.373,-1.573,-0.243,-0.5220,-0.511,0.5330]]) .T
94
95 m1 = np.array(
96     [[-1.169,0.813,-0.859,-0.608,-0.832,2.015,0.173,1.432,0.743,1.0328],
97      [ 2.065,2.441,0.247,1.806,1.286,0.928,1.923,0.1299,1.847,-0.052]]) .T
98
99 pts = [m0.shape[0],m1.shape[0]]
100 lam = 0.8
101
102 linspace0,parzen0 = class_parzen(m0.T,pts,lam)
103 linspace1,parzen1 = class_parzen(m1.T,pts,lam)
104
105
106
107 linspace = np.array([linspace0,linspace1])
108 parzen = np.array([parzen0,parzen1])
109
110 y = np.r_[np.zeros([data.N0,1]),np.ones([data.N1,1])]

```

```

111 y_hat = run_bayes_test(data.xtot, linspace, parzen)
112
113 num_err = sum(abs(y_hat - y))
114 print("Percent of errors: %.4f"%(float(num_err)/data.N))
115
116
117 xtest0 = gendata2(0,10000)
118 xtest1 = gendata2(1,10000)
119 test_data = data_frame(xtest0,xtest1)
120 y = np.r_[np.zeros([test_data.N1,1]), np.ones([test_data.N0,1])]
121
122 y_hat = run_bayes_test(test_data.xtot, linspace, parzen)
123
124 num_err = sum(abs(y_hat - y))
125 print("Percent of errors: %.4f"%(float(num_err)/test_data.N))
126
127 xpl = np.linspace(data.xlim[0],data.xlim[1], num=100)
128 ypl = np.linspace(data.ylim[0],data.ylim[1], num=100)
129
130 red_pts = np.array([],[])
131 blue_pts= np.array([],[])
132 for x in xpl:
133     for y in ypl:
134         prob0 = prob2d([x,y],linspace[0],parzen[0])
135         prob1 = prob2d([x,y],linspace[1],parzen[0])
136         if prob1 > prob0:
137             blue_pts = np.c_[blue_pts,[x,y]]
138         else:
139             red_pts = np.c_[red_pts,[x,y]]
140
141 plotData(data)
142 plt.scatter(blue_pts[0,:],blue_pts[1:],color='blue',s=0.25)
143 plt.scatter(red_pts[0,:],red_pts[1:],color='red',s=0.25)
144 plt.xlim(data.xlim)
145 plt.ylim(data.ylim)
146 plt.show()

```

9 Discussion

Problem 17: Discuss the relative merits of the different classification algorithms on this source of data. Comment on differences in performance between training and test data. Also, comment on operating speed of the classification algorithms (after they have been trained). Summarize what you have learned. Turn in with this assignment your table of results, plots of data and classification regions, and listings of your PYTHON code.

The highest performing data classifier was the Bayes Optimal classifier. It consistently performed with fewer errors when I increased the sample size of data to 10000 data points for each class. In real world applications, however, it is often not feasible to have such a precise model.

The Linear regression model was similar in run-time to the optimal classifier, and it performed well, regardless of its seeming simplicity.

The 15-nearest neighbor performed almost as well as the Bayes Optimal classifier, but it took considerably longer. There are ways to optimize the code, but the method requires the program to store all of the training data and process it continually—an obvious drawback. As seen in Table 2, most of the programs were within about a 20 percent error rate with the exception of the 1-Nearest Neighbor. They it had a 0 percent error on the training data, it was too “fitted” to the original data to yield correct results.

Method	Run-time	Errors in %	
		Training	Test
Linear Regression	1.23s	14.5	20.49
Quadratic Regression	1.70s	14.5	20.44
Linear Discriminant Analysis	2.49s	15.0	19.98
Quadratic Discriminant Analysis	3.26s	14.5	20.23
Logistic Regression	2.00s	14.0	20.00
1-Nearest Neighbor	35.02s	00.0	21.83
5-Nearest Neighbor	37.92s	12.0	20.29
15-Nearest Neighbor	36.47s	16.0	19.25
Bayes Naive	1.22s	14.0	20.04
Bayes Optimal Classifier	0.20s	14.0	19.14

Table 2: Binary Classifier Performance Comparison