

Data-Driven Classifiers

Neural Networks: ECE 5930

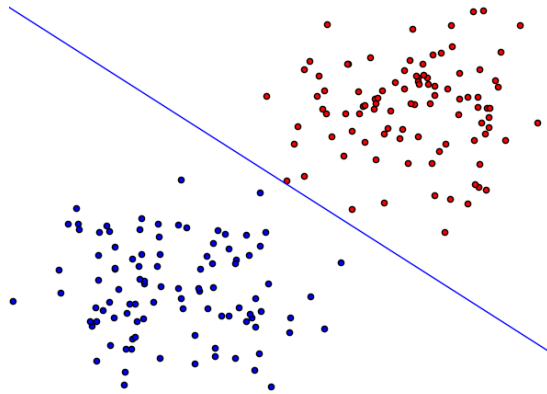


Figure: Linear Data Classifier

Clint Ferrin
Utah State University
Mon Sep 25, 2017

Contents

Overview	1
Linear Regression	1
Problem 1:	1
Problem 2:	1
Problem 3:	2
Quadratic Regression	4
Problem 4:	4
Problem 5:	5
Linear and Quadratic Discriminant Analysis	6
Linear Logistic Regression	6
Problem 8:	6
Problem 9:	6
Problem 10:	7
k-nearest Neighbor Classifier	8
Naive Bayes Classifier	8
Optimal Bayes Classifier	8
Discussion	8

List of Figures**List of Tables**

Overview

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 to find the minimized 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 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 $\hat{\mathbf{B}}$

$$\begin{aligned} \frac{d}{d\hat{\mathbf{B}}} \left\| \sqrt{\text{Tr}(\mathbf{X}^T \hat{\mathbf{B}}^T \mathbf{X}\hat{\mathbf{B}} - 2\mathbf{X}^T \hat{\mathbf{B}}^T \mathbf{Y} + \mathbf{Y}^T \mathbf{Y})} \right\|^2 &= 0 \\ 2\mathbf{X}^T \mathbf{X}\hat{\mathbf{B}} - 2\mathbf{X}^T \mathbf{Y} &= 0 \\ \mathbf{X}^T \mathbf{X}\hat{\mathbf{B}} &= \mathbf{X}^T \mathbf{Y} \\ \hat{\mathbf{B}} &= (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y} \end{aligned} \tag{2}$$

Problem 3: Re-write the function `gendata2.m` into Python. Using the 100 points of training data in `classasgntrain1.dat`, write PYTHON code to train the coefficient matrix $\hat{\beta}$.

```

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

```

```

71 num_err = 0;
72
73 for i in range(Ntest0):
74     yhat = np.dot(np.r_[1,xtest0[:,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]],Bhat)
80     if yhat[1] < yhat[0]:
81         num_err = num_err + 1;
82
83 print("Number of errors: %d"%(num_err))
84 err_rate_linregress_test = float(num_err) / (Ntest0 + Ntest1);
85 print("Percent of errors: %.2f"%(err_rate_linregress_test))
86
87
88 # find max and min of sets
89 x_tot = np.r_[x0[0,:],x1[0,:]]
90 y_tot = np.r_[x0[1,:],x1[1,:]]
91 xlim = [np.min(x_tot),np.max(x_tot)]
92 ylim = [np.min(y_tot),np.max(y_tot)]
93
94 # find x,y coordinate of separating line
95 x_cor_lin = [xlim[0],xlim[1]]
96 y_cor_lin = [
97     (Bhat[0,0]-Bhat[0,1]+(Bhat[1,0]-Bhat[1,1])*xlim[0])
98     / (Bhat[2,1]-Bhat[2,0]),
99
100    (Bhat[0,0]-Bhat[0,1]+(Bhat[1,0]-Bhat[1,1])*xlim[1])
101    / (Bhat[2,1]-Bhat[2,0])
102 ]
103
104 # create colored graph above/below line
105 xp1 = np.linspace(xlim[0],xlim[1], num=100)
106 yp1 = np.linspace(ylim[0],ylim[1], num=100)
107
108 red_pts = np.array([],[])
109 green_pts= np.array([],[])
110
111 for x in xp1:
112     for y in yp1:
113         yhat = np.dot(np.r_[1,x,y],Bhat)
114         if yhat[1] > yhat[0]:
115             green_pts = np.c_[green_pts,[x,y]]
116         else:
117             red_pts = np.c_[red_pts,[x,y]]
118
119 plotData(x0,x1)
120 plt.plot(x_cor_lin,y_cor_lin,color='black')
121 plt.scatter(green_pts[0,:],green_pts[1:],color='blue',s=0.25)
122 plt.scatter(red_pts[0,:],red_pts[1:],color='red',s=0.25)
123 plt.xlim(xlim)
124 plt.ylim(ylim)
125 plt.show()

```

Method	Errors in %	
	Training	Test
Linear Regression	14.0	20
Quadratic Regression	14.5	20
Linear Discriminant Analysis	15.0	20
Quadratic Discriminant Analysis	14.5	20
Logistic Regression	14.0	20
1-Nearest Neighbor	00.0	22
5-Nearest Neighbor	12.0	20
15-Nearest Neighbor	16.0	19
Bayes Naive		

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.

```

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

```

```

65 xtest0 = gendata2(0,Ntest0)
66 xtest1 = gendata2(1,Ntest1)
67 num_err = 0;
68
69 for i in range(Ntest0):
70     yhat = np.dot(np.r_[1,xtest0[:,i],xtest0[0,i]*xtest0[0,i], xtest0[0,i]*xtest0[1,i],xtest0
71         [1,i]*xtest0[1,i]],Bhat)
72     if yhat[1] > yhat[0]:
73         num_err = num_err + 1;
74
75 for i in range(Ntest1):
76     yhat = np.dot(np.r_[1,xtest1[:,i],xtest1[0,i]*xtest1[0,i], xtest1[0,i]*xtest1[1,i],xtest1
77         [1,i]*xtest1[1,i]],Bhat)
78     if yhat[1] < yhat[0]:
79         num_err = num_err + 1;
80
81 print("Number of errors: %d"%(num_err))
82
83 err_rate_linregress_test = float(num_err) / (Ntest0 + Ntest1);
84 print(err_rate_linregress_test)
85
86 # find max and min of sets
87 x_tot = np.r_[x0[0,:],x1[0,:]]
88 y_tot = np.r_[x0[1,:],x1[1,:]]
89 xlim = [np.min(x_tot),np.max(x_tot)]
90 ylim = [np.min(y_tot),np.max(y_tot)]
91
92 # create colored graph above/below line
93 xpl = np.linspace(xlim[0],xlim[1], num=100)
94 ypl = np.linspace(ylim[0],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         yhat = np.dot(np.r_[1,x,y,x*x,x*y,y*y],Bhat)
102         if yhat[1] > yhat[0]:
103             green_pts = np.c_[green_pts,[x,y]]
104         else:
105             red_pts = np.c_[red_pts,[x,y]]
106
107 plt.scatter(green_pts[0:],green_pts[1:],color='blue',s=0.25)
108 plt.scatter(red_pts[0:],red_pts[1:],color='red',s=0.25)
109 plt.xlim(xlim)
110 plt.ylim(ylim)
111 plt.show()

```

Problem 5: Show that (2) is true. In particular, make sure you understand what is meant by up to a constant which does not depend on the class”

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

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:

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

Now taking the log of the equation gives us:

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

Linear and Quadratic Discriminant Analysis

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

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

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:

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

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.

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

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,

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

k-nearest Neighbor Classifier

Naive Bayes Classifier

Optimal Bayes Classifier

Discussion