Data-Driven Classifiers

Neural Networks: ECE 5930

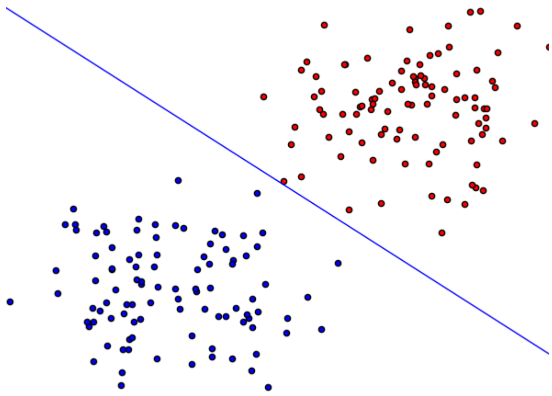

Figure: Linear Data Classifier

Clint Ferrin
Utah State University
Mon Sep 25, 2017

Contents

List of Figures

List of Tables

Overview

Linear Regression

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

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

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

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

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

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

Given that the Frobenius Norm for real numbers is:

$$\sqrt{Tr(\boldsymbol{A}\boldsymbol{A}^T)}$$

Then the Frobenius Norm of the problem statement is:

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

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

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

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

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 $\hat{\beta}$.

```
import sys
import numpy as np
import matplotlib.pyplot as plt

def gendata2(class_type,N):
    m0 = np.array(
         [[-0.132,0.320,1.672,2.230,1.217,-0.819,3.629,0.8210,1.808, 0.1700],
          [-0.711,-1.726,0.139,1.151,-0.373,-1.573,-0.243,-0.5220,-0.511,0.5330]])

    m1 = np.array(
          [[-1.169,0.813,-0.859,-0.608,-0.832,2.015,0.173,1.432,0.743,1.0328],
          [ 2.065,2.441,0.247,1.806,1.286,0.928,1.923,0.1299,1.847,-0.052]])

    x = np.array([[],[]])
    for i in range(N):
        idx = np.random.randint(10)
        if class_type == 0:
            m = m0[:,idx]
        elif class_type == 1:
            m = m1[:,idx]
        else:
            print("not a proper classifier")
            return 0
        x = np.c_[x, [[m[0]],[m[1]]] + np.random.randn(2,1)/np.sqrt(5)]
    return x

def plotData(x0,x1):
    fig = plt.figure() # make handle to save plot
    plt.scatter(x0[0,:],x0[1,:],c='red',label='$x_0$')
    plt.scatter(x1[0,:],x1[1,:],c='blue',label='$y_0$')
    plt.xlabel('X Coordinate')
    plt.ylabel('Y Coordinate')
    plt.legend()


data = np.loadtxt("../data/classasgntrain1.dat",dtype=float)
x0 = data[:,0:2].T
x1 = data[:,2:4].T
data_tot = np.c_[x0,x1]

N0 = x0.shape[1]
N1 = x1.shape[1];
N = N0 + N1

# linear regression classifier
X = np.r_[np.c_[np.ones((N0,1)),x0.T],
          np.c_[np.ones((N1,1)),x1.T]]

Y = np.r_[np.c_[np.ones((N0,1)),np.zeros((N0,1))],
          np.c_[np.zeros((N1,1)),np.ones((N1,1))]]

# find parameter matrix
Bhat = np.dot(np.linalg.inv(np.dot(X.T,X)),np.dot(X.T,Y))

# find approximate response
Yhat = np.dot(X,Bhat)
Yhathard = Yhat > 0.5

num_err = sum(sum(abs(Yhathard - Y)))/2
print("Number of errors: %d"%(num_err))

Ntest0 = 10000;
Ntest1 = 10000;

err_rate_linregress_train = float(num_err) / N
print("Percent of errors: %.2f"%(err_rate_linregress_train))

# generate the test data for class 0
xtest0 = gendata2(0,Ntest0)
xtest1 = gendata2(1,Ntest1)
```

```
num_err = 0;

for i in range(Ntest0):
    yhat = np.dot(np.r_[1,xtest0[:,i]],Bhat)
    if yhat[1] > yhat[0]:
        num_err = num_err + 1;

for i in range(Ntest1):
    yhat = np.dot(np.r_[1,xtest1[:,i]],Bhat)
    if yhat[1] > yhat[0]:
        num_err = num_err + 1;

print("Number of errors: %d"%(num_err))
err_rate_linregress_test = float(num_err) / (Ntest0 + Ntest1);
print("Percent of errors: %.2f"%(err_rate_linregress_test))


# find max and min of sets
x_tot = np.r_[x0[0,:],x1[0,:]]
y_tot = np.r_[x0[1,:],x1[1,:]]
xlim = [np.min(x_tot),np.max(x_tot)]
ylim = [np.min(y_tot),np.max(y_tot)]

# find x,y coordinate of separating line
x_cor_lin = [xlim[0],xlim[1]]
y_cor_lin = [
    (Bhat[0,0]-Bhat[0,1]+(Bhat[1,0]-Bhat[1,1])*xlim[0])
            /(Bhat[2,1]-Bhat[2,0]),

    (Bhat[0,0]-Bhat[0,1]+(Bhat[1,0]-Bhat[1,1])*xlim[1])
            /(Bhat[2,1]-Bhat[2,0])
]

# create colored graph above/below line
xp1 = np.linspace(xlim[0],xlim[1], num=100)
yp1 = np.linspace(ylim[0],ylim[1], num=100)

red_pts = np.array([[],[]])
green_pts= np.array([[],[]])

for x in xp1:
    for y in yp1:
        yhat = np.dot(np.r_[1,x,y],Bhat)
        if yhat[1] > yhat[0]:
            green_pts = np.c_[green_pts,[x,y]]
        else:
            red_pts = np.c_[red_pts,[x,y]]

plotData(x0,x1)
plt.plot(x_cor_lin,y_cor_lin,color='black')
plt.scatter(green_pts[0,:],green_pts[1,:],color='blue',s=0.25)
plt.scatter(red_pts[0,:],red_pts[1,:],color='red',s=0.25)
plt.xlim(xlim)
plt.ylim(ylim)
plt.show()
```

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.

```
import sys
import numpy as np
import matplotlib.pyplot as plt

def gendata2(class_type,N):
    m0 = np.array(
```

```
        [[-0.132,0.320,1.672,2.230,1.217,-0.819,3.629,0.8210,1.808, 0.1700],
         [-0.711,-1.726,0.139,1.151,-0.373,-1.573,-0.243,-0.5220,-0.511,0.5330]])

    m1 = np.array(
         [[-1.169,0.813,-0.859,-0.608,-0.832,2.015,0.173,1.432,0.743,1.0328],
         [ 2.065,2.441,0.247,1.806,1.286,0.928,1.923,0.1299,1.847,-0.052]])

    x = np.array([[],[]])
    for i in range(N):
        idx = np.random.randint(10)
        if class_type == 0:
            m = m0[:,idx]
        elif class_type == 1:
            m = m1[:,idx]
        else:
            print("not a proper classifier")
            return 0
        x = np.c_[x, [[m[0]],[m[1]]] + np.random.randn(2,1)/np.sqrt(5)]
    return x

data = np.loadtxt("../data/classasgntrain1.dat",dtype=float)
x0 = data[:,0:2].T
x1 = data[:,2:4].T
data_tot = np.c_[x0,x1]

fig = plt.figure() # make handle to save plot
plt.scatter(x0[0,:],x0[1,:],c='red',label='$x_0$')
plt.scatter(x1[0,:],x1[1,:],c='blue',label='$y_0$')
plt.xlabel('X Coordinate')
plt.ylabel('Y Coordinate')
plt.legend()

N0 = x0.shape[1]
N1 = x1.shape[1];
N = N0 + N1

# quadratic
X = np.c_[np.ones((N,1)),data_tot.T,data_tot[0].T*data_tot[0].T, data_tot[0]*data_tot[1],
    data_tot[1]*data_tot[1]]

Y = np.r_[np.c_[np.ones((N0,1)),np.zeros((N0,1))],
          np.c_[np.zeros((N1,1)),np.ones((N1,1))]]

# find parameter matrix
Bhat = np.linalg.lstsq(np.dot(X.T,X),np.dot(X.T,Y))[0]

# find approximate response
Yhat = np.dot(X,Bhat)
Yhathard = Yhat > 0.5

num_err = sum(sum(abs(Yhathard - Y)))/2
print("Number of errors: %d"%(num_err))

Ntest0 = 10000;
Ntest1 = 10000;

err_rate_linregress_train = num_err / N

# generate the test data for class 0
xtest0 = gendata2(0,Ntest0)
xtest1 = gendata2(1,Ntest1)
num_err = 0;

for i in range(Ntest0):
    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)
    if yhat[1] > yhat[0]:
        num_err = num_err + 1;

for i in range(Ntest1):
    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)
    if yhat[1] > yhat[0]:
```

```
        num_err = num_err + 1;

print("Number of errors: %d"%(num_err))

err_rate_linregress_test = num_err / (Ntest0 + Ntest1);


# find max and min of sets
x_tot = np.r_[x0[0,:],x1[0,:]]
y_tot = np.r_[x0[1,:],x1[1,:]]
xlim = [np.min(x_tot),np.max(x_tot)]
ylim = [np.min(y_tot),np.max(y_tot)]

# create colored graph above/below line
xp1 = np.linspace(xlim[0],xlim[1], num=100)
yp1 = np.linspace(ylim[0],ylim[1], num=100)

red_pts = np.array([[],[]])
green_pts= np.array([[],[]])

for x in xp1:
    for y in yp1:
        yhat = np.dot(np.r_[1,x,y,x*x,x*y,y*y],Bhat)
        if yhat[1] > yhat[0]:
            green_pts = np.c_[green_pts,[x,y]]
        else:
            red_pts = np.c_[red_pts,[x,y]]

plt.scatter(green_pts[0,:],green_pts[1,:],color='blue',s=0.25)
plt.scatter(red_pts[0,:],red_pts[1,:],color='red',s=0.25)
plt.xlim(xlim)
plt.ylim(ylim)
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”

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

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:

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

Now taking the log of the equation gives us:

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

Linear and Quadratic Discriminant Analysis

Linear Logistic Regression

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

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

Note:

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

We begin with the equation:

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

Problem 9: Show that

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

Starting with the equation from the previous problem.

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

Problem 10: Show that

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

Beginning with the first derivative from the previous problem:

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

Note:

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

So,

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

k-nearest Neighbor Classifier

Naive Bayes Classifier

Optimal Bayes Classifier

Discussion