Stats 519: HW 7 - Classification

Due on May 6, 2009

Dr. Stephen Lee 1:30

Monte J. Shaffer

Problem 2

The trace of the matrix (A) is the sum of the diagonal elements ($\sum_{i=1}^{n} a_{ii}$) which is mathematically equivalent to the sum of the eigenvalues of the matrix ($\sum_{i=1}^{n} \lambda ii$ where $Ax = \lambda x$ and x are the eigenvectors). This means the trace of a matrix is invariant to the basis. Rui Xu, in *Survey of Clustering Algorithms* describes this invariance as follows:

"It is not difficult to see that the criterion based on the trace of S_W is the same as the sum of squared error criterion. To minimize the squared error criterion is equivalent to minimizing the trace of S_W or maximizing the trace of S_B. We can obtain a rich class of criterion functions based on the characteristics of S_W and S_B."

Simply stated, a minimization problem of a given matrix (A) is equivalent to a minimization problem of its unique finger print (λ), so the minimization of the trace is equivalent to the minimization of the matrix.

Proof. From page 651, the squared error criterion is described as:

$$J(\Gamma, M) = \sum_{i=1}^{K} \sum_{j=1}^{N} \gamma_{ij} ||x_j - m_i||^2$$

where Γ is a partition matrix (k-partitions); M is the cluster centroid matrix (m_i is sample mean for the i^{th} cluster; γ_{ij} is an indicator variable whether or not x_j belongs to cluster i (in other words, only make calculation to data in the appropriate cluster.

S_W is the within-class scatter matrix and is defined as (page 652):

$$S_W = \sum_{i=1}^{K} \sum_{j=1}^{N} \gamma_{ij} (x_j - m_i)(x_j - m_i)'$$

Applying the indicator function, a matrix of S_W is defined as:

$$\begin{bmatrix} (x_1 - m_1)^2 & (x_1 - m_1)(x_2 - m_1)' & \cdots & \cdots & (x_1 - m_1)(x_N - m_1)' \\ (x_1 - m_1)(x_2 - m_1)' & (x_2 - m_2)^2 & (x_2 - m_2)(x_3 - m_2)' & \cdots & (x_2 - m_2)(x_N - m_2)' \\ \vdots & & \ddots & & \vdots \\ (x_1 - m_1)(x_N - m_1)' & (x_N - m_1)'(x_2 - m_1) & \cdots & \cdots & (x_N - m_j)^2 \end{bmatrix} \quad (1)$$

Definitionally, the squared criterion is the sum of the diagonals of S_W, which is the definition of the trace (the indicator variable γ_{ij} accounts for the placement of the appropriate data.

□

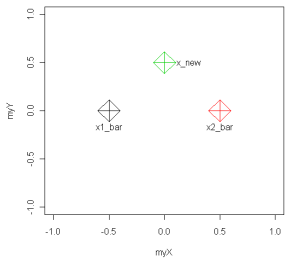
Problem 3

Lattin 12.3 Consider the following discriminant analysis problem. Sixty ($N = 60$) observations are drawn at random from the population; the associated summary statistics are shown below:

[With Fisher, I can adjust the cutoff of unequal using EQN 12.53 on page 454: $ln(\frac{q_1}{q_2})$, or just use the Mahalanobis' which is what I use]

```
n1=20;    x1_bar = matrix(c(-.5,0),nrow=2,ncol=1,byrow=T);
     rownames(x1_bar) = c("x","y");
n2=40;    x2_bar = matrix(c(.5,0),nrow=2,ncol=1,byrow=T);
     rownames(x2_bar) = c("x","y");
Cw = matrix(c(0.8,0.5,0.5,1),nrow=2,ncol=2,byrow=T);

q1=n1/(n1+n2); q2=n2/(n1+n2);

x_new = matrix(c(0,0.5),nrow=2,ncol=1,byrow=T);
     rownames(x_new) = c("x","y");

myX = cbind(x1_bar[1],x2_bar[1],x_new[1]);
myY = cbind(x1_bar[2],x2_bar[2],x_new[2]);

plot(myX,myY,col=1:3,xlim=c(-1,1),ylim=c(-1,1),pch = 9, cex=4);
identify(myX,myY,col=1:3,labels=c("x1_bar","x2_bar","x_new"),offset=1.2);
```

A new observation x_{new} is drawn at random from the population. Would you assign it to group 1 or group 2? Justify your answer.



Mathematically, it appears that the new data point is equidistant to the two clusters. Intuitively, the unequal group sizes would suggest that the probability is twice as large that the new data point should live in group 2. [See EQNs 12.48 and 12.49 in text; the prior probabilities $q_1 = \frac{20}{60}$ and $q_2 = \frac{40}{60}$ suggests a proportional chance criteria should be used (Table 12.6)]

Based on the formulation, I would classify x_{new} to Group 2.

Following Classification based on posterior probabilities [Bayes assumes $\Gamma_1 = \Gamma_2$], I can perform a log ratio to calculate the Group membership probabilistically:

```
Cw_inv    = solve(Cw);
Dsq_1     = t(x_new-x1_bar)%*%Cw_inv%*%(x_new-x1_bar);
Dsq_2     = t(x_new-x2_bar)%*%Cw_inv%*%(x_new-x2_bar);

(logRatio = log(q1/q2)-((Dsq_1-Dsq_2)/2));
if(logRatio > 0) myG = 1 else myG = 2;
myG;
```

This result aligns with my intuition based on equal Euclidean distances and proportional chances. Interestingly, the Mahalanobis' approach identifies that the $x1_{bar}$ is much closer to x_{new} than $x2_{bar}$ suggesting that the covariance structures of the data influence the classes. In the end, the best method to use accounts for unequal group sizes and Mahalanobis' distance. This posterior probability, in this case, aligned with my initial intuitions. Group 2 is my final answer.

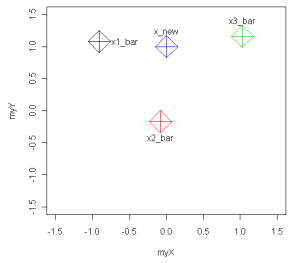
Problem 4

Lattin 12.4 Consider a three-group discriminant analysis on two variables (X and Y). Eighty-five ($N = 85$) observations are drawn at random from the population; the associated summary statistics are shown below:

```
n1=25;    x1_bar = matrix(c(-.91,1.08),nrow=2,ncol=1,byrow=T);
     rownames(x1_bar) = c("x","y");
          w1 = matrix(c(25.2,20.5,20.5,19.8),nrow=2,ncol=2,byrow=T);  s1 = w1/(n1-1);
n2=35;    x2_bar = matrix(c(-.08,-.17),nrow=2,ncol=1,byrow=T);
     rownames(x2_bar) = c("x","y");
          w2 = matrix(c(38.2,33.8,33.8,41.0),nrow=2,ncol=2,byrow=T);  s2 = w2/(n2-1);
n3=25;    x3_bar = matrix(c(1.03,1.16),nrow=2,ncol=1,byrow=T);
     rownames(x3_bar) = c("x","y");
          w3 = matrix(c(20.2,16.9,16.9,21.5),nrow=2,ncol=2,byrow=T);  s3 = w3/(n3-1);

x_new = matrix(c(0,1),nrow=2,ncol=1,byrow=T);
     rownames(x_new) = c("x","y");

myX = cbind(x1_bar[1],x2_bar[1],x3_bar[1],x_new[1]);
myY = cbind(x1_bar[2],x2_bar[2],x3_bar[2],x_new[2]);

plot(myX,myY,col=1:4,xlim=c(-1.5,1.5),ylim=c(-1.5,1.5),pch = 9, cex=4);
identify(myX,myY,col=1:4,labels=c("x1_bar","x2_bar","x3_bar","x_new"),offset=1.2);
```

A new observation x_{new} is drawn at random from the population. To which group would you assign this observation? Justify your answer.

Visually, it appears that x_{new} is closer to Group 1. Can I calculated a pooled variance to determine?




```
q1=n1/(n1+n2+n3);   q2=n2/(n1+n2+n3);   q3=n3/(n1+n2+n3);
Sp = Cw= (w1+w2+w3)/(n1+n2+n3-3);  # pooled
Cw_inv    = solve(Cw);
Dsq_1     = t(x_new-x1_bar)%*%Cw_inv%*%(x_new-x1_bar);
Dsq_2     = t(x_new-x2_bar)%*%Cw_inv%*%(x_new-x2_bar);
Dsq_3     = t(x_new-x3_bar)%*%Cw_inv%*%(x_new-x3_bar);

(logRatio = log(q1/q2)-((Dsq_1-Dsq_2)/2));   if(logRatio > 0) myG = 1 else myG = 2;
myG;

(logRatio = log(q1/q3)-((Dsq_1-Dsq_3)/2));   if(logRatio > 0) myG = 1 else myG = 3;
myG;

(logRatio = log(q2/q3)-((Dsq_2-Dsq_3)/2));   if(logRatio > 0) myG = 2 else myG = 3;
myG;
```

Using a pooled variance structure, I calculated pairwise comparisons using the log-ratio; the results suggest Group 3 is the best membership.

Problem 5

Lattin 12.8 Consider the following data obtained by a political science researcher. The variable Y_1 indicates whether the legislator voted in favor of ($Y_1 = 1$) or against ($Y_1 = 0$) a specific gun control bill; X_1 is the age of the legislator and X_2 is the number of guns owned by the legislator which are contained in the data file $GUN_{CONTROL}$.

```
setwd("C:/latex/statsMultiVariate/datasets");
     guns = read.table("GUN_CONTROL.txt");
          colnames(guns) = c("vote","age","guns");
               # EDA
               voteNo    = subset(guns,vote==0)[,-1];
               voteYes = subset(guns,vote==1)[,-1];

     library(MASS);
          (guns.LDA = lda(guns[,2:3],guns[,1]));
          (guns.PRED = predict(guns.LDA,guns[,2:3]));
     table(Actual = guns[,1], Predicted = guns.PRED$class);
```

Summary Data Six voted yes, Nine voted no. Those that voted yes have a median age of 35 (mean=34) and own a median of 0 guns (mean = 0.667). Those who voted no have a median age of 48 (mean=49.44) and own a median of 3 guns (mean = 3).

a) Find the coefficients of the discriminant function that explains the differences between legislators that voted in favor of and against the bill.

```
Coefficients of linear discriminants:
            LD1
   age  -0.1836884
   guns  0.2680674
```

b) Apply Box's test of covariance matrix equality. From 12.4.1, the hypothesis and test statistic is explained. B is calculated as outlined on page 443. [$\sigma^2 = C_{w_{(g)}} = \frac{SS_g}{n_g - 1} = fracSSd.f.$]

```
     n1=6;   # YES
          s1 = var(voteYes);
          w1 = s1*(n1-1);
     n2=9;     # NO
          s2 = var(voteNo);
          w2 = s2*(n2-1);

     G=2;      # number of groups
     p=2;      # number of variables
     N = n1+n2;

     Sp = Cw = (w1+w2)/(n1+n2-2); # pooled

          sum1 = 1/(n1-1) + 1/(n2-1);
          sum2 = (n1+n2-2);
     (c = (sum1-1/sum2)*(2*p^2+3*p-1)/(6*(p+1)*(G-1)));
          sum3 = (n1-1)*log(det(s1)) + (n2-1)*log(det(s2));
     (B = (1-c)*(sum2*log(det(Cw))-sum3));
[1] 1.027288
     (df = 0.5*p*(p+1)*(G-1));
     pchisq(B,df);
[1] 0.2053505
```

Based on these results, we Fail To Reject (FTR) null $H_0 : \Sigma_1 = \Sigma_2$.

c) How well does your model perform in correctly predicting the voting? With $N = 15$, the table (confusion matrix) comparing predicted to actual (with the same sample, no holdout data available) demonstrates some bias in the predictions. One vote was actually NO when it was predicted YES; two votes were actually YES and predicted NO. In total, 3 votes are misspecified; 3/15 = 20% error rate, which may suggest a better model may be found.

```
          Predicted
Actual 0 1
     0 8 1
     1 2 4
```

Problem 1

Write your R function for `kmeans` called `myKmeans` and compare them.

I used the following lines of code to compare the two algorithms:

```
set.seed(123);
x <- rbind(
          matrix(rnorm(100, sd = 0.3), ncol = 2),
          matrix(rnorm(55, mean = 1, sd = 0.3), ncol = 2),
          matrix(rnorm(63, mean = 1.5, sd = 0.3), ncol = 2)
     )
colnames(x) <- c("x", "y")
(cl <- kmeans(x, 3))
plot(x, col = cl$cluster)
points(cl$centers, col = 1:3, pch = 8, cex=2)
(c2 = myKmeans(x,3));
points(c2$centers, col = 1:3, pch = 9, cex=4)
```

My logic is similar to the `kmeans` with a few limitations. I did not implement the `nstep` feature, and resultingly increased my `maxIter` so the values converge more accurately. I wanted to based the convergence on a tolerance level, but decided to follow the stopping logic of `kmeans`. I also allow the initial centroids (`step 0`) to be passed to the function similar to how `kmeans` allows this. If the second parameter passed is a vector of data points, it will use those for the centroids, and figure out the number of partitions (k) from the vector. I did not implement error checking to guarantee the dimensions of the points match the dimensions of the data. It should however work in p-dimensional space. This was a good learning activity for me as I had never embedded subfunctions within a function. I place one iteration as a function and also created a distance function which calculates a Euclidean distance between two points (x, y). This is not as robust as the `dist` function as it manually does pair-wise distance comparisons, but it works.

Problem 1 continued on next page...

Program 1 myKmeans function

```
myKmeans = function(data,clusters,maxIter=25)
     {
     data = as.matrix(data);       n = nrow(data);          p = ncol(data);          # p-dim space
     myDist = function(x,y)
          {
          # simple Euclidean
          tDist = 0;
          for(d in 1:length(x))
               {
               tDist = tDist + (x[d]-y[d])^2;
               }
          tDist;    #sqrt not necessary as WSS is not calculated, just to find MIN
          }
     ## step 0:  initialize k-points as centers
     if(length(clusters)==1)
          {
          k = clusters;       init = data[sample(1:n, k),];
          }
          else
               {
               # cluster centers are passed
               k = nrow(cluster);       init = clusters;
               }
     kOneStep = function(data,init,n,k,myReturn="centroid")
          {
     ## step 1:  classify n points to k-centers (init are centroids)
          ## grab distances using custom Euclidean
          # loop and calculate every pairwise distance, could put into vector,
          # but this will work [although processor expensive]
          newData = matrix(0, nrow=n,ncol=k);
          for(i in 1:n)
               {
               for(j in 1:k)
                    {
                    newData[i,j] = myDist(data[i,],init[j,]);
                    }
               }
          newClass = matrix(0, nrow=1,ncol=k);
          ## loop through distances and classify based on min distance
          for(i in 1:n)
               {
               myMin = min(newData[i,]);
               for(j in 1:k)
                    {
                    if(newData[i,j]==myMin)
                         {
                         myIndex = j;        break;
                         }
                    }
               newClass[i]=myIndex;
               }
          if(myReturn != "centroid")
               {
               newClass;
               }
               else
                    {
                    ## step 2:  compute new center
                         myData = cbind(newClass,data);
                         centroid = matrix(0, nrow=k,ncol=p);
                         for(j in 1:k)
                              {
                              tempData = subset(myData,newClass==j)[,-1];
                              for(m in 1:p)
                                   {
                                   centroid[j,m] = mean(tempData[,m]);
                                   }
                              }
                         centroid;
                    }
          }
          for(iter in 1:maxIter)
               {
               init = kOneStep(data,init,n,k);
               }
               # one more time to get classifications
               newClass = kOneStep(data,init,n,k,"class");
     out = list(cluster = newClass, centers = init);        class(out) = "myKmeans";
     out;
     }
```