THE UNIVERSITY OF MINNESOTA

Statistics 5401 November 28, 2005

Examples of Linear Classification in MacAnova

This handout includes examples of the use of MacAnova in classification using linear discriminant functions. There is a two group example and a five group example. The latter also illustrates stepwise selection of variables and the use of macro **jackknife()** to estimate error probabilities.

Two groups with $\Sigma_1 = \Sigma_2$

This is example of computing a discriminant function for classifying two groups, assuming equal prior probablities and mis-classification costs, and equal covariance matrices. The data used are the Fisher Iris data.

Read in data and compute summary statistics

```
Cmd> irisdata <- read("","t11_05")
) Data from Table 11.5 p. 657-658 in
) Applied Mulivariate Statistical Analysis, 5th Edition
) by Richard A. Johnson and Dean W. Wichern, Prentice Hall, 2002
) These data were edited from file T11-5.DAT on disk from book
) The variety number was moved to column 1
) Measurements on petals of 4 varieties of Iris.  Originally published
in
) R. A. Fisher, The use of mltiple measurements in taxonomic problems,
) Annals of Eugenics, 7 (1936) 179-198
) Col. 1: variety number (1 = I. setosa, 2 = I. versicolor,
)                         3 = I. virginica)
) Col. 2: x1 = sepal length
) Col. 3: x2 = sepal width
) Col. 4: x3 = petal length
) Col. 5: x4 = petal width
) Rows 1-50:    group 1 = Iris setosa
) Rows 51-100:  group 2 = Iris versicolor in
) Rows 101-150: group 3 = Iris virginica in
Read from file "TP1:Stat5401:Data:JWData5.txt"

Cmd> varieties <- irisdata[,1]; y <- irisdata[,-1]

Cmd> setosa <- y[varieties == 1,]; n1 <- nrows(setosa)

Cmd> versicolor <- y[varieties == 2,]; n2 <- nrows(versicolor)

Cmd> virginica <- y[varieties == 3,]; n3 <- nrows(virginica)

Cmd> n <- vector(n1,n2,n3); n
(1)          50          50          50

Cmd> s1 <- tabs(setosa,covar:T); s2 <- tabs(versicolor,covar:T)

Cmd> s3 <- tabs(virginica,covar:T)

Cmd> labs <- getlabels(y,2) # columnlabels

Cmd> labs <- structure(labs,labs)
```

```
Cmd> setlabels(s1,labels:labs); setlabels(s2,labels:labs)
Cmd> setlabels(s3,labels:labs)
Cmd> ybar1 <- tabs(setosa,mean:T)
Cmd> ybar2 <- tabs(versicolor,mean:T)
Cmd> ybar3 <- tabs(virginica,mean:T)
```

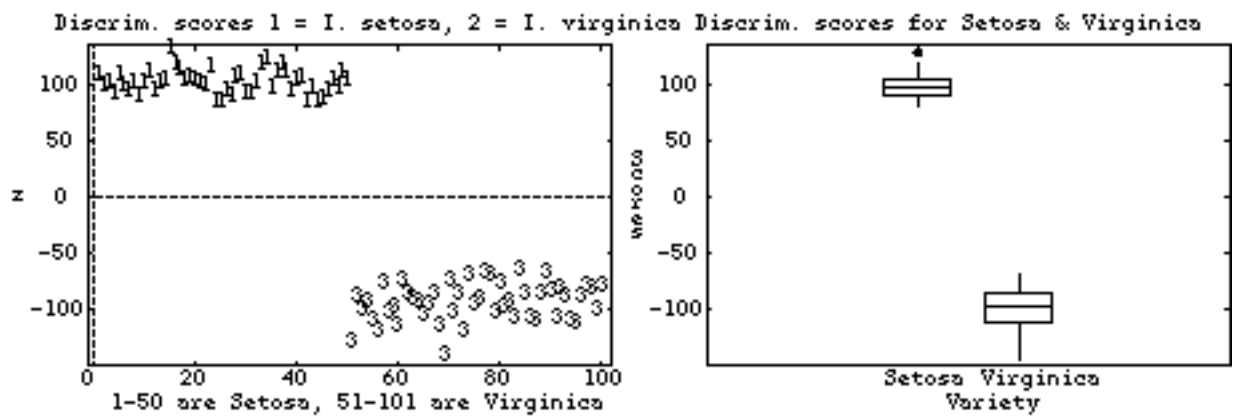
Discriminate between I. setosa and I. virginica

```
Cmd> spooled13 <- ((n1 - 1)*s1 + (n3 - 1)*s3)/(n1+  n3 - 2)
Cmd> l13 <- vector(solve(spooled13, ybar1 - ybar3));l13
      SepLen      SepWid      PetLen      PetWid
      15.841       12.02     -36.518     -36.758
Cmd> # something like a contrast of sepal size and petal size
Cmd> cutoff <- l13' %*% (ybar1 + ybar3)/2 ;cutoff
(1,1)     -39.518          Cut off value for l'x
```

$S^{-1}(\overline{\mathbf{x}}_1 - \overline{\mathbf{x}}_3)$

This is $(\overline{\mathbf{x}}_1 - \overline{\mathbf{x}}_3)' S^{-1} (\overline{\mathbf{x}}_1 + \overline{\mathbf{x}}_3)/2$.

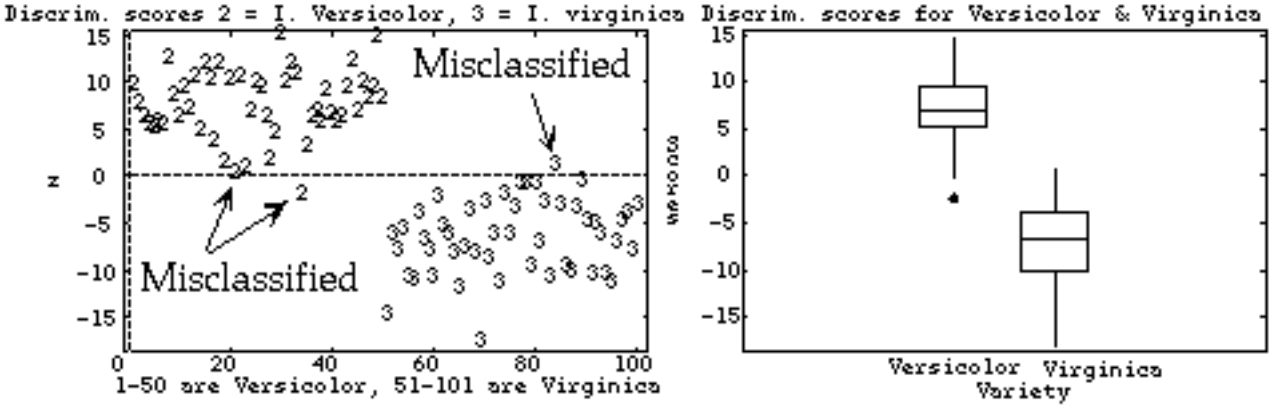
```
Cmd> z13 <- vconcat(setosa,virginica) %*% l13 - cutoff # scores
Cmd> plot(run(n1+n3),z13,symbols:vector(rep(1,n1),rep(3,n3)),title:\
    "Discrim. scores 1 = I. setosa, 2 = I. virginica",\
    xlab:"1-50 are Setosa, 51-101 are Virginica")
Cmd> vboxplot(split(z13,vector(rep(1,n1),rep(2,n3))),title:\
"Discrim. scores for Setosa & Virginica",\
ylab:"Scores",xticklabs:vector("Setosa","Virginica"),xlab:"Variety")
```



Note that in the first plot, all the *I. setosa* points are above zero line and all the *I. virginica* are below the line, indicating perfect discrimination. This perfect discrimination is equally apparent in the box plot.

Discriminate between I. versicolor and I. virginica

```
Cmd> spooled23 <- ((n2 - 1)*s2 + (n3 - 1)*s3)/(n2 + n3 - 2)

Cmd> l23 <- vector(solve(spooled23,ybar2 - ybar3)); l23
      SepLen      SepWid      PetLen      PetWid
      3.5563      5.5786     -6.9701     -12.386

Cmd> cutoff23 <- l23' %*% (ybar2+ybar3)/2; cutoff23
(1,1)     -16.663

Cmd> z23 <- vconcat(versicolor,virginica) %*% l23 - cutoff23

Cmd> sum(z23[run(n2)] < 0)
(1)          2                2 misclassified Versicolor cases

Cmd> sum(z23[-run(n3)] > 0)
(1)          1                1 misclassified Virginica case

Cmd> run(n2+n3)[run(n2+n3) <= n1 && z23 < 0 ||\
               run(n2+n3) > n1 && z23 >= 0]
(1)         21          34          84   misclassified cases

Cmd> z23[vector(21,34,84)]
(1)    -0.25463     -2.3021     0.56122

Cmd> plot(run(n2=n3),z23,symbols:vector(rep(2,n2),rep(3,n3)),title:\
  "Discrim. scores 2 = I. Versicolor, 3 = I. virginica",\
  xlab:"1-50 are Versicolor, 51-101 are Virginica")

Cmd> vboxplot(split(z23,vector(rep(1,n2),rep(2,n3))),title:\
 "Discrim. scores for Versicolor & Virginica",\
 ylab:"Scores",xticklabs:vector("Versicolor","Virginica"),xlab:"Variety")
```



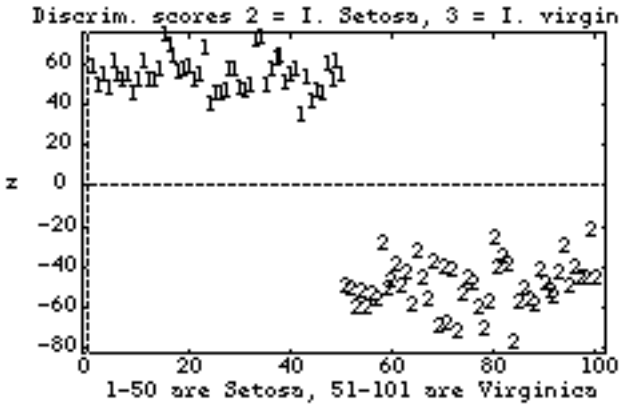
Discriminate between I. setosa and I. versicolor

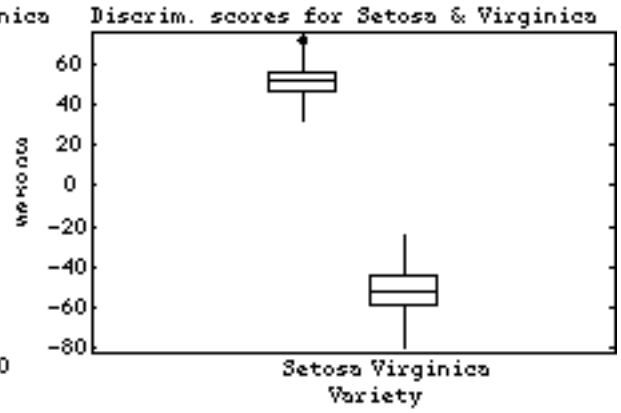
```
Cmd> spooled12 <- ((n1-1)*s1+(n2-1)*s2)/(n1+n2-2)

Cmd> l12 <- vector(solve(spooled12,(ybar1-ybar2))); l12
      SepLen      SepWid      PetLen      PetWid
      3.0528      18.023     -21.766     -30.844

Cmd> cutoff12 <- l12' %*% (ybar1 + ybar2)/2; cutoff12
(1,1)     -13.962

Cmd> z12 <- vconcat(setosa,versicolor) %*% l12 - cutoff12

Cmd> sum(z12[run(n1)] < 0)
(1)          0            No misclassified setosa cases
```

```
Cmd> sum(z12[-run(n1)] > 0)
(1)          0
```

No misclassified versicolor cases

```
Cmd> plot(run(n1+n2),z12,symbols:vector(rep(1,n1),rep(2,n2)),title:\
    "Discrim. scores 2 = I. Setosa, 3 = I. virginica",\
    xlab:"1-50 are Setosa, 51-101 are Virginica")

Cmd> vboxplot(split(z12,vector(rep(1,n1),rep(2,n2))),title:\
 "Discrim. scores for Setosa & Virginica",\
 ylab:"Scores",xticklabs:vector("Setosa","Virginica"),xlab:"Variety")
```





Multi Group Classification Example assuming $\Sigma_1 = \Sigma_2 = \ldots = \Sigma_g$

Here is a slightly edited listing of file **remote.txt** which contains data taken from the *SAS User's Guide* Version 5 Edition p. 329.

```
cropsense      36     5 format
) Remote sensing data on 5 crops (from SAS User's Guide: Statistics)
) Col 1: crop = crop number (1 = corn, 2 = soybeans, 3 = cotton,
)                            4 = sugarbeets, 5 = clover)
) Col 2-5: x1,x2,x3,x4:measurements from photographs of fields of crops
)"%lf %lf %lf %lf %lf"
(f2.0,4f3.0)
 1 12 15 16 73 |  2 22 32 31 43 |  3 53 48 75 26 |  5 31 31 11 11
 1 15 15 31 32 |  2 27 45 24 12 |  4 22 23 25 42 |  5 32 13 27 32
 1 15 23 30 30 |  2 20 23 23 25 |  4 25 25 24 26 |  5 32 32 62 16
 1 15 32 32 15 |  2 24 24 25 32 |  4 25 43 32 15 |  5 36 26 54 32
 1 16 27 27 26 |  3 26 25 23 24 |  4 26 54  2 54 |  5 51 31 31 16
 1 16 27 31 33 |  3 29 24 26 28 |  4 34 25 16 52 |  5 53  8  6 54
 1 18 20 25 23 |  3 31 32 33 34 |  4 54 23 21 54 |  5 56 13 13 71
 2 12 13 15 42 |  3 34 32 28 45 |  5 12 45 32 54 |  5 87 54 61 21
 2 21 25 23 24 |  3 34 35 25 78 |  5 24 58 25 34 |  5 96 48 54 62
```

The values represent four measurements X_1, X_2 ..., X_4 derived from aerial photographs of fields primarily planted in one of five crops -- corn, soybeans, cotton, sugar beets, or clover. The goal is to develop a classification algorithm that can determine the crop grown on a field from similar aerial measurements. The following MacAnova output assumes that the covariance matrices of the 5 groups of crops are identical.

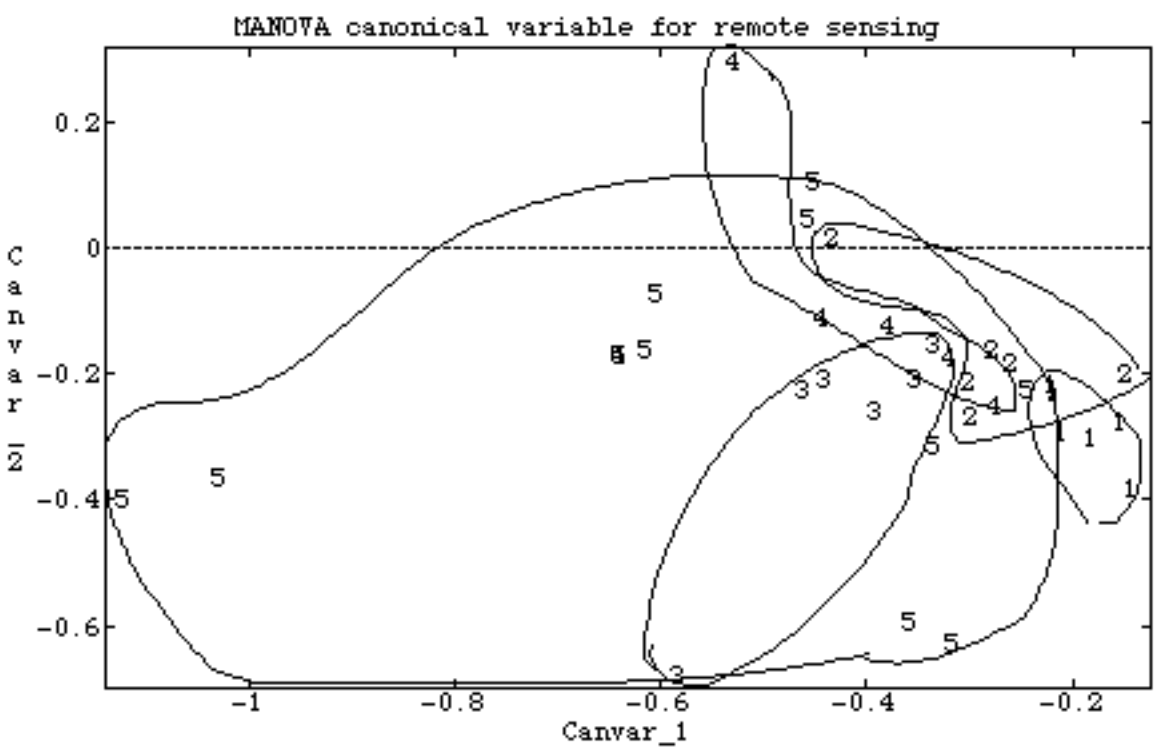
Canonical variable plot

```
Cmd> cropsense <- read("","cropsense") # read from remote.txt
cropsense      36     5 format
) Remote sensing data on 5 crops (from SAS User's Guide: Statistics)
) Col 1: crop = crop number (1 = corn, 2 = soybeans, 3 = cotton,
)                            4 = sugarbeets, 5 = clover)
) Col 2-5: x1,x2,x3,x4:measurements from photographs of fields of crops
Read from file "TP1:Stat5401:Data:remote.txt"

Cmd> crop <- factor(cropsense[,1])

Cmd> y <- matrix(cropsense[,-1],labels:structure("@","x"))

Cmd> N <- nrows(y); p <- ncols(y)

Cmd> g <- max(vector(crop));fe <- N - g; fh <- g-1

Cmd> vector(N,p,g,fh,fe,labels:vector("N","p","g","fh","fe"))
          N          p          g         fh         fe
         36          4          5          4         31
```

```
Cmd> manova("y=crop",silent:T) # suppress output

Cmd> h <- matrix(SS[2,,]); e <- matrix(SS[3,,]) # H and E

Cmd> eigs <- releigen(h,e); eigs # relative eigen structure
component: values
(1)     0.67419     0.18169    0.052753   0.0068474
component: vectors
           (1)         (2)         (3)         (4)
x1   -0.011041   0.0016551  -0.0053649  -0.0026367
x2  -0.0045781   0.0076941   0.0083184   0.0098499
x3   0.0029493   -0.014274   0.0035404   0.0016054
x4  9.2382e-06  -0.0024996    0.009666   -0.004619

Cmd> f <- (diag(h)/fh)/(diag(e)/fe);f #F-statistics
(1)      4.8126     0.88624       1.317     0.35766

Cmd> p*(1-cumF(f,fh,fe))# Bonferronized P-Values;P1 < .05
(1)     0.01555      1.9346      1.1419      3.3469
```

Comment: All the eigenvalue based tests have P-values around 0.1 and are therefore not significant at the 5% level.

```
Cmd> z <- y %*% eigs$vectors[,run(2)] # MANOVA canonical variables

Cmd> plot(Canvar_1:z[,1],Canvar_2:z[,2],symbols:crop,\
title:"MANOVA canonical variable for remote sensing")
```



It is clear that discriminating the groups is going to be difficult. There is overlap of groups. In particular group 5 completely encloses group 3 and most of group 4.

Direct white box computation of linear discriminant functions

```
Cmd> stats <- groupcovar(crop,y);stats
component: n
(1)          7          6          6          6         11
component: means       Mean row vectors for the 5 groups
(1,1)      15.286     22.714     27.429     33.143
(2,1)          21         27       23.5     29.667
(3,1)        34.5     32.667         35     39.167
(4,1)          31     32.167         20       40.5
(5,1)      46.364     32.636     34.182     36.636
component: covariance  Pooled variance matrix
(1,1)      259.08     33.485     90.046     43.183
(2,1)      33.485     160.65     80.599    -50.237
(3,1)      90.046     80.599     226.93    -94.958
(4,1)      43.183    -50.237    -94.958     337.78
```

`groupcovar()` returns the sample sizes in component `n`, the sample means as rows of component `means`, and the pooled variance matrix in component `covariance`.

```
Cmd> spooled <- stats$covariance # pooled variance matrix
```

You could also compute `spooled` as `spooled <- matrix(SS[3,,]/DF[3])`.

```
Cmd> ybar <- stats$means' # note the transpose; means are now columns
Cmd> discrimfn <- solve(spooled,ybar) #cols of ybar are group means
Cmd> discrimfn # compare with output from discrim() below
(1,1)   -0.041805  3.6928e-05    0.024624     0.04245    0.089073
(2,1)      0.1197     0.15896     0.17596     0.20988     0.17379
(3,1)     0.16511     0.10622      0.1588    0.065404     0.11899
(4,1)     0.16768     0.14133     0.18362     0.16408     0.15637
```

Columns of `discrimfn` are $\hat{\ell}_1, \ldots, \hat{\ell}_5$.

```
Cmd> addcon <- -.5 * sum(discrimfn * ybar); addcon #row vector
(1,1)     -6.0831     -5.4908     -9.6736       -8.01     -9.7989
```

`addcon[i]` is $-c_i = -\hat{\ell}'\overline{\mathbf{y}}_i/2$.

```
Cmd> d <- y %*% discrimfn + addcon # discrimanant function scores
Cmd> d[run(3),] # 1st 3 rows; largest in row are underlined
(1,1)      10.094      8.9105      9.2062      8.6715       7.196
(2,1)      5.5696      4.7095      4.1337      3.0528      2.8367
(3,1)      6.0268      5.5923      5.0154      4.3382      3.7953
```

These are the scores $\hat{\ell}_i'\mathbf{x} - c_i$ for the first 3 cases.

```
Cmd> temp <- exp(d - d[,1]) # start of conversion to probabilities
Cmd> prob <- temp / sum(temp')' ; prob[run(3),]# divide by row sums
            (1)         (2)         (3)         (4)         (5)
(1)     0.49639     0.15208     0.20441     0.11974    0.027381
(2)     0.55348     0.23418     0.13168    0.044673    0.035993
(3)     0.43413     0.28114     0.15789    0.080221    0.046611
```

The elements of **prob** are estimated *posterior* probabilities, assuming equal *prior* probabilities. The first column of **d** was subtracted from all columns of **d** in computing posterior probabilities (**prob <- exp(d - d[,1])**) to ensure that **exp()** was computable. **d[i,1]** is an example of the function K(**x**) discussed in lecture 34.

```
Cmd> # Determine which column is maximum for each row of prob
Cmd> class <- vector(grade(prob',down:T)[1,]);class[run(3)]
(1)          1           1           1
Cmd> # first 3 cases would be classified in group 1
```

grade(prob',down:T)[1,] computes the column numbers of the largest element in each row of **prob**. Type **help(grade)** for more information.

```
Cmd> print(format:"5.3f",hconcat(prob,crop,class))
MATRIX:   Probabilities                    True Guess
 (1,1) 0.496 0.152 0.204 0.120 0.027 1.000 1.000
 (2,1) 0.553 0.234 0.132 0.045 0.036 1.000 1.000
 (3,1) 0.434 0.281 0.158 0.080 0.047 1.000 1.000
 (4,1) 0.309 0.376 0.145 0.111 0.058 1.000 2.000*
 (5,1) 0.324 0.339 0.151 0.128 0.059 1.000 2.000*
 (6,1) 0.385 0.265 0.196 0.100 0.054 1.000 1.000
 (7,1) 0.346 0.365 0.120 0.106 0.064 1.000 2.000*
 (8,1) 0.449 0.305 0.101 0.112 0.033 2.000 1.000*
 (9,1) 0.232 0.370 0.134 0.175 0.090 2.000 2.000
(10,1) 0.249 0.206 0.293 0.162 0.090 2.000 3.000*
(11,1) 0.053 0.307 0.115 0.379 0.145 2.000 4.000*
(12,1) 0.267 0.367 0.131 0.154 0.081 2.000 2.000
(13,1) 0.241 0.301 0.179 0.170 0.109 2.000 2.000
(14,1) 0.177 0.347 0.142 0.203 0.132 3.000 2.000*
(15,1) 0.184 0.297 0.178 0.181 0.160 3.000 2.000*
(16,1) 0.153 0.208 0.279 0.179 0.181 3.000 3.000
(17,1) 0.104 0.161 0.285 0.248 0.202 3.000 3.000
(18,1) 0.077 0.067 0.430 0.287 0.139 3.000 3.000
(19,1) 0.043 0.029 0.566 0.021 0.341 3.000 3.000
(20,1) 0.295 0.250 0.213 0.155 0.087 4.000 1.000*
(21,1) 0.199 0.336 0.154 0.189 0.122 4.000 2.000*
(22,1) 0.110 0.314 0.187 0.249 0.140 4.000 2.000*
(23,1) 0.007 0.069 0.054 0.821 0.048 4.000 4.000
(24,1) 0.087 0.173 0.195 0.358 0.187 4.000 4.000
(25,1) 0.023 0.069 0.175 0.258 0.474 4.000 5.000*
(26,1) 0.245 0.157 0.365 0.192 0.041 5.000 3.000*
(27,1) 0.035 0.158 0.183 0.527 0.098 5.000 4.000*
(28,1) 0.039 0.342 0.054 0.408 0.157 5.000 4.000*
(29,1) 0.266 0.269 0.180 0.112 0.173 5.000 2.000*
(30,1) 0.318 0.131 0.389 0.024 0.138 5.000 3.000*
(31,1) 0.269 0.109 0.415 0.035 0.172 5.000 3.000*
(32,1) 0.025 0.138 0.125 0.190 0.521 5.000 5.000
(33,1) 0.028 0.107 0.093 0.329 0.443 5.000 5.000
(34,1) 0.025 0.057 0.171 0.282 0.466 5.000 5.000
(35,1) 0.000 0.004 0.083 0.029 0.883 5.000 5.000
(36,1) 0.000 0.001 0.104 0.030 0.865 5.000 5.000
```

18 misclassified (50%)

Use of **discrim()** for black box computation

discrim() computes linear discriminant functions from a data matrix **y** and a vector of group numbers by

```
Cmd> result <- discrim(group, y)
```

result is a structure with components **coefs** and **addcon**.

result$coefs is a p by g matrix with **result[,i]** containing the linear discriminant function coefficients $\hat{\ell}_i = S^{-1}\overline{y}_i$, where **S** is the pooled estimated covariance matrix and $\overline{y}_i$ is the sample mean vector for group i.

result$addcon is a length g vector with **result$addcon[i]** containing $-c_i = -\overline{y}_i' S^{-1} \overline{y}_i/2$.

```
Cmd> discrim(crop,y) # compare with output above
component: coefs   Linear  coefficients
          crop1       crop2       crop3       crop4       crop5
x1   -0.041805  3.6928e-05    0.024624     0.04245    0.089073
x2      0.1197     0.15896     0.17596     0.20988     0.17379
x3     0.16511     0.10622      0.1588    0.065404     0.11899
x4     0.16768     0.14133     0.18362     0.16408     0.15637
component: addcon  Additive  constants
           crop1       crop2       crop3       crop4       crop5
(1)     -6.0831     -5.4908     -9.6736       -8.01     -9.7989
```

Jackknife (leave-one-out) estimation of error probabilities

To illustrate what macro **jackknife()** does, I first compute the discriminant function based on all data except case 6.

```
Cmd> # Negative subscript deletes indicated case
Cmd> J <- 6; discrimfn6 <- discrim(crop[-J],y[-J,]); discrimfn6
component: coefs   Computed  omitting  case  6
       Group 1     Group 2     Group 3     Group 4     Group 5
x1   -0.039712 -0.00019358     0.02354    0.040829    0.085932
x2     0.11289     0.15474     0.17143      0.2041     0.16924
x3      0.1582     0.10328     0.15429    0.063828     0.11572
x4     0.16135     0.13705     0.17806      0.1591     0.15166
component: addcon
        Group 1     Group 2     Group 3     Group 4     Group 5
(1)     -5.7388     -5.3335     -9.3931     -7.7754     -9.5097
Cmd> d_6 <- y[J,] %*% discrimfn6$coefs + discrimfn6$addcon; d_6
        Group 1     Group 2     Group 3     Group 4     Group 5
(1)      6.9025       6.566       6.271      5.6174      5.0269
Cmd> p6 <- exp(d_6  - d_6[1])/sum(vector(exp(d_6-d_6[1]))); p6
        Group 1     Group 2     Group 3     Group 4     Group 5
(1)      0.3737     0.26692     0.19874     0.10337    0.057275
```

This last line contains estimated posterior probabilities for case 6 computed from function estimated from all the cases <u>except</u> case 6.

jackknife() does this computation for every case, that is, it computes the estimated posterior probability for each case using linear discriminant functions computed from the remaining data omitting that case.

```
Cmd> p <- jackknife(crop,y)

Cmd> list(p) # 1st 5 cols are post probs, 6th is guessed crop number
p                REAL   36    6

Cmd> print(format:"6.4f",hconcat(p,crop))
MATRIX:    Posterior  Probabilities      Guess   True
 (1,1) 0.2525 0.1959 0.3228 0.1910 0.0377 3.0000 1.0000*
 (2,1) 0.5175 0.2500 0.1432 0.0487 0.0406 1.0000 1.0000
 (3,1) 0.4248 0.2814 0.1610 0.0835 0.0494 1.0000 1.0000
 (4,1) 0.2531 0.4011 0.1572 0.1228 0.0658 2.0000 1.0000*
 (5,1) 0.3069 0.3417 0.1558 0.1327 0.0630 2.0000 1.0000*
 (6,1) 0.3737 0.2669 0.1987 0.1034 0.0573 1.0000 1.0000  Same as p6
 (7,1) 0.3124 0.3770 0.1269 0.1139 0.0698 2.0000 1.0000*
 (8,1) 0.5044 0.2266 0.1116 0.1217 0.0358 1.0000 2.0000*
 (9,1) 0.2349 0.3556 0.1373 0.1783 0.0939 2.0000 2.0000
(10,1) 0.2640 0.1542 0.3124 0.1716 0.0978 3.0000 2.0000*
(11,1) 0.0499 0.1438 0.1317 0.4874 0.1872 4.0000 2.0000*
(12,1) 0.2706 0.3520 0.1349 0.1579 0.0846 2.0000 2.0000
(13,1) 0.2419 0.2912 0.1820 0.1725 0.1124 2.0000 2.0000
(14,1) 0.1903 0.3709 0.0860 0.2160 0.1368 2.0000 3.0000*
(15,1) 0.1957 0.3122 0.1324 0.1919 0.1678 2.0000 3.0000*
(16,1) 0.1557 0.2093 0.2706 0.1815 0.1829 3.0000 3.0000
(17,1) 0.1081 0.1651 0.2700 0.2509 0.2059 3.0000 3.0000
(18,1) 0.1010 0.0796 0.1673 0.4690 0.1832 4.0000 3.0000*
(19,1) 0.0535 0.0268 0.1551 0.0168 0.7479 5.0000 3.0000*
(20,1) 0.3111 0.2640 0.2258 0.1077 0.0914 1.0000 4.0000*
(21,1) 0.2126 0.3518 0.1626 0.1439 0.1292 2.0000 4.0000*
(22,1) 0.1283 0.3567 0.2089 0.1511 0.1551 2.0000 4.0000*
(23,1) 0.0105 0.1992 0.1481 0.5113 0.1308 4.0000 4.0000
(24,1) 0.0944 0.1817 0.2051 0.3217 0.1971 4.0000 4.0000
(25,1) 0.0234 0.0698 0.1975 0.1326 0.5767 5.0000 4.0000*
(26,1) 0.3567 0.1485 0.3340 0.1582 0.0026 1.0000 5.0000*
(27,1) 0.0306 0.1639 0.1488 0.6401 0.0166 4.0000 5.0000*
(28,1) 0.0374 0.4219 0.0336 0.4604 0.0467 4.0000 5.0000*
(29,1) 0.3001 0.2931 0.1799 0.1120 0.1150 1.0000 5.0000*
(30,1) 0.4430 0.1266 0.3838 0.0119 0.0348 1.0000 5.0000*
(31,1) 0.3384 0.1122 0.4413 0.0263 0.0818 3.0000 5.0000*
(32,1) 0.0289 0.1635 0.1343 0.2183 0.4550 5.0000 5.0000
(33,1) 0.0328 0.1418 0.0891 0.4942 0.2422 4.0000 5.0000*
(34,1) 0.0300 0.0652 0.2023 0.4033 0.2993 4.0000 5.0000*
(35,1) 0.0000 0.0020 0.1239 0.0290 0.8450 5.0000 5.0000
(36,1) 0.0000 0.0001 0.2099 0.0297 0.7602 5.0000 5.0000

Cmd> sum(p[,6] != crop) # number of errors
(1,1)           23
```

Only 13/36 were correctly assigned, indicating the optimism of the naive method of evaluation of classification error probability.

Fisher discriminant functions based on canonical variables

Now compute discriminant function using first two canonical variables based on the canonical variables `z[,1]` and `z[,2]` and relative eigenvectors previously computed.

```
Cmd> discrimfnz <- discrim(crop,z); discrimfnz
component: coefs discriminant function based on 2 can. vars
          crop1       crop2       crop3       crop4       crop5
(1)     -5.9382     -8.8624     -13.233     -13.335     -17.365
(2)     -8.5027     -5.1796       -8.96      -2.725     -7.8004
component: addcon
          crop1       crop2       crop3       crop4       crop5
(1)     -1.7348     -1.6995     -4.1193      -2.988      -5.845
```

```
Cmd> dz <- z %*% discrimfnz$coefs + discrimfnz$addcon # scores
```

```
Cmd> pz <- exp(dz-dz[,1])/sum(exp(dz-dz[,1])')' # posterior probs
```

`sum(exp(d-d[,1])')'` computes the row sums of `exp(d-d[,1])` as a column vector. Note the two transposes.

```
Cmd> classz <- vector(grade(pz',down:T)[1,]) # columns with max prob
```

```
Cmd> print(format:"6.4f",hconcat(pz,classz,crop))
MATRIX:
 (1,1) 0.4490 0.2914 0.1436 0.0811 0.0350 1.0000 1.0000
 (2,1) 0.5247 0.2316 0.1629 0.0473 0.0336 1.0000 1.0000
 (3,1) 0.4262 0.2762 0.1702 0.0821 0.0454 1.0000 1.0000
 (4,1) 0.3341 0.3115 0.1667 0.1314 0.0563 1.0000 1.0000
 (5,1) 0.3265 0.3141 0.1657 0.1365 0.0571 1.0000 1.0000
 (6,1) 0.3845 0.2789 0.1852 0.0957 0.0557 1.0000 1.0000
 (7,1) 0.3400 0.3081 0.1687 0.1269 0.0562 1.0000 1.0000
 (8,1) 0.4059 0.3338 0.1200 0.1091 0.0312 1.0000 2.0000*
 (9,1) 0.2295 0.3140 0.1733 0.2023 0.0808 2.0000 2.0000
(10,1) 0.2572 0.2616 0.2347 0.1416 0.1048 2.0000 2.0000
(11,1) 0.0580 0.2268 0.1250 0.4541 0.1362 4.0000 2.0000*
(12,1) 0.2618 0.3164 0.1721 0.1775 0.0722 2.0000 2.0000
(13,1) 0.2295 0.2828 0.2091 0.1777 0.1009 2.0000 2.0000
(14,1) 0.1714 0.2832 0.1929 0.2384 0.1141 2.0000 3.0000*
(15,1) 0.1737 0.2528 0.2314 0.2028 0.1392 2.0000 3.0000*
(16,1) 0.1532 0.2110 0.2755 0.1774 0.1830 3.0000 3.0000
(17,1) 0.1034 0.1925 0.2576 0.2260 0.2205 3.0000 3.0000
(18,1) 0.0918 0.1726 0.2711 0.2149 0.2496 3.0000 3.0000
(19,1) 0.0529 0.0317 0.4691 0.0225 0.4238 3.0000 3.0000
(20,1) 0.2774 0.2805 0.2114 0.1432 0.0876 2.0000 4.0000*
(21,1) 0.1932 0.2859 0.1982 0.2147 0.1080 2.0000 4.0000*
(22,1) 0.1237 0.2584 0.1905 0.2868 0.1406 4.0000 4.0000
(23,1) 0.0096 0.1242 0.0364 0.7490 0.0808 4.0000 4.0000
(24,1) 0.0805 0.2110 0.1962 0.3211 0.1912 4.0000 4.0000
(25,1) 0.0203 0.0780 0.2141 0.2479 0.4397 5.0000 4.0000*
(26,1) 0.3032 0.3025 0.1831 0.1425 0.0687 1.0000 5.0000*
(27,1) 0.0455 0.2073 0.1147 0.4907 0.1418 4.0000 5.0000*
(28,1) 0.0373 0.2057 0.0884 0.5536 0.1150 4.0000 5.0000*
(29,1) 0.2315 0.2248 0.2791 0.1276 0.1369 3.0000 5.0000*
(30,1) 0.3291 0.1076 0.4071 0.0263 0.1299 3.0000 5.0000*
```

```
(31,1) 0.2621 0.1081 0.4277 0.0345 0.1677 3.0000 5.0000*
(32,1) 0.0249 0.0911 0.2163 0.2640 0.4037 5.0000 5.0000
(33,1) 0.0220 0.1051 0.1712 0.3618 0.3399 4.0000 5.0000*
(34,1) 0.0204 0.0779 0.2151 0.2462 0.4404 5.0000 5.0000
(35,1) 0.0005 0.0033 0.1047 0.0373 0.8541 5.0000 5.0000
(36,1) 0.0002 0.0015 0.0782 0.0228 0.8974 5.0000 5.0000

Cmd> sum(classz != crop) # number of errors(1,1)
(1,1)          14
```

Now there are only 14 errors out of 36 cases (39%). By concentrating on the important canonical variables we obtain better apparent classification.

Now express the coefficients in a form in which they can be applied directly to the original data. The vectors of coefficients to multiply x_1, x_2, ..., x_4 are linear combinations of the first two relative eigenvectors, weighted with `discrimfnz$coefs`. The additive constants are the same.

```
Cmd> coeffsz <- eigs$vectors[,run(2)] %*% discrimfnz$coefs;coeffsz
          crop1       crop2       crop3       crop4       crop5
x1      0.05149    0.089277     0.13128     0.14272     0.17882
x2    -0.038235  0.00072023   -0.008357    0.040084    0.019481
x3      0.10385    0.047793    0.088862 -0.00043522    0.060124
x4     0.021199    0.012865    0.022275   0.0066883    0.019338

Cmd> dz[run(3),] # scores as computed above from canonical variables
           crop1       crop2       crop3       crop4       crop5
(1)       1.5187      1.0865     0.37848    -0.19276     -1.0333
(2)       2.3617      1.5437       1.192   -0.045338    -0.38786
(3)       1.9096      1.4759     0.99171     0.26239    -0.33082

Cmd> dz1 <- y %*% coeffsz + discrimfnz$addcon;dz1[run(3),] # from data
           crop1       crop2       crop3       crop4       crop5
(1)       1.5187      1.0865     0.37848    -0.19276     -1.0333
(2)       2.3617      1.5437       1.192   -0.045338    -0.38786
(3)       1.9096      1.4759     0.99171     0.26239    -0.33082
```

The scores computed from the canonical variables (`dz`) and the actual data (`dz1`) are identical.

Selection of variables

It is important not to include variables that are not helpful in a classification rule. Including unnecessary variables increases the estimation error in the resulting classification rule without reducing error probabilities.

The most common methods for selection of variables are *forward* and *backward* variable selection. These are very similar to forward and backward selection of predictor variables in linear regression.

For forward selection, you start with all variables "out" and bring them in

one at a time, entering the variable that most distinguishes the groups at each stage. As with stepwise regression, this choice is based on an F-to-enter statistic. You bring variables in until none of the remaining "out" variables has a large enough F-to-enter statistic.

For backward selection, you start with all variables "in" and remove them one at a time, always removing the variable that is least important in distinguishing between the groups as measured by a *F-to-delete* or *F-to-remove* statistic. You remove variables until all the remaining "in"variables have a large enough F-to-delete statistic.

Forward selection algorithm

Start with all variables "out".

Pick the variable with largest F-to-enter, where for variable ℓ, F-to-enter is $F_\ell = (H_{\ell i}/f_h)/(E_{\ell i}/f_e)$, the F-statistic for testing the null hypothesis that all groups have the same mean. If it is not "significant", stop. Otherwise, bring this variable "in" and remove it from the list of "out" variables.

At each subsequent stage compute the F-to-enter as F-statistics ("partial F's") for each of the "out" variables using the "in" variables as covariates. If the largest such F is "significant" move the corresponding variable from the "out" list to the "in" list. Otherwise stop.

Macros **dastepsetup()** and **daentervar()** implement this algorithm in MacAnova. **dastepsetup()** sets things up, specifying which variables are in and then prints the F-to-enter statistics. **daentervar()** enters one or more variables specified by the user and then prints F-statistics.

```
Cmd> dastepsetup("y=crop") # intialize with all variables "out"
Model: "y=crop"
No variables are "in"

F(4,31) to enter
            x1          x2          x3          x4
F       4.8126     0.88624       1.317     0.35766
P    0.0038874     0.48366     0.28548     0.83671

Cmd> # x1 has largest F so enter it first

Cmd> daentervar(x1) # or daentervar(1); enter variable
Model: "y=crop"
F(4,31) to delete
            x1
F       4.8126
P    0.0038874
```

```
F(4,30) to enter
            x2          x3          x4
F      0.32984     0.92281     0.24245  x3 has largest F-to-enter
P      0.85569     0.46369     0.91196
```

Note the output now includes F-to-delete for the one variable that is "in". We'll see how that is used later. F-to-delete for the variable that has just been entered is the same as F-to-enter for that variable just before it was entered.

All the current F-to-enter values are small and non-significant, so you would normally stop. However, here are more steps, just to illustrate the process.

The largest F-to-enter is .92281 for x3, so variable 3 would be the next to bring in.

```
Cmd> daentervar(x3) # enter variable 3
Model: "y=crop"
F(4,30) to delete
            x1          x3
F       4.1701     0.92281
P    0.0083803     0.46369

F(4,29) to enter
            x2          x4
F      0.70206     0.36694  x2 has largest F-to-enter
P      0.59691     0.83015
Cmd> daentervar(x2) # enter variable 2
Model: "y=crop"
F(4,29) to delete
            x1          x2          x3
F       3.7749     0.70206      1.3043
P     0.013679     0.59691     0.29164
F(4,28) to enter
            x4
F       0.3516
P      0.84068
Cmd> daentervar(x4) # enter last variable
Model: "y=crop"
F(4,28) to delete
            x1          x2          x3          x4
F        3.264     0.67504      1.2928      0.3516
P     0.025674      0.6149     0.29669     0.84068
All variables are "in"
```

The history of what you have done is available using macro dasteplook()

```
Cmd> dasteplook(history)
(1)          1           3           2           4
```

This says variables 1, 3, 2 and 4 were entered in that order.

At any time, you can use **dastepstatus()** to see the current status:

```
Cmd> dastepstatus()
Model: "y=crop"
F(4,28) to delete
            x1          x2          x3          x4
F        3.264     0.67504      1.2928      0.3516
P     0.025674      0.6149     0.29669     0.84068

All variables are "in"
```

So far I have bypassed the question of <u>what critical values you should use</u> in determining whether a variable is important enough to move from the 'outs' to the 'ins.' Although the statistics are F-statistics, you can't just use ordinary critical values from an F-table (although that is what some computer programs do). At a minimum, you should Bonferronize all the F-statistics at each stage, using as the number K of tests the number of 'out' variables. It is too conservative to Bonferronize by K = p(p+1)/2, the maximum possible number of F-statistics that might be examined.

```
Cmd> k <- vector(4,3,2,1) #number of outs at each stage
Cmd> invF(.05/k,fh,fe-vector(0,1,2,3),upper:T) #Bonferronized crit vals
(1)      3.8049      3.5855      3.2674      2.7141
```

The first F = 4.8126 > 3.8049 and is thus significant and should be used in the discriminant functions. None of the later F's (0.92281, 0.70206 and 0.3516) even approach significance.

<u>Backward selection algorithm</u>

This starts with all variables "in".

For each variable, an F-to-delete statistic is computed. This is the value of the F-to-enter statistic for that variable if it were removed. If F-to-delete is "large enough", you stop. Otherwise you remove it.

You keep repeating the previous step with the reduced set of variables until you stop, having found a set of variables to use, or until all the variables are removed.

You use **dastepsetup()** with keyword phrase **allin:T** to start backward variable selection. At each step you use **daremovevar()** to remove a variable, selecting the variable with the *smallest* F-to-delete, unless all the F-to-delete values are "large enough." You don't have to respecify the model (although that's OK) since **dastepsetup()** remembers it from the first use of **stepsetup()**.

```
Cmd> dastepsetup(allin:T) # or dastepsetup("y=crops",allin:T)
Model: "y=1+crop"
F(4,28) to delete
           x1          x2          x3          x4
F       3.264     0.67504      1.2928      0.3516
P    0.025674      0.6149     0.29669     0.84068

All variables are "in"
```

x4 has smallest F-to-delete statistic (0.3516) so x4 is the first variable to remove.

```
Cmd> daremovevar(x4) # Remove x4
Model: "y=1+crop"
F(4,29) to delete
           x1          x2          x3
F      3.7749     0.70206      1.3043
P    0.013679     0.59691     0.29164

F(4,28) to enter
           x4
F      0.3516  Same as F-to-delete for x4 at previous step
P     0.84068

Cmd> daremovevar(x2) # Now x2 has smallest F-to-delete (.70206)
Model: "y=1+crop"
F(4,30) to delete
           x1          x3
F      4.1701     0.92281
P   0.0083803     0.46369

F(4,29) to enter
           x2          x4
F     0.70206     0.36694
P     0.59691     0.83015

Cmd> daremovevar(x3) # Now x3 has smallest F-to-delete (0.92281)
Model: "y=1+crop"
F(4,31) to delete
           x1
F      4.8126
P   0.0038874

F(4,30) to enter
           x2          x3          x4
F     0.32984     0.92281     0.24245
P     0.85569     0.46369     0.91196

Cmd> dasteplook(history) # negative numbers mean remove
(1)           1           2           3           4          -4
(6)          -2          -3
```

You would normally keep X_1, since the F is 'significant'. The conclusion would be that X_1 is the only variable with any discriminatory power.

Again, I have bypassed the issue of what critical values you should use. It is less obvious how to Bonferronize. It is attractive to use a method

that is consistent with the forward Bonferronized critical values. This results in stopping only when the smallest F is significant when Bonferronized by the current number of 'ins' plus 1. By this criterion you would keep X_1 because F exceeds $F_{1-.05/(3+1),4,31}$ = 3.8049 as computed above.

Since both forward and backward selection lead to using just X_1, let's see how we do.

```
Cmd> discrimfn1 <- discrim(crop,y[,1]); discrimfn1
component: coefs
(1,1)       0.059    0.081056     0.13316     0.11965     0.17896
component: addcon
          crop1       crop2       crop3       crop4       crop5
(1)    -0.45093    -0.85109     -2.2971     -1.8546     -4.1485

Cmd> d1 <- y[,1] %*% discrimfn1$coefs +  discrimfn1$addcon #scores

Cmd> p1 <- exp(d1-d1[,1])/sum(exp(d1-d1[,1])')' # probabilities

Cmd> class1 <- vector(grade(p1',down:T)[1,]) # classification

Cmd> print(format:"6.4f",hconcat(p1,class1,crop))
MATRIX:
 (1,1) 0.3483 0.3042 0.1339 0.1772 0.0364 1.0000 1.0000
 (2,1) 0.3151 0.2940 0.1513 0.1923 0.0472 1.0000 1.0000
 (3,1) 0.3151 0.2940 0.1513 0.1923 0.0472 1.0000 1.0000
 (4,1) 0.3151 0.2940 0.1513 0.1923 0.0472 1.0000 1.0000
 (5,1) 0.3041 0.2901 0.1573 0.1972 0.0514 1.0000 1.0000
 (6,1) 0.3041 0.2901 0.1573 0.1972 0.0514 1.0000 1.0000
 (7,1) 0.2822 0.2813 0.1693 0.2066 0.0606 1.0000 1.0000
 (8,1) 0.3483 0.3042 0.1339 0.1772 0.0364 1.0000 2.0000*
 (9,1) 0.2500 0.2663 0.1873 0.2195 0.0769 2.0000 2.0000
(10,1) 0.2395 0.2607 0.1932 0.2234 0.0831 2.0000 2.0000
(11,1) 0.1894 0.2302 0.2214 0.2393 0.1197 4.0000 2.0000*
(12,1) 0.2606 0.2715 0.1813 0.2154 0.0711 2.0000 2.0000
(13,1) 0.2189 0.2491 0.2049 0.2306 0.0965 2.0000 2.0000
(14,1) 0.1990 0.2367 0.2161 0.2367 0.1116 2.0000 3.0000*
(15,1) 0.1707 0.2169 0.2315 0.2436 0.1372 4.0000 3.0000*
(16,1) 0.1531 0.2033 0.2408 0.2465 0.1563 4.0000 3.0000*
(17,1) 0.1285 0.1824 0.2526 0.2484 0.1881 3.0000 3.0000
(18,1) 0.1285 0.1824 0.2526 0.2484 0.1881 3.0000 3.0000
(19,1) 0.0316 0.0682 0.2544 0.1935 0.4523 5.0000 3.0000*
(20,1) 0.2395 0.2607 0.1932 0.2234 0.0831 2.0000 4.0000*
(21,1) 0.2089 0.2430 0.2105 0.2338 0.1039 2.0000 4.0000*
(22,1) 0.2089 0.2430 0.2105 0.2338 0.1039 2.0000 4.0000*
(23,1) 0.1990 0.2367 0.2161 0.2367 0.1116 2.0000 4.0000*
(24,1) 0.1285 0.1824 0.2526 0.2484 0.1881 3.0000 4.0000*
(25,1) 0.0290 0.0640 0.2511 0.1885 0.4674 5.0000 4.0000*
(26,1) 0.3483 0.3042 0.1339 0.1772 0.0364 1.0000 5.0000*
(27,1) 0.2189 0.2491 0.2049 0.2306 0.0965 2.0000 5.0000*
(28,1) 0.1531 0.2033 0.2408 0.2465 0.1563 4.0000 5.0000*
(29,1) 0.1446 0.1963 0.2450 0.2475 0.1665 4.0000 5.0000*
(30,1) 0.1446 0.1963 0.2450 0.2475 0.1665 4.0000 5.0000*
(31,1) 0.1136 0.1684 0.2589 0.2478 0.2113 3.0000 5.0000*
(32,1) 0.0375 0.0774 0.2600 0.2032 0.4219 5.0000 5.0000
```

```
(33,1) 0.0316 0.0682 0.2544 0.1935 0.4523 5.0000 5.0000
(34,1) 0.0243 0.0560 0.2439 0.1782 0.4976 5.0000 5.0000
(35,1) 0.0010 0.0046 0.1003 0.0482 0.8459 5.0000 5.0000
(36,1) 0.0004 0.0020 0.0704 0.0300 0.8972 5.0000 5.0000
Cmd> sum(crop!=class1) # 36*APER
(1,1)          18
```

There appear to be 18 incorrect, the same as when using all the data. Two cases (14 and 23) are essentially ties and could be assigned to either crop 2 or 4.

You can use **jackknife()** to attempt to correct for the optimism of using the training sample to evaluate the classifier. However, this would do nothing to reflect what was done during the variable selection. In principle you could do forward or backward selection n times, omitting a different case, and then use the selected variables to classify that case, but that has not been tried here.

You can use these same macros to do variable selection by a "hybrid" of forward and backward selection. At each step you first examine the F-to-delete statistics and remove the variable with the smallest F if the F-to-delete statistic is not too large. If there are no variables that can be removed, you then look at the F-to-enter statistics. If the largest F is "large enough", then enter that variable. If not, then you are done. You can start either with all variables "out", as in forward selection, or with all variables "in", as in backward selection. With the current data set, you would never back track so the hybrid method results in the same as forward or back selection. In other data sets you can end up selecting a different set of variables from either forward or backward selection.

It is also possible to look at all $2^p - 1$ possible combinations of variables as long as p is not too large. Macro **dascreen()** does this. By default, it returns the best 5 subsets as determined by AIC = N Log(det(E)/det(H + E) + $2*n_{params}$ where $n_{params} = q \times f_h + q(q+1)/2$, q = size of subset

```
Cmd> result <- dascreen("y=crop");result
component: subsets
          Set 1       Set 2       Set 3       Set 4       Set 5
(1)           1           1           1           1           3
(2)           0           3           2           4           0
component: criterion
       Set 1       Set 2       Set 3       Set 4       Set 5
    -7.3892     0.43337      3.0614      3.4655      4.3499
```

The best subset is just x1. Next is {x1, x3} ... . The criterion values are the values of AICC (small or large negative is good).