MAS-12

2022-11-09

Chapter 12 Classification

```
library(MASS)
pdf(file="ch12.pdf", width=8, height=6, pointsize=9)
options(width=65, digits=5)
library(class)
library(nnet)
```

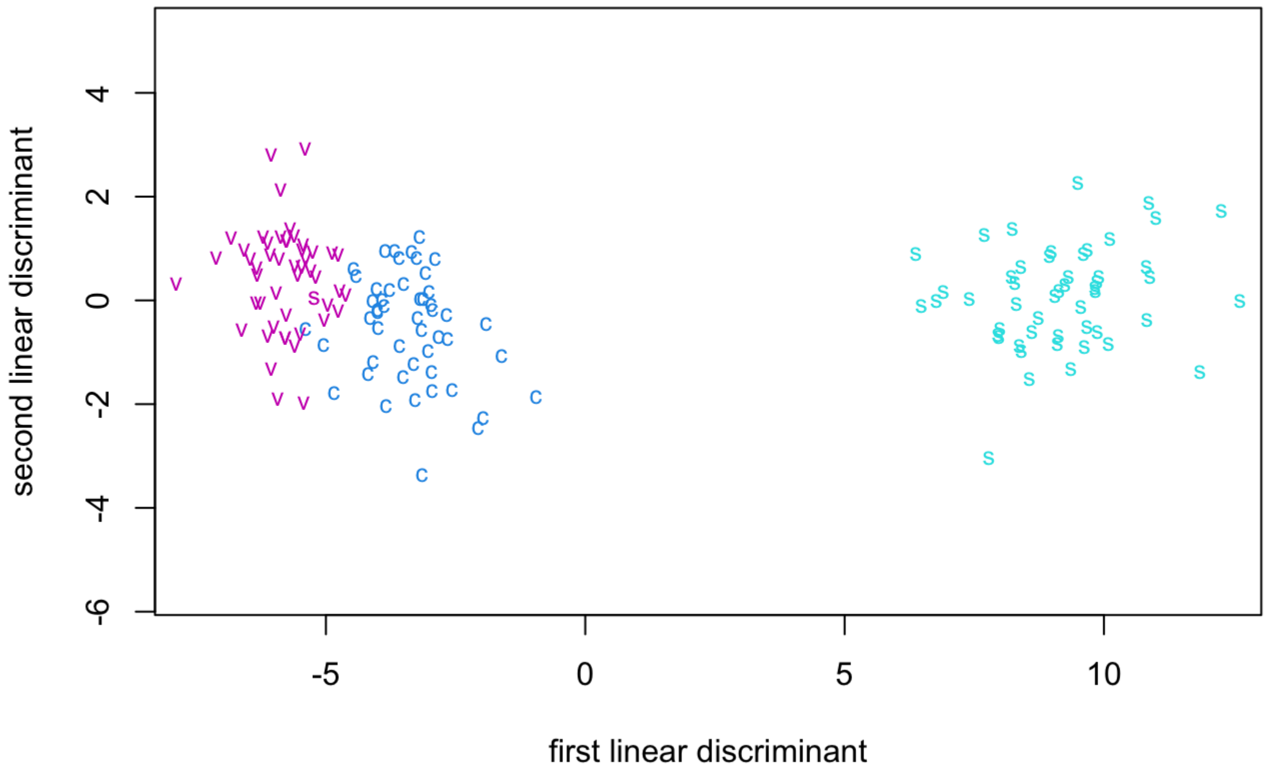
12.1 Discriminant Analysis

```
ir <- rbind(iris3[,,1], iris3[,,2], iris3[,,3])
ir.species <- factor(c(rep("s", 50), rep("c", 50), rep("v", 50)))
```

```
(ir.lda <- lda(log(ir), ir.species))
```

```
## Call:
## lda(log(ir), grouping = ir.species)
##
## Prior probabilities of groups:
##       c       s       v
## 0.33333 0.33333 0.33333
##
## Group means:
##   Sepal L. Sepal W. Petal L. Petal W.
## c   1.7773   1.0123  1.44293  0.27093
## s   1.6082   1.2259  0.37276 -1.48465
## v   1.8807   1.0842  1.70943  0.69675
##
## Coefficients of linear discriminants:
##              LD1      LD2
## Sepal L. -3.7798  4.27690
## Sepal W. -3.9405  6.59422
## Petal L.  9.0240  0.30952
## Petal W.  1.5328 -0.13605
##
## Proportion of trace:
##    LD1    LD2
## 0.9965 0.0035
```

```
ir.ld <- predict(ir.lda, dimen = 2)$x
ir.ld[,1]<- -ir.ld[,1]
eqscplot(ir.ld, type = "n", xlab = "first linear discriminant",
          ylab = "second linear discriminant")
text(ir.ld, labels = as.character(ir.species[-143]),
      col = 3 + unclass(ir.species), cex = 0.8)
```




```
# plot(ir.lda, dimen = 1)
# plot(ir.lda, type = "density", dimen = 1)
```

```
lcrabs <- log(crabs[, 4:8])
crabs.grp <- factor(c("B", "b", "O", "o")[rep(1:4, each = 50)])

(dcrabs.lda <- lda(crabs$sex ~ FL + RW + CL + CW, lcrabs))
```

```
## Call:
## lda(crabs$sex ~ FL + RW + CL + CW, data = lcrabs)
##
## Prior probabilities of groups:
##   F   M
## 0.5 0.5
##
## Group means:
##        FL     RW     CL     CW
## F 2.7087 2.5795 3.4210 3.5559
## M 2.7303 2.4668 3.4642 3.5838
##
## Coefficients of linear discriminants:
##          LD1
## FL  -2.8896
## RW -25.5176
## CL  36.3169
## CW -11.8280
```

```
table(crabs$sex, predict(dcrabs.lda)$class)
```

```
##
##      F  M
##   F 97  3
##   M  3 97
```

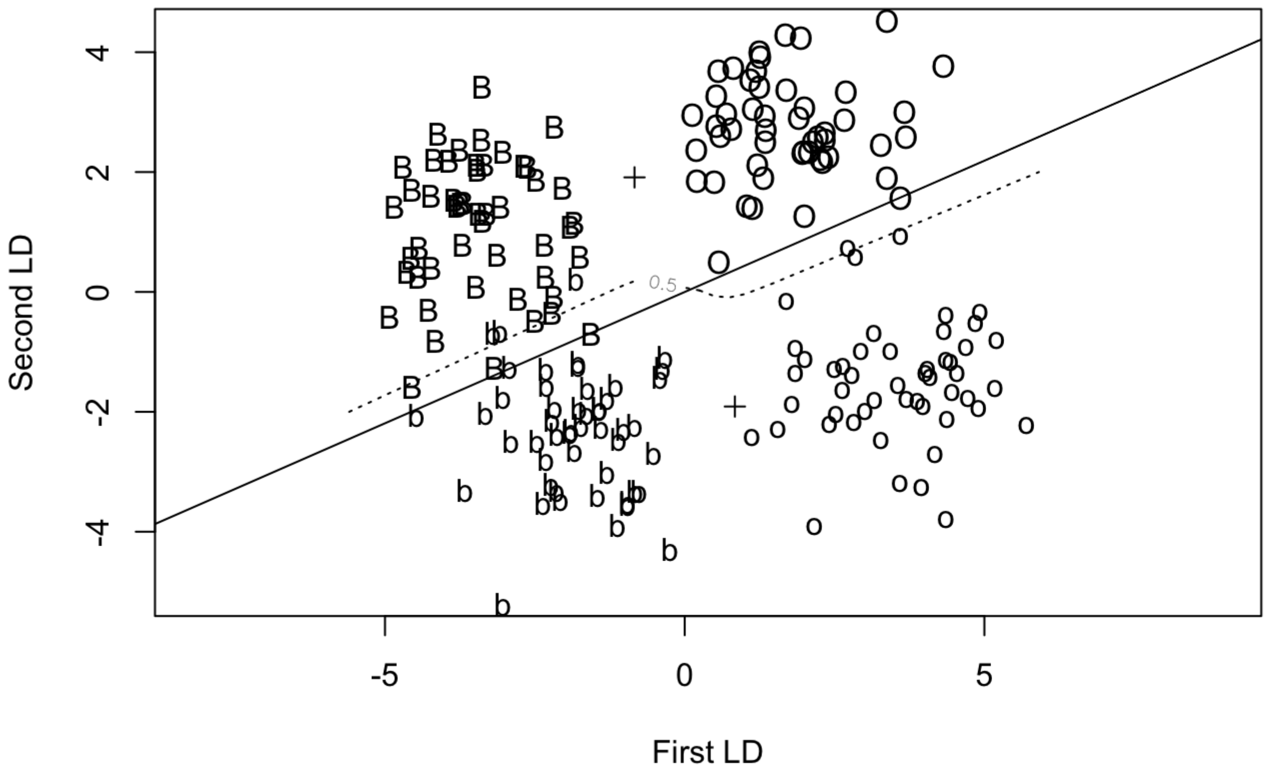
```
(dcrabs.lda4 <- lda(crabs.grp ~ FL + RW + CL + CW, lcrabs))
```

```
## Call:
## lda(crabs.grp ~ FL + RW + CL + CW, data = lcrabs)
##
## Prior probabilities of groups:
##    b    B    o    O
## 0.25 0.25 0.25 0.25
##
## Group means:
##       FL     RW     CL     CW
## b 2.5650 2.4752 3.3127 3.4623
## B 2.6727 2.4438 3.4380 3.5781
## o 2.8525 2.6838 3.5294 3.6496
## O 2.7879 2.4899 3.4904 3.5894
##
## Coefficients of linear discriminants:
##        LD1      LD2      LD3
## FL  36.256  -4.8446 -19.1065
## RW  13.384  22.7870   7.0771
## CL  20.289 -48.3804  58.3452
## CW -65.645  33.7102 -49.5127
##
## Proportion of trace:
##    LD1    LD2    LD3
## 0.6422 0.3491 0.0087
```

```
dcrabs.pr4 <- predict(dcrabs.lda4, dimen = 2)
dcrabs.pr2 <- dcrabs.pr4$post[, c("B", "O")] %*% c(1, 1)
table(crabs$sex, dcrabs.pr2 > 0.5)
```

```
##
##     FALSE TRUE
##   F    96    4
##   M     3   97
```

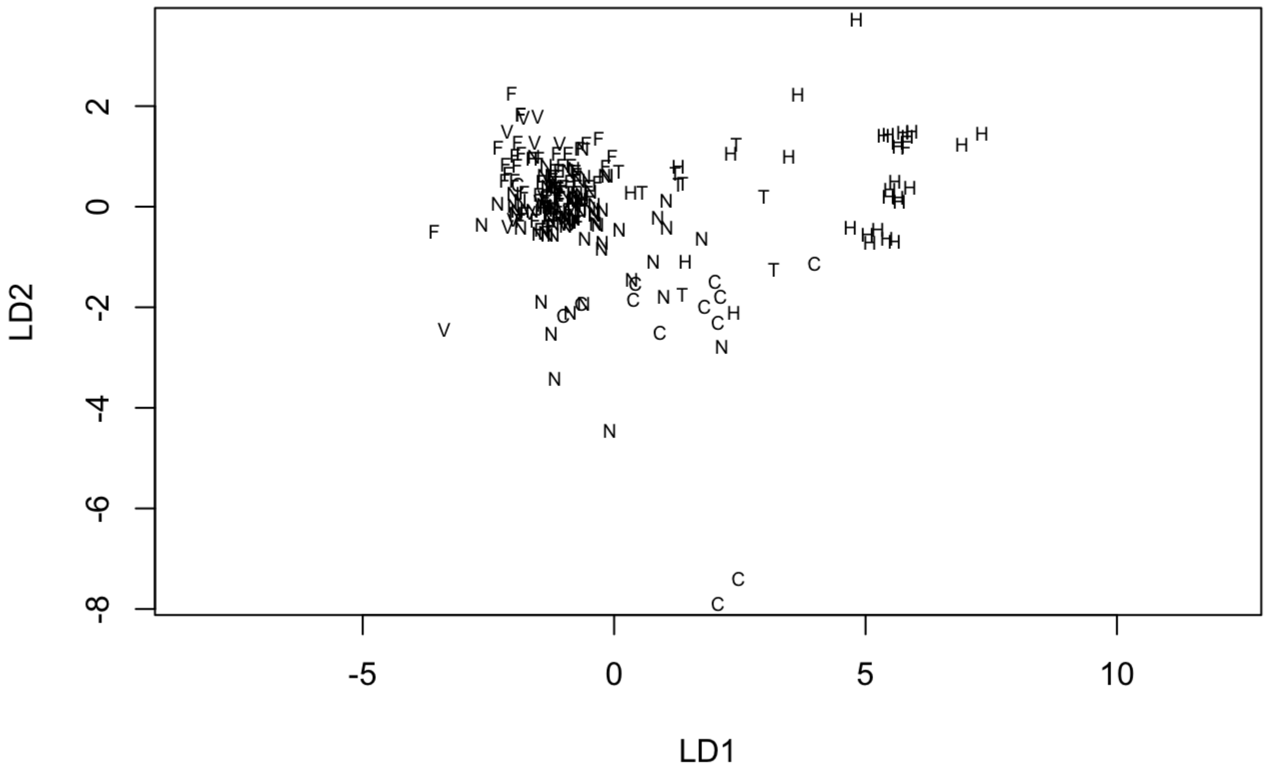
```r
cr.t <- dcrabs.pr4$x[, 1:2]
cr.t[,2]<--cr.t[,2]
eqscplot(cr.t, type = "n", xlab = "First LD", ylab = "Second LD")
text(cr.t, labels = as.character(crabs.grp))
perp <- function(x, y) {
   m <- (x+y)/2
   s <- - (x[1] - y[1])/(x[2] - y[2])
   abline(c(m[2] - s*m[1], s))
   invisible()
}
cr.m <- lda(cr.t, crabs$sex)$means
points(cr.m, pch = 3, mkh = 0.3)
perp(cr.m[1, ], cr.m[2, ])

cr.lda <- lda(cr.t, crabs.grp)
x <- seq(-6, 6, 0.25)
y <- seq(-2, 2, 0.25)
Xcon <- matrix(c(rep(x,length(y)),
              rep(y, rep(length(x), length(y)))),ncol=2)
cr.pr <- predict(cr.lda, Xcon)$post[, c("B", "O")] %*% c(1,1)
contour(x, y, matrix(cr.pr, length(x), length(y)),
       levels = 0.5, labex = 0, add = TRUE, lty=  3)
```




```r
for(i in c("O", "o",  "B", "b"))
 print(var(lcrabs[crabs.grp == i, ]))
```

```
##           FL       RW       CL       CW       BD
## FL 0.048712 0.040674 0.052366 0.052165 0.053486
## RW 0.040674 0.035038 0.044230 0.044110 0.045120
## CL 0.052366 0.044230 0.056893 0.056651 0.057999
## CW 0.052165 0.044110 0.056651 0.056510 0.057791
## BD 0.053486 0.045120 0.057999 0.057791 0.059476
##           FL       RW       CL       CW       BD
## FL 0.032173 0.029149 0.031774 0.031767 0.032387
## RW 0.029149 0.028188 0.029426 0.029453 0.029828
## CL 0.031774 0.029426 0.031953 0.031859 0.032594
## CW 0.031767 0.029453 0.031859 0.031921 0.032481
## BD 0.032387 0.029828 0.032594 0.032481 0.033844
##           FL       RW       CL       CW       BD
## FL 0.052964 0.043251 0.056633 0.056006 0.058748
## RW 0.043251 0.037247 0.046800 0.046336 0.048304
## CL 0.056633 0.046800 0.061045 0.060328 0.063118
## CW 0.056006 0.046336 0.060328 0.059738 0.062495
## BD 0.058748 0.048304 0.063118 0.062495 0.066021
##           FL       RW       CL       CW       BD
## FL 0.043578 0.043610 0.046080 0.045675 0.050083
## RW 0.043610 0.045040 0.046544 0.046172 0.050729
## CL 0.046080 0.046544 0.049147 0.048631 0.053537
## CW 0.045675 0.046172 0.048631 0.048261 0.053015
## BD 0.050083 0.050729 0.053537 0.053015 0.059463
```

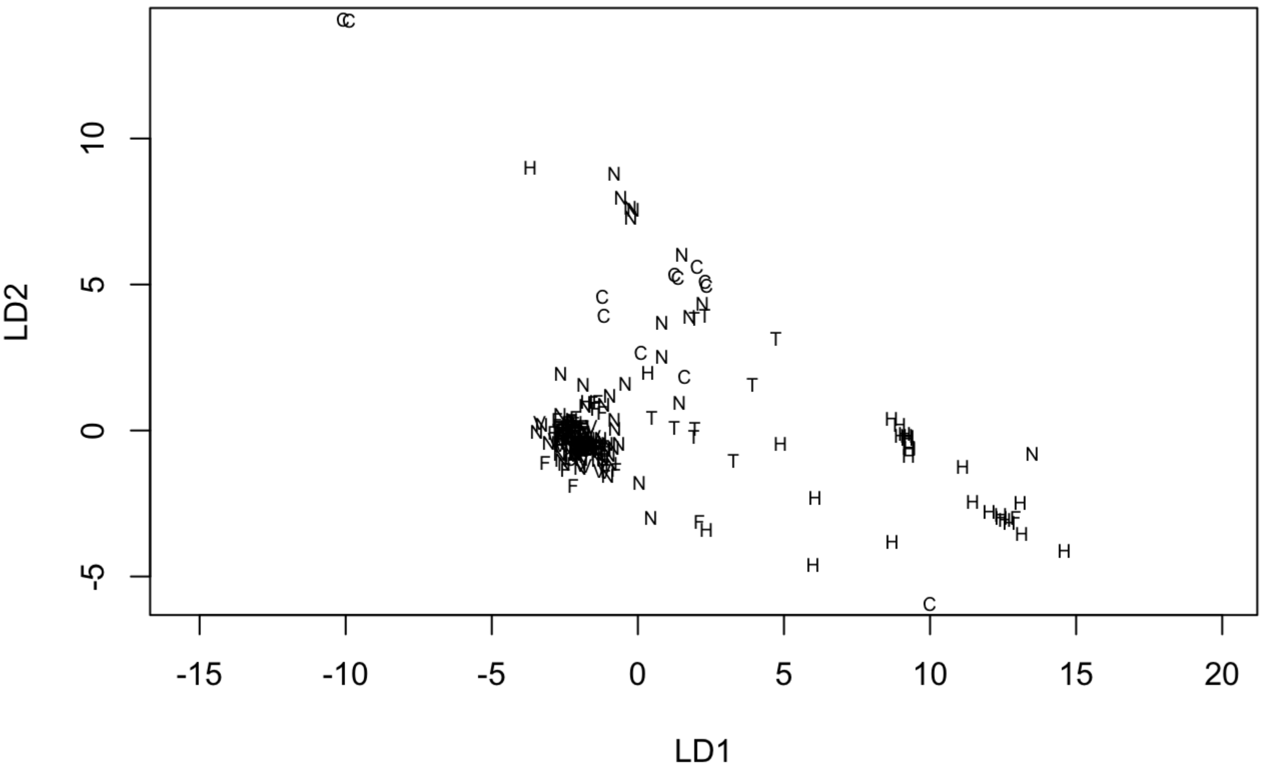
```
fgl.ld <- predict(lda(type ~ ., fgl), dimen = 2)$x
eqscplot(fgl.ld, type = "n", xlab = "LD1", ylab = "LD2")
# either
# for(i in seq(along = levels(fgl$type))) {
#    set <- fgl$type[-40] == levels(fgl$type)[i]
#    points(fgl.ld[set,], pch = 18, cex = 0.6, col = 2 + i)}
# key(text = list(levels(fgl$type), col = 3:8))
# or
text(fgl.ld, cex = 0.6,
     labels = c("F", "N", "V", "C", "T", "H")[fgl$type[-40]])
```




```
fgl.rld <- predict(lda(type ~ ., fgl, method = "t"), dimen = 2)$x
```

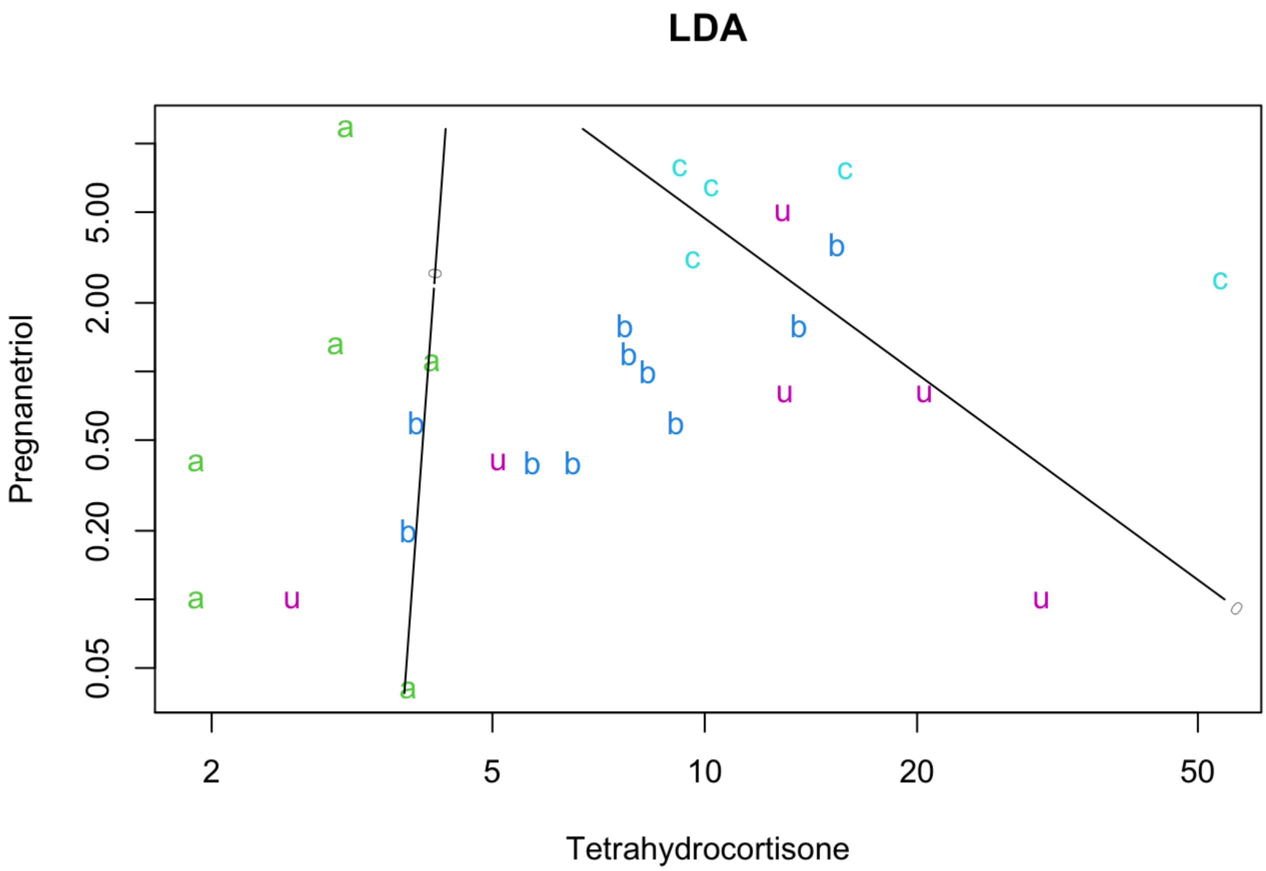
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   0.121   0.647   0.752   0.708   0.851   0.950 
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##  0.0333  0.4944  0.6223  0.5890  0.7593  0.9221 
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##  0.0111  0.3803  0.5441  0.5164  0.7016  0.8920 
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
## 0.00604 0.32897 0.48659 0.46824 0.64732 0.86836 
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
## 0.00435 0.28798 0.44940 0.43588 0.61783 0.86009 
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
## 0.00359 0.25104 0.42103 0.41390 0.59610 0.85347 
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
## 0.00317 0.21737 0.40133 0.39877 0.58171 0.84839 
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
## 0.00292 0.20372 0.38720 0.38825 0.57197 0.84461 
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
## 0.00275 0.19841 0.37973 0.38090 0.56414 0.84183 
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
## 0.00264 0.19387 0.37605 0.37573 0.55619 0.83980 
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
## 0.00256 0.18749 0.37425 0.37207 0.55012 0.83832 
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
## 0.00249 0.18381 0.37024 0.36942 0.55144 0.83724 
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
## 0.00245 0.17915 0.37041 0.36744 0.55115 0.83642 
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
## 0.00241 0.17724 0.36813 0.36590 0.54934 0.83579 
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
## 0.00238 0.17570 0.36631 0.36465 0.54789 0.83529 
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
## 0.00235 0.17444 0.36483 0.36360 0.54670 0.83486 
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
## 0.00233 0.17339 0.36361 0.36269 0.54570 0.83450 
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
## 0.00231 0.17250 0.36257 0.36190 0.54492 0.83418
```

```
fgl.rld[,1]<--fgl.rld[,1]
eqscplot(fgl.rld, type = "n", xlab = "LD1", ylab = "LD2")
# either
# for(i in seq(along = levels(fgl$type))) {
#   set <- fgl$type[-40] == levels(fgl$type)[i]
#   points(fgl.rld[set,], pch = 18, cex = 0.6, col = 2 + i)}
# key(text = list(levels(fgl$type), col = 3:8))
# or
text(fgl.rld, cex = 0.6,
     labels = c("F", "N", "V", "C", "T", "H")[fgl$type[-40]])
```

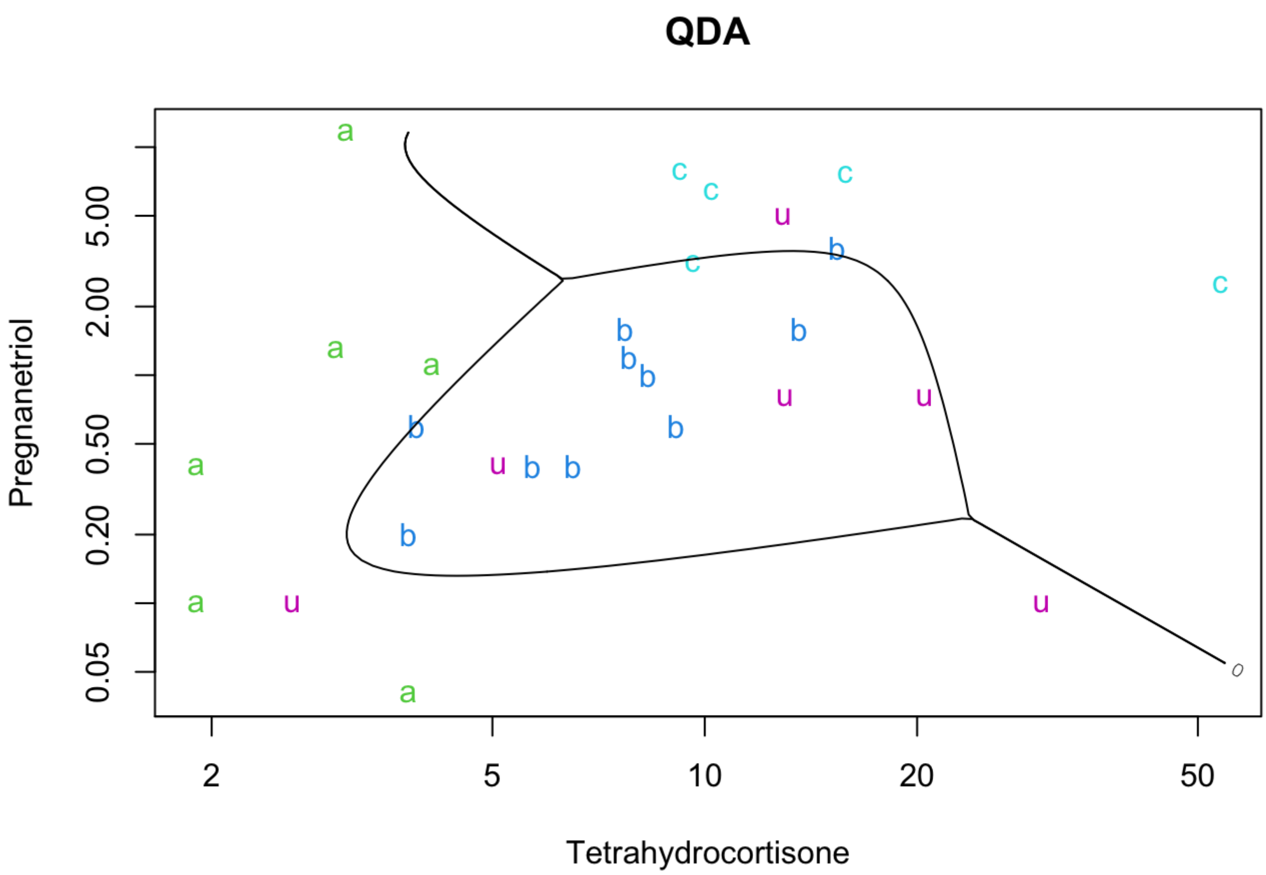


12.2 Classification theory

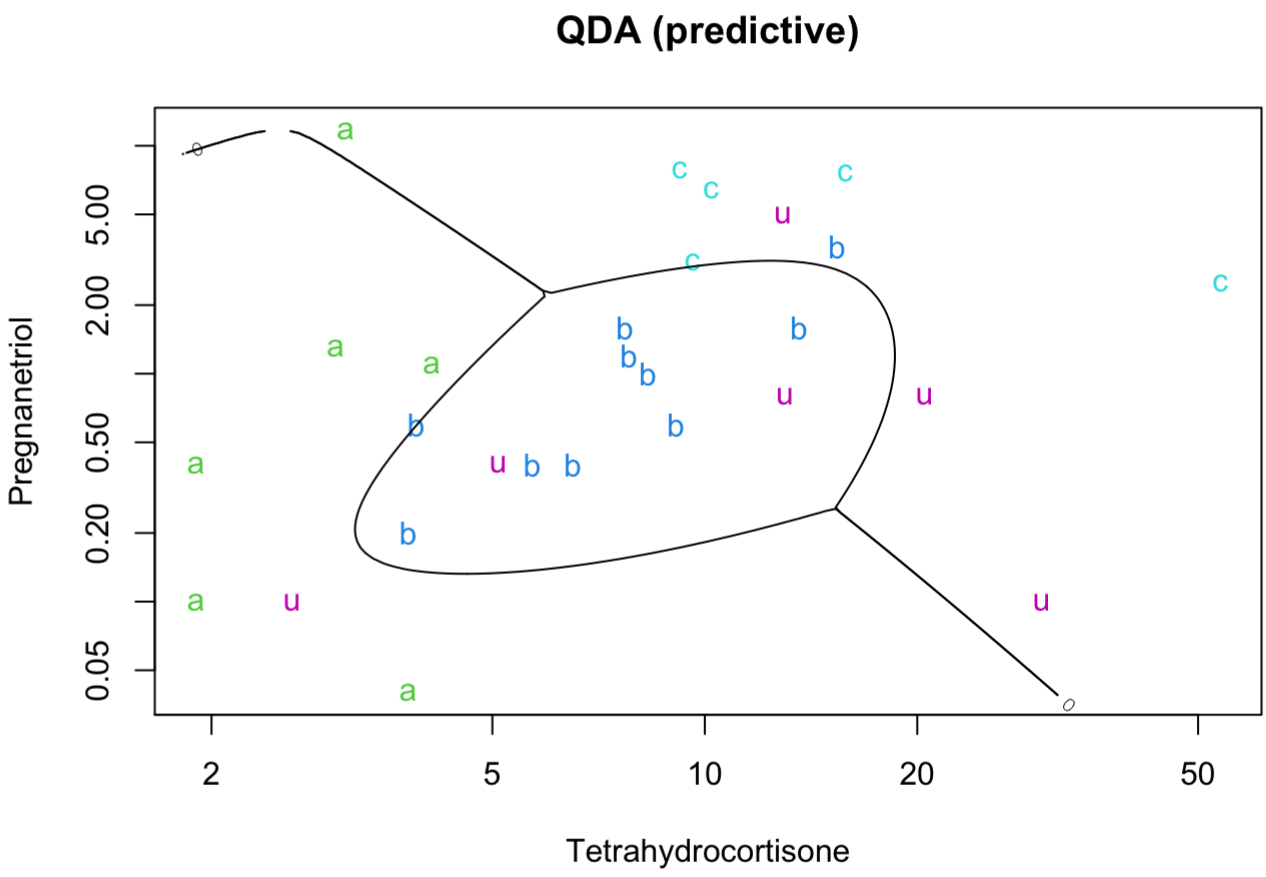
```
#install.packages("tree")
#decrease len if you have little memory.
predplot <- function(object, main="", len = 100, ...)
{
    plot(Cushings[,1], Cushings[,2], log="xy", type="n",
         xlab = "Tetrahydrocortisone", ylab = "Pregnanetriol", main = main)
    for(il in 1:4) {
        set <- Cushings$Type==levels(Cushings$Type)[il]
        text(Cushings[set, 1], Cushings[set, 2],
             labels=as.character(Cushings$Type[set]), col = 2 + il) }
    xp <- seq(0.6, 4.0, length=len)
    yp <- seq(-3.25, 2.45, length=len)
    cushT <- expand.grid(Tetrahydrocortisone = xp,
                         Pregnanetriol = yp)
    Z <- predict(object, cushT, ...); zp <- as.numeric(Z$class)
    zp <- Z$post[,3] - pmax(Z$post[,2], Z$post[,1])
    contour(exp(xp), exp(yp), matrix(zp, len),
            add = TRUE, levels = 0, labex = 0)
    zp <- Z$post[,1] - pmax(Z$post[,2], Z$post[,3])
    contour(exp(xp), exp(yp), matrix(zp, len),
            add = TRUE, levels = 0, labex = 0)
    invisible()
}

cushplot <- function(xp, yp, Z)
{
    plot(Cushings[, 1], Cushings[, 2], log = "xy", type = "n",
         xlab = "Tetrahydrocortisone", ylab = "Pregnanetriol")
    for(il in 1:4) {
        set <- Cushings$Type==levels(Cushings$Type)[il]
        text(Cushings[set, 1], Cushings[set, 2],
             labels = as.character(Cushings$Type[set]), col = 2 + il) }
    zp <- Z[, 3] - pmax(Z[, 2], Z[, 1])
    contour(exp(xp), exp(yp), matrix(zp, np),
            add = TRUE, levels = 0, labex = 0)
    zp <- Z[, 1] - pmax(Z[, 2], Z[, 3])
    contour(exp(xp), exp(yp), matrix(zp, np),
            add = TRUE, levels = 0, labex = 0)
    invisible()
}

cush <- log(as.matrix(Cushings[, -3]))
tp <- Cushings$Type[1:21, drop = TRUE]
cush.lda <- lda(cush[1:21,], tp); predplot(cush.lda, "LDA")
```



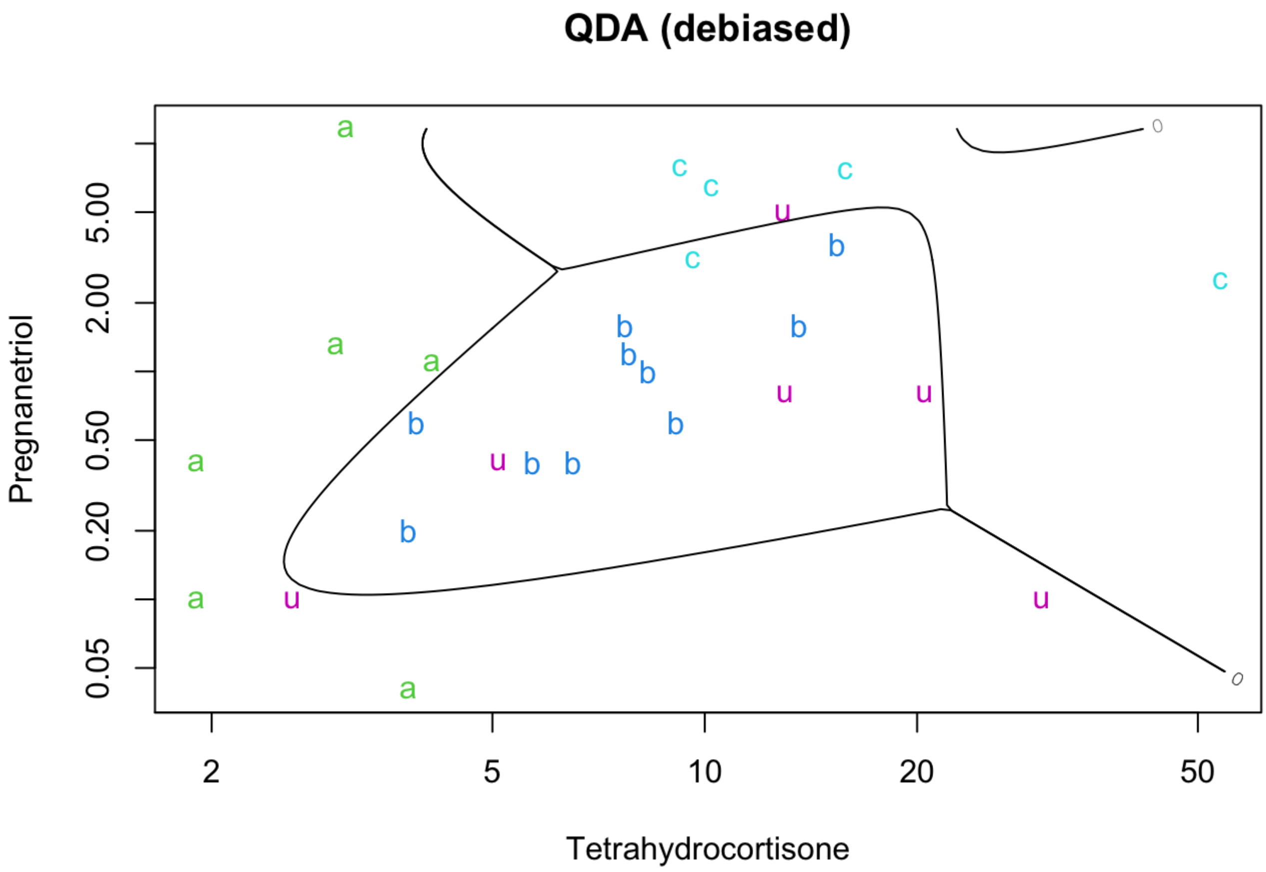

```
cush.qda <- qda(cush[1:21,], tp); predplot(cush.qda, "QDA")
```



```
predplot(cush.qda, "QDA (predictive)", method = "predictive")
```



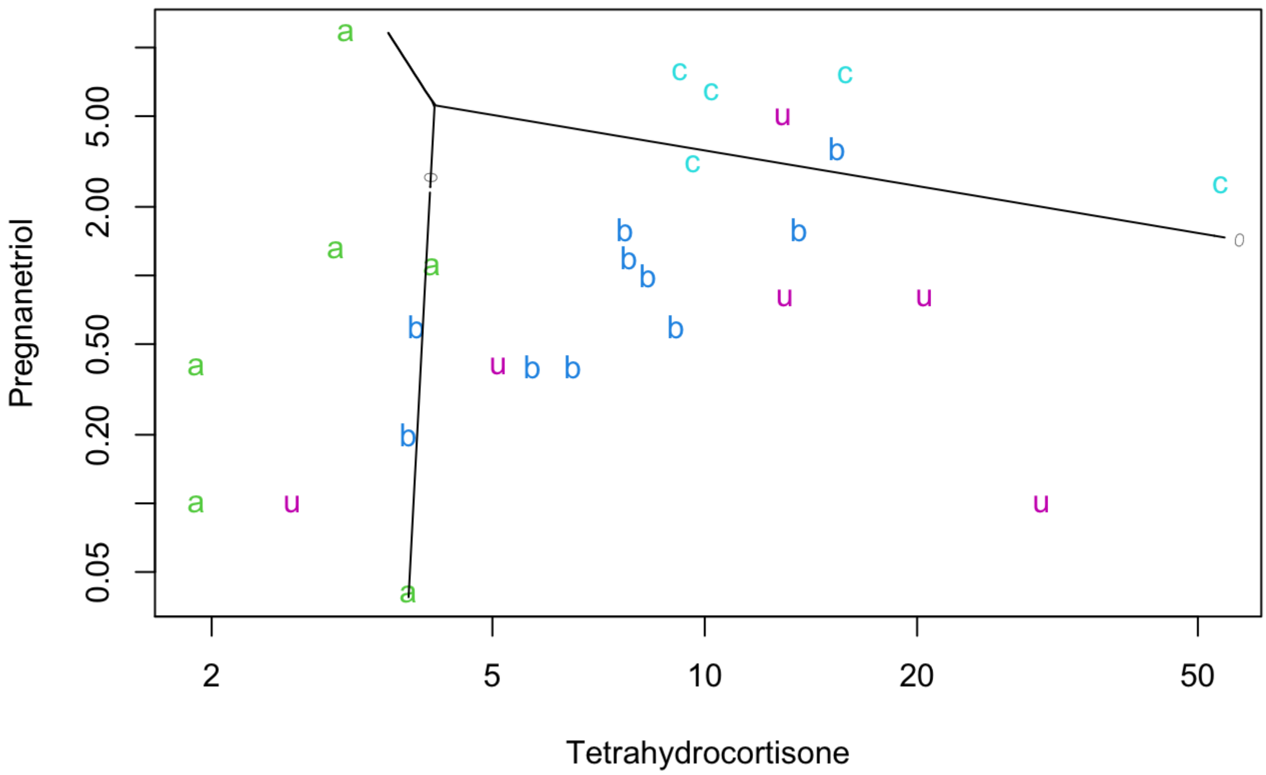

```
predplot(cush.qda, "QDA (debiased)", method = "debiased")
```



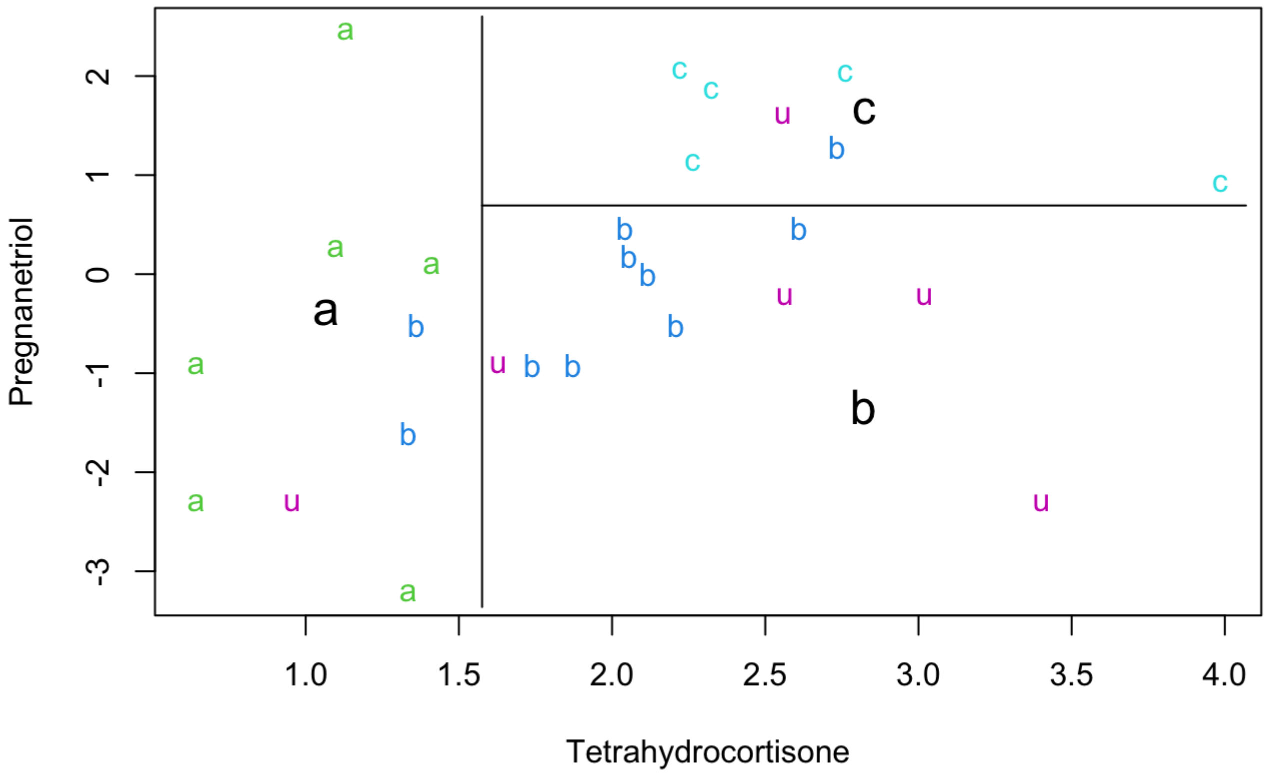

```
Cf <- data.frame(tp = tp,
  Tetrahydrocortisone = log(Cushings[1:21, 1]),
  Pregnanetriol = log(Cushings[1:21, 2]) )
cush.multinom <- multinom(tp ~ Tetrahydrocortisone
  + Pregnanetriol, Cf, maxit = 250)
```

```
## # weights:  12 (6 variable)
## initial  value 23.070858
## iter  10 value 6.623970
## iter  20 value 6.214841
## iter  30 value 6.182968
## iter  40 value 6.172650
## iter  50 value 6.167699
## iter  60 value 6.162723
## iter  70 value 6.156685
## iter  80 value 6.155298
## iter  90 value 6.153807
## iter 100 value 6.152597
## iter 110 value 6.152041
## iter 120 value 6.151229
## final  value 6.151167
## converged
```

```
xp <- seq(0.6, 4.0, length = 100); np <- length(xp)
yp <- seq(-3.25, 2.45, length = 100)
cushT <- expand.grid(Tetrahydrocortisone = xp,
                     Pregnanetriol = yp)
Z <- predict(cush.multinom, cushT, type = "probs")
cushplot(xp, yp, Z)
```

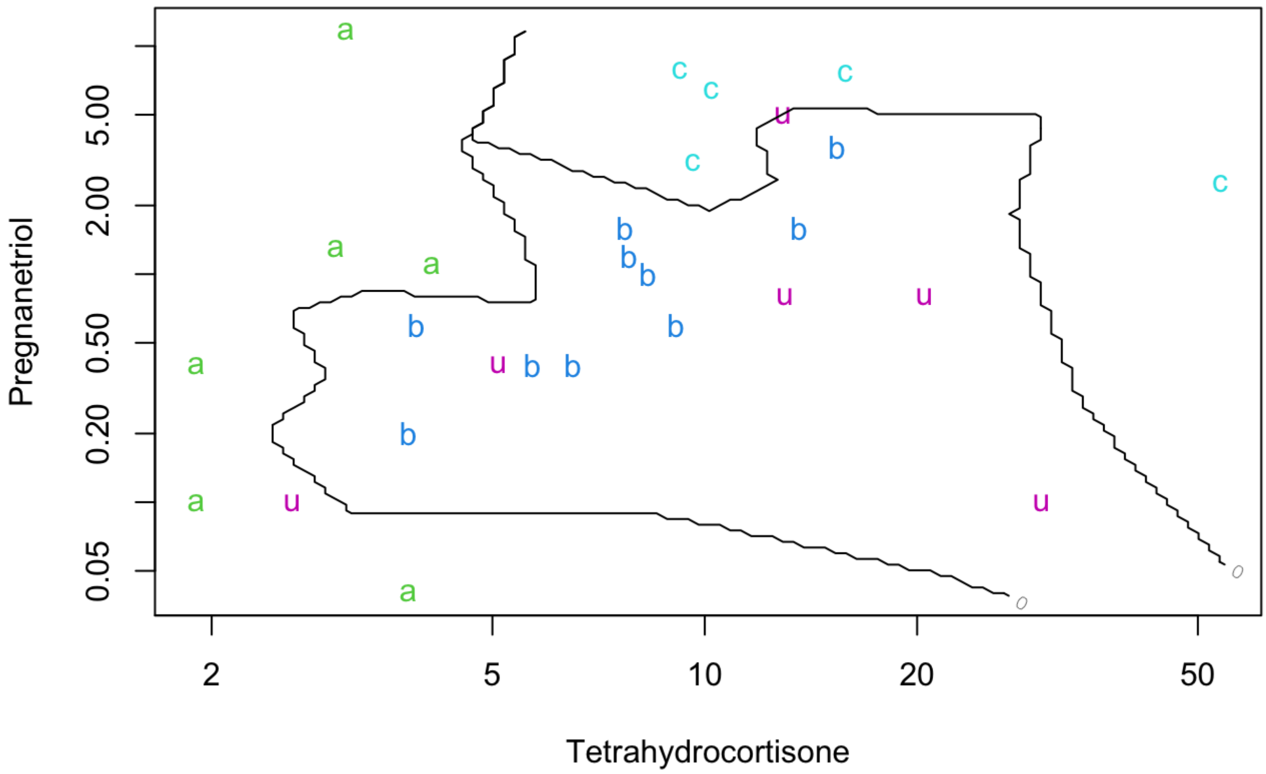



```
library(tree)
cush.tr <- tree(tp ~ Tetrahydrocortisone + Pregnanetriol, Cf)
plot(cush[, 1], cush[, 2], type = "n",
    xlab = "Tetrahydrocortisone", ylab = "Pregnanetriol")
for(il in 1:4) {
 set <- Cushings$Type==levels(Cushings$Type)[il]
 text(cush[set, 1], cush[set, 2],
      labels = as.character(Cushings$Type[set]), col = 2 + il) }
par(cex = 1.5); partition.tree(cush.tr, add = TRUE); par(cex = 1)
```

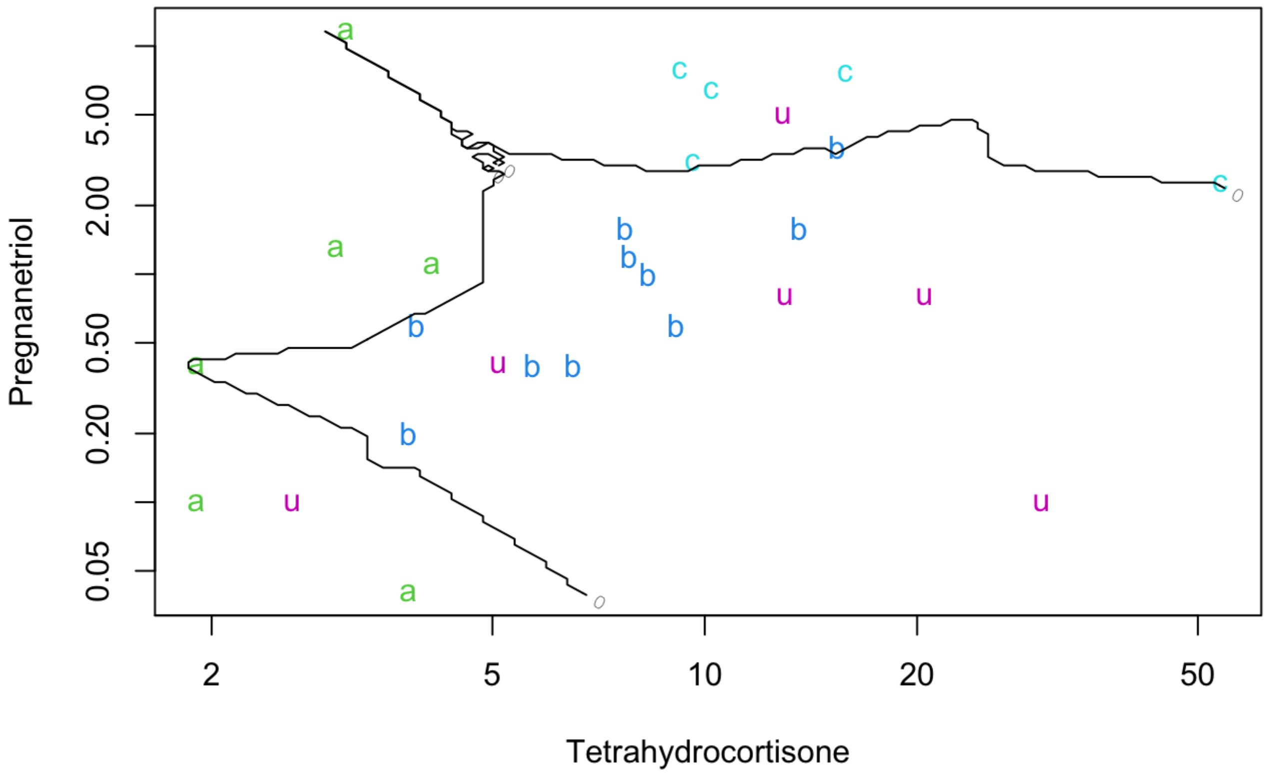


12.3 Non-parametric rules

```
Z <- knn(scale(cush[1:21, ], FALSE, c(3.4, 5.7)),
        scale(cushT, FALSE, c(3.4, 5.7)), tp)
cushplot(xp, yp, class.ind(Z))
```




```
Z <- knn(scale(cush[1:21, ], FALSE, c(3.4, 5.7)),
         scale(cushT, FALSE, c(3.4, 5.7)), tp, k = 3)
cushplot(xp, yp, class.ind(Z))
```



12.4 Neural networks

```r
pltnn <- function(main, ...) {
   plot(Cushings[,1], Cushings[,2], log="xy", type="n",
   xlab="Tetrahydrocortisone", ylab = "Pregnanetriol", main=main, ...)
   for(il in 1:4) {
       set <- Cushings$Type==levels(Cushings$Type)[il]
       text(Cushings[set, 1], Cushings[set, 2],
          as.character(Cushings$Type[set]), col = 2 + il) }
}

plt.bndry <- function(size=0, decay=0, ...)
{
   cush.nn <- nnet(cush, tpi, skip=TRUE, softmax=TRUE, size=size,
      decay=decay, maxit=1000)
   invisible(b1(predict(cush.nn, cushT), ...))
}

b1 <- function(Z, ...)
{
   zp <- Z[,3] - pmax(Z[,2], Z[,1])
   contour(exp(xp), exp(yp), matrix(zp, np),
      add=TRUE, levels=0, labex=0, ...)
   zp <- Z[,1] - pmax(Z[,3], Z[,2])
   contour(exp(xp), exp(yp), matrix(zp, np),
      add=TRUE, levels=0, labex=0, ...)
}

cush <- cush[1:21,]; tpi <- class.ind(tp)
# functions pltnn and plt.bndry given in the scripts
par(mfrow = c(2, 2))
pltnn("Size = 2")
set.seed(1); plt.bndry(size = 2, col = 2)
```

```
## # weights:  21
## initial  value 22.698541
## iter  10 value 4.299988
## iter  20 value 0.054092
## final  value 0.000064
## converged
```

```r
set.seed(3); plt.bndry(size = 2, col = 3)
```

```
## # weights:  21
## initial  value 33.677735
## iter  10 value 6.297601
## iter  20 value 1.699851
## iter  30 value 0.001510
## final  value 0.000069
## converged
```

```r
plt.bndry(size = 2, col = 4)
```

```
## # weights:  21
## initial  value 31.271486
## iter  10 value 4.332450
## iter  20 value 3.135307
## iter  30 value 0.023641
## final  value 0.000056
## converged
```

```
pltnn("Size = 2, lambda = 0.001")
set.seed(1); plt.bndry(size = 2, decay = 0.001, col = 2)
```

```
## # weights:  21
## initial  value 22.702079
## iter  10 value 4.569453
## iter  20 value 2.275944
## iter  30 value 1.171788
## iter  40 value 1.066464
## iter  50 value 1.043419
## iter  60 value 1.037356
## iter  70 value 1.030035
## iter  80 value 1.027478
## iter  90 value 1.026315
## iter 100 value 1.026099
## iter 110 value 1.025559
## iter 120 value 1.024550
## iter 130 value 1.023930
## iter 140 value 1.023885
## iter 150 value 1.023864
## iter 160 value 1.023856
## iter 170 value 1.023853
## iter 180 value 1.023852
## iter 190 value 1.023852
## final  value 1.023852
## converged
```

```
set.seed(2); plt.bndry(size = 2, decay = 0.001, col = 4)
```

```
## # weights:  21
## initial  value 28.510723
## iter  10 value 6.337890
## iter  20 value 3.386947
## iter  30 value 1.346611
## iter  40 value 1.044963
## iter  50 value 0.984966
## iter  60 value 0.913738
## iter  70 value 0.906185
## iter  80 value 0.903848
## iter  90 value 0.903330
## iter 100 value 0.903241
## iter 110 value 0.903210
## iter 120 value 0.903199
## iter 130 value 0.903198
## final  value 0.903198
## converged
```

```
pltnn("Size = 2, lambda = 0.01")
set.seed(1); plt.bndry(size = 2, decay = 0.01, col = 2)
```

```
## # weights:  21
## initial  value 22.733924
## iter  10 value 6.255641
## iter  20 value 5.007026
## iter  30 value 4.575414
## iter  40 value 4.553713
## iter  50 value 4.550005
## iter  60 value 4.545253
## iter  70 value 4.543880
## iter  80 value 4.543630
## final  value 4.543619
## converged
```

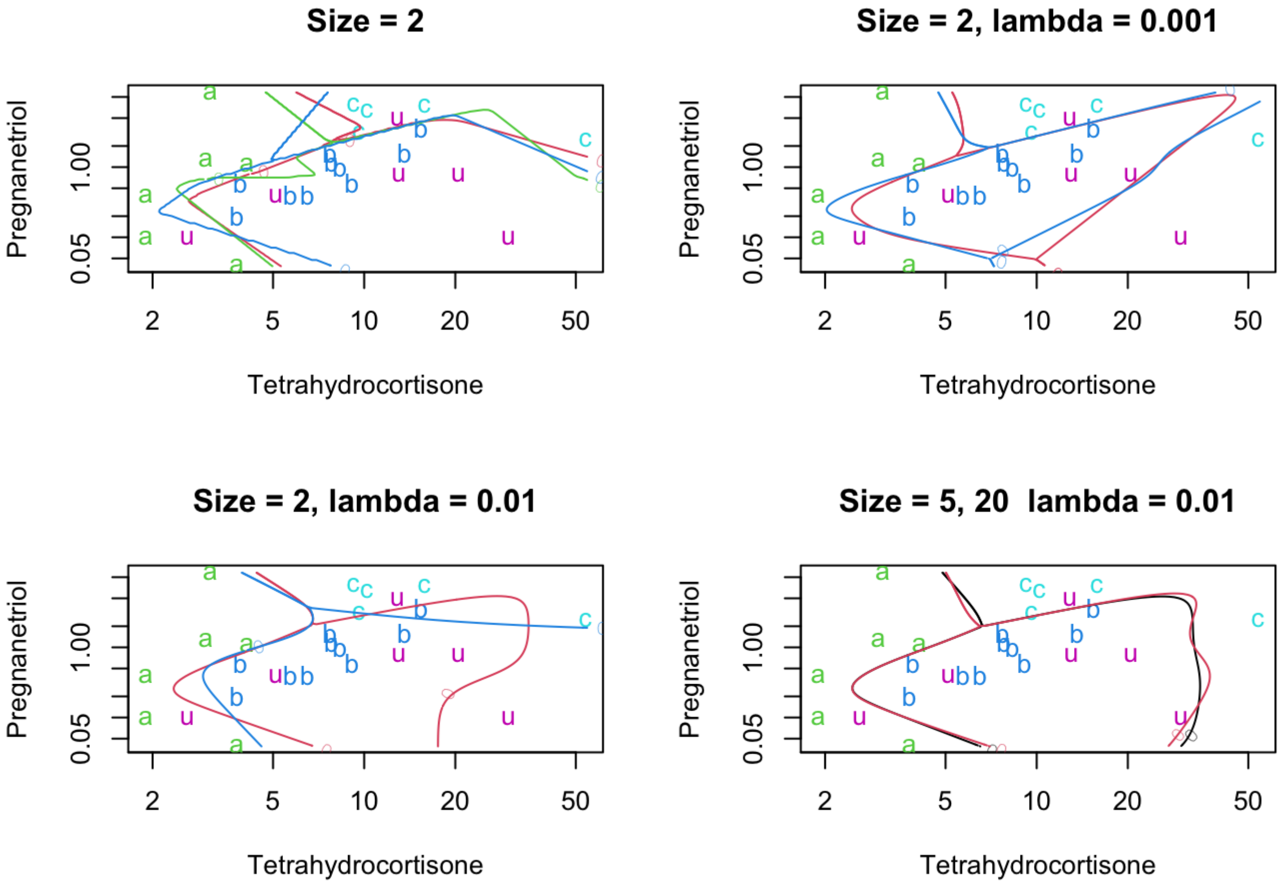
```
set.seed(2); plt.bndry(size = 2, decay = 0.01, col = 4)
```

```
## # weights:  21
## initial  value 28.542271
## iter  10 value 8.344781
## iter  20 value 5.841780
## iter  30 value 5.760888
## iter  40 value 5.739614
## iter  50 value 5.738457
## iter  60 value 5.738136
## iter  70 value 5.738102
## final  value 5.738101
## converged
```

```
pltnn("Size = 5, 20  lambda = 0.01")
set.seed(2); plt.bndry(size = 5, decay = 0.01, col = 1)
```

```
## # weights:  39
## initial  value 36.106392
## iter  10 value 7.345972
## iter  20 value 5.442419
## iter  30 value 5.198188
## iter  40 value 5.153866
## iter  50 value 5.102754
## iter  60 value 4.994906
## iter  70 value 4.520219
## iter  80 value 4.054997
## iter  90 value 3.995853
## iter 100 value 3.950100
## iter 110 value 3.941740
## iter 120 value 3.941275
## iter 130 value 3.941228
## iter 140 value 3.941173
## final  value 3.941163
## converged
```

```
set.seed(2); plt.bndry(size = 20, decay = 0.01, col = 2)
```

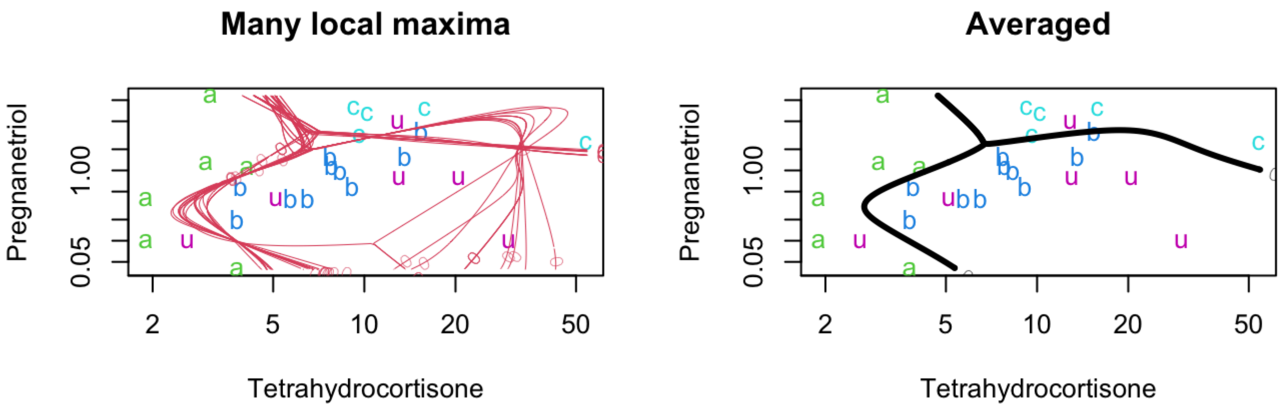


```
## # weights:  129
## initial  value 34.107768
## iter  10 value 6.103233
## iter  20 value 4.369676
## iter  30 value 3.893782
## iter  40 value 3.764147
## iter  50 value 3.698612
## iter  60 value 3.661172
## iter  70 value 3.642452
## iter  80 value 3.626142
## iter  90 value 3.612999
## iter 100 value 3.602778
## iter 110 value 3.597075
## iter 120 value 3.595404
## iter 130 value 3.594683
## iter 140 value 3.593182
## iter 150 value 3.588093
## iter 160 value 3.582150
## iter 170 value 3.577262
## iter 180 value 3.573303
## iter 190 value 3.572572
## iter 200 value 3.572339
## iter 210 value 3.572244
## iter 220 value 3.572184
## iter 230 value 3.572113
## iter 240 value 3.572047
## iter 250 value 3.572013
## iter 260 value 3.572004
## iter 260 value 3.572004
## final  value 3.572004
## converged
```

```
# functions pltnn and b1 are in the scripts
pltnn("Many local maxima")
Z <- matrix(0, nrow(cushT), ncol(tpi))
for(iter in 1:20) {
   set.seed(iter)
   cush.nn <- nnet(cush, tpi, skip = TRUE, softmax = TRUE, size = 3,
       decay = 0.01, maxit = 1000, trace = FALSE)
   Z <- Z + predict(cush.nn, cushT)
   cat("final value", format(round(cush.nn$value,3)), "\n")
   b1(predict(cush.nn, cushT), col = 2, lwd = 0.5)
}
```

```
## final value 5.296
## final value 5.349
## final value 5.724
## final value 4.053
## final value 5.741
## final value 5.724
## final value 4.177
## final value 5.257
## final value 4.073
## final value 4.126
## final value 4.08
## final value 5.724
## final value 5.842
## final value 5.349
## final value 4.126
## final value 3.997
## final value 5.282
## final value 4.113
## final value 4.143
## final value 4.126
```

```
pltnn("Averaged")
b1(Z, lwd = 3)
```



12.5 Support vector machines

```
library(e1071)
crabs.svm <- svm(crabs$sp ~ ., data = lcrabs, cost = 100, gamma = 1)
table(true = crabs$sp, predicted = predict(crabs.svm, lcrabs))
```

```
##     predicted
## true   B   O
##    B 100   0
##    O   0 100
```

```
svm(crabs$sp ~ ., data = lcrabs, cost = 100, gamma = 1, cross = 10)
```

```
##
## Call:
## svm(formula = crabs$sp ~ ., data = lcrabs, cost = 100,
##     gamma = 1, cross = 10)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  100
##
## Number of Support Vectors:  42
```

12.6 Forensic glass example

```
set.seed(123)
# dump random partition from S-PLUS
rand <- c(9, 6, 7, 10, 8, 8, 2, 2, 10, 1, 5, 2, 3, 8, 6, 8, 2, 6, 4,
4, 6, 1, 3, 2, 5, 5, 5, 3, 1, 9, 10, 2, 8, 2, 1, 6, 2, 7, 7, 8, 4, 1,
9, 5, 5, 1, 4, 6, 8, 6, 5, 7, 9, 2, 1, 1, 10, 9, 7, 6, 4, 7, 4, 8, 9,
9, 1, 8, 9, 5, 3, 3, 4, 8, 8, 6, 6, 9, 3, 10, 3, 10, 6, 6, 5, 10, 10,
2, 10, 6, 1, 4, 7, 8, 9, 10, 7, 10, 8, 4, 6, 8, 9, 10, 1, 9, 10, 6, 8,
4, 10, 8, 2, 10, 2, 3, 10, 1, 5, 9, 4, 4, 8, 2, 7, 6, 4, 8, 10, 4, 8,
10, 6, 10, 4, 9, 4, 1, 6, 5, 3, 2, 4, 1, 3, 4, 8, 4, 3, 7, 2, 5, 4, 5,
10, 7, 4, 2, 6, 3, 2, 2, 8, 4, 10, 8, 10, 2, 10, 6, 5, 2, 3, 2, 6, 2,
7, 7, 8, 9, 7, 10, 8, 6, 7, 9, 7, 10, 3, 2, 7, 5, 6, 1, 3, 9, 7, 7, 1,
8, 7, 8, 8, 8, 10, 4, 5, 9, 4, 6, 9, 6, 10, 2)

con <- function(...)
{
    print(tab <- table(...))
    diag(tab) <- 0
    cat("error rate = ",
        round(100*sum(tab)/length(list(...)[[1]]), 2), "%\n")
    invisible()
}
CVtest <- function(fitfn, predfn, ...)
{
    res <- fgl$type
    for (i in sort(unique(rand))) {
        cat("fold ", i, "\n", sep = "")
        learn <- fitfn(rand != i, ...)
        res[rand == i] <- predfn(learn, rand == i)
    }
    res
}
res.multinom <- CVtest(
  function(x, ...) multinom(type ~ ., fgl[x, ], ...),
  function(obj, x) predict(obj, fgl[x, ], type = "class"),
  maxit = 1000, trace = FALSE)
```

```
## fold 1
## fold 2
## fold 3
## fold 4
## fold 5
## fold 6
## fold 7
## fold 8
## fold 9
## fold 10
```

```
con(true = fgl$type, predicted = res.multinom)
```

```
##        predicted
## true    WinF WinNF Veh Con Tabl Head
##   WinF    44    20   4   0    2    0
##   WinNF   20    50   0   3    2    1
##   Veh      9     7   1   0    0    0
##   Con      0     4   0   8    0    1
##   Tabl     0     2   0   0    4    3
##   Head     1     1   0   3    1   23
## error rate =  39.25 %
```

```
res.lda <- CVtest(
  function(x, ...) lda(type ~ ., fgl[x, ], ...),
  function(obj, x) predict(obj, fgl[x, ])$class )
```

```
## fold 1
## fold 2
## fold 3
## fold 4
## fold 5
## fold 6
## fold 7
## fold 8
## fold 9
## fold 10
```

```
con(true = fgl$type, predicted = res.lda)
```

```
##        predicted
## true    WinF WinNF Veh Con Tabl Head
##   WinF    49    18   3   0    0    0
##   WinNF   21    50   0   2    2    1
##   Veh     10     7   0   0    0    0
##   Con      0     6   0   6    0    1
##   Tabl     1     2   0   0    4    2
##   Head     2     0   0   2    0   25
## error rate =  37.38 %
```

```
fgl0 <- fgl[ , -10] # drop type
{ res <- fgl$type
  for (i in sort(unique(rand))) {
      cat("fold ", i ,"\n", sep = "")
      sub <- rand == i
      res[sub] <- knn(fgl0[!sub, ], fgl0[sub, ], fgl$type[!sub],
                      k = 1)
  }
  res } -> res.knn1
```

```
## fold 1
## fold 2
## fold 3
## fold 4
## fold 5
## fold 6
## fold 7
## fold 8
## fold 9
## fold 10
```

```
con(true = fgl$type, predicted = res.knn1)
```

```
##        predicted
## true    WinF WinNF Veh Con Tabl Head
##   WinF    59     6   5   0    0    0
##   WinNF   12    57   3   3    1    0
##   Veh      2     4  11   0    0    0
##   Con      0     2   0   8    1    2
##   Tabl     1     0   0   1    6    1
##   Head     0     4   1   1    1   22
## error rate =  23.83 %
```

```
res.lb <- knn(fgl0, fgl0, fgl$type, k = 3, prob = TRUE, use.all = FALSE)
table(attr(res.lb, "prob"))
```

```
##
## 0.333333333333333 0.666666666666667                 1
##                10                64               140
```

```
library(rpart)
res.rpart <- CVtest(
  function(x, ...) {
    tr <- rpart(type ~ ., fgl[x,], ...)
    cp <- tr$cptable
    r <- cp[, 4] + cp[, 5]
    rmin <- min(seq(along = r)[cp[, 4] < min(r)])
    cp0 <- cp[rmin, 1]
    cat("size chosen was", cp[rmin, 2] + 1, "\n")
    prune(tr, cp = 1.01*cp0)
  },
  function(obj, x)
    predict(obj, fgl[x, ], type = "class"),
  cp = 0.001
)
```

```
## fold 1
## size chosen was 5
## fold 2
## size chosen was 7
## fold 3
## size chosen was 5
## fold 4
## size chosen was 5
## fold 5
## size chosen was 7
## fold 6
## size chosen was 8
## fold 7
## size chosen was 5
## fold 8
## size chosen was 7
## fold 9
## size chosen was 5
## fold 10
## size chosen was 5
```

```
con(true = fgl$type, predicted = res.rpart)
```

```
##        predicted
## true    WinF WinNF Veh Con Tabl Head
##   WinF    53    15   1   0    0    1
##   WinNF   18    52   1   4    0    1
##   Veh     11     5   1   0    0    0
##   Con      0     1   0  11    0    1
##   Tabl     2     3   0   4    0    0
##   Head     1     1   1   0    0   26
## error rate =  33.18 %
```

```
fgl1 <- fgl
fgl1[1:9] <- lapply(fgl[, 1:9], function(x)
               {r <- range(x); (x - r[1])/diff(r)})

CVnn2 <- function(formula, data,
                  size = rep(6,2), lambda = c(0.001, 0.01),
                  nreps = 1, nifold = 5, verbose = 99, ...)
{
    CVnn1 <- function(formula, data, nreps=1, ri, verbose,  ...)
    {
        truth <- data[,deparse(formula[[2]])]
        res <-  matrix(0, nrow(data), length(levels(truth)))
        if(verbose > 20) cat("  inner fold")
        for (i in sort(unique(ri))) {
            if(verbose > 20) cat(" ", i,  sep="")
            for(rep in 1:nreps) {
                learn <- nnet(formula, data[ri !=i,], trace = FALSE, ...)
                res[ri == i,] <- res[ri == i,] +
                    predict(learn, data[ri == i,])
            }
        }
        if(verbose > 20) cat("\n")
        sum(as.numeric(truth) != max.col(res/nreps))
    }
    truth <- data[,deparse(formula[[2]])]
    res <-  matrix(0, nrow(data), length(levels(truth)))
    choice <- numeric(length(lambda))
    for (i in sort(unique(rand))) {
        if(verbose > 0) cat("fold ", i,"\n", sep="")
        ri <- sample(nifold, sum(rand!=i), replace=TRUE)
        for(j in seq(along=lambda)) {
            if(verbose > 10)
                cat("  size =", size[j], "decay =", lambda[j], "\n")
            choice[j] <- CVnn1(formula, data[rand != i,], nreps=nreps,
                               ri=ri, size=size[j], decay=lambda[j],
                               verbose=verbose, ...)
        }
        decay <- lambda[which.is.max(-choice)]
        csize <- size[which.is.max(-choice)]
        if(verbose > 5) cat("  #errors:", choice, "  ") #
        if(verbose > 1) cat("chosen size = ", csize,
                            " decay = ", decay, "\n", sep="")
        for(rep in 1:nreps) {
            learn <- nnet(formula, data[rand != i,], trace=FALSE,
                          size=csize, decay=decay, ...)
            res[rand == i,] <- res[rand == i,] +
                predict(learn, data[rand == i,])
        }
    }
    factor(levels(truth)[max.col(res/nreps)], levels = levels(truth))
}

if(FALSE) { # only run this if you have time to wait
res.nn2 <- CVnn2(type ~ ., fgl1, skip = TRUE, maxit = 500, nreps = 10)
con(true = fgl$type, predicted = res.nn2)
```

```
}

res.svm <- CVtest(
  function(x, ...) svm(type ~ ., fgl[x, ], ...),
  function(obj, x) predict(obj, fgl[x, ]),
  cost = 100, gamma = 1 )
```

```
## fold 1
## fold 2
## fold 3
## fold 4
## fold 5
## fold 6
## fold 7
## fold 8
## fold 9
## fold 10
```

```
con(true = fgl$type, predicted = res.svm)
```

```
##        predicted
## true    WinF WinNF Veh Con Tabl Head
##   WinF    49    19   2   0    0    0
##   WinNF   17    55   3   0    0    1
##   Veh      6     7   4   0    0    0
##   Con      0     8   0   5    0    0
##   Tabl     1     5   0   0    3    0
##   Head     0     9   0   0    0   20
## error rate =  36.45 %
```

```
svm(type ~ ., data = fgl, cost = 100, gamma = 1, cross = 10)
```

```
##
## Call:
## svm(formula = type ~ ., data = fgl, cost = 100, gamma = 1,
##     cross = 10)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  100
##
## Number of Support Vectors:  172
```

```
cd0 <- lvqinit(fgl0, fgl$type, prior = rep(1, 6)/6, k = 3)
cd1 <- olvq1(fgl0, fgl$type, cd0)
con(true = fgl$type, predicted = lvqtest(cd1, fgl0))
```

```
##        predicted
## true    WinF WinNF Veh Con Tabl Head
##   WinF    60     9   1   0    0    0
##   WinNF    8    63   0   2    3    0
##   Veh      7     7   3   0    0    0
##   Con      0     1   0  11    0    1
##   Tabl     0     0   0   0    7    2
##   Head     4     0   0   0    0   25
## error rate =  21.03 %
```

```
CV.lvq <- function()
{
    res <- fgl$type
    for(i in sort(unique(rand))) {
        cat("doing fold", i, "\n")
        cd0 <- lvqinit(fgl0[rand != i,], fgl$type[rand != i],
                       prior = rep(1, 6)/6, k = 3)
        cd1 <- olvq1(fgl0[rand != i,], fgl$type[rand != i], cd0)
        cd1 <- lvq3(fgl0[rand != i,], fgl$type[rand != i],
                    cd1, niter = 10000)
        res[rand == i] <- lvqtest(cd1, fgl0[rand == i, ])
    }
    res
}
con(true = fgl$type, predicted = CV.lvq())
```

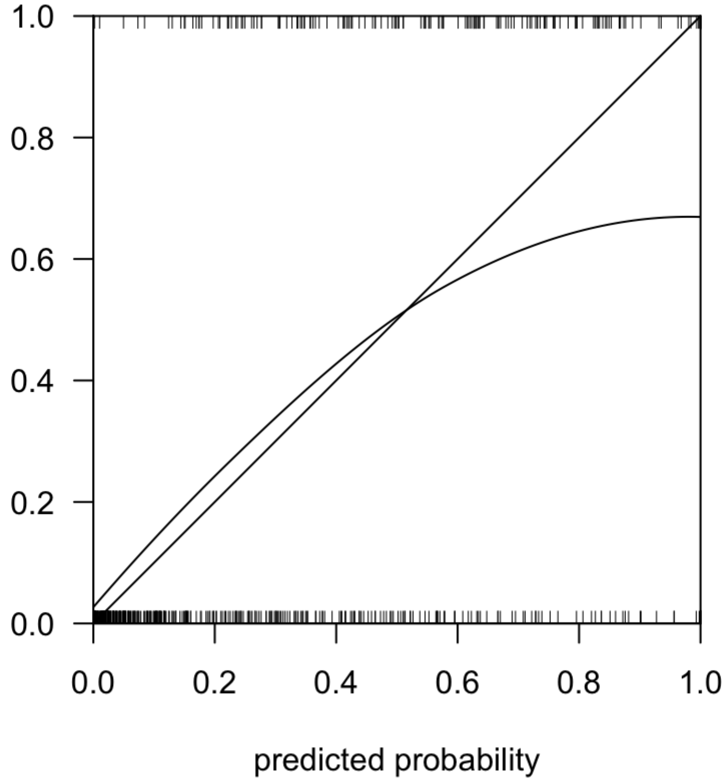
```
## doing fold 1
## doing fold 2
## doing fold 3
## doing fold 4
## doing fold 5
## doing fold 6
## doing fold 7
## doing fold 8
## doing fold 9
## doing fold 10
##        predicted
## true    WinF WinNF Veh Con Tabl Head
##   WinF    63     6   1   0    0    0
##   WinNF   12    57   1   6    0    0
##   Veh      5     9   3   0    0    0
##   Con      1     0   0  10    0    2
##   Tabl     1     0   0   0    6    2
##   Head     3     2   0   0    2   22
## error rate =  24.77 %
```

12.7 Calibration plots

```
CVprobs <- function(fitfn, predfn, ...)
{
    res <- matrix(nrow=214, ncol=6)
    for (i in sort(unique(rand))) {
        cat("fold ", i, "\n", sep = "")
        learn <- fitfn(rand != i, ...)
        res[rand == i, ] <- predfn(learn, rand == i)
    }
    res
}
probs.multinom <- CVprobs(
  function(x, ...) multinom(type ~ ., fgl[x, ], ...),
  function(obj, x) predict(obj, fgl[x, ], type = "probs"),
  maxit = 1000, trace = FALSE)
```

```
## fold 1
## fold 2
## fold 3
## fold 4
## fold 5
## fold 6
## fold 7
## fold 8
## fold 9
## fold 10
```

```
probs.yes <- as.vector(class.ind(fgl$type))
probs <- as.vector(probs.multinom)
par(pty = "s")
plot(c(0, 1), c(0, 1), type = "n", xlab = "predicted probability",
     ylab = "", xaxs = "i", yaxs = "i", las = 1)
rug(probs[probs.yes == 0], 0.02, side = 1, lwd = 0.5)
rug(probs[probs.yes == 1], 0.02, side = 3, lwd = 0.5)
abline(0, 1)
newp <- seq(0, 1, length = 100)
lines(newp, predict(loess(probs.yes ~ probs, span = 1), newp))
```




```
# End of ch12
```