Modern Data Analysis in S-PLUS

Brian Ripley and Bill Venables

Workshop, 7 October 1998

`lda` for S-PLUS 3.x

```
# file MASS/lda.q
# copyright (C) 1994-8 W. N. Venables and B. D. Ripley
#
lda <- function(x, ...)
{
  if(is.null(class(x))) class(x) <- data.class(x)
  UseMethod("lda")
}

lda.formula <- function(formula, data = NULL, ...,
                        subset, na.action = na.fail)
{
  m <- match.call(expand.dots = F)
  if(is.matrix(eval(m$data, sys.parent())))
    m$data <- as.data.frame(data)
  m$... <- NULL
  m[[1]] <- as.name("model.frame")
  m <- eval(m, sys.parent())
  Terms <- attr(m, "terms")
  grouping <- model.extract(m, "response")
  x <- model.matrix(Terms, m)
  xint <- match("(Intercept)", dimnames(x)[[2]], nomatch=0)
  if(xint > 0) x <- x[, -xint, drop=F]
  res <- lda.default(x, grouping, ...)
  res$terms <- Terms
  res$call <- match.call()
  res
}

lda.data.frame <- function(x, ...)
{
  res <- lda.matrix(data.matrix(x), ...)
  res$call <- match.call()
  res
}

lda.Matrix <- function(x, ...)
{
  res <- lda.matrix(as.matrix(x), ...)
  res$call <- match.call()
  res
}

lda.matrix <- function(x, grouping, ...,
                       subset, na.action = na.fail)
{
  if(!missing(subset)) {
    x <- x[subset, , drop = F]
```

```
    grouping <- grouping[subset]
  }
  if(!missing(na.action)) {
    dfr <- na.action(structure(list(g = grouping, x = x),
                               class = "data.frame"))
    grouping <- dfr$g
    x <- dfr$x
  }
  res <- NextMethod("lda")
  res$call <- match.call()
  res
}

lda.default <- function(x, grouping, prior = proportions, tol = 1.0e-4,
                        method = c("moment", "mle", "mve", "t"), CV=F,
                        nu = 5, ...)
{
  which.is.max <- function(x) {
    d <- (1:length(x))[x == max(x)]
    if(length(d) > 1) d <- sample(d, 1)
    d
  }

  if(is.null(dim(x))) stop("x is not a matrix")
  n <- nrow(x)
  p <- ncol(x)
  if(n != length(grouping)) stop("nrow(x) and length(grouping) are different")
  g <- as.factor(grouping)
  lev <- levels(g)
  counts <- table(g)
  if(any(counts == 0)) {
     warning(paste("group(s)", paste(levels(g)[counts == 0], collapse=" "),
                   "are empty"))
     g <- factor(g, levels=levels(g)[counts > 0])
     counts <- table(g)
  }
  proportions <- counts/n
  ng <- length(proportions)
  if(any(prior < 0) || round(sum(prior), 5) != 1) stop("invalid prior")
  if(length(prior) != ng) stop("prior is of incorrect length")
  dim(prior) <- dimnames(prior) <- NULL
  method <- match.arg(method)
  if(CV && !(method == "moment" || method == "mle"))
     stop(paste("Cannot use leave-one-out CV with method", method))
  group.means <- tapply(x, list(rep(g, p), col(x)), mean)
  f1 <- sqrt(diag(var(x - group.means[g,  ])))
  if(any(f1 < tol))
    stop(paste("variable(s)",
               paste(format((1:p)[f1 < tol]), collapse = " "),
               "appear to be constant within groups"))
```



```
# scale columns to unit variance before checking for collinearity
  scaling <- diag(1/f1,,p)
  if(method == "mve") {
# adjust to "unbiased" scaling of covariance matrix
    cov <- n/(n-ng) * cov.mve((x - group.means[g,  ]) %*% scaling, , F)$cov
    sX <- svd(cov, nu = 0)
    rank <- sum(sX$d > tol^2)
    if(rank < p) warning("variables are collinear")
    scaling <- scaling %*% sX$v[, 1:rank] %*%
                 diag(sqrt(1/sX$d[1:rank]),,rank)
  } else if(method == "t") {
    if(nu <= 2) stop("nu must exceed 2")
    w <- rep(1, n)
    repeat {
         w0 <- w
         X <- x - group.means[g, ]
         sX <- svd(sqrt((1 + p/nu)*w/n) * X, nu=0)
         X <- X %*% sX$v %*% diag(1/sX$d,, p)
         w <- 1/(1 + drop(X^2 %*% rep(1, p))/nu)
         print(summary(w))
         group.means <- tapply(w*x, list(rep(g, p), col(x)), sum)/
            tapply(w, g, sum)
         if(all(abs(w - w0) < 1e-2)) break
    }
    X <-  sqrt(nu/(nu-2)*(1 + p/nu)/n * w) * (x - group.means[g,  ]) %*% scaling
    X.s <- svd(X, nu = 0)
    rank <- sum(X.s$d > tol)
    if(rank < p) warning("variables are collinear")
    scaling <- scaling %*% X.s$v[, 1:rank] %*% diag(1/X.s$d[1:rank],,rank)
  } else {
    if(method == "moment") fac <- 1/(n-ng) else fac <- 1/n
    X <- sqrt(fac) * (x - group.means[g,  ]) %*% scaling
    X.s <- svd(X, nu = 0)
    rank <- sum(X.s$d > tol)
    if(rank < p) warning("variables are collinear")
    scaling <- scaling %*% X.s$v[, 1:rank] %*% diag(1/X.s$d[1:rank],,rank)
  }
# now have variables scaled so that W is the identity
  if(CV) {
    x <- x %*% scaling
    dm <- group.means %*% scaling
    K <- if(method == "moment") ng else 0
    dist <- matrix(0, n, ng)
    for(i in 1:ng) {
      dev <- x - matrix(dm[i,  ], n, p, byrow = T)
      dist[, i] <- apply(dev^2, 1, sum)
    }
    ind <- cbind(1:n, g)
    nc <- counts[g]
    cc <- nc/((nc-1)*(n-K))
```

```
    dist2 <- dist
    for(i in 1:ng) {
      dev <- x - matrix(dm[i,  ], n, p, byrow = T)
      dev2 <- x - dm[g, ]
      tmp <- apply(dev*dev2, 1, sum)
      dist[, i] <- (n-1-K)/(n-K) * (dist2[, i] +  cc*tmp^2/(1 - cc*dist2[ind]))
    }
    dist[ind] <- dist2[ind] * (n-1-K)/(n-K) * (nc/(nc-1))^2 /
      (1 - cc*dist2[ind])
    dist <- 0.5 * dist - matrix(log(prior), n, ng, byrow=T)
    dist <- exp(-(dist - min(dist, na.rm=T)))
    cl <- apply(dist, 1, which.is.max)
    levels(cl) <- lev
    class(cl) <- "factor"
    #  convert to posterior probabilities
    posterior <- dist/drop(dist %*% rep(1, length(prior)))
    dimnames(posterior) <- list(dimnames(x)[[1]], lev)
    return(list(class = cl, posterior = posterior))
  }
  xbar <- apply(prior %*% group.means, 2, sum)
  if(method == "mle") fac <-  1/ng else fac <- 1/(ng - 1)
  X <- sqrt((n * prior)*fac) * scale(group.means, xbar, F) %*% scaling
  X.s <- svd(X, nu = 0)
  rank <- sum(X.s$d > tol * X.s$d[1])
  scaling <- scaling %*% X.s$v[, 1:rank]
  if(is.null(dimnames(x)))
    dimnames(scaling) <- list(NULL, paste("LD", 1:rank, sep = ""))
  else {
    dimnames(scaling) <- list(dimnames(x)[[2]], paste("LD", 1:rank, sep = ""))
    dimnames(group.means)[[2]] <- dimnames(x)[[2]]
  }
  structure(list(prior = prior, counts = counts, means = group.means,
                 scaling = scaling, lev = lev, svd = X.s$d[1:rank],
                 N = n, call = match.call()),
            class = "lda")
}

predict.lda <- function(object, newdata, prior = object$prior, dimen,
                        method = c("plug-in", "predictive", "debiased"), ...)
{
  if(!inherits(object, "lda")) stop("object not of class lda")
  which.is.max <- function(x) {
    d <- (1:length(x))[x == max(x)]
    if(length(d) > 1) d <- sample(d, 1)
    d
  }
  if(!is.null(Terms <- object$terms)) { #
    # formula fit
    if(missing(newdata)) newdata <- model.frame.lm(object)
    else newdata <- model.frame(as.formula(delete.response(Terms)),
```



```
                                newdata, na.action=function(x) x)
    x <- model.matrix(delete.response(Terms), newdata)
    xint <- match("(Intercept)", dimnames(x)[[2]], nomatch=0)
    if(xint > 0) x <- x[, -xint, drop=F]
  } else { #
    # matrix or data-frame fit
    if(missing(newdata)) {
      xname <- eval(substitute(object$call$x))
      if(!is.null(sub <- object$call$subset))
        newdata <- eval(parse(text=paste(deparse(xname),"[",
                                deparse(sub),",]")), sys.parent())
      else newdata <- eval(parse(text=deparse(xname)), sys.parent())
      if(!is.null(nas <- object$call$na.action))
        newdata <- do.call(nas, list(newdata))
    }
    if(is.null(dim(newdata)))
      dim(newdata) <- c(1, length(newdata))  # a row vector
    x <- as.matrix(newdata)             # to cope with dataframes
  }

  if(dim(x)[2] != dim(object$means)[2]) stop("wrong number of variables")
  if(length(dimnames(x)[[2]]) > 0 &&
     any(dimnames(x)[[2]] != dimnames(object$means)[[2]]))
        warning("Variable names in newdata do not match those in object")
  ng <- length(prior)
#   remove overall means to keep distances small
  means <- apply(object$means, 2, mean)
  scaling <- object$scaling
  x <- scale(x, means, F) %*% scaling
  dm <- scale(object$means, means, F) %*% scaling
  method <- match.arg(method)
  if(missing(dimen)) dimen <- length(object$svd)
  else dimen <- min(dimen, length(object$svd))
  N <- object$N
  if(method == "plug-in") {
    dm <- dm[, 1:dimen, drop=F]
    dist <- matrix(0.5 * apply(dm^2, 1, sum) - log(prior), nrow(x),
                length(prior), byrow = T) - x[, 1:dimen, drop=F] %*% t(dm)
    dist <- exp( -(dist - min(dist, na.rm=T)))
  } else if (method == "debiased") {
    dm <- dm[, 1:dimen, drop=F]
    dist <- matrix(0.5 * apply(dm^2, 1, sum), nrow(x), ng, byrow = T) -
      x[, 1:dimen, drop=F] %*% t(dm)
    dist <- (N - ng - dimen - 1)/(N - ng) * dist -
      matrix(log(prior) - dimen/object$counts , nrow(x), ng, byrow=T)
    dist <- exp( -(dist - min(dist, na.rm=T)))
  } else { # predictive
    dist <- matrix(0, nrow = nrow(x), ncol = ng)
    p <- ncol(object$means)
# adjust to ML estimates of covariances
```

```
    X <- x * sqrt(N/(N-ng))
    for(i in 1:ng) {
      nk <- object$counts[i]
      dev <- scale(X, dm[i, ], F)
      dev <- 1 + apply(dev^2, 1, sum) * nk/(N*(nk+1))
      dist[, i] <- prior[i] * (nk/(nk+1))^(p/2) * dev^(-(N - ng + 1)/2)
    }
  }
  cl <- apply(dist, 1, which.is.max)
  levels(cl) <- object$lev
  class(cl) <- "factor"
  posterior <- dist / drop(dist %*% rep(1, ng))
  dimnames(posterior) <- list(dimnames(x)[[1]], object$lev)
  list(class = cl, posterior = posterior, x = x[, 1:dimen, drop=F])
}

print.lda <- function(x, ...)
{
  if(!is.null(cl <- x$call)) {
    names(cl)[2] <- ""
    cat("Call:\n")
    dput(cl)
  }
  cat("\nPrior probabilities of groups:\n")
  print(x$prior, ...)
  cat("\nGroup means:\n")
  print(x$means, ...)
  cat("\nCoefficients of linear discriminants:\n")
  print(x$scaling, ...)
  svd <- x$svd
  names(svd) <- dimnames(x$scaling)[[2]]
  if(length(svd) > 1) {
    cat("\nProportion of trace:\n")
    print(round(svd^2/sum(svd^2), 4),  ...)
  }
  invisible(x)
}

plot.lda <- function(x, panel=panel.lda, ..., cex=0.7,
                     dimen, abbrev=F, xlab="LD1", ylab="LD2")
{
  panel.lda <- function(x, y, ...) {
    text(x, y, as.character(g.lda), cex=tcex)
  }
  if(!is.null(Terms <- x$terms)) { #
    # formula fit
    data <- model.frame.lm(x)
    X <- model.matrix(delete.response(Terms), data)
    g <- model.extract(data, "response")
    xint <- match("(Intercept)", dimnames(X)[[2]], nomatch=0)
```



```
    if(xint > 0) X <- X[, -xint, drop=F]
  } else { #
    # matrix or data-frame fit
    xname <- eval(substitute(x$call$x))
    gname <- eval(substitute(x$call[[3]]))
    if(!is.null(sub <- x$call$subset)) {
      X <- eval(parse(text=paste(deparse(xname),"[", deparse(sub),",]")),
               sys.parent())
      g <- eval(parse(text=paste(deparse(gname),"[", deparse(sub),"]")),
               sys.parent())
    } else {
      X <- eval(parse(text=deparse(xname)), sys.parent())
      g <- eval(parse(text=deparse(gname)), sys.parent())
    }
    if(!is.null(nas <- x$call$na.action)) {
      df <- structure(list(g = g, X = X), class = "data.frame")
      df <- do.call(nas, list(df))
      g <- df$g
      X <- df$X
    }
  }
  if(abbrev) levels(g) <- abbreviate(levels(g), abbrev)
  assign("g.lda", g, frame=1)
  assign("tcex", cex, frame=1)
  means <- apply(x$means, 2, mean)
  X <- scale(X, means, F) %*% x$scaling
  if(!missing(dimen) && dimen < ncol(X)) X <- X[, 1:dimen, drop=F]
  if(ncol(X) > 2) {
    pairs(X, panel=panel, ...)
  } else if(ncol(X) == 2)  {
    eqscplot(X[, 1:2], xlab=xlab, ylab=ylab, type="n", ...)
    panel(X[, 1], X[, 2], ...)
  } else ldahist(X[,1], g, xlab=xlab, ...)
  invisible(NULL)
}
```

lda for S-PLUS 5.0

```
# file MASS/lda.q
# copyright (C) 1994-8 W. N. Venables and B. D. Ripley
#
setClass("lda", representation(prior = "named",
                               counts = "named",
                               means = "matrix",
                               scaling = "matrix",
                               lev = "character",
                               svd = "numeric",
                               N = "integer",
                               call = "call")
         )

lda = function(x, y, ...)
{
  stop("lda not implemented for class ", class(x))
}

lda.formula = function(x, y, ...)
{
  data = as.data.frame(y)
  m = match.call(expand.dots = F)
  m$... = NULL
  names(m)[2:3] = c("formula", "data")
  if(hasArg(subset)) m$subset = subset
  if(hasArg(na.action)) m$na.action = na.action
  m[[1]] = as.name("model.frame")
  m = eval(m, sys.parent())
  Terms = attr(m, "terms")
  y = model.extract(m, "response")
  x = model.matrix(Terms, m)
  xint = match("(Intercept)", dimnames(x)[[2]], nomatch=0)
  if(xint > 0) x = x[, -xint, drop=F]
  res = lda1(x = x, grouping = y, ...)
  res@call = match.call()
  res
}


lda.data.frame = function(x, y, ...)
{
  x = as.matrix(x)
  callGeneric()
}

lda.Matrix = function(x, y, ...)
{
  x = data.matrix(x)
```



```
  callGeneric()
}

lda.matrix = function(x, y, ...)
{
  x = as(x, "matrix")
  dots = list(...)
  if(hasArg(subset)) {
    subset = dots$subset
    x = x[subset, , drop = F]
    y = y[subset]
  }
  if(hasArg(na.action)) {
    na.action = dots$na.action
    dfr = na.action(data.frame(g = y, x = x))
    y = dfr$g
    x = dfr$x
  }
  res = lda1(x=x, grouping=y, ...)
  res@call = match.call()
  res
}

setMethod("lda", "formula", lda.formula)
setMethod("lda", "matrix", lda.matrix)
setMethod("lda", "Matrix", lda.Matrix)
setMethod("lda", "data.frame", lda.data.frame)


lda1 = function(x, grouping, prior = NULL, tol = 1.0e-4,
                method = c("moment", "mle", "mve", "t"), CV=F,
                nu = 5, ...)
{
  which.is.max = function(x) {
    d = (1:length(x))[x == max(x)]
    if(length(d) > 1) d = sample(d, 1)
    d
  }

  if(is.null(dim(x))) stop("x is not a matrix")
  n = nrow(x)
  p = ncol(x)

    ....

  if(is.null(dimnames(x)))
    dimnames(scaling) = list(NULL, paste("LD", 1:rank, sep = ""))
  else {
    dimnames(scaling) = list(dimnames(x)[[2]], paste("LD", 1:rank, sep = ""))
    dimnames(group.means)[[2]] = dimnames(x)[[2]]
```

```
  }
  res = new("lda")
  res@prior = prior
  res@counts = counts
  res@means = group.means
  res@scaling = scaling
  res@lev = lev
  res@svd = X.s$d[1:rank]
  res@N = n
  res
}


predict.lda = function(object, newdata, prior = object@prior, dimen,
                        method = c("plug-in", "predictive", "debiased"), ...)
{
  if(!inherits(object, "lda")) stop("object not of class lda")
  which.is.max = function(x) {
    d = (1:length(x))[x == max(x)]
    if(length(d) > 1) d = sample(d, 1)
    d
  }
  if(is.form(form <- object@call[[2]])) { #
    # formula fit
    data = model.frame(object)
    form = replace.(as.formula(form), data)
    if(missing(newdata)) newdata = data
    else newdata = model.frame(form, newdata, na.action=function(x) x)
    x = model.matrix(delete.response(terms(form)), newdata)
    xint = match("(Intercept)", dimnames(x)[[2]], nomatch=0)
    if(xint > 0) x = x[, -xint, drop=F]
  } else { #
    # matrix or data-frame fit
    if(missing(newdata)) {
      xname = eval(substitute(object@call$x))
      if(!is.null(sub <- object@call$subset))
        newdata = eval(parse(text=paste(deparse(xname),"[",
                                deparse(sub),",]")), sys.parent())
      else newdata = eval(parse(text=deparse(xname)), sys.parent())
      if(!is.null(nas <- object@call$na.action))
        newdata = do.call(nas, list(newdata))
    }
    if(is.null(dim(newdata)))
      dim(newdata) = c(1, length(newdata))  # a row vector
    x = as.matrix(newdata)             # to cope with dataframes
  }

    ....  replaace $ by @ several times

  cl = apply(dist, 1, which.is.max)
```



```
  levels(cl) = object@lev
  oldClass(cl) = "factor"
  posterior = dist / drop(dist %*% rep(1, ng))
  dimnames(posterior) = list(dimnames(x)[[1]], object@lev)
  list(class = cl, posterior = posterior, x = x[, 1:dimen, drop=F])
}

show.lda = function(object) print.lda(object)

print.lda = function(x, ...)
{
  if(!is.null(cl <- x@call)) {
    names(cl)[2] = ""
    names(cl)[3] = ""
    cat("Call:\n")
    dput(cl)
  }
  cat("\nPrior probabilities of groups:\n")
  print(x@prior, ...)
  cat("\nGroup means:\n")
  print(x@means, ...)
  cat("\nCoefficients of linear discriminants:\n")
  print(x@scaling)
  svd = x@svd
  names(svd) = dimnames(x@scaling)[[2]]
  if(length(svd) > 1) {
    cat("\nProportion of trace:\n")
    print(round(svd^2/sum(svd^2), 4))
  }
  invisible(x)
}

plot.lda = function(x, panel=panel.lda, ..., cex=0.7,
                    dimen, abbrev=F, xlab="LD1", ylab="LD2")
{
   ....  same changes as to predict.lda
}


setMethod("show", "lda", show.lda)

update.lda = function(object, formula, ..., evaluate = T, class)
{
  newcall = object@call
  tempcall = match.call(expand.dots = F)$...
  if(!missing(formula))
    newcall[[2]] = as.vector(update.formula(object, formula, evaluate = T))
  else {
    nc = names(sys.call())
    if(length(nc) && any(pmatch(nc, "formula", 0))) newcall$formula = NULL
```

```
  }
  if(length(tempcall) > 1) {
    def = getFunction(newcall[[1]])
    def$formula = NULL
    TT = match.call(def, tempcall)
    if((ndots = length(TT)) < length(tempcall)) {
      nt = pmatch(names(tempcall), names(def)[ - length(def)])
      # kill some args
      nt = names(def)[nt]
      nT = names(TT)
      for(i in nt[is.na(match(nt, nT))]) newcall[[i]] = NULL
    }
    if(ndots > 1) {
      ndots = names(TT)[-1]
      newcall[ndots] = TT[ndots]
    }
  }
  if(evaluate) eval(newcall, sys.parent())
  else newcall
}

formula.lda = function(object) object@call[[2]]

model.frame.lda =
function(formula, data = NULL, na.action = NULL, ...)
{
  oc = formula@call
  oc[[1]] = as.name("model.frame")
  names(oc)[2:3] = c("formula", "data")
  oc$prior = oc$tol = oc$method = oc$CV = oc$nu = NULL
  oc[[1]] = as.name("model.frame")
  if(length(data)) {
    oc$data = substitute(data)
    eval(oc, sys.parent())
  }
  else eval(oc, list())
}

is.form = function(x) is.call(x) && (x[[1]] == "~")
```