Package 'hddplot'

June 15, 2018

Type Package

Title Use Known Groups in High-Dimensional Data to Derive Scores for Plots

Version 0.59

Date 2018-06-15

Author John Maindonald

Maintainer John Maindonald <jhmaindonald@gmail.com>

VignetteBuilder knitr

Description Cross-validated linear discriminant calculations determine the optimum number of features. Test and training scores from successive cross-validation steps determine, via a principal components calculation, a low-dimensional global space onto which test scores are projected, in order to plot them. Further functions are included that are intended for didactic use. The package implements, and extends, methods described in J.H. Maindonald and C.J. Burden (2005) <https://journal.austms.org.au/V46/CTAC2004/Main/home.html>.

LazyLoad true

LazyData true

Depends R (>= 3.0.0)

Imports MASS, multtest

Suggests knitr

ZipData yes

License GPL (>= 2)

URL http://maths-people.anu.edu.au/~johnm/

Repository CRAN

NeedsCompilation no

Date/Publication 2018-06-15 19:55:43 UTC

R topics documented:

`hddplot-package` *Use Known Groups in High-Dimensional Data to Derive Scores for Plots*

Description

Cross-validated linear discriminant calculations determine the optimum number of features. Test and training scores from successive cross-validation steps determine, via a principal components calculation, a low-dimensional global space onto which test scores are projected, in order to plot them. Further functions are included that are intended for didactic use. The package implements, and extends, methods described in J.H. Maindonald and C.J. Burden (2005) <https://journal.austms.org.au/V46/CTAC2004/M

Details

The DESCRIPTION file:

Package:	hddplot
Type:	Package
Title:	Use Known Groups in High-Dimensional Data to Derive Scores for Plots
Version:	0.59
Date:	2018-06-15
Author:	John Maindonald
Maintainer:	John Maindonald <jhmaindonald@gmail.com>
VignetteBuilder:	knitr
Description:	Cross-validated linear discriminant calculations determine the optimum number of features. Test and traini
LazyLoad:	true
LazyData:	true
Depends:	R (>= 3.0.0)

Imports:	MASS, multtest
Suggests:	knitr
ZipData:	yes
License:	GPL (>=2)
URL:	http://maths-people.anu.edu.au/~johnm/
Repository:	CRAN

Index of help topics:

Further information is available in the following vignettes:

Cross-validated linear discriminant calculations determine the optimum number of features. Test and training scores from successive cross-validation steps determine, via a principal components calculation, a low-dimensional global space onto which test scores are projected, in order to plot them. Further functions are included for didactic purposes.

Package:	hddplot
Type:	Package
Version:	1.0
Date:	2006-01-09
License:	GPL Version 2 or later.

The most important functions are

`cvdisc`: Determine variation in cross-validated accuracy with number of features

`cvscores`: For a specific choice of number of features, determine scores that can be used for plotting

Note also `scoreplot` (plot scores), `qqthin` (qqplots, designed to avoid generating large files when there are many points), and functions that are intended to illustrate issues that arise in the plotting of expression array and other high-dimensional data

Author(s)

John Maindonald

Maintainer: John Maindonald <jhmaindonald@gmail.com>

References

J. H. Maindonald, C. J. Burden, 2005. Selection bias in plots of microarray or other data that have been sampled from a high-dimensional space. In R. May and A.J. Roberts, eds., *Proceedings of 12th Computational Techniques and Applications Conference CTAC-2004*, volume 46, pp. C59–C74.

http://journal.austms.org.au/V46/CTAC2004/Main/home.html [March 15, 2005].

See Also

cvscores, scoreplot

Examples

```
## Use first 500 rows (expression values) of Golub, for demonstration.
data(Golub)
data(golubInfo)
attach(golubInfo)
miniG.BM <- Golub[1:500, BM.PB=="BM"]  # 1st 500 rows only
cancer.BM <- cancer[BM.PB=="BM"]
miniG.cv <- cvdisc(miniG.BM, cl=cancer.BM, nfeatures=1:10,
                   nfold=c(10,4))
miniG.scores <- cvscores(cvlist=miniG.cv, nfeatures=4, cl.other=NULL)
subsetB <- (cancer=="allB") & (tissue.mf %in% c("BM:f","BM:m","PB:m"))
tissue.mfB <- tissue.mf[subsetB, drop=TRUE]
```

```
scoreplot(scorelist=miniG.scores, cl.circle=tissue.mfB,
       circle=tissue.mfB%in%c("BM:f","BM:m"),
       params=list(circle=list(col=c("cyan","gray"))),
       prefix="BM samples -")
detach(golubInfo)
## Not run: demo(biasedPlots)
## Not run: demo(CVscoreplot)
```

`accTrainTest` *Two subsets of data each take in turn the role of test set*

Description

A division of data is specified, for use of linear discriminant analysis, into a training and test set. Feature selection and model fitting is formed, first with I/II as training/test, then with II/I as training/test

Usage

```
accTrainTest(x = matrix(rnorm(1000), ncol=20), cl = factor(rep(1:3,c(7,9,4))),
 traintest = divideUp(cl, nset=2), nfeatures = NULL, print.acc = FALSE,
 print.progress=TRUE)
```

Arguments

`x`	Matrix; rows are features, and columns are observations ('samples')
`cl`	Factor that classifies columns into groups that will classify the data for purposes of discriminant calculations
`traintest`	Values that specify a division of observations into two groups. In the first pass (fold), one to be training and the other test, with the roles then reversed in a second pass or fold.
`nfeatures`	integer: numbers of features for which calculations are required
`print.acc`	logical: should accuracies be printed?
`print.progress`	logical: should progress by feature number be printed?

Value

`sub1.2`	row numbers of features, by order of values of the group separation measure, for the first subset (I) of the `x`
`acc1.2`	accuracies, with I as training set and II as test
`sub2.1`	row numbers of features, by order of values of the group separation measure, for the second subset (II) of the `x`
`acc2.1`	accuracies, with II as training set and I as test

Author(s)

John Maindonald

Examples

```
mat <- matrix(rnorm(1000), ncol=20)
cl <- factor(rep(1:3, c(7,9,4)))
gp.id <- divideUp(cl, nset=2)
accTrainTest(x=mat, cl=cl, traintest=gp.id,
             nfeatures=1:16, print.acc=TRUE, print.progress=TRUE)

## The function is currently defined as
function(x=matrix(rnorm(1000), ncol=20), cl = factor(rep(1:3, c(7,9,4))),
         traintest=divideUp(cl, nset=2), nfeatures=NULL, print.acc=FALSE){
    traintest <- factor(traintest)
    train <- traintest==levels(traintest)[1]
    testset <- traintest==levels(traintest)[2]
    cl1 <- cl[train]
    cl2 <- cl[testset]
    ng1 <- length(cl1)
    ng2 <- length(cl2)
    maxg <- max(c(ng1-length(unique(cl1))-2,
                  ng2-length(unique(cl2))-2))
    if(is.null(nfeatures)){
      max.features <- maxg
      nfeatures <- 1:max.features
    } else
    {
      if(max(nfeatures)>maxg)nfeatures <- nfeatures[nfeatures<=maxg]
      max.features <- max(nfeatures)
    }
    ord1 <- orderFeatures(x, cl, subset=train)[1:max.features]
    ord2 <- orderFeatures(x, cl, subset=testset)[1:max.features]
    ord <- unique(c(ord1, ord2))
    sub1 <- match(ord1, ord)
    sub2 <- match(ord2, ord)
    df1 <- data.frame(t(x[ord, train]))
    df2 <- data.frame(t(x[ord, testset]))
    acc1 <- acc2 <- numeric(max(nfeatures))
    for(i in nfeatures){
      if(print.progress)cat(paste(i, ":", sep=""))
      df1.lda <- lda(df1[, sub1[1:i], drop=FALSE], cl1)
      hat2 <- predict(df1.lda, newdata=df2[, sub1[1:i], drop=FALSE])$class
      tab <- table(hat2, cl2)
      acc1[i] <- sum(tab[row(tab)==col(tab)])/sum(tab)
      df2.lda <- lda(df2[, sub2[1:i], drop=FALSE], cl2)
      hat1 <- predict(df2.lda, newdata=df1[, sub2[1:i], drop=FALSE])$class
      tab <- table(hat1, cl1)
      acc2[i] <- sum(tab[row(tab)==col(tab)])/sum(tab)
    }
    cat("\n")
    if(print.acc){
```

```
  print(round(acc1,2))
  print(round(acc2,2))
}
maxacc1 <- max(acc1)
maxacc2 <- max(acc2)
sub1 <- match(maxacc1, acc1)
sub2 <- match(maxacc2, acc2)
nextacc1 <- max(acc1[acc1<1])
nextacc2 <- max(acc1[acc1<2])
lower1 <- maxacc1-sqrt(nextacc1*(1-nextacc1)/ng1)
lower2 <- maxacc2-sqrt(nextacc2*(1-nextacc2)/ng2)
lsub1 <- min((1:ng1)[acc1>lower1])
lsub2 <- min((1:ng2)[acc2>lower2])
lower <- c("Best accuracy, less 1SD  ",
           paste(paste(round(c(lower1, lower2),2), c(lsub1, lsub2),
                       sep=" ("), " features)   ", sep=""))
best <- c("Best accuracy",
          paste(paste(round(c(maxacc1, maxacc2),2), c(sub1, sub2),
                      sep=" ("), " features)", sep=""))
acc.df <- cbind(lower, best)
dimnames(acc.df) <- list(c("Training/test split",
                           "I (training) / II (test)    ",
                           "II (training) / I (test)    "),c("",""))
print(acc.df, quote=FALSE)
invisible(list(sub1.2=ord1, acc1.2=acc1, sub2.1=ord2, acc2.1=acc2))
}
```

---

aovFbyrow *calculate aov F-statistic for each row of a matrix*

Description

Returns on aov F-statistic for each row of x

Usage

```
aovFbyrow(x=matrix(rnorm(1000), ncol=20), cl = factor(rep(1:3, c(7,9,4))))
```

Arguments

x	features by observations matrix
cl	factor that classifies the values in each row

Details

This uses the functions qr() and qr.qty() for the main part of the calculation, for handling the calculations efficently

Value

one F-statistic for each row of x

Author(s)

John Maindonald

See Also

See also orderFeatures

Examples

```
mat <- matrix(rnorm(1000), ncol=20)
cl <- factor(rep(1:3, c(7,9,4)))
Fstats <- aovFbyrow(x = mat, cl = cl)

## The function is currently defined as
aovFbyrow <-
function(x=matrix(rnorm(1000), ncol=20),
         cl=factor(rep(1:3, c(7,9,4)))){
    y <- t(x)
    qr.obj <- qr(model.matrix(~cl))
    qty.obj <- qr.qty(qr.obj,y)
    tab <- table(factor(cl))
    dfb <- length(tab)-1
    dfw <- sum(tab)-dfb-1
    ms.between <- apply(qty.obj[2:(dfb+1), , drop=FALSE]^2, 2, sum)/dfb
    ms.within <- apply(qty.obj[-(1:(dfb+1)), , drop=FALSE]^2, 2, sum)/dfw
    Fstat <- ms.between/ms.within
  }
```

---

cvdisc *Cross-validated accuracy, in linear discriminant calculations*

Description

Determine cross-validated accuracy, for each of a number of features in a specified range, with feature selection repeated at each step of the cross-validation.

Usage

```
cvdisc(x, cl, nfold = c(10,1), test = "f", nfeatures = 2, seed = 31,
       funda = lda, print.progress = TRUE, subset = NULL)
```

Arguments

`x`	Matrix; rows are features, and columns are observations ('samples')
`cl`	Factor that classifies columns into groups
`nfold`	Number of folds for the cross-validation. Optionally, a second number species the number of repeats of the cross-validation.
`test`	What statistic will be used to measure separation between groups? Currently `"f"` is the only possibility.
`nfeatures`	Specifies the different numbers of features (e.g., 1:10) that will be tried, to determine cross-validation accuracy in each instance
`seed`	This can be used to specify a starting value for the random number generator, in order to make calculations repeatable
`funda`	Function that will be used for discrimination. Currently `lda` is the only option
`print.progress`	Set to `TRUE` (default) for printing out, as calculations proceed, the number of the current fold
`subset`	Allows the use of a subset of the samples (observations)

Value

`folds`	Each column gives, for one run of the cross-validation, numbers that identify the `nfold` distinct folds of the cross-validation
`xUsed`	returns the rows of `x` that were used, in at least one fold
`cl`	Factor that classifies columns into groups
`acc.cv`	Cross-validated accuracy
`genelist`	Array: `max(nfeatures)` by number of folds by number of repeats, identifying the features chosen at each repeat of each fold. (for `k < max(nfeatures)` features, take the initial `k` rows
`Fmatrix`	Array, with the same dimensions as `genelist`, that gives the anova F-statistic when that feature is used on its own to separate groups
`nfeatures`	Specifies the different numbers of features that were tried, to determine cross-validation accuracy in each instance

Author(s)

John Maindonald

See Also

See also `cvscores`, `scoreplot`

Examples

```
## Use first 500 rows (expression values) of Golub, for demonstration.
data(Golub)
data(golubInfo)
attach(golubInfo)
miniG.BM <- Golub[1:500, BM.PB=="BM"]  # 1st 500 rows only
cancer.BM <- cancer[BM.PB=="BM"]
miniG.cv <- cvdisc(miniG.BM, cl=cancer.BM, nfeatures=1:10,
                   nfold=c(3,1))
## Plot cross-validated accuracy, as a function of number of features
plot(miniG.cv$acc.cv, type="l")


## The function is currently defined as
function(x, cl, nfold=NULL, test="f",
           nfeatures=2, seed=31, funda=lda, print.progress=TRUE,
           subset=NULL){
    ## If nfold is not specified, use leave-one-out CV
    if(is.null(nfold))nfold <- sum(!is.na(cl))
    ## Option to omit one or more points
    if(!is.null(subset)){cl[!is.na(cl)][!subset] <- NA
                         nfold[1] <- min(nfold[1], sum(!is.na(cl)))
                        }
    if(any(is.na(cl))){x <- x[,!is.na(cl)]
                       cl <- cl[!is.na(cl)]
                      }
    if(length(nfold)==1)nfold <- c(nfold,1)
    cl <- factor(cl)
    ngp <- length(levels(cl))
    genes <- rownames(x)
    nobs <- dim(x)[2]
    if(is.null(genes)){
      genes <- paste(1:dim(x)[1])
      print("Input rows (features) are not named. Names")
      print(paste(1,":", dim(x)[1], " will be assigned.", sep=""))
      rownames(x) <- genes
    }
    require(MASS)
    if(!is.null(seed))set.seed(seed)
    Fcut <- NULL
    maxgenes <- max(nfeatures)
    ## Cross-validation calculations
    if(nfold[1]==nobs)foldids <- matrix(sample(1:nfold[1]),ncol=1) else
    foldids <- sapply(1:nfold[2], function(x)
                    divideUp(cl, nset=nfold[1]))
    genelist <- array("", dim=c(nrow=maxgenes, ncol=nfold[1], nleaf=nfold[2]))
    Fmatrix <- array(0, dim=c(nrow=maxgenes, ncol=nfold[1], nleaf=nfold[2]))
    testscores <- NULL
    acc.cv <- numeric(maxgenes)
    if(print.progress)
      cat("\n", "Preliminary per fold calculations","\n")
    for(k in 1:nfold[2])
```

```
    {
      foldk <- foldids[,k]
  ufold <- sort(unique(foldk))
  for(i in ufold){
    if(print.progress) cat(paste(i,":",sep=""))
    trainset <- (1:nobs)[foldk!=i]
    cli <- factor(cl[trainset])

    stat <- aovFbyrow(x=x[, trainset], cl=cli)
    ordi <- order(-abs(stat))[1:maxgenes]
    genelist[,i, k] <- genes[ordi]
    Fmatrix[, i, k] <- stat[ordi]
  }
 }
  ulist <- unique(as.vector(genelist))
  df <- data.frame(t(x[ulist, , drop=FALSE]))
  names(df) <- ulist
#########################################################################
  if(print.progress)cat("\n", "Show each choice of number of features:","\n")
  for(ng in nfeatures){
    hat <- cl
    if(print.progress)cat(paste(ng,":",sep=""))
  for(k in 1:nfold[2])
    {
      foldk <- foldids[,k]
      ufold <- sort(unique(foldk))
    for(i in ufold){
      testset <- (1:nobs)[foldk==i]
      trainset <- (1:nobs)[foldk!=i]
      ntest <- length(testset)
      ntrain <- nobs-ntest
      genes.i <- genelist[1:ng, i, k]
      dfi <- df[-testset, genes.i, drop=FALSE]
      newdfi <- df[testset, genes.i, drop=FALSE]
      cli <- cl[-testset]
      xy.xda <- funda(cli~., data=dfi)
      subs <- match(colnames(dfi), rownames(df))
      newpred.xda <- predict(xy.xda, newdata=newdfi, method="debiased")
      hat[testset] <- newpred.xda$class
    }
      tabk <- table(hat,cl)
      if(k==1)tab <- tabk else tab <- tab+tabk
    }
    acc.cv[ng] <- sum(tab[row(tab)==col(tab)])/sum(tab)
  }
  cat("\n")
  if(length(nfeatures)>1&all(diff(nfeatures)==1)){
    nobs <- length(cl)
    ng1 <- length(acc.cv)
    maxacc1 <- max(acc.cv)
    sub1 <- match(maxacc1, acc.cv)
    nextacc1 <- max(acc.cv[acc.cv<1])
    lower1 <- maxacc1-sqrt(nextacc1*(1-nextacc1)/nobs)
```

```
    lsub1 <- min((1:ng1)[acc.cv>lower1])
    lower <- c("Best accuracy, less 1SD  ",
              paste(paste(round(c(lower1),2), c(lsub1),
                          sep=" ("), " features)   ", sep=""))
    best <- c("Best accuracy",
              paste(paste(round(c(maxacc1),2), c(sub1),
                          sep=" ("), " features)", sep=""))
    acc.df <- cbind(lower, best)
    dimnames(acc.df) <- list(c("Accuracy",
                               "(Cross-validation)"),c("",""))
    print(acc.df, quote=FALSE)
  }
  invisible(list(foldids=foldids, xUsed=df, cl=cl, acc.cv=acc.cv,
                 genelist=genelist, Fmatrix=Fmatrix, nfeatures=nfeatures))
}
```

`cvscores` *For high-dimensional data with known groups, derive scores for plotting*

Description

This is designed to used with the output from `cvdisc`. Test and training scores from successive cross-validation steps determine, via a principal components calculation, a low-dimensional global space onto which test scores are projected, in order to plot them.

Usage

```
cvscores(cvlist, nfeatures, ndisc = NULL, cl.other,
         x.other, keepcols = NULL, print.progress = TRUE)
```

Arguments

`cvlist`	Output object from `cvdisc`
`nfeatures`	Number of features to use
`ndisc`	Dimension of space in which scores will be formed, at most one less than the number of groups
`cl.other`	Classifies additional observations that are to be projected onto the same low-dimensional space
`x.other`	Matrix from which additional observations will be taken
`keepcols`	Number of sets of principal component scores to use in discriminant calculations and consequent evaluation of scores that will determine the low-dimensional global space
`print.progress`	Set to `TRUE` (default) for printing out, as calculations proceed, the number of the current fold

Value

scores	Scores that can be plotted
cl	Factor that was used to classify observations into groups
other.scores	Other scores, if any, for plotting
cl.other	Factor that was used to classify the 'other' data into groups
nfeatures	Number of features used

Note

The methodology used here has developed beyond that described in Maindonald and Burden (2005)

Author(s)

John Maindonald

References

J. H. Maindonald and C. J. Burden, 2005. Selection bias in plots of microarray or other data that have been sampled from a high-dimensional space. In R. May and A.J. Roberts, eds., *Proceedings of 12th Computational Techniques and Applications Conference CTAC-2004*, volume 46, pp. C59–C74.

http://journal.austms.org.au/V46/CTAC2004/Main/home.html [March 15, 2005]

See Also

See also cvdisc, scoreplot

Examples

```
## Use first 500 rows (expression values) of Golub, for demonstration.
data(Golub)
data(golubInfo)
attach(golubInfo)
miniG.BM <- Golub[1:500, BM.PB=="BM"]  # 1st 500 rows only
cancer.BM <- cancer[BM.PB=="BM"]
miniG.cv <- cvdisc(miniG.BM, cl=cancer.BM, nfeatures=1:10,
                   nfold=c(3,1))
miniG.scores <- cvscores(cvlist=miniG.cv, nfeatures=4,
                         cl.other=NULL)
detach(golubInfo)

## The function is currently defined as
function(cvlist, nfeatures, ndisc=NULL, cl.other, x.other,
          keepcols=NULL, print.progress=TRUE
          ){
    library(MASS)
    foldids <- cvlist$foldids
    nfold <- c(length(unique(foldids)), dim(foldids)[2])
```

```
    ugenes <- unique(as.vector(cvlist$genelist[1:nfeatures, ,]))
    df <- cvlist$xUsed[, ugenes]
    cl <- cvlist$cl
    if(!length(cl)==dim(df)[1])
      stop(paste("length(cl) =", length(cl),"does not equal",
                 "dim(cvlist$df)[1] =", dim(df)[1]))
    levnames <- levels(cl)
    if(is.null(ndisc))ndisc <- length(levnames)-1
    ngp <- length(levnames)
    nobs <- dim(df)[1]
    allscores <- array(0, dim=c(nrow=nobs, ncol=ndisc*nfold[1], nleaf=nfold[2]))
    if(!is.null(cl.other)){
      cl.other <- factor(cl.other)
      if(is.null(dim(x.other)))stop("x.other must have dimension 2")
      if(!length(cl.other)==dim(x.other)[2])
        stop(paste("length(cl.other) =", length(cl.other),"does not equal",
                   "dim(x.other)[2] =", dim(x.other)[2]))
      df.other <- data.frame(t(x.other[ugenes, ,drop=FALSE]))
      colnames(df.other) <- ugenes
    }
    else other.scores <- NULL
    for(k in 1:nfold[2]){
      foldk <- foldids[,k]
      ufold <- sort(unique(foldk))
      j <- 0
      for(i in ufold){
        j <- j+1
        if(print.progress)cat(paste(if(j>1) ":" else "", i,sep=""))
        testi <- (1:nobs)[foldk==i]
        traini <- (1:nobs)[foldk!=i]
        ntest <- length(testi)
        ntrain <- nobs-ntest
        genes.i <- cvlist$genelist[1:nfeatures, i, k]
        dfi <- as.data.frame(df[-testi, genes.i, drop=FALSE])
        newdfi <- as.data.frame(df[testi, genes.i, drop=FALSE])
        cli <- cl[-testi]
        xy.xda <- lda(cli~., data=dfi)
        allscores[, ((i-1)*ndisc)+(1:ndisc), k] <-
          predict(xy.xda, newdata=df, dimen=ndisc)$x
      }
    }
    cat("\n")
    dim(allscores) <- c(nobs, ndisc*prod(nfold))
    if(is.null(keepcols))keepcols <- min(nfeatures, dim(allscores)[2])
    allscores.pcp <- data.frame(pcp(allscores, varscores=FALSE)$g[, 1:keepcols])
    globals <- predict(lda(cl ~ ., data=allscores.pcp))$x[,1:ndisc]
    fitscores <- array(0, dim=c(nrow=nobs, ncol=ndisc, nleaf=nfold[2]))
    for(k in 1:nfold[2]){
      foldk <- foldids[,k]
      ufold <- sort(unique(foldk))
##      ntimes.genes <- table(cvlist$genelist[1:nfeatures,,k])
      av <- colMeans(df)
      j <- 0
```

```
    for(i in ufold){
      j <- j+1
      cat(paste(if (j>1) ":" else "", i,sep=""))
      testi <- (1:nobs)[foldk==i]
      traini <- (1:nobs)[foldk!=i]
      genes.i <- cvlist$genelist[1:nfeatures, i, k]
      dfi <- data.frame(df[-testi, genes.i, drop=FALSE])
      newdfi <- data.frame(df[testi, genes.i, drop=FALSE])
      cli <- cl[-testi]
      traini.xda <- lda(cli~., data=dfi)
      scorei <- predict(traini.xda)$x[,1:ndisc]
      newpred.xda <- predict(traini.xda, newdata=newdfi)
      scorei.out <- newpred.xda$x[, 1:ndisc, drop=FALSE]
      scorei.all <- globals[-testi, 1:ndisc]
      avcol <- colMeans(scorei.all)
      scorei.all <- sweep(scorei.all, 2, avcol,"-")
      avi <- colMeans(scorei)
      scorei <- sweep(scorei, 2, avi,"-")
      trans <- qr.solve(scorei, scorei.all)
      scorei.out <- sweep(scorei.out, 2, avi, "-")
      fitscores[testi, , k] <- sweep(scorei.out%*%trans, 2, avcol, "+")
    }
  }
  fitscores <- apply(fitscores, 1:2, mean)

  if(!is.null(cl.other)){
    Fmatrix <- cvlist$Fmatrix
    ord <- order(Fmatrix)[1:nfeatures]
    rowcol <- cbind(as.vector(row(Fmatrix))[ord],as.vector(col(Fmatrix))[ord])
    ugenes <- unique(as.vector(cvlist$genelist[rowcol]))
    df <- cvlist$xUsed[, ugenes]
    xy.xda <- lda(cl~., data=df)
    train.scores <- predict(xy.xda, dimen=ndisc)$x
    other.scores <- predict(xy.xda, newdata=df.other,
                            dimen=ndisc)$x
    avcol <- colMeans(globals)
    all.scores <- sweep(globals, 2, avcol,"-")
    av.train <- colMeans(train.scores)
    train.scores <- sweep(train.scores, 2, av.train, "-")
    trans <- qr.solve(train.scores, all.scores)
    other.scores <- sweep(other.scores%*%trans, 2, avcol, "+")
  }
  if(print.progress)cat("\n")
  invisible(list(scores=fitscores, cl=cl, other=other.scores,
                 cl.other=cl.other, nfeatures=nfeatures))
}
```

`defectiveCVdisc` *defective accuracy assessments from linear discriminant calculations*

Description

Determine cross-validated accuracy, for each of a number of features in a specified range, in each case with a set of features that have been selected using the total data. The "accuracy" assessment are provided only for comparative purposes

Usage

```
defectiveCVdisc(x, cl, nfold = NULL, FUN = aovFbyrow, nfeatures = 2, seed = 31,
         funda = lda, foldids = NULL, subset = NULL, print.progress = TRUE)
```

Arguments

x	Matrix; rows are features, and columns are observations ('samples')
cl	Factor that classifies columns into groups
nfold	Number of folds for the cross-validation. Optionally, a second number species the number of repeats of the cross-validation
FUN	function used to calculate a measure, for each row, of separation into groups
nfeatures	Specifies the different numbers of features (e.g., 1:10) that will be tried, to determine cross-validation accuracy in each instance
seed	This can be used to specify a starting value for the random number generator, in order to make calculations repeatable
funda	Function that will be used for discrimination. Currently lda is the only option
foldids	Fold information, as output from cvdisc()
subset	Allows the use of a subset of the samples (observations)
print.progress	Set to TRUE (default) for printing out, as calculations proceed, the number of the current fold

Value

acc.resub	resubstitution measure of 'accuracy'
acc.sel1	'accuracy' from cross-validation, with the initially selected features

Author(s)

John Maindonald

See Also

cvdisc

Examples

```
mat <- matrix(rnorm(1000), ncol=20)
cl <- factor(rep(1:3, c(7,9,4)))
badaccs <- defectiveCVdisc(mat, cl, nfold=c(3,1), nfeatures=1:5)
## Note the list elements acc.resub and acc.sel1


## The function is currently defined as
function(x, cl, nfold=NULL, FUN=aovFbyrow,
           nfeatures=2, seed=31, funda=lda, foldids=NULL,
           subset=NULL, print.progress=TRUE){
    ## Option to omit one or more points
    if(!is.null(subset)) cl[!is.na(cl)][!subset] <- NA
    if(any(is.na(cl))){x <- x[,!is.na(cl)]
                       cl <- cl[!is.na(cl)]
                      }
    nobs <- dim(x)[2]
    ## Get fold information from foldids, if specified,
    ## else if nfold is not specified, use leave-one-out CV
    if(!is.null(foldids))
      nfold <- c(length(unique(foldids)), dim(foldids)[2])
    if(is.null(nfold)&is.null(foldids))nfold <- sum(!is.na(cl))
    else if(nfold[1]==nobs)foldids <- sample(1:nfold[1])
    else foldids <- sapply(1:nfold[2], function(x)
                     divideUp(cl, nset=nfold[1]))
    if(length(nfold)==1)nfold <- c(nfold,1)
    cl <- factor(cl)
    ngp <- length(levels(cl))
    genes <- rownames(x)
     if(is.null(genes)){
      genes <- paste(1:dim(x)[1])
      print("Input rows (features) are not named. Names")
      print(paste(1,":", dim(x)[1], " will be assigned.", sep=""))
      rownames(x) <- genes
    }
    require(MASS)
    if(!is.null(seed))set.seed(seed)
    Fcut <- NULL
    maxgenes <- max(nfeatures)

    stat <- FUN(x=x, cl)
    Fcut <- list(F=sort(stat, decreasing=TRUE)[nfeatures],
                 df=c(ngp-1, nobs-ngp))
    ord <- order(-abs(stat))[1:maxgenes]
    genes.ord <- genes[ord]
    selectonce.df <- data.frame(t(x[ord, , drop=FALSE]))
    acc.resub <- acc.sel1 <- numeric(maxgenes)
    if(nfold[1]==0)acc.sel1 <- NULL

    for(ng in nfeatures){
      resub.xda <- funda(cl~., data=selectonce.df[,1:ng,drop=FALSE])
      hat.rsb <- predict(resub.xda)$class
```

```
    tab.rsb <- table(hat.rsb, cl)
    acc.resub[ng] <- sum(tab.rsb[row(tab.rsb)==col(tab.rsb)])/sum(tab.rsb)
    if(nfold[1]==0)next
    if(nfold[1]==nobs){
      hat.sel1 <- funda(cl~., data=selectonce.df[,1:ng,drop=FALSE],
                        CV=TRUE)$class
      tab.one <- table(hat.sel1, cl)
      acc.sel1[ng] <- sum(tab.one[row(tab.one)==col(tab.one)])/sum(tab.one)
    } else
    {
    hat <- cl
    if(print.progress)cat(paste(ng,":",sep=""))
    for(k in 1:nfold[2])
    {
      foldk <- foldids[,k]
      ufold <- sort(unique(foldk))
      for(i in ufold){
        testset <- (1:nobs)[foldk==i]
        trainset <- (1:nobs)[foldk!=i]
        dfi <- selectonce.df[-testset, 1:ng, drop=FALSE]
        newdfi <- selectonce.df[testset, 1:ng, drop=FALSE]
        cli <- cl[-testset]
        xy.xda <- funda(cli~., data=dfi)
        subs <- match(colnames(dfi), rownames(df))
        newpred.xda <- predict(xy.xda, newdata=newdfi, method="debiased")
        hat[testset] <- newpred.xda$class
      }
      tabk <- table(hat,cl)
      if(k==1)tab <- tabk else tab <- tab+tabk
    }
    acc.sel1[ng] <- sum(tab[row(tab)==col(tab)])/sum(tab)
    }
  }
  if(print.progress)cat("\n")
  invisible(list(acc.resub=acc.resub, acc.sel1=acc.sel1, genes=genes.ord))
}
```

`divideUp` *Partition data into mutiple nearly equal subsets*

Description

Randomly partition data into nearly equal subsets. If `balanced=TRUE` the requirement is imposed that the subsets should as far as possible be balanced with respect to a classifying factor. The multiple sets are suitable for use for determining the folds in a cross-validation.

Usage

```
divideUp(cl, nset = 2, seed = NULL, balanced=TRUE)
```

Arguments

cl	classifying factor
nset	number of subsets into which to partition data
seed	set the seed, if required, in order to obtain reproducible results
balanced	logical: should subsets be as far as possible balanced with respect to the classifying factor?

Value

a set of indices that identify the nset subsets

Author(s)

John Maindonald

Examples

```
foldid <- divideUp(cl=rep(1:3, c(17,14,8)), nset=10)
table(rep(1:3, c(17,14,8)), foldid)
foldid <- divideUp(cl=rep(1:3, c(17,14,8)), nset=10,
            balanced=FALSE)
table(rep(1:3, c(17,14,8)), foldid)


## The function is currently defined as
function(cl = rep(1:3, c(7, 4, 8)), nset=2, seed=NULL, balanced=TRUE){
    if(!is.null(seed))set.seed(seed)
    if(balanced){
      ord <- order(cl)
      ordcl <- cl[ord]
      gp0 <- rep(sample(1:nset), length.out=length(cl))
      gp <- unlist(split(gp0,ordcl), function(x)sample(x))
      gp[ord] <- gp
    } else
    gp <- sample(rep(1:nset, length.out=length(cl)))
    as.vector(gp)
  }
```

Golub *Golub data (7129 rows by 72 columns), after normalization*

Description

These are a normalized version of the Golub leukemia data from the golubEsets package, available from:

http://www.bioconductor.org/download/experiments/

Usage

```
data(Golub)
```

Format

Numeric matrix: 7129 rows by 72 columns.

Details

Data have been normalized and are supplied, here, as a matrix.

Source

See the help page for the dataset golubMerge, in the golubEsets package, for details of the source of the original data.

References

Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring, Science, 531-537, 1999, T. R. Golub and D. K. Slonim and P. Tamayo and C. Huard and M. Gaasenbeek and J. P. Mesirov and H. Coller and M.L. Loh and J. R. Downing and M. A. Caligiuri and C. D. Bloomfield and E. S. Lander

Examples

```
data(Golub)
## Select 20 rows from the data; show boxplots of variation across chips
boxplot(data.frame(t(Golub[sample(1:7129, 20), ])))
```

---

golubInfo *Classifying factors for the 72 columns of the Golub data set*

Description

Details are given of the classifying factors for the 72 columns of the Golub data set.

Usage

```
data(golubInfo)
```

Format

A data frame with 72 observations on the following 6 variables, that identifies the samples (observations) in the data set Golub

Samples a numeric vector: sample number

BM.PB a factor with levels BM (from bone marrow) PB (from peripheral blood)

Gender a factor with levels F M

`Source` a factor with levels `CALGB CCG DFCI St-Jude`. These are the hospitals from which the sample came

`tissue.mf` a factor with levels `BM:NA BM:f BM:m PB:NA PB:f PB:m`. This factor identifies the several combinations of `source` and `Gender`

`cancer` a factor with levels `allB allT aml` There are two types of Acute Lymphoblastic Leukemia (`allB` and `allT`), plus Acute Myoblastic Leukemia (`aml`)

Source

See the help page for the dataset `golubMerge`, in the `golubEsets` package, for details of the source of the original data.

References

Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring, Science, 531-537, 1999, T. R. Golub and D. K. Slonim and P. Tamayo and C. Huard and M. Gaasenbeek and J. P. Mesirov and H. Coller and M.L. Loh and J. R. Downing and M. A. Caligiuri and C. D. Bloomfield and E. S. Lander

Examples

```
data(golubInfo)
str(golubInfo)
```

---

orderFeatures *Order features, based on their ability to discriminate*

Description

For each row of `data`, an F or (potentially) other statistic is calculated, using the function `FUN`, that measures the extent to which this variable separates the data into groups. This statistic is then used to order the rows.

Usage

```
 orderFeatures(x, cl, subset = NULL, FUN = aovFbyrow, values =
FALSE)
```

Arguments

`x`	Matrix; rows are features, and columns are observations ('samples')
`cl`	Factor that classifies columns into groups
`subset`	allows specification of a subset of the columns of `data`
`FUN`	specifies the function used to measure separation between groups
`values`	if `TRUE`, F-values as well as the ordering are returned

Value

Either (`values=FALSE`) a vector that orders the rows, or (`values=TRUE`)

`ord`	a vector that orders the rows
`stat`	ordered values of the statistic

Author(s)

John Maindonald

Examples

```
mat <- matrix(rnorm(1000), ncol=20)
cl <- factor(rep(1:3, c(7,9,4)))
ord <- orderFeatures(mat, cl)

## The function is currently defined as
function(x, cl, subset=NULL, FUN=aovFbyrow, values=FALSE){
    if(dim(x)[2]!=length(cl))stop(paste("Dimension 2 of x is",
                  dim(x)[2], "differs from the length of cl (=",
                  length(cl)))
    ## Ensure that cl is a factor & has no redundant levels
    if(is.null(subset))
      cl <- factor(cl)
    else
      cl <- factor(cl[subset])
    if(is.null(subset))
      stat <- FUN(x, cl)
    else
      stat <- FUN(x[, subset], cl)
    ord <- order(-abs(stat))
    if(!values)ord else(list(ord=ord, stat=stat[ord]))
  }
```

---

pcp *convenience version of the singular value decomposition*

Description

Packages results from an SVD on what can be either a cases by variables (features) or variables by cases layout, for use in principal component and related calculations

Usage

```
pcp(x = datasets::USArrests, varscores = TRUE, cases = "rows", center = "vars",
    standardize = FALSE, scale.cases = 1, log = FALSE, sc = 1, reflect = c(1, 1))
```

Arguments

x	matrix on which SVD is to be performed
varscores	logical; should scores be returned?
cases	specify either "rows" or "columns"
center	logical: if set to "vars", then values of variables will be centered
standardize	logical: should values of variables be standardized to zero mean and unit deviance. Takes precedence over the setting of center
scale.cases	set to a value in [0,1]. scale.cases=0 gives a pure rotation of the variables. scale.cases=1 weights a/c the singular values
log	logical: should logarithms be taken, prior to the calculation?
sc	the variable scores are divided by $sqrtsc - 1$. By default, sc = number of cases
reflect	a vector of two elements, by default c(1,1). Use of -1 in one or both positions can be useful in reconciling results with output from other software

Value

g	case scores
h	variable scores
avv	variable means
sdev	singular values, divides by the square root of one less than the number of cases

Author(s)

John Maindonald

See Also

La.svd

Examples

```
USArrests.svd <- pcp(x = datasets::USArrests)

## The function is currently defined as
function(x=datasets::USArrests,
           varscores=TRUE,
           cases="rows",
           center="vars",
           standardize=FALSE,
           scale.cases=1,
           log=FALSE,
           sc=1,
           reflect=c(1,1))
{
  x <- as.matrix(x)
  avv <- 0
  sdv <- 1
```

```
casedim <- 2-as.logical(cases=="rows")
vardim <- 3-casedim
## casedim=1 if rows are cases; otherwise casedim=2
## scale.cases=0 gives a pure rotation of the variables
## scale.cases=1 weights a/c the singular values
ncases <- dim(x)[casedim]
nvar <- dim(x)[vardim]
if(is.null(sc))sc <- dim(x)[casedim]-1
if(log)x <- log(x, base=2)
if(standardize){
  avv <- apply(x, vardim, mean)
  sdv <- apply(x, vardim, sd)
  x <- sweep(x, vardim, avv,"-")
  x <- sweep(x, vardim, sdv,"/")
}
else if(as.logical(match("vars", center, nomatch=0))){
  avv <- apply(x,vardim, mean)
  x <- sweep(x, vardim, avv,"-")}

svdx <- La.svd(x, method = c("dgesdd"))
h <- NULL
if(cases=="rows"){
  g <- sweep(svdx$u, 2, svdx$d^scale.cases, "*")*sqrt(sc)
  if(varscores)
    h <- t((svdx$d^(1-scale.cases)* svdx$vt ))/sqrt(sc)
}
else if(cases=="columns"){
  g <- sweep(t(svdx$vt), 2, svdx$d^scale.cases, "*")*sqrt(sc)
  if(varscores)
    h <- sweep(svdx$u, 2, svdx$d^(1-scale.cases),"*")/sqrt(sc)
}
invisible(list(g=g, rotation=h, av=avv, sdev=svdx$d/sqrt(ncases-1)))
}
```

`plotTrainTest` *Plot predictions for both a I/II train/test split, and the reverse*

Description

A division of data is specified, for use of linear discriminant analysis, into a training and test set. Feature selection and model fitting is formed, first with I/II as training/test, then with II/I as training/test. Two graphs are plotted – for the I (training) /II (test) scores, and for the II/I scores.

Usage

```
plotTrainTest(x, nfeatures, cl, traintest,
              titles = c("A: I/II (train with I, scores are for II)",
                         "B: II/I (train with II, scores are for I)"))
```

Arguments

x	Matrix; rows are features, and columns are observations ('samples')
nfeatures	integer: numbers of features for which calculations are required
cl	Factor that classifies columns into groups that will classify the data for purposes of discriminant calculations
traintest	Values that specify a division of observations into two groups. In the first pass (fold), one to be training and the other test, with the roles then reversed in a second pass or fold.
titles	A character vector of length 2 giving titles for the two graphs

Value

Two graphs are plotted.

Author(s)

John Maindonald

Examples

```
mat <- matrix(rnorm(1000), ncol=20)
cl <- factor(rep(1:3, c(7,9,4)))
gp.id <- divideUp(cl, nset=2)
plotTrainTest(x=mat, cl=cl, traintest=gp.id, nfeatures=c(2,3))



## The function is currently defined as
function(x, nfeatures, cl, traintest,
           titles=c("A: I/II (train with I, scores are for II)",
             "B: II/I (train with II, scores are for I)")){
    oldpar <- par(mfrow=c(1,2), pty="s")
    on.exit(par(oldpar))
    if(length(nfeatures)==1)nfeatures <- rep(nfeatures,2)
    traintest <- factor(traintest)
    train <- traintest==levels(traintest)[1]
    testset <- traintest==levels(traintest)[2]
    cl1 <- cl[train]
    cl2 <- cl[testset]
    nf1 <- nfeatures[1]
    ord1 <- orderFeatures(x, cl, subset=train)
    df1 <- data.frame(t(x[ord1[1:nf1], train]))
    df2 <- data.frame(t(x[ord1[1:nf1], testset]))
    df1.lda <- lda(df1, cl1)
    scores <- predict(df1.lda, newdata=df2)$x
    scoreplot(scorelist=list(scores=scores, cl=cl2,
             nfeatures=nfeatures[1], other=NULL, cl.other=NULL),
           prefix.title="")
    mtext(side=3, line=2, titles[1], adj=0)
    nf2 <- nfeatures[2]
```

```
  ord2 <- orderFeatures(x, cl, subset=testset)
  df2 <- data.frame(t(x[ord2[1:nf2], testset]))
  df1 <- data.frame(t(x[ord2[1:nf2], train]))
  df2.lda <- lda(df2, cl2)
  scores <- predict(df2.lda, newdata=df1)$x
  scoreplot(scorelist=list(scores=scores, cl=cl1,
           nfeatures=nfeatures[2], other=NULL, cl.other=NULL),
         prefix.title="")
  mtext(side=3, line=2, titles[2], adj=0)
}
```

---

qqthin *a version of qqplot() that thins out points that overplot*

Description

QQ-plots with large numbers of points typically generate graphics files that are unhelpfully large. This function handles the problem by removing points that are, for all practical purposes, redundant

Usage

```
qqthin(x, y, ends = c(0.01, 0.99), eps = 0.001, xlab = deparse(substitute(x)),
       adj.xlab = NULL, ylab = deparse(substitute(y)), show.line = TRUE,
       print.thinning.details=TRUE, centerline = TRUE, ...)
```

Arguments

x	ordered values of x will be plotted on the x-axis
y	ordered values of y will be plotted on the y-axis
ends	outside these cumulative proportions of numbers of points, all points will be included in the graph
eps	controls the extent of overplotting
xlab	label for x-axis
adj.xlab	positioning of x-label
ylab	label for y-axis
show.line	logical; show the line y=x?
print.thinning.details	logical; print number of points after thinning?
centerline	logical; draw a line though the part of the graph where some points have been omitted?
...	additional graphics parameters

Value

Gives a qqplot. The number of points retained is returned invisibly.

Author(s)

John Maindonald

References

~put references to the literature/web site here ~

Examples

```
mat <- matrix(rnorm(1000), ncol=20)
cl <- factor(rep(1:3, c(7,9,4)))
Fstats <- aovFbyrow(x = mat, cl = cl)
qqthin(qf(ppoints(length(Fstats)), 2, 17), Fstats, eps=0.01)


## The function is currently defined as
function(x, y, ends=c(.01,.99), eps=0.001,
           xlab = deparse(substitute(x)), adj.xlab=NULL,
           ylab = deparse(substitute(y)), show.line=TRUE,
           print.thinning.details=TRUE,
           centerline=TRUE, ...){
    ## qqthin() is a substitute for qqplot(), that thins
    ## out plotted points from the region where they are
    ## dense.  Apart from the overlaid curve that shows
    ## the region where points have been thinned, it may
    ## be hard to distinguish the result of qqthin()
    ## from that of qqplot()
    xlab <- xlab
    ylab <- ylab
    x <- sort(x)
    y <- sort(y)
    dx<-diff(x)
    epsdist <- sqrt(diff(range(x))^2+diff(range(y))^2)*eps
    dx<-0.5*(c(dx[1],dx)+c(dx,dx[length(dx)]))
    dy<-diff(y)
    dy<-0.5*(c(dy[1],dy)+c(dy,dy[length(dy)]))
    dpoints <- epsdist/sqrt(dx^2+dy^2)
    ## dpoints is a local measure of the number of points
    ## per unit distance along the diagonal, with the unit
    ## set to approximately eps*(length of diagonal)
    dig<-floor(dpoints)+1
    ## dig is, roughly, the number of points per unit distance.
    ## We wish to retain one point per unit distance.  For this
    ## retain points where cdig rounds to an integer. For such
    ## points, cdig has increased by approx 1, relative to the
    ## previous point that is retained.
    cdig<-round(cumsum(1/dig))
    subs<-match(unique(cdig), cdig)
    if(is.null(adj.xlab))
    plot(x[subs], y[subs], xlab=xlab, ylab=ylab)
    else {
      plot(x[subs], y[subs], xlab="", ylab=ylab)
```

```
      mtext(side=1, xlab, adj=adj.xlab, line=par()$mgp[1])
    }
    if(any(diff(subs)>1)){
    n1 <- min(subs[c(diff(subs),0)>1])
    n2 <- max(subs[c(0,diff(subs))>1])
    ns1 <- match(n1, subs)
    ns2 <- match(n2, subs)
    if(print.thinning.details)
       print(paste("Graph retains", length(subs), "points."))
    if(centerline)
      lines(smooth.spline(x[subs[ns1:ns2]], y[subs[ns1:ns2]]),
            col="grey", lwd=2)
    }
    if(show.line)abline(0, 1, col="red")
invisible(length(subs))
  }
```

---

scoreplot *Plot discriminant function scores, with various identification*

Description

There is provision for the plottting of two sets of scores on the same graph, possibly with different classifying factors. The function is designed for use with output from `cvscores()` or from `simulateScores()`.

Usage

```
scoreplot(scorelist, plot.disc = 1:2, xlab = NULL, ylab = NULL, params = NULL,
          circle = NULL, cl.circle = NULL, circle.pos = c(1, 1), adj.circle = 1,
          adj.title = 0.5, join.legends = TRUE, prefix.title = "", cex.title = 1,
           ratio = 1, plot.folds = FALSE, ...)
```

Arguments

`scorelist`	list, with elements `scores` (a matrix of scores) `cl` (a classifying factor), `other` (optional, a further sets of scores), `cl.other` (a a classifying factor for `other`, optional) and `nfeatures` (optional, used to label the graph)
`plot.disc`	choice of columns of `scorelist` to plot
`xlab`	label for x-axis
`ylab`	label for y-axis
`params`	List, with optional elements (lists) `points`, `other`, `circle` and `legend`. Allowed list elements for `points` and `other` are `cex`, `lwd`, `pch` and `col`. For `circle` they are `cex`, `lwd` and `col`. For `legend`, they are `cex` and `cex.other`
`circle`	identifies points that are to be circled
`cl.circle`	different colors may be used for different points, according to levels of `cl.circle`

circle.pos	This is a vector of length 2, that specifies where to place the legend information for the circling of points. Possibilities are c(0,0) (left, below), c(1,1) (right, above), etc.
adj.circle	controls positioning of circle legend
adj.title	controls positioning of title
join.legends	logical; should legends for points and other be combined?
prefix.title	prefix, to place before title
cex.title	cex for title
ratio	y-scale to x-scale ratio for graph
plot.folds	Plot individual fold information, comparing projected training scores with their projections onto the global space. This is not at present implemented
...	Other parameters to be passed to eqscplot()

Value

A graph is plotted.

Author(s)

John Maindonald

See Also

See also cvdisc, cvscores

Examples

```
## Use first 500 rows (expression values) of Golub, for demonstration.
data(Golub)
data(golubInfo)
attach(golubInfo)
miniG.BM <- Golub[1:500, BM.PB=="BM"]  # 1st 500 rows only
cancer.BM <- cancer[BM.PB=="BM"]
miniG.cv <- cvdisc(miniG.BM, cl=cancer.BM, nfeatures=1:10,
                   nfold=c(3,1))
miniG.scores <- cvscores(cvlist=miniG.cv, nfeatures=4,
                         cl.other=NULL)
subsetB <- (cancer=="allB") & (tissue.mf %in% c("BM:f","BM:m","PB:m"))
tissue.mfB <- tissue.mf[subsetB, drop=TRUE]
scoreplot(scorelist=miniG.scores, cl.circle=tissue.mfB,
       circle=tissue.mfB%in%c("BM:f","BM:m"),
       params=list(circle=list(col=c("cyan","gray"))),
       prefix="BM samples -")
detach(golubInfo)

## The function is currently defined as
  function(scorelist, plot.disc=1:2,
           xlab=NULL, ylab=NULL, params=NULL,
           circle=NULL, cl.circle=NULL, circle.pos=c(1,1),
```

```
       adj.circle=1,
       adj.title=0.5, join.legends=T, prefix.title="Golub data - ",
       cex.title=1.0, ratio=1, plot.folds=FALSE, ...){
library(MASS)
combine.params <-
  function(params=list(circle=list(col=c("cyan","gray")))){
    default.params=list(points=list(cex=1, lwd=1.25, pch=1:8, col=1:8),
      other=list(cex=0.65, lwd=1.25, pch=13:9, col=c(6:8,5:1)),
      circle=list(cex=2, lwd=1, pch=1.75, col="gray40"),
      legend=list(cex=1, cex.other=1))
    nam <- names(params)
    if(!is.null(nam))
      for(a in nam){
        nam2 <- names(params[[a]])
        for(b in nam2)default.params[[a]][[b]] <- params[[a]][[b]]
      }
    default.params
  }
params <- combine.params(params=params)
cl <- scorelist$cl
cl.other <- scorelist$cl.other
if(!is.null(cl.other)) cl.other <- factor(cl.other)
nfeatures <- scorelist$nfeatures
if(length(plot.disc)==2){
  n1 <- plot.disc[1]
  n2 <- plot.disc[2]
  if(is.null(xlab))xlab <- paste("Discriminant function", n1)
  if(is.null(ylab))ylab <- paste("Discriminant function", n2)
} else stop("plot.disc must be a vector of length 2")
if(!is.factor(cl))cl <- factor(cl)
levnames <- levels(cl)
fitscores <- scorelist$scores
other.scores <- scorelist$other
ngp <- length(levnames)
n1lim <- range(fitscores[,n1])
n2lim <- range(fitscores[,n2])
if(!is.null(cl.other)){
  n1lim <- range(c(n1lim, other.scores[,n1]))
  n2lim <- range(c(n2lim, other.scores[,n2]))
  levnum <- unclass(cl.other)
  levnames.other <- levels(cl.other)
  intlev.other <- unclass(cl.other)
  ngp.other <- length(levels(cl.other))
}
n1 <- plot.disc[1]; n2 <- plot.disc[2]
intlev <- unclass(cl)
oldpar <- par(lwd=1)
on.exit(par(oldpar))
eqscplot(n1lim, n2lim, type="n",
         xlab=xlab, ylab=ylab, ratio=ratio, ...)
with(params$points,
     points(fitscores[,n1], fitscores[,n2], col=col[intlev],
            pch=pch[intlev], cex=cex, lwd=lwd))
```

```
if(!is.null(cl.other))
  with(params$other,
       points(other.scores[,n1], other.scores[,n2],
              pch=pch[intlev.other],
              col=col[intlev.other],
              cex=cex, lwd=lwd))
if(!is.null(cl.circle)){
  cl.circle <- factor(cl.circle[circle])
  lev.circle <- levels(cl.circle)
  with(params$circle,
       points(fitscores[circle, n1], fitscores[circle,n2], pch=pch,
              cex=cex, col=col[unclass(cl.circle)], lwd=lwd))
}
par(xpd=TRUE)
chw <- par()$cxy[1]
chh <- par()$cxy[2]
par(lwd=1.5)
ypos <- par()$usr[4]
xmid <- mean(par()$usr[1:2])
top.pos <- 0
mtext(side=3, line=(top.pos+1), paste(prefix.title,
       nfeatures, "features"), cex=cex.title, adj=adj.title)
ypos.legend <- ypos+(top.pos-0.45)*chh*0.8

if(join.legends&!is.null(cl.other)){
  leg.info <- legend(xmid, ypos.legend, xjust=0.5, yjust=0, plot=FALSE,
                     x.intersp=0.5, ncol=ngp, legend=levnames,
                     pt.lwd=params$points$lwd,
                     pt.cex=params$points$cex,
                     cex=params$legend$cex,
                     pch=params$points$pch)
  legother.info <- legend(xmid, ypos.legend, xjust=0.5, yjust=0,
                          plot=FALSE, x.intersp=0.5,
                          ncol=ngp.other, legend=levnames.other,
                          pt.lwd=params$other$lwd,
                          pt.cex=params$other$cex,
                          cex=params$legend$cex.other,
                          pch=params$other$pch)
  leftoff <- 0.5*legother.info$rect$w-0.5*chw
  rightoff <- 0.5*leg.info$rect$w+0.5*chw
  ypos.other <- ypos.legend
}
else {
  leftoff <- 0
  rightoff <- 0
  ypos.other <- ypos+(top.pos-1.5)*chh*0.8
}
legend(xmid-leftoff, ypos.legend, xjust=0.5, yjust=0,
       bty="n", pch=params$points$pch,
       x.intersp=0.5, col=params$points$col, ncol=ngp,
       legend=levnames,
       pt.lwd=params$points$lwd,
       pt.cex=params$points$cex,
```

```
        cex=params$legend$cex)
    par(lwd=1)
    if(!is.null(cl.other))
      lego.info <- legend(xmid+rightoff, ypos.other, xjust=0.5, yjust=0,
                          pch=params$other$pch, x.intersp=0.5,
                          col=params$other$col, ncol=ngp.other,
                          pt.lwd=params$other$lwd,
                          pt.cex=params$other$cex,
                          legend=levnames.other,
                          cex=params$legend$cex.other,
                          bty="n")
    if(!is.null(cl.other)&join.legends)
      text(lego.info$rect$left+c(0.4*chw,lego.info$rect$w-0.25*chw),
           rep(ypos.other,2)+0.8*chh, labels=c("(",")"),
           cex=params$legend$cex,
           lwd=params$legend$lwd, bty="n")
    par(lwd=params$circle$lwd)
    if(!is.null(cl.circle))if(lev.circle[1]!=""){
      pch.circle <- params$circle$pch
      xy <- par()$usr[circle.pos+c(1,3)]
      legend(xy[1], xy[2],
             xjust=adj.circle[1], yjust=circle.pos[2], bty="n", x.intersp=0.5,
             pch=rep(pch.circle,length(lev.circle)), col=params$circle$col,
             ncol=1, legend=lev.circle, cex=0.85, pt.cex=1.5)
    }
    par(lwd=1, xpd=FALSE)
    if(plot.folds){
      mtext(side=1, line=1.25, "Discriminant function 1", outer=T)
      mtext(side=2, line=1.25, "Discriminant function 2", outer=T)
    }
  }
```

`simulateScores` *Generate linear discriminant scores from random data, after selection*

Description

Simulates the effect of generating scores from random data, possibly with predicted scores calculates also for additional 'observations'

Usage

```
simulateScores(nrows = 7129, cl = rep(1:3, c(19, 10, 2)), x = NULL, cl.other = NULL,
               x.other = NULL, nfeatures = 15, dimen=2, seed = NULL)
```

Arguments

`nrows`	number of rows of random data matrix
`cl`	classifying factor

x	data matrix, by default randomly generated
cl.other	classifying factor for additional observations
x.other	additional observations
nfeatures	number of features to select (by default uses aov F-statistic)
dimen	number of sets of discriminant scores to retain (at most one less than number of levels of cl)
seed	set, if required, so that calculations can be reproduced

Value

scores	matrix of scores
cl	classifying factor
other	matrix of 'other' scores
cl.other	classifying factor for scores.other
nfeatures	number of features used in generating the scores

Note

NB: Prior to 0.53, this function made (wrongly) a random selection of features.

Author(s)

John Maindonald

Examples

```
scorelist <- simulateScores(nrows=500, cl=rep(1:3, c(19,10,2)))
plot(scorelist$scores, col=unclass(scorelist$cl), pch=16)


## The function is currently defined as
simulateScores <-
  function (nrows = 7129, cl = rep(1:3, c(19, 10, 2)), x = NULL,
            cl.other = NULL, x.other = NULL, nfeatures = 15, dimen = 2,
            seed = NULL)
{
  if (!is.null(seed))
    set.seed(seed)
  m <- length(cl)
  m.other <- length(cl.other)
  if (is.null(x)) {
    x <- matrix(rnorm(nrows * m), nrow = nrows)
    rownames(x) <- paste(1:nrows)
  }
  else nrows <- dim(x)[1]
  if (is.null(x.other)) {
    x.other <- matrix(rnorm(nrows * m.other), nrow = nrows)
    rownames(x.other) <- paste(1:nrows)
```

```
  }
  if (is.numeric(cl))
    cl <- paste("Gp", cl, sep = "")
  if(!is.null(cl.other)){
    if (is.numeric(cl.other))
      cl.other <- paste("Gp", cl.other, sep = "")
    cl.other <- factor(cl.other)
  }
  cl <- factor(cl)
  if (dimen > length(levels(cl)) - 1)
    dimen <- length(levels(cl)) - 1
  ordfeatures <- orderFeatures(x, cl = cl, values = TRUE)
  stat <- ordfeatures$stat[1:nfeatures]
  ord.use <- ordfeatures$ord[1:nfeatures]
  xUse.ord <- data.frame(t(x[ord.use, ]))
  xUseOther.ord <- data.frame(t(x.other[ord.use, ]))
  ordUse.lda <- lda(xUse.ord, grouping = cl)
  scores <- predict(ordUse.lda, dimen = dimen)$x
  if(!is.null(cl.other))
    scores.other <- predict(ordUse.lda, newdata = xUseOther.ord,
                            dimen = dimen)$x else
  scores.other <- NULL
  invisible(list(scores = scores, cl = cl, other = scores.other,
                 cl.other = cl.other, nfeatures = nfeatures))
}
```

Index