Package ‘regtools’

October 14, 2022

Version 1.7.0

Title Regression and Classification Tools

Maintainer Norm Matloff <matloff@cs.ucdavis.edu>

Depends R (>= 3.5.0),FNN,gtools

Imports R.utils,mvtnorm,sandwich,MASS,car,data.table,glmnet,rje,text2vec, polyreg

Suggests knitr, rmarkdown, OpenImageR, cdparcoord, keras, magick, partools

VignetteBuilder knitr

License GPL (>= 2)

Description Tools for linear, nonlinear and nonparametric regression and classification. Novel graphical methods for assessment of parametric models using nonparametric methods. One vs. All and All vs. All multiclass classification, optional class probabilities adjustment. Nonparametric regression (k-NN) for general dimension, local-linear option. Nonlinear regression with Eickert-White method for dealing with heteroscedasticity. Utilities for converting time series to rectangular form. Utilities for conversion between factors and indicator variables. Some code related to ``Statistical Regression and Classification: from Linear Models to Machine Learning'', N. Matloff, 2017, CRC, ISBN 9781498710916.

URL https://github.com/matloff/regtools

BugReports https://github.com/matloff/regtools/issues

NeedsCompilation no

Author Norm Matloff [aut, cre] (<https://orcid.org/0000-0001-9179-6785>),
Robin Yancey [aut],
Bochao Xin [ctb],
Kenneth Lee [ctb],
Rongkui Han [ctb]

Repository CRAN

Date/Publication 2022-03-30 17:30:02 UTC

R topics documented:

`regtools-package` *Overview and Package Reference Guide*

Description

This package provides a broad collection of functions useful for regression and classification analysis, and machine learning.

Function List

Parametric modeling:

- nonlinear regression: nlshc
- ridge regression: ridgelm, plot
- missing values (also see our **toweranNA** package): lmac,makeNA,coef.lmac,vcov.lmac,pcac

Diagnostic plots:

- regression diagnostics: parvsnonparplot, nonparvsxplot, nonparvarplot
- other: boundaryplot, nonparvsxplot

Classification:

- unbalanced data: classadjust (see **UnbalancedClasses.md**)
- All vs. All: avalogtrn, avalogpred
- k-NN reweighting: exploreExpVars, plotExpVars, knnFineTune

Machine learning (also see qeML package):

- k-NN: kNN, kmin, knnest, knntrn, preprocessx, meany, vary, loclin, predict, kmin, pwplot, bestKperPoint, knnFineTune
- neural networks: krsFit,multCol
- advanced grid search: fineTuning, fineTuningPar, plot.tuner, knnFineTune
- loss: l1, l2, MAPE, ROC

Dummies and R factors Utilities:

- conversion between factors and dummies: dummiesToFactor, dummiesToInt, factorsToDummies, factorToDummies, factorTo012etc, dummiesToInt, hasFactors, charsToFactors, makeAllNumeric
- dealing with superset and subsets of factors: toSuperFactor, toSubFactor

Statistics:

- mm

Matrix:

- multCols, constCols

Time series:

- convert rectangular to TS: TStoX

Text processing:

- textToXY

Misc.:

- scaling: mmscale, unscale
- data frames: catDFRow, tabletofakedf
- R: getNamedArgs, ulist
- discretize

`courseRecords` *Records from several offerings of a certain course.*

Description

The data are in the form of an R list. Each element of the list corresponds to one offering of the course. Fields are: Class level; major (two different computer science majors, LCSI in Letters and Science and ECSE in engineering); quiz grade average (scale of 4.0, A+ counting as 4.3); homework grade average (same scale); and course letter grade.

`currency` *Pre-Euro Era Currency Fluctuations*

Description

From Wai Mun Fong and Sam Ouliaris, "Spectral Tests of the Martingale Hypothesis for Exchange Rates", Journal of Applied Econometrics, Vol. 10, No. 3, 1995, pp. 255-271. Weekly exchange rates against US dollar, over the period 7 August 1974 to 29 March 1989.

`day,day1` *Bike sharing data.*

Description

This is the Bike Sharing dataset (day records only) from the UC Irvine Machine Learning Dataset Repository. Included here with permission of Dr. Hadi Fanaee.

The `day` data is as on UCI; `day1` is modified so that the numeric weather variables are on their original scale.

The `day2` is the same as `day1`, except that `dteday` has been removed, and `season`, `mnth`, `weekday` and `weathersit` have been converted to R factors.

See https://archive.ics.uci.edu/ml/datasets/bike+sharing+dataset for details.

english — *English vocabulary data*

Description

The Stanford WordBank data on vocabulary acquisition in young children. The file consists of about 5500 rows. (There are many NA values, though, and only about 2800 complete cases.) Variables are age, birth order, sex, mother's education and vocabulary size.

factorsToDummies — *Factor Conversion Utilities*

Description

Utilities from converting back and forth between factors and dummy variables.

Usage

```
xyDataframeToMatrix(xy)
dummiesToInt(dms,inclLast=FALSE)
factorToDummies(f,fname,omitLast=FALSE,factorInfo=NULL)
factorsToDummies(dfr,omitLast=FALSE,factorsInfo=NULL,dfOut=FALSE)
dummiesToFactor(dms,inclLast=FALSE)
charsToFactors(dtaf)
factorTo012etc(f,earlierLevels = NULL)
discretize(x,endpts)
getDFclasses(dframe)
hasCharacters(dfr)
hasFactors(x)
toAllNumeric(w,factorsInfo=NULL)
toSubFactor(f,saveLevels,lumpedLevel="zzzOther")
toSuperFactor(inFactor,superLevels)
```

Arguments

`dfOut`	If TRUE, return a data frame, otherwise a matrix.
`dms`	Matrix or data frame of dummy columns.
`inclLast`	When forming a factor from dummies, include the last dummy as a level if this is TRUE.
`xy`	A data frame mentioned for prediction, "Y" in last column.
`saveLevels`	In collapsing a factor, which levels to retain.
`lumpedLevel`	Name of new level to be created from levels not retained.
`x`	A numeric vector, except in `hasFactors`, where it is a data frame.

endpts	Vector to be used as breaks in call to cut. To avoid NAs, range of the vector must cover the range of the input vector.
f	A factor.
inFactor	Original factor, to be extended.
superLevels	New levels to be added to the original factor.
earlierLevels	Previous levels found for this factor.
fname	A factor name.
dfr	A data frame.
w	A data frame.
dframe	A data frame, for which we wish to find the column classes.
omitLast	If TRUE, then generate only k-1 dummies from k factor levels.
factorsInfo	Attribute from output of factorsToDummies.
factorInfo	Attribute from output of factorToDummies.
dtaf	A data frame.

Details

Many R users prefer to express categorical data as R factors, or often work with data that is of this type to begin with. On the other hand, many regression packages, e.g. **lars**, disallow factors. These utilities facilitate conversion from one form to another.

Here is an overview of the roles of the various functions:

- `factorToDummies`: Convert one factor to dummies, yielding a matrix of dummies corresponding to that factor.
- `factorsToDummies`: Convert all factors to dummies, yielding a matrix of dummies, corresponding to all factors in the input data frame.
- `dummiesToFactor`: Convert a set of related dummies to a factor.
- `factorTo012etc`: Convert a factor to a numeric code, starting at 0.
- `dummiesToInt`: Convert a related set of dummies to a numeric code, starting at 0.
- `charsToFactors`: Convert all character columns in a data frame to factors.
- `toAllNumeric`: Convert all factors in a data frame to dummies, yielding a new version of the data frame, including its original nonfactor columns.
- `toSubFactor`: Coalesce some levels of a factor, yielding a new factor.
- `toSuperFactor`: Add levels to a factor. Typically used in prediction contexts, in which a factor in a data point to be predicted does not have all the levels of the same factor in the training set.

  \item `xyDataframeToMatrix`: Given a data frame to be used in a training set, with "Y" a factor in the last column, change to all numeric, with dummies in place of all "X" factors and in place of the "Y" factor.

The optional argument `factorsInfo` is intended for use in prediction contexts. Typically a set of new cases will not have all levels of the factor in the training set. Without this argument, only an incomplete set of dummies would be generated for the set of new cases.

A key point about changing factors to dummies is that, for later prediction after fitting a model in our training set, one needs to use the same transformations. Say a factor has levels 'abc', 'de' and 'f' (and `omitLast = FALSE`). If we later have a set of say two new cases to predict, and their values for this factor are 'de' and 'f', we would generate dummies for them but not for 'abc', incompatible with the three dummies used in the training set.

Thus the factor names and levels are saved in attributes, and can be used as input: The relations are as follows:

- `factorsToDummies` calls `factorToDummies` on each factor it finds in its input data frame
- `factorToDummies` outputs and later inputs `factorsInfo`
- `factorsToDummies` outputs and later inputs `factorsInfo`

Other functions:

- `getDFclasses`: Return a vector of the classes of the columns of a data frame.
- `discretize`: Partition range of a vector into (not necessarily equal-length) intervals, and construct a factor from the labels of the intervals that the input elements fall into.
- `hasCharacters, hasFactors`: Logical scalars, TRUE if the input data frame has any character or factor columns.

Value

The function `factorToDummies` returns a matrix of dummy variables, while `factorsToDummies` returns a new version of the input data frame, in which each factor is replaced by columns of dummies. The function `factorToDummies` is similar, but changes character vectors to factors.

Author(s)

Norm Matloff

Examples

```
x <- factor(c('abc','de','f','de'))
xd <- factorToDummies(x,'x')
xd
#      x.abc x.de
# [1,]     1    0
# [2,]     0    1
# [3,]     0    0
# [4,]     0    1
# attr(,"factorInfo")
# attr(,"factorInfo")$fname
# [1] "x"
#
# attr(,"factorInfo")$omitLast
# [1] TRUE
```

```
#
# attr(,"factorInfo")$fullLvls
# [1] "abc" "de"  "f"
w <- factor(c('de','abc','abc'))
wd <- factorToDummies(w,'x',factorInfo=attr(xd,'factorInfo'))
wd
#      x.abc x.de
# [1,]     0    1
# [2,]     1    0
# [3,]     1    0
# attr(,"factorInfo")
# attr(,"factorInfo")$fname
# [1] "x"
#
# attr(,"factorInfo")$omitLast
# [1] TRUE
#
# attr(,"factorInfo")$fullLvls
# [1] "abc" "de"  "f"
```

`falldetection` *Fall Detection Data*

Description

Detection falls in the elderly via physiological measurements. Obtained from Kaggle, but is no longer there.

`fineTuning,knnFineTune` *Grid Search Plus More*

Description

Adds various extra features to grid search for specified tuning parameter/hyperparameter combinations: There is a plot() function, using parallel coordinates graphs to show trends among the different combinations; and Bonferroni confidence intervals are computed to avoid p-hacking. An experimental smoothing facility is also included.

Usage

```
fineTuning(dataset,pars,regCall,nCombs=NULL,specCombs=NULL,nTst=500,
   nXval=1,up=TRUE,k=NULL,dispOrderSmoothed=FALSE,
   showProgress=TRUE,...)
## S3 method for class 'tuner'
plot(x,...)
knnFineTune(data,yName,k,expandVars,ws,classif=FALSE,seed=9999)
fineTuningPar(cls,dataset,pars,regCall,nCombs=NULL,specCombs=NULL,
   nTst=500,nXval=1,up=TRUE,k=NULL,dispOrderSmoothed=FALSE)
```

Arguments

`...`	Arguments to be passed on by `fineTuning` or `plot.tuner`.
`x`	Output object from `fineTuning`.
`cls`	A `parallel` cluster.
`dataset`	Data frame etc. containing the data to be analyzed.
`data`	The data to be analyzed.
`yName`	Quoted name of "Y" in the column names of `data`.
`expandVars`	Indices of columns in `data` to be weighted in distance calculations.
`ws`	Weights to be used for `expandVars`.
`classif`	Set to TRUE for classification problems.
`seed`	Seed for random number generation.
`pars`	R list, showing the desired tuning parameter values.
`regCall`	Function to be called at each parameter combination, performing the model fit etc.
`nCombs`	Number of parameter combinations to run. If Null, all will be run.
`nTst`	Number of data points to be in the test set.
`nXval`	Number of folds to be run for a given data partition and parameter combination.
`k`	Nearest-neighbor smoothing parameter.
`up`	If TRUE, display results in ascending order of performance value.
`dispOrderSmoothed`	Display in order of smoothed results.
`showProgress`	If TRUE, print each output line as it becomes ready.
`specCombs`	A data frame in which the user specifies # hyperparameter parameter combinations to evaluate.

Details

The user specifies the values for each tuning parameter in `pars`. This leads to a number of possible combinations of the parameters. In many cases, there are more combinations than the user wishes to try, so `nCombs` of them will be chosen at random.

For each combination, the function will run the analysis specified by the user in `regCall`. The latter must have the call form

`ftnName(dtrn,dtst,cmbi`

Again, note that it is `fineTuning` that calls this function. It will provide the training and test sets `dtrn` and `dtst`, as well as `cmbi` ("combination i"), the particular parameter combination to be run at this moment.

Each chosen combination is run in `nXval` folds. All specified combinations are run fully, as opposed to a directional "hill descent" search that hopes it might eliminate poor combinations early in the process.

The function `knnFineTune` is a wrapper for `fineTuning` for k-NN problems.

The function `plot.tuner` draws a parallel coordinates plot to visualize the grid. The argument `x` is the output of `fineTuning`. Arguments to specify in the ellipsis are: `col` is the column to be plotted; `disp` is the number to display, with `0`, `-m` and `+m` meaning cases with the `m` smallest 'smoothed' values, all cases and the `m` largest values of 'smoothed', respectively; `jit` avoids plotting coincident lines by adding jitter in the amount `jit * range(x) * runif(n,-0.5,0.5)`.

Value

Object of class **"tuner'**. Contains the grid results, including upper bounds of approximate one-sided 95 univariate and Bonferroni-Dunn (adjusted for the number of parameter combinations).

Author(s)

Norm Matloff

Examples

```
# mlb data set, predict weight using k-NN, try various values of k

tc <- function(dtrn,dtst,cmbi,...)
{
   knnout <- kNN(dtrn[,-3],dtrn[,3],dtst[,-3],as.integer(cmbi[1]))
   preds <- knnout$regests
   mean(abs(preds - dtst[,3]))
}

data(mlb)
mlb <- mlb[,3:6]
mlb.d <- factorsToDummies(mlb)
fineTuning(mlb.d,list(k=c(5,25)),tc,nTst=100,nXval=2)
```

---

knnest,meany,vary,loclin,predict.knn,preprocessx,kmin,parvsnonparplot,nonparvsxplot,l1,l2,kNN,bestKpe

k-NN Nonparametric Regression and Classification

Description

Full set of tools for k-NN regression and classification, including both for direct usage and as tools for assessing the fit of parametric models.

Usage

```
kNN(x,y,newx=x,kmax,scaleX=TRUE,PCAcomps=0,expandVars=NULL,expandVals=NULL,
   smoothingFtn=mean,allK=FALSE,leave1out=FALSE, classif=FALSE,
   startAt1=TRUE,saveNhbrs=FALSE,savedNhbrs=NULL)
knnest(y,xdata,k,nearf=meany)
preprocessx(x,kmax,xval=FALSE)
```

```
meany(nearIdxs,x,y,predpt)
mediany(nearIdxs,x,y,predpt)
vary(nearIdxs,x,y,predpt)
loclin(nearIdxs,x,y,predpt)
## S3 method for class 'knn'
predict(object,...)
kmin(y,xdata,lossftn=l2,nk=5,nearf=meany)

parvsnonparplot(lmout,knnout,cex=1.0)
nonparvsxplot(knnout,lmout=NULL)
nonparvarplot(knnout,returnPts=FALSE)
l2(y,muhat)
l1(y,muhat)
MAPE(yhat,y)
bestKperPoint(x,y,maxK,lossFtn="MAPE",classif=FALSE)
kNNallK(x,y,newx=x,kmax,scaleX=TRUE,PCAcomps=0,
   expandVars=NULL,expandVals=NULL,smoothingFtn=mean,
   allK=FALSE,leave1out=FALSE,classif=FALSE,startAt1=TRUE)
kNNxv(x,y,k,scaleX=TRUE,PCAcomps=0,smoothingFtn=mean,
   nSubSam=500)
knnest(y,xdata,k,nearf=meany)
loclogit(nearIdxs,x,y,predpt)
mediany(nearIdxs,x,y,predpt)
exploreExpVars(xtrn, ytrn, xtst, ytst, k, eVar, maxEVal, lossFtn,
    eValIncr = 0.05, classif = FALSE, leave1out = FALSE)
plotExpVars(xtrn,ytrn,xtst,ytst,k,eVars,maxEVal,lossFtn,
   ylim,eValIncr=0.05,classif=FALSE,leave1out=FALSE)
```

Arguments

`nearf`	Function to be applied to a neighborhood.
`ylim`	Range of Y values for plot.
`lossFtn`	Loss function for plot.
`eVar`	Variable to be expanded.
`eVars`	Variables to be expanded.
`maxEVal`	Maximum expansion value.
`eValIncr`	Increment in range of expansion value.
`xtrn`	Training set for X.
`ytrn`	Training set for Y.
`xtst`	Test set for X.
`ytst`	Test set for Y.
`nearIdxs`	Indices of the neighbors.
`nSubSam`	Number of folds.
`x`	"X" data, predictors, one row per data point, in the training set.

`y`	Response variable data in the training set. Vector or matrix, the latter case for vector-valued response, e.g. multiclass classification. In that case, can be a vector, either (0,1,2,...,) or (1,2,3,...), which automatically is converted into a matrix of dummies.
`newx`	New data points to be predicted. If NULL in `kNN`, compute regression functions estimates on `x` and save for future prediction with `predict.kNN`
`scaleX`	If TRUE, call `scale` on `x` and `newx`
`PCAcomps`	If positive, transform `x` and `newx` by PCA, using the top `PCAcomps` principal components. Disabled.
`expandVars`	Indices of columns in `x` to expand.
`expandVals`	The corresponding expansion values.
`smoothingFtn`	Function to apply to the "Y" values in the set of nearest neighbors. Built-in choices are `meany`, `mediany`, `vary` and `loclin`.
`allK`	If TRUE, find regression estimates for all `k` through `kmax`. Currently disabled.
`leave1out`	If TRUE, omit the 1-nearest neighbor from analysis
`classif`	If TRUE, compute the predicted class labels, not just the regression function values
`startAt1`	If TRUE, class labels start at 1, else 0.
`k`	Number of nearest neighbors
`saveNhbrs`	If TRUE, place output of `FNN::get.knnx` into `nhbrs` of component in return value
`savedNhbrs`	If non-NULL, this is the `nhbrs` component in the return value of a previous call; `newx` must be the same in both calls
`...`	Needed for consistency with generic. See Details below for ‘arguments.
`xdata`	X and associated neighbor indices. Output of `preprocessx`.
`object`	Output of `knnest`.
`predpt`	One point on which to predict, as a vector.
`kmax`	Maximal number of nearest neighbors to find.
`maxK`	Maximal number of nearest neighbors to find.
`xval`	Cross-validation flag. If TRUE, then the set of nearest neighbors of a point will not include the point itself.
`lossftn`	Loss function to be used in cross-validation determination of "best" `k`.
`nk`	Number of values of `k` to try in cross-validation.
`lmout`	Output of `lm`.
`knnout`	Output of `knnest`.
`cex`	R parameter to control dot size in plot.
`muhat`	Vector of estimated regression function values.
`yhat`	Vector of estimated regression function values.
`returnPts`	If TRUE, return matrix of plotted points.

Details

The kNN function is the main tool here; knnest is being deprecated. (Note too qeKNN, a wrapper for kNN; more on this below.) Here are the capabilities:

In its most basic form, the function will input training data and output predictions for new cases newx. By default this is done for a single value of the number k of nearest neighbors, but by setting allK to TRUE, the user can request that it be done for all k through the specified maximum.

In the second form, newx is set to NULL in the call to kNN. No predictions are made; instead, the regression function is estimated on all data points in x, which are saved in the return value. Future new cases can then be predicted from this saved object, via predict.kNN (called via the generic predict). The call form is predict(knnout,newx,newxK, with a default value of 1 for newxK.

In this second form, the closest k points to the newx in x are determined as usual, but instead of averaging their Y values, the average is taken over the fitted regression estimates at those points. In this manner, there is almost no computational cost in the prediction stage.

The second form is intended more for production use, so that neighbor distances need not be repeatedly recomputed.

Nearest-neighbor computation can be time-consuming. If more than one value of k is anticipated, for the same x, y and newx, first run with the largest anticipated value of k, with saveNhbrs set to TRUE. Then for other values of k, set savedNhbrs to the nhbrs component in the return value of the first call.

In addition, a novel feature allows the user to weight some predictors more than others. This is done by scaling the given predictor up or down, according to a specified value. Normally, this should be done with scaleX = TRUE, which applies scale() to the data. In other words, first we create a "level playing field" in which all predictors have standard deviation 1.0, then scale some of them up or down.

Alternatives are provided to calculating the mean Y in the given neighborhood, such as the median and the variance, the latter of possible use in dealing with heterogeneity in linear models.

Another choice of note is to allow local-linear smoothing, by setting smoothingFtn to loclin. Here the value of the regression function at a point is predicted from a linear fit to the point's neighbors. This may be especially helpful to counteract bias near the edges of the data. As in any regression fit, the number of predictors should be considerably less than the number of neighbors.

Custom functions for smoothing can easily be written, say following the pattern of loclin.

The main alternative to kNN is qeKNN in the qe* ("quick and easy") series. It is more convenient, e.g. allowing factor inputs, but less flexible.

The functions ovaknntrn and ovaknnpred are multiclass wrappers for knnest and knnpred, thus also deprecated. Here y is coded 0,1,...,m-1 for the m classes.

The tools here can be useful for fit assessment of parametric models. The parvsnonparplot function plots fitted values of parameteric model vs. kNN fitted, nonparvsxplot k-NN fitted values against each predictor, one by one.

The functions l2 and l1 are used to define L2 and L1 loss.

Author(s)

Norm Matloff

Examples

```
x <- rbind(c(1,0),c(2,5),c(0,5),c(3,3),c(6,3))
y <- c(8,3,10,11,4)
newx <- c(0,0)

kNN(x,y,newx,2,scaleX=FALSE)
# $whichClosest
#      [,1] [,2]
# [1,]    1    4
# $regests
# [1] 9.5

kNN(x,y,newx,3,scaleX=FALSE,smoothingFtn=loclin)$regests
# 7.307692

knnout <- kNN(x,y,newx,2,scaleX=FALSE)
knnout
# $whichClosest
#      [,1] [,2]
# [1,]    1    4
# ...

## Not run:
data(mlb)
mlb <- mlb[,c(4,6,5)]  # height, age, weight
# fit, then predict 75", age 21, and 72", age 32
knnout <- kNN(mlb[,1:2],mlb[,3],rbind(c(75,21),c(72,32)),25)
knnout$regests
# [1] 202.72 195.72

# fit now, predict later
knnout <- kNN(mlb[,1:2],mlb[,3],NULL,25)
predict(knnout,c(70,28))
# [1] 186.48

data(peDumms)
names(peDumms)
ped <- peDumms[,c(1,20,22:27,29,31,32)]
names(ped)

# fit, and predict income of a 35-year-old man, MS degree, occupation 101,
# worked 50 weeks, using 25 nearest neighbors
kNN(ped[,-10],ped[,10],c(35,1,0,0,1,0,0,0,1,50),25) $regests
# [1] 67540

# fit, and predict occupation 101 for a 35-year-old man, MS degree,
# wage $55K, worked 50 weeks, using 25 nearest neighbors
z <- kNN(ped[,-c(4:8)],ped[,4],c(35,1,0,1,55,50),25,classif=TRUE)
z$regests
# [1] 0.16  16
z$ypreds
```

```
# [1] 0  class 0, i.e. not occupation 101; round(0.24) = 0,
# computed by user request, classif = TRUE

# the y argument must be either a vector (2-class setting) or a matrix
# (multiclass setting)
occs <- as.matrix(ped[, 4:8])
z <- kNN(ped[,-c(4:8)],occs,c(35,1,0,1,72000,50),25,classif=TRUE)
z$ypreds
# [1] 3   occupation 3, i.e. 102, is predicted

# predict occupation in general; let's bring occ.141 back in (was
# excluded as a predictor due to redundancy)
names(peDumms)
#  [1] "age"     "cit.1"   "cit.2"   "cit.3"   "cit.4"   "cit.5"   "educ.1"
#  [8] "educ.2"  "educ.3"  "educ.4"  "educ.5"  "educ.6"  "educ.7"  "educ.8"
# [15] "educ.9"  "educ.10" "educ.11" "educ.12" "educ.13" "educ.14" "educ.15"
# [22] "educ.16" "occ.100" "occ.101" "occ.102" "occ.106" "occ.140" "occ.141"
# [29] "sex.1"   "sex.2"   "wageinc" "wkswrkd" "yrentry"
occs <- as.matrix(peDumms[,23:28])
z <- kNN(ped[,-c(4:8)],occs,c(35,1,0,1,72000,50),25,classif=TRUE)
z$ypreds
# [1] 3   prediction is occ.102

# try weight age 0.5, wkswrked 1.5; use leave1out to avoid overfit
knnout <- kNN(ped[,-10],ped[,10],ped[,-10],25,leave1out=TRUE)
mean(abs(knnout$regests - ped[,10]))
# [1] 25341.6

# use of the weighted distance feature; deweight age by a factor of 0.5,
# put increased weight on weeks worked, factor of 1.5
knnout <- kNN(ped[,-10],ped[,10],ped[,-10],25,
   expandVars=c(1,10),expandVals=c(0.5,1.5),leave1out=TRUE)
mean(abs(knnout$regests - ped[,10]))
# [1] 25196.61




## End(Not run)
```

---

krsFit *Tools for Neural Networks*

Description

Tools to complement existing neural networks software, notably a more "R-like" wrapper to fitting data with R's **keras** package.

Usage

```
krsFit(x,y,hidden,acts=rep("relu",length(hidden)),learnRate=0.001,
   conv=NULL,xShape=NULL,classif=TRUE,nClass=NULL,nEpoch=30,
   scaleX=TRUE,scaleY=TRUE)
krsFitImg(x,y,hidden=c(100,100),acts=rep("relu",length(hidden)),
    nClass,nEpoch=30)
## S3 method for class 'krsFit'
predict(object,...)
diagNeural(krsFitOut)
```

Arguments

`object`	An object of class 'krsFit'.
`...`	Data points to be predicted, 'newx'.
`x`	X data, predictors, one row per data point, in the training set. Must be a matrix.
`y`	Numeric vector of Y values. In classification case must be integers, not an R factor, and take on the values 0,1,2,..., `nClass`-1.
`hidden`	Vector of number of units per hidden layer, or the rate for a dropout layer.
`acts`	Vector of names of the activation functions, one per hidden layer. Choices inclde 'relu', 'sigmoid', 'tanh', 'softmax', 'elu', 'selu'.
`learnRate`	Learning rate.
`conv`	R list specifying the convolutional layers, if any.
`xShape`	Vector giving the number of rows and columns, and in the convolutional case with multiple channels, the number of channels.
`classif`	If TRUE, indicates a classification problem.
`nClass`	Number of classes.
`nEpoch`	Number of epochs.
`krsFitOut`	An object returned by `krstFit`.
`scaleX`	If TRUE, scale X columns.
`scaleY`	If TRUE, scale Y columns.

Details

The `krstFit` function is a wrapper for the entire pipeline in fitting the R **keras** package to a dataset: Defining the model, compiling, stating the inputs and so on. As a result, the wrapper allows the user to skip those details (or not need to even know them), and define the model in a manner more familiar to R users.

The paired `predict.krsFit` takes as its first argument the output of `krstFit`, and `newx`, the points to be predicted.

Author(s)

Norm Matloff

Examples

```
## Not run:
library(keras)
data(peDumms)
ped <- peDumms[,c(1,20,22:27,29,32,31)]
# predict wage income
x <- ped[,-11]
y <- ped[,11]
z <- krsFit(x,y,c(50,50,50),classif=FALSE,nEpoch=25)
preds <- predict(z,x)
mean(abs(preds-y))  # something like 25000

x <- ped[,-(4:8)]
y <- ped[,4:8]
y <- dummiesToInt(y,FALSE) - 1
z <- krsFit(x,y,c(50,50,0.20,50),classif=TRUE,nEpoch=175,nClass=6)
preds <- predict(z,x)
mean(preds == y)   # something like 0.39

# obtain MNIST training and test sets; the following then uses the
# example network of

# https://databricks-prod-cloudfront.cloud.databricks.com/
# public/4027ec902e239c93eaaa8714f173bcfc/2961012104553482/
# 4462572393058129/1806228006848429/latest.html

# converted to use the krsFit wrapper

x <- mntrn[,-785] / 255
y <- mntrn[,785]
xShape <- c(28,28)

# define convolutional layers
conv1 <- list(type='conv2d',filters=32,kern=3)
conv2 <- list(type='pool',kern=2)
conv3 <- list(type='conv2d',filters=64,kern=3)
conv4 <- list(type='pool',kern=2)
conv5 <- list(type='drop',drop=0.5)

# call wrapper, 1 dense hidden layer of 128 units, then dropout layer
# with proportion 0.5
z <- krsFit(x,y,conv=list(conv1,conv2,conv3,conv4,conv5),c(128,0.5),
   classif=TRUE,nClass=10,nEpoch=10,xShape=c(28,28),scaleX=FALSE,scaleY=FALSE)

# try on test set
preds <- predict(z,mntst[,-785]/255)
mean(preds == mntst[,785])  # 0.98 in my sample run


## End(Not run)
```

`lmac,makeNA,coef.lmac,vcov.lmac,pcac,loglinac,tbltofakedf`
Available Cases Method for Missing Data

Description

Various estimators that handle missing data via the Available Cases Method

Usage

```
lmac(xy,nboot=0)
makeNA(m,probna)
NAsTo0s(x)
ZerosToNAs(x,replaceVal=0)
## S3 method for class 'lmac'
coef(object,...)
## S3 method for class 'lmac'
vcov(object,...)
pcac(indata,scale=FALSE)
loglinac(x,margin)
tbltofakedf(tbl)
```

Arguments

`replaceVal`	Value to be replaced by NA.
`xy`	Matrix or data frame, X values in the first columns, Y in the last column.
`indata`	Matrix or data frame.
`x`	Matrix or data frame, one column per variable.
`nboot`	If positive, number of bootstrap samples to take.
`probna`	Probability that an element will be NA.
`scale`	If TRUE, call `cor` instead of `cov`.
`tbl`	An R table.
`m`	Number of synthetic NAs to insert.
`object`	Output from `lmac`.
`...`	Needed for consistency with generic function. Not used.
`margin`	A list of vectors specifying the model, as in `loglin`.

Details

The Available Cases (AC) approach applies to statistical methods that depend only on products of k of the variables, so that cases having non-NA values for those k variables can be used, as opposed to using only cases that are fully intact in all variables, the Complete Cases (CC) approach. In the case of linear regression, for instance, the estimated coefficients depend only on covariances between the variables (both predictors and response). This approach assumes thst the cases with missing values have the same distribution as the intact cases.

The lmac function forms OLS estimates as with lm, but applying AC, in contrast to lm, which uses the CC method.

The pcac function is an AC substitute for prcomp. The data is centered, corresponding to a fixed value of center = TRUE in prcomp. It is also scaled if scale is TRUE, corresponding scale = TRUE in prcomp. Due to AC, there is a small chance of negative eigenvalues, in which case stop will be called.

The loglinac function is an AC substitute for loglin. The latter takes tables as input, but loglinac takes the raw data. If you have just the table, use tbltofakedf to regenerate a usable data frame.

The makeNA function is used to insert random NA values into data, for testing purposes.

Value

For lmac, an object of class lmac, with components

- coefficients, as with lm; accessible directly or by calling coef, as with lm
- fitted.values, as with lm
- residuals, as with lm
- r2, (unadjusted) R-squared
- cov, for nboot > 0 the estimated covariance matrix of the vector of estimated regression coefficients; accessible directly or by calling vcov, as with lm

For pcac, an R list, with components

- sdev, as with prcomp
- rotation, as with prcomp

For loglinac, an R list, with components

- param, estimated coefficients, as in loglin
- fit, estimated expected call counts, as in loglin

Author(s)

Norm Matloff

Examples

```
n <- 25000
w <- matrix(rnorm(2*n),ncol=2)  # x and epsilon
x <- w[,1]
y <- x + w[,2]
# insert some missing values
nmiss <- round(0.1*n)
x[sample(1:n,nmiss)] <- NA
nmiss <- round(0.2*n)
```

```
y[sample(1:n,nmiss)] <- NA
acout <- lmac(cbind(x,y))
coef(acout)  # should be near pop. values 0 and 1
```

`ltrfreqs` *Letter Frequencies*

Description

This is data consists of capital letter frequencies obtained at http://www.math.cornell.edu/~mec/2003-2004/cryptography/subs/frequencies.h tml

---

`misc` *Utilities*

Description

Various helper functions.

Usage

```
replicMeans(nrep,toReplic,timing=FALSE)
stdErrPred(regObj,xnew)
pythonBlankSplit(s)
stopBrowser(msg = stop("msg not supplied"))
doPCA(x,pcaProp)
PCAwithFactors(x, nComps = ncol(x))
ulist(lst)
prToFile(filename)
partTrnTst(fullData,nTest=min(1000,round(0.2*nrow(fullData))))
findOverallLoss(regests,y,lossFtn = MAPE)
getNamedArgs(argVec)
multCols(x,cols,vals)
probIncorrectClass(yhat, y, startAt1 = TRUE)
propMisclass(y,yhat)
```

Arguments

`regests`	Fitted regression estimates, training set.
`y`	Y values, training set.
`yhat`	Predicted Y values
`startAt1`	TRUE if indexing starts at 1, FALSE if starting at 0.
`lossFtn`	Loss functin.
`fullData`	A data frame or matrix.

nTest	Number of rows for the test set.
filename	Name of output file.
lst	An R list.
x	Matrix or data frame.
pcaProp	Fraction in [0,1], specifying number of PCA components to compute, in terms of fraction of total variance.
nComps	Number of PCA components.
regObj	An object of class 'lm' or similar, for which there is a vcov generic function.
xnew	New X value to be predicted.
nrep	Number of replications.
s	A character string.
toReplic	Function call(s), as a quoted string, separated by semicolons if more than one call.
timing	If TRUE, find average elapsed time over the replicates.
msg	Character string, error message for existing debug browser.
argVec	R list or vector with named elements.
cols	A set of column numbers.
vals	A set of positive expansion numbers.

Details

The function PCAwithFactors is a wrapper for stats::prcomp, to be used on data frames that contain at least on R factor.

Value

The function PCAwithFactors returns an object of class 'PCAwithFactors'. with components pcout, the object returned by the wrapped call to prcomp; factorsInfo, factor conversion information to be used with predict; and preds, the PCA version of x.

The function getNamedArgs will assign in the caller's space variables with the names and values in argVec.

Author(s)

Norm Matloff

Examples

```
w <- list(a=3,b=8)
getNamedArgs(w)
a
b
u <- c(5,12,13)
names(u) <- c('x','y','z')
```

```
getNamedArgs(u)
x
y
z
```

mlb *Major Leage Baseball player data set.*

Description

Heights, weights, ages etc. of major league baseball players. A new variable has been added, consolidating positions into Infielders, Outfielders, Catchers and Pitchers.

Included here with the permission of the UCLA Statistics Department.

mlens *MovieLens User Summary Data*

Description

The MovieLens dataset, https://grouplens.org/, is a standard example in the recommender systems literature. Here we give demographic data for each user, plus the mean rating and number of ratings. One may explore, for instance, the relation between ratings and age.

mm *Method of Moments, Including Possible Regression Terms*

Description

Method of Moments computation for almost any statistical problem that has derivatives with respect to theta. Capable of handling models that include parametric regression terms, but not need be a regression problem. (This is not *Generalized* Method of Moments; see the package **gmm** for the latter.)

Usage

```
mm(m,g,x,init=rep(0.5,length(m)),eps=0.0001,maxiters=1000)
```

Arguments

- `m` Vector of sample moments, "left-hand sides" of moment equations.
- `g` Function of parameter estimates, forming the "right-hand sides." This is a multivariate-valued function, of dimensionality equal to that of `m`.
- `init` Vector of initial guesses for parameter estimates. If components are named, these will be used as labels in the output.
- `eps` Convergence criterion.
- `maxiters` Maximum number of iterations.
- `x` Input data.

Details

Standard Newton-Raphson methods are used to solve for the parameter estimates, with `numericDeriv` being used to find the approximate derivatives.

Value

R list consisting of components `tht`, the vector of parameter estimates, and `numiters`, the number of iterations performed.

Author(s)

Norm Matloff

Examples

```
x <- rgamma(1000,2)
m <- c(mean(x),var(x))
g <- function(x,theta) {  # from theoretical properties of gamma distr.
   g1 <-  theta[1] / theta[2]
   g2 <-  theta[1] / theta[2]^2
   c(g1,g2)
}
# should output about 2 and 1
mm(m,g,x)

## Not run:
library(mfp)
data(bodyfat)
# model as a beta distribution
g <- function(x,theta) {
   t1 <- theta[1]
   t2 <- theta[2]
   t12 <- t1 + t2
   meanb <- t1 / t12
   m1 <- meanb
   m2 <- t1*t2 / (t12^2 * (t12+1))
   c(m1,m2)
}
```

```
x <- bodyfat$brozek/100
m <- c(mean(x),var(x))
# about 4.65 and 19.89
mm(m,g,x)

## End(Not run)
```

---

multiclass routines *Classification with More Than 2 Classes*

Description

Tools for multiclass classification, parametric and nonparametric.

Usage

```
avalogtrn(trnxy,yname)
ovaknntrn(trnxy,yname,k,xval=FALSE)
avalogpred()
classadjust(econdprobs,wrongprob1,trueprob1)
boundaryplot(y01,x,regests,pairs=combn(ncol(x),2),pchvals=2+y01,cex=0.5,band=0.10)
```

Arguments

`pchvals`	Point size in base-R graphics.
`trnxy`	Data matrix, Y last.
`xval`	If TRUE, use leaving-one-out method.
`y01`	Y vector (1s and 0s).
`regests`	Estimated regression function values.
`x`	X data frame or matrix.
`pairs`	Two-row matrix, column i of which is a pair of predictor variables to graph.
`cex`	Symbol size for plotting.
`band`	If `band` is non-NULL, only points within `band`, say 0.1, of est. P(Y = 1) are displayed, for a contour-like effect.
`yname`	Name of the Y column.
`k`	Number of nearest neighbors.
`econdprobs`	Estimated conditional class probabilities, given the predictors.
`wrongprob1`	Incorrect, data-provenanced, unconditional P(Y = 1).
`trueprob1`	Correct unconditional P(Y = 1).

Details

These functions aid classification in the multiclass setting.

The function `boundaryplot` serves as a visualization technique, for the two-class setting. It draws the boundary between predicted Y = 1 and predicted Y = 0 data points in 2-dimensional feature space, as determined by the argument `regests`. Used to visually assess goodness of fit, typically running this function twice, say one for `glm` then for `kNN`. If there is much discrepancy and the analyst wishes to still use glm(), he/she may wish to add polynomial terms.

The functions not listed above are largely deprecated, e.g. in favor of `qeLogit` and the other `qe`-series functions.

Author(s)

Norm Matloff

Examples

```
## Not run:


data(oliveoils)
oo <- oliveoils[,-1]

# toy example
set.seed(9999)
x <- runif(25)
y <- sample(0:2,25,replace=TRUE)
xd <- preprocessx(x,2,xval=FALSE)
kout <- ovaknntrn(y,xd,m=3,k=2)
kout$regest  # row 2:  0.0,0.5,0.5
predict(kout,predpts=matrix(c(0.81,0.55,0.15),ncol=1))  # 0,2,0or2
yd <- factorToDummies(as.factor(y),'y',FALSE)
kNN(x,yd,c(0.81,0.55,0.15),2)  # predicts 0, 1or2, 2

data(peDumms)  # prog/engr data
ped <- peDumms[,-33]
ped <- as.matrix(ped)
x <- ped[,-(23:28)]
y <- ped[,23:28]
knnout <- kNN(x,y,x,25,leave1out=TRUE)
truey <- apply(y,1,which.max) - 1
mean(knnout$ypreds == truey)  # about 0.37
xd <- preprocessx(x,25,xval=TRUE)
kout <- knnest(y,xd,25)
preds <- predict(kout,predpts=x)
hats <- apply(preds,1,which.max) - 1
mean(yhats == truey)  # about 0.37

data(peFactors)
# discard the lower educ-level cases, which are rare
edu <- peFactors$educ
```

```
numedu <- as.numeric(edu)
idxs <- numedu >= 12
pef <- peFactors[idxs,]
numedu <- numedu[idxs]
pef$educ <- as.factor(numedu)
pef1 <- pef[,c(1,3,5,7:9)]

# ovalog
ovaout <- ovalogtrn(pef1,"occ")
preds <- predict(ovaout,predpts=pef1[,-3])
mean(preds == factorTo012etc(pef1$occ))  # about 0.39

# avalog

avaout <- avalogtrn(pef1,"occ")
preds <- predict(avaout,predpts=pef1[,-3])
mean(preds == factorTo012etc(pef1$occ))  # about 0.39

# knn

knnout <- ovalogtrn(pef1,"occ",25)
preds <- predict(knnout,predpts=pef1[,-3])
mean(preds == factorTo012etc(pef1$occ))  # about 0.43

data(oliveoils)
oo <- oliveoils
oo <- oo[,-1]
knnout <- ovaknntrn(oo,'Region',10)
# predict a new case that is like oo1[1,] but with palmitic = 950
newx <- oo[1,2:9,drop=FALSE]
newx[,1] <- 950
predict(knnout,predpts=newx)  # predicts class 2, South


## End(Not run)
```

---

newadult *UCI adult income data set, adapted*

Description

This data set is adapted from the Adult data from the UCI Machine Learning Repository, which was in turn adapted from Census data on adult incomes and other demographic variables. The UCI data is used here with permission from Ronny Kohavi.

The variables are:

- `gt50`, which converts the original `>50K` variable to an indicator variable; 1 for income greater than $50,000, else 0

- `edu`, which converts a set of education levels to approximate number of years of schooling
- `age`
- `gender`, 1 for male, 0 for female
- `mar`, 1 for married, 0 for single

Note that the education variable is now numeric.

`nlshc` *Heteroscedastic Nonlinear Regression*

Description

Extension of `nls` to the heteroscedastic case.

Usage

```
nlshc(nlsout,type='HC')
```

Arguments

`nlsout`	Object of type 'nls'.
`type`	Eickert-White algorithm to use. See documentation for **nls**.

Details

Calls `nls` but then forms a different estimated covariance matrix for the estimated regression coefficients, applying the Eickert-White technique to handle heteroscedasticity. This then gives valid statistical inference in that setting.

Some users may prefer to use `nlsLM` of the package **minpack.lm** instead of `nls`. This is fine, as both functions return objects of class 'nls'.

Value

Estimated covariance matrix

Author(s)

Norm Matloff

References

Zeileis A (2006), Object-Oriented Computation of Sandwich Estimators. *Journal of Statistical Software*, **16**(9), 1–16, https://www.jstatsoft.org/v16/i09/.

Examples

```
# simulate data from a setting in which mean Y is
# 1 / (b1 * X1 + b2 * X2)
n <- 250
b <- 1:2
x <- matrix(rexp(2*n),ncol=2)
meany <- 1 / (x %*% b)  # reg ftn
y <- meany + (runif(n) - 0.5) * meany  # heterosced epsilon
xy <- cbind(x,y)
xy <- data.frame(xy)
# see nls() docs
nlout <- nls(X3 ~ 1 / (b1*X1+b2*X2),
   data=xy,start=list(b1 = 1,b2=1))
nlshc(nlout)
```

---

oliveoils *Italian olive oils data set.*

Description

Italian olive oils data set, as used in *Graphics of Large Datasets: Visualizing a Million*, by Antony Unwin, Martin Theus and Heike Hofmann, Springer, 2006. Included here with permission of Dr. Martin Theus.

Penrose Linear *Penrose-Inverse Linear Models and Polynomial Regression*

Description

Provides mininum-norm solutions to linear models, identical to OLS in standard situations, but allowing exploration of overfitting in the overparameterized case. Also provides a wrapper for the polynomial case.

Usage

```
penroseLM(d,yName)
penrosePoly(d,yName,deg,maxInteractDeg=deg)
ridgePoly(d,yName,deg,maxInteractDeg=deg)
## S3 method for class 'penroseLM'
predict(object,...)
## S3 method for class 'penrosePoly'
predict(object,...)
```

Arguments

`...`	Arguments for the `predict` functions.
`d`	Dataframe, training set.
`yName`	Name of the class labels column.
`deg`	Polynomial degree.
`maxInteractDeg`	Maximum degree of interaction terms.
`object`	A value returned by `penroseLM` or `penrosePoly`.

Details

First, provides a convenient wrapper to the **polyreg** package for polynomial regression. (See `qePoly` here for an even higher-level wrapper.) Note that this computes true polynomials, with cross-product/interaction terms rather than just powers, and that dummy variables are handled properly (to NOT compute powers).

Second, provides a tool for exploring the "double descent" phenomenon, in which prediction error may improve upon fitting past the interpolation point.

Author(s)

Norm Matloff

`phoneme` *Phoneme Data*

Description

Phoneme detection, 2 types. Features are from harmonic analysis of th voice. From OpenML, `https://www.openml.org/d/1489`.

`prgeng` *Silicon Valley programmers and engineers data*

Description

This data set is adapted from the 2000 Census (5% sample, person records). It is mainly restricted to programmers and engineers in the Silicon Valley area. (Apparently due to errors, there are some from other ZIP codes.)

There are three versions:

- `prgeng`, the original data, with categorical variables, e.g. Occupation, in their original codes
- `peDumms`, same but with categorical variables converted to dummies; due to the large number of levels the birth and PUMA data is not included

- peFactors, same but with categorical variables converted to factors
- pef, same as peFactors, but having only columns for age, education, occupation, gender, wage income and weeks worked. The education column has been collapsed to Master's degree, PhD and other.

The variable codes, e.g. occupational codes, are available from https://usa.ipums.org/usa/volii/occ2000.shtml. (Short code lists are given in the record layout, but longer ones are in the appendix Code Lists.)

The variables are:

- age, with a U(0,1) variate added for jitter
- cit, citizenship; 1-4 code various categories of citizens; 5 means noncitizen (including permanent residents)
- educ: 01-09 code no college; 10-12 means some college; 13 is a bachelor's degree, 14 a master's, 15 a professional degree and 16 is a doctorate
- occ, occupation
- birth, place of birth
- wageinc, wage income
- wkswrkd, number of weeks worked
- yrentry, year of entry to the U.S. (0 for natives)
- powpuma, location of work
- gender, 1 for male, 2 for female

Usage

```
data(prgeng)
data(peDumms)
data(peFactors)
```

---

quizDocs *Course quiz documents*

Description

This data is suitable for NLP analysis. It consists of all the quizzes I've given in undergraduate courses, 143 quizzes in all.

It is available in two forms. First, quizzes is a data.frame, 143 rows and 2 columns. Row i consists of a single character vector comprising the entire quiz i, followed by the course name (as an R factor). The second form is an R list, 143 elements. Each list element is a character vector, one vector element per line of the quiz.

The original documents were LaTeX files. They have been run through the detex utility to remove most LaTeX commands, as well as removing the LaTeX preambles separately.

The names of the list elements are the course names, as follows:

ECS 50: a course in machine organization

ECS 132: an undergraduate course in probabilistic modeling

ECS 145: a course in scripting languages (Python, R)

ECS 158: an undergraduate course in parallel computation

ECS 256: a graduate course in probabilistic modeling

`ridgelm,plot.rlm` *Ridge Regression*

Description

Similar to `lm.ridge` in `MASS` packaged included with R, but with a different kind of scaling and a little nicer plotting.

Usage

```
ridgelm(xy,lambda = seq(0.01,1,0.01),mapback=TRUE)
## S3 method for class 'rlm'
plot(x,y,...)
```

Arguments

`xy`	Data, response variable in the last column.
`lambda`	Vector of desired values for the ridge parameter.
`mapback`	If TRUE, the scaling that had been applied to the original data will be map back to the original scale, so that the estimated regression coefficients are now on the scale of the original data.
`x`	Object of type 'rlm', output of `ridgelm`.
`y`	Needed for consistency with the generic. Not used.
`...`	Needed for consistency with the generic. Not used.

Details

Centers and scales the predictors X, and centers the response variable Y. Computes X'X and then solves [(X'X)/n + lambda I]b = X'Y/n for b. The 1/n factors are important, making the diagonal elements of (X'X)/n all 1s and thus facilitating choices for the lambdas in a manner independent of the data.

Calling `plot` on the output of `ridgelm` dispatches to `plot.rlm`, thus diplaying the ridge traces.

Value

The function `ridgelm` returns an object of class 'rlm', with components `bhats`, the estimated beta vectors, one column per lambda value, and `lambda`, a copy of the input.

Author(s)

Norm Matloff

SwissRoll *Swiss Roll*

Description

See http://people.cs.uchicago.edu/~dinoj/manifold/swissroll.html for this version of Swiss Roll.

Running `data(SwissRoll)` produces an object `sw`.

textToXY,textToXYpred *Tools for Text Classification*

Description

"R-style," classification-oriented wrappers for the **text2vec** package.

Usage

```
textToXY(docs,labels,kTop=50,stopWords='a')
textToXYpred(ttXYout,predDocs)
```

Arguments

`docs`	Character vector, one element per document.
`predDocs`	Character vector, one element per document.
`labels`	Class labels, as numeric, character or factor. NULL is used at the prediction stage.
`kTop`	The number of most-frequent words to retain; 0 means retain all.
`stopWords`	Character vector of common words, e.g. prepositions to delete. Recommended is `tm::stopwords('english')`.
`ttXYout`	Output object from `textToXY`.

Details

A typical classification/machine learning package will have as arguments a feature matrix X and a labels vector/factor Y. For a "bag of words" analysis in the text case, each row of X would be a document and each column a word.

The functions here are basically wrappers for generating X. Wrappers are convenient in that:

- The **text2vec** package is rather arcane, so a "R-style" wrapper would be useful.
- The **text2vec** are not directly set up to do classification, so the functions here provide the "glue" to do that.

The typical usage pattern is thus:

- Run the documents vector and labels vector/factor through `textToXY`, generating X and Y.
- Apply your favorite classification/machine learning package p to X and Y, returning o.
- When predicting a new document d, run o and d through `textToXY`, producing x.
- Run `x` on p's `predict` function.

Value

The function `textToXY` returns an R list with components `x` and `y` for X and Y, and a copy of the input `stopWords`.

The function `textToXY` returns X.

Author(s)

Norm Matloff

`TStoX` *Transform Time Series to Rectangular Form*

Description

Input a time series and transform it to a form suitable for prediction using `lm` etc.

Usage

```
TStoX(x,lg)
TStoXmv(xmat,lg,y)
```

Arguments

`x`	A vector.
`lg`	Lag, a positive integer.
`xmat`	A matrix, data frame etc., a multivariate time series. Each column is a time series, over a common time period.
`y`	A time series, again on that common time period. If NULL in `TStoXmv`, then `y` is set to `x` (i.e. for a univariate time series in which older values predict newer ones).

Details

Similar to `stats::embed`, but in lagged form, with applications such as `lm` in mind.

`TStoX` is for transforming vectors, while `TStoXmv` handles the multivariate time series case. Intended for use with `lm` or other regression/machine learning model, predicting `y[i]` from observations `i-lg, i-lg+1,...,i-1`.

Value

As noted, the idea is to set up something like `lm(Y ~ X)`. Let `m` denote length of `x`, and in the matrix input case, the number of rows in `xmat`. Let `p` be 1 in the vector case, `ncol(xmat)` in the matrix case. The return value is a matrix with `m-lg` rows. There will be `p*lg+1` columns, with "Y," the numbers to be predicted in the last column.

In the output in the multivariate case, let k denote `ncol(xmat)`. Then the first k columns of the output will be the k series at lag `lg`, the second k columns will be the k series at lag `lg-1`, ..., and the `lg`-th set of k columns will be the k series at lag 1,

Author(s)

Norm Matloff

Examples

```
x1 <- c(5,12,13,8,88,6)
x2 <- c(5,4,3,18,168,0)
y <- 1:6
xmat <- cbind(x1,x2)

TStoX(x1,2)
#      [,1] [,2] [,3]
# [1,]    5   12   13
# [2,]   12   13    8
# [3,]   13    8   88
# [4,]    8   88    6

xy <- TStoXmv(xmat,2,y)
xy
#      [,1] [,2] [,3] [,4] [,5]
# [1,]    5    5   12    4    3
# [2,]   12    4   13    3    4
# [3,]   13    3    8   18    5
# [4,]    8   18   88  168    6

lm(xy[,5] ~ xy[,-5])
# Coefficients:
# (Intercept)    xy[, -5]1    xy[, -5]2    xy[, -5]3    xy[, -5]4
#       -65.6          3.2         18.2         -3.2           NA
# need n > 7 here for useful lm() call, but this illustrates the idea
```

`unscale` *Miscellaneous Utilities*

Description

Utilities.

Usage

```
unscale(scaledx,ctrs=NULL,sds=NULL)
mmscale(m,scalePars=NULL,p=NULL)
catDFRow(dfRow)
constCols(d)
allNumeric(lst)
```

Arguments

`scaledx`	A matrix.
`m`	A matrix.
`ctrs`	Take the original means to be `ctrs`
`lst`	An R list.
`sds`	Take the original standard deviations to be `sds`
`dfRow`	A row in a data frame.
`d`	A data frame or matrix.
`scalePars`	If not NULL, a 2-row matrix, with column `i` storing the min and max values to be used in scaling column `i` of `m`. Typically, one has previously called `mmscale` on a dataset and saved the resulting scale parameters, and we wish to use those same scale parameters on new data.
`p`	If `m` is a vector, this specifies the number of columns it should have as a matrix. The code will try to take care of this by itself if `p` is left at NULL.

Details

The function `mmscale` is meant as a better-behaved alternative to `scale`. Using minimum and maximum values, it maps variables to [0,1], thus avoiding the problems arising from very small standard deviations in `scale`.

The function `catDFRow` nicely prints a row of a data frame.

The function `constCols` determines which columns of a data frame or matrix are constant, if any.

Value

The function `unscale` returns the original object to which `scale` had been applied. Or, the attributes `ctrs` and `sds` can be specified by the user.

Author(s)

Norm Matloff

weatherTS *Weather Time Series*

Description

Various measurements on weather variables collected by NASA. Downloaded via `nasapower`; see that package for documentation.

xyzPlot *Misc. Graphics*

Description

Graphics utiliites.

Usage

```
xyzPlot(xyz,clrs=NULL,cexText=1.0,xlim=NULL,ylim=NULL,
   xlab=NULL,ylab=NULL,legendPos=NULL,plotType='l')
```

Arguments

`xyz`	A matrix or data frame of at least 3 columns, the first 3 serving as 'x', 'y' and 'z' coordinates of points to be plotted. Grouping, if any, is specified in column 4, in which case `xyz` must be a data frame.
`clrs`	Colors to be used in the grouped case.
`cexText`	Text size, proportional to standard.
`xlim`	As in `plot`.
`ylim`	As in `plot`.
`xlab`	As in `plot`.
`ylab`	As in `plot`.
`legendPos`	As in `legend`.
`plotType`	Coded 'l' for lines, 'p' for points.

Details

A way to display 3-dimensional data in 2 dimensions. For each plotted point (x,y), a z value is written in text over the point. A grouping variable is also allowed, with different colors used to plot different groups.

A group (including the entire data in the case of one group) can be displayed either as a polygonal line, or just as a point cloud. The user should experiment with different argument settings to get the most visually impactful plot.

Author(s)

Norm Matloff

Examples

```
## Not run:

xyzPlot(mtcars[,c(3,6,1)],plotType='l',cexText=0.75)
xyzPlot(mtcars[,c(3,6,1)],plotType='p',cexText=0.75)
xyzPlot(mtcars[,c(3,6,1)],plotType='l',cexText=0.75)
xyzPlot(mtcars[,c(3,6,1,2)],clrs=c('red','darkgreen','blue'),plotType='l',cexText=0.75)


## End(Not run)
```

`yell10k` *New York Taxi Data*

Description

From public data on New York City taxi trips.

Index