Package ‘qeML’

July 22, 2025

Version 1.1

Title Quick and Easy Machine Learning Tools

Maintainer Norm Matloff <nsmatloff@ucdavis.edu>

Depends R (>= 3.5.0),regtools (>= 0.8.0),gtools,rmarkdown,tufte

Imports grf,gbm,toweranNA,tm,rpart,rpart.plot,partools,FOCI

Suggests knitr,partykit,randomForest,ranger,e1071,JOUSBoost,lightgbm, keras,neuralnet,polyreg,glmnet,umap,reticulate,party,pROC,xgboost,ROCR, autoimage,deepnet,ncvreg,uwot,cdparcoord

VignetteBuilder knitr

License GPL (>= 2)

Description The letters 'qe' in the package title stand for ``quick and easy,'' alluding to the convenience goal of the package. We bring together a variety of machine learning (ML) tools from standard R packages, providing wrappers with a simple, convenient, and uniform interface.

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

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

NeedsCompilation no

Author Norm Matloff [aut, cre] (ORCID: <https://orcid.org/0000-0001-9179-6785>)

Repository CRAN

Date/Publication 2023-11-09 15:20:02 UTC

Contents

Advanced Plots 2
CancerMenopause 3
courseRecords 3
currency 3
day,day1 4

Double Descent 4
empAttrition 5
english 5
EPI GrowthData 6
Feature Model Select 6
forest500 7
iranChurn 8
lsa 8
ltrfreqs 8
mlb 8
mlens 9
newadult 9
nyctaxi 9
oliveoils 10
Prediction with Missing Values 10
prgeng 11
qe-Series Predictive Functions 12
quizDocs 20
R Factor Utilities 21
ThyroidDisease 22
Utilities 22
Variable Importance Measures 24
weatherTS 25

Index **26**

`Advanced Plots` *Advanced Plots*

Description

Miscellaneous specialized plots.

Usage

```
plotPairedResids(data,qeOut)
plotClassesUMAP(data,classVar)
qeFreqParcoord(dataName,k=25,opts=NULL)
```

Arguments

`data`	A data frame or equivalent.
`qeOut`	An object returned from one of the qe-series predictive functions..
`classVar`	Name of the column containing class information.
`dataName`	Quoted name of a data frame.
`k`	Number of nearest neighbors.
`opts`	Options to be passed to `discparcoord`.

Details

The `plotPairedResids` function plots model residuals against pairs of features, for example for model validation. Pairs are chosen randomly.

The function `qeFreqParcoord` is a `qeML` interface to the `cdparcoord` package.

Author(s)

Norm Matloff

Examples

```
## Not run:
data(pef)
linout <- qeLin(pef,'wageinc')
plotPairedResids(pef,linout)

## End(Not run)
```

`CancerMenopause` *Swedish breast cancer.*

Description

Data on incidence of breast cancer among women in Sweden. Goal of the study was to investigate whether the incidence increases with the onset of menopause.

Included here with the permission of Prof. Yudi Pawitan, Karolinska Institutet, Stockholm.

`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.

`Double Descent` *Double Descent Phenomenon*

Description

Belkin and others have shown that some machine learning algorithms exhibit surprising behavior when in overfitting settings. The classic U-shape of mean loss plotted against model complexity may be followed by a surprise second "mini-U."

Alternatively, one might keep the model complexity fixed while varying the number of data points n, including over a region in which n is smaller than the complexity value of the model. The surprise here is that mean loss may actually increase with n in the overfitting region.

The function `doubleD` facilitates easy exploration of this phenomenon.

Usage

```
doubleD(qeFtnCall,xPts,nReps,makeDummies=NULL,classif=FALSE)
```

Arguments

`qeFtnCall`	Quoted string; somewhere should include 'xPts[i]'.
`xPts`	Range of values to be used in the experiments, e.g. a vector of degrees for polynomial models.
`nReps`	Number of repetitions for each experiment, typically the number in the holdout set.
`makeDummies`	If non-NULL, call `regtools::factorsToDummies` on the dataset of this name. This avoids the problem of some levels of a factor appearing in the holdout set but not the training set.
`classif`	Set TRUE if this is a classification problem.

Details

The function will run the code in `qeFtnCall` `nreps` times for each level specified in `xPts`, recording the test and training error in each case. So, for each level, we will have a mean test and training error.

Value

Each call in `xPts` results in one line in the return value of `doubleD`. The return matrix can then be plotted, using the generic `plot.doubleD`. Mean test (red) and training (blue) accuracy will be plotted against `xPts`.

Author(s)

Norm Matloff

Examples

```
   ## Not run:
      data(mlb1)
      hw <- mlb1[,2:3]
      doubleD('qePolyLin(hw,"Weight",deg=xPts[i])',1:20,250)

## End(Not run)
```

`empAttrition` *Employee Attrition Data*

Description

IBM data from Kaggle, https://www.kaggle.com/datasets/pavansubhasht/ibm-hr-analytics-attrition-dataset

Usage

```
data(empAttrition)
```

`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.

EPI GrowthData *EPI Growth Data*

Description

US economic growth measures.

Courtesy of the Economic Policy Institute.

Usage

```
data(EPIWgProduct)
```

Feature Model Select *Feature Selection and Model Building*

Description

Utilties to help build models, both in specific applications such as time series and text analysis, and in general tools..

Usage

```
qeCompare(data,yName,qeFtnList,nReps,opts=NULL,seed=9999)
qeFT(data,yName,qeftn,pars,nCombs,nTst,nXval,showProgress=TRUE)
qeText(data,yName,kTop=50,stopWords=tm::stopwords("english"),
   qeName,opts=NULL,holdout=floor(min(1000,0.1*nrow(data))))
qeTS(lag,data,qeName,opts=NULL,holdout=floor(min(1000,0.1*length(data))))
## S3 method for class 'qeText'
predict(object,newDocs,...)
## S3 method for class 'qeTS'
predict(object,newx,...)
```

Arguments

...	Further arguments.
object	Object returned by a qe-series function.
newx	New data to be predicted.
newDocs	Vector of new documents to be predicted.
lag	number of recent values to use in predicting the next.
qeName	Name of qe-series predictive function, e.g. 'qeRF'.
stopWords	Stop lists to use.
nTst	Number of parameter combinations.
kTop	Number of most-frequent words to use.

`data`	Dataframe, training set. Classification case is signaled via labels column being an R factor.
`yName`	Name of the class labels column.
`holdout`	If not NULL, form a holdout set of the specified size. After fitting to the remaining data, evaluate accuracy on the test set.
`qeFtnList`	Character vector of `qe*` function names.
`nReps`	Number of holdout sets to generate.
`opts`	R list of optional arguments for none, some or all of th functions in `qeFtnList`.
`seed`	Seed for random number generation.
`qeftn`	Quoted string, specifying the name of a qe-series machine learning method.
`pars`	R list of hyperparameter ranges. See `regtools::fineTuning`.
`nCombs`	Number of hyperparameter combinations to run. See `regtools::fineTuning`.
`nXval`	Number of cross-validations to run. See `regtools::fineTuning`.
`showProgress`	If TRUE, show results as they arise. See `regtools::fineTuning`.

Details

Overviews of the functions:

- `qeTs` is a tool for time series modeling
- `qeText` is a tool for textual modeling
- `qeCompare` facilitates comparison among models
- `qeFT` does a random grid search for optimal hyperparameter values

Author(s)

Norm Matloff

Examples

```
data(mlb1)
# predict Weight in the mlb1 dataset, using qeKNN, with k = 5 and 25,
# with 10 cross-validations
qeFT(mlb1,'Weight','qeKNN',list(k=c(5,25)),nTst=100,nXval=10)
```

`forest500` *Subset of the Covertype data.*

Description

Random subset of 500 records.

https://archive.ics.uci.edu/ml/datasets/covertype `iranChurn` *Iranian Customer Churn Data*

Description

From https://github.com/sharmaroshan/Churn-Modelling-Dataset.

Character variables and bernoulli variables have been converted to factors. The first three cols, e.g. customer ID, have been deleted.

The tenure col is apparently length of time with the firm.

`lsa` *Law School Admissions Data*

Description

Law School Admissions dataset from the Law School Admissions Council (LSAC). The dataset was originally collected for a study called 'LSAC National Longitudinal Bar Passage Study' by Linda Wightman in 1998.

Most of the names are self-explanatory, but note that: The two decile scores are class standing in the first and third years of law school, and 'cluster' refers to the reputed quality of the law school. Two variables of particular interest might be the student's score on the Law School Admission Test (LSAT) and a logical variable indicating whether the person passed the bar examination.

Note that the 'age' variable is apparently birth year, e.g. 69 meaning 1969.

`ltrfreqs` *Letter Frequencies*

Description

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

`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.

The `mlb1` version has only Position, Height, Weight and Age.

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.

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.

nyctaxi *New York City Taxi Data*

Description

10,000 records on five variables, extracted from https://data.cityofnewyork.us/Transportation/2018-Yellow-Taxi-Trip-Data/t29m-gskq.

Usage

```
data(nyctaxi)
```

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.

Prediction with Missing Values — *Prediction with Missing Values*

Description

ML methods for prediction in which features are subject to missing values.

Usage

```
qeLinMV(data,yName)
qeLogitMV(data,yName,yesYVal)
qeKNNMV(data,yName,kmax)
## S3 method for class 'qeLinMV'
predict(object,newx,...)
## S3 method for class 'qeLogitMV'
predict(object,newx,...)
## S3 method for class 'qeKNNMV'
predict(object,newx,...)
```

Arguments

`...`	Further arguments.
`object`	An object returned by one of the `qe*MV` functions.
`data`	Dataframe, training set. Classification case is signaled via labels column being an R factor.
`yName`	Name of the class labels column.
`newx`	New data to be predicted.
`kmax`	Number of nearest neighbors in training set.
`yesYVal`	Y value to be considered "yes," to be coded 1 rather than 0.

Details

These are wrappers to the **toweranNA** package. Linear, logistic and kNN interfaces are available.

Author(s)

Norm Matloff

Examples

```
sum(is.na(airquality))  # 44 NAs, good test example
z <- qeKNNMV(airquality,'Ozone',10)
# example of new case, insert an NA in 1st row
aq2 <- airquality[2,-1]
aq2$Wind <- NA
predict(z,aq2)  # 28.1
```

---

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
- `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, coded 'z14', 'z16' and 'zzzOther'. Most cases are in the latter category.
- `svcensus`, same as `pef`, but with the column name 'sex' replaced by 'gender'.

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

`qe-Series Predictive Functions` *Quick-and-Easy Machine Learning Wrappers*

Description

Quick access to machine learning methods, with a very simple interface. "Works right out of the box!": Just one call needed to fit, no preliminary setup of model etc. The simplicity also makes the series useful for teaching.

Usage

```
qeLogit(data,yName,holdout=floor(min(1000,0.1*nrow(data))),yesYVal=NULL)
qeLin(data,yName,noBeta0=FALSE,holdout=floor(min(1000,0.1*nrow(data))))
qeKNN(data,yName,k,scaleX=TRUE,smoothingFtn=mean,yesYVal=NULL,
   expandVars=NULL,expandVals =NULL,holdout=floor(min(1000,0.1*nrow(data))))
qeRF(data,yName,nTree=500,minNodeSize=10,mtry=floor(sqrt(ncol(data)))+1,
   holdout=floor(min(1000,0.1*nrow(data))))
qeRFranger(data,yName,nTree=500,minNodeSize=10,
   mtry=floor(sqrt(ncol(data)))+1,deweightPars=NULL,
   holdout=floor(min(1000,0.1*nrow(data))),yesYVal="")
qeRFgrf(data,yName,nTree=2000,minNodeSize=5,mtry=floor(sqrt(ncol(data)))+1,
   ll=FALSE,lambda=0.1,splitCutoff=sqrt(nrow(data)),
   holdout=floor(min(1000,0.1*nrow(data))))
qeSVM(data,yName,gamma=1.0,cost=1.0,kernel='radial',degree=2,
   allDefaults=FALSE,holdout=floor(min(1000,0.1*nrow(data))))
qeGBoost(data,yName,nTree=100,minNodeSize=10,learnRate=0.1,
   holdout=floor(min(1000,0.1*nrow(data))))
qeAdaBoost(data, yName, treeDepth = 3, nRounds = 100, rpartControl = NULL,
    holdout = floor(min(1000, 0.1 * nrow(data))))
qeLightGBoost(data,yName,nTree=100,minNodeSize=10,learnRate=0.1,
   holdout=floor(min(1000,0.1*nrow(data))))
qeNeural(data,yName,hidden=c(100,100),nEpoch=30,
   acts=rep("relu",length(hidden)),learnRate=0.001,
   conv=NULL,xShape=NULL,
   holdout=floor(min(1000,0.1*nrow(data))))
qeLASSO(data,yName,alpha=1,holdout=floor(min(1000,0.1*nrow(data))))
qePolyLin(data,yName,deg=2,maxInteractDeg = deg,
   holdout=floor(min(1000,0.1*nrow(data))))
qePolyLog(data,yName,deg=2,maxInteractDeg = deg,
   holdout=floor(min(1000,0.1*nrow(data))))
qePCA(data,yName,qeName,opts=NULL,pcaProp,
   holdout=floor(min(1000,0.1*nrow(data))))
qeUMAP(data,yName,qeName,opts=NULL,
   holdout=floor(min(1000,0.1*nrow(data))),scaleX=FALSE,
   nComps=NULL,nNeighbors=NULL)
qeDT(data,yName,alpha=0.05,minsplit=20,minbucket=7,maxdepth=0,mtry=0,
```

```
   holdout=floor(min(1000,0.1*nrow(data))))
qeFOCI(data,yName,numCores=1,parPlat="none",
   yesYLevel=NULL)
qeFOCIrand(data,yName,xSetSize,nXSets)
qeFOCImult(data,yName,numCores=1,
   parPlat="none",coalesce='union')
qeLinKNN(data,yName,k=25,scaleX=TRUE,smoothingFtn=mean,
   expandVars=NULL,expandVals=NULL,
   holdout=floor(min(1000,0.1*nrow(data))))
qePolyLASSO(data,yName,deg=2,maxInteractDeg=deg,alpha=0,
   holdout=floor(min(1000,0.1*nrow(data))))
qeROC(dataIn,qeOut,yLevelName)
qeXGBoost(data,yName,nRounds=250,
   params=list(eta=0.3,max_depth=6,alpha=0),
   holdout=floor(min(1000,0.1*nrow(data))))
qeDeepnet(data,yName,hidden=c(10),activationfun="sigm",
   learningrate=0.8,momentum=0.5,learningrate_scale=1,
   numepochs=3,batchsize=100,hidden_dropout=0,yesYVal=NULL,
   holdout=floor(min(1000,0.1*nrow(data))))
qeRpart(data,yName,minBucket=10,holdout=floor(min(1000,
   0.1*nrow(data))))
qeParallel(data,yName,qeFtnName,dataName,opts=NULL,cls=1,
   libs=NULL,holdout=NULL)
checkPkgLoaded(pkgName,whereObtain='CRAN')
## S3 method for class 'qeParallel'
predict(object,newx,...)
## S3 method for class 'qeLogit'
predict(object,newx,...)
## S3 method for class 'qeLin'
predict(object,newx,useTrainRow1=TRUE,...)
## S3 method for class 'qeKNN'
predict(object,newx,newxK=1,...)
## S3 method for class 'qeRF'
predict(object,newx,...)
## S3 method for class 'qeRFranger'
predict(object,newx,...)
## S3 method for class 'qeRFgrf'
predict(object,newx,...)
## S3 method for class 'qeSVM'
predict(object,newx,...)
## S3 method for class 'qeGBoost'
predict(object,newx,newNTree=NULL,...)
## S3 method for class 'qeLightGBoost'
predict(object,newx,...)
## S3 method for class 'qeNeural'
predict(object,newx,k=NULL,...)
## S3 method for class 'qeLASSO'
predict(object,newx,...)
```

```
## S3 method for class 'qePoly'
predict(object,newx)
## S3 method for class 'qePCA'
predict(object,newx,...)
## S3 method for class 'qeUMAP'
predict(object,newx,...)
## S3 method for class 'qeDeepnet'
predict(object,newx,...)
## S3 method for class 'qeRpart'
predict(object,newx,...)
## S3 method for class 'qeLASSO'
plot(x,...)
## S3 method for class 'qeRF'
plot(x,...)
## S3 method for class 'qeRpart'
plot(x,boxPalette=c("red","yellow","green","blue"),...)
```

Arguments

`...`	Further arguments.
`cls`	Cluster in the sense of **parallel** package. If not of class `cluster`, this is either a positive integer, indicating the desired number of cores, or a character vector, indicating the machines on which the cluster is to be formed.
`libs`	Character vector listing libraries needed to be loaded for `qeFtnName`.
`dataName`	Name of the `data` argument.
`hidden_dropout`	Drop out fraction for hidden layer.
`batchsize`	Batch size.
`numepochs`	Number of iterations to conduct.
`learningrate`	Learning rate.
`momentum`	Momemtum
`learningrate_scale`	Learning rate will be multiplied by this at each iteration, allowing for decay.
`activationfun`	Can be 'sigm', 'tanh' or 'linear'.
`newNTree`	Number of trees to use in prediction.
`newxK`	If predicting new cases, number of nearest neighbors to smooth in the object returned by `qeKNN`.
`useTrainRow1`	If TRUE, take names in `newx` from row 1 in the training data.
`newx`	New data to be predicted.
`object`	An object returned by a qe-series function.
`minsplit`	Minimum number of data points in a node.
`minbucket`	Minimum number of data points in a terminal node.
`minBucket`	Minimum number of data points in a terminal node.
`maxdepth`	Maximum number of levels in a tree.

`qeName`	Name of qe-series predictive function.
`qeFtnName`	Name of qe-series predictive function.
`conv`	R list specifying the convolutional layers, if any.
`deweightPars`	Values for de-emphasizing variables in a tree node split, e.g. 'list(age=0.2,gender=0.5)'.
`allDefaults`	Use all default values of the wrapped function.
`expandVars`	Columns to be emphasized.
`expandVals`	Emphasis values; a value less than 1 means de-emphasis.
`mtry`	Number of variables randomly tried at each split.
`yesYVal`	Y value to be considered "yes," to be coded 1 rather than 0.
`yesYLevel`	Y value to be considered "yes," to be coded 1 rather than 0.
`noBeta0`	No intercept term.
`pcaProp`	Desired proportion of overall variance for the PCs.‘
`data`	Dataframe, training set. Classification case is signaled via labels column being an R factor.
`dataIn`	See `data`.
`qeOut`	Output from a qe-series function.
`yName`	Name of the class labels column.
`holdout`	If not NULL, form a holdout set of the specified size. After fitting to the remaining data, evaluate accuracy on the test set.
`k`	Number of nearest neighbors. In functions other than `qeKNN` for which this is an argument, it is the number of neighbors to use in finding conditional probabilities via `knnCalib`.
`smoothingFtn`	As in `kNN`.
`scaleX`	Scale the features.
`nTree`	Number of trees.
`minNodeSize`	Minimum number of data points in a tree node.
`learnRate`	Learning rate.
`hidden`	Vector of units per hidden layer. Fractional values indicated dropout proportions. Can be specified as a string, e.g. '100,50', for use with `qeFT`.
`nEpoch`	Number of iterations in neural net.
`acts`	Vector of names of the activation functions, one per hidden layer. Choices inclde 'relu', 'sigmoid', 'tanh', 'softmax', 'elu', 'selu'.
`alpha`	In the case of `qeDT`, a p-value cutoff criterion. Otherwise 1 for LASSO, 2 for ridge.
`gamma`	Scale parameter in `e1071::svm`.
`cost`	Cost parameter in `e1071::svm`.
`kernel`	In the case of `qeSVM`, this is One of 'linear','radial','polynomial' and 'sigmoid'.
`degree`	Degree of SVM polynomial kernel, if any.
`opts`	R list of optional arguments for none, some or all of th functions in `qeFtnList`.

nComps	Number of UMAP components to extract.
nNeighbors	Number of nearest neighbors to use in UMAP.
ll	If TRUE, use local linear forest.
lambda	Ridge lambda for local linear forest.
splitCutoff	For leaves smaller than this value, do not fit linear model. Just use the linear model fit to the entire dataset.
xShape	Input X data shape, e.g. c(28,28) for 28x28 grayscale images. Must be non-NULL if conv is.
treeDepth	Number of levels in each tree.
nRounds	Number of boosting rounds.
rpartControl	An R list specifying properties of fitted trees.
numCores	Number of cores to use in parallel computation.
parPlat	Parallel platforParallel platform. Valid values are 'none', 'cluster' (output of parallel::makeCluster), and 'locThreads' (local cores).
xSetSize	Size of subsets of the predictor variables.
nXSets	Number of subsets of the predictor variables.
coalesce	Method for combining variable sets.
deg	Degree of a polynomial.
maxInteractDeg	Maximul degree of interaction terms in a polynomial.
yLevelName	Name of the class to be considered a positive response in a classification problem.
params	Tuning parameters for xgboost, e.g. params=list(eta=0.1,max_depth=8).
boxPalette	Color palette.
pkgName	Name of wrapped package.
whereObtain	Location.
x	A qe-series function return object.

Details

As noted, these functions are intended for quick, first-level analysis of regression/machine learning problems. Emphasis here is on convenience and simplicity.

The idea is that, given a new dataset, the analyst can quickly and easily try fitting a number of models in succession, say first k-NN, then random forests:

```
# built-in data on major league baseball players
> data(mlb)
> mlb <- mlb[,3:6]  # position, height, weight, age

# fit models
> knnout <- qeKNN(mlb,'Weight',k=25)
> rfout <- qeRF(mlb,'Weight')
```

```
# mean abs. pred. error on holdout set, in pounds
> knnout$testAcc
[1] 11.75644
> rfout$testAcc
[1] 12.6787

# predict a new case
> newx <- data.frame(Position='Catcher',Height=73.5,Age=26)
> predict(knnout,newx)
       [,1]
[1,] 204.04
> predict(rfout,newx)
      11
199.1714

# many of the functions include algorithm-specific output
> lassout <- qeLASSO(mlb,'Weight')
holdout set has  101 rows
> lassout$testAcc
[1] 14.27337
> lassout$coefs  # sparse result?
10 x 1 sparse Matrix of class "dgCMatrix"
                                    s1
(Intercept)              -109.2909416
Position.Catcher            0.4408752
Position.First_Baseman      4.8308437
Position.Outfielder         .
Position.Relief_Pitcher     .
Position.Second_Baseman    -0.7846501
Position.Shortstop         -4.2291338
Position.Starting_Pitcher   .
Height                      4.0039114
Age                         0.5352793
```

The `holdout` argument triggers formation of a holdout set and the corresponding cross-validation evaluation of predictive power. Note that if a holdout is formed, the return value will consist of the fit on the training set, not on the full original dataset.

The `qe*` functions do model fit. Each of them has a `predict` method, and some also have a `plot` method.

Arguments for `qe*` are at least:

- `data`
- `yName`
- `holdout`

Typically there are also algorithm-specific hyperparameter arguments.

Arguments for `predict` are at least:

- `object`, the return value from `qe*`
- `newx`, a data frame of points to be predicted

For both the fitting function and the prediction function, there may be additional algorithm-specific parameters; default values are provided.

Some notes on specific functions:

- The function `qeLin` handles not only the usual OLS models but also classification problems as multivariate-outcome linear models. If one's goal is prediction, it can be much faster than `qeLogit`, often with comparable accuracy.
- Regularization in linear/generalized linear models is implemented in `qeLASSO` and other functions with names containing 'LASSO', as well as `qeNCVregCV`. The latter, wrappping the MCP and other regularization methods, wraps the package of the same name.
- Several functions fit polynomial models. The `qePolyLin` function does polynomial regression of the indicated degree. In the above example degree 3 means all terms through degree 3, e.g. `Height * Age^2`. Dummy variables are handled properly, e.g. no powers of a dummy are generatd. The logistic polynomial regression version is `qePolyLog`, and there is a LASSO version, `qePolyLASSO`.
- Several random forests implementations are offered: `qeRF` wraps `randomForest` in the package of the same name; `qeRFranger` wraps `ranger` in the package of the same name; `qeRFgrf` wraps `regression_forest` and `ll_regression_forest` in **grf** (the latter does local linear smoothing). There is also `qeDT`, using the **party** package.
- Several implementations of gradient boosting are offered, including `qeGBoost` using the **gbm** package, `qelightGBoost` using **lightgbm**, and `qeXGBoost` wrapping **xgboost**.
- Several functions involve dimension reduction/feature selection. Pre-mapping to lower-dimensional manifolds can be done via `qePCA` and `qeUMAP`. For instance, the former will first extract the specified number of principal components, then fit the user's desired ML model, say k-NN (`qeKNN`) or gradient boosting (`qeGBoost`).
- The `qeFOCI` function does feature selection in a basically assumption-free manner. It handles numeric and binary Y (the latter coded 1,0). For categorical Y, use `qeFOCImult`. The function `qeFOCIrand` applies FOCI to many subsets of the input dataset, eventually returning the union of the outputs; this is useful if the dataset has many NA values.
- Neural network models are implemented by `qeNeural` and `qeDeepnet`, based on **keras** and **deepnet**.
- The `qeLinKNN` function offers a hybrid approach. It first fits a linear model, then applies k-Nearest Neighbors to the residuals. The `qePolyLinKNN` function does the same in with a polynomial fit.
- The `qeIso` function is intended mainly for use as a smoothing method in calibration actions.

In most cases, the full basket of options in the wrapped function is not reflected. Use of arguments not presented in the qe function requires direct use the relevant packages.

Value

The value returned by `qe*` functions depends on the algorithm, but with some commonality, e.g. `classif`, a logical value indicating whether the problem was of classification type.

If a holdout set was requested, an additional returned component will be `testAcc`, the accuracy on the holdout set. This will be Mean Absolute Prediction Error in the regression case, and proportion of misclassified cases in the classification case.

The value returned by the `predict` functions is an R list with components as follows:

Classification case:

- `predClasses`: R factor instance of predicted class labels
- `probs`: vector/matrix of class probabilities; in the 2-class case, a vector, the probabilities of Y = 1

Regression case: vector of predicted values

Author(s)

Norm Matloff

Examples

```
# see also 'details' above

## Not run:

data(peFactors)
pef <- peFactors[,c(1,3,5,7:9)]
# most people in the dataset have at least a Bachelor's degree; so let's
# just consider Master's (code 14) and PhD (code 16) as special
pef$educ <- toSubFactor(pef$educ,c('14','16'))

# predict occupation; 6 classes, 100, 101, 102, 106, 140, 141, using SVM
svmout <- qeSVM(pef,'occ',holdout=NULL)
# as example of prediction, take the 8th case, but change the gender and
# age to female and 25; note that by setting k to non-null, we are
# requesting that conditional probabilities be calculated, via
# knnCalib(), here using 25 nearest neighbors
newx <- pef[8,-3]
newx$sex <- '2'
newx$age <- 25
predict(svmout,newx,k=25)
# $predClasses
#   8
# 100
# Levels: 100 101 102 106 140 141
# $dvals
#      102/101    102/100   102/141  102/140  102/106    101/100  101/141
# 8 -0.7774038 -0.5132022 0.9997894 1.003251 0.999688 -0.4023077 1.000419
#    101/140   101/106  100/141  100/140  100/106    141/140    141/106   140/106
# 8 1.000474 0.9997371 1.000088 1.000026 1.000126 0.9460703 -0.4974625 -1.035721
#
# $probs
#       100  101  102  106 140  141
# [1,] 0.24 0.52 0.12 0.08   0 0.04
```

```
#
# so, occupation code 100 is predicted, with a 0.36 conditional
# probability

# if holdout evaluation is desired as well, say 1000 cases, seed 9999:
> svmout <- qeSVM(pef,'occ',holdout=c(1000,9999))
> svmout$testAcc
[1] 0.622  # 62

# linear
# lm() doesn't like numeric factor levels, so prepend an 'a'
pef$occ <- prepend('a',pef$occ)
lmout <- qeLin(pef,'occ')
predict(lmout,pef[1,-3])  # occ 100, prob 0.3316
lmout <- qeLin(pef,'wageinc')
predict(lmout,pef[1,-5])  # 70857.79


## End(Not run)
```

`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

R Factor Utilities *R Factor Utilities*

Description

Utilities to manipulate R factors, extending the ones in `regtools`.

Usage

```
levelCounts(data)
dataToTopLevels(data,lowCountThresholds)
factorToTopLevels(f,lowCountThresh=0)
cartesianFactor(dataName,factorNames,fNameSep = ".")
qeRareLevels(x, yName, yesYVal = NULL)
```

Arguments

`data`	A data frame or equivalent.
`f`	An R factor.
`lowCountThresh`	Factor levels will counts below this value will not be used for this factor.
`lowCountThresholds`	An R list of column names and their corresponding values of `lowCountThresh`.
`dataName`	A quoted name of a data frame or equivalent.
`factorNames`	A vector of R factor names.
`fNameSep`	A character to be used as a delimiter in the names of the levels of the output factor.
`x`	A data frame.
`yName`	Quoted name of the response variable.
`yesYVal`	In the case of binary Y, the factor level to be considered positive.

Details

Often one has an R factor in which one or more levels are rare in the data. This could cause problems, say in performing cross-validation; a level in the test set might be "new," not having appeared in the training set. Toward this end, `factorToTopLevels` will remove rare levels from a factor; `dataToTopLevels` applies this to an entire data frame.

Also toward this end, the function `levelCounts` simply applies `table()` to each column of `data`, returning the result as an R list. (If more than 10 levels, it returns NA.

The function `cartesianFactor` generates a "superfactor" from individual ones; e.g. if factors f1 and f2 have n1 and n2 levels, the output is a new factor with n1 * n2 levels.

The function `qeRareLevels` checks all columns in a data frame in terms of being an R factor with rare levels.

Author(s)

Norm Matloff

Examples

```
data(svcensus)
levelCounts(svcensus)  # e.g. finds there are 15182 men, 4908 women
f1 <- svcensus$gender  # 2 levels
f2 <- svcensus$occ  # 6 levels
z <- cartesianFactor('svcensus',c('gender','occ'))
head(z)
# [1] female.102 male.101   female.102 male.100   female.100 male.100
# 12 Levels: female.100 female.101 female.102 female.106 ... male.141
```

ThyroidDisease *Thyroid Disease*

Description

See OpenML repository, https://www.openml.org/search?type=data&sort=runs&id=38&status=active.

"Thyroid disease records supplied by the Garavan Institute and J. Ross Quinlan, New South Wales Institute, Syndney, Australia. 1987."

Usage

```
data(ThyroidDisease)
```

Utilities *Utilities*

Description

Miscellaneous functions, used mainly internally in the package, but of possible use externally.

Usage

```
buildQEcall(qeFtnName,dataName,yName=NULL,opts=NULL,holdout=NULL,
    holdoutArg=TRUE)
evalr(toexec)
newDFRow(dta,yName,x,dtaRowNum=1)
```

Arguments

`qeFtnName`	Quoted name of a **qeML** predictive function.
`dataName`	Quoted name of a data frame.
`yName`	Quoted name of a column to be predicted.
`opts`	Non-default arguments for the function specified in `qeFtnName`.
`holdout`	Size of holdout set, if any.
`holdoutArg`	A value TRUE means the function specified in `qeFtnName` has an argument 'holdout'.
`toexec`	Quoted string containing an R function call.
`dta`	A data frame.
`x`	An R list specifying fields to be set.
`dtaRowNum`	Row number in 'dta' to be used as a basis.

Details

The function `qeFtnName` does what its name implies: It assembles a string consisting of a **qeML** function call. Typically the latter is then executed via evalr. See for instance the source code of `qeLeaveOut1Var`.

R's generic `predict` function generally required that the input rows match the original training data in name and class. The `newDFRow` function can be used to construct such a row.

Author(s)

Norm Matloff

Examples

```
# function to list all the objects loaded by the specified package
lsp <- function(pkg) {
   cmd <- paste('ls(package:',pkg,')')
   evalr(cmd)
}
lsp('regtools')
# outputs
#  [1] "clusterApply"        "clusterApplyLB"      "clusterCall"
#  [4] "clusterEvalQ"        "clusterExport"       "clusterMap"
# ...
```

`Variable Importance Measures` *Variable Importance Measures*

Description

Various approaches to assessing relative importance of one's features.

Usage

```
qeLeaveOut1Var(data,yName,qeFtnName,nReps,opts=list())
```

Arguments

`data`	Dataframe, training set. Classification case is signaled via labels column being an R factor.
`yName`	Name of the class labels column.
`qeFtnName`	Quoted `qe*` function name.
`nReps`	Number of holdout sets to generate.
`opts`	R list of optional arguments for none, some or all of th functions in `qeFtnList`.

Details

Many methods have been developed assessing relative importance of one's features. A few that we consider most useful are accessible here.

As a quick assessment, the `qeLeave1VarOut` function, with call form as above, simply compares predictive ability with and without the given feature.

Some methods rely on reweighting:

- `qeKNN`
- `qeRFranger`

Others make use of order of entry of a variable into the prediction model:

- `qeFOCI`
- `qeLASSO`

Author(s)

Norm Matloff

Examples

```
data(pef)
qeLeaveOut1Var(pef,'wageinc','qeLin',5)
# in order of impact, wkswrkd largest, then education etc.
```

`weatherTS` *Weather Time Series*

Description

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

Index

Advanced Plots, 2

buildQEcall (Utilities), 22

CancerMenopause, 3
cartesianFactor (R Factor Utilities), 21
checkPkgLoaded (qe-Series Predictive Functions), 12
courseRecords, 3
currency, 3

dataToTopLevels (R Factor Utilities), 21
day (day,day1), 4
day,day1, 4
day1 (day,day1), 4
day2 (day,day1), 4
Double Descent, 4
doubleD (Double Descent), 4

empAttrition, 5
english, 5
EPI GrowthData, 6
EPIWgProduct (EPI GrowthData), 6
evalr, *23*
evalr (Utilities), 22

factorToTopLevels (R Factor Utilities), 21
Feature Model Select, 6
forest500, 7

iranChurn, 8

levelCounts (R Factor Utilities), 21
lsa, 8
ltrfreqs, 8

mlb, 8
mlb1 (mlb), 8
mlens, 9

newAdult (newadult), 9
newadult, 9
newDFRow (Utilities), 22
nyctaxi, 9

oliveoils, 10

pef (prgeng), 11
plot.doubleD (Double Descent), 4
plot.qeLASSO (qe-Series Predictive Functions), 12
plot.qePoly (qe-Series Predictive Functions), 12
plot.qeRF (qe-Series Predictive Functions), 12
plot.qeRpart (qe-Series Predictive Functions), 12
plotClassesUMAP (Advanced Plots), 2
plotPairedResids (Advanced Plots), 2
predict.qeAdaBoost (qe-Series Predictive Functions), 12
predict.qeDeepnet (qe-Series Predictive Functions), 12
predict.qeGBoost (qe-Series Predictive Functions), 12
predict.qeIso (qe-Series Predictive Functions), 12
predict.qeKNN (qe-Series Predictive Functions), 12
predict.qeKNNMV (Prediction with Missing Values), 10
predict.qeLASSO (qe-Series Predictive Functions), 12
predict.qeLightGBoost (qe-Series Predictive Functions), 12
predict.qeLin (qe-Series Predictive Functions), 12
predict.qeLinMV (Prediction with Missing Values), 10
predict.qeLogit (qe-Series Predictive Functions), 12

predict.qeLogitMV *(*Prediction with Missing Values*)*, 10
predict.qeNCVregCV *(*qe-Series Predictive Functions*)*, 12
predict.qeNeural *(*qe-Series Predictive Functions*)*, 12
predict.qeParallel *(*qe-Series Predictive Functions*)*, 12
predict.qePCA *(*qe-Series Predictive Functions*)*, 12
predict.qePoly *(*qe-Series Predictive Functions*)*, 12
predict.qePolyLin *(*qe-Series Predictive Functions*)*, 12
predict.qePolyLinKNN *(*qe-Series Predictive Functions*)*, 12
predict.qePolyLog *(*qe-Series Predictive Functions*)*, 12
predict.qeRF *(*qe-Series Predictive Functions*)*, 12
predict.qeRFgrf *(*qe-Series Predictive Functions*)*, 12
predict.qeRFranger *(*qe-Series Predictive Functions*)*, 12
predict.qeRpart *(*qe-Series Predictive Functions*)*, 12
predict.qeSVM *(*qe-Series Predictive Functions*)*, 12
predict.qeText *(*Feature Model Select*)*, 6
predict.qeTS *(*Feature Model Select*)*, 6
predict.qeUMAP *(*qe-Series Predictive Functions*)*, 12
Prediction with Missing Values, 10
prgeng, 11

qe-Series Predictive Functions, 12
qeAdaBoost *(*qe-Series Predictive Functions*)*, 12
qeCompare *(*Feature Model Select*)*, 6
qeDeepnet *(*qe-Series Predictive Functions*)*, 12
qeDT *(*qe-Series Predictive Functions*)*, 12
qeFOCI *(*qe-Series Predictive Functions*)*, 12
qeFOCImult *(*qe-Series Predictive Functions*)*, 12
qeFOCIrand *(*qe-Series Predictive Functions*)*, 12
qeFreqParcoord *(*Advanced Plots*)*, 2
qeFT *(*Feature Model Select*)*, 6
qeGBoost *(*qe-Series Predictive Functions*)*, 12
qeIso *(*qe-Series Predictive Functions*)*, 12
qeKNN *(*qe-Series Predictive Functions*)*, 12
qeKNNMV *(*Prediction with Missing Values*)*, 10
qeLASSO *(*qe-Series Predictive Functions*)*, 12
qeLeaveOut1Var *(*Variable Importance Measures*)*, 24
qeLightGBoost *(*qe-Series Predictive Functions*)*, 12
qeLin *(*qe-Series Predictive Functions*)*, 12
qeLinKNN *(*qe-Series Predictive Functions*)*, 12
qeLinMV *(*Prediction with Missing Values*)*, 10
qeLogit *(*qe-Series Predictive Functions*)*, 12
qeLogitMV *(*Prediction with Missing Values*)*, 10
qeNCVregCV *(*qe-Series Predictive Functions*)*, 12
qeNeural *(*qe-Series Predictive Functions*)*, 12
qeParallel *(*qe-Series Predictive Functions*)*, 12
qePCA *(*qe-Series Predictive Functions*)*, 12
qePoly *(*qe-Series Predictive Functions*)*, 12
qePolyLASSO *(*qe-Series Predictive Functions*)*, 12
qePolyLin *(*qe-Series Predictive Functions*)*, 12
qePolyLinKNN *(*qe-Series Predictive Functions*)*, 12
qePolyLog *(*qe-Series Predictive Functions*)*, 12
qeRareLevels *(*R Factor Utilities*)*, 21
qeRF *(*qe-Series Predictive Functions*)*, 12
qeRFgrf *(*qe-Series Predictive

Functions), 12
qeRFranger (qe-Series Predictive Functions), 12
qeROC (qe-Series Predictive Functions), 12
qeRpart (qe-Series Predictive Functions), 12
qeSVM (qe-Series Predictive Functions), 12
qeText (Feature Model Select), 6
qeTS (Feature Model Select), 6
qeUMAP (qe-Series Predictive Functions), 12
qeXGBoost (qe-Series Predictive Functions), 12
quizDocs, 20
quizzes (quizDocs), 20

R Factor Utilities, 21

svcensus (prgeng), 11

ThyroidDisease, 22

Utilities, 22

Variable Importance Measures, 24

weatherTS, 25