Package 'ldt'

March 1, 2023

Title Let Data Talk

Version 0.2.0.0

Description Methods and tools for creating a model set and estimating and evaluating the explanation or prediction power of its members.
'SUR' modelling (for parameter estimation), 'logit'/'probit' modelling (for binary classification), and 'VARMA' modelling (for time-series forecasting) are implemented.
Evaluations are both in-sample and out-of-sample.
It can be used for stepwise regression analysis <https://en.wikipedia.org/wiki/Stepwise_regression>,
automatic model selection and model averaging (Claeskens and Hjort (2008, ISBN:1139471805, 9781139471800)),
calculating benchmarks, and doing sensitivity analysis (Leamer (1983) <https://www.jstor.org/stable/1803924> proposal).

License GPL (>= 3)

URL https://github.com/rmojab63/LDT

VignetteBuilder knitr

Encoding UTF-8

SystemRequirements C++17

RoxygenNote 7.2.3

Depends R (>= 3.5.0)

Imports Rcpp, readxl, jsonlite

Suggests knitr, testthat, rmarkdown, kableExtra, MASS

LinkingTo BH, Rcpp

Config/testthat/edition 3

LazyData true

NeedsCompilation yes

Author Ramin Mojab [aut, cre],
Michael Hutt [cph] (MIT license. Original code for Nelder-Mead algorithm.),
Stephen Becker [cph] (BSD 3-clause license. Original code for L-BFGS-B algorithm. The L-BFGS-B algorithm was written in the 1990s (mainly

1994, some revisions 1996) by Ciyou Zhu (in collaboration with R.H. Byrd, P. Lu-Chen and J. Nocedal). L-BFGS-B Version 3.0 is an algorithmic update from 2011, with coding changes by J. L. Morales), Math.NET [cph] (MIT license. Code from the 'Math.NET Numerics' library is used in calculating running statistics.)

Maintainer Ramin Mojab <rmojab63@gmail.com>

Repository CRAN

Date/Publication 2023-03-01 08:42:33 UTC

R topics documented:

BindVariables *Binds a List of Variables*

Description

Binds a List of Variables

Usage

```
BindVariables(varList)
```

Arguments

varList	A list of variables ((i.e., ldtv objects)) with similar frequency class

Value

A matrix with variables in the columns and frequencies as the row names.

Examples

```
v1 = ldt::Variable(c(1,2,3,2,3,4,5),"V1",F_Monthly(2022,12), list())
v2 = ldt::Variable(c(10,20,30,20,30,40,50),"V2",F_Monthly(2022,8), list())
vs = ldt::BindVariables(list(v1,v2))
```

ClusterH *Hierarchical Clustering*

Description

Hierarchical Clustering

Usage

```
ClusterH(distances, numVariables, linkage = "single")
```

Arguments

distances	(numeric vector) Determines the distances. This must be the lower triangle of a (symmetric) distance matrix (without the diagonal).
numVariables	(int) Determines the number of variables. This should hold: '2 * length(distances) = numVariables(numVariables - 1)'.
linkage	(string) Determines how Distances are calculated in a left-right node merge. It can be single, complete, uAverage, wAverage, ward.

Value

A list:

merge	(integer matrix)
height	(numeric vector)
order	(integer vector)

`ClusterHGroup` *Groups Variables with Hierarchical Clustering*

Description

Groups Variables with Hierarchical Clustering

Usage

```
ClusterHGroup(
  data,
  nGroups = 2L,
  threshold = 0,
  distance = "correlation",
  linkage = "single",
  correlation = "pearson"
)
```

Arguments

data	(numeric matrix) Data with variables in the columns.
nGroups	(int) Number of groups
threshold	(double) A threshold for omitting variables. If distance between two variables in a group is less than this value, the second one will be omitted. Note that a change in the order of the columns might change the results.
distance	(string) Determines how distances are calculated. It can be `correlation`, `absCorrelation`, `euclidean`, `manhattan`, `maximum`.
linkage	(string) Determines how Distances are calculated in a left-right node merge. It can be `single`, `complete`, `uAverage`, `wAverage`, `ward`.
correlation	(string) If `distance` is correlation, it determines the type of the correlation. It can be `pearson`, `spearman`.

Details

The results might be different from R's 'cutree' function. I don't know how 'cutree' works, but here I iterate over the nodes and whenever a split occurs, I add a group until the required number of groups is reached.

Value

A list:

groups	(List of integer vectors) indexes of variables in each group.
removed	(integer vector) indexes of removed variables.

CoefTable *Extract Coefficients from a list of* ldtestim *object*

Description

Extract Coefficients from a list of ldtestim object

Usage

```
CoefTable(
  list,
  depInd = 1,
 regInfo = list(c("", " "), c("num_obs", "No. Obs."), c("num_eq", "No. Eq."), c("num_x",
    "No. Exo."), c("sigma2", "S.E. Reg."), c("aic", "AIC"), c("sic", "SIC")),
  hnameFun = function(x) x,
  vnamesFun = function(x) x,
  vnamesFun_sub = list(c("%", "\\\\%"), c("_", "\\\\_")),
  vnamesFun_max = 20,
  tableFun = "coef_star",
  formatNumFun = function(colIndex, x) {
     x
 },
  numCoefs = NA,
  formatLatex = TRUE
)
```

Arguments

list	a named list of ldtestim objects.
depInd	index of the dependent variable.
regInfo	A list of pairs of keys and names to determine the information at the bottom of the table. Use "" (empty) for empty rows. num_eq and num_endo (and num_x and num_exo) will be different with PCA analysis enabled.
hnameFun	A function to change the name of the headers.
vnamesFun	A function to change the name of the variables or the codes in regInfo.
vnamesFun_sub	A list for replacing special characters vectors in vnamesFun.
vnamesFun_max	Maximum length for names in vnamesFun.

tableFun	A function (i.e., function(coef,std,pvalue,minInColm,maxInCol)) one of the following for default sign or coefficients table: "sign", "sign_star", "coef", "coef_star", "coef_star_std"
formatNumFun	A function to format the numbers if tableFun uses default values.
numCoefs	if NA, it inserts all coefficients. If a positive number, it inserts that number of coefficients.
formatLatex	If true, default options are for 'latex', otherwise, 'html'.

Details

#' Possible codes (first element) for regInfo:

- "" : empty line
- num_obs : No. Obs.; number of observations.
- num_endo : No. Eq. (orig.); original number of equations or endogenous variables before being changed by PCA analysis.
- pca_y_exact : PCA Count (y);
- pca_y_cutoff : PCA Cutoff (y)
- pca_y_max : PCA Max (y)
- num_eq : No. Eq.; number of equations after PCA analysis.
- num_exo : No. Exo. (orig.)
- pca_x_exact : PCA Count (x)
- pca_x_cutoff : PCA Cutoff (x)
- pca_x_max : PCA Max (x)
- num_x : No. Exo.
- num_x_all : No. Exo. (all); number of explanatory variables in all equations.
- num_rest : No. Rest.; number of restrictions in the equation
- sigma2 : S.E. Reg.
- ... others can be a measure name (i.e., elements of 'measures' item in the results)

Value

the generated table.

combineSearch *Combine More Than One* ldtsearch *Objects*

Description

Combine More Than One ldtsearch Objects

Usage

```
combineSearch(list, type1Name = "coefs")
```

Arguments

list	a list with ldtsearch objects
type1Name	the name of 'type1' in the object

Value

the combined ldtsearch object

CreateProject *Creates JSON Data for an* LDTSurvey *Project*

Description

Creates JSON Data for an LDTSurvey Project

Usage

```
CreateProject(
  data,
  description = list(c("Title", "Short Description", "Long Description", "en")),
  relatedIds = list(),
  survey_IsEnabled = TRUE,
  survey_RulesChange = TRUE,
  survey_MaxHorizon = 2,
  survey_MinRequired = 1,
  survey_showAI = FALSE,
  survey_showUser = FALSE,
  survey_RestrictTo = list(),
  survey_RestrictType = c("none", "view", "submit"),
  survey_EndConditionOn = c("none", "dayOfYear", "dayOfHalfYear", "dayOfQuarter",
    "dayOfMonth", "dayOfWeek", "hourOfDay"),
  survey_EndConditionValue = 0,
  forecast_IsEnabled = TRUE,
```

```
  forecast_External = list(),
  forecast_ExternalDesc = ""
)
```

Arguments

data	A list of Variables with consistent frequency. Use Variable function. Target is the one with 'role:target; field or if missing, the first variable.
description	A list of string arrays that provides basic information. Each array provides 4 elements: 1. title of the project, 2. a short description in plain text, 3. a longer description in mark-down format, and, 4. culture-name of the information. The first array is the default and culture-name must be unique.
relatedIds	ID of the related projects (e.g., this can be a project with the same data but in another frequency).
survey_IsEnabled	If FALSE, users cannot submit prediction.
survey_RulesChange	If TRUE, owner can change the survey rules in the future edits.
survey_MaxHorizon	Prediction horizons. E.g., 2 means a users can submit her prediction for the next 2 periods. It can be 1 to 5.
survey_MinRequired	Required minimum number of data points to be predicted by a user.
survey_showAI	If TRUE, user must submit her prediction first, before being able to see any automatic algorithm-based forecast
survey_showUser	If TRUE, user must submit her prediction first, before being able to see any other user-based prediction.
survey_RestrictTo	A list of e-mails for restricting access (see survey_RestrictType). Leave it empty for a public page. Otherwise, don't forget to add your email or you cannot submit prediction.
survey_RestrictType	Type of the restriction (see survey_RestrictTo). view means only the permitted users can view the page. submit means everyone can view, but the permitted users can submit prediction. none means no restriction (use it for communication purposes).
survey_EndConditionOn	Determines the type of the condition to end a survey automatically (see survey_EndCondition).
survey_EndConditionValue	Determines a condition to end a survey automatically. E.g., if survey_EndConditionOn is hourOfDay and this value is 20, the session will end (and users cannot submit predictions) on and after 20:00 (based on Gregorian calendar and UTC).
forecast_IsEnabled	If TRUE, an automatic algorithm-based forecast is reported (see also survey_showAI).

`forecast_External`
: An array for providing an external forecast up to `survey_MaxHorizon`. A forecast should be 'up' or 'down' for a direction forecast, a number for a point forecast, and 'dist:distribution-name(comma-separated parameters)' for a distribution forecast (e.g., 'normal(0,1)')

`forecast_ExternalDesc`
: A short description on what is provided in `forecast_External` (e.g., the name of the numerical method)

Value

The JSON content

`Data_BerkaLoan` *Use 'Berka' Data and create* `Loan-Series` *Table*

Description

Use 'Berka' Data and create `Loan-Series` Table

Usage

```
Data_BerkaLoan(
  dirPath,
  positive = c("B", "D"),
  negative = c("A", "C"),
  rateFun = function(amount, duration, paymentPerMonth) {
     ((paymentPerMonth *
    duration)/amount - 1) * 100
 }
)
```

Arguments

`dirPath`	path to the downloaded data directory.
`positive`	determines the positive class. There are four types of loans: A' stands for contract finished, no problems, 'B' stands for contract finished, loan not payed, 'C' stands for running contract, OK so far, 'D' stands for running contract, client in debt
`negative`	similar to `positive`
`rateFun`	a function to calculate interest rate in loans

Value

data.frame with the following columns:

- loan_id: record identifier
- status: original status of the data (A, B, C, or D)
- label: status of paying off the loan transformed to numeric (0,1) by using `positive` and `negative` arguments. value=1 means default.
- amount: amount of money
- payments: monthly payments
- rate: rates calculated by `rateFun` function
- duration_# (#=12,24,36,48,60): dummy variables for the duration of the loan
- account_frequency_?: dummy variables for the frequency of issuance of statements. ?="POPLATEK MESICNE" stands for monthly issuance, ?="POPLATEK TYDNE" stands for weekly issuance, ?="POPLATEK PO OBRATU" stands for issuance after transaction
- order_num: number of the payment orders issued for the account of the loan
- order_sum_amount: sum of amounts of the payment orders issued for the account of the loan
- order_related_account_num: unique number of 'account of the recipient' in the payment orders issued for the account of the loan
- order_related_bank_num: unique number of 'bank of the recipient' in the payment orders issued for the account of the loan
- order_has_?: dummy variables fo 'characterization of the payment' in the payment orders issued for the account of the loan
- trans_?num: number of transactions dealt with the account of the loan (in different groups)
- trans_?amount_mean: mean of 'amount of money' in the transactions dealt with the account of the loan (in different groups)
- trans_?amount_div_balance: mean of 'amount of money'/'balance after transaction' in the transactions dealt with the account of the loan (in different groups)
- trans_related_account_num: unique number of 'account of the partner' in the transactions dealt with the account of the loan
- trans_related_account_num: unique number of 'bank of the partner' in the transactions dealt with the account of the loan
- dist_inhabitants_num: no. of inhabitants in the location of the branch of the account of the loan
- dist_muni_#1#2: no. of municipalities with inhabitants #1-#2 in the location of the branch of the account of the loan
- dist_cities_num: no. of cities in the location of the branch of the account of the loan
- dist_ratio_urban_inhabitants: ratio of urban inhabitants in the location of the branch of the account of the loan
- dist_avg_salary: average salary in the location of the branch of the account of the loan
- dist_unemployment95: unemployment rate '95 in the location of the branch of the account of the loan

- dist_unemployment96: unemployment rate '96 in the location of the branch of the account of the loan
- dist_entrepreneurs_num_per1000: no. of entrepreneurs per 1000 inhabitants in the location of the branch of the account of the loan
- dist_crimes95_num: no. of committed crimes '95 in the location of the branch of the account of the loan
- dist_crimes96_num: no. of committed crimes '96 in the location of the branch of the account of the loan

---

`Data_Pcp` *Use 'PCP' Data (i.e., 'IMF's Primary Commodity Prices') and create* `Date-Series` *Table*

Description

Use 'PCP' Data (i.e., 'IMF's Primary Commodity Prices') and create `Date-Series` Table

Usage

```
Data_Pcp(dirPath, makeReal = FALSE)
```

Arguments

`dirPath`	path to the downloaded data data. It must also contain a file with the US CPI.
`makeReal`	uses the first column (which must be US-CPI) and converts nominal variables to real

Value

a list with data, descriptions, etc.

`Data_VestaFraud` *Use 'Vesta' Data (i.e., 'IEEE-CIS Fraud Detection') and create* `Fraud-Series` *Table*

Description

Use 'Vesta' Data (i.e., 'IEEE-CIS Fraud Detection') and create `Fraud-Series` Table

Usage

```
Data_VestaFraud(
  dirPath,
  training = TRUE,
  t_dumCols = NULL,
  i_dumCols = NULL,
  cat_min_unique_skip = 6
)
```

Arguments

dirPath	path to the downloaded data directory.
training	If FALSE, it loads test data
t_dumCols	a list with name and values of (categorical) columns in 'transaction' file to be converted to dummy variables. If training is FALSE and this is NULL, a warning is raised.
i_dumCols	similar to t_dumCols but for 'identity' file.
cat_min_unique_skip	If t_dumCols or i_dumCols is NULL, for a categorical variable, if number of unique values is equal or larger than this value, it is omitted.

Value

a list:

- data: a data.frame with the data
- t_dumCols: a list with name and values in 'transaction' data, used for creating the dummy variable
- i_dumCols: a list with name and values in 'identity' data, used for creating the dummy variable

---

Data_Wdi *Aggregate WDI Data and create* Country-Series *Table*

Description

Aggregate WDI Data and create Country-Series Table

Usage

```
Data_Wdi(
  dirPath,
  minYear = 1960,
  maxYear = 2020,
  aggFunction = function(data, code, name, unit, definition, aggMethod) {
     isPerc <-
    unit == "%" || grepl(".ZG", code)
     if (isPerc) {
         NA
     }
     else {

            LongrunGrowth(data, 30, 5, FALSE, TRUE, isPerc)
     }
 },
```

```
  keepFunction = function(X) {
     var(X, na.rm = TRUE) > 1e-12 && sum((is.na(X)) ==
    FALSE) >= 50
 },
  ...
)
```

Arguments

dirPath	(character) path to the data directory in CSV format. It must have 'WDICountry-Series.csv', 'WDIData.csv', 'WDICountry.csv', 'WDISeries.csv'. Download it from the WDI site.
minYear	(integer) a year where aggregation starts
maxYear	(integer) a year where aggregation ends.
aggFunction	(function) aggregation function, such as: function(data,code,name,unit,defintion,aggMethod)mean(data, na.rm = TRUE); where 'data' is the data-points from minYear to maxYear, 'unit' is the unit of measurement, 'definition' is the long definition of the series, 'aggMethod' is the method of aggregation.
keepFunction	(function) a function to determine how to keep or omit a series (i.e., column). default function skips growth rates, checks the variance and the number of non-NA data-points.
...	additional arguments

Value

data, countries information (rows in data), and series information (columns in data)

Data_WdiSearchFor *Search For Series in WDI Data*

Description

it searches in code, (name and description) of the series.

Usage

```
Data_WdiSearchFor(
  series,
  keywords,
  searchName = TRUE,
  searchDesc = FALSE,
  topickeywords = NULL,
  findOne = FALSE,
  ...
)
```

Arguments

series	The series member of an output from Data_Wdi function.
keywords	(character array) strings to search for.
searchName	if FALSE, it does not search in the name
searchDesc	if FALSE, it does not search in the description
topickeywords	If given, topic of a matched case must contain this string, too.
findOne	Raises error if TRUE and more than 1 series is found. default is FALSE.
...	additional arguments

Value

a list with series information or if findOne is TRUE a series information.

Examples

```
#data <- Data_Wdi() # this is time-consuming and requires WDI dataset files
#res <- Data_WdiSearchFor(data$series, c("GDP per capita"),
#                         TRUE, topickeywords = "national account")
```

DcEstim *Estimates an Discrete Choice Model*

Description

Estimates an Discrete Choice Model

Usage

```
DcEstim(
  y,
  x,
  w = NULL,
  distType = "logit",
  newX = NULL,
  pcaOptionsX = NULL,
  costMatrices = NULL,
  aucOptions = NULL,
  simFixSize = 200L,
  simTrainRatio = 0.5,
  simTrainFixSize = 0L,
  simSeed = 0L,
  weightedEval = FALSE,
  printMsg = FALSE
)
```

Arguments

y	(numeric matrix) Data with dependent variable in the column. Given the number of choices 'n', it must contain 0,1,...,n-1 and 'sum(y==i)>0' for i=0,...,n-1.
x	(numeric matrix) Exogenous data with variables in the columns.
w	(numeric vector) Weights of the observations in y. Null means equal weights.
distType	(string) Distribution assumption. It can be logit or probit.
newX	(numeric matrix) If not null, probabilities are projected for each row of this matrix.
pcaOptionsX	(list) A list of options in order to use principal components of the x, instead of the actual values. set null to disable. Use GetPcaOptions() for initialization.
costMatrices	(list of matrices) Each cost table determines how you score the calculated probabilities.
aucOptions	(nullable list) AUC calculation options. see [GetRocOptions()].
simFixSize	(int) Number of pseudo out-of-sample simulations. Use zero to disable the simulation. (see GetMeasureOptions()).
simTrainRatio	(double) Size of the training sample as a ratio of the number of the observations. It is effective only if simTrainFixSize is zero.
simTrainFixSize	(int) A fixed size for the training sample. If zero, simTrainRatio is used.
simSeed	(int) A seed for the pseudo out-of-sample simulation.
weightedEval	(bool) If true, weights will be used in evaluations.
printMsg	(bool) Set false to disable printing the details.

Value

A list:

DcSearch *Discrete Choice Search*

Description

Discrete Choice Search

Usage

```
DcSearch(
  y,
  x,
  w = NULL,
  xSizes = NULL,
  xPartitions = NULL,
  costMatrices = NULL,
```

```
  searchLogit = TRUE,
  searchProbit = FALSE,
  optimOptions = NULL,
  aucOptions = NULL,
  measureOptions = NULL,
  modelCheckItems = NULL,
  searchItems = NULL,
  searchOptions = NULL
)
```

Arguments

y	(numeric matrix) endogenous data with variable in the column.
x	(numeric matrix) exogenous data with variables in the columns.
w	(numeric vector) weights of the observations in y. null means equal weights.
xSizes	(nullable int vector) Number of exogenous variables in the regressions. E.g., c(1,2) means the model set contains all the regressions with 1 and 2 exogenous variables. If null, c(1) is used.
xPartitions	(nullable list of int vector) a partition over the indexes of the exogenous variables. No regression is estimated with two variables in the same group. If null, each variable is placed in its own group and the size of the model set is maximized.
costMatrices	(list of numeric matrix) each frequency cost matrix determines how to score the calculated probabilities. Given the number of choices 'n', a frequency cost matrix is a 'm x n+1' matrix. The first column determines the thresholds. Cells in the j-th column determines the costs corresponding to the (j-1)-th choice in y. It can be null if it is not selected in measureOptions.
searchLogit	(bool) if TRUE, logit regressions are added to the model set.
searchProbit	(bool) if TRUE, probit regressions are added to the model set.
optimOptions	(nullable list) Newton optimization options. see [GetNewtonOptions()].
aucOptions	(nullable list) AUC calculation options. see [GetRocOptions()].
measureOptions	(nullable list) see [GetMeasureOptions()].
modelCheckItems	(nullable list) see [GetModelCheckItems()].
searchItems	(nullable list) see [GetSearchItems()].
searchOptions	(nullable list) see [GetSearchOptions()].

Value

A list

DcSearch_s *Step-wise Discrete Choice Search*

Description

A helper class to deal with large model sets. It selects a subset of variables from smaller models and moves to the bigger ones.

Usage

```
DcSearch_s(
  x,
  xSizes = list(c(1, 2), c(3, 4), c(5), c(6:10)),
  counts = c(NA, 40, 30, 20),
  savePre = NULL,
  ...
)
```

Arguments

x	exogenous data
xSizes	a list of model dimension to be estimated in each step.
counts	a list of suggested number of variables to be used in each step. NA means all variables. Variables are selected based on best estimations (select an appropriate value for searchItems$bestK). All variables in the best models (all measures and targets) are selected until corresponding suggested number is reached.
savePre	if not NULL, it saves and tries to load the progress of search step in a file (name=paste0(savePre,i) where i is the index of the step).
...	other arguments to pass to DcSearch() function such as endogenous data. Note that xSizes is treated differently.

Value

A combined LdtSearch object

F_CrossSection *Creates a Cross-Section Frequency*

Description

This frequency is generally for indexed (or, non-time-series) data. It is an integer that represents the position of the observation.

Usage

```
F_CrossSection(position)
```

Arguments

position	Position of the observation

Details

- **Value String:** "#" (number is position)
- **Class String:** "cs"

Value

An object of class 'ldtf'

F_Daily *Creates a* Daily *Frequency*

Description

Frequency for a series that happens every day

Usage

```
F_Daily(year, month, day)
```

Arguments

year	Year of the observation
month	Month of the observation
day	Day of the observation.

Details

- **Value String:** "YYYYMMDD" (similar to Weekly)
- **Class String:** "d"

Value

An object of class 'ldtf'

F_DailyInWeek *Creates an* Daily-In-Week *Frequency*

Description

Frequency for a series that happens every in the days of a week

Usage

```
F_DailyInWeek(year, month, day, weekStart, weekEnd, forward)
```

Arguments

year	Year of the observation
month	Month of the observation
day	First day of the observation
weekStart	First day of the week. It can be sun, mon, tue, wed, thu, fri, and sat
weekEnd	Last day of the week. See weekStart. Together, they define the week
forward	If current date in not in the week, if true, it moves forward to the first day of the week. Otherwise, it moves backward to the last day of the week.

Details

- **Value String:** "YYYYMMDD" (similar to Weekly)
- **Class String:** "i:...-..." (the first ... is weekStart and the second ... is weekEnd; e.g., i:mon-fri means a week that is from Monday to Friday)

Value

An object of class 'ldtf'

F_Hourly *Creates an 'Hourly' Frequency*

Description

Frequency for a series that happens every hour

Usage

```
F_Hourly(day, hour)
```

Arguments

day	A 'Day-based' frequency such as Daily or Daily-In-Week
hour	Index of hour in the day (1 to 24)

Details

- **Value String:** "YYYYMMDD:#" (the number is hour)
- **Class String:** ho|... (the ... is the 'Class String' of day)

Value

An object of class 'ldtf'

F_ListDate *Creates an* List-Date *Frequency*

Description

Frequency for a series that is labeled by dates

Usage

```
F_ListDate(items, value)
```

Arguments

items	Items of the list in string format: YYYYMMDD
value	Current value in string format: YYYYMMDD

Details

- **Value String:** "YYYYMMDD" (i.e., item)
- **Class String:** Ld or Ld:... (in which ... is the semi-colon separated items)

Value

An object of class 'ldtf'

F_ListString *Creates an* List-String *Frequency*

Description

Frequency for a series that is labeled by string

Usage

```
F_ListString(items, value)
```

Arguments

items	Items of the list
value	Current item

Details

- **Value String:** "..." (in which ... is the value)
- **Class String:** Ls or Ls:... (in which ... is the semi-colon separated items)

Value

An object of class 'ldtf'

F_Minute_ly *Creates an 'Minute-ly' Frequency*

Description

Frequency for a series that happens every minute

Usage

```
F_Minute_ly(day, minute)
```

Arguments

day	A 'Day-based' frequency such as daily or daily-in-week
minute	Index of Minute in the day (1 to 1440)

Details

- **Value String:** "YYYYMMDD:#" (the number is minute)
- **Class String:** mi|... (the ... is the 'Class String' of day)

Value

An object of class 'ldtf'

`F_Monthly` *Creates a* `Monthly` *Frequency*

Description

Frequency for a series that happens every month

Usage

```
F_Monthly(year, month)
```

Arguments

`year`	Year of the observation
`month`	Month of the observation

Details

- **Value String:** `"#m#"` (first # is the `year`, second # is `month` (1 to 12); e.g., 2010m8 or 2010m12. Note that 2000m0 or 2000m13 are invalid.
- **Class String:** `"m"`

Value

An object of class 'ldtf'

`F_MultiDaily` *Creates an* `Multi-Daily` *Frequency*

Description

Frequency for a series that happens every `k` days

Usage

```
F_MultiDaily(year, month, day, k)
```

Arguments

`year`	Year of the observation
`month`	Month of the observation
`day`	First day of the observation
`k`	Number of the days

Details

- **Value String:** "YYYYMMDD" (similar to Weekly)
- **Class String:** "d#" (the number is k)

Value

An object of class 'ldtf'

F_MultiWeekly *Creates a* Multi-Weekly *Frequency*

Description

Frequency for a series that happens every 'k' weeks

Usage

```
F_MultiWeekly(year, month, day, k)
```

Arguments

year	Year of the observation
month	Month of the observation
day	First day of the observation. It points to the first day of the week
k	Number of weeks

Details

- **Value String:** "YYYYMMDD" (similar to Weekly)
- **Class String:** "w#" (the number is k; e.g., w3 means every 3 weeks)

Value

An object of class 'ldtf'

F_MultiYearly *Creates a* Multi-Yearly *Frequency*

Description

Frequency for a series that happens every z years

Usage

```
F_MultiYearly(year, z)
```

Arguments

year	Year of the observation
z	Number of years

Details

- **Value String:** "#" (similar to Yearly)
- **Class String:** "z#" (integer represents the z; e.g., z3)

Value

An object of class 'ldtf'

F_Quarterly *Creates a* Quarterly *Frequency*

Description

Frequency for a series that happens every quarter

Usage

```
F_Quarterly(year, quarter)
```

Arguments

year	Year of the observation
quarter	Quarter of the observation (1 to 4)

Details

- **Value String:** "#q#" (first # is year, second # is quarter; e.g., 2010q3 or 2010q4. Note that 2000q0 or 2000q5 are invalid.
- **Class String:** "q"

Value

An object of class 'ldtf'

F_Second_ly *Creates an 'Second-ly' Frequency*

Description

Frequency for a series that happens every second

Usage

```
F_Second_ly(day, second)
```

Arguments

day	A 'Day-based' frequency such as daily or daily-in-week
second	Index of second in the day (1 to 86400)

Details

- **Value String:** "YYYYMMDD:#" (the number is second)
- **Class String:** se|... (the ... is the 'Class String' of day)

Value

An object of class 'ldtf'

F_Weekly *Creates a* Weekly *Frequency*

Description

Frequency for a series that happens every week

Usage

```
F_Weekly(year, month, day)
```

Arguments

year	Year of the observation
month	Month of the observation
day	Day of the observation. It points to the first day of the week

Details

- **Value String:** "YYYYMMDD" (YYYY is the year, MM is month and DD is day)
- **Class String:** "w"

Value

An object of class 'ldtf'

F_XTimesADay *Creates an 'X-Times-A-Day' Frequency*

Description

Frequency for a series that happens x times in a day

Usage

```
F_XTimesADay(day, x, position)
```

Arguments

day	A 'Day-based' frequency such as daily or daily-in-week
x	Number of observations in a day
position	Current position

Details

- **Value String:** "#" (the number is hour)
- **Class String:** "da#|..." (the number is x and ... is the 'Class String' of day))

Value

An object of class 'ldtf'

F_XTimesAYear *Creates an* X-Times-A-Year *Frequency*

Description

Frequency for a series that happens x times every year

Usage

```
F_XTimesAYear(year, x, position)
```

Arguments

year	Year of the observation
x	Number of observation in each year
position	Position of the current observation

Details

- **Value String:** "#:#" (first # is year and second # is position; e.g., 2010:8/12 or 2010:10/10. Note that 2000:0/2 or 2000:13/12 are invalid.
- **Class String:** "y#" (the number is x)

Value

An object of class 'ldtf'

F_XTimesZYear *Creates an* X-Times-Z-Years *Frequency*

Description

Frequency for a series that happens x times each z years

Usage

```
F_XTimesZYear(year, x, z, position)
```

Arguments

year	Year of the observation
x	Number of partitons in each z years
z	Number of years
position	Position of the current observation

Details

- **Value String:** `"#:#"` (Similar to `X-Times-A-Year`)
- **Class String:** `"x#z#"` (first # is `x`, second # is `z`; e.g., x23z4 means 23 times every 4 years)

Value

An object of class 'ldtf'

`F_Yearly` *Creates a* `Yearly` *Frequency*

Description

Frequency for a series that happens every year

Usage

```
F_Yearly(year)
```

Arguments

`year`	Year of the observation

Details

- **Value String:** `"#"` (number is `year`)
- **Class String:** `"y"`

Value

An object of class 'ldtf'

`GetCombination4Moments` *Combines Two Distributions Defined by their First 4 Moments*

Description

Combines Two Distributions Defined by their First 4 Moments

Usage

```
GetCombination4Moments(mix1, mix2)
```

Arguments

mix1	(list) First distribution which is defined by a list with mean, variance, skewness, kurtosis, sumWeights, count
mix2	(list) Second distribution (similar to mix1).

Value

(list) A list similar to mix1

Examples

```
#see its \code{test_that} function
```

GetDistance *Gets Distances Between Variables*

Description

Gets Distances Between Variables

Usage

```
GetDistance(
  data,
  distance = "correlation",
  correlation = "pearson",
  checkNan = TRUE
)
```

Arguments

data	(numeric matrix) Data with variables in the columns.
distance	(string) Determines how distances are calculated. It can be correlation, absCorrelation, euclidean, manhattan, maximum.
correlation	(string) If distance is correlation, it determines the type of the correlation. It can be pearson, spearman.
checkNan	(bool) If false, NANs are not omitted.

Value

A symmetric matrix (lower triangle as a vector).

getDummy *Title*

Description

Title

Usage

```
getDummy(table, colName, pre = "", min_unique_skip = 6, uniques = NULL)
```

Arguments

table	data
colName	categorical column
pre	a string to put before the name of the variables
min_unique_skip	if number of unique values is equal or larger, it returns NULL
uniques	if not NULL, it skips finding unique values and uses the given list. Also, if colName column is missing, it creates zero variables for the given items

Value

data (list of dummy variables) and uniques (unique values)

GetEstim *Get Estimation from Search Result*

Description

Get Estimation from Search Result

Usage

```
GetEstim(searchRes, endoIndices, exoIndices, y, x, printMsg, ...)
```

Arguments

searchRes	an object of class ldtsearch
endoIndices	endogenous indices
exoIndices	exogenous indices
y	dependent variables data
x	exogenous variables data
printMsg	argument to be passed to the estimation methods
...	additional arguments

Value

estimation result

`GetGldFromMoments` *Gets the GLD-FKML Parameters from the moments*

Description

Calculates the parameters of the generalized lambda distribution (FKML), given the first four moments of the distribution.

Usage

```
GetGldFromMoments(
  mean = 0,
  variance = 1,
  skewness = 0,
  excessKurtosis = 0,
  type = 0L,
  start = NULL,
  nelderMeadOptions = NULL,
  printMsg = FALSE
)
```

Arguments

`mean`	(double) mean of the distribution.
`variance`	(double) variance of the distribution.
`skewness`	(double) skewness of the distribution.
`excessKurtosis`	(double) excess kurtosis of the distribution.
`type`	(int) The type of the distribution.
`start`	(numeric vector, length=2) starting value for L3 and L4. Use null for c(0,0).
`nelderMeadOptions`	(list) The optimization parameters. Use null for default.
`printMsg`	(bool) If `TRUE`, details are printed.

Details

The type of the distribution is determined by one or two restrictions:

- **type 0:** general
- **type 1:** symmetric 'type 0'
- **type 2:** uni-modal continuous tail: L3<1 & L4<1
- **type 3:** symmetric 'type 2' L3==L4

- **type 4:** uni-modal continuous tail finite slope L3<=0.5 & L4<=5
- **type 5:** symmetric 'type 4' L3==L4
- **type 6:** uni-modal truncated density curves: L3>=2 & L4>=2 (includes uniform distribution)
- **type 7:** symmetric 'type 6' L3==L4
- **type 8:** S shaped L3>2 & 1<L4<2 or 1<L3<2 & L4>2
- **type 9:** U shaped 1<L3<=2 and 1<L4<=2
- **type 10:** symmetric 'type 9' L4==L4
- **type 11:** monotone L3>1 & L4<=1

Value

a vector with the parameters of the GLD distribution.

Examples

```
res = GetGldFromMoments(0,1,0,0,0,c(0,0))
```

`GetLmbfgsOptions` *Options for LMBFGS Optimization*

Description

Options for LMBFGS Optimization

Usage

```
GetLmbfgsOptions(
  maxIterations = 100L,
  factor = 1e+07,
  projectedGradientTol = 0,
  maxCorrections = 5L
)
```

Arguments

`maxIterations`	(int) A positive integer for maximum number of iterations.
`factor`	(double) A condition for stopping the iterations. The iteration will stop when (f^k - f^k+1)/max|f^k|,|f^k+1|,1 < `factor`*epsmch where epsmch is the machine precision, which is automatically generated by the code. Use e.g., 1e12 for low accuracy, 1e7 (default) for moderate accuracy and 1e1 for extremely high accuracy. default is 1e7
`projectedGradientTol`	(double) The iteration will stop when `max{|proj g_i | i = 1, ..., n} < projectedGradientTol` where `pg_i` is the ith component of the projected gradient. default is zero.
`maxCorrections`	(int) Maximum number of variable metric corrections allowed in the limited memory Matrix. default is 5.

Value

A list with the given options.

GetMeasureFromWeight *Converts a Measure to Weight*

Description

Converts a Measure to Weight

Usage

```
GetMeasureFromWeight(value, measureName)
```

Arguments

value	(double) the measure
measureName	(string) measure name

Value

the measure

Examples

```
weight <- GetWeightFromMeasure(-3.4, "sic")
measure <- GetMeasureFromWeight(weight, "sic")
```

GetMeasureOptions *Options for 'Measuring Performance'*

Description

Options for 'Measuring Performance'

Usage

```
GetMeasureOptions(
  typesIn = NULL,
  typesOut = NULL,
  simFixSize = 10L,
  trainRatio = 0.75,
  trainFixSize = 0L,
  seed = 0L,
  horizons = NULL,
  weightedEval = FALSE
)
```

Arguments

typesIn	(nullable string vector) Evaluations when model is estimated using all available data. It can be aic, sic, frequencyCostIn, aucIn. Null means no measure.
typesOut	(nullable string vector) Evaluations in an pseudo out-of-sample simulation. It can be sign, direction, rmse, scaledRmse, mae, scaledMae, crps, frequencyCostOut, aucOut. Null means no measure.
simFixSize	(int) Number of pseudo out-of-sample simulations. Use zero to disable the simulation.
trainRatio	(double) Number of data-points, as a ratio of the available size, in the training sample in the pseudo out-of-sample simulation.
trainFixSize	(int) Number of data-points in the training sample in the pseudo out-of-sample simulation. If zero, trainRatio will be used.
seed	(int) A seed for random number generator. Use zero for a random value.
horizons	(nullable integer vector) prediction horizons to be used in pseudo out-of-sample simulations, if model supports time-series prediction. If null, c(1) is used.
weightedEval	(bool) If true, weights are used in evaluationg discrete-choice models

Value

A list with the given options.

GetModelCheckItems *Options for 'Model Check Items'*

Description

Options for 'Model Check Items'

Usage

```
GetModelCheckItems(
  estimation = TRUE,
  maxConditionNumber = 1.7e+308,
  minObsCount = 0L,
  minDof = 0L,
  minOutSim = 0L,
  minR2 = -1.7e+308,
  maxAic = 1.7e+308,
  maxSic = 1.7e+308,
  prediction = FALSE,
  predictionBoundMultiplier = 4
)
```

Arguments

estimation	(bool) If true, model is estimated with all data. If false, you might get a 'best model' that cannot be estimated.
maxConditionNumber	(double) Maximum value for the condition number (if implemented in the search).
minObsCount	(int) Minimum value for the number of observations. Use 0 to disable.
minDof	(int) Minimum value for the degrees of freedom (equation-wise). Use 0 to disable.
minOutSim	(int) Minimum value for the number of valid out-of-sample simulations (if implemented in the search).
minR2	(double) Minimum value for R2 (if implemented in the search).
maxAic	(double) Maximum value for AIC (if implemented in the search).
maxSic	(double) Maximum value for SIC (if implemented in the search).
prediction	(bool) If true, model data is predicted given all data. If false, you might get a 'best model' that cannot be used in prediction.
predictionBoundMultiplier	(double) If positive, a bound is created by multiplying this value to the average growth rate. A model is ignored, if its prediction lies outside of this bound.

Value

A list with the given options.

GetNelderMeadOptions *Options for Nelder-Mead Optimization*

Description

Options for Nelder-Mead Optimization

Usage

```
GetNelderMeadOptions(
  maxIterations = 100L,
  epsilon = 1e-08,
  alpha = 1,
  beta = 0.5,
  gamma = 2,
  scale = 1
)
```

Arguments

maxIterations	(int) Maximum number of iterations.
epsilon	(double) A small value to test convergence.
alpha	(double) the reflection coefficient.
beta	(double) the contraction coefficient.
gamma	(double) the expansion coefficient.
scale	(double) A scale in initializing the simplex.

Value

A list with the given options.

GetNewtonOptions *Options for Newton Optimization*

Description

Options for Newton Optimization

Usage

```
GetNewtonOptions(
  maxIterations = 100L,
  functionTol = 1e-04,
  gradientTol = 0,
  useLineSearch = TRUE
)
```

Arguments

maxIterations	(int) Maximum number of iterations.
functionTol	(double) A small value to test convergence of the objective function.
gradientTol	(double) A small value to test convergence of the gradient.
useLineSearch	(bool) If true, it uses line search.

Value

A list with the given options.

GetPca *Principle Component Analysis*

Description

Principle Component Analysis

Usage

```
GetPca(x, center = TRUE, scale = TRUE, newX = NULL)
```

Arguments

x	(numeric matrix) data with variables in columns.
center	(bool) if TRUE, it demeans the variables.
scale	(bool) if TRUE, it scales the variables to unit variance.
newX	(numeric matrix) data to be used in projection. Its structure must be similar to the x.

Value

(list) results

removed0Var	(integer vector) Zero-based indices of removed columns with zero variances.
directions	(numeric matrix) Directions
stds	(integer vector) Standard deviation of the principle components
stds2Ratio	(integer vector) stds^2/sum(stds^2)
projections	(numeric matrix) Projections if newX is given.

GetPcaOptions *Options for PCA*

Description

Options for PCA

Usage

```
GetPcaOptions(ignoreFirst = 1L, exactCount = 0L, cutoffRate = 0.8, max = 1000L)
```

Arguments

ignoreFirst	(int) Excludes variables at the beginning of data matrices (such as intercept) from PCA.
exactCount	(int) Determines the number of components to be used. If zero, number of components are determined by the cutoffRate.
cutoffRate	(double between 0 and 1) Determines the cutoff rate for cumulative variance ratio in order to determine the number of PCA components. It is not used if exactCount is positive.
max	(int) Maximum number of components when cutoffRate is used.

Value

A list with the given options.

GetRoc *ROC curve for a binary case*

Description

It does not draw the ROC, but calculatates the required points. It also Calculates the AUC with different options

Usage

```
GetRoc(y, scores, weights = NULL, options = NULL, printMsg = FALSE)
```

Arguments

y	(numeric vector, Nx1) Actual values
scores	(numeric vector, Nx1) Calculated probabilities for the negative observations
weights	(numeric vector, Nx1) Weights of the observations. Use NULL for equal weights.
options	(list) More options. See GetRocOptions() function for details.
printMsg	(bool) Set true to report some details.

Value

A list with the following items:

N	(integer) Number of observations
AUC	(numeric) Value of AUC
Points	(numeric matrix) Points for ploting ROC

Examples

```
y <- c(1, 0, 1, 0, 1, 1, 0, 0, 1, 0)
scores <- c(0.1, 0.2, 0.3, 0.5, 0.5, 0.5, 0.7, 0.8, 0.9, 1)
res1 = GetRoc(y,scores, printMsg = FALSE)
costs <- c(1,2,1,4,1,5,1,1,0.5,1)
costMatrix = matrix(c(0.02,-1,-3,3),2,2)
opt <- GetRocOptions(costs = costs, costMatrix = costMatrix)
res2 = GetRoc(y,scores,NULL,options = opt, printMsg = FALSE)
#plot(res1$Points)
#lines(res2$Points)
```

`GetRocOptions` *Options for ROC and AUC*

Description

Options for ROC and AUC

Usage

```
GetRocOptions(
  lowerThreshold = 0,
  upperThreshold = 1,
  epsilon = 1e-12,
  pessimistic = FALSE,
  costs = NULL,
  costMatrix = NULL
)
```

Arguments

`lowerThreshold`	(double) Lower bound for calculating partial AUC.
`upperThreshold`	(double) Upper bound for calculating partial AUC.
`epsilon`	(double) A value to ignore small floating point differences in comparing scores.
`pessimistic`	(bool) If true, sequences of equally scored instances are treated differently and a pessimistic measure is calculated (see Fawcett (2006) An introduction to roc analysis, fig. 6).
`costs`	(numeric vector) cost of each observations. If null, cost of all observations will be 1.
`costMatrix`	(numeric matrix) a 2x2 cost matrix in which: (1,1) is cost of TN, (2,2) is cost of TP, (1,2) is cost of FP and (2,1) is cost of FN. First column is multiplied by the corresponding value in costs vector (see Fawcett (2006), ROC graphs with instance-varying costs).

Value

A list with the given options.

`GetSearchItems` *Options for 'Search Items'*

Description

Creates a list with predefined items which determines the information to be saved and retrieved.

Usage

```
GetSearchItems(
  model = TRUE,
  type1 = FALSE,
  type2 = FALSE,
  bestK = 1L,
  all = FALSE,
  inclusion = FALSE,
  cdfs = NULL,
  extremeMultiplier = 0,
  mixture4 = FALSE
)
```

Arguments

`model`	(bool) If true, information about the models is saved.
`type1`	(bool) If true and implemented, extra information is saved. This can be the coefficients in the SUR search or predictions in VARMA search.
`type2`	(bool) If true and implemented, extra information is saved. This is similar to `type1`. **It is reserved for future updates.**
`bestK`	(int) Number of best items to be saved in `model`, `type1`, or `type2` information.
`all`	(bool) If true, all available information is saved.
`inclusion`	(bool) If true, inclusion weights are saved in `model`.
`cdfs`	(nullable numeric vector) Weighted average of the CDFs at each given point is calculated (for `type1` and `type2`).
`extremeMultiplier`	(double) Determined the multiplier in the extreme bound analysis (for `type1` and `type2`).
`mixture4`	(bool) If true, the first 4 moments of the average distributions are calculated in `type1` and `type2`.

Value

A list with the given options.

GetSearchOptions *Options for 'Search Options'*

Description

Creates a list with predefined Search options.

Usage

```
GetSearchOptions(parallel = FALSE, reportInterval = 2L, printMsg = FALSE)
```

Arguments

parallel	(bool) If true, it uses a parallel search. It generally changes the speed and memory usage.
reportInterval	(int) Time interval (in seconds) for reporting the progress (if the change is significant). Set zero to disable.
printMsg	(bool) Set false to disable printing the details.

Value

A list with the given options.

GetWeightFromMeasure *Converts a Measure to Weight*

Description

Converts a Measure to Weight

Usage

```
GetWeightFromMeasure(value, measureName)
```

Arguments

value	(double) the measure
measureName	(string) measure name

Value

the weight

Examples

```
weight <- GetWeightFromMeasure(-3.4, "sic")
```

GldDensityQuantile *Gets GLD Density Quantile*

Description

Gets GLD Density Quantile

Usage

```
GldDensityQuantile(data, L1, L2, L3, L4)
```

Arguments

data	(numeric vector) data
L1	(double) First parameter
L2	(double) Second parameter
L3	(double) Third parameter
L4	(double) Fourth parameter

Value

(numeric vector) result

GldQuantile *Gets GLD Quantile*

Description

Gets GLD Quantile

Usage

```
GldQuantile(data, L1, L2, L3, L4)
```

Arguments

data	(numeric vector) data
L1	(double) First parameter
L2	(double) Second parameter
L3	(double) Third parameter
L4	(double) Fourth parameter

Value

(numeric vector) result

`IsEmailValid` *Determines if an email address is valid (this is not exact. Just use it to avoid mistakes)*

Description

Determines if an email address is valid (this is not exact. Just use it to avoid mistakes)

Usage

```
IsEmailValid(x)
```

Arguments

`x`	email

Value

`TRUE` if email is valid, `FALSE` otherwise.

`IsGuidValid` *Determines if a GUID is valid*

Description

Determines if a GUID is valid

Usage

```
IsGuidValid(x)
```

Arguments

`x`	GUID

Value

`TRUE` if GUID is valid, `FALSE` otherwise.

LongrunGrowth *Calculate Long-run Growth*

Description

Calculate Long-run Growth

Usage

```
LongrunGrowth(
  data,
  trimStart = 0,
  trimEnd = 0,
  cont = FALSE,
  skipZero = TRUE,
  isPercentage = FALSE,
  ...
)
```

Arguments

data	(integer vector) data
trimStart	(integer) if the number of leading NAs is larger than this number, it returns NA. Otherwise, it finds the first number and continue the calculations.
trimEnd	(integer) if the number of trailing NAs is larger than this number, it returns NA. Otherwise, it finds the first number and continue the calculations.
cont	(logical) if TRUE it will use the continuous formula.
skipZero	(logical) if TRUE leading and trailing zeros are skipped.
isPercentage	(logical) if the unit of measurement in data is percentage (e.g., growth rate) use TRUE. Long-run growth rate is calculated by arithmetic mean for continuous case, and geometric mean otherwise. If missing data exists, it returns NA.
...	additional arguments

Value

the growth rate (percentage)

Examples

```
y <- c(NA, 0, c(60, 70, 80, 90), 0, NA, NA)
g <- LongrunGrowth(y, 2, 3, skipZero = TRUE, isPercentage = TRUE, cont = TRUE)
```

Parse_F *Converts back a String to* ldtf *Object*

Description

The format is explained in F_? functions.

Usage

```
Parse_F(str, classStr)
```

Arguments

str	value of the frequency. It must be an ldtf object returned from F_? functions.
classStr	class of the frequency

Value

An object of class 'ldtf'

PlotCoefs *Plots Estimated Coefficients*

Description

Plots Estimated Coefficients

Usage

```
PlotCoefs(
  points = NULL,
  bounds = NULL,
  intervals = NULL,
  distributions = NULL,
  newPlot = TRUE,
  xlim = NULL,
  ylim = NULL,
  boundFun = function(b, type) {
     if (type == "xmin" || type == "ymin") {

    0.9 * b
     }
     else {
         1.1 * b
     }
```

```
 },
  legendsTitle = c("Point", "Bound", "Interval", "Density"),
  legendSize = 5,
  ...
)
```

Arguments

points	(list of list) each element is a point estimation to be drawn as a shape; defined by 1.value, 2.y (default=0), 3.shape (default="circle"), ...
bounds	(list of list) each element is a bound estimation (e.g. extreme bound analysis) to be drawn as a rectangle; defined by 1.xmin, 2.xmax, 3.ymin (default=-0.1), 4.ymax, (default=+0.1), 5.alpha, ...
intervals	(list of list) each element is an interval estimation (similar to bounds but with a value) to be drawn as an interval; defined by 1.value, 2.xmin, 3.xmax, ...
distributions	(list of list) each element is a distribution estimation (eg., a known distribution) to be drawn as its density function; defined by 1.type, and for type=normal, 2.mean, 3.var, 4.sdMultiplier, for type=GLD, 2.p1,..., 5.p4, 6.quantiles, for type==cdfs 2.xs, 3.cdfs, 4.smoothFun, ...
newPlot	(logical) if TRUE, a new plot is initialized.
xlim	(numeric vector) two limits for the x axis. If NULL, it is auto generated.
ylim	(numeric vector) two limits for the y axis. If NULL, it is auto generated.
boundFun	(function) a function to control the xlim and ylim in the plot. Its arguments are the computed bounds.
legendsTitle	(list) a list of titles for legends.
legendSize	(numeric) size of the legend (width or height) in lines of text (it is passed to oma).
...	additional properties for plot or legend: xlab, ylab

Value

if plot is FALSE, a ggplot to be printed.

Examples

```
points <- list()
points$one <- list(value = 1, label = "Point 1")
points$two <- list(value = 2, label = "Point 2", col = "red", pch = 22, cex = 4)
PlotCoefs(points = points)

bounds <- list()
bounds$one <- list(xmin = -1, xmax = 0.5, label = "Bound 1")
bounds$two <- list(
  xmin = 0, xmax = 1, ymin = 0.2, ymax = 0.3,
  label = "Bound 2", alpha = 0.2, col = rgb(0, 0, 1.0, alpha = 0.3)
)
PlotCoefs(points = points, bounds = bounds)
```

```
intervals <- list()
intervals$one <- list(value = 2, xmin = 0, xmax = 3, label = "Interval 1")
intervals$two <- list(
  value = 1.5, xmin = 1, xmax = 2, y = 4,
  label = "Interval 2", col = "blue", lwd = 3, pch = 11, cex = c(1.2, 3, 1.2)
)
PlotCoefs(points = points, bounds = bounds, intervals = intervals)

distributions <- list()
distributions$one <- list(type = "normal", mean = 0, var = 1, label = "Distribution 1")
distributions$two <- list(
  type = "gld", p1 = 0, p2 = 1.5, p3 = 1.2,
  p4 = 1.2, label = "Distribution 2", col = "blue", lwd = 3
)
distributions$three <- list(
  type = "cdfs", xs = seq(-2, 2, 0.1),
  cdfs = pnorm(seq(-2, 2, 0.1)), label = "Distribution 3",
  col = rgb(1, 0, 0, alpha = 0.5), lwd = 8
)
PlotCoefs(
  points = points, bounds = bounds, intervals = intervals,
  distributions = distributions, legendsTitle = NULL, legendSize = 7
)
```

`print.ldtf` *Prints an* `ldtf` *object*

Description

Prints an `ldtf` object

Usage

```
## S3 method for class 'ldtf'
print(x, ...)
```

Arguments

`x`	An `ldtf` object
`...`	additional arguments

Value

`NULL`

print.ldtsearch *Print an* ldtsearch *object*

Description

Print an ldtsearch object

Usage

```
## S3 method for class 'ldtsearch'
print(x, ...)
```

Arguments

x	ldtsearch object
...	additional arguments

Value

NULL

print.ldtv *Prints an* ldtv *object*

Description

Prints an ldtv object

Usage

```
## S3 method for class 'ldtv'
print(x, ...)
```

Arguments

x	An ldtv object
...	additional arguments

Value

NULL

`RemoveNaStrategies` *Remove NA and Count the Number of Observations in Different Scenarios*

Description

When a matrix has NA, one can omit columns with NA or rows with NA or a combination of these two. Total number of observations is a function the order. This function tries all combinations returns the results.

Usage

```
RemoveNaStrategies(
  data,
  countFun = function(nRows, nCols) nRows * nCols,
  rowIndices = NULL,
  colIndices = NULL,
  printMsg = FALSE
)
```

Arguments

`data`	A matrix with NA
`countFun`	a function to determine how strategies are sorted. Default counts the number of observations. You might want to give columns a higher level of importance for example by using `nRows*nCols^1.5`.
`rowIndices`	Indices of sorted rows to search. Use it to create jumps for large number of rows (E.g., if the first sorted strategies suggest small number of columns and you are looking for other strategies). Use `NULL` to disable
`colIndices`	similar to `rowsMaxCount` for columns.
`printMsg`	If `TRUE`, it prints progress.

Value

a list of lists with four elements:

- nRows: number of rows in the matrix
- nCols: number of cols in the matrix
- colFirst: whether to remove columns or rows first
- colRemove: indexes of the columns to be removed
- rowRemove: indexes of the rows to be removed

Examples

```
data <- matrix(c(NA, 2, 3, 4, NA, 5, NA, 6, 7, NA, 9, 10, 11, 12, 13, 14, 15, NA, 16, 17), 4, 5)
RemoveNaStrategies(data)
```

Search_s *Stepwise estimation*

Description

Stepwise estimation

Usage

```
Search_s(
  method,
  data,
  sizes = list(c(1, 2), c(3, 4), c(5), c(6:10)),
  counts = c(NA, 40, 30, 20),
  savePre,
  printMsg = FALSE,
  ...
)
```

Arguments

method	sur, dc or varma
data	exogenous (for sur and dc) or endogenous (for varma)
sizes	detemines the steps
counts	determines the size in each step
savePre	if not NULL, it saves and tries to load the progress of search step in a file (name=paste0(savePre,i) where i is the index of the step).
printMsg	If true, some information about the steps is printed. Note that it is different from searchers' printMsg.
...	Additional arguments

Value

the result

Sequence_F *Generates a Sequence for a frequency*

Description

Generates a Sequence for a frequency

Usage

```
Sequence_F(start, length)
```

Arguments

start	first element of the sequence. It must be an ldtf object returned from F_? functions.
length	Length of the sequence

Value

A list of strings

---

summary.ldtsearch *Summarize an* ldtsearch *object*

Description

Summarize an ldtsearch object

Usage

```
## S3 method for class 'ldtsearch'
summary(
  object,
  y,
  x = NULL,
  addModelBests = TRUE,
  addModelAll = FALSE,
  addItem1Bests = FALSE,
  printMsg = FALSE,
  w = NULL,
  newX = NULL,
  test = FALSE,
  ...
)
```

Arguments

object	ldtsearch object
y	dependent variables data (Data is not saved in object)
x	exogenous variables data (Data is not saved in object)
addModelBests	if TRUE and 'model$bests' exists (see [GetSearchItems()]), it estimates them.
addModelAll	if TRUE and 'all' exists (see [GetSearchItems()]), it estimates them.
addItem1Bests	if TRUE and 'item1' exists (see [GetSearchItems()]), it estimates them.

printMsg	if TRUE details are printed.
w	weight of observations (if available, e.g., in discrete choice estimation. Data is not saved in object)
newX	new exogenous data (if available, e.g., in varma estimation. Data is not saved in object)
test	If TRUE, it helps you make sure everything is working. Please report errors.
...	additional arguments

Value

a list with estimated models along with other kind of information. Its structure is similar to the given ldtsearch object.

SurEstim *Estimates an SUR Model*

Description

Estimates an SUR Model

Usage

```
SurEstim(
  y,
  x,
  addIntercept = TRUE,
  searchSigMaxIter = 0L,
  searchSigMaxProb = 0.1,
  restriction = NULL,
  newX = NULL,
  pcaOptionsY = NULL,
  pcaOptionsX = NULL,
  simFixSize = 0L,
  simTrainRatio = 0.75,
  simTrainFixSize = 0L,
  simSeed = 0L,
  simMaxConditionNumber = 1.7e+308,
  printMsg = FALSE
)
```

Arguments

y	(numeric matrix) Endogenous data with variables in the columns.
x	(numeric matrix) Exogenous data with variables in the columns.
addIntercept	(bool) If true, intercept is added automatically to x.

`searchSigMaxIter`
: (int) Maximum number of iterations in searching for significant coefficients. Use 0 to disable the search.

`searchSigMaxProb`
: (double) Maximum value of type I error to be used in searching for significant coefficients. If p-value is less than this, it is interpreted as significant and removed in the next iteration (if any exists).

`restriction`
: (nullable numeric matrix) A km x q matrix in which m=ncols(y), k=ncols(x) and q is the number of unrestricted coefficients.

`newX`
: (nullable numeric matrix) Data of new exogenous variables to be used in the predictions. Its columns must be the same as `x`. If null, projection is disabled.

`pcaOptionsY`
: (nullable list) A list of options in order to use principal components of the `y`, instead of the actual values. Set null to disable. Use `[GetPcaOptions()]` for initialization.

`pcaOptionsX`
: (nullable list) Similar to `pcaOptionsY` but for `x`. see `pcaOptionsY`.

`simFixSize`
: (int) Number of pseudo out-of-sample simulations. Use zero to disable the simulation. See also `GetMeasureOptions()]`.

`simTrainRatio`
: (double) Size of the training sample as a ratio of the number of the observations. It is effective only if `simTrainFixSize` is zero.

`simTrainFixSize`
: (int) A fixed size for the training sample. If zero, `simTrainRatio` is used.

`simSeed`
: (int) A seed for the pseudo out-of-sample simulation.

`simMaxConditionNumber`
: (double) Maximum value for the condition number in the simulation.

`printMsg`
: (bool) Set true to enable printing details.

Value

A list:

`SurSearch` *SUR Search*

Description

SUR Search

Usage

```
SurSearch(
  y,
  x,
  numTargets = 1L,
  xSizes = NULL,
  xPartitions = NULL,
```

```
  numFixXPartitions = 0L,
  yGroups = NULL,
  searchSigMaxIter = 0L,
  searchSigMaxProb = 0.1,
  measureOptions = NULL,
  modelCheckItems = NULL,
  searchItems = NULL,
  searchOptions = NULL
)
```

Arguments

y	(numeric matrix) endogenous data with variables in the columns.
x	(numeric matrix) exogenous data with variables in the columns.
numTargets	(int) determines the number of variable in the first columns of y for which the information is saved. It must be positive and cannot be larger than the number of endogenous variables.
xSizes	(nullable integer vector) Number of exogenous variables in the regressions. E.g., c(1,2) means the model set contains all the regressions with 1 and 2 exogenous variables. If null, c(1) is used.
xPartitions	(nullable list of integer vector) a partition over the indexes of the exogenous variables. No regression is estimated with two variables in the same group. If NULL, each variable is placed in its own group and the size of the model set is maximized.
numFixXPartitions	(int) number of partitions at the beginning of xPartitions to be included in all regressions.
yGroups	(nullable list of integer vector) different combinations of the indexes of the endogenous variables to be used as endogenous variables in the SUR regressions.
searchSigMaxIter	(int) maximum number of iterations in searching for significant coefficients. Use 0 to disable the search.
searchSigMaxProb	(double) maximum value of type I error to be used in searching for significant coefficients. If p-value is less than this, it is interpreted as significant.
measureOptions	(nullable list) see [GetMeasureOptions()].
modelCheckItems	(nullable list) see [GetModelCheckItems()].
searchItems	(nullable list) see [GetSearchItems()].
searchOptions	(nullable list) see [GetSearchOptions()].

Value

A list

SurSearch_s *Step-wise SUR Search*

Description

A helper class to deal with large model sets. It selects a subset of variables from smaller models and moves to the bigger ones.

Usage

```
SurSearch_s(
  x,
  xSizes = list(c(1, 2), c(3, 4), c(5), c(6:10)),
  counts = c(NA, 40, 30, 20),
  savePre = NULL,
  ...
)
```

Arguments

x	exogenous data
xSizes	a list of model dimension to be estimated in each step.
counts	a list of suggested number of variables to be used in each step. NA means all variables. Variables are selected based on best estimations (select an appropriate value for searchItems$bestK). All variables in the best models (all measures and targets) are selected until corresponding suggested number is reached.
savePre	if not NULL, it saves and tries to load the progress of search step in a file (name=paste0(savePre,i) where i is the index of the step).
...	other arguments to pass to SurSearch() function such as endogenous data. Note that xSizes is treated differently.

Value

A combined LdtSearch object

to.data.frame *Converts an* ldtv *object to a* data.frame

Description

There are five types of indices in this function: measures, targets, bests, type1's items, equations. Use NULL to use all available information or specify them.

Usage

```
to.data.frame(
  x,
  types = c("bestweights", "allweights", "inclusion", "type1bests", "cdf",
    "extremebounds", "mixture"),
  measures = NULL,
  targets = NULL,
  rows = NULL,
  columns = NULL,
  itemIndices = NULL,
  colNamFun = function(ns) {
     paste(ns[lengths(ns) > 0], collapse = ".")
 },
  rowContent = c("measure", "target", "item", "row", "column"),
  cdfIndex = 0,
  ...
)

to.data.frame(
  x,
  types = c("bestweights", "allweights", "inclusion", "type1bests", "cdf",
    "extremebounds", "mixture"),
  measures = NULL,
  targets = NULL,
  rows = NULL,
  columns = NULL,
  itemIndices = NULL,
  colNamFun = function(ns) {
     paste(ns[lengths(ns) > 0], collapse = ".")
 },
  rowContent = c("measure", "target", "item", "row", "column"),
  cdfIndex = 0,
  ...
)
```

Arguments

x	an ldtsearch object
types	(string vector) one or more that one type of information to be included in the the data.frame
measures	(integer or character array) measures to be used.
targets	(integer or character array) targets to be used.
rows	(integer or character array) If the requested object is a matrix (or an array), it determines the rows and cannot be NULL. For type1bests this is the name of the variables.
columns	(integer or character array) If the requested object is a matrix, it determines the columns and cannot be NULL. For type1bests this is the name of the fields:

	weight, mean, var
itemIndices	(integer array) items such as bests to be used.
colNamFun	(function) a function to determine the column names. The argument is a list of names, i.e., one of the following items: target, measure, row, column, item.
rowContent	(string) determines the type of information in the rows of returned data.frame. Some items are not available for some types. row is generally for variables in the rows of matrices such as inclusion or mixture. column is generally for the columns of such matrices. item is for the best models or models in the all field.
cdfIndex	(integer) The index of CDF if type is cdf
...	additional arguments

Value

a data.frame that contains data.

ToClassString_F *Converts an* ldtf *Object to String*

Description

The format is explained in F_? functions.

Usage

```
ToClassString_F(value)
```

Arguments

value	value of the frequency. It must be an ldtf object returned from F_? functions.

Value

An object of class 'ldtf'

ToString_F *Converts an* ldtf *Object to String*

Description

The format is explained in F_? functions.

Usage

```
ToString_F(value)
```

Arguments

value	value of the frequency. It must be an ldtf object returned from F_? functions.

Value

An object of class 'ldtf'

ToString_F0 *Similar to* ToString_F *and Return Value and Class as String*

Description

The format is explained in F_? functions.

Usage

```
ToString_F0(value)
```

Arguments

value	value of the frequency. It must be an ldtf object returned from F_? functions.

Value

An object of class 'ldtf'

Variable *Creates a Variable*

Description

Creates a Variable

Usage

```
Variable(data, name, startFrequency, fields)
```

Arguments

data	Data of the variable
name	Name of the variable
startFrequency	Frequency of the first data-point. It is an ldtf object. See F_? functions.
fields	Named list of any other fields

Value

An object of class ldtv.

Examples

```
v1 = ldt::Variable(c(1,2,3,2,3,4,5),"V1",F_Monthly(2022,12),
     list(c("key1","value1"), c("key2", "value2")))
```

VariableToString *Converts a Variable to String*

Description

Converts a Variable to String

Usage

```
VariableToString(w)
```

Arguments

w	The variable

Value

String representation of the variable in compact form

VarmaEstim *Estimates an VARMA Model*

Description

Estimates an VARMA Model

Usage

```
VarmaEstim(
  y,
  x = NULL,
  params = NULL,
  seasonsCount = 0L,
  addIntercept = TRUE,
  lmbfgsOptions = NULL,
  olsStdMultiplier = 2,
  pcaOptionsY = NULL,
  pcaOptionsX = NULL,
  maxHorizon = 0L,
  newX = NULL,
  simFixSize = 0L,
  simHorizons = NULL,
  simUsePreviousEstim = TRUE,
  simMaxConditionNumber = 1e+20,
  printMsg = FALSE
)
```

Arguments

y	(matrix) endogenous data with variables in the columns.
x	(matrix) exogenous data with variables in the columns.
params	(integer vector, length=6) parameters of the VARMA model (p,d,q,P,D,Q).
seasonsCount	(integer) number of observations per unit of time
addIntercept	(logical) if TRUE, intercept is added automatically to x.
lmbfgsOptions	(list) optimization options. See [GetLmbfgsOptions()].
olsStdMultiplier	(numeric) a multiplier for the standard deviation of OLS, used for restricting the maximum likelihood estimation.
pcaOptionsY	(list) a list of options in order to use principal components of the y, instead of the actual values. set NULL to disable. Use [GetPcaOptions()] for initialization.
pcaOptionsX	(list) similar to pcaOptionsY but for x. see pcaOptionsY.
maxHorizon	(integer) maximum prediction horizon. Set zero to disable.
newX	(matrix) data of new exogenous variables to be used in the predictions. Its columns must be the same as x.

`simFixSize` (integer) number of pseudo out-of-sample simulations. Use zero to disable the simulation. see also `[GetMeasureOptions()]`.

`simHorizons` (integer vector) prediction horizons to be used in pseudo out-of-sample simulations. see also `[GetMeasureOptions()]`.

`simUsePreviousEstim`
(logical) if `TRUE`, parameters are initialized in just the first step of the simulation. The initial values of the n-th simulation (with one more observation) is the estimations in the previous step.

`simMaxConditionNumber`
(numeric) maximum value for the condition number in the pseudo out-of-sample simulations.

`printMsg` (logical) set `FALSE` to disable printing the details.

Value

A list:

`VarmaSearch` *VARMA Search*

Description

VARMA Search

Usage

```
VarmaSearch(
  y,
  x = NULL,
  numTargets = 1L,
  ySizes = NULL,
  yPartitions = NULL,
  xGroups = NULL,
  maxParams = NULL,
  seasonsCount = 0L,
  maxHorizon = 0L,
  newX = NULL,
  interpolate = TRUE,
  adjustLeadsLags = TRUE,
  simUsePreviousEstim = TRUE,
  olsStdMultiplier = 2,
  lmbfgsOptions = NULL,
  measureOptions = NULL,
  modelCheckItems = NULL,
  searchItems = NULL,
  searchOptions = NULL
)
```

Arguments

y	(numeric vector) Endogenous data with variables in the columns.
x	(nullable numeric matrix) Exogenous data with variables in the columns. It can be null.
numTargets	(int) Number of variables in the first columns of y, regarded as targets. It must be positive and cannot be larger than the number of endogenous variables.
ySizes	(nullable integer vector) Determines the number of endogenous variables (or equations) in the regressions.
yPartitions	(nullable list of int vector) A partition over the indexes of the endogenous variables. No regression is estimated with two variables in the same group. If NULL, each variable is placed in its own group.
xGroups	(nullable list of int vector) different combinations of the indexes of the exogenous variables to be used as exogenous variables in the SUR regressions.
maxParams	(integer vector, length=6) Maximum values for the parameters of the VARMA model (p,d,q,P,D,Q). If null, c(1,1,1,0,0,0) is used.
seasonsCount	(integer) number of observations per unit of time
maxHorizon	(integer) maximum value for the prediction horizon if type1 is TRUE in checkItems. Also, it is used as the maximum prediction horizon in checking the predictions.
newX	(matrix) New exogenous data for out-of-sample prediction. It must have the same number of columns as x.
interpolate	(logical) if TRUE, missing observations are interpolated.
adjustLeadsLags	(logical) if TRUE, leads and lags in the sample are adjusted.
simUsePreviousEstim	(logical) if TRUE, parameters are initialized in just the first step of the simulation. The initial values of the n-th simulation (with one more observation) is the estimations in the previous step.
olsStdMultiplier	(numeric) a multiplier for the standard deviation of OLS, used for restricting the maximum likelihood estimation.
lmbfgsOptions	(list) Optimization options. see [GetLmbfgsOptions()]. Use null for default values.
measureOptions	(nullable list) see [GetMeasureOptions()].
modelCheckItems	(nullable list) see [GetModelCheckItems()].
searchItems	(nullable list) see [GetSearchItems()].
searchOptions	(nullable list) see [GetSearchOptions()].

Value

A list

VarmaSearch_s *Step-wise VARMA Search*

Description

A helper class to deal with large model sets. It selects a subset of variables from smaller models and moves to the bigger ones.

Usage

```
VarmaSearch_s(
  y,
  ySizes = list(c(1, 2), c(3, 4), c(5), c(6:10)),
  counts = c(NA, 40, 30, 20),
  savePre = NULL,
  ...
)
```

Arguments

y	endogenous data
ySizes	a list of model dimension to be estimated in each step.
counts	a list of suggested number of variables to be used in each step. NA means all variables. Variables are selected based on best estimations (select an appropriate value for searchItems$bestK). All variables in the best models (all measures and targets) are selected until corresponding suggested number is reached.
savePre	if not NULL, it saves and tries to load the progress of search step in a file (name=paste0(savePre,i) where i is the index of the step).
...	other arguments to pass to VarmaSearch() function such as endogenous data. Note that ySizes is treated differently.

Value

A combined LdtSearch object

vig_data *Data for Vignettes (and Tests)*

Description

A subset of different data sets generally for tests and vignettes. Data is generated from Data_? functions.

Format

A list

Details

- wdi. data from WDI data set.
- berka. data from Berka data set.
- vesta. data from Vesta data set.
- pcp. data from PCP data set.

Index