Package ‘Centaur’

April 5, 2017

Type Package

Title Centaur Propensity Score Balancing Workflow and Toolkit

Version 1.0.0

Date 2017-04-06

Author Sara Dempster
Stephen Kottmann
Alex Bayeh

Maintainer Alex Bayeh <alex.bayeh.centaur@gmail.com>

Description Performs propensity score based population balancing. This package is a toolkit to calculate propensity scores, balancea population dataset via either weighting or matching, and perform a variety of diagnostics to assess the scientific validity of the approach. The authors acknowledge the following team from AstraZeneca Pharmaceuticals, Robert LoCasale, Michael Goodman, Ramin Arani, Yiduo Zhang, and Sudeep Karve for contributing to the requirements with their expertise in epidemiology, safety informatics, health economics and biostatics and for reviewing the final product. The authors also acknowledge Jonathan Herz and Pramod Kumar for help with testing early versions of the package.

License Apache License 2.0

LazyData no

RoxygenNote 6.0.1

Imports AUC, broom, data.table, dplyr, ff, gtools, Hmisc, MASS, MatchIt, plyr, RJDBC, SDMTools, sm, survival, twang, vioplot

Suggests knitr, rmarkdown, testthat

VignetteBuilder knitr

R topics documented:

`ps.accuracy` *Accuracy*

Description

Calculates the accuracy curve of calculated propensity scores.

Usage

```
ps.accuracy(data)
```

Arguments

`data`	Data Frame - containing the dataset with previously calculated PS. The data frame must contain a treatment indicator variable called 'treat' and a propensity score value called 'ps_values'.

Details

This function uses the 'AUC' R package to calculate the accuracy of the propensity scores. For this to work properly, the argument ... should be a data frame containing the dataset with previously calculated propensity scores in a variable called ps_values.

Value

Calculated accuracy of propensity scores

Examples

```
ps.accuracy(myData)
```

---

ps.auc *AUC*

Description

Calculates the area under the curve for a set of calculated propensity scores.

Usage

```
ps.auc(data)
```

Arguments

`data`	Data Frame - containing the dataset with previously calculated PS. The data frame must contain a treatment indicator variable called 'treat' and a propensity score value called 'ps_values'.

Details

This function uses the 'AUC' R package to generate a AUC value of the propensity scores. For this to work properly, the argument `...` should be a data frame containing the dataset with previously calculated propensity scores in a variable called ps_values.

Value

The area under the curve

Examples

```
ps.auc(myData)
```

---

ps.balance *Propensity Score Population Balancing*

Description

Calculates the weights and/or perform matching of subject to balance the population

Usage

```
ps.balance(data, covariates, estimand = "ATT", match.subjects = TRUE,
  match.exact = NULL, match.ratio = 1, caliper.sigma = 0,
  use.logit = FALSE, truncate.quantile = 0.95, truncate.method = "cap",
  max.matching = 50000)
```

Arguments

`data`	Data Frame - containing the dataset with previously calculated PS. The data frame must contain a treatment indicator variable called 'treat' and a propensity score value called 'ps_values'.
`covariates`	Vector, containing the variable names to be included as potential confounding variables
`estimand`	String, specifying the desired estimand. Options are "ATT" (default) for the Average Treatment Effect in the Treated, or "ATE" for the Average Treatment Effect.
`match.subjects`	Boolean, indicating if matching should be used (default TRUE). This is only applicable when the estimand is "ATT" as "ATE" can only be estimated via IPTW. If the estimand is "ATT", SMRW will always be used to generate weights, but if match.subjects is set to TRUE, matches will also be generated.
`match.exact`	Vector, containing the list of covariate names to perform exact matching on.
`match.ratio`	Number, indicating the match ratio of control:treat
`caliper.sigma`	Number, indicating the width of the caliper to use in matching. A value of 0 indicates that calipers will not be used. A non-zero value will turn on calipers
`use.logit`	Boolean, indicating if the propensity score should be converted to logit before matching. This is generally recommended when using calipers and leads to better balance in most cases.
`truncate.quantile`	Number, indicating the upper quantile at which to apply weight trimming.
`truncate.method`	String, indicating the approach to use to trim the dataset. Large weights can adversely affect the ultimate balance of the population. Two approaches appear in the literature, capping weights at a quantile value or dropping subjects from the dataset. The default for this parameter is "cap", which will downwardly adjust any weights larger than the specified quantile to the value at that quantile. Alternatively, set this parameter to "drop" to remove the subjects from the dataset completely. User will be notified of how many subjects are lost in this step.
`max.matching`	The maximum number of samples that can be used in matching (default 50k)

Details

This function performs propensity score based population balancing. The details of how the population is balanced depend on the parameters specified by the user, including the requested estimand.

Value

psBalanceData - Object containing parameters used in balancing, along with the resulting data frame. The dataframe has additional variable(s) added for the weights and matches. When matching is used, the is_matched is [0,1] indicating if the subject was matched or not.

Examples

```
ps.balance(myData, covariates)
ps.balance(myData, covariates, match.subjects = TRUE, match.exact = c("GENDER"))
```

`ps.compare` *Propensity Score Comparison*

Description

Generates a graph used to compare the distribution of propensity score values in the cotnrol and treatment groups

Usage

```
ps.compare(data)
```

Arguments

`data`	Data Frame - containing the dataset with previously calculated PS. The data frame must contain a treatment indicator variable called 'treat' and a propensity score value called 'ps_values'.

Details

This function uses sm.density.compare to plot an overlapping density distribution plot of the propensity scores for the two groups allowing one to visually assess the area of common support

Examples

```
ps.compare(myData)
```

`ps.compute.survival` *Compute Survival Objects*

Description

Compute survival information using the survival package to be used in a response model

Usage

```
ps.compute.survival(data, outcome.dates, index.dates,
  obs.end.dates = "OBSEND", death.dates = "DEATHDATE")
```

Arguments

`data`	Data frame, containing the dataset to be analyzed
`outcome.dates`	String, containing the variable name with outcome dates
`index.dates`	String, containing the variable name for the index dates
`obs.end.dates`	String, containing the variable name for observation end dates (default = "OBSEND")
`death.dates`	String, containing the variable name for death dates (default = "DEATHDATE")

Details

This function leverages the survival package Surv function to compute survival information to be used as an outcome variable in outcome analysis. See survival::Surv documentation for more information.

Value

Vector, containing computed survival objects based on the outcome dates specified

Examples

```
ps.compute.survival(myData, "outcome")
```

`ps.connect.to.netezza` *Connect to Netezza*

Description

Establishes a connection to the Netezza database

Usage

```
ps.connect.to.netezza(user.name, password, driver.location, db.prefix, db.host)
```

Arguments

`user.name`	Your user name (prid)
`password`	Your password
`driver.location`	Location of your netezza driver, usually "nzjdbc.jar". Can be specified as complete path or the directory where the driver file is located.
`db.prefix`	This + prid = your database
`db.host`	URL of netezza appliance

Details

This function creates a connection to the Netezza database using the supplied credentials. This connection can then be used to pull data from the database into R.

Value

Database connection object

Examples

```
ps.connect.to.netezza("user123", "password123", "/users/user123/Documents/databaseDrivers")
```

`ps.covariate.biasplot` *Covariate Standard Differences of Means Plot*

Description

Visualization of the covariate standardized differences of means

Usage

```
ps.covariate.biasplot(balanced, unbalanced, abs = FALSE)
```

Arguments

`balanced`	Matrix, containing the summary statistics for covariates in the balanced population
`unbalanced`	Matrix, containing the summary statistics for covariates in the unbalanced / total population
`abs`	Boolean, indicating if the absolute percent reduction should be calculated. Default (FALSE)

Details

This function creates a plot of each covariate's standardized differences of means before and after balancing

Examples

```
ps.covariate.biasplot(summary.balanced, summary.unbalanced)
ps.covariate.biasplot(summary.balanced, summary.unbalanced, abs = TRUE)
```

`ps.covariate.qq` *QQ Plots*

Description

Create QQ plots for each of the continuous covariates

Usage

```
ps.covariate.qq(data, covariates, weights = NULL)
```

Arguments

`data`	Data frame, containing the balanced population data. The data frame must contain a treatment indicator variable called 'treat'.
`covariates`	Vector, containing the list of covariate names
`weights`	Vector, containing the weights to use in balanced population calculations. Defaults to unweighted.

Details

This function identifies each of the continuous covariates and creates a QQ plot, which allows one to visually inspect the similarity of distributions of covariates in the control and treatment populations

Examples

```
ps.covariate.qq(myData, covariates)
ps.covariate.qq(myData, covariates, myData$is_matched)
```

---

ps.covariate.residualplot
Covariate Residuals Plot

Description

A graphical balance diagnostic examining the ratio of residuals orthogonal to the propensity scores

Usage

```
ps.covariate.residualplot(data, covariates, weights = NULL,
  dichotomous = FALSE)
```

Arguments

`data`	Data Frame, containing the dataset. The data frame must contain a treatment indicator variable called 'treat'.
`covariates`	Vector, containing the list of covariates to include in the plot
`weights`	Vector, containing the weights to use in assessment of balanced population. Defaults to unweighted
`dichotomous`	Boolean value indicating if dichotomous covariates should be included in the calculation or not

Details

This function creates a plot of the residuals orthogonal to the propensity scores. Each covariate is regressed on the propensity scores for the control and treatment populations. The variance of the residuals of this regression are then compared. For a balanced population, the ratio of variances should be close to 1. An imbalance in the variances indicates that there is additional variance potentially attributed to the covariate in one of the populations that has not been captured in the propensity scores.

Examples

```
ps.covariate.residualplot(myData, covariates)
```

`ps.covariate.statistics` *Covariate Summary Matrix*

Description

Calculate the summary statistics for covariates

Usage

```
ps.covariate.statistics(data, covariates, weights = NULL, outputs = c("all",
  "dichotomous", "continuous"))
```

Arguments

`data`	Data Frame - containing the dataset with previously calculated weights or matches. The data frame must contain a treatment indicator variable called 'treat'.
`covariates`	Vector - containing the set of covariates for which to evaluate statistics.
`weights`	Vector, containing the weights to use in evaluating statistics for population. Defaults to equal weighting.
`outputs`	String - indicates which covariate type(s) to output. Accepted inputs are "all" (Default), "dichotomous", and "continuous"

Details

This function calculates a variety of summary statistics and organizes them into a matrix. Included summary statistics are the mean and sd in the treatment and control populations, the difference in means between the treat and control populations and the standardized difference of the means between the populations.

Value

Matrix - containing summary statistics for covariates

Examples

```
ps.covariate.statistics(myData, covariates)
ps.covariate.statistics(myData, covariates, weights = myData$weights)
ps.covariate.statistics(myData, covariates, weights = myData$is_matched)
ps.covariate.statistics(myData, covariates, outputs = "dichotomous")
```

ps.disconnect.from.netezza

Disconnect from Netezza

Description

Close a connection to Netezza, freeing resources

Usage

```
ps.disconnect.from.netezza(conn)
```

Arguments

conn	Database connection object created by ps.connect.to.netezza

Details

This function closes an existing connection to the netezza appliance to free resources

Examples

```
ps.disconnect.from.netezza(nzconn)
```

ps.getCohortMethodData

Cohort Method Data

Description

Reformat a data frame into a CohortMethodData object for use with OHDSI package

Usage

```
ps.getCohortMethodData(data, outcomes, covariates, treat.name = "treat",
  person.id.name = "PERSON_ID")
```

Arguments

data	Data Frame - containing the dataset to reformat.
outcomes	- outcome vector, or list of outcome vectors to include in data object. For TTE outcomes, include event times (NA for no event), for dichtomous data include 1/0 indicator
covariates	Vector of covariate names to include in the data object
treat.name	Name of the treatment indicator [0,1] variable (default "treat")
person.id.name	Name of the subject id variable (default "PERSON_ID")

Details

This function takes a data frame containing an observational study dataset and reformats into an object of type CohortMethodData that can be used with the OHDSI CohortMethod library.

Value

CohortMethodData - reformatted data object

Examples

```
ps.getCohortMethodData(myData)
```

`ps.km.plot` *Kaplan-Meier Plot*

Description

Creates a basic Kaplan-Meier plot for the survival analysis

Usage

```
ps.km.plot(km.analysis, title = "Kaplan-Meier Curves")
```

Arguments

`km.analysis`	A survival analysis object. See ps.survival()
`title`	Chart title (default "Kaplan-Meier Curves")

Details

This function creates a basic Kaplan-Meier plot for the supplied survival analysis. Basic assumptions are included about extent of axes, labels, etc.

Examples

```
ps.km.plot(model$km)
```

`ps.matched.boxplot` *Matched Sample Box and Whisker*

Description

Simple visual representation of matched samples

Usage

```
ps.matched.boxplot(data)
```

Arguments

`data`	Data Frame - containing the dataset with previously calculated matches. The data frame must contain a treatment indicator variable called 'treat' and a variable indicating whether the subject was matched called 'is_matched'.

Details

This function creates a box and whisker plot of matched and unmatched samples.

Examples

```
ps.matched.boxplot(myData)
```

`ps.matched.scatterplot` *Matched Sample Scatterplot*

Description

Simple visual representation of matched samples

Usage

```
ps.matched.scatterplot(data)
```

Arguments

`data`	Data Frame - containing the dataset with previously calculated matches. The data frame must contain a treatment indicator variable called 'treat' and a variable indicating whether the subject was matched called 'is_matched'.

Details

This function creates a simple scatter plot of matched and unmatched samples. Works best for smaller sample sizes.

Examples

```
ps.matched.scatterplot(myData)
```

`ps.matched.summary` *Simple population summary of matched samples*

Description

Summarize the matched samples.

Usage

```
ps.matched.summary(data)
```

Arguments

`data`	Data Frame - containing the dataset with previously calculated is_matched. The data frame must contain a treatment indicator variable called 'treat' and a variable indicating whether the subject was matched called 'is_matched'.

Details

This function does a simple summary counting of the matched vs unmatched samples. Print the resulting table to see counts of the original population, vs the populations that result from previously completed matching. (Not really useful for weighting as all subjects will have a non-zero weight)

Value

Matrix - containing the summary counts of treat and control in original and matched populations

Examples

```
ps.matched.summary(myData)
```

`ps.matched.violinplot` *Matched Sample Violinplot*

Description

Simple visual representation of matched samples

Usage

```
ps.matched.violinplot(data)
```

Arguments

`data`	Data Frame - containing the dataset with previously calculated matches. The data frame must contain a treatment indicator variable called 'treat' and a variable indicating whether the subject was matched called 'is_matched'.

Details

This function creates a violin plot of matched and unmatched samples. This plot shows the kernel density for the two populations - with an inner box and whisker plot of summary statistics. This plot tends to work better for large populations than the matched scatter plot.

Examples

```
ps.matched.violinplot(myData)
```

ps.percent.reduction *Percent Reduction in Standardized Differences of Means*

Description

Calculate the percent reduction in the standardized differences of the means for covariates

Usage

```
ps.percent.reduction(balanced, unbalanced, abs = FALSE)
```

Arguments

balanced	Matrix, containing the summary statistics for covariates in the balanced population
unbalanced	Matrix, containing the summary statistics for covariates in the unbalanced / total population
abs	Boolean, indicating if the absolute percent reduction should be calculated. Default (FALSE)

Details

This function uses the calculated covariate summary information for a balanced and total / unbalanced population and computes the percent reduction in the standardized differences of the means

Value

Matrix, containing the percent reduction in standardized differences of means for all covariates

Examples

```
ps.percent.reduction(summary.balanced, summary.unbalanced)
```

`ps.regression` *Outcome Regression Model*

Description

Fits a regression model to the specified outcome variable

Usage

```
ps.regression(data, outcome, covariates = character(), family = binomial(),
  w = NULL)
```

Arguments

`data`	Data frame, containing the dataset to be analyzed. The data frame must contain a treatment indicator variable called 'treat' and propensity score values called 'ps_values'.
`outcome`	String, containing the outcome variable name to be analyzed
`covariates`	Vector, containing the set of covariate variable names to include in the regression
`family`	Model family, passed through to glm. Defaults to binomial() - see ?glm for additional documentation
`w`	Vector, containing the subject weights. Defaults to equally weighted. If analysis of matched data is desired, set this value to myData$is_matched.

Details

This function uses the glm package to fit a regression model to the specified outcome variable. By default, the regression model will fit outcome ~ treat + ps_values. Inclusion of the PS values in the outcome model is recommended by literature producing a "doubly robust" analysis. In addition, any unbalanced covariates be included in the vector of covariate names parameter. These will also be included in the regression model. Weights for the regression model can be specified, or default to use the calculated weights in the data frame.

Value

Object, containing fitted model values. In addition to standard glm/lm output, the treatment effect is appended to the model object as model$treatment.effect. For dichotomous outcome variables, this is the odds ratio with confidence interval.

Examples

```
ps.regression(myData, "outcome")
ps.regression(myData, "outcome", covariates, w = myData$is_matched)
```

ps.roc *ROC*

Description

Generates a ROC curve for a set of calculated propensity scores.

Usage

```
ps.roc(data)
```

Arguments

`data`	Data Frame - containing the dataset with previously calculated PS. The data frame must contain a treatment indicator variable called 'treat' and a propensity score value called 'ps_values'.

Details

This function uses the 'AUC' R package to generate a ROC curve of the propensity scores. For this to work properly, the argument `...` should be a data frame containing the dataset with previously calculated propensity scores in a variable called ps_values.

Examples

```
ps.roc(myData)
```

ps.score *Centaur Propensity Score Calculation*

Description

Performs propensity score Calculation.

Usage

```
ps.score(data, covariates, ps.method = "glm", max.covariates = 200,
  max.twang = 30000, min.twang = 25, control.ratio = 5,
  random.seed = 43762116, odds.ratio = FALSE, lr.summary.file = NULL)
```

Arguments

`data`	Data Frame - containing the dataset to be balanced. Must include treatment indicator as 0, 1 factor.
`covariates`	Vector - containing the list of variable names to be included as confounding variables
`ps.method`	String - name of the method to use for calculation of propensity scores. Options are `glm` (default) and `twang`

`max.covariates`	The maximum number of covariates that can be used to calculate propensity scores (default 200)
`max.twang`	The maximum number of samples that can be used with the twang ps.method (default 30k)
`min.twang`	The minimum number of covariates required to use the twang ps.method (default 25)
`control.ratio`	The desired ratio of control to treatment subjects. A large imbalance between control and treatment subjects can cause problems with algorithm convergence. If the control population exceeds control.ratio times as many subjects as the treatment group, the control population will be randomly sampled to the desired size.
`random.seed`	Sets the random number generator seed, which determines how the control population is sampled. Override the default (43762116) to generate a different sample of control subjects.
`odds.ratio`	A logical argument indicating if the odds ratio and 95 This is an intensive calculation that can take a while.
`lr.summary.file`	The file name where the logistic regression summary should be written to. If directory is provided as part of the file name, it has to already exist. Default value is NULL. If not given, no file will be written.

Details

This function calculates the propensity score based on the specified options. In order for this function to work correctly, the `data` argument must be a data frame containing the collection of confounding variables and a treatment indicator factor variable. Additionally, the `covariates` argument must contain a vector of the variable names to be included as covariates or confounding variables. See list of arguments for additional options.

Value

Data Frame - containing the original dataset, trimmed of any incomplete cases with additional variable added for `ps_values` (the calculated propensity scores)

Examples

```
ps.score(myData, myCovariates)
ps.score(myData, myCovariates, ps.method = "twang")
```

`ps.score.summary` *Propensity Score Balance*

Description

Summarizes the balance of the propensity scores in treatment and control populations

Usage

```
ps.score.summary(data, weights = NULL)
```

Arguments

`data`	Data frame, containing the balanced population, including weights and/or matches. The data frame must contain a treatment indicator variable called 'treat'.
`weights`	Vector, containing the weights to use in assessing the balanced population (defaults to unweighted).

Details

This function calculates key summary statistics based on the propensity scores in the treatment and control populations. For a balanced population, one should observe similar mean values of the propensity scores in the two populations. Similarly to covariate diagnostics, one should observe a decrease in the standardized difference of the mean propensity score after balancing. Finally, one should see the ratio of variances in the propensity score move closer to 1 after balancing.

Value

Matrix, containing the summary statistics for the calculated propensity score

Examples

```
ps.score.summary(myData)
ps.score.summary(myData, weights = myData$is_matched)
```

`ps.sensitivity` *Sensitivity*

Description

Calculates the sensitivity curve of calculated propensity scores.

Usage

```
ps.sensitivity(data)
```

Arguments

`data`	Data Frame - containing the dataset with previously calculated PS. The data frame must contain a treatment indicator variable called 'treat' and a propensity score value called 'ps_values'.

Details

This function uses the 'AUC' R package to calculate the sensitivity of the propensity scores. For this to work properly, the argument ... should be a data frame containing the dataset with previously calculated propensity scores in a variable called ps_values.

Value

Calculated sensitivity of propensity scores

Examples

```
ps.sensitivity(myData)
```

`ps.specificity` *Specificity*

Description

Calculates the specificity curve of calculated propensity scores.

Usage

```
ps.specificity(data)
```

Arguments

`data`	Data Frame - containing the dataset with previously calculated PS. The data frame must contain a treatment indicator variable called 'treat' and a propensity score value called 'ps_values'.

Details

This function uses the 'AUC' R package to calculate the specificity of the propensity scores. For this to work properly, the argument `...` should be a data frame containing the dataset with previously calculated propensity scores in a variable called ps_values.

Value

Calculated specificity of propensity scores

Examples

```
ps.specificity(myData)
```

`ps.sql2df` *SQL to Data.Frame*

Description

Execute a sql query and return the results as a data.frame

Usage

```
ps.sql2df(conn, sql.query)
```

Arguments

`conn`	Database connection object created by ps.connect.to.netezza
`sql.query`	SQL query string, just as you might type into Aginity

Details

This function uses a database connection to execute a sql query string. The results returned by the query are returned as an R data frame.

Value

Calculated accuracy of propensity scores

Examples

```
myData <- ps.sql2df(nzconn, "select * from database.table limit 100")
```

`ps.strata.balance` *Strata Balance*

Description

Visualize the covariate balance within each stratum

Usage

```
ps.strata.balance(data, strata, covariates)
```

Arguments

`data`	Data Frame, containing the dataset. The data frame must contain a treatment indicator variable called 'treat'.
`strata`	Vector, containing the strata for the dataset
`covariates`	Vector, containing the covariate variable names to include

Details

This function creates a bar or box-and-whisker plot visualizing the summary statistics for each covariate in each of the strata

Examples

```
ps.strata.balance(myData, myData$strata, covariates)
```

`ps.strata.regression` *Evaluate a regression model*

Description

Evaluate a regression outcome model for the stratified dataset.

Usage

```
ps.strata.regression(data, strata, outcome, estimand = "ATT",
  covariates = character(), family = binomial())
```

Arguments

`data`	Data Frame - containing the stratified dataset. The data frame must contain a treatment indicator variable called 'treat' and a propensity score value called 'ps_values'.
`strata`	Vector - containing the stratum that each subject is assigned to
`outcome`	The outcome variable name
`estimand`	String, the desired estimand ("ATT" or "ATE"), default (ATT).
`covariates`	Vector, containing the names of the additional covariates to include in the outcome analysis
`family`	- Model family, passed through to glm

Details

This function evaluates an outcome model for each strata and then pools the models to evaluate the overall treatment effect

Value

An object of type "strata.model". This object contains the model information for each strata in object$models, as well as the pooled treatment effect information in object$treatment.effect.

Examples

```
ps.strata.regression(myData, strata, "outcome")
```

`ps.strata.survival` *Evaluate a survival model*

Description

Evaluate a survival outcome model for the stratified dataset.

Usage

```
ps.strata.survival(data, strata, outcome, estimand = "ATT",
  covariates = character())
```

Arguments

`data`	Data Frame - containing the stratified dataset. The data frame must contain a treatment indicator variable called 'treat' and a propensity score value called 'ps_values'.
`strata`	Vector - containing the stratum that each subject is assigned to
`outcome`	The outcome variable name. The variable must contain a precomputed survival object
`estimand`	String, the desired estimand ("ATT" or "ATE"), default (ATT).
`covariates`	Vector, containing the names of the additional covariates to include in the outcome analysis

Details

This function evaluates an outcome model for each stratum and then pools the models to evaluate the overall treatment effect

Value

An object of type "strata.model". This object contains the model information for each strata in object$models, as well as the pooled treatment effect information in object$treatment.effect.

Examples

```
ps.strata.survival(myData, myData$strata, "MISURVIVAL")
```

`ps.stratify` *Compute Strata*

Description

Stratifies the popultion.

Usage

```
ps.stratify(data, n = 5)
```

Arguments

`data`	Data Frame - containing the dataset with previously calculated PS. The data frame must contain propensity scores in a variable called 'ps_values'.
`n`	Number of strata to divide the dataset into (default 5)

Details

This function stratifies a popultion based on an estimated propensity score. The stratum of each subject is added to the dataset as a categorical value.

Value

Vector - containing the categorical strata number for each subject

Examples

```
ps.stratify(myData)
ps.stratify(myData, n = 10)
```

`ps.survival` *Outcome Survival Model*

Description

Performs survival analysis of an outcome variable

Usage

```
ps.survival(data, outcome, covariates = character(), w = NULL)
```

Arguments

`data`	Data frame, containing the dataset to be analyzed. The data frame must contain a treatment indicator variable called 'treat' and propensity score values called 'ps_values'.
`outcome`	String, containing the outcome variable name to be analyzed. The variable must be a Surv object
`covariates`	Vector, containing the set of covariate variable names to include in the model
`w`	Vector, containing the subject weights. Defaults to equal weighting. If analysis of matched data is desired, set this value to myData$is_matched.

Details

This function uses the survival package to do basic survival analysis for an outcome variable. The outcome variable should be previously computed as a survival object (see ps.compute.survival). The analysis will automatically perform survival::survfit and survival::coxph.

Value

Object, containing fitted model values. In addition to standard glm/lm output, the treatment effect is appended to the model object as model$treatment.effect. For dichotomous outcome variables, this is the odds ratio with confidence interval.

Examples

```
ps.survival(myData, "outcome")
```

`ps.trim` *Trim Population*

Description

Trims the population based on the propensity scores

Usage

```
ps.trim(data, trim.method = "overlap", trim.quantile = 0.95,
  quantile.group = "all")
```

Arguments

`data`	Data Frame - containing the dataset with previously calculated PS. The data frame must contain a treatment indicator variable called 'treat' and propensity score values called 'ps_values'.
`trim.method`	String, specifying the method to use to trim the dataset. Available options are "overlap" (default) to trim the non-overlapping tails of the control and treatment distributions, or "quantile" to trim to an interquantil range.
`trim.quantile`	Number, specifying the interquantile range to be trimmed. For example, 0.95 indicates that the dataset should be trimmed to the 0.025 / 0.975 interquantile range.
`quantile.group`	Specifies the group on which to determine the quantile values. Options are "all" (default) to determine the quantile values based on the propensity scores for the entire population, or "treat" to determine the quantile values based on the propensity scores for only the treatment group

Details

This function trims the supplied population based on propensity scores. Trimming can be based on the overlapping regions, or a specified interquantile range.

Value

Data Frame - trimmed dataset

Examples

```
ps.trim(myData)
ps.trim(myData, trim.method = "quantile", trime.quantile = 0.90)
```

Index