# Package 'uplift2'

December 21, 2017

**Title** Uplift Modeling

**Version** 0.0.1

**Maintainer** Leo Guelman <leo.guelman@gmail.com>

**Description** An integrated package for building and testing uplift models.

**Depends** R (>= 3.2.0), ggplot2 (>= 2.0.0), grid

**Imports** brglm (>= 0.5-9), car (>= 2.1-1), coin(>= 1.1-2), data.table (>= 1.9.6), doParallel (>= 1.0.10), dplyr (>= 0.7.4), foreach (>= 1.4.3), glmnet (>= 2.0-3), gridExtra (>= 2.0.0), iterators (>= 1.0.8), lazyeval (>= 0.1.10), magrittr (>= 1.5), MASS (>= 7.3-45), partykit (>= 1.0-5), plyr (>= 1.8.3), reshape2 (>= 1.4.1), RItools (>= 0.1-13), scales (>= 0.3.0), survival (>= 2.38-3), tables (>= 0.7.79)

**License** GPL-2 | GPL-3

**LazyData** true

**RoxygenNote** 6.0.1

**NeedsCompilation** no

**Author** Leo Guelman [aut, cre]

## R topics documented:

---

`create_data_partition` *Data splitting functions for uplift.*

---

## Description

`create_data_partition` creates one or more random data partitions into training and test sets. `create_kfolds` splits the data into k-folds (or groups) with approximatly the same number of observations. `create_bootstrap` draws bootstrap samples.

## Usage

```
create_data_partition(y, trt = NULL, p = 0.5, times = 1, groups = 5,
  replace = FALSE)

create_kfolds(y, trt = NULL, k = 10, times = 1, groups = 5)

create_bootstrap(y, times = 10)
```

## Arguments

| | |
|---|---|
| `y` | An atomic vector. |
| `trt` | An optional treatment variable. |
| `p` | The proportion of training observations. |
| `times` | The number of partitions to create. |
| `groups` | For numeric y, the number of breaks in the quantiles. |
| `replace` | Should sampling be done with replacement? |
| `k` | The number of folds. |

## Details

If `y` is a factor, sampling is done within the levels of `y` in an attempt to balance the class distributions between the partitions. If `y` is numeric, groups are first created based on the quantiles of its distribution and then sampling is done within these groups.

If `trt` is supplied, the data partitions are stratified by the treatment variable.

Notice that in addition to `create_bootstrap`, bootstrap samples can also by created using `create_data_partition` with `p = 1` and `replace = TRUE`.

## Value

create_data_partition and create_bootstrap return a matrix of row position integers corresponding to the training set and to the bootstrap sample, respectively. create_kfolds returns a matrix with the row integers corresponding to the folds.

## Author(s)

Leo Guelman <leo.guelman@gmail.com>

## Examples

```
set.seed(545)
r <- factor(sample(c(0,1), 1000, replace = TRUE))
t <- factor(sample(c(0,1), 1000, replace = TRUE))
df <- data.frame(r, t)
trainIndex <- create_data_partition(df$r, df$t)
dfTrain <- df[trainIndex, ]
dfTest <- df[-trainIndex, ]
table(df$r, df$t)
table(dfTrain$r, dfTrain$t)
table(dfTest$r, dfTest$t)
# Create k-folds
head(create_kfolds(r, t, times = 5))

# Create 10 bootstrap samples
set.seed(1)
x <- rnorm(100)
xb <- create_bootstrap(x, times = 10)
```

---

# create_uplift_formula *Create an uplift formula.*

---

## Description

create_uplift_formula is a helper function to create a formula object as required by most functions in the uplift package.

## Usage

```
create_uplift_formula(x, y, trt, env = parent.frame())
```

## Arguments

| | |
|---|---|
| x | A character vector of predictor names. |
| y | A character vector of the response name. |
| trt | A character vector of the treatment name. |
| env | The environment associated with the result, the calling environment by default. |

## Value

A formula object.

## Author(s)

Leo Guelman <leo.guelman@gmail.com>

## Examples

```
set.seed(1)
df <- sim_uplift(n = 100, p = 20, response = "binary")
f <- create_uplift_formula(names(df)[-c(1:3)], "y", "T")
class(f)
environment(f) # the callling environment
```

---

# ggplot.inspect_balance

*Plot a 'inspect_balance' object.*

---

## Description

ggplot method for class `inspect_balance`.

## Usage

```
## S3 method for class 'inspect_balance'
ggplot(x, i.var = NULL, n.type = "boxplot",
  f.type = "percent", legend.position = NULL, nrows = NULL,
  ncols = NULL)
```

## Arguments

| | |
|---|---|
| `x` | An object of class `inspect_balance`. |
| `i.var` | A numeric vector of indices of the variables to plot. The variables should be indexed in the same order that they appear in the initial inspect_balance formula. The default is to plot all variables. |
| `n.type` | The type of plot for numeric variables. Boxplots are generated by default. Alternatively, use `n.type = "density"` for density plots, or `n.type = "qqplot"` for qq-plots. |
| `f.type` | The type of plot for categorical variables. 100-percent stacked columns are generated by default. The alternative option is `"counts"`, which show stacked columns of counts. |
| `legend.position` | The position of legends (`"left"`, `"right"`,`"bottom"`, `"top"`). The default is `"right"`. |
| `nrows` | Number of rows for plots. |
| `ncols` | Number of columns for plots. |

## Author(s)

Leo Guelman <leo.guelman@gmail.com>

## Examples

```
set.seed(343)
df <- sim_uplift(n = 200, p = 50, response = "binary")
df$T <- ifelse(df$T == 1, 1, 0)
ib <- inspect_balance(T~ X1 + X2 + X3, data = df, method ="pdev", nPerm = 500)
ggplot(ib)
ggplot(ib, n.type = "density")
ggplot(ib, i.var = c(1,2), ncols =2, n.type ="density")
```

---

# ggplot.inspect_performance

*Qini and Calibration plots from a* `inspect_performance` *object.*

---

## Description

ggplot method for class 'inspect_performance'

## Usage

```
## S3 method for class 'inspect_performance'
ggplot(x, type = "qini",
  legend.position = "top", fillCol = "white", facets = FALSE,
  pointSize = FALSE, diagCol = "grey", xlim = NULL, ylim = NULL,
  xBreaks = NULL, yBreaks = NULL, xlab = NULL, ylab = NULL,
  title = NULL)
```

## Arguments

| | |
|---|---|
| `x` | An object of class `"inspect_performance"`. |
| `type` | The type of plot. Possible values are `qini` for Qini curves (default) and `calib` for calibration plots. |
| `legend.position` | The position of legends ("left", "right", "bottom", "top"). |
| `fillCol` | Panel's background color. |
| `facets` | Lay out panels in a grid? |
| `pointSize` | For `type = "calib"`, make size of points in plot proportional to the number of observations? |
| `diagCol` | For `type = "calib"`, the color of the diagonal line. |
| `xlim, ylim` | Numeric vectors of length 2, giving the x and y coordinates ranges. |
| `xBreaks, yBreaks` | Points at which x, y gridlines appear. |
| `xlab, ylab` | Title for the x, y axes. |
| `title` | The main title for the plot. |

## Author(s)

Leo Guelman <leo.guelman@gmail.com>

## Examples

```
set.seed(324)
df_train <- sim_uplift(p = 30, response = "binary")
df_test <- sim_uplift(n = 10000, p = 30, response = "binary")
fit_t1 <- glm(as.formula(paste('y ~', paste('X', 1:30, sep = '', collapse = "+"))),
              family = "binomial", data = df_train, subset = T==1)
fit_t0 <- glm(as.formula(paste('y ~', paste('X', 1:30, sep = '', collapse = "+"))),
              family = "binomial", data = df_train, subset = T==-1)
uplift_score <- predict(fit_t1, df_test, type = "response") -
                predict(fit_t0, df_test, type = "response")
df_test$uplift_score <- uplift_score
res <- inspect_performance(y ~ uplift_score + trueUplift + trt(T),
                           data = df_test, qini = TRUE)
res
summary(res)
ggplot(res)
ggplot(res, type = "calib", pointSize = TRUE)
```

---

`ggplot.niv` *Net weight of evidence plots from a 'niv' object.*

---

## Description

ggplot method for class `niv`.

## Usage

```
## S3 method for class 'niv'
ggplot(x, i.var = NULL, same.limits = FALSE, col = NULL,
  xlab = NULL, ylab = NULL, title = NULL, nrows = NULL, ncols = NULL)
```

## Arguments

| | |
|---|---|
| `x` | An object of class `niv`. |
| `i.var` | A numeric vector of indices of the variables to plot. The variables should be indexed in the same order that they appear in the initial niv formula. The default is to plot all variables. |
| `same.limits` | Should limits for the y-axis be the same in all plots? |
| `col` | The fill colour of the bars. |
| `xlab` | A character string giving the title for the x axis (the length should be the same as the number of plots). |
| `ylab` | A character string of length 1 giving the title for the y axis. |
| `title` | The main title for the plot (the length should be the same as the number of plots). |
| `nrows` | Number of rows for plots. |
| `ncols` | Number of columns for plots. |

## Author(s)

Leo Guelman <leo.guelman@gmail.com>

## Examples

```
set.seed(1)
df <- sim_uplift(n = 1000, p = 20, response = "binary")
netInf <- niv(y ~ trt(T) + X1 + X2 + X9 + X10, data = df,
            B=10, parallel = FALSE, digitsB = 1)
ggplot(netInf)
```

---

`ggplot.score_profile` *Plot a* `score_profile` *object.*

---

## Description

ggplot method for class 'score_profile'

## Usage

```
## S3 method for class 'score_profile'
ggplot(x, i.var = NULL, n.type = "boxplot",
  f.type = "percent", statistic = "mean", geom = "point",
  legend.position = NULL, col = NULL, size = NULL, nrows = NULL,
  ncols = NULL, xlab = NULL, ...)
```

## Arguments

| | |
|---|---|
| `x` | An object of class `score_profile`. |
| `i.var` | A numeric vector of indices of the variables to plot. The variables should be indexed in the same order that they appear in the initial inspect_balance formula. The default is to plot all variables. |
| `n.type` | The type of plot for numeric variables. Boxplots are generated by default. For alternative summary statistics, use `summary`. |
| `f.type` | The type of plot for categorical variables. 100 percent stacked columns are generated by default. The alternative option is `counts`, which show stacked columns of counts. |
| `statistic` | Functions that operate on a vector and produce a single value, as `mean` and `sd` do. It may be a user-defined function. |
| `geom` | The geometric object to display the data. Argument is passed to `ggplot2::stat_summary` when `type = summary`. For `type = summary`, `geom = "point"`. |
| `legend.position` | The position of legends (`"left"`, `"right"`,`"bottom"`, `"top"`). The default is `"right"`. |
| `col` | Color of `geom`. The default is `"red"`. |
| `size` | Size of `geom`. The default is `2`. |
| `nrows` | Number of rows for plots. |
| `ncols` | Number of columns for plots. |
| `xlab` | Title for the x. |
| `...` | Additional arguments passed to `ggplot2::stat_summary`. |

## Author(s)

Leo Guelman <leo.guelman@rbc.com>

## Examples

```
set.seed(123)
N <- 10000
eps <- rnorm(N)
age <- round(rnorm(N, 50, 10))
income <- rnorm(N, 60000, 10000) + 200 * age
gender <- gl(2, N/2,  labels = c("F", "M"))
insurance <- gl(4, N/4,  labels = c("HOME", "AUTO", "LIFE", "HEALTH"))
z <- 1e-01 + 0.1 * age - 1e-04 * income + 0.3 * (gender == "F") + eps
pr <- 1 / (1 + exp( -z))
purchase <- rbinom(N, 1, pr)
df <- data.frame(purchase, age, income, gender, insurance)
### Fit glm
pred <- fitted(glm(purchase ~ age + income + gender,
                   data = df, family = "binomial"))
profileForm <- pred ~ age + income + gender + insurance
prof1 <- score_profile(profileForm, data = df)
prof1
ggplot(prof1)
```

---

`inspect_balance` *Inspect balance of covariates.*

---

## Description

`inspect_balance` calculates standardized differences for each covariate between two treatment levels, and tests for conditional independence between the treatment and the covariates.

## Usage

```
## S3 method for class 'formula'
inspect_balance(formula, data, method = "dev",
  nPerm = NULL, midpval = TRUE, na.rm = FALSE, treatLevel = NULL, ...)

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

## S3 method for class 'inspect_balance'
summary(object, ...)
```

## Arguments

| | |
|---|---|
| `formula` | A formula containing an indicator of treatment assignment on the left hand side and covariates on the right. The treatment indicator should be numeric with 0/1 values. |
| `data` | A data frame in which to interpret the variables named in the formula. |

| | |
|---|---|
| method | The method used to compute a p-value associated with the balance test. See details. |
| nPerm | The number of random permutations of the treatment assignment. Only applicable with methods "pdev" and "paic". |
| midpval | Should the mid p-value be used? |
| na.rm | Should observations with NAs on any variables named in the RHS of formula be removed from the covariate balance summary table? |
| treatLevel | A character string for the treatment level of interest. By default, the treatment is coerced to a factor and the last level is used as the treatLevel. This argument is only relevant for calculating the standardized bias of covariates. |
| ... | Additional arguments passed to the various methods. Specifically, for methods "dev" and "pdev", arguments are passed to stats::glm. For "paic", arguments are passed to brglm::brglm, and for "hansen" they are passed to RItools::xBalance. |
| x | A inspect_balance object. |
| object | A inspect_balance object. |

## Details

In randomized experiments, the assignment of subjects to treatment and control groups is independent of their baseline covariates. As the sample size grows, random assignment tends to balance covariates, in the sense that both groups have similar distributions of covariates. Following Rosenbaum and Rubin (1985), we define the standardized bias on a covariate as

$$\frac{\bar{x}_t - \bar{x}_c}{\sqrt{\frac{s_t^2 + s_c^2}{2}}}$$

where $\bar{x_t}$ and $\bar{x_c}$ represent the sample means of a covariate in the treated and control groups, respectively, and $s_t^2$ and $s_c^2$ reresent their sample variances.

Another way to think about balance is that covariates $X$ should have no predictive power for treatment assignment $Z$. That is, $Prob(Z|X) = Prob(Z)$. Logistic regression is well suited for this task. If method = "dev" (default), we follow the approach suggested by Imai (2005). First regress treatment assignment $Z$ on the covariates $X$ and a constant, then on a constant alone, and then compare the two fits using a standard asymptotic likelihood-ratio test. This test is likely to perform poorly (i.e., high Type I error rates) in small samples (see Hansen, 2008). If method = "pdev", we compute a permutation distribution of the likelihood ratio statistic between the two models and compare it to the observe test statistic to obtain a p-value. Models are fitted using standard logistic regression. If method = "paic", the test statistic is given by the difference in AIC between the two models. A permutation distribution of this test statistic is computed and compared to its observed value to obtain a p-value for the test. Models are fitted using penalized likelihood using Jeffreys prior (Firth, 1993). Finally, if method = "hansen", p-values are computed using RItools::xBalance (Hansen, 2008).

We note that balance tests of this kind are subject to criticism, since balance is a characteristic of the sample, not some hypothetical population (see Ho et al., 2007).

## Value

An object of class inspect_balance, which is a list with the following components

- fit The fitted model object (NULL for method ="hansen").

- pvalue The p-value of the test.
- nObs The number of observations used by the procedure.
- cbs The covariate balance summary table.
- pdata The underlying data used in ggplot.inspect_balance.
- treatLevel The treatment level of interest.
- yLabel The name of the treatment indicator.
- call The call to inspect_balance.

## Author(s)

Leo Guelman <leo.guelman@gmail.com>

## References

Firth, D. (1993). "Bias reduction of maximum likelihood estimates". Biometrika 80, pp.27-38

Hansen, B.B. and Bowers, J. (2008)."Covariate Balance in Simple, Stratified and Clustered Comparative Studies". Statistical Science, 23, pp.219–236.

Ho, D., Kosuke I., King, G. and Stuart, E. (2007). "Matching as Nonparametric Preprocessing for Reducing Model Dependence in Parametric Causal Inference". Political Analysis 15, pp.199–236.

Kosuke, I. (2005). "Do Get-Out-The-Vote Calls Reduce Turnout? The Importance of Statistical Methods for Field Experiments". American Political Science Review, Vol. 99, No. 2 (May), pp. 283–300.

Rosenbaum, P.R. and Rubin, D.B. (1985). "Constructing a control group using multivariate matched sampling methods that incorporate the propensity score". The American Statistician, 39, pp.33–38.

## See Also

ggplot.inspect_balance.

## Examples

```
set.seed(343)
df <- sim_uplift(n = 200, p = 50, response = "binary")
df$T <- ifelse(df$T == 1, 1, 0)
ib <- inspect_balance(T~ X1 + X2 + X3, data = df, method ="pdev", nPerm = 500)
ib
summary(ib)
```

---

# inspect_performance *Inspect performance from a fitted uplift model.*

---

## Description

inspect_performance returns various performance measures from a fitted uplift model, including predicted uplift versus observed response under each treatment, and the Qini coefficient. The results can be used to seamlessly produce Qini curves and calibration plots.

## Usage

```
## S3 method for class 'formula'
inspect_performance(formula, data, subset,
  na.action = na.fail, method = "quantile", nBins = 10, qini = FALSE,
  qini.nBins = NULL, userBreaks = NULL, classLevel = NULL,
  treatLevel = NULL)

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

## S3 method for class 'inspect_performance'
summary(object, ...)
```

## Arguments

| | |
|---|---|
| formula | A model formula of the form y ~ pred_1 + ....+ pred_n + trt(), where the left-hand side corresponds to the observed response, and the right-hand side corresponds to the predicted values from an arbitrary number of uplift models, and 'trt' is the special expression to mark the treatment term. If the treatment term is not a factor, it is converted to one. |
| data | A data frame in which to interpret the variables named in the formula. |
| subset | Expression indicating which subset of the rows of data should be included. All observations are included by default. |
| na.action | A missing-data filter function. |
| method | Possible values are "quantile" (default) if you want to create intervals from the predicted uplift values with approximately the same number of observations in each group, "bucket" if you want to divide the predicted response values into equally spaced intervals, or "user" to create intervals from user-specified breaks (see details below). |
| nBins | The number of bins to create. |
| qini | Return the Qini coefficient for each model? |
| qini.nBins | The number of cutoffs in the predicted values used for constructing the qini curves. Cutoffs are created based on quantiles of the distribution of the predicted values. By default, cutoffs are given by unique values of the predictions. |
| userBreaks | A user-specified numeric vector of breaks for the predicted uplift values from which to create bins. It is required when method = "user", and ignored otherwise. |
| classLevel | A character string for the class of interest. Only applicable when the response is a factor. Defaults to the last level of the factor. |
| treatLevel | A character string for the treatment level of interest. Defaults to the last level of the treatment factor. |
| x | A inspect_performance object. |
| ... | Additional arguments for the S3 methods. |
| object | A inspect_performance object. |

## Details

The Qini curve (Radcliffe, 2007) is a two-dimensional depiction of model performance for uplift models. It represents a natural extension of the Gains curve (Blattberg et al., 2008, p. 319) for uplift models. It is constructed as follows: (i) observations are sorted by the uplift predictions in descending order, (ii) the cumulative number of responses are computed for each treatment and expressed as a percentage of the total number of observations within each treatment, (iii) the "net lift" is computed as the difference between the values obtained in (ii) for the reference treatment (`treatLevel`) and the alternative treatment. The Qini curve is a plot of the net lift versus the corresponding fraction of observations in the data.

The interpretation of the Qini curve is as follows: on the x-axis we show the fraction of subjects in the population in which the treatment is performed, and on the y-axis we show the difference in the success rate between the reference treatment and the alternative treatment.

A benchmark for a given uplift model can be represented by the strategy of randomly selecting subjects to perform the treatment. This is represented in the figure by the diagonal line. For example, if we perform the treatment on 30 percent of the population, we expect to obtain 30 percent of the net lift relative to performing the action on the entire population.

The Qini coefficient is a single estimate of model performance ranging from +100 (best possible model), to -100 (worst possible model). A value of zero represents a performance equivalent to that of a random model. The Qini coefficient is computed as

$$(AUQC_m - AUQC_r)/(AUQC_o - AUQC_r)$$

where $AUQC_m$, $AUQC_r$, and $AUQC_o$ represent the area under the Qini curve for the fitted uplift model, the random model, and the optimal model, respectively.

If `method = "quantile"` (the default), 'n' bins (`nBins`) are created based on quantiles of the distribution of the predicted values (in an attempt to have the same number of observations in each bin). If `method = "bucket"`, bins are created by dividing the predicted values into equally spaced intervals based on the difference between the minimum and maximum values. Unlike `method = "quantile"`, the number of observations in each group is typically unequal. If `method = "user"`, bins are created according to user-specified breaks (`userBreaks`).

In some cases, it may not be feasible to obtain the number of bins requested (`nBins`) (e.g., due to many ties in the predicted values). The function returns the effective number of bins created for each model (`actualBins`).

## Value

An object of class `"inspect_performance"`, which is a list with the following components:

- `data` The data for underlying calibration plots
- `method` The method used to create the bins
- `models` The labels of the variables supplied in the right-hand side of the model formula
- `nBins` The number of bins requested
- `actualnBins` The effective number of bins created for each model
- `yFactor` Is response a factor?
- `classLevel` The class of interest
- `treatLevel` The treatment level of interest
- `qiniCall` Whether the qini coefficient was requested in the function call
- `qiniData` The data for plotting the qini curves

- `qiniC` The qini coefficient
- `treatInd` The position of the treatment level of interest
- `call` The original call to `inspect_performance`

## Author(s)

Leo Guelman <leo.guelman@gmail.com>

## References

Blattberg, R. C., Do, K. B., and Scott, N. A. (2008). "Database Marketing: Analyzing and Managing Customers". Springer Science+Business Media, New York, NY.

Radcliffe, N (2007). "Using control groups to target on predicted lift: Building and assessing uplift models." Direct Marketing Analytics Journal, An Annual Publication from the Direct Marketing Association Analytics Council: pp. 14–21.

## See Also

`ggplot.inspect_performance`.

## Examples

```
set.seed(324)
df_train <- sim_uplift(p = 30, response = "binary")
df_test <- sim_uplift(n = 10000, p = 30, response = "binary")
fit_t1 <- glm(as.formula(paste('y ~', paste('X', 1:30, sep = '', collapse = "+"))),
              family = "binomial", data = df_train, subset = T==1)
fit_t0 <- glm(as.formula(paste('y ~', paste('X', 1:30, sep = '', collapse = "+"))),
              family = "binomial", data = df_train, subset = T==-1)
uplift_score <- predict(fit_t1, df_test, type = "response") -
                predict(fit_t0, df_test, type = "response")
df_test$uplift_score <- uplift_score
res <- inspect_performance(y ~ uplift_score + trueUplift + trt(T),
                           data = df_test, qini = TRUE)
res
summary(res)
```

---

`mom` *Modified outcome method for uplift modeling.*

---

## Description

`mom` transforms the response variable in a way that is relevant for subsequent uplift modeling. It handles continuous (uncensored) and categorical responses. A model fitted to this transformed response has a causal interpretation for the treatment effect conditional on the covariates.

## Usage

```
## S3 method for class 'formula'
mom(formula, data, subset, na.action, sampling = "none",
  newRespName = "z", classLevel = NULL, treatLevel = NULL)

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

## Arguments

| | |
|---|---|
| formula | A model formula of the form y ~ x1 + ....+ xn + trt(), where the left-hand side corresponds to the observed response, the right-hand side corresponds to the predictors, and 'trt' is the special expression to mark the treatment term. If the treatment term is not a factor, it is converted to one. |
| data | A data frame in which to interpret the variables named in the formula. |
| subset | Expression indicating which subset of the rows of data should be included. All observations are included by default. |
| na.action | A missing-data filter function. Defaults to na.omit. |
| sampling | The sampling method used to balance the treatment variable. See details. |
| newRespName | The name for the transformed response variable. |
| classLevel | A character string for the class of interest. Only applicable when the response is a factor. Defaults to the last level of the factor. |
| treatLevel | A character string for the treatment level of interest. Defaults to the last level of the treatment factor. |
| x | A mom object. |
| ... | Additional arguments for the S3 methods. |

## Details

Let $T \in [-1, 1]$ be a binary treatment indicator with $T = 1$ being the treatment level of interest (i.e., the treatment group). Also, let $y$ be a response variable. If the response is a factor, the transformed response is set to 1 if $T = 1$ and $y = 1$, or if $T = -1$ and $y = 0$ (assuming the classLevel of interest for $y$ is 1). Otherwise, the transformed response is set to 0. Under the specific case in which $Prob(T = 1) = Prob(T = -1) = 1/2$, it is easy to show that

$$2 * Prob(z = 1|X) - 1 = Prob(y = 1|T = 1, X) - Prob(y = 1|T = -1, X)$$

(Jaskowski and Jaroszewicz, 2012), where y, z, and X denote the original response variable, the transformed response, and the covariates, respectively.

If the response is numeric, it is transformed as $z = 2 * (y - \bar{y}) * T$. A model fitted to $z$ effectively estimates $E[y|T = 1, X] - E[y|T = -1, X]$ (Tian et al., 2014).

The argument sampling can be used to obtain a balanced treatment distribution. Specifically, if sampling = "oversample", observations from the treatment minority class are duplicated (by sampling with replacement), so that the resulting data frame has exactly the same number of observations under each treatment level. Alternatively, if sampling = "undersample", observations from the treatment majority class are dropped (by sampling without replacement), so that the resulting data frame has exactly the same number of observations under each treatment level. If sampling = "none", no sampling is done. Lastly, if sampling = "weights", the returned data frame includes a weight variable that equals (1 - $\pi$) for T = treatLevel and $\pi$ otheriwse, where $\pi = Prob(T = treatLevel)$. The weight variable can be subsequently used to perform case-weighted regression/classification on the transformed response.

## Value

An object of class "mom", which is a list with the following components (among others passed to the S3 methods):

- data The data set including the original response variable, the treatment indicator, the transformed response, the predictors, and (optionally) a weight variable.
- call The original call to mom.

## Author(s)

Leo Guelman <leo.guelman@gmail.com>

## References

Guelman, L., Guillen, M., and Perez-Marin A.M. (2015). "A decision support framework to implement optimal personalized marketing interventions." Decision Support Systems, Vol. 72, pp. 24–32.

Jaskowski, M. and Jaroszewicz, S. (2012)."Uplift modeling for clinical trial data". In ICML 2012 Workshop on Machine Learning for Clinical Data Analysis, Edinburgh, Scotland.

Tian, L., Alizadeh, A., Gentles, A. and Tibshirani, R. (2014). "A simple method for detecting interactions between a treatment and a large number of covariates." Journal of the American Statistical Association, 109:508, pp. 1517–1532,

## Examples

```
set.seed(324)
df_train <- sim_uplift(p = 15, response = "binary")
df_train_mcm <- mom(y ~  X1 + X2 + X3 + trt(T),
                    data = df_train, sampling = "undersample")
```

---

# niv *Net Information Value.*

---

## Description

niv computes the net information value for each uplift predictor. This can be a helpful exploratory tool to (preliminary) determine the predictive power of each variable for uplift.

## Usage

```
## S3 method for class 'formula'
niv(formula, data, subset, na.action = na.pass,
  nBins = 10, continuous = 4, B = 10, woeAdj = 0.5, parallel = TRUE,
  nCore = NULL, digitsB = NULL, classLevel = NULL, treatLevel = NULL)

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

## S3 method for class 'niv'
summary(object, ...)
```

## Arguments

| | |
|---|---|
| formula | A model formula of the form y ~ x1 + ....+ xn + trt(), where the left-hand side corresponds to the observed response, the right-hand side corresponds to the predictors, and 'trt' is the special expression to mark the treatment term. If the treatment term is not a factor, it is converted to one. niv only handles response variables of class factor. |
| data | A data frame in which to interpret the variables named in the formula. |
| subset | Expression indicating which subset of the rows of data should be included. All observations are included by default. |
| na.action | A missing-data filter function. Defaults to na.pass. |
| nBins | The number of bins created from numeric predictors. The bins are created based on sample quantiles, with a default value of 10 (deciles). |
| continuous | Specifies the threshold for when bins should be created from numeric predictors. If there are less or equal than n (i.e., continuous = n) unique values in the numeric predictor, it is coverted to a factor without binning. The default is continuous = 4. |
| B | The number of bootstraps. |
| woeAdj | The adjustment factor used to avoid an undefined WOE. The value should be between [0, 1]. By default woeAdj = 0.5. See details. |
| parallel | If TRUE, computations are performed in parallel, otherwise they are done sequentially. |
| nCore | The number of cores used. Default is: number of available cores-1. |
| digitsB | Number of digits used in formatting the breaks in numeric predictors. |
| classLevel | A character string for the class of interest. Defaults to the last level of the factor. |
| treatLevel | A character string for the treatment level of interest. Defaults to the last level of the treatment factor. |
| x | A niv object. |
| object | A niv object. |

## Details

Given a binary response variable $y \in (0, 1)$, the information value (Siddiqi, 2006) from a predictor $x$ is given by

$$IV = \sum_{i=1}^{G} (P(x = i|y = 1) - P(x = i|y = 0)) \times WOE_i$$

where $G$ is the number of groups created from a numeric predictor or levels from a categorical predictor, and $WOE_i = ln(\frac{P(x=i|y=1)}{P(x=i|y=0)})$.

To avoid an undefined WOE, an adjustment factor A is used. Specifically, $WOE_i = ln(\frac{(N(x=i|y=1)+A)/(N(y=1))}{(N(x=i|y=0)+A)/(N(y=0))})$, where $N$ represents observation counts.

The net information value (NIV) proposed by Larsen (2009) is a natural extension of the IV for the case of uplift. It is computed as

$$NIV = \sum_{i=1}^{G} (P(x = i|y = 1)^T \times P(x = i|y = 0)^C - P(x = i|y = 0)^T \times P(x = i|y = 1)^C) \times NWOE_i$$

where $NWOE_i = WOE_i^T - WOE_i^C$, and $T$ and $C$ refer to treatment and control groups, respectively.

The adjusted net information value (ANIV) is computed as follows

1. Draw B bootstrap samples from the training data and compute the NIV for each variable in each sample.
2. Compute the mean of the NIV ($NIV_{mean}$) and sd of the NIV ($NIV_{sd}$) for each variable over the $B$ replications.
3. The ANIV for a given variable is computed by subtracting a penalty term from the mean NIV. Specifically, $ANIV = NIV_{mean} - \frac{NIV_{sd}}{\sqrt{B}}$.

## Value

An object of class `niv`, which is a list with the following components (among others passed to the S3 methods):

- `nwoeData` A list of data frames, one for each variable. The columns represent:
  - `y00` the number of non-event records (response != `classLevel`) in the control group (treatment != `treatLevel`).
  - `y10` the number of event records (response == `classLevel`) in the control group (treatment != `treatLevel`).
  - `y01` the number of non-event records in the treatment group (treatment == `treatLevel`).
  - `y11` the number of event records in the treatment group.
  - `py00` proportion of non-event records in the control group.
  - `py10` proportion of event records in the control group.
  - `py01` proportion of non-event records in the treatment group.
  - `py11` proportion of event records in the treatment group.
  - `woe0` the control group weight-of-evidence.
  - `woe1` the treatment group weight-of-evidence.
  - `nwoe` the net weight-of-evidence.
  - `niv` the net information value.

  The values above are computed based on the entire data.
- `nivData` A data frame with the following columns: niv (the average net information value for each variable over all bootstrap samples), the penalty term, and the adjusted net information value.

## Author(s)

Leo Guelman <leo.guelman@gmail.com>

## References

Larsen, K. (2009). Net lift models. In: M2009 - 12th Annual SAS Data Mining Conference.

Siddiqi, N. (2006). Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring. Wiley, Hoboken, NJ.

## See Also

`ggplot.niv`.

## Examples

```
set.seed(1)
df <- sim_uplift(n = 1000, p = 20, response = "binary")
f <- create_uplift_formula(names(df)[-c(1:3)], "y", "T")
netInf <- niv(f, data = df, B=10, parallel = FALSE)
head(netInf$nivData)
```

---

# plot.utree *Visualization of uplift trees.*

---

## Description

plot method for class utree.

## Usage

```
## S3 method for class 'utree'
plot(x, ...)
```

## Arguments

| | |
|---|---|
| x | An object of class utree. |
| ... | Arguments passed to partykit::plot.constparty. |

## Author(s)

Leo Guelman <leo.guelman@gmail.com>

## Examples

```
set.seed(1)
df <- sim_uplift(n = 1000, p = 50, response = "binary")
form <- create_uplift_formula(x = names(df)[-c(1:3)], y = "y", trt = "T")
fit <- utree(form, data = df, maxdepth = 3)
plot(fit, main = "uplift tree", gp = gpar(cex = 0.5))
```

---

# predict.uforest *Predict method for uforest fits.*

---

## Description

Obtains predictions from a fitted uforest object.

## Usage

```
## S3 method for class 'uforest'
predict(object, newdata, ntrees = NULL, type = "uplift",
  agg.fun = "mean", ...)
```

### Arguments

| | |
|---|---|
| object | A fitted object inheriting from uforest. |
| newdata | A data frame to predict on. |
| ntrees | Number of trees used in prediction. All fitted trees are used by default. |
| type | The type of predictions required. Only uplift predictions are allowed. |
| agg.fun | The function used to combine the predictions from the individual trees. Must be functions that operate on a vector and produce a single value, as "mean" and "median" do. It may be a user-defined function. |
| ... | Additional arguments passed to agg.fun. |

### Value

A numeric vector of predictions.

### Author(s)

Leo Guelman <leo.guelman@gmail.com>

### See Also

uforest.

---

## predict.uplift_glm *Predict method for uplift_glm fits.*

---

### Description

Obtains predictions from a fitted uplift_glm object.

### Usage

```
## S3 method for class 'uplift_glm'
predict(object, newdata = NULL, type = "uplift",
  na.action = na.omit, ...)
```

### Arguments

| | |
|---|---|
| object | A fitted object inheriting from uplift_glm. |
| newdata | An optional set of data to predict on. If NULL, then the original data are used. |
| type | The type of predictions required. Only "uplift" predictions are allowed. |
| na.action | The method for handling missing data. |
| ... | Additional arguments to be passed to other methods. |

### Value

A numeric vector of predictions for all methods, except when the model was fitted using method = "glmnet", in which case the returned object is a matrix of predictions for the entire sequence of the penalty parameter used to create the model.

### Author(s)

Leo Guelman <leo.guelman@gmail.com>

### See Also

uplift_glm

---

predict.utree *Predict method for utree fits.*

---

### Description

Obtains predictions from a fitted utree object.

### Usage

```
## S3 method for class 'utree'
predict(object, newdata, type = "uplift", ...)
```

### Arguments

| | |
|---|---|
| object | A fitted object inheriting from utree. |
| newdata | A data frame to predict on. |
| type | The type of predictions required. The default is "uplift" for uplift predictions Alternatively, "node" returns an integer vector of terminal node identifiers. |
| ... | Additional arguments passed to partykit::predict.party. |

### Value

A numeric vector of predictions.

### Author(s)

Leo Guelman <leo.guelman@gmail.com>

### See Also

utree.

score_profile *Profile deciles from a fitted model.*

## Description

This function can be used to profile the deciles from a fitted model. Given a vector of numeric scores (fitted values) and predictors, it computes basic summary statistics for each predictor by score quantile.

## Usage

```
## S3 method for class 'formula'
score_profile(formula, data, groups = 10,
  statistic = "mean", direction = "D", categorize = TRUE, nBins = 4,
  continuous = 4, digitsN = NULL, digitsF = NULL, digitsB = NULL,
  groupVar = NULL, excludeNA = FALSE, LaTex = FALSE)

## S3 method for class 'score_profile'
is(x)

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

## Arguments

| | |
|---|---|
| formula | A formula expression of the form score ~ predictors, where the score represents the predictions from a fitted model. |
| data | A data frame in which to interpret the variables named in the formula. |
| groups | Number of groups of equal observations in which to partition the data set to show results. The default value is 10 (deciles). |
| statistic | Functions that operate on a vector and produce a single value, as mean and sd do. It may be a user-defined function. To request several statistics, use the + operator. For example, statistic = "mean + min + max". This argument only applies to numeric variables when categorize = FALSE. Factors are always shows as percentages within each group. |
| direction | Possible values are "D" or "I", for group number labels which are decreasing or increasing with the model score, respectively. |
| categorize | Should numeric predictors be categorized at their quantiles? |
| nBins | The number of bins created for numeric variables. The bins are created based on quantiles, with a default value of 4 (quartiles). Only applicable when categorize=TRUE. |
| continuous | When categorize=TRUE, it specifies the threshold for when a numeric variable should be categorized at their quantiles, or at their unique values. When there are at least continuous unique values, bins are created based on quantiles. Otherwise, the variables is converted to factor with levels being equal to the variable's unique values. |
| digitsN | Number of decimal places to show for numeric predictors. |
| digitsF | Number of decimal places to show for factor predictors. |
| digitsB | Number of digits used in formatting the breaks |

| | |
|---|---|
| groupVar | A character string with the variable name in the data which holds the grouped predictions. If this argument is not null, groups of predictions are not created based on their quantiles but already declared from the named variable supplied to this argument. |
| excludeNA | Should the results exclude observations with missing values in any of the variables named in the formula? |
| LaTex | Should the function output LaTex code? |
| x | A score_profile object. |
| ... | Additional arguments for the S3 methods. |

## Details

This function ranks the variable supplied in the left-hand side of the model formula and classifies it into groups with approximately the same number of observations. It subsequently calls the function tables::tabular to compute the average of each numeric predictor, and the distribution of each factor within each group.

## Value

An object of class score_profile, which is a list with the following components:

- data The data frame containing the data used for plotting.
- Table An object of class tabular See ?tables::tabular for details.

## Author(s)

Leo Guelman <leo.guelman@rbc.com>

## See Also

ggplot.score_profile.

## Examples

```
### Simulate some data
set.seed(123)
x1 <- rnorm(1000)
x2 <- rnorm(1000)
f1 <- sample(c(0, 1), 1000, replace = TRUE)
z <- 1 + 2 * x1 + 3 * x2  + f1
pr <- 1 / (1 + exp( -z))
y <- rbinom(1000, 1, pr)
df <- data.frame(y = y, x1 = x1, x2 = x2, f1 = factor(f1))
### Fit model and get fitted values
Fitted <- fitted(glm(y ~ x1 + x2 + f1, data = df, family = "binomial"))
### Profile deciles
score_profile(Fitted ~ x1 + x2 + f1, data = df, direction = "I")
```

`sim_uplift` *Uplift simulations.*

## Description

Numerical simulations for uplift modeling, as described in Tian et al. (2014).

## Usage

```
sim_uplift(n = 1000, p = 20, rho = 0.2, beta.par = sqrt(6),
  sigma0 = sqrt(2), response = "gaussian")
```

## Arguments

| | |
|---|---|
| `n` | The number of observations. |
| `p` | The number of predictors. |
| `rho` | The correlation between predictors. |
| `beta.par` | Size of main effects. See details. |
| `sigma0` | Multiplier of error term. See details. |
| `response` | The type of response distribution. Possible values are `"gaussian"` and `"binary"`. |

## Details

For the gaussian case, `sim_uplift` simulates data according to the following specification:

$$y = (\beta_0 + \sum_{j=1}^{p} \beta_j X_j)^2 + (\gamma_0 + \sum_{j=1}^{p} \gamma_j X_j + 0.8 X_1 X_2) T + \sigma_0 \epsilon$$

where the covariates $(X_1, \ldots, X_p)$ follow a mean zero multivariate normal distribution with a compound symmetric variance-covariance matrix, $(1-\rho)\mathbf{I}_p + \rho \mathbf{1}^{'}\mathbf{1}$, $\beta_0$ = `beta.par`^-1, $\beta_j$ = (2 * `beta.par`)^-1, $\gamma_0 = 0.4$, $\gamma_j = (0.8, -0.8, 0.8, -0.8, 0, \ldots, 0)$, $T = [-1, 1]$ is the treatment indicator gerated with equal probability at random, $\epsilon$ is $N(0, 1)$, and $\sigma_0$ = `sigma0`.

For the binary case,

$$y = I((\beta_0 + \sum_{j=1}^{p} \beta_j X_j)^2 + (\gamma_0 + \sum_{j=1}^{p} \gamma_j X_j + 0.8 X_1 X_2) T + \sigma_0 \epsilon \geq 0)$$

For further details, see Tian et al. (2014).

## Value

A data frame including the response variable (`y`), the treatment indicator (`T`), the "true" uplift effect (`trueUplift`), and the predictors (`X`).

## Author(s)

Leo Guelman <leo.guelman@gmail.com>

## References

Tian, L., Alizadeh, A., Gentles, A. and Tibshirani, R. (2014). "A simple method for detecting interactions between a treatment and a large number of covariates." Journal of the American Statistical Association, 109:508, pp. 1517–1532.

## Examples

```
set.seed(324)
df1 <- sim_uplift(p = 30, response = "binary")
str(df1)
df2 <- sim_uplift(n = 10000, p = 20)
str(df2)
```

---

# trt *Mark treatment term.*

---

## Description

An identity function, simply used to mark the treatment term in the model formula.

## Usage

```
trt(x)
```

## Arguments

| | |
|---|---|
| x | The treatment indicator variable. |

## Value

Same as x.

## Author(s)

Leo Guelman <leo.guelman@gmail.com>

---

# uforest *Fitting uplift random forest.*

---

## Description

uforest implements uplift random forests.

## Usage

```
## S3 method for class 'formula'
uforest(formula, data, na.action, classLevel = NULL,
  treatLevel = NULL, control = uforest_control(...), ...)

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

## Arguments

| | |
|---|---|
| formula | A model formula of the form y ~ x1 + ....+ xn + trt(), where the left-hand side corresponds to the observed response, the right-hand side corresponds to the predictors, and 'trt' is the special expression to mark the treatment term. At the moment, uforest only handles binary responses. |
| data | A data frame in which to interpret the variables named in the formula. |
| na.action | A missing-data filter function. |
| classLevel | A character string for the class of interest. Defaults to the last level of the factor. |
| treatLevel | A character string for the treatment level of interest. Defaults to the last level of the treatment factor. |
| control | A list with control parameters, see uforest_control. |
| ... | Arguments passed to uforest_control. |
| x | An object of class "uforest" |

## Details

uforest builds a sequence of de-correlated uplift trees (see utree) fitted on bootstrap samples of the training data. Additionally, the best split at each node is selected among a subset of predictors randomly selected at that node. See Guelman et al. (2015) for details.

## Value

An object of class "uforest".

## Author(s)

Leo Guelman <leo.guelman@gmail.com>

## References

Guelman, L., Guillen, M., and Perez-Marin A.M. (2015). "A decision support framework to implement optimal personalized marketing interventions." Decision Support Systems, Vol. 72, pp. 24–32.

Hothorn, T., Hornik, K. and Zeileis, A. (2006). "Unbiased recursive partitioning: A conditional inference framework". Journal of Computational and Graphical Statistics, 15(3): 651–674.

Rzepakowski, Piotr and Jaroszewicz, Szymon. (2011). "Decision trees for uplift modeling with single and multiple treatments". Knowledge and Information Systems, 32(2) 303–327.

Strasser, H. and Weber, C. (1999). "On the asymptotic theory of permutation statistics". Mathematical Methods of Statistics, 8: 220–250.

Su, X., Tsai, C.-L., Wang, H., Nickerson, D. M. and Li, B. (2009). "Subgroup Analysis via Recursive Partitioning". Journal of Machine Learning Research 10, 141–158.

## Examples

```
set.seed(1)
df <- sim_uplift(n = 1000, p = 50, response = "binary")
form <- create_uplift_formula(x = names(df)[-c(1:3)], y = "y", trt = "T")
fit <- uforest(form, data = df, maxdepth = 3, ntree = 10, nCore = 2)
fit
```

```
t1 <- fit$forest[[1]] # see structure of first tree
plot(t1,  main = "first tree...", gp = grid::gpar(cex = 0.5))
```

---

`uforest_control` *Control for uplift random forest.*

---

## Description

Various parameters that control aspects of the `uforest` fit.

## Usage

```
uforest_control(ntree = 100, minsplit = 40L, minbucket.t = 20L,
  minbucket.c = 20L, var.select.criterion = "pvalue",
  var.select.test = "asymptotic()", alpha = 0.05, bonferroni = FALSE,
  balance.sample = "undersample", split.criterion = "uplift",
  maxdepth = Inf, mtry = NULL, parallel = TRUE, nCore = NULL)
```

## Arguments

| | |
|---|---|
| `ntree` | Number of uplift trees to fit. |
| `minsplit` | The minimum number of observations in a node in order to be considered for splitting. |
| `minbucket.t` | The minimum number of treatment observations in any terminal <leaf> node. The `treatLevel` can be used to determine the treatment level of interest. |
| `minbucket.c` | The minimum number of control observations in any terminal <leaf> node. |
| `var.select.criterion` | The criterion used to select the variable for splitting. At the moment, only `"pvalue"` is accepted. The variable with minimum pvalue is selected for splitting. |
| `var.select.test` | The conditional null distribution of the test statistic. This is passed to the `distribution` argument in `coin::independence_test`. For example, for an approximative (Monte Carlo) reference distribution with B Monte Carlo replicates, use `approximate(B=999)`. |
| `alpha` | The maximum acceptable pvalue required in order to make a split. |
| `bonferroni` | Apply bonferroni adjustment to pvalue? |
| `balance.sample` | The sampling method used to balance the treatment variable. This attempts to have an equal representation of each treatment before implementing the independence test described in `var.select.test`. The options are `"undersample"` (default), `"oversample"`, `"none"`. See the argument `sampling` in `mom` for details. |
| `split.criterion` | The split criteria used at each node of each tree; Possible values are: `"uplift"` (default), `"kld"` (Kullback-Leibler divergence), `"ed"` (Euclidean divergence), `"l1d"` (L1-norm divergence). See details in Guelman et al. (2015). |
| `maxdepth` | Maximum depth of the tree. The default `maxdepth = Inf` means that no restrictions are applied to tree sizes. |
| `mtry` | Number of input variables randomly sampled as candidates at each node. The default is $\sqrt{p}$, where $p$ represents the number of covariates. |

| | |
|---|---|
| parallel | If TRUE, computations are performed in parallel, otherwise they are done sequentially. |
| nCore | The number of cores used. Default is: number of available cores-1. |

## Value

A list.

## Author(s)

Leo Guelman <leo.guelman@gmail.com>

---

uplift_glm *Fitting Uplift Generalized Linear Models.*

---

## Description

uplift_glm fits Uplift Generalized Linear Models, optionally with lasso or elasticnet regularization.

## Usage

```
## S3 method for class 'formula'
uplift_glm(formula, data, subset, na.action,
  family = "gaussian", method = "glm", sampling = "weights",
  treatLevel = NULL, Anova = FALSE, ...)

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

## Arguments

| | |
|---|---|
| formula | A model formula of the form y ~ x1 + ....+ xn + trt(), where the left-hand side corresponds to the observed response, the right-hand side corresponds to the predictors, and 'trt' is the special expression to mark the treatment term. If the treatment term is not a factor, it is converted to one. |
| data | A data frame in which to interpret the variables named in the formula. |
| subset | Expression indicating which subset of the rows of data should be included. All observations are included by default. |
| na.action | A missing-data filter function. |
| family | Response type. For family = "gaussian" (default), the response must be presented as numeric. For family = "binomial", the response must be a factor with two levels. If the response is numeric, it will be coerced into a factor. For family = "cox", the response must be a survival object, as returned by survival::Surv. |
| method | The method used for model fitting. If method = "glm" (default), the model is fitted on the modified covariates (see details) using stats::glm. If method = "glmStepAIC", the model is first fitted using stats::glm and then this is passed to MASS::stepAIC for AIC stepwise selection. Alternatively, for method = "glmnet" and method = "cv.glmnet", models are fitted using glmnet::glmnet and glmnet::cv.glmnet, respectively. |

| | |
|---|---|
| sampling | The sampling method used to balance the treatment variable. See details. |
| treatLevel | A character string for the treatment level of interest. Defaults to the last level of the treatment factor. |
| Anova | If TRUE, the analysis-of-variance table is returned using the function car::Anova. It does not apply to method = "cv.glmnet" or method = "glmnet". |
| ... | Additional arguments passed to the regression method selected in method. |
| x | A uplift_glm object. |

## Details

The function follows the method for uplift modeling proposed by Tian et al. (2014). This method consists in modifying the covariates in a simple way, and then fitting an appropriate regression model using the modified covariates and no main effects. See Tian et al. (2014) for details.

The argument sampling can be used to obtain a balanced treatment distribution. Specifically, if sampling = "oversample", observations from the treatment minority class are duplicated (by sampling with replacement), so that the data frame used in model fitting has exactly the same number of observations under each treatment level. Alternatively, if sampling = "undersample", observations from the treatment majority class are dropped (by sampling without replacement), so that the data frame used in model fitting has exactly the same number of observations under each treatment level. If sampling = "none", no sampling is done. Lastly, if sampling = "weights", the returned data frame includes a weight variable that equals (1 - $\pi$) for T = treatLevel and $\pi$ otherwise, where $\pi = Prob(T = treatLevel)$. These weights are subsequently used as case weights in the fitting process.

## Value

An object of class "uplift_glm", which is a list with the following components, in addition to the ones returned by the specific fitting method:

- call The calling expression
- na.action Information returned by model.frame on the special handling of NAs.
- xlevels The levels of predictors of class factor.
- Family The family used.
- method The method used.
- sampling The sampling method used.
- dataClasses The data classes of predictors.
- treatLevel The reference treatment level.
- ttReLabel The label of the transformed treatment indicator.
- modForm The model formula.
- modData The data frame used in model fitting.
- inbag The index of of which observations were used for fitting.
- weightVector The vector of weights used for fitting.

## Author(s)

Leo Guelman <leo.guelman@gmail.com>

## References

Tian, L., Alizadeh, A., Gentles, A. and Tibshirani, R. (2014). "A simple method for detecting interactions between a treatment and a large number of covariates." Journal of the American Statistical Association, 109:508, pp. 1517–1532.

## Examples

```
set.seed(1)
df_train <- sim_uplift(p = 50, response = "binary")
df_test<- sim_uplift(p = 50, n = 10000, response = "binary")
form <- as.formula(paste('y ~', 'trt(T) +',
       paste('X', 1:(ncol(df_train)-3), sep = '', collapse = "+")))
fit1 <- uplift_glm(form,
                   family = "binomial",
                   method = "glm",
                   data = df_train)
fit1
fit2 <- uplift_glm(form,
                   family = "binomial",
                   method = "glmStepAIC",
                   data = df_train)
fit2
fit3 <- uplift_glm(form,
                   family = "binomial",
                   method = "cv.glmnet",
                   data = df_train)
lambda.opt <- fit3$lambda.min
fit3 <- uplift_glm(form,
                   family = "binomial",
                   method = "glmnet",
                   data = df_train)
upliftPred1 <- predict(fit1, df_test)
upliftPred2 <- predict(fit2, df_test)
upliftPred3 <- predict(fit3, df_test, s=lambda.opt)
df_eval<- data.frame(upliftPred1 = upliftPred1,
                     upliftPred2 = upliftPred2,
                     upliftPred3 = upliftPred3,
                     y = df_test$y,
                     T = df_test$T)
res <- inspect_performance(y ~ upliftPred1 + upliftPred2 + upliftPred3 + trt(T),
                           data = df_eval, qini = TRUE)
res
summary(res)
ggplot(res)
res$qiniC
```

---

# utree *Fitting uplift trees.*

---

## Description

utree implements recursive partitioning for uplift modeling.

## Usage

```
## S3 method for class 'formula'
utree(formula, data, na.action, classLevel = NULL,
  treatLevel = NULL, control = utree_control(...), ...)

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

## S3 method for class 'utree'
nodeprune(x, ...)
```

## Arguments

| | |
|---|---|
| formula | A model formula of the form y ~ x1 + ....+ xn + trt(), where the left-hand side corresponds to the observed response, the right-hand side corresponds to the predictors, and 'trt' is the special expression to mark the treatment term. At the moment, utree only handles binary responses. |
| data | A data frame in which to interpret the variables named in the formula. |
| na.action | A missing-data filter function. |
| classLevel | A character string for the class of interest. Defaults to the last level of the factor. |
| treatLevel | A character string for the treatment level of interest. Defaults to the last level of the treatment factor. |
| control | A list with control parameters, see utree_control. |
| ... | Arguments passed to utree_control. |
| x | An object of class "utree" |

## Details

Roughly, the algorithm works as follows:

1. For each terminal node in the tree we test the global null hypothesis of no interaction effect between the treatment indicator and any of the covariates. Stop if this hypothesis cannot be rejected. Otherwise, select the input variable with strongest interaction effect. The interaction effect is measured by a p-value corresponding to an asymptotic or permutation test (Strasser and Weber, 1999) for the partial null hypothesis of independence between each covariate and a transformed response. Specifically, the response is transformed so the impact of the covariate on the response has a causal interpretation for the treatment effect (see details in Guelman et al. 2015)
2. Implement a binary split in the selected input variable.
3. Recursively repeate the two steps above.

Function nodeprune is not yet implemented for utree objects.

## Value

An object of class "utree".

## Author(s)

Leo Guelman <leo.guelman@gmail.com>

## References

Guelman, L., Guillen, M., and Perez-Marin A.M. (2015). "A decision support framework to implement optimal personalized marketing interventions." Decision Support Systems, Vol. 72, pp. 24–32.

Hothorn, T., Hornik, K. and Zeileis, A. (2006). "Unbiased recursive partitioning: A conditional inference framework". Journal of Computational and Graphical Statistics, 15(3): 651–674.

Rzepakowski, Piotr and Jaroszewicz, Szymon. (2011). "Decision trees for uplift modeling with single and multiple treatments". Knowledge and Information Systems, 32(2) 303–327.

Strasser, H. and Weber, C. (1999). "On the asymptotic theory of permutation statistics". Mathematical Methods of Statistics, 8: 220–250.

Su, X., Tsai, C.-L., Wang, H., Nickerson, D. M. and Li, B. (2009). "Subgroup Analysis via Recursive Partitioning". Journal of Machine Learning Research 10, 141–158.

## See Also

plot.utree.

## Examples

```
set.seed(1)
df <- sim_uplift(n = 1000, p = 50, response = "binary")
form <- create_uplift_formula(x = names(df)[-c(1:3)], y = "y", trt = "T")
fit <- utree(form, data = df, maxdepth = 3)
fit
```

---

# utree_control *Control for uplift trees.*

---

## Description

Various parameters that control aspects of the utree fit.

## Usage

```
utree_control(minsplit = 40L, minbucket.t = 20L, minbucket.c = 20L,
  var.select.criterion = "pvalue", var.select.test = "asymptotic()",
  alpha = 0.05, bonferroni = FALSE, balance.sample = "undersample",
  split.criterion = "uplift", maxdepth = Inf, mtry = Inf)
```

## Arguments

| | |
|---|---|
| minsplit | The minimum number of observations in a node in order to be considered for splitting. |
| minbucket.t | The minimum number of treatment observations in any terminal <leaf> node. The treatLevel can be used to determine the treatment level of interest. |
| minbucket.c | The minimum number of control observations in any terminal <leaf> node. |

`var.select.criterion`
: The criterion used to select the variable for splitting. At the moment, only `"pvalue"` is accepted. The variable with minimum pvalue is selected for splitting.

`var.select.test`
: The conditional null distribution of the test statistic. This is passed to the `distribution` argument in `coin::independence_test`. For example, for an approximative (Monte Carlo) reference distribution with B Monte Carlo replicates, use `approximate(B=999)`.

`alpha`
: The maximum acceptable pvalue required in order to make a split.

`bonferroni`
: Apply bonferroni adjustment to pvalue?

`balance.sample`
: The sampling method used to balance the treatment variable. This attempts to have an equal representation of each treatment before implementing the independence test described in `var.select.test`. The options are `"undersample"` (default), `"oversample"`, `"none"`. See the argument `sampling` in `mom` for details.

`split.criterion`
: The split criteria used at each node of each tree; possible values are: `"uplift"` (default), `"kld"` (Kullback-Leibler divergence), `"ed"` (Euclidean divergence), or `"l1d"` (L1-norm divergence). See details in Guelman et al. (2015).

`maxdepth`
: Maximum depth of the tree. The default `maxdepth = Inf` means that no restrictions are applied to tree sizes.

`mtry`
: Number of input variables randomly sampled as candidates at each node. The default `mtry = Inf` means that no random selection takes place.

## Value

A list.

## Author(s)

Leo Guelman <leo.guelman@gmail.com>

---

# `var_importance.uforest` *Variable importance for uplift trees and uplift random forest.*

---

## Description

This is the extractor function for variable importance measures as produced by `utree` and `uforest`.

## Usage

```
## S3 method for class 'uforest'
var_importance(x, type = "I", valid.data = NULL,
  error.fun = "sel")

## S3 method for class 'utree'
var_importance(x, type = "I", valid.data = NULL,
  error.fun = "sel")
```

## Arguments

| | |
|---|---|
| x | An object of class "utree" or "uforest" |
| type | Either "I" or "II", specifying the type of importance measure. See details. |
| valid.data | For type = "II", importance is measured based on a validation data frame, which must be provided. |
| error.fun | The prediction error used to compute variable importance when type = "II". Possible values are "sel" for squared-error loss (default), or "abs" for absolute loss. See details. |

## Details

For type I, the measure of importance given to a predictor is the sum of the values given by the split-criterion produced over all internal nodes for which it was chosen as the splitting variable. For uplift random forest, this relative influence measure is naturally extended by averaging the importance for each variable over the collection of trees. For type II, variable importance is measured based on an independent validation sample, with the aim of quantifying the prediction strength of each variable. This is achieved by first measuring the prediction accuracy on this validation sample. Subsequently, the values for the jth variable are randomly permuted, and the accuracy again computed. The decrease in accuracy as a result of this permutation is the importance attributed to the jth variable.The accuracy is measured by the squared-error or absolute error between the predicted and true uplift on each terminal node of the tree.

## Value

A data frame with the variable importance.

## Author(s)

Leo Guelman <leo.guelman@gmail.com>

## Examples

```
set.seed(1)
df <- sim_uplift(n = 1000, p = 50, response = "binary")
form <- create_uplift_formula(x = names(df)[-c(1:3)], y = "y", trt = "T")
fit <- utree(form, data = df, maxdepth = 3)
var_importance(fit)
```

# Index