Package 'varPro'

November 13, 2026

Version 3.4.0

Date 2026-11-13

Title Model-Independent Variable Selection via the Rule-Based Variable Priority

Author Min Lu [aut],
Aster K. Shear [aut],
Udaya B. Kogalur [aut, cre],
Hemant Ishwaran [aut]

Maintainer Udaya B. Kogalur <ubk@kogalur.com>

BugReports https://github.com/kogalur/varPro/issues/

Depends R (>= 4.3.0),

Imports randomForestSRC (>= 3.4.5), glmnet, parallel, foreach, gbm, BART, survival

Suggests mlbench, doMC, caret, MASS, igraph

SystemRequirements OpenMP

Description A new framework of variable selection, which instead of generating artificial covariates such as permutation importance and knockoffs, creates release rules to examine the affect on the response for each covariate where the conditional distribution of the response variable can be arbitrary and unknown.

License GPL (>= 3)

URL https://www.varprotools.org/ https://www.luminwin.net/
https://ishwaran.org/

NeedsCompilation yes

Repository CRAN

Date/Publication 2026-11-13 17:00:02 UTC

Contents

alzheimers 2
cv.varpro 4

gliomas 11
hrrecov 11
importance 14
isopro 19
ivarpro 23
outpro 29
partialpro 35
plot.ivarpro 43
plot.partialpro 46
predict.isopro 50
predict.ivarpro 52
predict.uvarpro 54
predict.varpro 56
uvarpro 58
varpro 68
varpro.news 75
varpro.strength 76

Index **78**

`alzheimers` *Alzheimer's Disease Dataset*

Description

Health, lifestyle, and clinical data for 2,149 individuals used for studying Alzheimer's Disease. Variables include demographics, cognitive assessments, medical conditions, and symptoms.

Usage

```
data(alzheimers)
```

Format

A data frame with 2,149 observations on the following variables:

Age: Age in years (60 to 90).

Gender: Gender (0 = Male, 1 = Female).

Ethnicity: Ethnicity (0 = Caucasian, 1 = African American, 2 = Asian, 3 = Other).

EducationLevel: Education level (0 = None, 1 = High School, 2 = Bachelor's, 3 = Higher).

BMI: Body Mass Index (15 to 40).

Smoking: Smoking status (0 = No, 1 = Yes).

AlcoholConsumption: Weekly alcohol consumption in units (0 to 20).

PhysicalActivity: Weekly physical activity in hours (0 to 10).

DietQuality: Diet quality score (0 to 10).

SleepQuality: Sleep quality score (4 to 10).

FamilyHistoryAlzheimers: Family history of Alzheimer's (0 = No, 1 = Yes).

CardiovascularDisease: Cardiovascular disease (0 = No, 1 = Yes).

Diabetes: Diabetes (0 = No, 1 = Yes).

Depression: Depression (0 = No, 1 = Yes).

HeadInjury: History of head injury (0 = No, 1 = Yes).

Hypertension: Hypertension (0 = No, 1 = Yes).

SystolicBP: Systolic blood pressure (90 to 180 mmHg).

DiastolicBP: Diastolic blood pressure (60 to 120 mmHg).

CholesterolTotal: Total cholesterol (150 to 300 mg/dL).

CholesterolLDL: LDL cholesterol (50 to 200 mg/dL).

CholesterolHDL: HDL cholesterol (20 to 100 mg/dL).

CholesterolTriglycerides: Triglycerides (50 to 400 mg/dL).

MMSE: Mini-Mental State Examination score (0 to 30). Lower is worse.

FunctionalAssessment: Functional score (0 to 10). Lower is worse.

MemoryComplaints: Memory complaints (0 = No, 1 = Yes).

BehavioralProblems: Behavioral problems (0 = No, 1 = Yes).

ADL: Activities of Daily Living score (0 to 10). Lower is worse.

Confusion: Presence of confusion (0 = No, 1 = Yes).

Disorientation: Presence of disorientation (0 = No, 1 = Yes).

PersonalityChanges: Presence of personality changes (0 = No, 1 = Yes).

DifficultyCompletingTasks: Difficulty completing tasks (0 = No, 1 = Yes).

Forgetfulness: Forgetfulness (0 = No, 1 = Yes).

Diagnosis: Alzheimer's diagnosis (No, Yes).

Details

This dataset is suitable for modeling Alzheimer's risk, performing exploratory analysis, and evaluating statistical and machine learning algorithms. All individuals are uniquely identified and evaluated on a standardized set of clinical and behavioral measures.

Source

Rabie El Kharoua (2024). Alzheimer's Disease Dataset. Available from Kaggle at https://www.kaggle.com/datasets/rabieelkharoua/alzheimers-disease-dataset

Examples

```
## load the data
data(alzheimers, package = "varPro")
o <- varpro(Diagnosis~.,alzheimers)
imp <- importance(o)
print(imp)
```

`cv.varpro` *Select VarPro Predictors and Assess Selection by Cross-Validation*

Description

Uses VarPro importance scores and random forest prediction error to select predictors. Returns a minimum-error selection and conservative and liberal alternatives. Optional cross-validation assesses how well the selection procedure predicts new observations and how consistently variables are selected.

Usage

```
cv.varpro(formula, data, nvar = 30, ntree = 150,
          local.std = TRUE, zcut = seq(0.1, 2, length = 50), nblocks = 10,
          split.weight = TRUE, split.weight.method = NULL, sparse = TRUE,
          nodesize = NULL, max.rules.tree = 150, max.tree = min(150, ntree),
          verbose = FALSE, seed = NULL,
          fast = FALSE, crps = FALSE,
          cv.folds = 0, foldid = NULL, ...)
```

Arguments

`formula`	Formula specifying the response and predictors. The supported outcome families are those of `varpro`: regression, multivariate regression, classification, and right-censored survival. See Details for multivariate scoring.
`data`	Training data frame. Incomplete observations are removed before ranking predictors and evaluating candidate models.
`nvar`	Maximum number of predictor columns kept during preliminary split-weight screening in `varpro`. Categorical predictors may contribute several indicator columns. Candidate models use the original predictors. Ignored by the screening step when neither computed nor custom split-weights are used.
`ntree`	Number of trees in the VarPro rule forest, each candidate prediction forest, and each outer assessment forest.
`local.std`	Should locally standardized rule comparisons be used to calculate importance? See `varpro`.
`zcut`	Numeric vector of positive importance cutoffs. Predictors with `z >= zcut` enter a candidate model. Cutoffs are sorted into increasing order, and each distinct variable set is evaluated once.
`nblocks`	Positive integer controlling how each candidate forest is divided into blocks of trees to calculate prediction error and its variation. Larger values give smaller blocks. Use `nblocks = 1` to evaluate the whole forest. See Details.
`split.weight`	Should preliminary split-weights guide the VarPro screening and rule generation? See `varpro`.

`split.weight.method`
: Preliminary weighting methods passed to `varpro`, for example `c("lasso", "tree")`. The default chooses the combination automatically.

`sparse`
: Should preliminary weighting concentrate more strongly on promising predictors? Passed to `varpro`.

`nodesize`
: Minimum terminal node size for the VarPro forest and, with `fast = FALSE`, the candidate prediction forests. Chosen automatically when `NULL`. Fast candidate forests use the settings of `rfsrc.fast`.

`max.rules.tree, max.tree`
: Rule-extraction limits passed to `varpro`. They determine the comparisons available for the importance ranking.

`verbose`
: Should progress and candidate-model performance summaries be printed?

`seed`
: Seed for VarPro, candidate forests, and optional outer cross-validation. Also call `set.seed` before `cv.varpro` to control all random sampling.

`fast`
: Should candidate prediction forests use `randomForestSRC::rfsrc.fast`? This uses smaller samples to grow each tree. When enough data are available, a common test sample is set aside to compare the candidate forests. See Details.

`crps`
: For survival, should prediction error be based on predicted survival probabilities using an integrated Brier score adjusted for censoring? With `FALSE`, the forest's default concordance error is used. See Details for the score definitions.

`cv.folds`
: Number of cross-validation folds for assessing the selection procedure. The default, zero, performs variable selection only. A value of at least two also evaluates prediction error on left-out observations and summarizes how often variables are selected under each of the three rules. Generated folds are stratified by response class or survival event status, as appropriate. The fold count is reduced with a warning when the data cannot support the requested number. This is separate from `nblocks` and the preliminary lasso's `nfolds` option.

`foldid`
: Optional numeric vector of fold labels, one per input data row, to specify the cross-validation groups. Observations used in the analysis must have consecutive integer labels starting at one. Entries for omitted observations are ignored. Supplying `foldid` enables cross-validation even when `cv.folds = 0`; a positive `cv.folds` must match the number of supplied folds. Use common labels to keep records from the same subject together.

`...`
: Additional named arguments passed to `varpro`, including `parallel`, `cores`, `nfolds`, `rmst`, and external-forest controls. The `sampsize` argument also controls candidate-forest sampling. See Details.

Details

`cv.varpro` helps choose which predictors to use in a prediction model. It ranks predictors using `varpro`, then compares the prediction errors of random forests built with different sets of higher-scoring variables. The result is a minimum-error selection, with conservative and liberal alternatives that allow fewer or more variables for a similar prediction error.

By default, the function selects variables using the supplied data. Optional cross-validation also assesses prediction on observations left out of variable selection and model fitting, and shows how consistently variables are selected. Use `cv.folds` to request this additional assessment.

Choosing variables: The `zcut` argument gives the importance cutoffs to compare. For each cutoff, variables whose importance score `z` is at least that cutoff are used together in a prediction forest. Increasing the cutoff removes lower-scoring variables. Each distinct variable set is evaluated once.

Selection uses the original variables. For a categorical predictor represented by several indicator columns, its score is the largest importance among those columns. Multivariate regression also takes the largest score across responses; classification uses overall importance. Prediction forests use the original outcome and predictors, so selecting a categorical variable includes all its levels.

Comparing prediction errors: By default, candidate forests are compared using out-of-bag (OOB) predictions: each observation is predicted using trees grown without that observation. Each forest is evaluated in blocks of trees, giving one prediction error per block. The mean of these errors is used to compare variable sets, and their standard deviation describes variation across blocks. With `nblocks = 1`, the whole forest is evaluated and the block standard deviation is zero.

To allow alternatives with similar prediction errors, an error limit is calculated as the smallest mean error plus the average block standard deviation across candidate models. Three selections are returned:

Minimum error The variable set with the lowest mean error, returned in `imp`.

Conservative The smallest variable set with mean error at or below the error limit and also below 1, returned in `imp.conserve`.

Liberal The largest variable set with mean error at or below the error limit, returned in `imp.liberal`.

Use the `variable` column in a returned table to fit a final prediction model with those predictors; see Examples.

Without outer cross-validation, `imp.conserve` is `NULL` when no model meets the conservative criterion. With outer cross-validation, an empty selection is replaced by the highest-ranked variable from the data used for that selection. This applies to the full-data analysis and each fold's training sample. If no candidate model has a finite error, a warning is issued and all three rules return the importance ranking without applying a cutoff, with cutoffs set to zero.

Performance measures: Lower error is better for every supported family.

Regression Mean squared prediction error divided by the response variance. Multivariate regression uses the first response for prediction error and combines responses for the importance ranking.

Classification Normalized Brier prediction error. For imbalanced two-class data, the random forests quantile classifier (RFQ) uses geometric-mean error, $1 - \sqrt{\text{sensitivity} \times \text{specificity}}$. The controls `use.rfq` and `iratio.threshold` apply to both variable ranking and candidate-model evaluation.

Survival, `crps = FALSE` Concordance error, $1 - C$, from the survival forest.

Survival, `crps = TRUE` An integrated Brier score for predicted survival probabilities, adjusted for censoring and divided by the maximum evaluation time.

Supplying `rmst` changes the survival summary used for the VarPro ranking. Prediction error is still evaluated against the original survival outcome using concordance error or CRPS.

Checking performance with cross-validation: The errors above guide variable selection. Outer cross-validation checks prediction on observations left out of both variable selection and model fitting.

For example, `cv.folds = 5` divides the observations into five groups, called folds. Each fold is left out in turn. The remaining folds are used to repeat variable ranking and cutoff selection, then fit prediction forests for the three selected variable sets. These forests predict the left-out observations. The selected variables can differ between folds.

Each fold gives one prediction error per selection rule, using the whole prediction forest. The mean and standard deviation of these fold errors summarize performance. The tree blocks controlled by `nblocks` are used to choose variables within each training sample.

The returned `imp`, `imp.conserve`, and `imp.liberal` tables give the selections and importance scores from the analysis of all available data. Added columns show how often each variable was selected across folds and how much its importance varied. Prediction errors on the left-out folds are summarized in each table's `cv` attribute; see Value and the `peakVO2` example.

Cross-validation takes additional computation because the analysis is repeated for each fold. Supply `foldid` to specify the groups, for example to keep records from the same subject together.

Computing options: `fast = TRUE` uses `rfsrc.fast` for candidate prediction forests. When enough data are available, a common test sample is set aside to compare these forests. Otherwise, they are compared using OOB predictions. The VarPro ranking uses all observations available in each selection run. Supply `sampsize` to control the number of observations sampled to grow each tree.

The `parallel` and `cores` arguments control the preliminary lasso calculations. Forest threading is controlled by **randomForestSRC**.

Value

An object of class `"cv.varpro"` containing:

`imp` Selected predictors and their importance scores for the minimum-error selection, with columns `variable` and `z`. Predictors are identified by their original names.

`imp.conserve` Corresponding conservative selection table.

`imp.liberal` Corresponding liberal selection table.

`err` Prediction errors for the candidate variable sets in the full-data analysis, with columns `zcut`, `nvar`, `err`, and `sd`: the importance cutoff, number of selected original predictors, mean block error, and standard deviation across tree blocks.

`zcut`, `zcut.conserve`, `zcut.liberal`
Cutoffs for the three full-data selections. See Details for fallback behavior.

With outer cross-validation, `variable` and `z` still describe the selection using all available data. Four columns are added to each importance table:

`cv.select` Percentage of outer folds in which the variable was selected under that table's rule, including top-variable fallbacks. A variable excluded during preliminary screening counts as unselected.

`cv.z.mean` Mean importance score across outer training samples. Uses all available scores, including those from folds where the variable was not selected.

`cv.z.sd` Sample standard deviation of those importance scores. Larger values indicate greater variation in importance across training samples. It is `NA` when fewer than two scores are available.

`cv.z.n` Number of training-fold importance scores contributing to `cv.z.mean` and `cv.z.sd`. This can be smaller than the number of folds when preliminary screening excludes a variable or its score is unavailable.

These importance summaries use finite scores before applying the selection cutoff. With five folds, a variable scored in every fold and selected in two has `cv.z.n = 5` and `cv.select = 40`. The mean is `NA` when no score is available.

With outer cross-validation, each importance table has a `cv` attribute containing its mean fold error (`err`, with equal weight for each fold), standard deviation of fold errors (`sd`), performance measure (`perf.type`), and per-fold results (`folds`). It also records selection percentages for all original predictors (`selection.frequency`) and whether the full-data selection used the top-variable fallback (`fallback`). For example, use `attr(o$imp, "cv")` to inspect the minimum-error rule's assessment. Fold assignments are available through `attr(o, "cv")$foldid`, in input-row order, with `NA` for omitted observations.

Other attributes contain the importance object for individual predictor columns (`imp.org`), processed and original predictor names (`xvar.names`, `xvar.org.names`), and the VarPro forest family (`family`). Use `get.vimp(o)` to inspect scores for individual columns, including indicator columns for categorical predictors. Use `get.orgvimp(o, pretty = FALSE)` for original-variable scores with unselected variables filled by zero.

The print method displays the selections and a compact cross-validation performance summary when available.

Note

For serial lasso calculations, supply `parallel = FALSE`. Outer error summaries are `NA` when a fold error is undefined or folds use different performance measures. Inspect `attr(o$imp, "cv")$folds` for the individual fold results.

Author(s)

Min Lu and Hemant Ishwaran

References

Lu, M. and Ishwaran, H. (2024). Model-independent variable selection via the rule-based variable priority. arXiv:2409.09003. doi:10.48550/arXiv.2409.09003.

O'Brien, R. and Ishwaran, H. (2019). A random forests quantile classifier for class imbalanced data. *Pattern Recognition*, 90, 232–249.

See Also

`varpro`, `importance.varpro`, `predict.varpro`, `randomForestSRC::rfsrc`, `randomForestSRC::rfsrc.fast`.

Examples

```
## Nonlinear regression with a categorical effect and known signal set.
set.seed(137)
n <- 500
x <- matrix(runif(n * 10), nrow = n)
```

```
colnames(x) <- sprintf("x%02d", seq_len(ncol(x)))
group <- factor(sample(c("low", "middle", "high"), n, replace = TRUE),
                levels = c("low", "middle", "high"))
mu <- 10 * sin(pi * x[, 1] * x[, 2]) +
      20 * (x[, 3] - 0.5)^2 + 10 * x[, 4] + 5 * x[, 5] +
      3 * (group == "high")
d <- data.frame(y = mu + rnorm(n), x, group)
signal <- c(colnames(x)[1:5], "group")

set.seed(137)
o <- cv.varpro(y ~ ., d, ntree = 80,
               zcut = seq(0.1, 3, length.out = 10), nblocks = 6,
               max.tree = 60, max.rules.tree = 40,
               nfolds = 5, parallel = FALSE)

## Compare the three full-data selections.
print(o$imp)
print(o$imp.conserve)
print(o$imp.liberal)

print(data.frame(
  rule = c("minimum", "conservative", "liberal"),
  cutoff = c(o$zcut, o$zcut.conserve, o$zcut.liberal),
  nvar = c(NROW(o$imp), NROW(o$imp.conserve), NROW(o$imp.liberal))))
print(data.frame(variable = o$imp$variable, z = o$imp$z,
                 signal = o$imp$variable %in% signal))

## Original-variable scores and encoded-column scores at each cutoff.
print(get.orgvimp(o, pretty = FALSE))
print(get.vimp(o, pretty = FALSE))

## Plot the error path. Bars show one tree-block standard deviation.
e <- as.data.frame(o$err)
e <- e[is.finite(e$err) & is.finite(e$sd), , drop = FALSE]
if (nrow(e) > 0) {
  ylim <- range(c(pmax(0, e$err - e$sd), e$err + e$sd))
  plot(e$zcut, e$err, type = "b", pch = 19, ylim = ylim,
       xlab = "Importance cutoff",
       ylab = "Mean tree-block prediction error")
  arrows(e$zcut, pmax(0, e$err - e$sd),
         e$zcut, e$err + e$sd, angle = 90, code = 3, length = 0.04)
  abline(h = min(e$err) + mean(e$sd), lty = 2)
  abline(v = o$zcut, lty = 3)
}

## Train a final learner with the minimum-error variable set.
selected <- o$imp$variable
if (length(selected) > 0 && any(is.finite(o$err[, "err"]))) {
  final <- randomForestSRC::rfsrc(
    y ~ ., d[, c("y", selected), drop = FALSE], ntree = 300)
}

## Outer cross-validation: survival prediction with peakVO2.
```

```
## Each fold repeats ranking and cutoff selection on its training data.
library(survival)
data(peakVO2, package = "randomForestSRC")
peak <- na.omit(peakVO2)

set.seed(137)
peak.cv <- cv.varpro(Surv(ttodead, died) ~ ., peak,
                     cv.folds = 5, ntree = 100, seed = 137,
                     zcut = seq(0.1, 3, length.out = 12), nblocks = 6,
                     max.tree = 75, max.rules.tree = 40,
                     nfolds = 5, parallel = FALSE)

## The selections and z scores come from the full-data analysis.
## cv.select gives the percentage of outer folds selecting each variable.
print(peak.cv$imp)
print(peak.cv$imp.conserve)
print(peak.cv$imp.liberal)

## Compare held-out performance for the three rules.
## Error is 1 - C; lower is better. cv.sd is the deviation across folds.
## nvar.full counts predictors selected in the full-data analysis.
rules <- c("imp", "imp.conserve", "imp.liberal")
cv.summary <- do.call(rbind, lapply(rules, function(rule) {
  tab <- peak.cv[[rule]]
  assessment <- attr(tab, "cv")
  data.frame(rule = rule, nvar.full = nrow(tab),
             cv.err = assessment$err, cv.sd = assessment$sd,
             perf.type = assessment$perf.type)
}))
print(cv.summary)

## Inspect fold errors, variable counts, cutoffs, and fallback status.
print(attr(peak.cv$imp, "cv")$folds)

## Compare selection percentages for all original predictors,
## including those absent from the full-data selections.
cv.select <- sapply(rules, function(rule) {
  attr(peak.cv[[rule]], "cv")$selection.frequency
})
print(cv.select[order(cv.select[, "imp"], decreasing = TRUE),
                , drop = FALSE])

## Check the sizes of the actual outer folds.
print(table(attr(peak.cv, "cv")$foldid))

## Survival: RMST defines the ranking target; CRPS evaluates the
## resulting survival prediction models on their OOB observations.
library(survival)
data(pbc, package = "randomForestSRC")
pbc <- na.omit(pbc)
set.seed(137)
s <- cv.varpro(Surv(days, status) ~ ., pbc,
               rmst = 1000, crps = TRUE, fast = FALSE,
```

```
                ntree = 100, nblocks = 6,
                zcut = seq(0.1, 3, length.out = 12),
                max.tree = 75, max.rules.tree = 40,
                nfolds = 5, parallel = FALSE)
print(s$imp)
print(s$err)
```

---

gliomas *Diffuse Adult Glioma*

Description

Subset of the data used in Ceccarelli et al. (2016) for molecular profiling of adult diffuse gliomas. As part of the analysis, the authors developed a supervised analysis using DNA methylation data. Their original dataset was collected from a core set of 25,978 CpG probes which was reduced to eliminate sites that were methylated. This reduced set of 1206 probes from 880 tissues makes up part of the features of this data. Also included are clinical data and other molecular data collected for the samples. The outcome is a supervised class label developed in the study with labels: Classic-like, Codel, G-CIMP-high, G-CIMP-low, LGm6-GBM, Mesenchymal-like and PA-like.

References

Ceccarelli, M., Barthel, F.P., Malta, T.M., Sabedot, T.S., Salama, S.R., Murray, B.A., Morozova, O., Newton, Y., Radenbaugh, A., Pagnotta, S.M. et al. (2016). Molecular profiling reveals biologically discrete subsets and pathways of progression in diffuse glioma. Cell, 164, 550-563.

Examples

```
data(glioma, package = "varPro")
o <- varpro(y~., glioma, nodesize=2, max.tree=250)
imp <- importance(o)
print(head(imp$unconditional))
print(imp$conditional.z)
```

---

hrrecov *Exercise Heart Rate Recovery and Mortality*

Description

A survival dataset from a cohort of patients referred for symptom-limited exercise testing, originally used to study exercise heart rate recovery as a predictor of all-cause mortality.

Usage

```
data(hrrecov)
```

Format

A data frame with 23701 observations on 83 variables.

`ttodead` Follow-up time to death or censoring, in years (rounded to 2 decimals).

`died` Event indicator (1 = death, 0 = right-censored).

`hrrecov` Heart rate recovery (beats/min): peak heart rate minus heart rate 1 minute into recovery.

`lowrec` Indicator for low/abnormal heart rate recovery.

`lowcri` Indicator for low chronotropic response index/chronotropic incompetence.

`peak_hr` Peak heart rate during exercise (beats/min).

`peak_met` Peak workload achieved (metabolic equivalents, METs).

`fitness` Fitness category (ordinal 1-5; coding as in source data).

`heart_ra` Resting heart rate prior to testing (beats/min).

`sbprest` Resting systolic blood pressure (mm Hg).

`dbprest` Resting diastolic blood pressure (mm Hg).

`age` Age at exercise test (years).

`gender` Sex indicator (1=men).

`race` Race category (integer code; coding as in source data).

`black` Indicator for Black race (0/1).

`height` Height (m).

`weight` Weight (kg).

`bmi` Body mass index (kg/m^2).

`bsa` Body surface area.

`wtht` Weight-to-height ratio (weight/height; kg/m).

`obese` Indicator for obesity (0/1).

`priorcad` Known or suspected coronary artery disease prior to test (0/1).

`mihist` History of myocardial infarction (0/1).

`pcabg` Prior coronary artery bypass grafting (CABG) (0/1).

`ppci` Prior percutaneous coronary intervention (PCI) (0/1).

`cva` History of cerebrovascular accident/stroke (0/1).

`tia` History of transient ischemic attack (0/1).

`pvd` Peripheral vascular disease (0/1).

`diabetes` History of diabetes mellitus (0/1).

`insulin` Insulin therapy (0/1).

`htn` History of hypertension (0/1).

`htnrx` Antihypertensive treatment (0/1).

`hichol` History of high cholesterol/hyperlipidemia (0/1).

`smknow` Smoking history/status indicator.

`asthma` History of asthma (0/1).

`copd` History of chronic obstructive pulmonary disease (0/1).

`esrd` End-stage renal disease (0/1).

`lv_dysf` Left ventricular dysfunction (0/1).

`ecgmi` ECG evidence of myocardial infarction (0/1).

`ecglvh` ECG evidence of left ventricular hypertrophy (0/1).

`lbbb` Left bundle branch block (0/1).

`rbbb` Right bundle branch block (0/1).

`reststst` Resting ST-segment abnormality (0/1).

`stnond` Non-diagnostic ST-segment response (0/1).

`stabn` ST-segment abnormality during testing (0/1).

`stabnb` ST-segment abnormality subtype/flag.

`stabnv` ST-segment abnormality subtype/flag.

`rtachy` Tachycardia indicator.

`typcp` Typical chest pain indicator.

`ntangina` Angina history/symptom indicator.

`ttangina` Angina during treadmill test (0/1).

`ttclaud` Claudication during treadmill test (0/1).

`image` Exercise test performed with imaging (0/1).

`acei` Angiotensin-converting enzyme (ACE) inhibitor use (0/1).

`aspirin` Aspirin use (0/1).

`betablok` Beta-blocker use (0/1).

`dilver` Diltiazem/verapamil use (0/1).

`nifed` Nifedipine use (0/1).

`diuretic` Diuretic use (0/1).

`lipidrx` Lipid-lowering therapy use (0/1).

`nitrates` Nitrate therapy use (0/1).

`bdilat` Bronchodilator use (0/1).

`rs_ami` Reason for referral/testing: acute myocardial infarction (0/1).

`rs_mi` Reason for referral/testing: myocardial infarction (0/1).

`rs_ptca` Reason for referral/testing: PTCA/angioplasty (0/1).

`rs_cabg` Reason for referral/testing: CABG (0/1).

`rs_arrth` Reason for referral/testing: arrhythmia (0/1).

`rs_htran` Reason for referral/testing: heart transplant (0/1).

`bpvc_rst` Bigeminal premature ventricular contractions at rest (0/1).

`bpvc_ex` Bigeminal premature ventricular contractions during exercise (0/1).

`bpvc_rec` Bigeminal premature ventricular contractions during recovery (0/1).

`fpvc_rst` Frequent premature ventricular contractions at rest (0/1).

`fpvc_ex` Frequent premature ventricular contractions during exercise (0/1).

`fpvc_rec` Frequent premature ventricular contractions during recovery (0/1).

`nsvt_rst` Non-sustained ventricular tachycardia at rest (0/1).

`nsvt_ex` Non-sustained ventricular tachycardia during exercise (0/1).

`nsvt_rec` Non-sustained ventricular tachycardia during recovery (0/1).

`vtrp_rst` Ventricular triplets at rest (0/1).

`vtrp_ex` Ventricular triplets during exercise (0/1).

`vtrp_rec` Ventricular triplets during recovery (0/1).

`hbm2_rec` Heart rate-related recovery measure at 2 minutes.

`exec` Indicator variable (0/1); coding as in source data.

`aso` Indicator variable (0/1); coding as in source data.

Details

Heart rate recovery (`hrrecov`) is defined as peak heart rate minus the heart rate measured 1 minute into recovery. The survival outcome is all-cause mortality with right-censoring. Unless otherwise noted, indicator variables are coded 0/1 (0 = no/absent, 1 = yes/present).

References

Ishwaran, H., Blackstone, E. H., Pothier, C. E. and Lauer, M. S. (2004). Relative risk forests for exercise heart rate recovery as a predictor of mortality. *Journal of the American Statistical Association*, 99, 591-600.

Examples

```
data(hrrecov)
vp <- varpro(Surv(ttodead, died)~., hrrecov, ntree=50, split.weight=FALSE)
ivp <- ivarpro(vp)
plot(ivp, var = "peak_met", col.var = "fitness", size.var = "peak_hr",
       col.legend.n = 7, smooth.n = 7, x.dist = "auto")
```

---

importance *Calculate and Plot Variable Importance*

Description

Calculates and optionally plots variable importance for fitted `varpro`, `uvarpro`, and `rhf` objects. Larger scores indicate greater importance.

Usage

```
importance(x, ...)

## S3 method for class 'varpro'
importance(x, local.std = TRUE, y.external = NULL,
  cutoff = 0.79, trim = 0.1, plot.it = FALSE, conf = TRUE, sort = TRUE,
  ylab = if (conf) "Importance" else "Standardized Importance",
  max.rules.tree, max.tree,
  ...)

## S3 method for class 'uvarpro'
importance(x, local.std = FALSE, y.external = NULL,
  cutoff = 0.79, trim = 0.1, plot.it = FALSE, conf = TRUE, sort = TRUE,
  ylab = if (conf) "Importance" else "Standardized Importance",
  max.rules.tree, max.tree,
  ...)

## S3 method for class 'rhf'
importance(x, local.std = TRUE, y.external = NULL,
  cutoff = 0.79, trim = 0.1, plot.it = FALSE, conf = TRUE, sort = TRUE,
  ylab = if (conf) "Importance" else "Standardized Importance",
  max.rules.tree, max.tree,
  ...)
```

Arguments

x	A varpro, uvarpro, or rhf object.
local.std	Logical. If TRUE, standardizes each rule comparison before combining the results. If FALSE, divides mean importance by its standard deviation across trees. UVarPro always uses FALSE; RHF always uses TRUE. See Details.
y.external	Optional replacement response, used when local.std = TRUE. Supply values for the same observations, in the same order, as those used in the fitted object. See Details for the required form. Ignored for uvarpro objects.
cutoff	Importance-score threshold used in the plot. Variables with z >= cutoff are highlighted. Also sets the box limits when conf = TRUE. The returned scores are unchanged.
trim	Fraction of values capped at each end of the across-tree importance distribution before calculating summaries (winsorization). Must satisfy 0 <= trim < 0.5. Use trim = 0 for no capping.
plot.it	Logical. If TRUE, plots importance when more than one tree contributes. Classification plots use overall importance. Multivariate regression returns tables only.
conf	Logical. With plot.it = TRUE, uses a box-style display when TRUE, or a bar plot of z when FALSE. See Details for the box limits.

sort	Logical. If TRUE, orders results by decreasing importance. Otherwise, uses the predictor order in the fitted object.
ylab	Label for the plot's vertical axis.
max.rules.tree	Optional maximum number of rules per tree used to calculate importance. Defaults to the fitted object's setting for varpro and uvarpro, or 150 for rhf.
max.tree	Optional maximum number of trees used to calculate importance. Defaults to the fitted object's setting for varpro and uvarpro, or min(150, x$ntree) for rhf.
...	Additional graphical arguments passed to bxp or barplot when plot.it = TRUE.

Details

After fitting a model with varpro, use importance to see which predictors have the strongest relationship with the response. The function combines importance measurements from the forest's rules into a score for each evaluated predictor. Larger scores indicate greater importance, and results are sorted from largest to smallest by default.

Use importance(x) to obtain the scores or importance(x, plot.it = TRUE) to add a plot. Classification also provides scores for each class, and multivariate regression provides a separate summary for each response.

How the scores are calculated: The forest's rules define groups of observations for comparing response values. For regression, the comparisons measure differences in response means; for classification, they compare class proportions. The default, local.std = TRUE, standardizes each comparison using the variability in the observations being compared. For survival, the comparisons use the numeric survival summaries calculated by varpro.

Rule scores are averaged within each tree, weighted by the number of out-of-bag observations in the rule. These are observations not used to grow that tree. The tree scores are then averaged to obtain z.

With local.std = FALSE, the function instead summarizes the stored rule scores within each tree, then divides the mean tree score by the standard deviation across trees. This ratio is returned as z, together with mean and std.

In both cases, trim limits the influence of unusually small or large tree scores. The default, trim = 0.1, replaces values below the 10th percentile or above the 90th percentile by those percentiles before calculating the mean and standard deviation. Set trim = 0 to use the values unchanged.

Plots and variable selection: Set plot.it = TRUE to plot the scores. Variables with z >= cutoff are shown in red. Changing cutoff changes the plot, while the returned importance table continues to include the same predictors and scores.

For a bar plot of z, set conf = FALSE. With conf = TRUE, each box is centered on mean importance and extends by cutoff * std in each direction. Under local standardization, the center is z and std is set to 1. Otherwise, std is the standard deviation across trees after winsorization. The boxes provide an informal visual guide to the cutoff.

To select predictors for a prediction model, use cv.varpro, which compares prediction errors for variable sets formed using different importance cutoffs.

Using another response: Supply y.external with local.std = TRUE to calculate importance for another response using the same forest rules. For example, supplying predictions from another model shows which predictors are important for those predictions.

For a single numeric response, including a survival summary, supply a numeric vector. For multiple numeric responses, supply a matrix with the same dimensions and response-column order as `x$y`. For classification, supply a factor with the same levels and coding as `x$y`. In every case, values must follow the observation order in the fitted object, including any omissions made during fitting.

UVarPro and RHF objects: For `uvarpro` objects, the function summarizes the rule scores from the unsupervised analysis using `local.std = FALSE`. See `uvarpro` for the interpretation of these scores.

For `rhf` objects, importance uses log-transformed integrated hazard estimates and always applies local standardization. Out-of-bag estimates are used when available, and inbag estimates otherwise.

Value

For regression, survival with a single response summary, and RHF, a data frame with predictor names as row names. With `local.std = TRUE`, the table contains only `z`: the mean locally standardized importance across trees after winsorization. With `local.std = FALSE`, including UVarPro analyses, it contains:

`mean` Mean importance across trees after winsorization.

`std` Standard deviation of importance across trees after winsorization.

`z` Standardized importance, `mean / std`.

For classification, a list with two components:

`unconditional` Overall importance table, with the same columns described above.

`conditional.z` Matrix of class-specific importance scores, with predictors in rows and the fitted object's class labels in columns.

For multivariate regression, including survival analyses with multiple response summaries, a list containing one importance table per response, in response-column order.

The summaries cover predictors represented in the evaluated rules. Categorical predictors can appear as several indicator columns. For VarPro or UVarPro objects, `get.orgvimp(x)` gives scores under the original predictor names, using the largest available score for each predictor.

Author(s)

Min Lu and Hemant Ishwaran

References

Lu, M. and Ishwaran, H., (2024). Model-independent variable selection via the rule-based variable priority. arXiv e-prints, pp.arXiv-2409.

See Also

`cv.varpro` `varpro` `uvarpro`

Examples

```
## ------------------------------------------------------------
## toy example - needed to pass CRAN test
## ------------------------------------------------------------

## mtcars regression
o <- varpro(mpg ~ ., mtcars, ntree = 1)
imp <- importance(o, local.std = FALSE)
print(imp)



## ------------------------------------------------------------
## iris example
## ------------------------------------------------------------

## apply varpro to the iris data
o <- varpro(Species ~ ., iris, max.tree = 5)

## print/plot the results
imp <- importance(o, plot.it = TRUE)
print(imp)

## ------------------------------------------------------------
## boston housing: regression
## ------------------------------------------------------------

data(BostonHousing, package = "mlbench")

## call varpro
o <- varpro(medv~., BostonHousing)

## extract importance values
imp <- importance(o)
print(imp)

## plot the results
imp <- importance(o, plot.it = TRUE)
print(imp)



## ------------------------------------------------------------
## illustrates y-external: regression example
## ------------------------------------------------------------

## friedman1 - standard application of varpro
d <- data.frame(mlbench::mlbench.friedman1(250),noise=matrix(runif(250*10,-1,1),250))
o <- varpro(y~.,d)
print(importance(o))
```

```
## importance using external rf predictor
print(importance(o,y.external=randomForestSRC::rfsrc(y~.,d)$predicted.oob))

## importance using external lm predictor
print(importance(o,y.external=lm(y~.,d)$fitted))

## importance using external randomized predictor
print(importance(o,y.external=sample(o$y)))

## ------------------------------------------------------------
## illustrates y-external: classification example
## ------------------------------------------------------------

## iris - standard application of varpro
o <- varpro(Species~.,iris)
print(importance(o))

## importance using  external rf predictor
print(importance(o,y.external=randomForestSRC::rfsrc(Species~.,iris)$class.oob))

## importance using  external randomized predictor
print(importance(o,y.external=sample(o$y)))

## ------------------------------------------------------------
## illustrates y-external: survival
## ------------------------------------------------------------
data(pbc, package = "randomForestSRC")
o <- varpro(Surv(days, status)~., pbc)
print(importance(o))

## importance using  external rsf predictor
print(importance(o,y.external=randomForestSRC::rfsrc(Surv(days, status)~., pbc)$predicted.oob))

## importance using  external randomized predictor
print(importance(o,y.external=sample(o$y)))
```

`isopro` *Identify Anomalous Data*

Description

Use isolation forests to identify rare/anomalous data.

Usage

```
isopro(object,
       method = c("unsupv", "rnd", "auto"),
```

```
sampsize = function(x){min(2^6, .632 * x)},
ntree = 500, nodesize = 1,
formula = NULL, data = NULL, ...)
```

Arguments

object	varpro object returned from a previous call.
method	Isolation forest method. Options are "unsupv" (unsupervised analysis, default), "rnd" (pure random splitting), and "auto" (auto-encoder, a type of multivariate forest).
sampsize	Function or numeric value specifying the sample size used for constructing each tree. Sampling is without replacement.
ntree	Number of trees to grow.
nodesize	Minimum terminal node size.
formula	Formula used for supervised isolation forest. Ignored if object is provided.
data	Data frame used to fit the isolation forest. Ignored if object is provided.
...	Additional arguments passed to rfsrc.

Details

Isolation Forest (Liu et al., 2008) is a random forest-based method for detecting anomalous observations. In its original form, trees are constructed using pure random splits, with each tree built from a small subsample of the data, typically much smaller than the standard 0.632 fraction used in random forests. The idea is that anomalous or rare observations are more likely to be isolated early, requiring fewer splits to reach terminal nodes. Thus, observations with relatively small depth values (i.e., shallow nodes) are considered anomalies.

There are several ways to apply the method:

- The default approach is to supply a formula and data to build a supervised isolation forest. If only data is provided (i.e., no response), an unsupervised analysis is performed. In this case, the method option is used to specify the type of isolation forest (e.g., "unsupv", "rnd", or "auto").
- If both a formula and data are provided, a supervised model is fit. In this case, method is ignored. While less conventional, this approach may be useful in certain applications.
- Alternatively, a varpro object may be supplied, but other configurations are also supported. In this setting, isolation forest is applied to the reduced feature matrix extracted from theobject. This is similar to using the data option alone but with the advantage of prior dimension reduction.

Users are encouraged to experiment with the choice of method, as the original isolation forest ("rnd") performs well in many scenarios but can be improved upon in others. For example, in some cases, "unsupv" or "auto" may yield better detection performance.

In terms of computational cost, "rnd" is the fastest, followed by "unsupv". The slowest is "auto", which is best suited for low-dimensional settings.

Value

Trained isolation forest and anomaly scores.

Author(s)

Min Lu and Hemant Ishwaran

References

Liu, Fei Tony, Kai Ming Ting, and Zhi-Hua Zhou. (2008). Isolation forest. 2008 Eighth IEEE International Conference on Data Mining. IEEE.

Ishwaran H. (2025). Multivariate Statistics: Classical Foundations and Modern Machine Learning, CRC (Chapman and Hall), in press.

See Also

predict.isopro uvarpro varpro

Examples

```
## ------------------------------------------------------------
##
## satellite data: convert some of the classes to "outliers"
## unsupervised isopro analysis
##
## ------------------------------------------------------------

## load data, make three of the classes into outliers
data(Satellite, package = "mlbench")
is.outlier <- is.element(Satellite$classes,
          c("damp grey soil", "cotton crop", "vegetation stubble"))

## remove class labels, make unsupervised data
x <- Satellite[, names(Satellite)[names(Satellite) != "classes"]]

## isopro calls
i.rnd <- isopro(data=x, method = "rnd", sampsize=32)
i.uns <- isopro(data=x, method = "unsupv", sampsize=32)
i.aut <- isopro(data=x, method = "auto", sampsize=32)

## AUC and precision recall (computed using true class label information)
perf <- cbind(get.iso.performance(is.outlier,i.rnd$howbad),
              get.iso.performance(is.outlier,i.uns$howbad),
              get.iso.performance(is.outlier,i.aut$howbad))
colnames(perf) <- c("rnd", "unsupv", "auto")
print(perf)


## ------------------------------------------------------------
```

```
##
## boston housing analysis
## isopro analysis using a previous VarPro (supervised) object
##
## ------------------------------------------------------------

data(BostonHousing, package = "mlbench")

## call varpro first and then isopro
o <- varpro(medv~., BostonHousing)
o.iso <- isopro(o)

## identify data with extreme percentiles
print(BostonHousing[o.iso$howbad <= quantile(o.iso$howbad, .01),])

## ------------------------------------------------------------
##
## boston housing analysis
## supervised isopro analysis - direct call using formula/data
##
## ------------------------------------------------------------

data(BostonHousing, package = "mlbench")

## direct approach uses formula and data options
o.iso <- isopro(formula=medv~., data=BostonHousing)

## identify data with extreme percentiles
print(BostonHousing[o.iso$howbad <= quantile(o.iso$howbad, .01),])


## ------------------------------------------------------------
##
## monte carlo experiment to study different methods
## unsupervised isopro analysis
##
## ------------------------------------------------------------

## monte carlo parameters
nrep <- 25
n <- 1000

## simulation function
twodimsim <- function(n=1000) {
  cluster1 <- data.frame(
    x = rnorm(n, -1, .4),
    y = rnorm(n, -1, .2)
  )
  cluster2 <- data.frame(
    x = rnorm(n, +1, .2),
    y = rnorm(n, +1, .4)
  )
  outlier <- data.frame(
```

```
    x = -1,
    y =  1
  )
  x <- data.frame(rbind(cluster1, cluster2, outlier))
  is.outlier <- c(rep(FALSE, 2 * n), TRUE)
  list(x=x, is.outlier=is.outlier)
}

## monte carlo loop
hbad <- do.call(rbind, lapply(1:nrep, function(b) {
  cat("iteration:", b, "\n")
  ## draw the data
  sim0 <- twodimsim(n)
  x <- sim0$x
  is.outlier <- sim0$is.outlier
  ## iso pro calls
  i.rnd <- isopro(data=x, method = "rnd")
  i.uns <- isopro(data=x, method = "unsupv")
  i.aut <- isopro(data=x, method = "auto")
  ## save results
  c(tail(i.rnd$howbad,1),
    tail(i.uns$howbad,1),
    tail(i.aut$howbad,1))
}))


## compare performance
colnames(hbad) <- c("rnd", "unsupv", "auto")
print(summary(hbad))
boxplot(hbad,col="blue",ylab="outlier percentile value")
```

`ivarpro` *Individual Variable Priority: Case-Specific Local Gradients*

Description

Calculates variable importance for individual training observations. The scores describe the direction and strength of local changes in a model's predictions as each predictor changes. Supports regression, classification, and survival analyses.

Usage

```
ivarpro(object,
        adaptive = TRUE,
        cut = NULL,
        cut.max = 1,
        ncut = 51,
        nmin = 20, nmax = 150,
```

```
        y.external = NULL,
        noise.na = TRUE,
        max.rules.tree = NULL,
        max.tree = NULL,
        use.loo = TRUE,
        use.abs = FALSE,
        path.store.membership = TRUE,
        save.data = TRUE,
        save.model = TRUE,
        scale = c("local", "global", "none"))
```

Arguments

object	A fitted varpro object, or a supervised rfsrc grow object with a single outcome and numeric predictors. Use a varpro object for categorical predictors and multivariate regression.
adaptive	Logical. If TRUE, automatically limits the largest candidate neighborhood according to the sample size. Used when cut is not supplied.
cut	Optional vector of nonnegative neighborhood widths for continuous predictors. Each width gives the maximum distance from the rule's center, in standard deviations of the predictor values among usable observations in the enlarged comparison group described in Details. Supplied values are sorted and duplicates removed; adaptive, cut.max, and ncut are then ignored. Binary 0/1 predictors use a comparison of the two levels.
cut.max	Upper bound on candidate neighborhood width when cut is not supplied. Smaller values restrict the search to narrower neighborhoods.
ncut	Number of candidate widths when cut is not supplied. Must be a positive integer.
nmin	Minimum number of usable observations for a local estimate. Must be an integer of at least two.
nmax	Maximum number of observations used for a local estimate. Must be an integer of at least two. The effective maximum is also limited according to the training sample size and is at least nmin.
y.external	Optional numeric vector or matrix containing the values to explain, such as predictions from another model. Rows must match the training observations used by object, in the same order. Each matrix column supplies a separate target; column names identify the targets. The default uses out-of-bag predictions from the forest in object.
noise.na	Logical. If TRUE, averages available rule estimates and uses NA where none are available. With FALSE, unavailable rule estimates contribute zero to the average, and observations with no applicable rules receive zero.
max.rules.tree	Maximum number of rules examined per tree. Defaults to the setting in a varpro object, or 150 for direct rfsrc input.
max.tree	Maximum number of trees used to obtain rules. Defaults to the setting in a varpro object, or 150 for direct rfsrc input.

`use.loo` Logical. If `TRUE`, chooses the neighborhood for each continuous-predictor fit by leave-one-out prediction error. With `FALSE`, uses the valid candidate with the most observations, subject to `nmin` and `nmax`.

`use.abs` Logical. If `TRUE`, averages the absolute values of the rule-level scores. The default `FALSE` preserves their signs.

`path.store.membership` Logical. If `TRUE`, stores the observation indices for each rule and its additional comparison group. Setting `FALSE` reduces storage without changing the calculated scores or their plots.

`save.data` Logical. If `TRUE`, stores the processed predictors and target values in the result for plotting.

`save.model` Logical. If `TRUE`, stores the supplied model in the result.

`scale` Scaling of the local slopes. `"local"` uses the predictor's spread in the selected neighborhood; `"global"` uses its standard deviation across the training observations; `"none"` gives slopes per unit of the predictor. See Details.

Details

`ivarpro` shows how a predictor's relationship with a model's predictions varies from one observation to another. It returns a separate importance score for each training observation and predictor. The scores summarize local slopes, called gradients, which describe both the direction and strength of the relationship.

Use `x <- ivarpro(object)` to calculate the scores from a fitted model. `print(head(x))` displays the first few observations for a single prediction target. The plotting functions below help explore how the scores vary across the data.

Interpreting the scores: A positive score indicates that larger predictor values locally accompany larger predicted values; a negative score indicates the opposite direction. Larger magnitudes represent stronger local changes on the chosen scale. For example, a negative score for a mortality target indicates a local relationship with lower predicted mortality as the predictor increases.

The default preserves signs when averaging rule-level scores, so positive and negative slopes can cancel. Use `use.abs = TRUE` to summarize strength alone by averaging their absolute values.

Scores are calculated for the predictors in `object$xvar.names`. VarPro converts categorical predictors to numeric 0/1 indicators, with a separate score for each indicator. With the default `noise.na = TRUE`, `NA` marks an unavailable estimate, for example because an observation has no applicable rules or there is too little local predictor variation.

Which values are explained: The values being explained are called the prediction target. By default, `ivarpro` uses out-of-bag (OOB) predictions from the forest in `object`. Regression uses predicted responses, and classification uses predicted class probabilities. Binary classification uses the first probability column; use `attr(x, "target")` to identify it. Multiclass classification returns a named list with one gradient table per class.

A survival forest supplies mortality predictions. When VarPro uses a multivariate regression forest for restricted mean survival time (RMST) targets, each predicted target has its own gradient table. More generally, multivariate VarPro regression returns one table per predicted response.

Supply `y.external` to explain another numeric target, such as predictions from a different model or a particular class probability, using the same forest rules. A matrix supplies one target per

column, including when it has two columns. Its rows must match the observations used in the fitted model, after any omissions during model fitting.

How local scores are calculated: Each forest rule describes a group of observations through restrictions on predictor values. To study one predictor, `ivarpro` removes that predictor's restrictions while keeping those on the other predictors. This step, called releasing the predictor, adds observations to the comparison group.

For a continuous predictor, the method fits a straight line to the target values using observations from the enlarged group that are near the predictor's mean among the rule's OOB members. The fitted slope describes the local change in the target. For a 0/1 predictor, the slope is the difference in mean target values between levels 1 and 0.

After any requested scaling, each rule's score is assigned to its original OOB members: observations satisfying the rule that were not used to grow its tree. Each observation's final score averages the applicable rule scores for that predictor.

Scaling and neighborhood size: With the default `scale = "local"`, each rule's slope is multiplied by the predictor's spread in the selected neighborhood. For continuous predictors, this spread is measured around the rule's OOB mean; for 0/1 predictors, it is the sample standard deviation. With `scale = "global"`, the multiplier is the predictor's standard deviation across the training observations. Both choices express the score in target units. With `scale = "none"`, scores are in target units per predictor unit; for a 0/1 predictor, the slope is the change from level 0 to 1.

Neighborhood size determines which nearby observations contribute to a continuous-predictor slope. By default, candidate neighborhoods are compared by leave-one-out prediction error: each observation is predicted from a line fitted without it, and the neighborhood with the smallest mean squared error is chosen. With `use.loo = FALSE`, the valid candidate containing the most observations is used instead.

`cut.max` limits the candidate widths, and `adaptive` allows the maximum to be reduced according to sample size. Supply `cut` to specify the widths directly. Increasing `nmin` requires more observations for each local fit; `nmax` limits how many are used. Binary predictors use a comparison of both levels, subject to these sample-size controls, rather than a neighborhood search.

Displaying local importance: `shap.ivarpro(x)` compares the distribution of scores across predictors. `plot(x, var = "x1")` shows one predictor's scores against its values. Add `col.var` to color points by another variable and examine how the local relationship varies across observations. See `plot.ivarpro` for interpreting and adjusting these plots.

For a list of target-specific tables, supply `target` by name or index to either plotting function. Without it, the first target is used with a warning. The data stored by `ivarpro` are used automatically. With `save.data = FALSE`, supply the matching processed data through `dat` to `shap.ivarpro` or `data` to `plot`.

New observations: Use `predict(x, newdata = test)` to obtain gradients for new cases from the stored rule estimates. Supply predictors in their original form; the training hot-encoding is reused. Omitting `newdata` recovers out-of-bag training scores. See `predict.ivarpro` for an example.

Value

An object of class `ivarpro`. A single target produces a numeric data frame with one row per training observation used by the model and one column per predictor in `object$xvar.names`. Row names

match the processed training data. Multiple targets produce a named list of these data frames. `print(x)` summarizes a list; `print(x, full = TRUE)` displays all of its tables.

The `target` attribute identifies the target or targets. When requested, the `data` attribute contains the processed predictors and target values, and the `model` attribute contains the supplied model. Target columns in the stored data are named `y` or `y.<target>`, with suffixes added when needed to give each column a unique name. The target-to-column mapping is available through `attr(attr(x, "data"), "response.names")`.

The `ivarpro.path` attribute stores neighborhood settings and diagnostics for the selected rule estimates, such as their slopes and numbers of observations. For multiple targets, shared settings are stored on the list and target-specific diagnostics on each table. Membership indices are included when `path.store.membership = TRUE`. The gradient tables and matching data are sufficient for plotting.

Author(s)

Min Lu and Hemant Ishwaran

References

Lu, M. and Ishwaran, H. (2025). Individual variable priority: a model-independent local gradient method for variable importance. *Artificial Intelligence Review*, 58:407.

See Also

`varpro`, `plot.ivarpro`, `shap.ivarpro`, `predict.ivarpro`

Examples

```
## Survival: local changes in predicted mortality.
library(survival)
data(peakVO2, package = "randomForestSRC")
peak <- na.omit(peakVO2)

## Keep all predictors available for the local plots.
set.seed(137)
vp <- varpro(Surv(ttodead, died) ~ ., peak,
             split.weight = FALSE, ntree = 100,
             parallel = FALSE)
ivp <- ivarpro(vp)
print(head(ivp))
shap.ivarpro(ivp)

## Exercise time colors the local relationship with peak oxygen uptake.
plot(ivp, var = "peak.vo2", col.var = "interval")

## Specify a fixed maximum neighborhood width.
ivp.local <- ivarpro(vp, adaptive = FALSE, cut.max = 0.5)
plot(ivp.local, var = "peak.vo2", col.var = "interval")

## Interaction example: Model 3 of Lu and Ishwaran (2025).
## The signal is 6*x1*x2.
```

```
## Its slopes are 6*x2 for x1 and 6*x1 for x2.
set.seed(137)
n <- 500
X <- matrix(runif(n * 10), nrow = n)
colnames(X) <- paste0("x", seq_len(ncol(X)))
d <- data.frame(y = 6 * X[, 1] * X[, 2] + rnorm(n), X)
vp <- varpro(y ~ ., d, split.weight = FALSE, ntree = 100,
             parallel = FALSE)

set.seed(137)
iv.local <- ivarpro(vp)
set.seed(137)
iv.global <- ivarpro(vp, scale = "global")
set.seed(137)
iv.slope <- ivarpro(vp, scale = "none")

## Compare local magnitudes under the three scaling choices.
scale.summary <- data.frame(
  variable = colnames(iv.local),
  local = colMeans(abs(iv.local), na.rm = TRUE),
  global = colMeans(abs(iv.global), na.rm = TRUE),
  slope = colMeans(abs(iv.slope), na.rm = TRUE))
print(scale.summary)

## Map both slopes over the (x1, x2) plane, as in the paper's
## gradient displays. Each point is a case; color gives its slope.
## The top row shows the true slopes and the bottom row iVarPro estimates.
## scale = "none" puts the estimates in the same units as the derivatives.
slopes <- cbind("True slope for x1" = 6 * d$x2,
                "True slope for x2" = 6 * d$x1,
                "iVarPro slope for x1" = iv.slope$x1,
                "iVarPro slope for x2" = iv.slope$x2)

slope.map <- function(x1, x2, slopes) {
  ## Use one color scale for all four panels. Positive slopes run
  ## from white through orange to red; negative slopes are blue.
  values <- slopes[is.finite(slopes)]
  limit <- max(1, ceiling(max(abs(values))))
  pal <- c(grDevices::colorRampPalette(c("navy", "white"))(101)[-101],
           grDevices::colorRampPalette(c("white", "orange", "red"))(101))
  slope.col <- function(z) pal[1L + round(100 * (z / limit + 1))]

  op <- par(no.readonly = TRUE)
  on.exit(par(op), add = TRUE)
  layout(matrix(c(1, 2, 3, 4, 5, 5), ncol = 2, byrow = TRUE),
         heights = c(1, 1, 0.3))
  par(mar = c(3.5, 3.5, 2, 1), mgp = c(2, 0.6, 0))
  for (j in seq_len(ncol(slopes))) {
    ok <- is.finite(slopes[, j])
    plot(x1, x2, type = "n", xlim = c(0, 1), ylim = c(0, 1),
         xlab = "x1", ylab = "x2", main = colnames(slopes)[j], asp = 1)
    points(x1[ok], x2[ok], pch = 16, cex = 0.7,
           col = slope.col(slopes[ok, j]))
```

```
    points(x1[!ok], x2[!ok], pch = 4, cex = 0.6, col = "grey60")
  }
  par(mar = c(0, 0, 0, 0))
  plot.new()
  at <- seq(-limit, limit, length.out = 7)
  legend("center", legend = format(signif(at, 2), trim = TRUE),
         fill = slope.col(at), ncol = length(at), bty = "n", cex = 0.8,
         title = "Slope (grey crosses: unavailable)")
}
slope.map(d$x1, d$x2, slopes)

## With slopes on the vertical axis, color by the other predictor.
plot(iv.slope, var = "x1", col.var = "x2")
plot(iv.slope, var = "x2", col.var = "x1")

## Explain two supplied targets using the same forest rules.
## Name the columns so targets can be selected explicitly in plots.
targets <- cbind(interaction = 6 * d$x1 * d$x2,
                 linear = 2 * d$x1 - d$x2)
iv.targets <- ivarpro(vp, y.external = targets, scale = "none")
print(iv.targets)
print(head(iv.targets[["linear"]]))
plot(iv.targets, var = "x1", target = "interaction", col.var = "x2")
shap.ivarpro(iv.targets, target = "linear")
```

---

outpro *Out-of-Distribution Scoring with outPro*

Description

outpro identifies observations whose predictor values differ from those in the training data. It returns a distance score for each observation, with larger scores indicating greater differences in the predictors used for scoring. With a varpro model, variable priorities guide predictor selection and weighting. outpro.null supplies a reference distribution for deciding which scores are unusually large.

Usage

```
outpro(object,
       newdata,
       neighbor = NULL,
       distancef = "knn",
       reduce = TRUE,
       cutoff = NULL,
       max.rules.tree = 150,
       max.tree = 150,
       knn.chunk.size = 100L,
       newdata.xscale = FALSE)
```

```
outpro.null(object,
            nulldata = NULL,
            neighbor = NULL,
            distancef = "knn",
            reduce = TRUE,
            cutoff = NULL,
            max.rules.tree = 150,
            max.tree = 150,
            knn.chunk.size = 100L,
            nulldata.xscale = FALSE)
```

Arguments

`object`	A fitted `varpro` object, or an `rfsrc` grow object with classes `c("rfsrc", "grow")`.
`newdata`	Data frame of observations to score. The response is not needed. If omitted, the training observations are scored. For a `varpro` object, supply predictors in their original form; factors are converted to the training model's indicator columns automatically. For an `rfsrc` object, supply the training predictor columns; those selected for distance calculation must be numeric.
`neighbor`	Number of training neighbors used for each observation. With `NULL`, the number is chosen from the training sample size. The value is rounded and limited to the number of available observations. The default distance method chooses the nearest training observations; the other methods use neighbors selected by the forest.
`distancef`	How distances are calculated. The default, `"knn"`, averages weighted Manhattan distances to the nearest training observations. The alternatives `"prod"`, `"euclidean"`, `"mahalanobis"`, `"manhattan"`, `"minkowski"`, and `"kernel"` use neighbors selected by the forest. See Details.
`reduce`	Which predictors to use and how to weight them. Use `TRUE` for automatic selection using varPro variable priorities, `FALSE` for all predictors in the fitted forest with equal weights, a character vector for named predictors with equal weights, or a named numeric vector for predictors and relative weights. For an `rfsrc` object, `TRUE` also uses all predictors with equal weights. See Details.
`cutoff`	Minimum variable-priority z score for automatic selection with a `varpro` object and `reduce = TRUE`. With `NULL`, the cutoff is chosen automatically. If fewer than two predictors meet the cutoff, all predictors in the priority table are used. This cutoff selects predictors; use reference percentiles from `outpro.null` to flag unusually large distances.
`max.rules.tree`	Maximum number of rules per tree used to select forest neighbors. Ignored when `distancef = "knn"`.
`max.tree`	Maximum number of trees used to select forest neighbors. Ignored when `distancef = "knn"`.
`knn.chunk.size`	Positive integer giving the number of observations processed together for `distancef = "knn"`. Smaller values reduce temporary memory use without changing the scores. Ignored by the other distance methods.

`newdata.xscale` Advanced option for `varpro` objects. Leave `FALSE` for data in their original form. Set `TRUE` only when `newdata` already contains the model's encoded predictor columns. Distances still use the training means and standard deviations to standardize values.

`nulldata` Optional data frame of observations considered typical of the population being modeled. Their distances from the training data form the reference distribution. Supply predictors in the same form as `newdata`. With `NULL`, the training observations provide the reference distances.

`nulldata.xscale`
Advanced option for `varpro` objects. Has the same meaning for `nulldata` as `newdata.xscale` has for `newdata`.

Details

outPro helps identify observations that differ from the data used to fit a model. It compares each observation with training observations using a selected set of predictors. With a `varpro` model, the default selection and weights emphasize predictors that are important to the response.

Use `outpro` to calculate distance scores, then `outpro.null` to compare them with scores from a reference sample of typical observations. Large distances relative to this reference identify observations that are unusual in the predictors used for scoring.

Scoring observations: Use `op <- outpro(object, newdata = test)` to score new observations. The vector `op$distance` contains one score per row of `test`, in the same order.

The default method, `distancef = "knn"`, finds the `neighbor` closest training observations and averages their distances. Each distance is a weighted sum of absolute predictor differences, called a weighted Manhattan distance. Predictors are first standardized using their training means and standard deviations, so differences are measured in training standard deviations rather than the predictors' original units.

Smaller values of `neighbor` focus on the closest observations; larger values average over a broader group. To score the training observations, omit `newdata`. The default method then excludes each observation from its own neighbors, leaving at most `n - 1` neighbors for a training sample of size `n`. Supplying the training data explicitly as `newdata` allows each observation to match itself.

Comparing scores with a reference sample: Use `ref <- outpro.null(object, nulldata = reference.data)` to score a separate sample of typical observations. Its distances provide a reference for interpreting new scores. The neighbors still come from the model's training data; `nulldata` supplies the observations whose scores form the reference distribution. Omitting `nulldata` uses the training scores as the reference instead.

For a scoring result `op`, `ref$cdf(op$distance)` gives each observation's reference percentile on a zero-to-one scale. A value of `0.99` means that 99 percent of reference distances are no larger than that observation's distance. For example, `ref$cdf(op$distance) > 0.95` flags observations with reference percentiles above `0.95`.

The support score `1 - ref$cdf(op$distance)` is the fraction of reference distances greater than the scored distance. Smaller values indicate less similarity to the training data relative to the reference sample. Both summaries describe position within the reference distribution.

Use the same model, predictors, weights, `neighbor`, and `distancef` for reference and new observations. For forest-based distances, also use the same `max.tree` and `max.rules.tree`. Recalculate the reference scores when changing these settings. See the note on `"kernel"` below about comparing scores from separate calls.

Choosing predictors and weights: With `reduce = TRUE`, varPro priorities determine which predictors to use and give more weight to higher-priority predictors. The `cutoff` argument controls this selection. Use `reduce = FALSE` to give equal weight to all predictors in the fitted forest. Predictors removed during model fitting are outside this set. For an `rfsrc` object, both `TRUE` and `FALSE` use all forest predictors with equal weights.

Automatic selection treats a factor as one predictor. Its indicator columns present in the fitted forest are then used for scoring, each with the factor's priority as its relative weight.

A character vector, such as `reduce = c("x1", "x2")`, selects named predictors with equal weights. A named numeric vector supplies relative weights. Priorities and supplied weights are squared and normalized to sum to one. For example, `reduce = c(x1 = 2, x2 = 1)` gives distance weights of `0.8` and `0.2`. Supply nonnegative weights, with at least one positive value.

Names supplied in `reduce` must match the fitted forest's predictor columns: `object$rf$xvar.names` for a `varpro` object or `object$xvar.names` for an `rfsrc` object. For factors converted to indicator columns, use those column names. Predictors with zero or nonfinite training standard deviation are omitted. The returned `selected.variables` and `distance.args$weights.used` identify the predictors and weights used in the calculation.

Other distance methods: The other `distancef` options use neighbors selected by the forest rather than the closest observations under the default distance. They also standardize predictors using the training means and standard deviations.

`"prod"` Averages weighted geometric means of absolute predictor differences across neighbors. A small positive constant is added to each difference before multiplication.

`"euclidean"` Averages weighted straight-line (Euclidean) distances to the neighbors.

`"manhattan"` Averages weighted sums of absolute predictor differences. This uses the same distance calculation as `"knn"`, but different neighbors.

`"minkowski"` Averages weighted Minkowski distances of order 4: the fourth root of the weighted sum of fourth powers of absolute predictor differences.

`"mahalanobis"` Averages Mahalanobis-type distances, using the covariance of standardized training predictors to adjust weighted absolute predictor differences.

`"kernel"` Calculates one minus the average Gaussian similarity to the neighbors. Similarity is based on weighted squared Euclidean distances. Its scale is estimated separately for each scoring call, so scores from separate calls can have different scales.

Value

`outpro` returns a list with the following components:

`distance` Numeric vector with one score per observation. Larger values indicate greater differences from the training data in the predictors used for scoring.

`selected.variables` Names of predictors used in scoring, after omitting those with zero or nonfinite training standard deviation.

`selected.weights` Relative predictor weights after rescaling and squaring. These are normalized to sum to one for scoring; the final weights are in `distance.args$weights.used`.

`distance.args` Distance settings, including `distancef` and `weights.used`. For `"knn"`, also includes the number of neighbors used (`knn.neighbor.used`), whether observations were excluded from their own neighbors (`knn.self.excluded`), and `knn.chunk.size`.

`neighbor` Number of neighbors requested, after rounding and limiting it to the training sample size. For KNN training scores, `distance.args$knn.neighbor.used` gives the number used after excluding each observation from its own neighbors.

`cutoff` Variable-priority cutoff. Affects selection only for `varpro` objects with `reduce = TRUE`.

`means, sds` Training means and standard deviations for the predictors used in scoring.

`dropped.zero.sd.variables` Names of predictors omitted because their training standard deviations were zero or nonfinite.

`distance.object` Data used to calculate distances. Includes standardized training and scored predictors (`xorg.scale` and `xnew.scale`), predictor names and relative weights (`xvar.names` and `xvar.wt`), training means and standard deviations, and neighbor information. For forest-based methods, `dist.xvar` contains absolute predictor differences in standardized units; it is `NULL` for `"knn"`.

`score` Information about neighbors selected by the forest. This is `NULL` for `distancef = "knn"`; the outlier scores are in `distance`.

`oob.bits` `0` when `newdata` is omitted and `1` when it is supplied.

`newdata.xscale` Whether supplied data were treated as already converted to the model's encoded predictor columns.

`call` The matched function call.

`outpro.null` returns the same components, together with:

`cdf` Empirical cumulative distribution function of the reference distances. Use `ref$cdf(op$distance)` to obtain reference percentiles for new scores.

`quantile` Reference percentiles for the reference observations themselves, on a zero-to-one scale. Computed as `cdf(distance)`.

See Also

`varpro`, `rfsrc`.

Examples

```
## Simulate a regression problem with two signal and two noise variables.
set.seed(123)
n <- 800
dta <- data.frame(x1 = rnorm(n), x2 = rnorm(n),
                  noise1 = rnorm(n), noise2 = rnorm(n))
dta$y <- 2 * dta$x1 + dta$x2^2 + rnorm(n)
```

```
## Use separate training, reference, and test samples.
vp <- varpro(y ~ ., data = dta[1:500, ])
reference.data <- dta[501:650, ]
test.data <- dta[651:800, ]

## Shift one signal variable in half of the test observations.
shifted <- seq_len(nrow(test.data)) > 75
test.data$x1[shifted] <- test.data$x1[shifted] + 6

## Score test observations using the default KNN method.
op <- outpro(vp, newdata = test.data)
print(op$selected.variables)
print(setNames(op$distance.args$weights.used, op$selected.variables))

## Compare the test distances with an in-distribution reference sample.
ref <- outpro.null(vp, nulldata = reference.data)
percentile <- ref$cdf(op$distance)
support <- 1 - percentile
flag <- percentile > 0.95

print(head(data.frame(distance = op$distance, percentile = percentile,
                      support = support, flag = flag)))
print(tapply(percentile, shifted, median))
print(table(shifted = shifted, flagged = flag))

## Alternatively, use training distances as the reference distribution.
## Omit nulldata so each training observation excludes itself in KNN scoring.
ref.train <- outpro.null(vp)
print(head(ref.train$cdf(op$distance)))

## Use all predictors in the fitted forest with equal weights.
op.all <- outpro(vp, newdata = test.data, reduce = FALSE)

## Specify predictors with equal weights.
op.subspace <- outpro(vp, newdata = test.data,
                      reduce = c("x1", "x2"))

## Specify relative weights. Squaring and normalization give 0.8 and 0.2.
op.weighted <- outpro(vp, newdata = test.data,
                      reduce = c(x1 = 2, x2 = 1))
print(op.weighted$distance.args$weights.used)

## Use forest-derived neighborhoods with a product distance.
op.prod <- outpro(vp, newdata = test.data, distancef = "prod")
print(head(op.prod$distance))

## Recompute the reference distribution when changing scoring settings.
ref.prod <- outpro.null(vp, nulldata = reference.data, distancef = "prod")
print(head(ref.prod$cdf(op.prod$distance)))

## Compare all five methods on the same test observations.
## AUC measures discrimination using raw distances, with larger values
## identifying shifted observations. Tied distances receive average ranks.
```

```
## At a 0.95 reference-percentile cutoff, report the false-positive rate
## (FPR) among unshifted observations and the true-positive rate (TPR)
## among shifted observations, using each reference strategy.
compare.performance <- function(op, ref, ref.train) {
  distance <- op$distance
  n1 <- sum(shifted)
  n0 <- sum(!shifted)
  auc <- (sum(rank(distance, ties.method = "average")[shifted]) -
          n1 * (n1 + 1) / 2) / (n1 * n0)

  flag.reference <- ref$cdf(distance) > 0.95
  flag.training <- ref.train$cdf(distance) > 0.95

  c(AUC = auc,
    FPR.reference = mean(flag.reference[!shifted]),
    TPR.reference = mean(flag.reference[shifted]),
    FPR.training = mean(flag.training[!shifted]),
    TPR.training = mean(flag.training[shifted]))
}

## Each reference distribution uses the same scoring settings as its
## corresponding test scores. Reuse the references already computed.
performance <- rbind(
  KNN.varPro = compare.performance(op, ref, ref.train),
  KNN.all = compare.performance(
    op.all,
    outpro.null(vp, nulldata = reference.data, reduce = FALSE),
    outpro.null(vp, reduce = FALSE)),
  KNN.signal = compare.performance(
    op.subspace,
    outpro.null(vp, nulldata = reference.data,
                reduce = c("x1", "x2")),
    outpro.null(vp, reduce = c("x1", "x2"))),
  KNN.weighted = compare.performance(
    op.weighted,
    outpro.null(vp, nulldata = reference.data,
                reduce = c(x1 = 2, x2 = 1)),
    outpro.null(vp, reduce = c(x1 = 2, x2 = 1))),
  Forest.product = compare.performance(
    op.prod, ref.prod, outpro.null(vp, distancef = "prod"))
)
print(round(performance, 3))
```

---

partialpro *Partial Prediction Profiles for Individual Observations*

Description

Shows how a model's predictions change as one predictor is varied, while the other predictors are held fixed for each observation. Produces individual prediction profiles that can be displayed with `plot`. Optional filtering identifies predictor combinations that are unusual relative to the training data.

Usage

```
partialpro(object, xvar.names, nvar,
           target, learner, newdata, method = c("unsupv", "rnd", "auto"),
           verbose = FALSE, vt.filter = c("isopro", "outpro", "none"),
           ...)
```

Arguments

`object`	A fitted `varpro` object for regression, classification, or survival analysis. Multivariate regression is not supported.
`xvar.names`	Optional character vector of predictors to examine. The default uses the VarPro importance ranking returned by `get.topvars(object)`. Names must match `object$xvar.names`, including indicator-column names for factors. Unavailable names are skipped with a warning.
`nvar`	Optional positive integer limiting the number of predictors examined. Uses the first `nvar` entries of `xvar.names`, or the first `nvar` predictors in the default ranking. The limit is applied before unavailable names are skipped. Use `Inf` for all requested predictors.
`target`	For classification, a class label or an integer probability-column index. The default is the last level of the original response factor. Class labels are matched to named probability columns; unnamed columns must follow the response's class-level order. For regression and survival, the first prediction column is used and `target` is ignored.
`learner`	Optional prediction function. The default uses the forest in `object$rf`. A supplied function must accept a data frame with the predictor columns in `object$x` and return numeric predictions in the same row order. Classification requires one probability column per class. See Details.
`newdata`	Optional data frame of observations for which to calculate profiles. All supplied rows are used. The data must contain the numeric predictor columns stored in `object$x`, including any factor indicator columns. For example, use `newdata = object$x[1:6, , drop = FALSE]` to examine six training observations. Extra columns are ignored. When omitted, up to `nsmp` training observations are sampled separately for each predictor.
`method`	Isolation forest method passed to `isopro`. Choices are `"unsupv"`, `"rnd"`, and `"auto"`. Used only when filtering with `vt.filter = "isopro"`.
`verbose`	Print the name of each predictor as it is processed?
`vt.filter`	Method for identifying unusual predictor combinations: `"isopro"` uses isolation forests, `"outpro"` uses distances from the training data, and `"none"` turns filtering off. See Details.

... Named options controlling the prediction grid, filtering, and curve fitting. See the options below. Unnamed, unrecognized, and duplicate options cause an error.

Details

`partialpro` helps explain a model by showing how its predictions change when one predictor changes. For each observation, it varies that predictor over a grid of values while keeping the other predictors fixed at the observation's values. The resulting predictions are summarized by a curve, called a partial profile. Comparing these curves shows how the model's relationship with a predictor varies across observations.

Use `p <- partialpro(object, nvar = 3)` to calculate profiles for the first three predictors in the VarPro importance ranking, then `plot(p)` to display them. The function uses the model's training observations by default; supply `newdata` to examine particular observations instead.

Interpreting the profiles: A rising profile indicates that larger predictor values give larger predictions when the other predictors are held fixed. Different curve shapes show how this relationship depends on the values of the other predictors.

The prediction scale is set by the learner. The default forest supplies predicted responses for regression and mortality predictions for survival. For classification, the selected class probability is converted to log-odds, so larger values indicate a greater predicted probability of that class. Probabilities are limited to `[0.001, 0.999]` before this conversion to keep log-odds finite.

For continuous predictors, the fitted curves share a common average intercept so their shapes can be compared. The returned `yhat.causal` component instead expresses each curve as changes relative to its value at the first grid point. These are model-based contrasts; a causal interpretation requires appropriate study-design and substantive assumptions.

Predictors and observations: Profiles use the numeric predictor columns stored in `object$x`. A factor can therefore have several profiles, one for each indicator column available in the fitted forest. Each profile varies that column while keeping the others fixed. Supplied `newdata` must use these same columns; original factor columns are not converted automatically.

The grid always comes from the training data, including when `newdata` is supplied. Predictors with few distinct values use those values. For other predictors, the grid uses quantiles within the central training range, controlled by `alpha` and `nvirtual`.

Use `set.seed` before calling `partialpro` for reproducible results. Randomness includes sampling training observations and dividing grid values for the curve-fitting comparisons described below.

Filtering unusual predictor combinations: Changing one predictor can create a combination of values that is uncommon in the training data. The modified observations are called *virtual twins*. Unlimited Virtual Twins (UVT) filtering assesses these combinations before fitting the profiles.

The default `vt.filter = "isopro"` uses an isolation forest fitted to the predictors in the VarPro importance ranking. A virtual twin passes the filter when its `isopro` support score is at least `cut`.

With `vt.filter = "outpro"`, distances are compared with reference distances from `outpro.null`. The support score is `1 - F0(distance)`, where `F0` is the reference empirical distribution function. For example, with `cut = 0.1`, a virtual twin passes when at least ten percent of the reference distances are greater than its distance. The default uses nearest-neighbor distances and training observations for the reference distribution.

Larger values of cut allow fewer virtual twins to pass. Use cut = 0 or vt.filter = "none" to turn off filtering. The two filtering methods use different support scores, so the same cut need not accept the same twins.

Fitting the curves: A polynomial curve is fitted separately for each observation and predictor; the default is quadratic. When filtering is active and enough values are available, the function compares curves fitted with and without filtering. It fits them on part of the grid and compares their prediction errors on the remaining grid values. The unrestricted curve is used only when both errors are available and its standardized mean squared error is smaller by more than mse.tolerance. Otherwise, fitting uses the values that pass the filter.

A fitted curve is evaluated over the full grid. Values rejected by the filter can therefore involve extrapolation. For a predictor with two distinct values, predictions at those values are used directly, and both must pass the filter for a profile to be available. A polynomial that cannot be fitted has missing profile values.

Grid and curve-fitting options: Supply these options through ...:

nsmp Maximum number of training observations sampled for each predictor. Default is 250. Ignored when newdata is supplied.

nvirtual Requested number of grid values for predictors with many distinct values. Default is 100. Repeated quantiles are combined, so the grid may be shorter.

alpha Fraction excluded from each tail when constructing a continuous predictor's grid. Default is 0.025. Predictors with few distinct values use all their values.

df Polynomial degree. Default is 2 for quadratic curves. The value is rounded and is at least one.

nmin Required number of grid values passing the filter for a continuous-predictor profile. Default is 15, reduced to half the number of distinct training values when that number is smaller.

mse.tolerance Required improvement in standardized prediction error before using an unrestricted curve. Default is 0.

Filtering options: Supply these options through ...:

cut Minimum support score for a virtual twin to pass the filter. Default is 0.1; 0 turns filtering off.

sampsize, ntree, nodesize Isolation forest settings used with vt.filter = "isopro". Defaults are function(x) min(2^8, 0.632 * x), 500, and 1, respectively. See isopro.

out.distancef Distance method used with vt.filter = "outpro". Default is "knn". See outpro for other distance methods and their interpretation.

out.neighbor Number of neighbors used by outpro. Default is NULL, which uses the outpro default.

out.reduce Predictors and weights for outpro. Default is NULL, which uses the VarPro-ranked predictors together with the predictor being profiled. A character vector specifies predictor columns; a named numeric vector also supplies relative weights. With either vector, the profiled predictor is added if absent, with relative weight one. Logical values are passed directly to outpro.

out.cutoff Variable-selection cutoff passed to outpro when out.reduce = TRUE. Default is NULL, which uses the outpro default. The virtual-twin support threshold is set by cut.

out.max.rules.tree, out.max.tree Rule and tree limits for forest-based outpro distances. Both default to 150. Ignored for "knn".

`out.knn.chunk.size` Number of virtual observations processed together for nearest-neighbor distances. Default is `100L`. Smaller values reduce temporary memory use.

`out.null` Optional `outpro.null` result, or a named list of results with one entry per profiled predictor. Use reference distances calculated with the same model, predictors, weights, and distance settings as the virtual twins. When omitted, reference distances are calculated from the training observations and reused when settings match.

Using another prediction model: Supply `learner` to explain predictions from another model using the same observations and predictor grids. The function receives only predictor columns and must return one prediction row for every input row, in the same order. For regression and survival, a numeric vector is sufficient; a numeric matrix or data frame is also accepted, using its first column. For classification, supply a numeric probability matrix or data frame with one column per class. Class names allow `target` to select a class regardless of column order; a numeric `target` selects by column position.

The package supplies three learner constructors:

`rf.learner(object, ...)` Fits another random forest, giving equal split weights to the VarPro-ranked predictors and zero weight to the others. Recognized forest arguments can be supplied through `...`; the formula, data, split weights, and performance setting are set by the helper.

`gbm.learner(object, ...)` Fits a gradient-boosted model for a numeric target or a two-class response. Options are `n.trees = 500`, `shrinkage = 0.1`, `interaction.depth = 3`, `cv.folds = 5`, and `n.cores = get.mc.cores()`.

`bart.learner(object, ...)` Fits a Bayesian additive regression tree model for a numeric target. Accepts `mc.cores` for prediction, defaulting to `get.mc.cores()`.

For survival analyses, these helpers fit the numeric target stored in `object$y`, such as mortality or a single restricted mean survival time. This can differ from the default learner, which predicts directly from `object$rf`. The helpers reject unnamed, duplicate, and unrecognized options.

Value

Invisibly, a named list of class `"partialpro"`, with one component per available requested predictor. A component is `NULL` when no profiles can be constructed for that predictor. Otherwise it contains:

`case` Row numbers in `object$x`, or in `newdata` when supplied, for the observations included in the result.

`xorg` The predictor's values in the training data.

`xvirtual` Grid values at which profiles are evaluated.

`goodvt` Matrix indicating which virtual twins pass the filter: `1` for a passing value and `NA` otherwise. All entries are one when filtering is off.

`yhat.par` Matrix of fitted profiles. Continuous predictors use polynomial curves with a common average intercept; two-valued predictors use direct predictions. Classification profiles are on the log-odds scale.

`yhat.nonpar` The same fitted profiles, calculated by centering each fitted curve and adding the common average intercept for continuous predictors. Despite the component name, these also use the polynomial fits.

`yhat.causal` Profile changes relative to the first grid value, calculated separately for each observation.

All matrices have one row per observation in `case` and one column per value in `xvirtual`. Unavailable fits have `NA` profile values. If none of the requested predictors are available, the function returns `NULL`.

Author(s)

Min Lu and Hemant Ishwaran

References

Ishwaran H. (2025). Multivariate Statistics: Classical Foundations and Modern Machine Learning, CRC (Chapman and Hall), in press.

See Also

`varpro`, `isopro`, `outpro`

Examples

```
##------------------------------------------------------------------
##
## Boston housing
##
##------------------------------------------------------------------

library(mlbench)
data(BostonHousing)
oldpar <- par(mfrow=c(2,3))
plot((oo.boston<-partialpro(varpro(medv~.,BostonHousing),nvar=6)))
par(oldpar)

##------------------------------------------------------------------
##
## Boston housing using newdata option
##
##
##------------------------------------------------------------------

library(mlbench)
data(BostonHousing)
o <- varpro(medv~.,BostonHousing)
oldpar <- par(mfrow=c(2,3))
plot(partialpro(o,nvar=3))
## same analysis using outpro/KNN filtering
plot(partialpro(o,nvar=3,vt.filter="outpro"))
## same but using newdata (set to first 6 cases of the training data)
plot(partialpro(o,newdata=o$x[1:6,],nvar=3))
par(oldpar)

##------------------------------------------------------------------
```

```
##
## Boston housing with externally constructed rf learner
##
##------------------------------------------------------------------

## varpro analysis
library(mlbench)
data(BostonHousing)
o <- varpro(medv~.,BostonHousing)

## default partial pro call
pro <- partialpro(o, nvar=3)

## partial pro call using built in rf learner
mypro <- partialpro(o, nvar=3, learner=rf.learner(o))

## compare the two
oldpar <- par(mfrow=c(2,3))
plot(pro)
plot(mypro, ylab="external rf learner")
par(oldpar)

##------------------------------------------------------------------
##
## Boston housing:  tree gradient boosting learner, bart learner
##
##------------------------------------------------------------------

if (library("gbm", logical.return=TRUE) &&
    library("BART", logical.return=TRUE)) {

## varpro analysis
library(parallel)
library(mlbench)
data(BostonHousing)
o <- varpro(medv~.,BostonHousing)

## default partial pro call
pro <- partialpro(o, nvar=3)

## partial pro call using built in gradient boosting learner
mypro <- partialpro(o, nvar=3, learner=gbm.learner(o, n.trees=1000, n.cores=get.mc.cores()))

## partial pro call using built in bart learner
mypro2 <- partialpro(o, nvar=3, learner=bart.learner(o, mc.cores=get.mc.cores()))

## compare the learners
oldpar <- par(mfrow=c(3,3))
plot(pro)
plot(mypro, ylab="external boosting learner")
plot(mypro2, ylab="external bart learner")
par(oldpar)
}
```

```
##------------------------------------------------------------------
##
## peak VO2: use five-year RMST for variable priority
## Profiles below explain the default survival forest's mortality predictions.
##
##------------------------------------------------------------------

data(peakVO2, package = "randomForestSRC")
oldpar <- par(mfrow=c(2,3))
plot((oo.peak<-partialpro(varpro(Surv(ttodead,died)~.,peakVO2,rmst=5),nvar=6)))
par(oldpar)

##------------------------------------------------------------------
##
## veteran data set with celltype as a factor
##
##------------------------------------------------------------------

data(veteran, package = "randomForestSRC")
dta <- veteran
dta$celltype <- factor(dta$celltype)
oldpar <- par(mfrow=c(2,3))
plot((oo.veteran<-partialpro(varpro(Surv(time, status)~., dta), nvar=6)))
par(oldpar)

##------------------------------------------------------------------
##
## iris: classification analysis showing partial effects for all classes
##
##------------------------------------------------------------------

o.iris <- varpro(Species~.,iris)
yl <- paste("log-odds", levels(iris$Species))
oldpar <- par(mfrow=c(3,2))
plot((oo.iris.1 <- partialpro(o.iris, target=1, nvar=2)),ylab=yl[1])
plot((oo.iris.2 <- partialpro(o.iris, target=2, nvar=2)),ylab=yl[2])
plot((oo.iris.3 <- partialpro(o.iris, target=3, nvar=2)),ylab=yl[3])
par(oldpar)

##------------------------------------------------------------------
##
## iowa housing data
##
##------------------------------------------------------------------

## quickly impute the data; log transform the outcome
data(housing, package = "randomForestSRC")
housing <- randomForestSRC::impute(SalePrice~., housing, splitrule="random", nimpute=1)
dta <- data.frame(data.matrix(housing))
dta$y <- log(housing$SalePrice)
dta$SalePrice <- NULL
```

```
## partial effects analysis
o.housing <- varpro(y~., dta, nvar=Inf)
oo.housing <- partialpro(o.housing,nvar=15)
oldpar <- par(mfrow=c(3,5))
plot(oo.housing)
par(oldpar)
```

`plot.ivarpro` *Plot Individual Variable Priority*

Description

Plot the local importance scores calculated by `ivarpro`. Examine how one predictor's scores vary across observations, or compare score distributions across predictors in a summary plot.

Usage

```
## S3 method for class 'ivarpro'
plot(x, var, col.var = NULL, size.var = NULL,
  data = NULL, target = NULL, pch = 16, cex = 0.8,
  cex.range = c(0.5, 2), main = NULL, xlab = NULL,
  ylab = "iVarPro gradient", legend = TRUE, ...)

shap.ivarpro(ivar, dat = NULL, feature_names = NULL,
  max.points = 5000, max.points.per.feature = NULL,
  point.alpha = 1.0, point.size = 0.35, point.pch = 16,
  scale.value = TRUE, style = c("blobby", "jitter"),
  blobby.separation = 3, target = NULL)
```

Arguments

`x`, `ivar`	An `ivarpro` object, or a numeric data frame or matrix with one row per observation and one column per predictor containing the local importance scores.
`var`	Predictor to display, given by its column name or index in the gradient table. For a categorical predictor converted to 0/1 indicators, use an indicator column's name or index.
`col.var`	Name of a column in the plotting data used to color points. Factors and numeric variables with few distinct values give group colors; continuous variables give a color gradient.
`size.var`	Name of a column in the plotting data used to scale point sizes.
`data`, `dat`	Data frame or matrix with the same observations in the same order as the gradient table. By default, uses the data stored by `ivarpro` or `predict.ivarpro`. The plotted predictor columns must be numeric; the stored data include the 0/1 indicators used for categorical predictors.

target	For an object with several targets, the response or class name, or its integer position. If omitted, the first target is used with a warning.
pch, cex	Point symbol and size for the single-predictor plot. Group-specific point styles can override pch.
cex.range	Range of point sizes when size.var is supplied.
main, xlab, ylab	Plot title and axis labels.
legend	Logical. If TRUE, displays the color legend in the single-predictor plot when col.var is supplied.
...	Graphical arguments and optional display controls for plot; see Details.
feature_names	Names for an unnamed gradient matrix in the summary plot. Existing column names take precedence.
max.points	Maximum total number of points displayed in the summary plot. A random subset is plotted when this limit is exceeded.
max.points.per.feature	Optional additional limit on the number of displayed points per predictor.
point.alpha, point.size, point.pch	Point opacity, size, and symbol for the summary plot.
scale.value	Logical. If TRUE, rescales each predictor's displayed values to the range zero to one for coloring. With FALSE, uses a common range across predictors. This affects the colors, not the importance scores.
style	Arrangement of points in the summary plot. "blobby" separates overlapping points to show where scores are concentrated; "jitter" adds random vertical displacement.
blobby.separation	Spacing between points for style = "blobby". Larger values increase separation.

Details

These plots show how the local relationships estimated by ivarpro vary across observations. Use shap.ivarpro(x) for an overview of the predictors, then plot(x, var = "x1") to examine one predictor more closely. Each point represents an observation's local importance score for a predictor.

The scores summarize local slopes, called gradients. With signed scores, positive values indicate that larger predictor values locally accompany larger values of the prediction target; negative values indicate the opposite direction. Distance from zero shows the strength of the local change on the chosen scale. Results calculated with use.abs = TRUE show strength without direction.

Single-predictor plot: plot(x, var = "x1") places values of x1 on the horizontal axis and its local scores on the vertical axis. A line at zero separates positive and negative scores. Add col.var = "x2" to see whether the relationship with x1 differs according to x2. size.var can represent a further variable through point size.

Smooth curves are added by default when enough observations are available: one curve without a color variable, or separate curves for its groups. Continuous color variables are divided into ranges based on quantiles for these curves.

Adjusting the single-predictor plot: Supply these controls through ...:

Smooth curves Use `smooth = FALSE` to omit them, or `smooth.span` to adjust smoothing.

Overlapping points Small horizontal jitter is added by default. Use `jitter = FALSE` to disable it, or `jitter.seed` for reproducible jitter. `col.style` selects `"auto"`, `"solid"`, `"outline"`, or `"binary"` point styling. The automatic style distinguishes categorical color groups using outlined or filled and open symbols.

Predictor distribution Use `x.dist = "rug"` for ticks showing observed predictor values, `x.dist = "hist"` for a histogram, or `x.dist = "density"` for a density curve. Combinations such as `x.dist = c("hist", "rug")` are allowed. These displays use observations with plotted scores.

Zero reference line Use `zero.line = FALSE` to omit it.

Summary plot: `shap.ivarpro(x)` gives each predictor a separate row. Horizontal position shows the score; color shows the observation's predictor value. Predictors are ordered by their mean absolute scores. In the default `"blobby"` display, overlapping points are spread vertically to show where scores are concentrated. This SHAP-style layout displays the local gradients calculated by `ivarpro`.

By default, colors show low to high values separately for each predictor. `scale.value` controls this color scaling. The gradient scale is set by the original `ivarpro` call: for example, `scale = "global"` uses the predictor's training-sample standard deviation, and `scale = "none"` gives slopes per predictor unit. See `ivarpro` for the scaling choices.

Columns with only zero or nonfinite scores are omitted. Each point requires a finite score and predictor value. `max.points` and `max.points.per.feature` limit the display size without changing the stored scores.

Choosing a target and supplying data: For a list of responses or classes, supply the whole object and choose one with `target`, for example `plot(x, var = "x1", target = 2)`. Omitting `target` selects the first element with a warning.

Both functions use stored data by default. When no data are stored, supply the matching predictor values through `data` to `plot` or `dat` to `shap.ivarpro`. Rows must match the gradient table, and column names should match the plotted predictors. For categorical predictors, use the same numeric 0/1 indicator columns used to calculate their scores.

Value

Draws the requested plot. `plot.ivarpro` returns `TRUE` invisibly; `shap.ivarpro` returns `NULL` invisibly.

See Also

`ivarpro`, `predict.ivarpro`

Examples

```
## Compare local mortality gradients and inspect peak oxygen uptake.
library(survival)
data(peakVO2, package = "randomForestSRC")
peak <- na.omit(peakVO2)
set.seed(137)
```

```
vp <- varpro(Surv(ttodead, died) ~ ., peak,
             split.weight = FALSE, ntree = 100,
             parallel = FALSE)
ivp <- ivarpro(vp)
print(head(ivp))
shap.ivarpro(ivp)
plot(ivp, var = "peak.vo2", col.var = "interval")
## scale points to "y" (here equal to mortality)
plot(ivp, var = "peak.vo2", col.var = "interval", size.var = "y",
     x.dist = c("hist", "rug"), jitter = FALSE)
```

`plot.partialpro` *Plot Partial Profiles from a VarPro Analysis*

Description

Display partial profiles from a `partialpro` analysis. Examine how a model's predictions change with a predictor, compare groups of observations, or show changes relative to a baseline value.

Usage

```
## S3 method for class 'partialpro'
plot(x, xvar.names, nvar,
  parametric = FALSE, se = TRUE,
  causal = FALSE, subset = NULL, plot.it = TRUE, ...)
```

Arguments

`x`	An object returned by `partialpro`.
`xvar.names`	Predictor names or positive integer positions in `x` to plot. Defaults to all predictors in `x`. Unavailable names are skipped with a warning.
`nvar`	Optional nonnegative integer limiting how many predictors to plot. Applies after choosing `xvar.names`. The default, or `Inf`, uses all selected predictors.
`parametric`	Average the stored polynomial profiles directly? The default `FALSE` smooths the average profile using loess. Ignored when `causal = TRUE` and for binary predictors.
`se`	Display approximate uncertainty guides? Continuous curves have bands at plus and minus two standard errors. Binary predictors use box-style displays based on the same limits. See Details.
`causal`	Display changes relative to each profile's value at the first grid point? This takes precedence over `parametric`. See Details for interpretation.
`subset`	Optional factor for comparing groups, or a logical or positive integer vector selecting observations. Applies to both values of `parametric` and to `causal = TRUE`. See Details for matching observations.
`plot.it`	Draw the plots? Set `FALSE` to return the curves and plotting summaries without drawing them.

... Additional graphical arguments, such as ylab, xlab, ylim, and lwd. Axis labels may be vectors, and ylim may be a list, with entries corresponding to the selected predictors.

Details

plot.partialpro summarizes the partial profiles calculated by partialpro. Each profile describes how a model's predictions change as one predictor varies while the other predictors stay fixed at an observation's values. The plots show the average relationship with that predictor and can compare this relationship across groups of observations.

After computing pro <- partialpro(model), use plot(pro) to display the profiles. Supply xvar.names to choose predictors or nvar to limit how many are shown. The plotting function summarizes the stored results; predictor selection, profile fitting, and filtering of unsupported predictor combinations are controlled in the original partialpro call.

Interpreting the plots: For a continuous predictor, the horizontal axis shows the grid of predictor values and the vertical axis shows the average partial profile. By default, this average is smoothed using loess, with greater weight given to grid values supported by more observations. In the ungrouped display, the solid black line shows this curve. Set parametric = TRUE to average the stored polynomial profiles directly, without this additional smoothing.

Binary predictors have a box-style display at each of their two values. The center shows the average profile value. These boxes summarize the mean and its uncertainty guides, rather than the distribution of individual observations.

For regression and survival, the stored profiles use the learner's prediction scale. For classification, they use the log-odds scale for the class selected in partialpro.

Uncertainty guides: With se = TRUE, continuous curves have dotted lines at plus and minus two standard errors. The lines are red in the ungrouped display and match the group colors in grouped displays. Binary boxes are constructed from the mean and these two limits; their whiskers mark the limits. Set se = FALSE to omit the guides.

For continuous predictors, the standard error uses the variation across stored profiles and the number of observations whose virtual twins pass the filter at each grid value. For binary predictors, it uses the number of available profile values. These are informal summaries of variation across observations, not confidence intervals accounting for model fitting and variable selection. A standard error is unavailable when fewer than two profile values can be used; the mean can still be displayed.

Comparing groups or selecting observations: Supply a factor through subset to compare its groups, for example plot(pro, subset = group). Its levels define the groups shown in the plots. A logical vector selects rows marked TRUE; a vector of positive integers selects those row positions. Grouping and row selection work with both smoothed and polynomial summaries.

Match subset to the observations used to create the profiles. When partialpro sampled training observations, use the row order of the fitted VarPro object's x data. When partialpro used newdata, use the row order of that data instead. Supply the factor or logical vector for that complete set of observations, not just the sampled profile rows. Each predictor's stored case indices identify which rows contribute to its summary. Missing group labels and empty groups are omitted.

Changes relative to a baseline: Set `causal = TRUE` to summarize changes from a common predictor baseline. For each observation, `partialpro` subtracts the profile value at the first grid point, `xvirtual[1]`, from the values at all grid points. The plotting function averages these contrasts and applies the default smoothing for continuous predictors. Positive contrasts indicate an increase from the baseline; negative contrasts indicate a decrease.

The baseline is the first grid value, which need not be zero or the observation's own predictor value. These are model-based contrasts; a causal interpretation requires additional assumptions about the study and the relationships between variables. For an ungrouped binary display, only the contrast at the second value is drawn; grouped binary displays also show the zero baseline.

Obtaining the plotting summaries: Use `plot(pro, plot.it = FALSE)` to obtain the plotted values without drawing. There is one entry per predictor, or per predictor and group when `subset` is a factor. Each entry contains the predictor values, mean profile, and standard errors. These summaries can be used for inspection or custom plots.

Value

With `plot.it = TRUE`, produces the requested plots and returns `NULL`.

With `plot.it = FALSE`, returns a named list of summaries. Ungrouped entries are named by predictor; grouped entries combine the predictor and group names. Each entry contains:

`x` Predictor grid values used for the summary.

`y` Mean profile values, after smoothing when applicable.

`y.se` Approximate standard errors at the corresponding grid values. Unavailable standard errors are `NA`. With `se = FALSE`, this component is zero.

Predictors and groups with no usable summary are omitted. Returns `NULL` when no summaries are available. Binary contrast summaries include both grid values even when the ungrouped plot shows only the second.

Author(s)

Min Lu and Hemant Ishwaran

References

Ishwaran H. (2025). Multivariate Statistics: Classical Foundations and Modern Machine Learning, CRC (Chapman and Hall), in press.

See Also

`partialpro`

Examples

```
##------------------------------------------------------------------
##
## Boston housing
##
##------------------------------------------------------------------
```

```
library(mlbench)
data(BostonHousing)
o.boston <- varpro(medv~., BostonHousing)
oo.boston <- partialpro(o.boston, nvar = 4, learner = rf.learner(o.boston))

oldpar <- par(mfrow = c(2, 4))

## Polynomial summary of the partial profiles
plot(oo.boston, parametric = TRUE, ylab = "parametric est.")

## Smoothed partial-profile summary (default)
plot(oo.boston, parametric = FALSE, ylab = "non-parametric est.")

par(oldpar)

##------------------------------------------------------------------
##
## Boston housing with subsetting
##
##------------------------------------------------------------------

library(mlbench)
data(BostonHousing)
o.boston <- varpro(medv~., BostonHousing)
oo.boston <- partialpro(o.boston, nvar = 3, learner = rf.learner(o.boston))

## Compare profiles for lower- and higher-priced homes
price <- BostonHousing$medv
pricef <- factor(price > median(price), labels = c("low priced", "high priced"))
oldpar <- par(mfrow = c(1, 1))
plot(oo.boston, subset = pricef, nvar = 1)
par(oldpar)

##------------------------------------------------------------------
##
## veteran data with subsetting using celltype as a factor
##
##------------------------------------------------------------------

data(veteran, package = "randomForestSRC")
dta <- veteran
dta$celltype <- factor(dta$celltype)
o.vet <- varpro(Surv(time, status) ~ ., dta)
oo.vet <- partialpro(o.vet, nvar = 6, nsmp = Inf, learner = rf.learner(o.vet))

## Compare partial profiles across cell types
oldpar <- par(mfrow = c(2, 3))
plot(oo.vet, subset = dta$celltype)
par(oldpar)

## Compare changes from the first grid value across cell types
oldpar <- par(mfrow = c(2, 3))
```

```
plot(oo.vet, subset = dta$celltype, causal = TRUE)
par(oldpar)

## Inspect the plotting summaries without drawing
obj <- plot(oo.vet, subset = dta$celltype, plot.it = FALSE)
str(obj, max.level = 1)
```

`predict.isopro` *Prediction for Isopro for Identifying Anomalous Data*

Description

Use isolation forests to identify rare/anomalous values using test data.

Usage

```
## S3 method for class 'isopro'
predict(object, newdata, quantiles = TRUE, ...)
```

Arguments

`object`	`isopro` object returned from a previous call.
`newdata`	Optional test data. If not provided, the training data is used.
`quantiles`	Logical. If `TRUE` (default), returns quantile values; if `FALSE`, returns case depth values.
`...`	Additional arguments passed to internal methods.

Details

Uses a previously constructed `isopro` object to assess anomalous observations in the test data. By default, returns quantile values representing the depth of each test observation relative to the original training data. Smaller values indicate greater outlyingness.

To return raw depth values instead of quantiles, set `quantiles = FALSE`.

Value

Anomaly scores for the test data (or training data).

Author(s)

Min Lu and Hemant Ishwaran

References

Liu, Fei Tony, Kai Ming Ting, and Zhi-Hua Zhou. (2008). Isolation forest. 2008 Eighth IEEE International Conference on Data Mining. IEEE.

Ishwaran H. (2025). Multivariate Statistics: Classical Foundations and Modern Machine Learning, CRC (Chapman and Hall), in press.

See Also

isopro uvarpro varpro

Examples

```
## ------------------------------------------------------------
##
## boston housing
## unsupervised isopro analysis
##
## ------------------------------------------------------------

## training
data(BostonHousing, package = "mlbench")
o <- isopro(data=BostonHousing)

## make fake data
fake <- do.call(rbind, lapply(1:nrow(BostonHousing), function(i) {
  fakei <- BostonHousing[i,]
  fakei$lstat <- quantile(BostonHousing$lstat, .99)
  fakei$nox <- quantile(BostonHousing$nox, .99)
  fakei
}))

## compare depth values for fake data to training data
depth.fake <- predict(o, fake)
depth.train <- predict(o)
depth.data <- rbind(data.frame(whichdata="fake", depth=depth.fake),
                    data.frame(whichdata="train", depth=depth.train))
boxplot(depth~whichdata, depth.data, xlab="data", ylab="depth quantiles")


## ------------------------------------------------------------
##
## boston housing
## isopro supervised analysis with different split rules
##
## ------------------------------------------------------------

data(BostonHousing, package="mlbench")

## supervised isopro analysis using different splitrules
o <- isopro(formula=medv~.,data=BostonHousing)
o.hvwt <- isopro(formula=medv~.,data=BostonHousing,splitrule="mse.hvwt")
```

```
o.unwt <- isopro(formula=medv~.,data=BostonHousing,splitrule="mse.unwt")

## make fake data
fake <- do.call(rbind, lapply(1:nrow(BostonHousing), function(i) {
  fakei <- BostonHousing[i,]
  fakei$lstat <- quantile(BostonHousing$lstat, .99)
  fakei$nox <- quantile(BostonHousing$nox, .99)
  fakei
}))

## compare depth values for fake data to training data
depth.train <- predict(o)
depth.hvwt.train <- predict(o.hvwt)
depth.unwt.train <- predict(o.unwt)
depth.fake <- predict(o, fake)
depth.hvwt.fake <- predict(o.hvwt, fake)
depth.unwt.fake <- predict(o.unwt, fake)
depth.data <- rbind(data.frame(whichdata="fake", depth=depth.fake),
                    data.frame(whichdata="fake.hvwt", depth=depth.hvwt.fake),
                    data.frame(whichdata="fake.unwt", depth=depth.unwt.fake),
                    data.frame(whichdata="train", depth=depth.train),
                    data.frame(whichdata="train.hvwt", depth=depth.hvwt.train),
                    data.frame(whichdata="train.unwt", depth=depth.unwt.train))
boxplot(depth~whichdata, depth.data, xlab="data", ylab="depth quantiles")
```

`predict.ivarpro` *Predict Individual Variable Priority for New Cases*

Description

Apply an iVarPro analysis to new observations, or recover its out-of-bag training scores. The result gives a local gradient for each case and processed predictor.

Usage

```
## S3 method for class 'ivarpro'
predict(object, newdata = NULL, model = NULL,
  noise.na = NULL, path.store.membership = FALSE,
  save.data = TRUE, ...)
```

Arguments

`object`	An object returned by `ivarpro`.
`newdata`	A data frame containing the predictors for new cases. Outcome columns are optional. For a VarPro model, supply the original predictor columns; the training hot-encoding is applied automatically. If `NULL`, recover the original training scores using out-of-bag rule membership.

model	The VarPro object or random forest used to obtain object. By default, use the model stored by ivarpro(save.model = TRUE). Supply it here if it was not stored.
noise.na	How to average unavailable rule scores: TRUE omits them and returns NA where no usable score is available; FALSE includes them as zeros. By default, inherit the setting from object.
path.store.membership	Retain the case-to-rule membership indices with the result? These are optional diagnostic information.
save.data	Store the processed predictor data for subsequent plots? Default is TRUE.
...	Additional arguments passed to the forest prediction method when memberships are obtained.

Details

New cases are routed through the original forest. For each predictor, their scores average the stored gradients of applicable rules that release that predictor. The local slopes and neighborhood choices remain those learned in the original iVarPro analysis.

Scores retain the target, scaling and signed or absolute convention chosen in ivarpro. Multiple targets produce a named list of score tables. Use plot or shap.ivarpro to inspect the result, with target selecting a response or class when needed.

Omitting newdata recovers training scores using out-of-bag membership. Passing the training predictors explicitly treats those observations as new cases and uses all applicable forest rules, so the two calls can give different scores.

Value

An object of class ivarpro, with one row per prediction case and one column per processed predictor. A single target produces a data frame; multiple targets produce a named list of data frames. Row names identify the prediction cases.

The result retains the target labels, original model and rule information needed for further prediction. With save.data = TRUE, its data attribute contains aligned plotting data. For new cases this contains processed predictors; the original explanation targets are not recalculated. See ivarpro for score interpretation.

See Also

ivarpro, plot.ivarpro, shap.ivarpro

Examples

```
## Train on one sample and explain new observations.
set.seed(137)
n <- 400
sim <- data.frame(x1 = runif(n, -1, 1), x2 = runif(n, -1, 1))
sim$y <- 6 * sim$x1 * sim$x2 + rnorm(n, sd = 0.3)
trn <- sample.int(n, 300)
vp <- varpro(y ~ ., sim[trn, , drop = FALSE],
```

```
             split.weight = FALSE, ntree = 100,
             parallel = FALSE)
iv <- ivarpro(vp, scale = "none", nmin = 10)

## Outcome values are not needed for new-case gradients.
test <- sim[-trn, c("x1", "x2"), drop = FALSE]
iv.test <- predict(iv, newdata = test)
print(head(iv.test))
plot(iv.test, var = "x1", col.var = "x2")

## Recover the original out-of-bag training scores.
iv.oob <- predict(iv)
print(all.equal(as.matrix(iv.oob), as.matrix(iv),
                check.attributes = FALSE))
```

`predict.uvarpro` *Prediction on Test Data using Unsupervised VarPro*

Description

Obtain predicted values on test data for unsupervised forests.

Usage

```
## S3 method for class 'uvarpro'
predict(object, newdata, ...)
```

Arguments

`object`	Unsupervised VarPro object from a previous call to `uvarpro`. Only applies if `method = "auto"` was used.
`newdata`	Optional test data. If not provided, the training data is used.
`...`	Additional arguments passed to internal methods.

Details

Applies to unsupervised VarPro objects built using the autoencoder (`method = "auto"`). The object contains a multivariate random forest used to generate predictions for the test data.

Value

Returns a matrix of predicted values, where each column corresponds to a feature (with one-hot encoding applied). The result includes the following attributes:

1. `mse`: Standardized mean squared error averaged across features.
2. `mse.all`: Standardized mean squared error for each individual feature.

Author(s)

Min Lu and Hemant Ishwaran

See Also

uvarpro

Examples

```
## ------------------------------------------------------------
##
## boston housing
## obtain predicted values for the training data
##
## ------------------------------------------------------------

## unsupervised varpro on boston housing
data(BostonHousing, package = "mlbench")
o <- uvarpro(data=BostonHousing)

## predicted values for the training features
print(head(predict(o)))

## ------------------------------------------------------------
##
## mtcars
## obtain predicted values for test data
## also illustrates hot-encoding working on test data
##
## ------------------------------------------------------------

## mtcars with some factors
d <- data.frame(mpg=mtcars$mpg,lapply(mtcars[, c("cyl", "vs", "carb")], as.factor))

## training
o <- uvarpro(d[1:20,])

## predicted values on test data
print(predict(o, d[-(1:20),]))

## predicted values on bad test data with strange factor values
dbad <- d[-(1:20),]
dbad$carb <- as.character(dbad$carb)
dbad$carb <-  sample(LETTERS, size = nrow(dbad))
print(predict(o, dbad))
```

`predict.varpro` *Prediction on Test Data using VarPro*

Description

Obtain predicted values on test data for VarPro object.

Usage

```
## S3 method for class 'varpro'
predict(object, newdata, ...)
```

Arguments

`object`	VarPro object returned from a previous call to `varpro`.
`newdata`	Optional test data. If not provided, predictions are computed using the training data (out-of-bag).
`...`	Additional arguments passed to internal methods.

Details

VarPro uses rules extracted from a random forest built using guided tree-splitting, where variables are selected based on split-weights computed in a preprocessing step.

Value

Returns predicted values for the input data. If `newdata` is provided, predictions are made on that data; otherwise, out-of-bag predictions for the training data are returned.

Author(s)

Min Lu and Hemant Ishwaran

References

Lu, M. and Ishwaran, H. (2024). Model-independent variable selection via the rule-based variable priority. arXiv e-prints, pp.arXiv-2409.

See Also

`varpro`

Examples

```
## ------------------------------------------------------------
## toy example - needed to pass CRAN test
## ------------------------------------------------------------

## train call
o <- varpro(mpg~., mtcars[1:20,], ntree = 1)

## predict call
print(predict(o, mtcars[-(1:20),]))



## ------------------------------------------------------------
##
## boston housing regression
## obtain predicted values for the training data
##
## ------------------------------------------------------------

## varpro applied to boston housing data
data(BostonHousing, package = "mlbench")
o <- varpro(medv~., BostonHousing)

## predicted values for the training features
print(head(predict(o)))

## ------------------------------------------------------------
##
## iris classification
## obtain predicted values for test data
##
## ------------------------------------------------------------

## varpro applied to iris data
trn <- sample(1:nrow(iris), size = 100, replace = FALSE)
o <- varpro(Species~., iris[trn,])

## predicted values on test data
print(data.frame(Species=iris[-trn, "Species"], predict(o, iris[-trn,])))

## ------------------------------------------------------------
##
## mtcars regression: illustration of hot-encoding on test data
##
## ------------------------------------------------------------

## mtcars with some factors
d <- data.frame(mpg=mtcars$mpg,lapply(mtcars[, c("cyl", "vs", "carb")], as.factor))

## varpro on training data
o <- varpro(mpg~., d[1:20,])
```

```
## predicted values on test data
print(predict(o, d[-(1:20),]))

## predicted values on bad test data with strange factor values
dbad <- d[-(1:20),]
dbad$carb <- as.character(dbad$carb)
dbad$carb <-  sample(LETTERS, size = nrow(dbad))
print(predict(o, dbad))
```

---

uvarpro *Unsupervised Variable Selection and Dependence Graphs using UVarPro*

Description

Identifies variables that help explain dependence in data without a specified outcome. uvarpro forms local comparisons using forest rules. get.beta.entropy summarizes which variables help distinguish the groups in these comparisons. sdependent displays the resulting relationships as a graph and highlights candidate signal variables.

Usage

```
uvarpro(data,
        method = c("auto", "unsupv", "rnd"),
        ntree = 200, nodesize = NULL,
        max.rules.tree = 20, max.tree = 200,
        verbose = FALSE, seed = NULL,
        ...)

get.beta.entropy(o,
                 second.stage = FALSE,
                 pre.filter = TRUE,
                 papply = mclapply,
                 vimp.min = 0,
                 nfolds = 10,
                 maxit = 2500,
                 thresh = 1e-3,
                 parallel = FALSE,
                 use.cv = TRUE,
                 lambda.sel = c("lambda.1se", "lambda.min"),
                 nlambda = NULL,
                 lambda.min.ratio = NULL,
                 nlambda.lasso = nlambda,
                 lambda.min.ratio.lasso = lambda.min.ratio)
```

```
sdependent(I,
           threshold = .25,
           layout = "grid",
           q.signal = .75,
           directed = TRUE,
           min.degree = NULL,
           title = "s-Dependent Variable Detection",
           plot = TRUE)
```

Arguments

data	Data frame containing the features to be analyzed, without an outcome or class-label column. Rows with missing values are omitted, unused factor levels are dropped, and categorical features are hot-encoded. See Details.
method	Method for growing the forest: "auto" learns to reproduce the supplied variables, "unsupv" uses unsupervised splitting, and "rnd" uses random splitting. See Details.
ntree	Number of trees to grow.
nodesize	Minimum terminal node size. Larger values give broader groups of observations. The default is chosen from the number of complete observations.
max.rules.tree	Maximum number of rules sampled per tree for the local comparisons.
max.tree	Maximum number of trees used to extract rules.
verbose	Currently unused.
seed	Optional seed passed to the forest calculation. Use set.seed before uvarpro to also control R-level sampling, and seed the lasso analysis separately; see Note.
...	Additional named forest arguments recognized by rfsrc, such as mtry or ytry, and a custom scoring function supplied as entropy; see Details. Supply lasso controls to get.beta.entropy.
o	An object returned by uvarpro, containing the processed data in x and the group membership lists in entropy.
second.stage	If FALSE, return the mean absolute logistic-lasso coefficient matrix. If TRUE, perform an additional local linear lasso using the predictors selected by the logistic stage, and return its mean absolute coefficients divided by the standard deviation of the released variable. See Details.
pre.filter	Should preliminary importance from uvarpro restrict the lasso analysis to variables with get.vimp(o, pretty = FALSE) > vimp.min? The filter applies to both predictors and release variables. It is skipped when preliminary importance is unavailable.
papply	An lapply-like function for processing released variables. Use lapply for serial execution. The default mclapply uses separate R worker sessions (a PSOCK cluster) when getOption("mc.cores", 1L) is greater than one, and serial execution otherwise. The parallel argument controls parallelism across cross-validation folds.
vimp.min	Strict lower cutoff for preliminary importance when pre.filter = TRUE. The default retains values greater than zero. Ignored when pre.filter = FALSE.

`nfolds` Number of cross-validation folds for each local logistic-lasso analysis. At least three folds are required. Ignored when `use.cv = FALSE`.

`maxit` Maximum number of iterations passed to `glmnet` in both lasso stages.

`thresh` Convergence tolerance passed to `glmnet`.

`parallel` Should `cv.glmnet` process its folds in parallel? A suitable `foreach` backend must be registered. Ignored when `use.cv = FALSE`. Account for available resources when enabling both levels of parallelism.

`use.cv` Should the logistic-lasso penalty be chosen by cross-validation? If `FALSE`, coefficients are taken at the last, smallest penalty on the computed `glmnet` path.

`lambda.sel` Cross-validation penalty choice for the logistic stage. `"lambda.1se"` selects the largest penalty whose cross-validation error is within one standard error of the minimum; `"lambda.min"` selects the penalty minimizing that error. Ignored when `use.cv = FALSE`.

`nlambda` Number of penalty values requested for the logistic stage. `NULL` leaves the `glmnet` default unchanged. The computed path can end before all requested values are reached.

`lambda.min.ratio`
Smallest requested logistic-stage penalty relative to the largest penalty on the path. `NULL` uses the `glmnet` default.

`nlambda.lasso`, `lambda.min.ratio.lasso`
Corresponding path controls for the optional second-stage linear lasso. They default to the logistic-stage settings and are used only when `second.stage = TRUE`. The second stage uses the last computed penalty.

`I` A finite, nonnegative numeric importance matrix, usually obtained from `get.beta.entropy`, with released variables as rows and predictors as columns. Supply unique, nonempty row and column names. Row names must belong to the column names and may occur in any order. Rows are aligned by name; missing release rows are filled with zeros.

`threshold` Positive edge cutoff on the scale of `I`. An off-diagonal entry greater than or equal to this value creates an edge. Increasing the cutoff removes edges; decreasing it can recover weaker connections.

`layout` Graph layout. Supported names are `"fr"`, `"dh"`, `"gem"`, `"kk"`, `"lgl"`, `"mds"`, `"sugiyama"`, `"graphopt"`, `"nicely"`, `"random"`, `"sphere"`, `"grid"`, `"circle"`, `"star"`, `"tree"`, and `"bipartite"`; unambiguous abbreviations are accepted. Layout-specific requirements of **igraph** still apply. Alternatively, supply a numeric coordinate matrix with one row per nonisolated vertex, in graph vertex order.

`q.signal` Quantile of the global importance scores among nonisolated vertices used for signal designation. Values range from zero to one; the default is the upper quartile. A signal must meet both this cutoff and the degree condition.

`directed` Should the graph be directed? In the directed graph, an edge goes from the released variable to the predictor used to distinguish its local comparison groups. For an undirected display, supply a symmetric importance matrix; see Note.

`min.degree` Minimum degree for candidate signal designation. With directed graphs this is the out-degree. The default is 1 for directed graphs and 2 for undirected graphs.

`title`	Title for the graph.
`plot`	Should the graph be plotted? If `FALSE`, graph construction and signal selection are still performed. The **igraph** package is required in either case.

Details

UVarPro helps identify variables that explain dependence in a data set without requiring a response. Every supplied column is treated as a variable to be studied. The results can be inspected as an importance matrix, summarized as a ranking, or displayed as a graph.

The main workflow uses three functions:

1. `o <- uvarpro(data)` uses a forest to define groups of observations for local comparisons.
2. `I <- get.beta.entropy(o, papply = lapply)` estimates which variables distinguish those groups and returns an importance matrix.
3. `sdependent(I)` displays the relationships as a graph and identifies candidate signal variables.

Inspect the matrix before plotting; `get.beta.entropy` can return `NULL` when no usable coefficient estimates are found. `importance(o)` gives a separate, preliminary summary used for optional screening. It is not the lasso importance matrix used to construct the graph.

Understanding a local comparison: A forest rule defines a group through conditions on variable values. For example, a rule might require $x_1 \leq 0.5$ and $x_2 > 0.3$. Removing the condition on x_1, while keeping the other condition, is called *releasing* x_1. The newly admitted observations form the *near-miss*, or complementary, group.

For a rule region R and released variable s, write the enlarged region as $R^{(s)}$. The near-miss set is

$$C_s = R^{(s)} \setminus R.$$

Original-region observations are out-of-bag for the corresponding tree. Both groups must be nonempty for a comparison to be used.

`get.beta.entropy` assigns class 1 to the original group and class 0 to the near-miss group. It then uses the other variables to predict group membership with logistic lasso regression. The released variable is excluded because its rule condition already separates the groups. Variables that help distinguish the groups provide evidence of local dependence. Repeating this calculation across rules allows the relationships to differ across the data.

Interpreting the importance matrix: In the matrix returned by `get.beta.entropy`, rows identify released variables and columns identify predictors. For example, `I["x1", "x2"]` measures the average coefficient magnitude for `x2` when distinguishing groups formed by releasing `x1`. The reverse entry answers a different question, so the matrix is generally asymmetric.

Each entry is the mean absolute logistic-lasso coefficient across the usable comparisons for that row. The diagonal is zero because a variable is excluded from its own release comparison. Predictors not selected in a usable comparison contribute zero. Thus

$$I_{sj} = \frac{1}{|\mathcal{K}_s|} \sum_{r \in \mathcal{K}_s} |\widehat{\beta}_{srj}|, \qquad j \neq s, \qquad I_{ss} = 0,$$

where $\mathcal{K}_s$ contains the usable comparisons for released variable s.

A comparison is usable when fitting succeeds and at least one finite, nonzero predictor coefficient is obtained. Failed fits and intercept-only fits are excluded. Comparisons receive equal weight. A row is omitted when none of its comparisons is usable, so the matrix can have fewer rows than columns.

`colSums(I)` summarizes each predictor's contributions across all release variables. Larger sums indicate larger total contributions; column means give the same ranking.

Displaying relationships and selecting candidate signals: `sdependent(I)` draws an arrow from variable `s` to variable `j` when `I[s, j] >= threshold`. The arrow means that `j` helps distinguish the groups formed by releasing `s`. It describes a predictive relationship, not a causal direction. Arrows in both directions and cycles are allowed.

Increase `threshold` to show fewer connections or decrease it to include weaker connections. The cutoff applies to the coefficient magnitudes in `I`. The graph is based on the adjacency matrix

$$A_{sj} = \mathbf{1}\{I_{sj} \geq t\}, \quad s \neq j, \qquad A_{ss} = 0,$$

where t is `threshold`. Rows are matched to columns by variable name, with zero rows added for missing release variables. Variables with no connections are removed from the graph.

Candidate signals are selected using both connectivity and overall importance. With the default directed graph, a variable must have at least `min.degree` outgoing arrows and a column-sum score at or above the `q.signal` quantile among variables remaining in the graph. The defaults require one outgoing arrow and a score in the upper quartile, including ties.

These two summaries describe different roles. Out-degree counts predictors selected when a variable is released; column-sum importance measures that variable's contributions when other variables are released. In-degree counts the incoming arrows. Column sums use all importance values, including those below the edge cutoff, and are calculated before isolated variables are removed. Consequently, changing `threshold` can also change the set of variables used to determine the importance quantile.

Candidate signal vertices are blue and other vertices are gray. Directed arrows are blue when their source is a candidate signal. Vertex sizes increase with column-sum importance, and arrow widths increase with their importance entries. Specifically, the size is `6 + 2 * log1p(imp.score)` and an edge with value w has width $1 + 3w/w_{\max}$; widths are rescaled within each plot. Use `plot = FALSE` to obtain the graph summaries without drawing it.

The population goal is a signal set that accounts for dependence among the remaining variables (Zhou et al., 2026). Redundant or interchangeable variables can give more than one such set. The graph's signal designations are exploratory candidates.

Choosing a forest and preparing the data: The `method` argument chooses how the forest forms its rules:

`"auto"` The default uses copies of the variables as multiple regression responses, with the variables themselves as predictors. This forest autoencoder forms groups by learning to reproduce the observed variable values.

`"unsupv"` Uses unsupervised splitting, with some variables temporarily serving as responses. `ytry` controls how many are used.

`"rnd"` Uses random splitting with an independently generated Gaussian response.

All three methods use the same subsequent rule comparisons and lasso analysis. `ntree` controls the number of trees grown; `max.tree` and `max.rules.tree` limit rule extraction. Larger `nodesize` values form broader groups of observations.

Rows with missing values are omitted. Categorical variables are converted to numeric indicator columns, called hot-encoding. The processed data are stored in `o$x`. All membership indices refer to these rows, and each processed column is studied separately. A factor can therefore contribute several rows or columns to the importance matrix and several graph vertices.

Preliminary importance and screening: `importance(o)` summarizes the preliminary scores calculated by `uvarpro`. The default rule score compares how much the variables vary within the original and near-miss groups with how much they vary when the groups are pooled. If $d(M)$ is the average column standard deviation of a matrix M, the score is

$$H = \frac{d(X_C) + d(X_O)}{2d(X_C \cup X_O)}.$$

The union denotes row concatenation. Before scoring, each variable is divided by its full-data standard deviation without centering; near-zero standard deviations are replaced by one. All variables, including the released variable, enter this calculation.

By default, `get.beta.entropy` limits both release variables and lasso predictors to those with preliminary importance above `vimp.min`. Set `pre.filter = FALSE` to make all processed variables available to the lasso analysis. The filter is skipped if the preliminary importance calculation fails.

Lasso controls, scaling, and the second stage: Each local classification uses binomial lasso regression from **glmnet**, with $\alpha = 1$. By default, cross-validation chooses a separate penalty for each comparison using binomial deviance. `lambda.sel = "lambda.1se"` uses the largest penalty within one standard error of the minimum; `"lambda.min"` uses the penalty with minimum error. Set `use.cv = FALSE` to use the last, smallest penalty on the computed path instead. `nlambda` and `lambda.min.ratio` control that path.

Before fitting, `scale(X, center = FALSE)` divides each predictor by $\sqrt{\sum_i X_{ij}^2/(m-1)}$, where m is the local sample size. This measures spread around zero rather than around the local mean. `glmnet` also uses its default internal standardization and returns coefficients on the scale of the matrix supplied to it. Predictors that are constant within a comparison are excluded from that fit; their contributions are zero in the importance matrix. A comparison can use a single varying predictor.

With `second.stage = TRUE`, the variables selected by the logistic lasso enter a second local regression. This time the released variable is the response. The regression uses the pooled observations from both groups and a Gaussian lasso, with the same predictor scaling and response values from `o$x`. It uses the last computed penalty, controlled by `nlambda.lasso` and `lambda.min.ratio.lasso`.

The absolute second-stage coefficients are averaged over the comparisons used by the logistic stage. Unselected predictors, failed second-stage fits, and second-stage fits selecting no predictors contribute zeros. Each row is then divided by the released variable's full-data standard deviation:

$$I_{sj}^{(2)} = \frac{1}{\widehat{\sigma}_s |\mathcal{K}_s|} \sum_{r \in \mathcal{K}_s} |\widehat{\gamma}_{srj}|.$$

A zero or nonfinite response standard deviation gives `NA` for that row. `get.beta.entropy` returns this second-stage matrix instead of the logistic matrix. Choose the graph cutoff for the scale of the matrix being displayed.

papply = lapply runs the release-variable analyses in sequence. parallel separately controls the cross-validation folds. Avoid using more simultaneous workers than the available cores when enabling both levels of parallelism. See Note for reproducibility.

The latent-variable example: The example uses four independent standard normal source variables, w, x, y, and z. Each has five noisy copies with independent Gaussian errors of variance 0.1. Two additional variables are h1 = w + x + e21 and h2 = y + z + e22, with independent Gaussian errors of variance 0.4. All errors are independent of the sources, and all 26 variables are supplied to uvarpro.

The four sources form the signal set: conditional on them, the other 22 variables are mutually independent. Releasing a noisy copy of w has w as its s-dependent signal; the other copy blocks have their respective sources. Releasing h1 has signals w and x, while releasing h2 has signals y and z.

Conversely, when w is released, its copies and h1 can help distinguish the groups; x can also contribute through its component of h1. The graph can therefore have arrows in both directions. Repeated contributions across releases motivate the column-sum ranking. The example uses pre.filter = FALSE to include every variable and compares the estimates with this known dependence structure.

Custom scoring functions: Supply entropy through ... to replace the preliminary scoring function. It must accept xC, x0, and .... The first two arguments contain the scaled near-miss and original-group data, with the released variable first. Additional information includes compMembership, oobMembership, the processed data, and xvar.names.

Return a numeric rule score, or a list with the score first and additional information second. To use get.beta.entropy afterwards, the second element must provide the memberships as list(comp = compMembership, oob = oobMembership), in that order. The custom-scoring example reproduces the default score while preserving these memberships.

Value

uvarpro returns an object of class "uvarpro". Its principal components are:

rf The forest used to generate rules.

results Rule-level information with columns tree, branch, variable, n.oob, and imp. The variable index refers to xvar.names. Unevaluated comparisons have NA importance.

x The complete-case, hot-encoded feature data, before the scaling used in scoring or local regressions.

xvar.names, xvar.org.names Processed and original feature names, respectively.

entropy With the default scoring function, membership pairs grouped by released variable. Each pair contains comp and oob observation indices into x. May be NULL when no usable memberships are available. Custom callbacks can change the auxiliary information stored here.

max.rules.tree, max.tree Rule extraction limits.

family The value "unsupv", irrespective of the forest construction method.

get.beta.entropy returns a named numeric matrix, with released variables as rows and eligible predictors as columns. Entries are mean absolute logistic-lasso coefficients when second.stage = FALSE, and response-scaled mean absolute linear-lasso coefficients when second.stage = TRUE. Rows with no retained comparisons are omitted. Returns NULL if no usable release rows remain.

sdependent invisibly returns a list with components:

signal.vars Names of vertices meeting both the degree and importance-quantile conditions. Can be empty even when the graph is nonempty.

imp.score Named global column-sum scores for nonisolated vertices, sorted in increasing order.

degree Named degrees for nonisolated vertices, in graph vertex order. These are out-degrees when `directed = TRUE`.

If no edges remain, `sdependent` returns a character diagnostic. See Note.

Note

For reproducible serial lasso calculations, call `set.seed` immediately before `get.beta.entropy(o, papply = lapply)` to control its randomized cross-validation folds. Also call `set.seed` before `uvarpro` to control R-level sampling in forest construction and rule extraction. The `seed` argument supplies the forest seed but does not replace these R-level seeds.

Small local classes, degenerate predictors, and excessive preliminary filtering can leave no usable comparisons. A failed local lasso fit is excluded from the coefficient average.

For an undirected display, supply a symmetric importance matrix. For square input with matching row and column order, `pmax(I, t(I))` retains either direction. Symmetrization can change global scores and signal designations. Undirected edge colors use the first endpoint in the internal edge list.

To admit weaker positive edges in an empty graph, decrease `threshold`. Keep the cutoff positive, since `>=` admits zero-valued entries at a zero cutoff. Check for `NULL` lasso output before graph construction.

Author(s)

Min Lu and Hemant Ishwaran

References

Tang F. and Ishwaran H. (2017). Random forest missing data algorithms. *Statistical Analysis and Data Mining*, 10:363-377.

Zhou L., Lu M. and Ishwaran H. (2026). Variable priority for unsupervised variable selection. *Pattern Recognition*, 172:112727. doi:10.1016/j.patcog.2025.112727.

See Also

`varpro`, `importance`, `cv.glmnet`, `glmnet`, `scale`, `graph_from_adjacency_matrix`

Examples

```
## Small forest for a quick check
set.seed(1)
o <- uvarpro(mtcars, ntree = 1)


## ------------------------------------------------------------
## Latent-variable model: known signals and s-dependence
## ------------------------------------------------------------
```

```
set.seed(21)
n <- 1000
w <- rnorm(n)
x <- rnorm(n)
y <- rnorm(n)
z <- rnorm(n)
ei <- matrix(rnorm(n * 20, sd = sqrt(.1)), ncol = 20)
e21 <- rnorm(n, sd = sqrt(.4))
e22 <- rnorm(n, sd = sqrt(.4))
wi <- w + ei[, 1:5]
xi <- x + ei[, 6:10]
yi <- y + ei[, 11:15]
zi <- z + ei[, 16:20]
h1 <- w + x + e21
h2 <- y + z + e22
dta <- data.frame(w = w, wi = wi, x = x, xi = xi,
                  y = y, yi = yi, z = z, zi = zi, h1 = h1, h2 = h2)
signal <- c("w", "x", "y", "z")

## All 26 columns are observed features. The remaining 22 are
## mutually independent conditional on the four signal variables.
o <- uvarpro(dta)
print(importance(o))

## Retain every feature for the local lasso analysis, including
## the noisy-copy and mixed-variable releases that reveal signals.
set.seed(22)
beta <- get.beta.entropy(o, pre.filter = FALSE,
                         papply = lapply, nfolds = 5)

if (!is.null(beta)) {
  ## Compare the column-sum ranking with the known signal set.
  score <- sort(colSums(beta), decreasing = TRUE)
  print(data.frame(variable = names(score),
                   importance = unname(score),
                   signal = names(score) %in% signal))

  ## Inspect one copy from each block and the two mixed variables.
  ## Population targets: each copy points to its source; h1 points
  ## to w and x, and h2 to y and z. See Details for source releases.
  release <- c("wi.1", "xi.1", "yi.1", "zi.1", "h1", "h2")
  release <- intersect(release, rownames(beta))
  print(round(beta[release, signal, drop = FALSE], 2))

  ## Inspect the corresponding adjacency block.
  ## A[s, j] = 1 gives the arrow s -> j.
  threshold <- 0.25
  A <- 1L * (beta[release, signal, drop = FALSE] >= threshold)
  print(A)

  if (all(is.finite(beta)) &&
      requireNamespace("igraph", quietly = TRUE)) {
    set.seed(23)
```

```
    ## Pass the lasso matrix directly; row alignment is automatic.
    graph.info <- sdependent(beta, threshold = threshold, layout = "fr",
                             title = "Latent model: s-dependence")
    if (is.list(graph.info)) {
      print(graph.info$signal.vars)
      print(graph.info$degree)
    } else {
      print(graph.info)
    }

    ## Return graph summaries without drawing the figure:
    ## graph.info <- sdependent(beta, threshold = threshold, plot = FALSE)
  }
}

## Optional second-stage lasso, keeping the same feature set.
## set.seed(22)
## beta.second <- get.beta.entropy(o, pre.filter = FALSE,
##                                 second.stage = TRUE,
##                                 papply = lapply, nfolds = 5)

## ------------------------------------------------------------
## Boston housing: alternative forest and categorical features
## ------------------------------------------------------------
if (requireNamespace("mlbench", quietly = TRUE)) {
  data(BostonHousing, package = "mlbench")

  ## Every column is treated as a feature; no response is specified.
  o.unsupv <- uvarpro(BostonHousing, method = "unsupv")
  print(importance(o.unsupv))

  ## Random splitting uses the same subsequent analysis.
  ## o.random <- uvarpro(BostonHousing, method = "rnd")

  Boston <- BostonHousing
  Boston$zn <- factor(Boston$zn)
  Boston$chas <- factor(Boston$chas)
  Boston$lstat <- factor(round(0.2 * Boston$lstat))
  Boston$nox <- factor(round(20 * Boston$nox))
  Boston$rm <- factor(round(Boston$rm))

  o.factor <- uvarpro(Boston)
  print(importance(o.factor))
  print(get.orgvimp(o.factor))
  print(get.orgvimp(o.factor, pretty = FALSE))
}

## ------------------------------------------------------------
## Iowa housing: larger lasso analysis
## ------------------------------------------------------------
data(housing, package = "randomForestSRC")
iowa <- roughfix(housing)
## Numeric coding below is only a speed-oriented illustration;
```

```
## retain factors to use the usual hot-encoding instead.
iowa <- data.frame(data.matrix(iowa))
o.iowa <- uvarpro(iowa, ntree = 50, max.tree = 50,
                  max.rules.tree = 5)
set.seed(22)
beta.iowa <- get.beta.entropy(o.iowa, papply = lapply, nfolds = 5)
if (!is.null(beta.iowa)) {
  print(sort(colMeans(beta.iowa), decreasing = TRUE))
  ## sdependent(beta.iowa)
}

## ------------------------------------------------------------
## Custom scoring callback: reproduces the default dispersion ratio
## ------------------------------------------------------------
my.entropy <- function(xC, xO, ...) {
  mean.sd <- function(x) mean(apply(x, 2, sd, na.rm = TRUE))
  imp <- 0.5 * (mean.sd(xC) + mean.sd(xO)) /
    mean.sd(rbind(xC, xO))
  dots <- list(...)
  list(imp = imp,
       membership = list(comp = dots$compMembership,
                         oob = dots$oobMembership))
}

if (requireNamespace("mlbench", quietly = TRUE)) {
  data(BostonHousing, package = "mlbench")
  o.custom <- uvarpro(BostonHousing, entropy = my.entropy,
                      ntree = 50, max.tree = 50, max.rules.tree = 5)
  print(importance(o.custom))
  ## The stored memberships also allow the lasso analysis.
  ## beta.custom <- get.beta.entropy(o.custom, papply = lapply)
}
```

`varpro` *Model-Independent Variable Selection via the Rule-Based Variable Priority (VarPro)*

Description

Identifies predictors associated with a response. VarPro uses forest rules to compare groups of observations and measure each predictor's importance. Supports regression, multivariate regression, classification, and right-censored survival. Use `importance` to summarize the results and `cv.varpro` to select variables using prediction error.

Usage

```
varpro(formula, data, nvar = 30, ntree = 500,
       split.weight = TRUE, split.weight.method = NULL, sparse = TRUE,
       nodesize = NULL, max.rules.tree = 150, max.tree = min(150, ntree),
```

```
        parallel = TRUE, cores = get.mc.cores(),
        verbose = FALSE, seed = NULL, ...)
```

Arguments

formula	Formula specifying the response and predictors, such as `y ~ .`, `cbind(y1, y2) ~ .`, or `Surv(time, status) ~ .`. Use a factor response for classification and numeric responses for regression. Survival status is coded 0 for censoring and 1 for an event.
data	Data frame containing the response and predictors. Incomplete observations are removed, unused factor levels are dropped, and categorical predictors are hot-encoded. If observations are omitted, a warning reports the input, omitted, and retained counts. See Details.
nvar	Maximum number of processed predictor columns used after preliminary weighting and before rule generation. A categorical predictor can contribute several columns. Applies when computed or custom split-weights are used; `split.weight = FALSE` without custom weights uses all processed predictors.
ntree	Number of trees in the rule-generating forest.
split.weight	Use preliminary weights to choose and favor predictors when growing the forest? At most `nvar` positive-weight predictor columns are used. See Details.
split.weight.method	Character string or vector selecting preliminary weighting methods: `"lasso"`, `"tree"`, and `"vimp"`. Methods can be combined, for example `c("lasso", "tree")`. The default `NULL` selects a combination automatically. See Details.
sparse	Should preliminary weighting concentrate more strongly on promising predictors? Useful when relatively few predictors are expected to carry signal. See Details.
nodesize	Minimum terminal node size for the rule-generating forest. Larger values produce broader regions with more observations for the local comparisons. The default is chosen automatically from the data dimensions.
max.rules.tree	Maximum number of tree branches sampled per selected tree. A branch can yield several comparisons, one for each variable released from its rule.
max.tree	Maximum number of trees sampled for rule extraction. Increasing this value provides more trees for importance aggregation, up to the number grown.
parallel	Should the preliminary lasso cross-validation folds run in parallel?
cores	Number of cores for parallel lasso cross-validation. The default respects `options(mc.cores = ...)` or `MC_CORES`. Forest parallelism is controlled separately by **randomForestSRC**.
verbose	Should progress messages and preliminary split-weights be printed?
seed	Seed supplied to the main forest calculations. Use `set.seed` before the call to also control R-level sampling, lasso folds, and auxiliary calculations. See Note.
...	Additional named controls for split-weight construction, rule generation, and external survival estimation. The supported controls used most often are described in Details. Unnamed controls, unrecognized names, and duplicate names cause an error.

Details

VarPro helps identify predictors that are related to an outcome. It compares groups of observations to measure each predictor's association with changes in the response. A forest defines the groups, allowing the relationships to differ across the data. The comparisons are combined into an importance score for each predictor.

Start with `o <- varpro(y ~ ., data)`, then use `imp <- importance(o)` and `print(imp)` to inspect the scores. Use `importance(o, plot.it = TRUE)` to add a plot. To select a set of predictors using prediction error, call `cv.varpro` with the same formula and data. It performs the VarPro analysis and compares prediction models using different importance cutoffs.

How the comparisons work: A forest rule is a set of conditions on predictor values. For example, a rule might require $x_1 \leq 0.5$ and $x_2 > 0.3$. To assess x_1, VarPro removes its condition while keeping the condition on x_2. This is called *releasing* a predictor.

The observations newly admitted by this change satisfy $x_1 > 0.5$ and $x_2 > 0.3$. They form the rule's *near-miss set*. Comparing their responses with those in the original rule region measures the local contribution of x_1, with the restriction on x_2 unchanged.

More generally, for a rule region R, releasing predictor s gives an enlarged region $R^{(s)}$. The near-miss set is

$$C_s = R^{(s)} \setminus R.$$

The original-region observations used in each comparison are out-of-bag for that tree: they were not used to grow it. Repeating the comparisons across rules allows a predictor to be important in some groups of observations and less important in others.

For regression, the comparisons concern mean responses; for classification, they concern class proportions. Survival uses a numeric summary estimated by a preliminary survival forest, as described below.

Importance scores and variable selection: `importance(o)` combines the rule comparisons into predictor scores. Larger scores indicate stronger response differences when that predictor is released. Classification supplies both an overall score and class-specific scores. Multivariate regression supplies a separate importance table for each response.

The default `local.std = TRUE` in `importance` accounts for the variability and sample sizes of each comparison. In regression, the rule score is the absolute difference in group means divided by its estimated standard error:

$$T = \frac{|\bar{y}_R - \bar{y}_{C_s}|}{\sqrt{s_R^2/n_R + s_{C_s}^2/n_{C_s}}}.$$

This is the absolute Welch statistic. The same calculation is used for each numeric response, including survival summaries. Classification uses a chi-squared comparison of class proportions for the overall score and standardized differences in proportions for the class-specific scores. The overall rule score is $\sqrt{\chi^2/d}$, where d is the degrees of freedom; two-class tables use a continuity correction.

Comparisons within a tree are averaged using the numbers of original-region observations as weights. Class-specific scores use the corresponding class counts. Before averaging across trees, unusually small and large tree scores are limited to the lower and upper 10 percent quantiles by default. This is *winsorization*. The resulting mean is reported as `z`. With `local.std = FALSE`,

importance instead summarizes the stored rule scores and divides their winsorized mean by their winsorized standard deviation across trees.

The importance cutoff, 0.79 by default, highlights variables in its plots; it does not filter the returned importance table. get.topvars(o) gives the predictor names represented in that table. Use cv.varpro for selection based on prediction error. Its optional cv.folds argument also assesses prediction on observations left out of selection and reports how often each variable is selected. See importance.varpro and cv.varpro.

Choosing predictors for rule generation: By default, preliminary weights help the forest find useful predictors before the rule comparisons are made. The weights determine which predictors are used and how often they are offered as candidates for a split. Three weighting methods are available:

"lasso" Absolute coefficients from a penalized regression fitted to standardized predictors.

"tree" How often each predictor is used to split a shallow forest.

"vimp" Positive permutation importance from a preliminary forest.

split.weight.method = NULL chooses a combination automatically. Supply a method or a combination such as c("lasso", "tree") to choose the sources of the weights. The lasso uses the more strongly penalized solution within one standard error of the minimum cross-validation error. Coefficient magnitudes are combined across responses or classes when needed.

nvar limits the number of positive-weight predictor columns used to generate rules. It is a preliminary screening limit, not the requested number of variables in a final selection. Factors can contribute several columns. Increase nvar to allow more columns, or set split.weight = FALSE to use all processed predictors without preliminary weighting. With sparse = TRUE, the weighting puts more emphasis on predictors with larger preliminary weights.

Supply split.weight.custom through ... to use a named vector of your own weights. Names must match the processed predictor columns; unspecified columns receive zero weight. Use get.splitweight.custom(formula, data) to obtain a starting vector with the correct names. Custom weights also use the nvar limit. The final importance scores come from the rule comparisons, rather than from the preliminary weights.

Categorical predictors and classification: Categorical predictors are converted to numeric columns, called hot-encoding. A two-level factor gives one binary column; a factor with more levels gives indicator columns. Numeric predictors keep their values. Screening, rules, and importance(o) use these processed columns. o$x contains the full processed predictor data, and o$xvar.names identifies the columns used by the rule-generating forest.

Use get.orgvimp(o) for scores under the original predictor names. It takes the largest score among a predictor's encoded columns and, for multivariate regression, across responses. cv.varpro uses this summary to form its candidate variable sets. get.vimp(o) gives named scores for the processed columns. With pretty = FALSE, these two helpers include all original or processed predictors, respectively, filling missing scores with zero.

Binary classification uses working labels 0 for the majority class and 1 for the minority class. The original labels are stored in y.org and used by the rule-generating forest. When the class-imbalance rule is met, the forest uses RFQ and AUC splitting; preliminary permutation importance uses geometric-mean performance. The controls use.rfq and iratio.threshold govern this behavior. model.info$classification records the class mapping and imbalance settings.

Survival targets: For survival data, a preliminary survival forest estimates a numeric response summary for each training observation. The default is forest mortality. Supply rmst = tau through

... to use restricted mean survival time: the estimated survival curve integrated up to time tau. A vector of horizons requests a separate summary for each horizon. Horizons use the same time units as the survival outcome. Beyond the last stored time point, the calculation continues using the last estimated survival value.

These full-ensemble training predictions guide preliminary weighting and the default locally standardized importance. With mortality or a single RMST response, rules are generated from the original survival outcome. With multiple RMST responses, rules are generated by multivariate regression on those responses. ntree.external, nodesize.external, and ntime.external control the preliminary survival forest.

model.info distinguishes the original outcome, the working numeric targets, and the family of the rule-generating forest. This is useful when inspecting an analysis with survival-derived responses.

Computation and reproducibility: ntree controls the number of trees grown. max.tree and max.rules.tree limit the trees and branches used for the comparisons. Smaller limits reduce computation. Larger nodesize values give broader comparison groups with less local detail.

parallel and cores control the preliminary lasso cross-validation, with nfolds supplied through .... Forest threading is controlled separately by **randomForestSRC**. For reproducible serial calculations, use set.seed before the analysis and parallel = FALSE; see Note.

Value

An object of class "varpro" with the following components:

rf	The rule-generating rfsrc forest.
split.weight	Named weights for the predictor columns used by the forest, or NULL when computed and custom weighting are both absent.
split.weight.raw	List of untransformed preliminary components actually calculated, named lasso, tree, or vimp. Empty when preliminary estimation is skipped.
results	Rule-level data frame with tree and branch identifiers, released-variable index, original-region sample size n.oob, and importance columns. Indices refer to xvar.names. Classification includes class-specific counts and scores; multivariate regression includes one score per response. importance supplies the variable-level summary.
x	Complete-case processed predictor data, including columns removed by preliminary screening. Encoding information is stored as attributes.
xvar.names	Processed predictor names used for rule generation.
xvar.org.names	Original predictor names before encoding.
y	Working response: numeric response(s), recoded class labels, or external survival summary values.
y.org	Original response values for the retained observations. For survival, these are the observed times and event indicators.
yvar.names	Names of the working response columns.
family	Family of the rule-generating forest: "regr", "regr+", "class", or "surv". Multiple RMST horizons use "regr+".

`max.rules.tree`, `max.tree`
: Stored rule-extraction limits used by subsequent importance calculations.

`model.info`
: Description of the original outcome (`original`), working responses (`working`), and rule-generating forest (`forest`), with additional survival or classification information when applicable. The `observations` component records `input`, `retained`, and `omitted` row counts for the data supplied to this `varpro` call.

Note

For reproducible serial analyses, call `set.seed` immediately before `varpro` and use `parallel = FALSE`. The `seed` argument controls only the forest calls to which it is passed. Rule resampling and lasso folds also use random numbers. Calling `importance` again can resample rules. Save its result to inspect or reuse the same summary without recalculating it.

The `...` controls are consumed by specific stages. General `rfsrc` arguments such as `na.action`, `splitrule`, and `perf.type` are not forwarded wholesale. Perform any desired imputation before calling `varpro`.

Author(s)

Min Lu and Hemant Ishwaran

References

Lu, M. and Ishwaran, H. (2024). Model-independent variable selection via the rule-based variable priority. arXiv:2409.09003. doi:10.48550/arXiv.2409.09003.

Friedman, J., Hastie, T. and Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1), 1–22. doi:10.18637/jss.v033.i01.

O'Brien, R. and Ishwaran, H. (2019). A random forests quantile classifier for class imbalanced data. *Pattern Recognition*, 90, 232–249.

Ishwaran, H. (2025). *Multivariate Statistics: Classical Foundations and Modern Machine Learning*. Chapman and Hall/CRC.

See Also

`importance.varpro`, `cv.varpro`, `predict.varpro`, `ivarpro`, `uvarpro`, `partialpro`, `outpro`, `isopro`, `randomForestSRC::rfsrc`, `glmnet::cv.glmnet`.

Examples

```
## Small regression example.
o <- varpro(mpg ~ ., mtcars, ntree = 1)


## Nonlinear regression with five signal and five noise predictors.
set.seed(137)
n <- 300
x <- matrix(runif(n * 10), nrow = n)
colnames(x) <- sprintf("x%02d", seq_len(ncol(x)))
mu <- 10 * sin(pi * x[, 1] * x[, 2]) +
```

```
      20 * (x[, 3] - 0.5)^2 + 10 * x[, 4] + 5 * x[, 5]
d <- data.frame(y = mu + rnorm(n), x)
signal <- colnames(x)[1:5]

## Shallow-tree-guided rule generation.
o <- varpro(y ~ ., d, ntree = 25, ntree.reduce = 25,
            split.weight.method = "tree", nodesize = 10,
            max.tree = 25, max.rules.tree = 20, parallel = FALSE)
imp <- importance(o)
print(data.frame(variable = rownames(imp), z = imp$z,
                 signal = rownames(imp) %in% signal))

## Automatic weighting with more trees and comparisons.
set.seed(137)
o <- varpro(y ~ ., d, ntree = 150, max.tree = 100,
            max.rules.tree = 50, nfolds = 5, parallel = FALSE)
imp <- importance(o, plot.it = TRUE)
print(imp)
print(get.vimp(o))
print(get.vimp(o, pretty = FALSE))
print(get.topvars(o))

## Choose a variable set using predictive performance.
set.seed(137)
## Add cv.folds = 5 to assess selection and obtain stability summaries.
cv <- cv.varpro(y ~ ., d, ntree = 100,
                zcut = seq(0.1, 3, length.out = 12), nblocks = 6,
                max.tree = 75, max.rules.tree = 40,
                nfolds = 5, parallel = FALSE)
print(cv$imp)
print(cv$imp.conserve)

## Hot-encoding: compare processed-column and original-variable scores.
## Several predictors are converted to factors before analysis.
data(BostonHousing, package = "mlbench")
Boston <- BostonHousing
Boston$zn <- factor(Boston$zn)
Boston$chas <- factor(Boston$chas)
Boston$lstat <- factor(round(0.2 * Boston$lstat))
Boston$nox <- factor(round(20 * Boston$nox))
Boston$rm <- factor(round(Boston$rm))
set.seed(137)
b <- varpro(medv ~ ., Boston, ntree = 100,
            max.tree = 75, max.rules.tree = 40,
            nfolds = 5, parallel = FALSE)

## Calculate the processed-column importance
set.seed(137)
b.imp <- importance(b)
print(b.imp)

## Map max scores back to original predictors.
set.seed(137)
```

```
b.org <- get.orgvimp(b)
print(b.org)
print(get.orgvimp(b, vmp = b.imp)) ## should be identical; no seed required

## Include every original predictor, filling absent scores with zero.
set.seed(137)
print(get.orgvimp(b, pretty = FALSE))


## Multiclass classification: overall and class-specific importance.
data(wine, package = "randomForestSRC")
wine$quality <- factor(wine$quality)
set.seed(137)
w <- varpro(quality ~ ., wine, ntree = 100,
            max.tree = 75, max.rules.tree = 40,
            nfolds = 5, parallel = FALSE)
w.imp <- importance(w)
print(w.imp$unconditional)
print(w.imp$conditional.z)

## Survival: canonical example
data(peakVO2, package = "randomForestSRC")
s <- varpro(Surv(ttodead, died)~., peakVO2)
print(importance(s))


## Survival: prioritize variables affecting RMST through 500 days.
data(pbc, package = "randomForestSRC")
pbc <- na.omit(pbc)
set.seed(137)
s <- varpro(Surv(days, status) ~ ., pbc, rmst = 500,
            ntree = 100, max.tree = 75, max.rules.tree = 40,
            nfolds = 5, parallel = FALSE)
print(importance(s))
print(get.orgvimp(s))
print(get.vimp(s, pretty = FALSE))

## Multiple horizons give one importance summary per horizon.
set.seed(137)
s.multi <- varpro(Surv(days, status) ~ ., pbc,
                  rmst = c(500, 1000), ntree = 100,
                  max.tree = 75, max.rules.tree = 40,
                  nfolds = 5, parallel = FALSE)
print(importance(s.multi))
```

`varpro.news` *Show the NEWS file*

Description

Show the NEWS file of the **varPro** package.

Usage

```
varpro.news(...)
```

Arguments

...	Further arguments passed to or from other methods.

Value

None.

Author(s)

Min Lu and Hemant Ishwaran

---

varpro.strength *Obtain Strength Array and Other Values from a VarPro Object*

Description

Used to parse values from a VarPro object.

Usage

```
varpro.strength(object,
                newdata,
                m.target = NULL,
                max.rules.tree = 150,
                max.tree = 150,
                stat = c("importance", "complement", "oob", "none"),
                membership = FALSE,
                neighbor = 5,
                seed = NULL,
                do.trace = FALSE, ...)
```

Arguments

object	rfsrc object.
newdata	Optional test data. If provided, returns branch and complementary branch membership of the training data corresponding to the test cases.
m.target	Character string specifying the target outcome for multivariate families. If unspecified, a default is selected automatically.
max.rules.tree	Maximum number of rules extracted per tree.
max.tree	Maximum number of trees used for rule extraction.

stat	Statistic to return: "importance" for release importance, "complement" for the complementary-region response summary, "oob" for the original-region OOB response summary, or "none" to omit response summaries.
membership	Return out-of-bag and complementary membership indices for each rule?
neighbor	Nearest neighbor parameter, used only when newdata is specified.
seed	Seed for reproducibility.
do.trace	Enable detailed trace output.
...	Additional arguments.

Details

Not intended for direct end-user use; primarily designed for internal package operations.

Value

Object coerced so as to work with other functions in the package.

Examples

```
## ------------------------------------------------------------
## regression example: boston housing
## ------------------------------------------------------------

## load the data
data(BostonHousing, package = "mlbench")

o <- randomForestSRC::rfsrc(medv~., BostonHousing, ntree=100)

## call varpro.strength
vs <- varpro.strength(object = o, max.rules.tree = 10, max.tree = 15)

## call varpro.strength with test data
vs <- varpro.strength(object = o, newdata = BostonHousing[1:3,], max.rules.tree = 10, max.tree = 15)
```

Index

∗ **datasets**
  alzheimers, 2
  gliomas, 11
  hrrecov, 11
∗ **documentation**
  varpro.news, 75
∗ **graphs**
  uvarpro, 58
∗ **hplot**
  plot.ivarpro, 43
∗ **individual importance**
  ivarpro, 23
∗ **models**
  cv.varpro, 4
  predict.ivarpro, 52
  varpro, 68
∗ **multivariate**
  uvarpro, 58
∗ **outlier**
  isopro, 19
∗ **plot**
  partialpro, 35
  plot.partialpro, 46
∗ **predict outlier**
  predict.isopro, 50
∗ **predict uvarpro**
  predict.uvarpro, 54
∗ **predict varpro**
  predict.varpro, 56
∗ **tree**
  cv.varpro, 4
  varpro, 68
∗ **varpro.strength**
  varpro.strength, 76
∗ **varpro**
  importance, 14

alzheimers, 2

cv.glmnet, *65*
cv.varpro, 4, *16, 17*, *71*, *73*

get.beta.entropy *(*uvarpro*)*, 58
glioma *(*gliomas*)*, 11
gliomas, 11
glmnet, *65*
glmnet::cv.glmnet, *73*
graph_from_adjacency_matrix, *65*

hrrecov, 11

importance, 14, *65*
importance.varpro, *8*, *71*, *73*
isopro, 19, *38*, *40*, *51*, *73*
ivarpro, 23, *45*, *53*, *73*

outpro, 29, *38*, *40*, *73*

partialpro, 35, *48*, *73*
plot.ivarpro, *26, 27*, 43, *53*
plot.partialpro, 46
predict.isopro, *21*, 50
predict.ivarpro, *26, 27*, *45*, 52
predict.uvarpro, 54
predict.varpro, *8*, 56, *73*

randomForestSRC::rfsrc, *8*, *73*
randomForestSRC::rfsrc.fast, *8*
rfsrc, *33*

scale, *65*
sdependent *(*uvarpro*)*, 58
shap.ivarpro, *27*, *53*
shap.ivarpro *(*plot.ivarpro*)*, 43

uvarpro, *17*, *21*, *51*, *55*, 58, *73*

varpro, *4, 5*, *8*, *16, 17*, *21*, *27*, *33*, *40*, *51*, *56*, *65*, 68
varpro.news, 75
varpro.strength, 76