Package 'cpi'

May 8, 2026

Type Package

Title Conditional Predictive Impact

Version 0.1.5

Date 2024-11-05

Maintainer Marvin N. Wright <cran@wrig.de>

Description A general test for conditional independence in supervised learning algorithms as proposed by Watson & Wright (2021) <doi:10.1007/s10994-021-06030-6>. Implements a conditional variable importance measure which can be applied to any supervised learning algorithm and loss function. Provides statistical inference procedures without parametric assumptions and applies equally well to continuous and categorical predictors and outcomes.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.2

URL https://github.com/bips-hb/cpi, https://bips-hb.github.io/cpi/

BugReports https://github.com/bips-hb/cpi/issues

Imports foreach, mlr3, lgr, knockoff

Suggests mlr3learners, ranger, glmnet, testthat (>= 3.0.0), knitr, rmarkdown, doParallel

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Author Marvin N. Wright [aut, cre] (ORCID: <https://orcid.org/0000-0002-8542-6291>), David S. Watson [aut]

Repository CRAN

Date/Publication 2024-11-25 15:10:02 UTC

Contents

`cpi` *Conditional Predictive Impact (CPI).*

Description

A general test for conditional independence in supervised learning algorithms. Implements a conditional variable importance measure which can be applied to any supervised learning algorithm and loss function. Provides statistical inference procedures without parametric assumptions and applies equally well to continuous and categorical predictors and outcomes.

Usage

```
cpi(
  task,
  learner,
  resampling = NULL,
  test_data = NULL,
  measure = NULL,
  test = "t",
  log = FALSE,
  B = 1999,
  alpha = 0.05,
  x_tilde = NULL,
  aggr_fun = mean,
  knockoff_fun = function(x) knockoff::create.second_order(as.matrix(x)),
  groups = NULL,
  verbose = FALSE
)
```

Arguments

`task`	The prediction `mlr3` task, see examples.
`learner`	The `mlr3` learner used in CPI. If you pass a string, the learner will be created via `mlr3::lrn`.
`resampling`	Resampling strategy, `mlr3` resampling object (e.g. `rsmp("holdout")`), "oob" (out-of-bag) or "none" (in-sample loss).
`test_data`	External validation data, use instead of resampling.
`measure`	Performance measure (loss). Per default, use MSE (`"regr.mse"`) for regression and logloss (`"classif.logloss"`) for classification.

test	Statistical test to perform, one of "t" (t-test, default), "wilcox" (Wilcoxon signed-rank test), "binom" (binomial test), "fisher" (Fisher permutation test) or "bayes" (Bayesian testing, computationally intensive!). See Details.
log	Set to TRUE for multiplicative CPI (λ), to FALSE (default) for additive CPI (Δ).
B	Number of permutations for Fisher permutation test.
alpha	Significance level for confidence intervals.
x_tilde	Knockoff matrix or data.frame. If not given (the default), it will be created with the function given in knockoff_fun. Also accepts a list of matrices or data.frames.
aggr_fun	Aggregation function over replicates.
knockoff_fun	Function to generate knockoffs. Default: knockoff::create.second_order with matrix argument.
groups	(Named) list with groups. Set to NULL (default) for no groups, i.e. compute CPI for each feature. See examples.
verbose	Verbose output of resampling procedure.

Details

This function computes the conditional predictive impact (CPI) of one or several features on a given supervised learning task. This represents the mean error inflation when replacing a true variable with its knockoff. Large CPI values are evidence that the feature(s) in question have high *conditional variable importance* – i.e., the fitted model relies on the feature(s) to predict the outcome, even after accounting for the signal from all remaining covariates.

We build on the mlr3 framework, which provides a unified interface for training models, specifying loss functions, and estimating generalization error. See the package documentation for more info.

Methods are implemented for frequentist and Bayesian inference. The default is test = "t", which is fast and powerful for most sample sizes. The Wilcoxon signed-rank test (test = "wilcox") may be more appropriate if the CPI distribution is skewed, while the binomial test (test = "binom") requires basically no assumptions but may have less power. For small sample sizes, we recommend permutation tests (test = "fisher") or Bayesian methods (test = "bayes"). In the latter case, default priors are assumed. See the BEST package for more info.

For parallel execution, register a backend, e.g. with doParallel::registerDoParallel().

Value

For test = "bayes" a list of BEST objects. In any other case, a data.frame with a row for each feature and columns:

Variable/Group	Variable/group name
CPI	CPI value
SE	Standard error
test	Testing method
statistic	Test statistic (only for t-test, Wilcoxon and binomial test)
estimate	Estimated mean (for t-test), median (for Wilcoxon test), or proportion of Δ-values greater than 0 (for binomial test).

p.value	p-value
ci.lo	Lower limit of (1 - alpha) * 100% confidence interval

Note that NA values are no error but a result of a CPI value of 0, i.e. no difference in model performance after replacing a feature with its knockoff.

References

Watson, D. & Wright, M. (2020). Testing conditional independence in supervised learning algorithms. *Machine Learning*, *110*(8): 2107-2129. doi:10.1007/s10994021060306

Candès, E., Fan, Y., Janson, L, & Lv, J. (2018). Panning for gold: 'model-X' knockoffs for high dimensional controlled variable selection. *J. R. Statistc. Soc. B*, *80*(3): 551-577. doi:10.1111/rssb.12265

Examples

```
library(mlr3)
library(mlr3learners)

# Regression with linear model and holdout validation
cpi(task = tsk("mtcars"), learner = lrn("regr.lm"),
    resampling = rsmp("holdout"))


# Classification with logistic regression, log-loss and t-test
cpi(task = tsk("wine"),
    learner = lrn("classif.glmnet", predict_type = "prob", lambda = 0.1),
    resampling = rsmp("holdout"),
    measure = "classif.logloss", test = "t")

# Use your own data (and out-of-bag loss with random forest)
mytask <- as_task_classif(iris, target = "Species")
mylearner <- lrn("classif.ranger", predict_type = "prob", keep.inbag = TRUE)
cpi(task = mytask, learner = mylearner,
    resampling = "oob", measure = "classif.logloss")

# Group CPI
cpi(task = tsk("iris"),
    learner = lrn("classif.ranger", predict_type = "prob", num.trees = 10),
    resampling = rsmp("cv", folds = 3),
    groups = list(Sepal = 1:2, Petal = 3:4))

## Not run:
# Bayesian testing
res <- cpi(task = tsk("iris"),
           learner = lrn("classif.glmnet", predict_type = "prob", lambda = 0.1),
           resampling = rsmp("holdout"),
           measure = "classif.logloss", test = "bayes")
plot(res$Petal.Length)

# Parallel execution
doParallel::registerDoParallel()
```

```
cpi(task = tsk("wine"),
    learner = lrn("classif.glmnet", predict_type = "prob", lambda = 0.1),
    resampling = rsmp("cv", folds = 5))

# Use sequential knockoffs for categorical features
# package available here: https://github.com/kormama1/seqknockoff
mytask <- as_task_regr(iris, target = "Petal.Length")
cpi(task = mytask, learner = lrn("regr.ranger"),
    resampling = rsmp("holdout"),
    knockoff_fun = seqknockoff::knockoffs_seq)

## End(Not run)
```

Index