Package: XAItest (via r-universe)

June 9, 2026

Title XAItest: Enhancing Feature Discovery with eXplainable AI

Version 1.4.0

Description XAItest is an R Package that identifies features using eXplainable AI (XAI) methods such as SHAP or LIME. This package allows users to compare these methods with traditional statistical tests like t-tests, empirical Bayes, and Fisher's test. Additionally, it includes simThresh, a system that enables the comparison of feature importance with p-values by incorporating calibrated simulated data.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

biocViews Software, StatisticalMethod, FeatureExtraction, Classification, Regression

Suggests knitr, ggforce, shapr (>= 1.0.1), airway, xgboost, BiocGenerics, RUnit, S4Vectors

Imports limma, randomForest, kernelshap, caret, lime, DT, methods, SummarizedExperiment, ggplot2

VignetteBuilder knitr

URL https://github.com/GhislainFievet/XAItest

BugReports https://github.com/GhislainFievet/XAItest/issues

Config/pak/sysreqs cmake make libicu-dev libuv1-dev zlib1g-dev

Repository https://bioc-release.r-universe.dev

Date/Publication 2026-04-28 13:04:39 UTC

RemoteUrl https://github.com/bioc/XAItest

RemoteRef RELEASE_3_23

RemoteSha a7f2cb4dd33d58e8084261a0ab12f72d2298e581

Contents

`addSimThresh` *Add "simThresh" Simulated Threshold Feature*

Description

Adds a simulated feature to the dataset that has a p-value close to a specified threshold when tested against the target variable. This function is useful for testing feature selection methods and significance thresholds.

Usage

```
addSimThresh(
  data,
  target_column = "y",
  target_type = "default",
  pval_target = 0.049,
  tolerance = 5e-04,
  adj_bonf = FALSE
)
```

Arguments

`data`	A data frame containing the dataset.
`target_column`	Character; name of the target column in the dataset. Default is "y".
`target_type`	Character; specifies the type of the target variable. Can be "default", "regression", or "classification". If "default", the function will automatically determine the type based on the target column.
`pval_target`	Numeric; the desired p-value for the simulated feature. Default is 0.049.
`tolerance`	Numeric; the acceptable deviation from the target p-value. Default is 0.0005.
`adj_bonf`	Logical; if TRUE, applies Bonferroni correction when calculating the p-value. Default is FALSE.

Value

The original data frame with an additional column named "simThresh" containing the simulated feature.

Examples

```
# Example with a regression dataset
df <- data.frame(
  feature1 = rnorm(100),
  feature2 = rnorm(100, mean = 5),
  feature3 = runif(100, min = 0, max = 10),
  y = 1:100
)

# Add a simulated feature with p-value close to 0.05
df_with_sim <- addSimThresh(df, target_column = "y", pval_target = 0.05)

# Check that the correlation between simThresh and the target variable is close to 0.05
print(cor.test(df_with_sim$simThresh, df_with_sim$y)$p.value)

# Example with a classification dataset
df_class <- data.frame(
  feature1 = rnorm(100),
  feature2 = rnorm(100, mean = 5),
  feature3 = runif(100, min = 0, max = 10),
  y = factor(c(rep("A", 50), rep("B", 50)))
)

# Add a simulated feature with p-value close to 0.01
df_class_with_sim <- addSimThresh(df_class, target_column = "y",
                                  pval_target = 0.05)

# Check that the p-value of the simThresh column is close to 0.05
print(t.test(simThresh ~ y, data = df_class_with_sim)$p.value)
```

---

genScenario *Generate benchmark scenarios for feature discovery*

Description

Creates synthetic datasets used in the vignette to compare classical statistical tests and XAI-based feature discovery methods across several classification and regression settings.

Usage

```
genScenario(
  scenario = 1,
```

```
  n_samples = 100,
  n_norm_noise_features = NULL,
  n_unif_noise_features = NULL
)
```

Arguments

scenario	Integer between 1 and 6 selecting the simulation design.
n_samples	Number of samples to generate.
n_norm_noise_features	Number of Gaussian noise features to add. Defaults depend on the chosen scenario.
n_unif_noise_features	Number of uniform noise features to add. Defaults depend on the chosen scenario.

Value

A data frame containing the simulated response y and the corresponding explanatory variables for the selected scenario.

Examples

```
df <- genScenario(1, n_samples = 20, n_norm_noise_features = 2)
head(df)
```

---

getFeatImpThresholds *The getFeatImpThresholds function identifies the minimum level of feature importance required to exceed a specified significance threshold, which is determined by the p-value.*

Description

The getFeatImpThresholds function identifies the minimum level of feature importance required to exceed a specified significance threshold, which is determined by the p-value.

Usage

```
getFeatImpThresholds(
  df,
  refPvalColumn = "adjpval",
  featImpColumns = "feat",
  refPval = 0.05
)
```

Arguments

df	A dataframe containing p-value columns and feature importance columns.
refPvalColumn	Optional; the name of the column containing the reference p-values. If not provided, the function will search for a column name containing "adjpval", if not existing a column name containing "pval" (case insensitive).
featImpColumns	Optional; a vector of column names containing the feature importance values. If not provided, the function will search for column names containing "feat" (case insensitive).
refPval	The reference p-value threshold for filtering features. Defaults to 0.05.

Details

The reference p-value column can be given by the refPvalColumn argument. If not provided, the function will search for the first df column name containing "pval". The feature importance columns can be given by the featImpColumns argument. If not provided, the function will search for all df column names containing "feat".

It then selects feature importance values of features with p-values under the specified threshold and returns the lowest.

This is useful for identifying the most significant features in a dataset based on statistical testing, aiding in the interpretation of machine learning models and exploratory data analysis.

Value

A named vector of minimum feature importance values for each feature passing the p-value filter. The names of the vector elements correspond to the feature importance columns in df.

Examples

```
# Assuming `df` is a dataframe with columns `feature1_pval`,
# `feature2_pval`, `feature1_imp`, `feature2_imp`
df <- data.frame(pval = c(0.04, 0.02, 0.06, 0.8),
                 adjPval = c(0.01, 0.03, 0.05, 0.9),
                 feat_imp_1 = c(0.2, 0.3, 0.1, 0.6),
                 feat_imp_2 = c(0.4, 0.5, 0.3, 0.6))
thresholds <- getFeatImpThresholds(df)
print(thresholds)
```

---

getMetricsTable *Get the Metrics Table*

Description

This method retrieves the metrics table from an ObjXAI object.

Usage

```
getMetricsTable(object)
```

Arguments

object	An ObjXAI object.

Value

A data frame containing the metrics.

Examples

```
obj <- new("ObjXAI",
           data = data.frame(),
           dataSim = data.frame(),
           metricsTable = data.frame(Metric = c("Accuracy", "Precision"),
                                     Value = c(0.95, 0.89)),
           map = list(),
           models = list(),
           modelPredictions = list(),
           args = list())
getMetricsTable(obj)
```

---

mapPvalImportance

The mapPvalImportance function displays a datatable with color-coded cells based on significance thresholds for feature importance and p-value columns.

Description

The mapPvalImportance function displays a datatable with color-coded cells based on significance thresholds for feature importance and p-value columns.

Usage

```
mapPvalImportance(
  objXAI,
  refPvalColumn = "adjpval",
  featImpColumns = "feat",
  pvalColumns = NULL,
  refPval = 0.05
)
```

Arguments

objXAI	An object of class ObjXAI.
refPvalColumn	Optional; the name of the column containing reference p-values for feature importance. If not provided, the function will attempt to auto-detect.
featImpColumns	Optional; a vector of column names containing feature importance values. If not provided, the function will attempt to auto-detect.
pvalColumns	Optional; a vector of column names containing p-values. If not provided, the function searches for columns containing "pval" (case insensitive).
refPval	The reference p-value threshold used for filtering. Defaults to 0.05.

Details

The function first identifies the relevant p-value columns and feature importance columns, if not explicitly provided. It then calculates feature importance thresholds based on the specified p-value threshold. Then the dataframe is displaid with color-coded cells based on significance thresholds for feature importance and p-value columns.

Value

A dataframe and a datatable object with color-coded cells based on significance thresholds for feature importance and p-value columns.

Examples

```
df <- data.frame(
  feature1 = rnorm(10),
  feature2 = rnorm(10, mean = 5),
  feature3 = runif(10, min = 0, max = 10),
  feature4 = c(rnorm(5), rnorm(5, mean = 5)),
  categ = c(rep("Cat1",5), rep("Cat2", 5))
)

results <- XAI.test(df, y = "categ", simData = TRUE)

my_map <- mapPvalImportance(results)
my_map$df
my_map$dt
```

---

modelsOverview *Models Overview*

Description

Returns **mse**, **rmse**, **mae** and **r2** of regression models or **accuracy**, **precision**, **recall** and **f1_score** of classification models.

Usage

```
modelsOverview(objXAI, verbose = FALSE)
```

Arguments

objXAI	An object of class ObjXAI.
verbose	Logical; if TRUE, prints the models names.

Value

Returns **mse**, **rmse**, **mae** and **r2** of regression models or **accuracy**, **precision**, **recall** and **f1_score** of classification models.

Examples

```
# Example with SummarizedExperiment object with a regression dataset.

library(S4Vectors)
library(SummarizedExperiment)

df <- data.frame(
 feature1 = rnorm(100),
 feature2 = rnorm(100, mean = 5),
 feature3 = runif(100, min = 0, max = 10),
 feature4 = c(rnorm(50), rnorm(50, mean = 5)),
 y = 1:100
)

assays <- SimpleList(counts = as.matrix(t(df[, 1:4])))
colData <- DataFrame(y = df[,"y"])
se <- SummarizedExperiment(assays = assays,
                           colData = colData)

resultsRegr <- XAI.test(se, y = "y", verbose = TRUE)

modelsOverview(resultsRegr)

# Example with a dataframe with a classification dataset.
df <- data.frame(
 feature1 = rnorm(100),
 feature2 = rnorm(100, mean = 5),
 feature3 = runif(100, min = 0, max = 10),
 feature4 = c(rnorm(50), rnorm(50, mean = 5)),
 y = c(rep("Cat1", 50), rep("Cat2", 50))
)
resultsClassif <- XAI.test(df, y = "y", verbose = TRUE)

modelsOverview(resultsClassif)
```

ObjXAI-class *ObjXAI class*

Description

ObjXAI is a class used to store the output values of the XAI.test function.

Value

A ObjXAI object

Examples

```
obj <- new("ObjXAI",
           data = data.frame(),
           dataSim = data.frame(),
           metricsTable = data.frame(Metric = c("Accuracy", "Precision"),
                                     Value = c(0.95, 0.89)),
           map = list(),
           models = list(),
           modelPredictions = list(),
           args = list())
```

plotModel *Plot the model*

Description

This function plots the model.

Usage

```
plotModel(objXAI, modelName, xFeature, yFeature = "")
```

Arguments

objXAI	The ObjXAI object created with the XAItest function
modelName	The name of the model, can be found in 'names(objXAI@models)'
xFeature	The x feature
yFeature	The y feature

Value

A plot

Examples

```
data(iris)
iris = subset(iris, Species == "setosa" | Species == "versicolor")
iris$Species = as.character(iris$Species)
objXAI <- XAI.test(iris, y = "Species")
plotModel(objXAI, "RF_feat_imp", "Sepal.Length", "Sepal.Width")
```

`setMetricsTable` *Set the Metrics Table*

Description

This method sets the metrics table for an ObjXAI object.

Usage

```
setMetricsTable(object, value)
```

Arguments

`object`	An ObjXAI object.
`value`	A data frame to set as the metrics table.

Value

The modified ObjXAI object.

Examples

```
obj <- new("ObjXAI",
           data = data.frame(),
           dataSim = data.frame(),
           metricsTable = data.frame(Metric = c("Accuracy", "Precision"),
                                     Value = c(0.95, 0.89)),
           map = list(),
           models = list(),
           modelPredictions = list(),
           args = list())

setMetricsTable(obj, data.frame(Metric = c("Accuracy", "Precision", "Recall"),
                                Value = c(0.95, 0.89, 0.91)))
```

`show,ObjXAI-method` *Show Method for ObjXAI*

Description

Prints the first 5 rows of the metrics table from an ObjXAI object.

Usage

```
## S4 method for signature 'ObjXAI'
show(object)
```

Arguments

`object`	An ObjXAI object.

Value

The first 5 rows of the metrics table.

`XAI.test` *The XAI.test function complements t-test and correlation analyses in feature discovery by integrating eXplainable AI techniques such as feature importance, SHAP, LIME, or custom functions. It provides the option of automatic integration of simulated data to facilitate matching significance between p-values and feature importance.*

Description

The XAI.test function complements t-test and correlation analyses in feature discovery by integrating eXplainable AI techniques such as feature importance, SHAP, LIME, or custom functions. It provides the option of automatic integration of simulated data to facilitate matching significance between p-values and feature importance.

Usage

```
XAI.test(
  data,
  y = "y",
  featImpAgr = "mean",
  simData = FALSE,
  simMethod = "regrnorm",
  simPvalTarget = 0.045,
  adjMethod = "bonferroni",
  customPVals = NULL,
  customFeatImps = NULL,
```

```
  customFIPV = NULL,
  modelType = "default",
  corMethod = "pearson",
  defaultMethods = c("ttest", "ebayes", "cor", "lm", "rf", "shap", "lime"),
  caretMethod = "rf",
  caretTrainArgs = NULL,
  verbose = FALSE
)
```

Arguments

data	SummarizedExperiment or dataframe containing the data. If dataframe rows are samples and columns are features.
y	Name of the SummarizedExperiment metadata or column of the dataframe containing the target variable. Default to "y".
featImpAgr	Can be "mean" or "max_abs". It defines how the feature importance is aggregated.
simData	If TRUE, a simulated feature column is added to the dataframe to target a defined p-value that will serve as a benchmark for determining the significance thresholds of feature importances.
simMethod	Method used to generate the simulated data. Can be "regrnorm" or "rnorm", "regnorm" by default. "regrnorm" creates simulated data points that match specific percentiles within a normal distribution, defined by a given mean and standard deviation. "rnorm" creates simulated data points that follow a normal distribution. "regrnorm is more accurate in targeting the specified p-value.
simPvalTarget	Target p-value for the simulated data. It is used to determine the significance thresholds of feature importances.
adjMethod	Method used to adjust the p-values. "bonferroni" by default, can be any other method available in the p.adjust function.
customPVals	List of custom functions that compute p-values. The functions must take the dataframe and the target variable as arguments and return a names list with: • 'pvals' => a dataframe with the p-values. • 'adjPVal' => a dataframe with the adjusted p-values. Optional. • 'model' => the prediction model object. Optional.
customFeatImps	List of custom functions that compute feature importances. The functions must take the dataframe and the target variable as arguments and return a names list with: • 'featImps' => a dataframe with the feature importances. The names of the functions will be used as the column names in the output dataframe. Mandatory. • 'model' => the predictionmodel object. Optional.
customFIPV	List of custom functions that compute feature importances and corresponding p-values. The functions must take the dataframe and the target variable as arguments and return a names list with: • 'featImps' => a dataframe with the feature importances.

- 'pvals' => a dataframe with the corresponding p-values.
- 'model' => the prediction model object. Optional.

`modelType` Type of the model. Can be "classification", "regression" or "default". If "default", the function will try to infer the model type from the target variable. If the target variable is a character, the model type will be "classification". If the target variable is numeric, the model type will be "regression".

`corMethod` Method used to compute the correlation between the features and the target variable. "pearson" by default, can be any other method available in the cor.test function.

`defaultMethods` List of default p-values and feature importances methods to compute. By default "ttest", "ebayes", "cor", "lm", "rf", "shap" and "lime".

`caretMethod` Method used by the caret package to train the model. "rf" by default.

`caretTrainArgs` List of arguments to pass to the caret::train function. Optional.

`verbose` If TRUE, the function will print messages to the console.

Details

The XAI.test function is designed to extend the capabilities of conventional statistical analysis methods for feature discovery, such as t-tests and correlation, by incorporating techniques from explainable AI (XAI), such as feature importance, SHAP, LIME, or custom functions. This function aims at identifying significant features that influence a given target variable in a dataset, supporting both categorical and numerical target values. A key feature of XAI.test is its ability to automatically incorporate simulated data into the analysis. This simulated data is specifically designed to establish significance thresholds for feature importance values based on the p-values. This capability is useful for reinforcing the reliability of the feature importance metrics derived from machine learning models, by directly comparing them with established statistical significance metrics.

Value

A dataframe containing the pvalues and the feature importances of each features computed by the different methods.

Examples

```
library(S4Vectors)
library(SummarizedExperiment)

# With a dataframe
data <- data.frame(
  feature1 = rnorm(100),
  feature2 = rnorm(100, mean = 5),
  feature3 = runif(100, min = 0, max = 10),
  feature4 = c(rnorm(50), rnorm(50, mean = 5)),
  y = c(rep("Cat1", 50), rep("Cat2", 50))
)

results <- XAI.test(data, y = "y", verbose = TRUE)
results
```

```
# With a SummarizedExperiment
assays <- SimpleList(counts = as.matrix(t(data[, 1:4])))
colData <- DataFrame(y = data[,"y"])
se <- SummarizedExperiment(assays = assays,
                           colData = colData)
results <- XAI.test(se, y = "y", verbose = TRUE)
results
```

Index