Package: rai (via r-universe)

October 25, 2024

Type Package

Title Revisiting-Alpha-Investing for Polynomial Regression

Version 1.0.1

Description A modified implementation of stepwise regression that greedily searches the space of interactions among features in order to build polynomial regression models. Furthermore, the hypothesis tests conducted are valid-post model selection due to the use of a revisiting procedure that implements an alpha-investing rule. As a result, the set of rejected sequential hypotheses is proven to control the marginal false discover rate. When not searching for polynomials, the package provides a statistically valid algorithm to run and terminate stepwise regression. For more information, see Johnson, Stine, and Foster (2019) <arXiv:1510.06322>.

License GPL-3

Encoding UTF-8

LazyData true

URL https://github.com/korydjohnson/rai

BugReports https://github.com/korydjohnson/rai/issues

Imports stats, dplyr, ggplot2, readr, rlang

Suggests testthat

RoxygenNote 7.0.2

Repository https://korydjohnson.r-universe.dev

RemoteUrl https://github.com/korydjohnson/rai

RemoteRef HEAD

RemoteSha b3eeaf76184c1077ca9a48d245e2a4821a4aad07

Contents

Auction *Internal function to manage multiple experts.*

Description

runAuction is the workhorse of the rai package: it takes an initial expert list and runs the Revisiting Alpha-Investing algorithm to greedily fit (optional) polynomials and interactions to data. The term "auction" is the result of multiple experts bidding to perform the test which determines stepwise ordering. This function is not intended to be called directly, but through `rai`.

Usage

```
vif(res, y, X, x, n, p, m, TSS, lmFit)

runAuction(
  experts,
  gWealth,
  theData,
  y,
  alg,
  poly,
  searchType,
  m,
  sigma,
  omega,
  reuse,
  nMaxTest,
  verbose,
  save,
  lmFit,
  baseModel
)
```

Arguments

`res`	residuals from current model.
`y`	the response as a single column matrix.

`X`	covariates in the current model.
`x`	covariate being tested for addition into the model.
`n`	number of observations.
`p`	number of predictors in the *current* model.
`m`	number of observations used in subsampling for variance inflation factor estimate of r.squared.
`TSS`	total sum of squares; considering current residuals to be the response.
`lmFit`	The core function that will be used to estimate linear model fits. The default is .lm.fit, but other alternatives are possible. Note that it does not use formula notation as this is costly. Another recommended option is fastLmPure from RcppEigen or related packages.
`experts`	list of expert objects. Each expert is the output of makeStepwiseExpert or makeScavengerExpert.
`gWealth`	global wealth object, output of gWealthStep.
`theData`	covariate matrix.
`alg`	algorithm can be one of "rai", "raiPlus", or "RH" (Revisiting Holm).
`poly`	logical. Should the algorithm look for higher-order polynomials?
`searchType`	A character string specifying the prioritization of higher-order polynomials. One of "breadth" (more base features) or "depth" (higher order).
`sigma`	type of error estimate used in gWealthStep; one of "ind" or "step".
`omega`	return from rejecting a test in Alpha-Investing.
`reuse`	logical. Should repeated tests of the same covariate be considered a test of the same hypothesis? Reusing wealth isn't implemented for RAI or RAIplus (effect is negligible).
`nMaxTest`	maximum number of tests
`verbose`	logical. Should auction output be printed?
`save`	logical. Should the auction results be saved? If TRUE, returns a summary matrix.
`baseModel`	Features to include as the initial model. When NULL, the base model only includes the intercept. baseModel must be specified as a list of desired features. Each list element is a vector of column names or indices, where vectors of length > 1 specify an interaction term of those features. Please check the transformed data using `prepareData` in order to determine the correct column names and indexes of your desired model.

Value

A list which includes the following components:

`formula`	final model formula.
`y`	response.
`X`	model matrix from final model.
`features`	list of interactions included in formula.
`summary`	included if save=TRUE; matrix where each row contains the summary information of a single test.

Bidders *Making Bidder Objects*

Description

These functions create objects that manage alpha-wealth. There is only one stepwise "bidder" that manages the global wealth (gWealth) but it can have multiple "offspring" when searching for polynomials. The outer `rai` function creates one gWealthStep object and one stepwise bidder at the beginning. The stepwise bidder makes a local modification to gWealth, though bidAccepted/bidRejected still call gWealth. More stepwise bidders are created as "scavengers" tied to the global wealth. Defaults are not set because these are internal functions called by `rai` and `runAuction` and all arguments are required.

Usage

```
gWealthStep(wealth, alg, r, TSS, p, reuse, rmse, df)

makeStepwiseBidder(gWealth)
```

Arguments

`wealth`	starting alpha-wealth.
`alg`	algorithm can be one of "rai", "raiPlus", or "RH" (Revisiting Holm).
`r`	RAI rejects tests which increase R^2 by a factor r^s, where s is the epoch.
`TSS`	total sum of squares of the response.
`p`	number of covariates (only used when alg == "RH").
`reuse`	logical. Should repeated tests of the same covariate be considered a test of the same hypothesis?
`rmse`	initial (or independent) estimate of residual standard error
`df`	degrees of freedom of rmse.
`gWealth`	a global wealth object; output of gWealthStep.

Value

A closure containing a list of functions.

Experts *Making Expert Objects*

Description

Experts are the "actors" which "bid" to see who conducts the next test. They contain an object "bidder" that determines bidding strategy and an object "constructor" that determines which feature it wants to text next. The `runAuction` function calls functions from experts and gWealth. The makeExpert function is not called directly, but through makeStepwiseExpert or makeScavengerExpert. Defaults are not set because these are internal functions called by `rai` and `runAuction` and all arguments are required.

Usage

```
makeExpert(bidder, constructor)

makeStepwiseExpert(gWealth, ncolumns)

makeScavengerExpert(gWealth, theModelFeatures, name)
```

Arguments

`bidder`	bidder object; output of makeStepwiseBidder.
`constructor`	constructor object; output of makeRawSource or makeLocalScavenger.
`gWealth`	global wealth object, output of gWealthStep.
`ncolumns`	number of features the constructor should manage, thought of as columns of the design matrix.
`theModelFeatures`	list of feature names in the model when the feature was rejected.
`name`	name of base feature used in interactions with other features in the model.

Value

A closure containing a list of functions.

Feature-Constructors *Making Source Objects*

Description

These functions create and manage the features to test. The raw source only tests marginal features (the covariates in the design matrix) while the scavenger source tests for interactions between a base feature and those features already in the model. makeLocalScavenger builds on makeRawSource. Defaults are not set because these are internal functions called by `rai` and `runAuction` and all arguments are required.

Usage

```
makeRawSource(ncolumns)

makeLocalScavenger(theModelFeatures, name)
```

Arguments

ncolumns	number of features this constructor should manage, thought of as columns of the design matrix.
theModelFeatures	other features currently in the model.
name	name of the base feature with which to create interactions.

Value

A closure containing a list of functions.

ProcessRAI *Summarising RAI Output*

Description

Processes the output from the rai function. Requires dplyr, tibble, and ggplot2 packages.

Usage

```
plot_ntest_rS(rawSum)

plot_ntest_wealth(rawSum)

## S3 method for class 'rai'
predict(object, newdata = NULL, alpha = NULL, omega = NULL, ...)

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

Arguments

rawSum	processed version of rai summary stored as a tibble with correct column parsing.
object	an object of class rai; expected to be the list output from the rai function.
newdata	an optional data frame in which to look for variables with which to predict. If omitted, the fitted values are used.
alpha	level of procedure.
omega	return from rejecting a test in Alpha-Investing (<= alpha).
...	additional arguments affecting the summary or predict methods.

Value

A list which includes the following components:

plot_rS	plot of the change in r.squared over time (number of tests conducted).
plot_wealth	plot of the change in r.squared over time (number of tests conducted).
experts	summary of expert performance: number of features, number of rejections, order in which they were added to the expert list.
tests	table of number of times features were tested: how many features tested k times; which expert(s) conducted tests.
epochs	in which epochs were tests rejected and the corresponding rejection thresholds.
stats	summary statistics: number of tests, number of epochs, bound on percentage reduction in ESS by adding a single feature, number of passes through to features, final r.squared, cost of raiPlus (0 for rai).
options	options given to RAI: algorithm, searchType, poly, startDegree, r.

Examples

```
data("CO2")
theResponse = CO2$uptake
theData = CO2[ ,-5]
rai_out = rai(theData, theResponse)
summary(rai_out)  # summary information including graphs
predict(rai_out)  # fitted values from selected model
```

RAI — *Main function for Revisiting Alpha-Investing (RAI) regression.*

Description

The function rai is a wrapper that creates and manages the inputs and outputs of the runAuction function. Using poly=FALSE is an efficient and statistically valid way to run and terminate stepwise regression. The function prepareData is provided in order to make generating predictions on test data easier: it is used by rai to process the data prior to running, and is necessary to make column names and information match in order to use the model object returned by rai.

Usage

```
prepareData(theData, poly = TRUE, startDeg = 1)

is.rai(x)

rai(
  theData,
  theResponse,
  alpha = 0.1,
  alg = "rai",
```

```
  r = 0.8,
  poly = alg != "RH",
  startDeg = 1,
  searchType = "breadth",
  m = 500,
  sigma = "step",
  rmse = NA,
  df = NA,
  omega = alpha,
  reuse = (alg == "RH"),
  maxTest = Inf,
  verbose = FALSE,
  save = TRUE,
  lmFit = .lm.fit,
  baseModel = NULL
)
```

Arguments

`theData`	matrix or data.frame of covariates.
`poly`	logical. Should the algorithm look for higher-order polynomials?
`startDeg`	This is the starting degree for polynomial regression. It allows the search to start with lower order polynomials such as square roots. This alleviates some problems with high-dimensional polynomials as a 4th degree polynomial where startDeg=1/2 is only a quadratic on the original scale.
`x`	an R object.
`theResponse`	response vector or single column matrix.
`alpha`	level of procedure.
`alg`	algorithm can be one of "rai", "raiPlus", or "RH" (Revisiting Holm).
`r`	threshold parameter, with 0 < r < 1. RAI rejects tests which increase remaining R^2 by a factor r^s, where s is the epoch. Larger values of r yield a closer approximation to stepwise regression.
`searchType`	A character string specifying the prioritization of higher-order polynomials. One of "breadth" (more base features) or "depth" (higher orders).
`m`	number of observations used in subsampling for variance inflation factor estimate of r.squared. Set m=Inf to use full data.
`sigma`	type of error estimate used; one of "ind" or "step". If "ind", you must provide a numeric value for rmse and df.
`rmse`	user provided value for rmse. Must be used with sigma="ind".
`df`	degrees of freedom for user specified rmse. Must be used with sigma="ind".
`omega`	return from rejecting a test in Alpha-Investing (<= alpha).
`reuse`	logical. Should repeated tests of the same covariate be considered a test of the same hypothesis? reusing wealth isn't implemented for RAI or RAIplus as the effect is negligible.

`maxTest`	maximum number of tests.
`verbose`	logical. Should auction output be printed?
`save`	logical. Should the auction results be saved? If TRUE, returns a summary matrix.
`lmFit`	The core function that will be used to estimate linear model fits. The default is .lm.fit, but other alternatives are possible. Note that it does not use formula notation as this is costly. Another recommended option is fastLmPure from RcppEigen or related packages.
`baseModel`	Features to include as the initial model. When NULL, the base model only includes the intercept. baseModel must be specified as a list of desired features. Each list element is a vector of column names or indices, where vectors of length > 1 specified an interaction term of those features. Please check the transformed data using `prepareData` in order to determine the correct column names and indexes of your desired model.

Details

Missing values are treated as follows: all observations with missing values in theResponse are removed; numeric columns in theData have missing values imputed by the mean of the column and an indicator column is added to note missingness; missing values in factor or binary columns are given the value "NA", which creates an additional group for missing values. Note that as rai is run using the output of model.matrix, it is not guaranteed that all categories from a factor are included in the regression. Column names may also be modified to be syntactically valid. The model object can be used to generate predictions on test data. Note that if default conversions were used when running rai, then they must be used again with prepareData for the test data prior to producing predictions.

Value

A list which includes the following components:

`y`	response.
`X`	model matrix from final model.
`formula`	final model formula.
`features`	list of interactions included in formula.
`summary`	if save=TRUE, contains information on each test made by the algorithm.
`time`	run time.
`options`	options given to RAI: alg, searchType, poly, r, startDeg, alpha, omega, m.
`subData`	subset of columns from theData that are used in the final model.
`model`	linear model object using selected model

Summary and predict methods are provided in order to generate further output and graphics.

Examples

```
data("CO2")
theResponse = CO2$uptake
theData = CO2[ ,-5]
rai_out = rai(theData, theResponse)
summary(rai_out)  # summary information including graphs
```

Index