Package 'stepseg'

May 23, 2020

Title Stepwise Segmented Regression Analysis

Version 0.9.0

Description This package facilitates stepwise segmented regression analysis, an approach for breakpoint detection and evaluation that is especially useful for data with high error variance (e.g., longitudinal behavioral data). If you use this package, please cite the original publication: Britt, B. C. (2015). Stepwise segmented regression analysis: An iterative statistical algorithm to detect and quantify evolutionary and revolutionary transformations in longitudinal data. In S. A. Matei, M. G. Russell, & E. Bertino (Eds.), Transparency in social media: Tools, methods, and algorithms for mediating online interactions (pp. 125-144). Heidelberg, Germany: Springer.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0

Imports stats

R topics documented:

print.stepseg_output 1
stepseg 2

Index **8**

`print.stepseg_output` *Print*

Description

This method prints the `output` element of an object with the `stepseg_output` class, such as output obtained from the `stepseg` function.

Usage

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

Arguments

`x`	An object with the `stepseg_output` class
`...`	Other arguments inherited from the generic `print` function

`stepseg` *Stepwise Segmented Regression Analysis*

Description

This function performs stepwise segmented regression analysis, as described by Britt (2015).

Usage

```
stepseg(
  dv,
  ivs = data.frame(1:nrow(data.frame(dv))),
  start_formula = NULL,
  add = 0.15,
  remove = 0.2,
  add_mode = "mse",
  family = stats::gaussian(link = "identity"),
  order = 1,
  allow_interactions = FALSE,
  update = 0,
  verbose = FALSE
)
```

Arguments

`dv`	A vector or data.frame representing values of the dependent variable, which is coerced to a data.frame if not provided as one
`ivs`	A data.frame representing values of the independent variables, which is coerced to a data.frame if not provided as one (default = `c(1:nrow(dv))`, which implicitly treats `dv` as sequentially ordered data and attempts to detect breakpoints in that sequence)
`start_formula`	A `formula` to be used for the regression model prior to the first stepwise model selection iteration, with all listed coefficients permanently retained in the model (default = `NULL`, which adds an intercept and a linear coefficient for each independent variable to the regression model)
`add`	A numeric atomic vector between `0` and `1` indicating the *p*-value threshold to add coefficients to the model during each forward selection iteration, which must be less than or equal to the `remove` argument in order to avoid infinite loops (default = `.15`, as recommended by Britt, 2015)
`remove`	A numeric atomic vector between `0` and `1` indicating the *p*-value threshold to remove coefficients from the model during each backward selection iteration, which must be greater than or equal to the `add` argument in order to avoid infinite loops (default = `.20`, as recommended by Britt, 2015)

`add_mode`	A character atomic vector (either `"p"` or `"mse"` indicating what criterion should be used to select the best candidate block during each forward selection iteration (default = `"mse"`)
`family`	A `family` object specifying the distribution and link function to be used in the general linear model (default = `stats::gaussian(link = "identity")`, which is appropriate for normally-distributed data and which facilitates a simple or multiple linear regression)
`order`	A non-negative numeric atomic vector indicating the maximum exponent that will be applied to coefficients added to the regression model (default = `1`; it is generally recommended to use either `order = 0` or `order = 1`)
`allow_interactions`	A boolean value indicating whether interactions may be created between indicator functions representing breakpoints along one independent variable with other independent variables in the model (default = `FALSE`, which restricts each indicator function to only interact with the independent variable along which its breakpoint was defined)
`update`	A numeric value indicating how many loop iterations should elapse between progress updates (default = `0`, which suppresses output)
`verbose`	A boolean value indicating whether the current regression model and results should be outputted as part of the progress report every `update` iterations (default = `FALSE`; note that if `update = 0`, no updates will be printed even if `verbose = TRUE`)

Details

When using stepwise segmented regression analysis, coefficients are added to and removed from the model in "blocks," with a given block consisting of the indicator function representing a given breakpoint location as well as all interaction terms between that indicator function and independent variables that are eligible to be added to the model.

The `order` and `allow_interactions` arguments jointly indicate what interaction terms are valid. `order` indicates the maximum exponent that can be applied to the independent variables, while `allow_interactions` indicates whether interaction terms can be created between a given independent variable and an indicator function that is defined by values of a different independent variable.

All interaction terms that are considered valid, with exponents ranging from `0` to `order`, are included in a given block. Consider, for instance, an analysis in which `ivs`, the data.frame representing the independent variables, has three columns signifying three variables. If `order = 1` and `allow_interactions = FALSE`, which are their default values in `stepseg`, then the block corresponding to a potential breakpoint at `ivs[, 1] = 10` would include two coefficients:

- `I(ivs[, 1] = 10)`
- `I(ivs[, 1] = 10):(ivs[, 1]^1)`

Setting `order = 2` would retain both of the aforementioned coefficients and also add

- `I(ivs[, 1] = 10):(ivs[, 1]^2)`

to the block. Maintaining `order = 2` and setting `allow_interactions = TRUE` would result in the block containing a total of seven coefficients:

- `ivs[, 1] = 10`
- `I(ivs[, 1] = 10):(ivs[, 1]^1)`
- `I(ivs[, 1] = 10):(ivs[, 1]^2)`

- `I(ivs[, 1] = 10):(ivs[, 2]^1)`
- `I(ivs[, 1] = 10):(ivs[, 2]^2)`
- `I(ivs[, 1] = 10):(ivs[, 3]^1)`
- `I(ivs[, 1] = 10):(ivs[, 3]^2)`

Note that if `order = 0`, then each block will only include the indicator function itself (in this example, `I(ivs[, 1] = 10)`) regardless of the value of `allow_interactions`.

During each forward selection iteration, the algorithm selects the "best" block of coefficients that can be added. If `add_mode = "p"`, then all possible candidate blocks that could be added to the model are evaluated, and whichever block has a *p*-value less than or equal to all *p*-values in all other blocks is treated as the "best" block. As long as that *p*-value is less than the threshold specified by the `add` argument, all coefficients included in the block are added to the model. If `add_mode = "mse"`, then the candidate block whose coefficients would jointly reduce the model mean squared error by the greatest amount is treated as the 'best' block, and all of its coefficients are added to the model if the resulting *p*-value for at least one of those coefficients would be less than `add`.

During each backward selection iteration, all coefficients listed in `start_formula` are retained in the regression model. Among all remaining coefficients, any blocks whose *p*-values are all greater than or equal to `remove` are removed from the model. If at least one *p*-value contained in a given block is less than `remove`, then none of the coefficients in the block are removed from the model.

When `order = 0`, `add_mode = "p"` is generally acceptable to follow common conventions of stepwise model selection. When `order > 0`, however, `add_mode = "p"` is more likely to result in spurious breakpoints being added to the model due to intercept and higher-order terms competing with one another, and in rare cases the algorithm may entirely fail to converge. `add_mode = "mse"` is more robust against these issues and is strongly recommended whenever `order > 0`.

Note that this function uses Type II sums of squares (via `drop1`) for parameter estimation, as Type I sums of squares (the default in `glm`) can result in an infinite loop. If Type I sums of squares for the final model are desired, the corresponding `glm` object can be retrieved via the `raw_glm` element of the output from `stepseg`.

The `start_formula` argument can be used to provide a regression model whose coefficients will be inputted prior to the first stepwise model selection iteration. However, only first-order independent variables should be listed in this formula. Higher-order terms are not outputted in a predictable manner by some of the external functions used called by `stepseg` and may lead to invalid results. If an interaction term is desired, you should create a new column in the data.frame submitted as the `ivs` argument to represent that interaction, and that column can subsequently be used in the analysis. Similarly, listing indicator functions representing breakpoints in this argument may lead to unexpected behavior during backward selection iterations of the `stepseg` function, so forcibly adding them to the model via the `start_formula` argument is not advised. For most use cases, this argument can and should be omitted.

The larger the value of `order`, the more likely spurious breakpoints are to emerge. As with other regression contexts, you should only increase the complexity of your model (such as by increasing the value of `order`) when you have a clear reason to do so. Moreover, whenever `order > 0`, singularities may occur that render some coefficients inestimable. For instance, if `order = 3` and a pair of breakpoints is identified along the same independent variable, with those two breakpoints occurring two data points apart, there will be insufficient data between the breakpoints to estimate the standard error of the interaction between the indicator function and a cubic term, so it will be reported as `NA`. This is normal, expected behavior in stepwise segmented regression analysis. In such cases, those `NA` coefficients can be treated as though they were absent from the model. The `cleaned_output`, `cleaned_formula`, and `cleaned_factors` elements of the output from `stepseg` remove any `NA` coefficients accordingly, while `cleaned_raw_glm` uses `cleaned_formula` in a `glm` function call that, like `raw_glm`, uses Type I sums of squares.

However, if `family = "binomial"` or `family = "poisson"`, any singularities will generate an error. This is due to a known limitation in `drop1`, which `stepseg` calls to generate statistics using Type II sums of squares for general linear models. A warning is generated when either of these two distribution families are used as input arguments.

Finally, use caution when setting `order > 1`, as increasingly high-order terms may compete with lower-order terms in the model and yield unpredictable results. Refer to Britt (2015) for more information.

Value

An object with the `stepseg_output` class that contains the following list elements:

`output`: The summary of the final regression model that resulted from the stepwise segmented regression analysis, constructed using `glm` and passed through `drop1`) to obtain results using Type II sums of squares

`raw_glm`: The raw output for the final regression model using `glm`, without passing it through `drop1`, thereby using Type I sums of squares rather than Type II

`cleaned_output`: The summary of the final regression model that resulted from the stepwise segmented regression analysis, removing any coefficients that had `NA` *p*-values due to singularities or other issues, constructed using `glm` and passed through `drop1`) to obtain results using Type II sums of squares

`cleaned_raw_glm`: The raw output for the final regression model using `glm`, removing any coefficients that had `NA` *p*-values due to singularities or other issues, but without passing it through `drop1`, thereby using Type I sums of squares rather than Type II

`start_formula`: The formula for the regression model prior to the first iteration of the stepwise segmented regression analysis

`final_formula`: The final formula for the regression model that resulted from the stepwise segmented regression analysis

`cleaned_formula`: The final formula for the regression model that resulted from the stepwise segmented regression analysis, removing any coefficients that had `NA` *p*-values due to singularities or other issues

`start_factors`: The coefficients used in the regression model prior to the first iteration of the stepwise segmented regression analysis

`final_factors`: The coefficients used in the regression model that resulted from the stepwise segmented regression analysis

`cleaned_factors`: The coefficients used in the regression model that resulted from the stepwise segmented regression analysis, removing any coefficients that had `NA` *p*-values due to singularities or other issues

`add_iterations`: The number of forward selection iterations elapsed

`remove_iterations`: The number of backward selection iterations elapsed

`model_iterations`: A list showing how the regression model appeared at the end of each forward selection and backward selection iteration

`dv`: The value set for the `dv` argument

`ivs`: The value set for the `ivs` argument

`add`: The value set for the `add` argument

`remove`: The value set for the `remove` argument

`add_mode`: The value set for the `add_mode` argument `family`: The value set for the `family` argument

`order`: The value set for the `order` argument

`allow_interactions`: The value set for the `allow_interactions` argument

`update`: The value set for the `update` argument

`verbose`: The value set for the `verbose` argument

References

Britt, B. C. (2015). Stepwise segmented regression analysis: An iterative statistical algorithm to detect and quantify evolutionary and revolutionary transformations in longitudinal data. In S. A. Matei, M. G. Russell, & E. Bertino (Eds.), *Transparency in social media: Tools, methods, and algorithms for mediating online interactions* (pp. 125-144). Heidelberg, Germany: Springer.

Examples

```
set.seed(123)
dv <- c(11:30, seq(70, 12, by = -2)) + (rnorm(50)/10)
results1 <- stepseg(dv)
print(results1)
# Single term deletions
#
# Model:
# dv ~ 1 + I(ivs[, 1]^1) + I(ivs[, 1] > 20) + I(ivs[, 1] > 20):I(ivs[,
#     1]^1)
#                                Df Deviance    AIC F value    Pr(>F)
# <none>                                0.4 -89.11
# I(ivs[, 1]^1)                   1    663.3 279.15   75604 < 2.2e-16 ***
# I(ivs[, 1] > 20)                1  12818.4 427.22 1461966 < 2.2e-16 ***
# I(ivs[, 1]^1):I(ivs[, 1] > 20)  1   4607.1 376.06  525415 < 2.2e-16 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

ivs <- cbind(c(1:10, 21:30, 1:30), c(1:40,50:41))
results2 <- stepseg(dv, ivs, order = 1, allow_interactions = TRUE, update = 2)
# "Beginning the stepwise procedure..."
# "Completed iteration #2"
# "Completed iteration #4"

print(results2)
# Single term deletions
#
# Model:
# dv ~ 1 + I(ivs[, 1]^1) + I(ivs[, 2]^1) + I(ivs[, 2] > 20) + I(ivs[,
#     2] > 20):I(ivs[, 1]^1) + I(ivs[, 2] > 20):I(ivs[, 2]^1) +
#     I(ivs[, 1] > 25) + I(ivs[, 1] > 25):I(ivs[, 1]^1) + I(ivs[,
#     1] > 25):I(ivs[, 2]^1)
#                                Df Deviance     AIC    F value    Pr(>F)
# <none>                               0.31 -92.709
# I(ivs[, 1]^1)                   1     0.32 -92.585 1.7790e+00  0.189631
# I(ivs[, 2]^1)                   1    28.33 131.492 3.7394e+03 < 2.2e-16 ***
# I(ivs[, 2] > 20)                1  1104.78 314.663 1.4737e+05 < 2.2e-16 ***
# I(ivs[, 1] > 25)                1     0.32 -92.518 1.8364e+00  0.182794
# I(ivs[, 1]^1):I(ivs[, 2] > 20)  1   182.43 224.610 2.4301e+04 < 2.2e-16 ***
# I(ivs[, 2]^1):I(ivs[, 2] > 20)  1    21.79 118.370 2.8668e+03 < 2.2e-16 ***
# I(ivs[, 1]^1):I(ivs[, 1] > 25)  1     0.31 -93.556 9.5640e-01  0.333834
# I(ivs[, 2]^1):I(ivs[, 1] > 25)  1     0.37 -85.378 8.4115e+00  0.005969 **
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

results3 <- stepseg(dv, ivs, order = 2, allow_interactions = FALSE,
  family = "poisson", update = 10) #Generates warnings due to non-Poisson DV
# "Beginning the stepwise procedure..."
# There were 50 or more warnings (use warnings() to see the first 50)

print(results3)
# Single term deletions
#
# Model:
# dv ~ 1 + I(ivs[, 1]^1) + I(ivs[, 1]^2) + I(ivs[, 2]^1) + I(ivs[,
#     2]^2) + I(ivs[, 2] > 20) + I(ivs[, 2] > 20):I(ivs[, 2]^1) +
#     I(ivs[, 2] > 20):I(ivs[, 2]^2)
#                                 Df Deviance AIC     LRT  Pr(>Chi)
# <none>                               0.8352 Inf
# I(ivs[, 1]^1)                    1   6.3978 Inf  5.5627 0.0183474 *
# I(ivs[, 1]^2)                    1  13.0428 Inf 12.2076 0.0004759 ***
# I(ivs[, 2]^1)                    1   1.7744 Inf  0.9392 0.3324749
# I(ivs[, 2]^2)                    1   5.8674 Inf  5.0322 0.0248799 *
# I(ivs[, 2] > 20)                 1  16.5465 Inf 15.7114 7.378e-05 ***
# I(ivs[, 2]^1):I(ivs[, 2] > 20)  1  12.8583 Inf 12.0231 0.0005255 ***
# I(ivs[, 2]^2):I(ivs[, 2] > 20)  1   3.7525 Inf  2.9174 0.0876314 .
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Index

drop1, *4, 5*

family, *3*
formula, *2*

glm, *4, 5*

print.stepseg_output, 1

stepseg, *1*, 2