Package 'butcher'

December 9, 2025

Title Model Butcher

Version 0.4.0

Description Provides a set of S3 generics to axe components of fitted model objects and help reduce the size of model objects saved to disk.

License MIT + file LICENSE

URL https://butcher.tidymodels.org/, https://github.com/tidymodels/butcher

BugReports https://github.com/tidymodels/butcher/issues

Depends R (>= 4.1.0)

Imports cli (>= 3.3.0), lobstr (>= 1.1.2), methods, purrr (>= 0.3.4), rlang (>= 1.0.2), tibble (>= 3.1.7), utils, vctrs (>= 0.4.1)

Suggests C50, caret, censored, ClusterR, clustMixType, covr, dbarts, ddalpha, dplyr, e1071, earth, flexsurv, fs, ipred, kernlab, kknn, klaR, knitr, MASS, mda, mgcv, modeldata, nnet, parsnip (>= 1.3.3), pkgload, pls, QSARdata, randomForest, ranger, RANN, recipes (>= 0.2.0), rmarkdown, rpart, rsample, RSpectra, sparklyr, survival (>= 3.2-10), tabnet, testthat (>= 3.0.0), torch, TH.data, tune, usethis (>= 1.5.0), workflowsets, xgboost (>= 1.3.2.1), xrf

VignetteBuilder knitr

Config/Needs/check bioc::mixOmics

Config/Needs/website tidyverse/tidytemplate

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation no

Author Joyce Cahoon [aut] (ORCID: <https://orcid.org/0000-0001-7217-4702>), Davis Vaughan [aut], Max Kuhn [cre, aut] (ORCID: <https://orcid.org/0000-0003-2402-136X>), Alex Hayes [aut],

Julia Silge [aut] (ORCID: <https://orcid.org/0000-0002-3671-836X>),
Posit Software, PBC [cph, fnd] (ROR: <https://ror.org/03wc8by49>)

Maintainer Max Kuhn <max@posit.co>

Repository CRAN

Date/Publication 2025-12-09 06:10:57 UTC

Contents

`axe-bart` *Axing a bart model.*

Description

Axing a bart model.

Usage

```
## S3 method for class 'bart'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'bart'
axe_fitted(x, verbose = FALSE, ...)
```

Arguments

`x`	A model object.
`verbose`	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is `FALSE`.
`...`	Any additional arguments related to axing.

Value

Axed bart object.

Examples

```
library(dbarts)
x <- dbarts::bart(mtcars[,2:5], mtcars[,1], verbose = FALSE, keeptrees = TRUE)
res <- butcher(x, verbose = TRUE)
```

axe-C5.0 *Axing a C5.0.*

Description

C5.0 objects are created from the C50 package, which provides an interface to the C5.0 classification model. The models that can be generated include basic tree-based models as well as rule-based models.

Usage

```
## S3 method for class 'C5.0'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'C5.0'
axe_ctrl(x, verbose = FALSE, ...)

## S3 method for class 'C5.0'
axe_fitted(x, verbose = FALSE, ...)
```

Arguments

x	A model object.
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Any additional arguments related to axing.

Value

Axed C5.0 object.

Examples

```
# Load libraries
library(parsnip)
library(rsample)
library(rpart)

# Load data
set.seed(1234)
split <- initial_split(kyphosis, prop = 9/10)
spine_train <- training(split)

# Create model and fit
c5_fit <- decision_tree(mode = "classification") |>
  set_engine("C5.0") |>
  fit(Kyphosis ~ ., data = spine_train)
```

```
out <- butcher(c5_fit, verbose = TRUE)

# Try another model from parsnip
c5_fit2 <- boost_tree(mode = "classification", trees = 100) |>
  set_engine("C5.0") |>
  fit(Kyphosis ~ ., data = spine_train)
out <- butcher(c5_fit2, verbose = TRUE)

# Create model object from original library
library(C50)
library(modeldata)
data(mlc_churn)
c5_fit3 <- C5.0(x = mlc_churn[, -20], y = mlc_churn$churn)
out <- butcher(c5_fit3, verbose = TRUE)
```

---

axe-coxph *Axing a coxph.*

Description

Axing a coxph.

Usage

```
## S3 method for class 'coxph'
axe_env(x, verbose = FALSE, ...)

## S3 method for class 'coxph'
axe_data(x, verbose = FALSE, ...)
```

Arguments

x	A model object.
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Any additional arguments related to axing.

Details

The survival::coxph() model is unique in how it uses environments in its components, and butchering such an object can behave in surprising ways in any environment other than the global environment (such as when wrapped in a function). We do not recommend that you use butcher() with a coxph object anywhere other than the global environment.

Do this:

```
my_coxph_func <- function(df) {
    coxph(Surv(time, status) ~ x + strata(covar), df)
}
## in global environment only:
butcher(my_coxph_func(df))
```

Do *not* do this:

```
my_coxph_func <- function(df) {
    res <- coxph(Surv(time, status) ~ x + strata(covar), df)
    ## no:
    butcher(res)
}

## will not work correctly:
my_coxph_func(df)
```

Value

Axed coxph object.

Examples

```
library(survival)

example_data <-
  tibble::tibble(
    time = rpois(1000, 2) + 1,
    status = rbinom(1000, 1, .5),
    x = rpois(1000, .5),
    covar = rbinom(1000, 1, .5)
  )

example_data

make_big_model <- function() {
  boop <- runif(1e6)
  coxph(Surv(time, status) ~ x + strata(covar), example_data)
}

res <- make_big_model()

weigh(res)
weigh(butcher(res))
```

axe-earth *Axing an earth object.*

Description

earth objects are created from the **earth** package, which is leveraged to do multivariate adaptive regression splines.

Usage

```
## S3 method for class 'earth'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'earth'
axe_data(x, verbose = FALSE, ...)

## S3 method for class 'earth'
axe_fitted(x, verbose = FALSE, ...)
```

Arguments

x	A model object.
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Any additional arguments related to axing.

Value

Axed earth object.

Examples

```
# Load libraries
library(parsnip)

# Create model and fit
earth_fit <- mars(mode = "regression") |>
  set_engine("earth") |>
  fit(Volume ~ ., data = trees)

out <- butcher(earth_fit, verbose = TRUE)

# Another earth model object
suppressWarnings(suppressMessages(library(earth)))
earth_mod <- earth(Volume ~ ., data = trees)
out <- butcher(earth_mod, verbose = TRUE)
```

`axe-elnet` *Axing an elnet.*

Description

elnet objects are created from the **glmnet** package, leveraged to fit generalized linear models via penalized maximum likelihood.

Usage

```
## S3 method for class 'elnet'
axe_call(x, verbose = FALSE, ...)
```

Arguments

`x`	A model object.
`verbose`	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is `FALSE`.
`...`	Any additional arguments related to axing.

Value

Axed model object.

Examples

```
# Load libraries
library(parsnip)
library(rsample)

# Load data
split <- initial_split(mtcars, prop = 9/10)
car_train <- training(split)

# Create model and fit
elnet_fit <- linear_reg(mixture = 0, penalty = 0.1) |>
  set_engine("glmnet") |>
  fit_xy(x = car_train[, 2:11], y = car_train[, 1, drop = FALSE])

out <- butcher(elnet_fit, verbose = TRUE)
```

`axe-flexsurvreg` *Axing an flexsurvreg.*

Description

flexsurvreg objects are created from the **flexsurv** package. They differ from survreg in that the fitted models are not limited to certain parametric distributions. Users can define their own distribution, or leverage distributions like the generalized gamma, generalized F, and the Royston-Parmar spline model.

Usage

```
## S3 method for class 'flexsurvreg'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'flexsurvreg'
axe_env(x, verbose = FALSE, ...)
```

Arguments

`x`	A model object.
`verbose`	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is `FALSE`.
`...`	Any additional arguments related to axing.

Value

Axed flexsurvreg object.

Examples

```
# Load libraries
library(parsnip)
library(censored)
library(flexsurv)

# Create model and fit
flexsurvreg_fit <- survival_reg(dist = "gengamma") |>
  set_engine("flexsurv") |>
  set_mode("censored regression") |>
  fit(Surv(Tstart, Tstop, status) ~ trans, data = bosms3)

out <- butcher(flexsurvreg_fit, verbose = TRUE)

# Another flexsurvreg model object
wrapped_flexsurvreg <- function() {
  some_junk_in_environment <- runif(1e6)
  fit <- flexsurvreg(Surv(futime, fustat) ~ 1,
```

```
                      data = ovarian, dist = "weibull")
  return(fit)
}

out <- butcher(wrapped_flexsurvreg(), verbose = TRUE)
```

`axe-formula` *Axing formulas.*

Description

formulas might capture an environment from the modeling development process that carries objects that will not be used for any post- estimation activities.

Usage

```
## S3 method for class 'formula'
axe_env(x, verbose = FALSE, ...)
```

Arguments

`x`	A model object.
`verbose`	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is `FALSE`.
`...`	Any additional arguments related to axing.

Value

Axed formula object.

Examples

```
wrapped_formula <- function() {
  some_junk_in_environment <- runif(1e6)
  ex <- as.formula(paste("y ~", paste(LETTERS, collapse = "+")))
  return(ex)
}

lobstr::obj_size(wrapped_formula())
lobstr::obj_size(butcher(wrapped_formula()))

wrapped_quosure <- function() {
  some_junk_in_environment <- runif(1e6)
  out <- rlang::quo(x)
  return(out)
}
lobstr::obj_size(wrapped_quosure())
lobstr::obj_size(butcher(wrapped_quosure))
```

`axe-function` *Axing functions.*

Description

Functions stored in model objects often have heavy environments and bytecode attached. To avoid breaking any post-estimation functions on the model object, the `butchered_function` class is not appended.

Usage

```
## S3 method for class '`function`'
axe_env(x, verbose = FALSE, ...)
```

Arguments

`x`	A model object.
`verbose`	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is `FALSE`.
`...`	Any additional arguments related to axing.

Value

Axed function.

Examples

```
# Load libraries
library(caret)

data(iris)
train_data <- iris[, 1:4]
train_classes <- iris[, 5]

train_fit <- train(train_data, train_classes,
                   method = "knn",
                   preProcess = c("center", "scale"),
                   tuneLength = 10,
                   trControl = trainControl(method = "cv"))

out <- axe_env(train_fit$modelInfo$prob, verbose = TRUE)
out <- axe_env(train_fit$modelInfo$levels, verbose = TRUE)
out <- axe_env(train_fit$modelInfo$predict, verbose = TRUE)
```

axe-gam *Axing a gam.*

Description

gam objects are created from the **mgcv** package.

Usage

```
## S3 method for class 'gam'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'gam'
axe_ctrl(x, verbose = FALSE, ...)

## S3 method for class 'gam'
axe_data(x, verbose = FALSE, ...)

## S3 method for class 'gam'
axe_env(x, verbose = FALSE, ...)

## S3 method for class 'gam'
axe_fitted(x, verbose = FALSE, ...)
```

Arguments

x	A model object.
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Any additional arguments related to axing.

Value

Axed gam object.

Examples

```
cars_gam <- mgcv::gam(mpg ~ s(disp, k = 3) + s(wt), data = mtcars)
cleaned_gam <- butcher(cars_gam, verbose = TRUE)
```

axe-gausspr *Axing a gausspr.*

Description

gausspr objects are created from **kernlab** package, which provides a means to do classification, regression, clustering, novelty detection, quantile regression and dimensionality reduction. Since fitted model objects from **kernlab** are S4, the butcher_gausspr class is not appended.

Usage

```
## S3 method for class 'gausspr'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'gausspr'
axe_data(x, verbose = FALSE, ...)

## S3 method for class 'gausspr'
axe_env(x, verbose = FALSE, ...)

## S3 method for class 'gausspr'
axe_fitted(x, verbose = FALSE, ...)
```

Arguments

x	A model object.
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Any additional arguments related to axing.

Value

Axed gausspr object.

Examples

```
library(kernlab)

test <- gausspr(Species ~ ., data = iris, var = 2)

out <- butcher(test, verbose = TRUE)

# Example with simulated regression data
x <- seq(-20, 20, 0.1)
y <- sin(x)/x + rnorm(401, sd = 0.03)
test2 <- gausspr(x, y)
out <- butcher(test2, verbose = TRUE)
```

axe-glm *Axing a glm.*

Description

glm objects are created from the base **stats** package.

Usage

```
## S3 method for class 'glm'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'glm'
axe_data(x, verbose = FALSE, ...)

## S3 method for class 'glm'
axe_env(x, verbose = FALSE, ...)

## S3 method for class 'glm'
axe_fitted(x, verbose = FALSE, ...)
```

Arguments

x	A model object.
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Any additional arguments related to axing.

Value

Axed glm object.

Examples

```
cars_glm <- glm(mpg ~ ., data = mtcars)
cleaned_glm <- butcher(cars_glm, verbose = TRUE)
```

`axe-glmnet` *Axing a glmnet.*

Description

glmnet objects are created from the **glmnet** package, leveraged to fit generalized linear models via penalized maximum likelihood.

Usage

```
## S3 method for class 'glmnet'
axe_call(x, verbose = FALSE, ...)
```

Arguments

`x`	A model object.
`verbose`	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is `FALSE`.
`...`	Any additional arguments related to axing.

Value

Axed glmnet object.

Examples

```
library(parsnip)

# Wrap a parsnip glmnet model
wrapped_parsnip_glmnet <- function() {
  some_junk_in_environment <- runif(1e6)
  model <- logistic_reg(penalty = 10, mixture = 0.1) |>
    set_engine("glmnet") |>
    fit(as.factor(vs) ~ ., data = mtcars)
  return(model$fit)
}

out <- butcher(wrapped_parsnip_glmnet(), verbose = TRUE)
```

axe-ipred *Axing a bagged tree.*

Description

*_bagg objects are created from the **ipred** package, which is used for bagging classification, regression and survival trees.

Usage

```
## S3 method for class 'regbagg'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'classbagg'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'survbagg'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'regbagg'
axe_ctrl(x, verbose = FALSE, ...)

## S3 method for class 'classbagg'
axe_ctrl(x, verbose = FALSE, ...)

## S3 method for class 'survbagg'
axe_ctrl(x, verbose = FALSE, ...)

## S3 method for class 'regbagg'
axe_data(x, verbose = FALSE, ...)

## S3 method for class 'classbagg'
axe_data(x, verbose = FALSE, ...)

## S3 method for class 'survbagg'
axe_data(x, verbose = FALSE, ...)

## S3 method for class 'regbagg'
axe_env(x, verbose = FALSE, ...)

## S3 method for class 'classbagg'
axe_env(x, verbose = FALSE, ...)

## S3 method for class 'survbagg'
axe_env(x, verbose = FALSE, ...)
```

Arguments

x	A model object.
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Any additional arguments related to axing.

Value

Axed *_bagg object.

Examples

```
library(ipred)

fit_mod <- function() {
  boop <- runif(1e6)
  bagging(y ~ x, data.frame(y = rnorm(1e4), x = rnorm(1e4)))
}

mod_fit <- fit_mod()
mod_res <- butcher(mod_fit)

weigh(mod_fit)
weigh(mod_res)
```

---

axe-kknn *Axing an kknn.*

Description

kknn objects are created from the **kknn** package, which is utilized to do weighted k-Nearest Neighbors for classification, regression and clustering.

Usage

```
## S3 method for class 'kknn'
axe_env(x, verbose = FALSE, ...)

## S3 method for class 'kknn'
axe_fitted(x, verbose = FALSE, ...)
```

Arguments

x	A model object.
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Any additional arguments related to axing.

Value

Axed kknn object.

Examples

```
# Load libraries
library(parsnip)
library(rsample)
library(rpart)
library(kknn)

# Load data
set.seed(1234)
split <- initial_split(kyphosis, prop = 9/10)
spine_train <- training(split)

# Create model and fit
kknn_fit <- nearest_neighbor(mode = "classification",
                             neighbors = 3,
                             weight_func = "gaussian",
                             dist_power = 2) |>
  set_engine("kknn") |>
  fit(Kyphosis ~ ., data = spine_train)

out <- butcher(kknn_fit, verbose = TRUE)


# Another kknn model object
m <- dim(iris)[1]
val <- sample(1:m,
              size = round(m/3),
              replace = FALSE,
              prob = rep(1/m, m))
iris.learn <- iris[-val,]
iris.valid <- iris[val,]
kknn_fit <- kknn(Species ~ .,
                 iris.learn,
                 iris.valid,
                 distance = 1,
                 kernel = "triangular")
out <- butcher(kknn_fit, verbose = TRUE)
```

`axe-KMeansCluster` *Axing a KMeansCluster.*

Description

Axing a KMeansCluster.

Usage

```
## S3 method for class 'KMeansCluster'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'KMeansCluster'
axe_fitted(x, verbose = FALSE, ...)
```

Arguments

x	A model object.
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Any additional arguments related to axing.

Value

Axed KMeansCluster object.

Examples

```
library(ClusterR)
data(dietary_survey_IBS)
dat <- scale(dietary_survey_IBS[, -ncol(dietary_survey_IBS)])
km <- KMeans_rcpp(dat, clusters = 2, num_init = 5)
out <- butcher(km, verbose = TRUE)
```

---

axe-kproto *Axing a kproto.*

Description

Axing a kproto.

Usage

```
## S3 method for class 'kproto'
axe_data(x, verbose = FALSE, ...)

## S3 method for class 'kproto'
axe_fitted(x, verbose = FALSE, ...)
```

Arguments

x	A model object.
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Any additional arguments related to axing.

Value

Axed kproto object.

Examples

```
library(clustMixType)

kproto_fit <- kproto(
  ToothGrowth,
  k = 2,
  lambda = lambdaest(ToothGrowth),
  verbose = FALSE
)

out <- butcher(kproto_fit, verbose = TRUE)
```

---

axe-ksvm *Axing a ksvm object.*

Description

ksvm objects are created from **kernlab** package, which provides a means to do classification, regression, clustering, novelty detection, quantile regression and dimensionality reduction. Since fitted model objects from **kernlab** are S4, the `butcher_ksvm` class is not appended.

Usage

```
## S3 method for class 'ksvm'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'ksvm'
axe_data(x, verbose = FALSE, ...)

## S3 method for class 'ksvm'
axe_fitted(x, verbose = FALSE, ...)
```

Arguments

`x`	A model object.
`verbose`	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is `FALSE`.
`...`	Any additional arguments related to axing.

Value

Axed ksvm object.

Examples

```
# Load libraries
library(parsnip)
library(kernlab)

# Load data
data(spam)

# Create model and fit
ksvm_class <- svm_poly(mode = "classification") |>
  set_engine("kernlab") |>
  fit(type ~ ., data = spam)

out <- butcher(ksvm_class, verbose = TRUE)
```

---

axe-lm *Axing an lm.*

Description

lm objects are created from the base **stats** package.

Usage

```
## S3 method for class 'lm'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'lm'
axe_env(x, verbose = FALSE, ...)

## S3 method for class 'lm'
axe_fitted(x, verbose = FALSE, ...)
```

Arguments

x	A model object.
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Any additional arguments related to axing.

Value

Axed lm object.

Examples

```
# Load libraries
library(parsnip)
library(rsample)

# Load data
split <- initial_split(mtcars, prop = 9/10)
car_train <- training(split)

# Create model and fit
lm_fit <- linear_reg() |>
  set_engine("lm") |>
  fit(mpg ~ ., data = car_train)

out <- butcher(lm_fit, verbose = TRUE)

# Another lm object
wrapped_lm <- function() {
  some_junk_in_environment <- runif(1e6)
  fit <- lm(mpg ~ ., data = mtcars)
  return(fit)
}

# Remove junk
cleaned_lm <- axe_env(wrapped_lm(), verbose = TRUE)

# Check size
lobstr::obj_size(cleaned_lm)

# Compare environment in terms component
lobstr::obj_size(attr(wrapped_lm()$terms, ".Environment"))
lobstr::obj_size(attr(cleaned_lm$terms, ".Environment"))
```

`axe-mass` *Axing a MASS discriminant analysis object.*

Description

lda and qda objects are created from the **MASS** package, leveraged to carry out linear discriminant analysis and quadratic discriminant analysis, respectively.

Usage

```
## S3 method for class 'lda'
axe_env(x, verbose = FALSE, ...)

## S3 method for class 'qda'
axe_env(x, verbose = FALSE, ...)
```

```
## S3 method for class 'polr'
axe_env(x, verbose = FALSE, ...)
```

Arguments

x	A model object.
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Any additional arguments related to axing.

Value

Axed lda or qda object.

Examples

```
library(MASS)

fit_da <- function(fit_fn) {
  boop <- runif(1e6)
  fit_fn(y ~ x, data.frame(y = rep(letters[1:4], 10000), x = rnorm(40000)))
}

lda_fit <- fit_da(lda)
qda_fit <- fit_da(qda)

lda_fit_b <- butcher(lda_fit)
qda_fit_b <- butcher(qda_fit)

weigh(lda_fit)
weigh(lda_fit_b)

weigh(qda_fit)
weigh(qda_fit_b)

wrapped_polr <- function(fit_fn) {
  boop <- runif(1e6)
  fit <- fit_fn(Sat ~ Infl + Type + Cont, weights = Freq, data = housing)
  return(fit)
}
polr_fit <- wrapped_polr(polr)
polr_fit_b <- butcher(polr_fit)
weigh(polr_fit)
weigh(polr_fit_b)
```

axe-mda *Axing a mda.*

Description

mda and fda objects are created from the **mda** package, leveraged to carry out mixture discriminant analysis and flexible discriminat analysis, respectively.

Usage

```
## S3 method for class 'mda'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'fda'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'mda'
axe_env(x, verbose = FALSE, ...)

## S3 method for class 'fda'
axe_env(x, verbose = FALSE, ...)

## S3 method for class 'mda'
axe_fitted(x, verbose = FALSE, ...)

## S3 method for class 'fda'
axe_fitted(x, verbose = FALSE, ...)
```

Arguments

x	A model object.
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Any additional arguments related to axing.

Value

Axed mda object.

Examples

```
library(mda)

mtcars$cyl <- as.factor(mtcars$cyl)

fit <- mda(cyl ~ ., data = mtcars)
out <- butcher(fit, verbose = TRUE)
```

```
fit2 <- fda(cyl ~ ., data = mtcars)
out2 <- butcher(fit2, verbose = TRUE)

# Another mda object
data(glass)
wrapped_mda <- function(fit_fn) {
  some_junk_in_environment <- runif(1e6)
  fit <- fit_fn(Type ~ ., data = glass)
  return(fit)
}

lobstr::obj_size(wrapped_mda(mda))
lobstr::obj_size(butcher(wrapped_mda(mda)))

lobstr::obj_size(wrapped_mda(fda))
lobstr::obj_size(butcher(wrapped_mda(fda)))
```

`axe-model_fit` *Axing an model_fit.*

Description

model_fit objects are created from the `parsnip` package.

Usage

```
## S3 method for class 'model_fit'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'model_fit'
axe_ctrl(x, verbose = FALSE, ...)

## S3 method for class 'model_fit'
axe_data(x, verbose = FALSE, ...)

## S3 method for class 'model_fit'
axe_env(x, verbose = FALSE, ...)

## S3 method for class 'model_fit'
axe_fitted(x, verbose = FALSE, ...)
```

Arguments

`x`	A model object.
`verbose`	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is `FALSE`.
`...`	Any additional arguments related to axing.

Value

Axed model_fit object.

Examples

```
library(parsnip)
library(rpart)

# Create model and fit
lm_fit <- linear_reg() |>
  set_engine("lm") |>
  fit(mpg ~ ., data = mtcars)

out <- butcher(lm_fit, verbose = TRUE)

# Another parsnip model
gam_fit <- gen_additive_mod() |>
  set_mode("regression") |>
  fit(mpg ~ s(disp) + wt + gear, data = mtcars)

out <- butcher(gam_fit, verbose = TRUE)
```

---

axe-multnet *Axing an multnet.*

Description

multnet objects are created from carrying out multinomial regression in the **glmnet** package.

Usage

```
## S3 method for class 'multnet'
axe_call(x, verbose = FALSE, ...)
```

Arguments

x	A model object.
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Any additional arguments related to axing.

Value

Axed multnet object.

Examples

```
# Load libraries
library(parsnip)

# Load data
set.seed(1234)
predictrs <- matrix(rnorm(100*20), ncol = 20)
colnames(predictrs) <- paste0("a", seq_len(ncol(predictrs)))
response <- as.factor(sample(1:4, 100, replace = TRUE))

# Create model and fit
multnet_fit <- multinom_reg(penalty = 0.1) |>
  set_engine("glmnet") |>
  fit_xy(x = predictrs, y = response)

out <- butcher(multnet_fit, verbose = TRUE)
```

---

axe-NaiveBayes *Axing a NaiveBayes.*

Description

NaiveBayes objects are created from the **klaR** package, leveraged to fit a Naive Bayes Classifier.

Usage

```
## S3 method for class 'NaiveBayes'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'NaiveBayes'
axe_data(x, verbose = FALSE, ...)
```

Arguments

x	A model object.
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Any additional arguments related to axing.

Value

Axed NaiveBayes object.

Examples

```
library(klaR)

fit_mod <- function() {
  boop <- runif(1e6)
  NaiveBayes(
    y ~ x,
    data = data.frame(y = as.factor(rep(letters[1:4], 1e4)), x = rnorm(4e4))
  )
}

mod_fit <- fit_mod()
mod_res <- butcher(mod_fit)

weigh(mod_fit)
weigh(mod_res)
```

---

axe-nnet *Axing a nnet.*

Description

nnet objects are created from the **nnet** package, leveraged to fit multilayer perceptron models.

Usage

```
## S3 method for class 'nnet'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'nnet'
axe_env(x, verbose = FALSE, ...)

## S3 method for class 'nnet'
axe_fitted(x, verbose = FALSE, ...)
```

Arguments

x	A model object.
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Any additional arguments related to axing.

Value

Axed nnet object.

Examples

```
# Load libraries
library(parsnip)
library(nnet)

# Create and fit model
nnet_fit <- mlp("classification", hidden_units = 2) |>
  set_engine("nnet") |>
  fit(Species ~ ., data = iris)

out <- butcher(nnet_fit, verbose = TRUE)

# Another nnet object
targets <- class.ind(c(rep("setosa", 50),
                       rep("versicolor", 50),
                       rep("virginica", 50)))

fit <- nnet(iris[,1:4],
            targets,
            size = 2,
            rang = 0.1,
            decay = 5e-4,
            maxit = 20)

out <- butcher(fit, verbose = TRUE)
```

---

axe-pls *Axing mixOmics models*

Description

`mixo_pls` (via `pls()`), `mixo_spls` (via `spls()`), and `mixo_plsda` (via `plsda()`) objects are created with the mixOmics package, leveraged to fit partial least squares models.

Usage

```
## S3 method for class 'mixo_pls'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'mixo_spls'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'mixo_pls'
axe_data(x, verbose = FALSE, ...)

## S3 method for class 'mixo_spls'
axe_data(x, verbose = FALSE, ...)
```

```
## S3 method for class 'mixo_pls'
axe_fitted(x, verbose = FALSE, ...)

## S3 method for class 'mixo_spls'
axe_fitted(x, verbose = FALSE, ...)
```

Arguments

x	A model object.
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Any additional arguments related to axing.

Details

The mixOmics package is not available on CRAN, but can be installed from the Bioconductor repository via `remotes::install_bioc("mixOmics")`.

Value

Axed `mixo_pls`, `mixo_spls`, or `mixo_plsda` object.

Examples

```
library(butcher)
do.call(library, list(package = "mixOmics"))

# pls ----------------------------------------------------------------
fit_mod <- function() {
  boop <- runif(1e6)
  pls(matrix(rnorm(2e4), ncol = 2), rnorm(1e4), mode = "classic")
}

mod_fit <- fit_mod()
mod_res <- butcher(mod_fit)

weigh(mod_fit)
weigh(mod_res)

new_data <- matrix(1:2, ncol = 2)
colnames(new_data) <- c("X1", "X2")
predict(mod_fit, new_data)
predict(mod_res, new_data)
```

`axe-randomForest` *Axing an randomForest.*

Description

randomForest objects are created from the `randomForest` package, which is used to train random forests based on Breiman's 2001 work. The package supports ensembles of classification and regression trees.

Usage

```
## S3 method for class 'randomForest'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'randomForest'
axe_ctrl(x, verbose = FALSE, ...)

## S3 method for class 'randomForest'
axe_env(x, verbose = FALSE, ...)
```

Arguments

`x`	A model object.
`verbose`	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is `FALSE`.
`...`	Any additional arguments related to axing.

Value

Axed randomForest object.

Examples

```
# Load libraries
library(parsnip)
library(rsample)
library(randomForest)
data(kyphosis, package = "rpart")

# Load data
set.seed(1234)
split <- initial_split(kyphosis, prop = 9/10)
spine_train <- training(split)

# Create model and fit
randomForest_fit <- rand_forest(mode = "classification",
                                mtry = 2,
                                trees = 2,
```

```
                                 min_n = 3) |>
  set_engine("randomForest") |>
  fit_xy(x = spine_train[,2:4], y = spine_train$Kyphosis)

out <- butcher(randomForest_fit, verbose = TRUE)

# Another randomForest object
wrapped_rf <- function() {
  some_junk_in_environment <- runif(1e6)
  randomForest_fit <- randomForest(mpg ~ ., data = mtcars)
  return(randomForest_fit)
}

# Remove junk
cleaned_rf <- axe_env(wrapped_rf(), verbose = TRUE)

# Check size
lobstr::obj_size(cleaned_rf)
```

`axe-ranger` *Axing an ranger.*

Description

ranger objects are created from the **ranger** package, which is used as a means to quickly train random forests. The package supports ensembles of classification, regression, survival and probability prediction trees. Given the reliance of post processing functions on the model object, like `importance_pvalues` and `treeInfo`, on the first class listed, the `butcher_ranger` class is not appended.

Usage

```
## S3 method for class 'ranger'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'ranger'
axe_fitted(x, verbose = FALSE, ...)
```

Arguments

`x`	A model object.
`verbose`	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is `FALSE`.
`...`	Any additional arguments related to axing.

Value

Axed ranger object.

Examples

```
# Load libraries
library(parsnip)
library(rsample)
library(ranger)

# Load data
set.seed(1234)
split <- initial_split(iris, prop = 9/10)
iris_train <- training(split)

# Create model and fit
ranger_fit <- rand_forest(mode = "classification",
                          mtry = 2,
                          trees = 20,
                          min_n = 3) |>
  set_engine("ranger") |>
  fit(Species ~ ., data = iris_train)

out <- butcher(ranger_fit, verbose = TRUE)

# Another ranger object
wrapped_ranger <- function() {
  n <- 100
  p <- 400
  dat <- data.frame(y = factor(rbinom(n, 1, .5)), replicate(p, runif(n)))
  fit <- ranger(y ~ ., dat, importance = "impurity_corrected")
  return(fit)
}

cleaned_ranger <- axe_fitted(wrapped_ranger(), verbose = TRUE)
```

---

axe-rda *Axing an rda.*

Description

rda objects are created from the **klaR** package, leveraged to carry out regularized discriminant analysis.

Usage

```
## S3 method for class 'rda'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'rda'
axe_env(x, verbose = FALSE, ...)
```

Arguments

x	A model object.
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Any additional arguments related to axing.

Value

Axed rda object.

Examples

```
library(klaR)

fit_mod <- function() {
  boop <- runif(1e6)
  rda(
    y ~ x,
    data = data.frame(y = rep(letters[1:4], 1e4), x = rnorm(4e4)),
    gamma = 0.05,
    lambda = 0.2
  )
}

mod_fit <- fit_mod()
mod_res <- butcher(mod_fit)

weigh(mod_fit)
weigh(mod_res)
```

---

axe-recipe *Axing a recipe object.*

Description

recipe objects are created from the **recipes** package, which is leveraged for its set of data pre-processing tools. These recipes work by sequentially defining each pre-processing step. The implementation of each step, however, results its own class so we bundle all the axe methods related to recipe objects in general here. Note that the butchered class is only added to the recipe as a whole, and not to each pre-processing step.

Usage

```
## S3 method for class 'recipe'
axe_env(x, verbose = FALSE, ...)

## S3 method for class 'step'
axe_env(x, ...)

## S3 method for class 'step_arrange'
axe_env(x, ...)

## S3 method for class 'step_filter'
axe_env(x, ...)

## S3 method for class 'step_mutate'
axe_env(x, ...)

## S3 method for class 'step_slice'
axe_env(x, ...)

## S3 method for class 'step_impute_bag'
axe_env(x, ...)

## S3 method for class 'step_bagimpute'
axe_env(x, ...)

## S3 method for class 'step_impute_knn'
axe_env(x, ...)

## S3 method for class 'step_knnimpute'
axe_env(x, ...)

## S3 method for class 'step_geodist'
axe_env(x, ...)

## S3 method for class 'step_interact'
axe_env(x, ...)

## S3 method for class 'step_ratio'
axe_env(x, ...)

## S3 method for class 'quosure'
axe_env(x, ...)

## S3 method for class 'recipe'
axe_fitted(x, verbose = FALSE, ...)
```

Arguments

x	A model object.
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Any additional arguments related to axing.

Value

Axed recipe object.

Examples

```
library(recipes)
data(biomass, package = "modeldata")

biomass_tr <- biomass[biomass$dataset == "Training",]
rec <- recipe(HHV ~ carbon + hydrogen + oxygen + nitrogen + sulfur,
              data = biomass_tr) |>
  step_center(all_predictors()) |>
  step_scale(all_predictors()) |>
  step_spatialsign(all_predictors())

out <- butcher(rec, verbose = TRUE)

# Another recipe object
wrapped_recipes <- function() {
  some_junk_in_environment <- runif(1e6)
  return(
    recipe(mpg ~ cyl, data = mtcars) |>
      step_center(all_predictors()) |>
      step_scale(all_predictors()) |>
      prep()
  )
}

# Remove junk in environment
cleaned1 <- axe_env(wrapped_recipes(), verbose = TRUE)
# Replace prepared training data with zero-row slice
cleaned2 <- axe_fitted(wrapped_recipes(), verbose = TRUE)

# Check size
lobstr::obj_size(cleaned1)
lobstr::obj_size(cleaned2)
```

---

axe-rpart *Axing a rpart.*

Description

rpart objects are created from the **rpart** package, which is used for recursive partitioning for classification, regression and survival trees.

Usage

```
## S3 method for class 'rpart'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'rpart'
axe_ctrl(x, verbose = FALSE, ...)

## S3 method for class 'rpart'
axe_data(x, verbose = FALSE, ...)

## S3 method for class 'rpart'
axe_env(x, verbose = FALSE, ...)
```

Arguments

x	A model object.
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Any additional arguments related to axing.

Value

Axed rpart object.

Examples

```
# An rpart object
wrapped_rpart <- function() {
  require("rpart")
  some_junk_in_environment <- runif(1e6)
  fit <- rpart(Kyphosis ~ Age + Number + Start,
               data = kyphosis,
               x = TRUE, y = TRUE)
  return(fit)
}

# Remove junk
cleaned_rpart <- axe_env(wrapped_rpart(), verbose = TRUE)

# Check size
lobstr::obj_size(cleaned_rpart)
```

`axe-rsample-data` *Axe data within rsample objects.*

Description

Replace the splitting and resampling objects with a placeholder.

Usage

```
axe_rsample_data(x, verbose = FALSE, ...)

## Default S3 method:
axe_rsample_data(x, verbose = FALSE, ...)

## S3 method for class 'rsplit'
axe_rsample_data(x, verbose = FALSE, ...)

## S3 method for class 'three_way_split'
axe_rsample_data(x, verbose = FALSE, ...)

## S3 method for class 'rset'
axe_rsample_data(x, verbose = FALSE, ...)

## S3 method for class 'tune_results'
axe_rsample_data(x, verbose = FALSE, ...)

## S3 method for class 'workflow_set'
axe_rsample_data(x, verbose = FALSE, ...)
```

Arguments

`x`	An object.
`verbose`	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is `FALSE`.
`...`	Any additional arguments related to axing.

Details

Resampling and splitting objects produced by **rsample** contain `rsplit` objects. These contain the original data set. These data might be large so we sometimes wish to remove them when saving objects. This method creates a zero-row slice of the dataset, retaining only the column names and their attributes, while replacing the original data.

Value

An updated object without data in the `rsplit` objects.

Methods

See the following help topics for more details about individual methods:

`butcher`

- `axe-rsample-data`: `default`, `rset`, `rsplit`, `three_way_split`, `tune_results`, `workflow_set`

Examples

```
large_cars <- mtcars[rep(1:32, 50), ]
large_cars_split <- rsample::initial_split(large_cars)
butcher(large_cars_split, verbose = TRUE)
```

`axe-rsample-indicators`

Axe indicators within rsample objects.

Description

Replace the splitting and resampling objects with a placeholder.

Usage

```
axe_rsample_indicators(x, verbose = FALSE, ...)

## Default S3 method:
axe_rsample_indicators(x, verbose = FALSE, ...)

## S3 method for class 'rsplit'
axe_rsample_indicators(x, verbose = FALSE, ...)

## S3 method for class 'three_way_split'
axe_rsample_indicators(x, verbose = FALSE, ...)

## S3 method for class 'rset'
axe_rsample_indicators(x, verbose = FALSE, ...)

## S3 method for class 'tune_results'
axe_rsample_indicators(x, verbose = FALSE, ...)

## S3 method for class 'workflow_set'
axe_rsample_indicators(x, verbose = FALSE, ...)
```

Arguments

x	An object.
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Any additional arguments related to axing.

Details

Resampling and splitting objects produced by **rsample** contain `rsplit` objects. These contain the original data set as well as indicators that specify which rows go into which data partitions. These size of these integers might be large so we sometimes wish to remove them when saving objects. This method saves a zero-row integer in their place.

Value

An updated object without the indicators in the `rsplit` objects.

Methods

See the following help topics for more details about individual methods:

`butcher`

- axe-rsample-indicators: `default`, `rset`, `rsplit`, `three_way_split`, `tune_results`, `workflow_set`

Examples

```
large_cars <- mtcars[rep(1:32, 50), ]
large_cars_split <- rsample::initial_split(large_cars)
butcher(large_cars_split, verbose = TRUE)
```

---

axe-sclass *Axing a sclass object.*

Description

sclass objects are byproducts of classbagg objects.

Usage

```
## S3 method for class 'sclass'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'sclass'
axe_env(x, verbose = FALSE, ...)
```

Arguments

x	A model object.
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Any additional arguments related to axing.

Value

Axed sclass object.

Examples

```
# Load libraries
library(ipred)
library(rpart)
library(MASS)

# Load data
data("GlaucomaM", package = "TH.data")

classbagg_fit <- bagging(Class ~ ., data = GlaucomaM, coob = TRUE)

out <- butcher(classbagg_fit$mtrees[[1]], verbose = TRUE)

# Another classbagg object
wrapped_classbagg <- function() {
  some_junk_in_environment <- runif(1e6)
  fit <- bagging(Species ~ .,
                 data = iris,
                 nbagg = 10,
                 coob = TRUE)
  return(fit)
}

# Remove junk
cleaned_classbagg <- butcher(wrapped_classbagg(), verbose = TRUE)

# Check size
lobstr::obj_size(cleaned_classbagg)
```

---

axe-spark *Axing a spark object.*

Description

spark objects are created from the **sparklyr** package, a R interface for Apache Spark. The axe methods available for spark objects are designed such that interoperability is maintained. In other words, for a multilingual machine learning team, butchered spark objects instantiated from **sparklyr** can still be serialized to disk, work in Python, be deployed on Scala, etc. It is also worth noting here that spark objects created from **sparklyr** have a lot of metadata attached to it, including but not limited to the formula, dataset, model, index labels, etc. The axe functions provided are for parsing down the model object both prior saving to disk, or loading from disk. Traditional R save functions are not available for these objects, so functionality is provided in `sparklyr::ml_save`. This function gives the user the option to keep either the `pipeline_model` or the `pipeline`, so both of these objects are retained from butchering, yet removal of one or the other might be conducive to freeing up memory on disk.

Usage

```
## S3 method for class 'ml_model'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'ml_model'
axe_ctrl(x, verbose = FALSE, ...)

## S3 method for class 'ml_model'
axe_data(x, verbose = FALSE, ...)

## S3 method for class 'ml_model'
axe_fitted(x, verbose = FALSE, ...)
```

Arguments

`x`	A model object.
`verbose`	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is `FALSE`.
`...`	Any additional arguments related to axing.

Value

Axed spark object.

Examples

```
library(sparklyr)

sc <- spark_connect(master = "local")

iris_tbls <- sdf_copy_to(sc, iris, overwrite = TRUE) |>
  sdf_random_split(train = 2/3, validation = 2/3, seed = 2018)

train <- iris_tbls$train
spark_fit <- ml_logistic_regression(train, Species ~ .)
```

```
out <- butcher(spark_fit, verbose = TRUE)

spark_disconnect(sc)
```

`axe-survreg` *Axing an survreg.*

Description

survreg objects are created from the **survival** package. They are returned from the `survreg` function, representing fitted parametric survival models.

Usage

```
## S3 method for class 'survreg'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'survreg'
axe_data(x, verbose = FALSE, ...)

## S3 method for class 'survreg'
axe_env(x, verbose = FALSE, ...)
```

Arguments

`x`	A model object.
`verbose`	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is `FALSE`.
`...`	Any additional arguments related to axing.

Value

Axed survreg object.

Examples

```
# Load libraries
library(parsnip)
library(survival)

# Create model and fit
survreg_fit <- survival_reg(dist = "weibull") |>
  set_engine("survival") |>
  fit(Surv(futime, fustat) ~ 1, data = ovarian)

out <- butcher(survreg_fit, verbose = TRUE)
```

```
# Another survreg object
wrapped_survreg <- function() {
  some_junk_in_environment <- runif(1e6)
  fit <- survreg(Surv(time, status) ~ ph.ecog + age + strata(sex),
                 data = lung)
  return(fit)
}

# Remove junk
cleaned_survreg <- butcher(wrapped_survreg(), verbose = TRUE)

# Check size
lobstr::obj_size(cleaned_survreg)
```

`axe-survreg.penal` *Axing an survreg.penal*

Description

survreg.penal objects are created from the **survival** package. They are returned from the `survreg` function, representing fitted parametric survival models.

Usage

```
## S3 method for class 'survreg.penal'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'survreg.penal'
axe_data(x, verbose = FALSE, ...)

## S3 method for class 'survreg.penal'
axe_env(x, verbose = FALSE, ...)
```

Arguments

`x`	A model object.
`verbose`	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is `FALSE`.
`...`	Any additional arguments related to axing.

Value

Axed survreg object.

Examples

```
# Load libraries
library(parsnip)
library(survival)

# Create model and fit
survreg_fit <- survival_reg(dist = "weibull") |>
  set_engine("survival") |>
  fit(Surv(time, status) ~ rx, data = rats)

out <- butcher(survreg_fit, verbose = TRUE)

# Another survreg.penal object
wrapped_survreg.penal <- function() {
  some_junk_in_environment <- runif(1e6)
  fit <- survreg(Surv(time, status) ~ rx,
                 data = rats, subset = (sex == "f"))
  return(fit)
}

# Remove junk
cleaned_sp <- axe_env(wrapped_survreg.penal(), verbose = TRUE)

# Check size
lobstr::obj_size(cleaned_sp)
```

`axe-tabnet_fit` *Axing a tabnet_fit.*

Description

Axing a tabnet_fit.

Remove fitted values.

Usage

```
## S3 method for class '`_tabnet_fit`'
axe_fitted(x, verbose = FALSE, ...)
```

Arguments

`x`	A model object.
`verbose`	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is `FALSE`.
`...`	Any additional arguments related to axing.

Value

Axed tabnet_fit object.

Examples

```
# Load libraries
suppressWarnings(suppressMessages(library(parsnip)))
suppressWarnings(suppressMessages(library(rsample)))
suppressWarnings(suppressMessages(library(tabnet)))

# Load data
split <- initial_split(mtcars, prop = 9/10)
car_train <- training(split)


if (interactive() & torch::torch_is_installed()) {
  torch::torch_manual_seed(1)

  # Create model and fit
  mtcar_fit <- tabnet::tabnet() |>
  set_mode("regression") |>
  set_engine("torch") |>
  fit(mpg ~ ., data = car_train)

  out <- butcher(mtcar_fit, verbose = TRUE)
}
```

---

axe-terms *Axing for terms inputs.*

Description

Generics related to axing objects of the term class.

Usage

```
## S3 method for class 'terms'
axe_env(x, verbose = FALSE, ...)
```

Arguments

x	A model object.
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Any additional arguments related to axing.

Value

Axed terms object.

Examples

```
# Using lm
wrapped_lm <- function() {
  some_junk_in_environment <- runif(1e6)
  fit <- lm(mpg ~ ., data = mtcars)
  return(fit)
}

# Remove junk
cleaned_lm <- axe_env(wrapped_lm(), verbose = TRUE)

# Check size
lobstr::obj_size(cleaned_lm)

# Compare environment in terms component
lobstr::obj_size(attr(wrapped_lm()$terms, ".Environment"))
lobstr::obj_size(attr(cleaned_lm$terms, ".Environment"))

# Using rpart
library(rpart)

wrapped_rpart <- function() {
  some_junk_in_environment <- runif(1e6)
  fit <- rpart(Kyphosis ~ Age + Number + Start,
               data = kyphosis,
               x = TRUE,
               y = TRUE)
  return(fit)
}

lobstr::obj_size(wrapped_rpart())
lobstr::obj_size(axe_env(wrapped_rpart()))
```

---

axe-train *Axing a train object.*

Description

train objects are created from the **caret** package.

Usage

```
## S3 method for class 'train'
axe_call(x, verbose = FALSE, ...)
```

```
## S3 method for class 'train'
axe_ctrl(x, verbose = FALSE, ...)

## S3 method for class 'train'
axe_data(x, verbose = FALSE, ...)

## S3 method for class 'train'
axe_env(x, verbose = FALSE, ...)

## S3 method for class 'train'
axe_fitted(x, verbose = FALSE, ...)
```

Arguments

x	A model object.
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Any additional arguments related to axing.

Value

Axed train object.

Examples

```
# Load libraries
library(caret)

data(iris)
train_data <- iris[, 1:4]
train_classes <- iris[, 5]

train_fit <- train(train_data, train_classes,
                   method = "knn",
                   preProcess = c("center", "scale"),
                   tuneLength = 10,
                   trControl = trainControl(method = "cv"))

out <- butcher(train_fit, verbose = TRUE)
```

---

axe-train.recipe *Axing a train.recipe object.*

Description

train.recipe objects are slightly different from train objects created from the `caret` package in that it also includes instructions from a `recipe` for data pre-processing. Axing functions specific to train.recipe are thus included as additional steps are required to remove parts of train.recipe objects.

Usage

```
## S3 method for class 'train.recipe'
axe_call(x, ...)

## S3 method for class 'train.recipe'
axe_ctrl(x, ...)

## S3 method for class 'train.recipe'
axe_data(x, ...)

## S3 method for class 'train.recipe'
axe_env(x, ...)

## S3 method for class 'train.recipe'
axe_fitted(x, ...)
```

Arguments

x	A model object.
...	Any additional arguments related to axing.

Value

Axed train.recipe object.

Examples

```
library(recipes)
library(caret)
data(biomass, package = "modeldata")

data(biomass)
recipe <- biomass |>
  recipe(HHV ~ carbon + hydrogen + oxygen + nitrogen + sulfur) |>
  step_center(all_predictors()) |>
  step_scale(all_predictors()) |>
  step_spatialsign(all_predictors())

train.recipe_fit <- train(recipe, biomass,
                          method = "svmRadial",
                          metric = "RMSE")

out <- butcher(train.recipe_fit, verbose = TRUE)
```

axe-xgb.Booster *Axing a xgb.Booster.*

Description

xgb.Booster objects are created from the **xgboost** package, which provides efficient and scalable implementations of gradient boosted decision trees. Given the reliance of post processing functions on the model object, like `xgb.Booster.complete`, on the first class listed, the `butcher_xgb.Booster` class is not appended.

Usage

```
## S3 method for class 'xgb.Booster'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'xgb.Booster'
axe_env(x, verbose = FALSE, ...)
```

Arguments

`x`	A model object.
`verbose`	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is `FALSE`.
`...`	Any additional arguments related to axing.

Value

Axed xgb.Booster object.

Examples

```
library(xgboost)
library(parsnip)

data(agaricus.train)

if (utils::packageVersion("xgboost") > "2.0.0.0") {
  bst <- xgboost(x = agaricus.train$data,
                 y = as.factor(agaricus.train$label),
                 learning_rate = 1,
                 nthread = 2,
                 nrounds = 2,
                 eval_metric = "logloss",
                 objective = "binary:logistic")
} else {
  bst <- xgboost(data = agaricus.train$data,
                 label = agaricus.train$label,
                 eta = 1,
```

```
                  nthread = 2,
                  nrounds = 2,
                  eval_metric = "logloss",
                  objective = "binary:logistic",
                  verbose = 0)
}

out <- butcher(bst, verbose = TRUE)

# Another xgboost model
fit <- boost_tree(mode = "classification", trees = 20) |>
  set_engine("xgboost", eval_metric = "mlogloss") |>
  fit(Species ~ ., data = iris)

out <- butcher(fit, verbose = TRUE)
```

`axe-xrf` *Axing a xrf.*

Description

Axing a xrf.

Usage

```
## S3 method for class 'xrf'
axe_call(x, verbose = FALSE, ...)

## S3 method for class 'xrf'
axe_env(x, verbose = FALSE, ...)
```

Arguments

`x`	A model object.
`verbose`	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is `FALSE`.
`...`	Any additional arguments related to axing.

Value

Axed xrf object.

Examples

```
library(xrf)

xrf_big <- function() {
  boop <- runif(1e6)
  xrf(
    mpg ~ .,
    mtcars,
    xgb_control = list(nrounds = 2, max_depth = 2),
    family = 'gaussian'
  )
}

heavy_m <- xrf_big()

m <- butcher(heavy_m, verbose = TRUE)

weigh(heavy_m)
weigh(m)
```

---

axe_call *Axe a call.*

Description

Replace the call object attached to modeling objects with a placeholder.

Usage

```
axe_call(x, verbose = FALSE, ...)
```

Arguments

x	A model object.
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Any additional arguments related to axing.

Value

Model object without call attribute.

Methods

See the following help topics for more details about individual methods:

butcher

- axe-C5.0: C5.0
- axe-KMeansCluster: KMeansCluster
- axe-NaiveBayes: NaiveBayes
- axe-bart: bart
- axe-earth: earth
- axe-elnet: elnet
- axe-flexsurvreg: flexsurvreg
- axe-gam: gam
- axe-gausspr: gausspr
- axe-glm: glm
- axe-glmnet: glmnet
- axe-ipred: classbagg, regbagg, survbagg
- axe-ksvm: ksvm
- axe-lm: lm
- axe-mda: fda, mda
- axe-model_fit: model_fit
- axe-multnet: multnet
- axe-nnet: nnet
- axe-pls: mixo_pls, mixo_spls
- axe-randomForest: randomForest
- axe-ranger: ranger
- axe-rda: rda
- axe-rpart: rpart
- axe-sclass: sclass
- axe-spark: ml_model
- axe-survreg: survreg
- axe-survreg.penal: survreg.penal
- axe-train: train
- axe-train.recipe: train.recipe
- axe-xgb.Booster: xgb.Booster
- axe-xrf: xrf

`axe_ctrl` *Axe controls.*

Description

Remove the controls from training attached to modeling objects.

Usage

```
axe_ctrl(x, verbose = FALSE, ...)
```

Arguments

`x`	A model object.
`verbose`	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is `FALSE`.
`...`	Any additional arguments related to axing.

Value

Model object without control tuning parameters from training.

Methods

See the following help topics for more details about individual methods:

`butcher`

- `axe-C5.0`: `C5.0`
- `axe-gam`: `gam`
- `axe-ipred`: `classbagg`, `regbagg`, `survbagg`
- `axe-model_fit`: `model_fit`
- `axe-randomForest`: `randomForest`
- `axe-rpart`: `rpart`
- `axe-spark`: `ml_model`
- `axe-train`: `train`
- `axe-train.recipe`: `train.recipe`

axe_data *Axe data.*

Description

Remove the training data attached to modeling objects.

Usage

```
axe_data(x, verbose = FALSE, ...)
```

Arguments

x	A model object.
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Any additional arguments related to axing.

Value

Model object without the training data

Methods

See the following help topics for more details about individual methods:

butcher

- axe-NaiveBayes: NaiveBayes
- axe-coxph: coxph
- axe-earth: earth
- axe-gam: gam
- axe-gausspr: gausspr
- axe-glm: glm
- axe-ipred: classbagg, regbagg, survbagg
- axe-kproto: kproto
- axe-ksvm: ksvm
- axe-model_fit: model_fit
- axe-pls: mixo_pls, mixo_spls
- axe-rpart: rpart
- axe-spark: ml_model
- axe-survreg: survreg
- axe-survreg.penal: survreg.penal
- axe-train: train
- axe-train.recipe: train.recipe

axe_env *Axe an environment.*

Description

Remove the environment(s) attached to modeling objects as they are not required in the downstream analysis pipeline. If found, the environment is replaced with `rlang::base_env()`.

Usage

```
axe_env(x, verbose = FALSE, ...)
```

Arguments

x	A model object.
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Any additional arguments related to axing.

Value

Model object with empty environments.

Methods

See the following help topics for more details about individual methods:

butcher

- axe-coxph: coxph
- axe-flexsurvreg: flexsurvreg
- axe-formula: formula
- axe-function: function
- axe-gam: gam
- axe-gausspr: gausspr
- axe-glm: glm
- axe-ipred: classbagg, regbagg, survbagg
- axe-kknn: kknn
- axe-lm: lm
- axe-mass: lda, polr, qda
- axe-mda: fda, mda
- axe-model_fit: model_fit
- axe-nnet: nnet
- axe-randomForest: randomForest

- axe-rda: rda
- axe-recipe: quosure, recipe, step, step_arrange, step_bagimpute, step_filter, step_geodist, step_impute_bag, step_impute_knn, step_interact, step_knnimpute, step_mutate, step_ratio, step_slice
- axe-rpart: rpart
- axe-sclass: sclass
- axe-survreg: survreg
- axe-survreg.penal: survreg.penal
- axe-terms: terms
- axe-train: train
- axe-train.recipe: train.recipe
- axe-xgb.Booster: xgb.Booster
- axe-xrf: xrf

`axe_fitted` *Axe fitted values.*

Description

Remove the fitted values attached to modeling objects.

Usage

```
axe_fitted(x, verbose = FALSE, ...)
```

Arguments

x	A model object.
verbose	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is FALSE.
...	Any additional arguments related to axing.

Value

Model object without the fitted values.

Methods

See the following help topics for more details about individual methods:

butcher

- axe-C5.0: C5.0
- axe-KMeansCluster: KMeansCluster

- axe-bart: bart
- axe-earth: earth
- axe-gam: gam
- axe-gausspr: gausspr
- axe-glm: glm
- axe-kknn: kknn
- axe-kproto: kproto
- axe-ksvm: ksvm
- axe-lm: lm
- axe-mda: fda, mda
- axe-model_fit: model_fit
- axe-nnet: nnet
- axe-pls: mixo_pls, mixo_spls
- axe-ranger: ranger
- axe-recipe: recipe
- axe-spark: ml_model
- axe-tabnet_fit: _tabnet_fit
- axe-train: train
- axe-train.recipe: train.recipe

`butcher` *Butcher an object.*

Description

Reduce the size of a model object so that it takes up less memory on disk. Currently, the model object is stripped down to the point that only the minimal components necessary for the `predict` function to work remain. Future adjustments to this function will be needed to avoid removal of model fit components to ensure it works with other downstream functions.

Usage

```
butcher(x, verbose = FALSE, ...)
```

Arguments

`x`	A model object.
`verbose`	Print information each time an axe method is executed. Notes how much memory is released and what functions are disabled. Default is `FALSE`.
`...`	Any additional arguments related to axing.

Value

Axed model object with new butcher subclass assignment.

locate *Locate part of an object.*

Description

Locate where a specific component of a object might exist within the model object itself. This function is restricted in that only items that can be axed can be found.

Usage

```
locate(x, name = NULL)
```

Arguments

x	A model object.
name	A name associated with model component of interest. This defaults to NULL. Possible components include: `env`, `call`, `data`, `ctrl`, and `fitted`.

Value

Location of specific component in a model object.

Examples

```
lm_fit <- lm(mpg ~ ., data = mtcars)
locate(lm_fit, name = "env")
locate(lm_fit, name = "call")
```

new_model_butcher *New axe functions for a modeling object.*

Description

`new_model_butcher()` will instantiate the following to help us develop new axe functions around removing parts of a new modeling object:

- Add modeling package to `Suggests`
- Generate and populate an axe file under `R/`
- Generate and populate an test file under `testthat/`

Usage

```
new_model_butcher(
  model_class,
  package_name,
  open = interactive(),
  call = rlang::caller_env()
)
```

Arguments

model_class	A string that captures the class name of the new model object.
package_name	A string that captures the package name from which the new model is made.
open	Check if user is in interactive mode, and if so, opens the new files for editing.
call	The execution environment of a currently running function, e.g. caller_env(). The function will be mentioned in error messages as the source of the error.

---

weigh *Weigh the object.*

Description

Evaluate the size of each element contained in a model object.

Usage

```
weigh(x, threshold = 0, units = "MB", ...)
```

Arguments

x	A model object.
threshold	The minimum threshold desired for model component size to display.
units	The units in which to display the size of each component within the model object of interest. Defaults to MB. Other options include KB and GB.
...	Any additional arguments for weighing.

Value

Tibble with weights of object components in decreasing magnitude.

Examples

```
simulate_x <- matrix(runif(1e+6), ncol = 2)
simulate_y <- runif(dim(simulate_x)[1])
lm_out <- lm(simulate_y ~ simulate_x)
weigh(lm_out)
```

Index