Package ‘baguette’

April 14, 2020

Title Efficient Model Functions for Bagging

Version 0.0.1

Description Tree- and rule-based models can be bagged using
this package and their predictions equations are stored
in an efficient format to reduce the model objects size
and speed.

License MIT + file LICENSE

Depends parsnip (>= 0.1.0)

Suggests testthat, AmesHousing, recipes, modeldata, covr, yardstick

Encoding UTF-8

LazyData true

Imports hardhat, butcher, rpart, C50, withr, rsample, dplyr, purrr,
furrr, tibble, tidyr, rlang, earth, magrittr, utils, generics,
dials

URL `https://github.com/tidymodels/baguette`

BugReports `https://github.com/tidymodels/baguette/issues`

RoxygenNote 7.1.0.9000

NeedsCompilation no

Author Max Kuhn [aut, cre] (<https://orcid.org/0000-0003-2402-136X>),
RStudio [cph]

Maintainer Max Kuhn <`max@rstudio.com`>

Repository CRAN

Date/Publication 2020-04-14 14:20:04 UTC

R topics documented:

bagger *Bagging functions*

Description

General suite of bagging functions for several models.

Usage

```
bagger(x, ...)

## Default S3 method:
bagger(x, ...)

## S3 method for class 'data.frame'
bagger(
  x,
  y,
  base_model = "CART",
  times = 11L,
  control = control_bag(),
  cost = NULL,
  ...
)

## S3 method for class 'matrix'
bagger(
  x,
  y,
  base_model = "CART",
  times = 11L,
  control = control_bag(),
  cost = NULL,
  ...
)

## S3 method for class 'formula'
bagger(
  formula,
  data,
  base_model = "CART",
  times = 11L,
  control = control_bag(),
  cost = NULL,
  ...
)

## S3 method for class 'recipe'
bagger(
  x,
  data,
  base_model = "CART",
  times = 11L,
  control = control_bag(),
  cost = NULL,
  ...
)
```

Arguments

`x` A data frame, matrix, or recipe (depending on the method being used).

`...` Optional arguments to pass to the base model function.

`y` A numeric or factor vector of outcomes. Categorical outcomes (i.e classes) should be represented as factors, not integers.

`base_model` A single character value for the model being bagged. Possible values are "CART", "MARS", and "C5.0" (classification only).

`times` A single integer greater than 1 for the maximum number of bootstrap samples/ensemble members (some model fits might fail).

`control` A list of options generated by `control_bag()`.

`cost` A non-negative scale (for two class problems) or a cost matrix.

`formula` An object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to be fitted. Note that this package does not support multivariate outcomes and that, if some predictors are factors, dummy variables will *not* be created unless by the underyling model function.

`data` A data frame containing the variables used in the formula or recipe.

Details

`bagger()` fits separate models to bootstrap samples. The prediction function for each model object is encoded in an R expression and the original model object is discarded. When making predictions, each prediction formula is evaluated on the new data and aggregated using the mean.

Variable importance scores are calculated using implementations in each package. When requested, the results are in a tibble with column names `term` (the predictor), `value` (the importance score), and `used` (the percentage of times that the variable was in the prediction equation).

The models can be fit in parallel using the **future** package. The enable parallelism, use the `future::plan()` function to declare *how* the computations should be distributed. Note that this will almost certainly multiply the memory requirements required to fit the models.

Examples

```
library(recipes)
library(dplyr)

data(biomass, package = "modeldata")

biomass_tr <-
  biomass %>%
  dplyr::filter(dataset == "Training") %>%
  dplyr::select(-dataset, -sample)

biomass_te <-
  biomass %>%
  dplyr::filter(dataset == "Testing") %>%
  dplyr::select(-dataset, -sample)

# ------------------------------------------------------------------------------

ctrl <- control_bag(var_imp = TRUE)

# ------------------------------------------------------------------------------

# `times` is low to make the examples run faster

set.seed(7687)
mars_bag <- bagger(x = biomass_tr[, -6], y = biomass_tr$HHV,
                   base_model = "MARS", times = 5, control = ctrl)
mars_bag
var_imp(mars_bag)

set.seed(7687)
cart_bag <- bagger(x = biomass_tr[, -6], y = biomass_tr$HHV,
                   base_model = "CART", times = 5, control = ctrl)
cart_bag

# ------------------------------------------------------------------------------
# Other interfaces

# Recipes can be used
biomass_rec <-
  recipe(HHV ~ ., data = biomass_tr) %>%
  step_pca(all_predictors())

set.seed(7687)
cart_pca_bag <- bagger(biomass_rec, data = biomass_tr, base_model = "CART",
                       times = 5, control = ctrl)

cart_pca_bag

# Using formulas
mars_bag <- bagger(HHV ~ ., data = biomass_tr, base_model = "MARS", times = 5,
                   control = ctrl)
```

mars_bag

bag_mars *General Interface for Bagged MARS Models*

Description

bag_mars() is a way to generate a *specification* of a model before fitting and allows the model to be created using different packages in R. The main arguments for the model are:

- num_terms: The number of features that will be retained in the final model.
- prod_degree: The highest possible degree of interaction between features. A value of 1 indicates and additive model while a value of 2 allows, but does not guarantee, two-way interactions between features.
- prune_method: The type of pruning. Possible values are listed in ?earth.

These arguments are converted to their specific names at the time that the model is fit. Other options and argument can be set using set_engine(). If left to their defaults here (NULL), the values are taken from the underlying model functions. If parameters need to be modified, update() can be used in lieu of recreating the object from scratch.

Usage

```
bag_mars(
  mode = "unknown",
  num_terms = NULL,
  prod_degree = NULL,
  prune_method = NULL
)

## S3 method for class 'bag_mars'
update(
  object,
  parameters = NULL,
  num_terms = NULL,
  prod_degree = NULL,
  prune_method = NULL,
  fresh = FALSE,
  ...
)
```

Arguments

mode A single character string for the type of model. Possible values for this model are "unknown", "regression", or "classification".

num_terms The number of features that will be retained in the final model, including the intercept.

`prod_degree` The highest possible interaction degree.

`prune_method` The pruning method.

`object` A bagged MARS model specification.

`parameters` A 1-row tibble or named list with *main* parameters to update. If the individual arguments are used, these will supersede the values in `parameters`. Also, using engine arguments in this object will result in an error.

`fresh` A logical for whether the arguments should be modified in-place of or replaced wholesale.

`...` Not used for `update()`.

Details

The model can be created using the `fit()` function using the following *engines*:

- **R**: `"earth"` (the default)

Examples

```
library(parsnip)

set.seed(7396)
bag_mars(num_terms = 7) %>%
  set_mode("regression") %>%
  set_engine("earth", times = 3) %>%
  fit(mpg ~ ., data = mtcars)


model <- bag_mars(num_terms = 10, prune_method = "none")
model
update(model, num_terms = 2)
update(model, num_terms = 2, fresh = TRUE)
```

`bag_tree` *General Interface for Bagged Decision Tree Models*

Description

`bag_tree()` is a way to generate a *specification* of a model before fitting and allows the model to be created using different packages in R. The main arguments for the model are:

- `cost_complexity`: The cost/complexity parameter (a.k.a. `Cp`) used by CART models (`rpart` only).
- `tree_depth`: The *maximum* depth of a tree (`rpart`).
- `min_n`: The minimum number of data points in a node that are required for the node to be split further.
- `class_cost`: A cost value to asign to the class corresponding to the first factor level (for 2-class models, `rpart` and `C5.0` only).

These arguments are converted to their specific names at the time that the model is fit. Other options and argument can be set using `set_engine()`. If left to their defaults here (`NULL`), the values are taken from the underlying model functions. If parameters need to be modified, `update()` can be used in lieu of recreating the object from scratch.

Usage

```
bag_tree(
  mode = "unknown",
  cost_complexity = 0,
  tree_depth = NULL,
  min_n = 2,
  class_cost = NULL
)

## S3 method for class 'bag_tree'
update(
  object,
  parameters = NULL,
  cost_complexity = NULL,
  tree_depth = NULL,
  min_n = NULL,
  class_cost = NULL,
  fresh = FALSE,
  ...
)
```

Arguments

`mode` A single character string for the type of model. Possible values for this model are "unknown", "regression", or "classification".

`cost_complexity` A positive number for the the cost/complexity parameter (a.k.a. `Cp`) used by CART models (`rpart` only).

`tree_depth` An integer for maximum depth of the tree.

`min_n` An integer for the minimum number of data points in a node that are required for the node to be split further.

`class_cost` A non-negative scalar for a class cost (where a cost of 1 means no extra cost). This is useful for when the first level of the outcome factor is the minority class. If this is not the case, values between zero and one can be used to bias to the second level of the factor.

`object` A bagged tree model specification.

`parameters` A 1-row tibble or named list with *main* parameters to update. If the individual arguments are used, these will supersede the values in `parameters`. Also, using engine arguments in this object will result in an error.

`fresh` A logical for whether the arguments should be modified in-place of or replaced wholesale.

`...` Not used for `update()`.

Details

The model can be created using the `fit()` function using the following *engines*:

- **R**: `"rpart"` (the default) or `"C5.0"` (classification only)

Note that, for `rpart` models, but `cost_complexity` and `tree_depth` can be both be specified but the package will give precedence to `cost_complexity`. Also, `tree_depth` values greater than 30 `rpart` will give nonsense results on 32-bit machines.

Examples

```
library(parsnip)

set.seed(9952)
bag_tree(tree_depth = 5) %>%
  set_mode("classification") %>%
  set_engine("rpart", times = 3) %>%
  fit(Species ~ ., data = iris)


model <- bag_tree(cost_complexity = 10, min_n = 3)
model
update(model, cost_complexity = 1)
update(model, cost_complexity = 1, fresh = TRUE)
```

`class_cost` *Cost parameter for minority class*

Description

Used in `bag_treer()`.

Usage

```
class_cost(range = c(0, 5), trans = NULL)
```

Arguments

`range` A two-element vector holding the *defaults* for the smallest and largest possible values, respectively.

`trans` A `trans` object from the `scales` package, such as `scales::log10_trans()` or `scales::reciprocal_trans()`. If not provided, the default is used which matches the units used in `range`. If no transformation, `NULL`.

Details

This parameter reflects the cost of a mis-classified sample relative to a baseline cost of 1.0. For example, if the first level of an outcome factor occurred rarely, it might help if this parameter were set to values greater than 1.0. If the second level of the outcome factor is in the minority, values less than 1.0 would cause the model to emphasize the minority class more than the majority class.

Examples

```
class_cost()
```

`control_bag` *Controlling the bagging process*

Description

`control_bag()` can set options for ancillary aspects of the bagging process.

Usage

```
control_bag(
  var_imp = TRUE,
  allow_parallel = TRUE,
  sampling = "none",
  reduce = TRUE,
  extract = NULL
)
```

Arguments

`var_imp`	A single logical: should variable importance scores be calculated?
`allow_parallel`	A single logical: should the model fits be done in parallel (even if a parallel `plan()` has been created)?
`sampling`	Either "none" or "down". For classification only. The training data, after bootstrapping, will be sampled down within each class (with replacement) to the size of the smallest class.
`reduce`	Should models be modified to reduce their size on disk?
`extract`	A function (or NULL) that can extract model-related aspects of each ensemble member. See Details and example below.

Details

Any arbitrary item can be saved from the model object (including the model object itself) using the `extract` argument, which should be a function with arguments `x` (for the model object), and `...`. The results of this function are saved into a list column called `extras` (see the example below).

Value

A list.

Examples

```
# Extracting model components

num_term_nodes <- function(x, ...) {
  tibble::tibble(num_nodes = sum(x$frame$var == "<leaf>"))
}

set.seed(7687)
with_extras <- bagger(mpg ~ ., data = mtcars,
                      base_model = "CART", times = 5,
                      control = control_bag(extract = num_term_nodes))

dplyr::bind_rows(with_extras$model_df$extras)
```

---

predict.bagger *Predictions from a bagged model*

Description

The `predict()` function computes predictions from each of the models in the ensembles and returns a single aggregated value for each sample in `new_data`.

Usage

```
## S3 method for class 'bagger'
predict(object, new_data, type = NULL, ...)
```

Arguments

`object`	An object generated by `bagger()`.
`new_data`	A data frame of predictors. If a recipe or formula were originally used, the **original** data should be passed here instead of a preprocessed version.
`type`	A single character value for the type of predictions. For regression models, `type = 'numeric'` is valid and `'class'` and `'prob'` are valid for classification models.
`...`	Not currently used.

Examples

```
data(airquality)

set.seed(7687)
mars_bag <- bagger(Ozone ~ ., data = airquality, base_model = "MARS", times = 5)
predict(mars_bag, new_data = airquality[, -1])
```

var_imp.bagger — *Obtain variable importance scores*

Description

Obtain variable importance scores

Usage

```
## S3 method for class 'bagger'
var_imp(object, ...)
```

Arguments

`object`	An object.
`...`	Not currently used.

Details

`baguette` can compute different variable importance scores for each model in the ensemble. The `var_imp()` function returns the average importance score for each model. Additionally, the function returns the number of times that each predictor is included in the final prediction equation.

Specific methods used by the models are:

CART: The model accumulates the improvement of the model that occurs when a predictor is used in a split. These values are taken form the `rpart` object. See `rpart::rpart.object()`.

MARS: MARS models include a backwards elimination feature selection routine that looks at reductions in the generalized cross-validation (GCV) estimate of error. The `earth()` function tracks the changes in model statistics, such as the GCV, for each predictor and accumulates the reduction in the statistic when each predictor's feature is added to the model. This total reduction is used as the variable importance measure. If a predictor was never used in any of the MARS basis functions in the final model (after pruning), it has an importance value of zero. `baguette` wraps `earth::evimp()`.

C5.0: `C5.0` measures predictor importance by determining the percentage of training set samples that fall into all the terminal nodes after the split. For example, the predictor in the first split automatically has an importance measurement of 100 percent since all samples are affected by this split. Other predictors may be used frequently in splits, but if the terminal nodes cover only a handful of training set samples, the importance scores may be close to zero.

Note that the `value` column that is the average of the importance scores form each model. The divisor of this average (and the corresponding standard error) is the number of models (as opposed to the number of models that used the predictor). This means that the importance scores for a predictor that was not used in the model has an implicit zero importance.

Value

A tibble with columns for `term` (the predictor), `value` (the mean importance score), `std.error` (the standard error), and `used` (the occurrences of the predictors).

Index