Package ‘brulee’

October 12, 2022

Title  High-Level Modeling Functions with 'torch'

Version  0.2.0

Description  Provides high-level modeling functions to define and train models using the 'torch' R package. Models include linear, logistic, and multinomial regression as well as multilayer perceptrons.

License  MIT + file LICENSE

URL  https://github.com/tidymodels/brulee,
     https://tidymodels.github.io/brulee/

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

Imports  cli, coro (>= 1.0.1), dplyr, generics, ggplot2, glue, hardhat, rlang, stats, tibble, torch (>= 0.6.0), utils

Suggests  covr, modeldata, purrr, recipes, spelling, testthat, yardstick

Config/Needs/website  tidyverse/tidytemplate

Config/testthat/edition  3

Encoding  UTF-8

Language  en-US

RoxygenNote  7.2.1.9000

NeedsCompilation  no

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

Maintainer  Max Kuhn <max@rstudio.com>

Repository  CRAN

Date/Publication  2022-09-19 23:36:10 UTC
R topics documented:

- brulee-autoplot .................................................. 2
- brulee-coefs .......................................................... 3
- brulee_linear_reg .................................................. 4
- brulee_logistic_reg .................................................. 9
- brulee_mlp ............................................................ 13
- brulee_multinomial_reg ............................................. 20
- matrix_to_dataset .................................................. 24
- predict.brulee_linear_reg .......................................... 25
- predict.brulee_logistic_reg ......................................... 26
- predict.brulee_mlp .................................................. 27
- predict.brulee_multinomial_reg .................................... 29
- schedule_decay_time ............................................... 30

Index 32

---

### Description

Plot model loss over epochs

### Usage

```r
## S3 method for class 'brulee_mlp'
autoplot(object, ...)

## S3 method for class 'brulee_logistic_reg'
autoplot(object, ...)

## S3 method for class 'brulee_multinomial_reg'
autoplot(object, ...)

## S3 method for class 'brulee_linear_reg'
autoplot(object, ...)
```

### Arguments

- **object**
  - A `brulee_mlp`, `brulee_logistic_reg`, `brulee_multinomial_reg`, or `brulee_linear_reg` object.

- **...**
  - Not currently used

### Details

This function plots the loss function across the available epochs. A vertical line shows the epoch with the best loss value.
Value

A ggplot object.

Examples

```r
if (torch::torch_is_installed()) {
  library(ggplot2)
  library(recipes)
  theme_set(theme_bw())

  data(ames, package = "modeldata")
  ames$Sale_Price <- log10(ames$Sale_Price)
  set.seed(1)
  in_train <- sample(1:nrow(ames), 2000)
  ames_train <- ames[ in_train,]
  ames_test  <- ames[-in_train,]
  ames_rec  <-
      recipe(Sale_Price ~ Longitude + Latitude, data = ames_train) %>%
      step_normalize(all_numeric_predictors())
  set.seed(2)
  fit <- brulee_mlp(ames_rec, data = ames_train, epochs = 50, batch_size = 32)
  autoplot(fit)
}
```

**brulee-coefs**

*Extract Model Coefficients*

**Description**

Extract Model Coefficients

**Usage**

```r
## S3 method for class 'brulee_logistic_reg'
coef(object, epoch = NULL, ...)

## S3 method for class 'brulee_linear_reg'
coef(object, epoch = NULL, ...)

## S3 method for class 'brulee_mlp'
coef(object, epoch = NULL, ...)
```
## S3 method for class 'brulee_multinomial_reg'

```
coef(object, epoch = NULL, ...)
```

### Arguments

- **object**: A model fit from `brulee`.
- **epoch**: A single integer for the training iteration. If left NULL, the estimates from the best model fit (via internal performance metrics).
- **...**: Not currently used.

### Value

For logistic/linear regression, a named vector. For neural networks, a list of arrays.

### Examples

```r
if (torch::torch_is_installed()) {
  data(ames, package = "modeldata")
  ames$Sale_Price <- log10(ames$Sale_Price)
  set.seed(1)
  in_train <- sample(1:nrow(ames), 2000)
  ames_train <- ames[in_train,]
  ames_test <- ames[-in_train,]
  # Using recipe
  library(recipes)
  ames_rec <-
    recipe(Sale_Price ~ Longitude + Latitude, data = ames_train) %>%
    step_normalize(all_numeric_predictors())
  set.seed(2)
  fit <- brulee_linear_reg(ames_rec, data = ames_train,
                          epochs = 50, batch_size = 32)
  coef(fit)
  coef(fit, epoch = 1)
}
```

---

### Description

`brulee_linear_reg()` fits a linear regression model.
Usage

brulee_linear_reg(x, ...)

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

## S3 method for class 'data.frame'
brulee_linear_reg(
  x,
  y,
  epochs = 20L,
  penalty = 0.001,
  mixture = 0,
  validation = 0.1,
  optimizer = "LBFGS",
  learn_rate = 1,
  momentum = 0,
  batch_size = NULL,
  stop_iter = 5,
  verbose = FALSE,
  ...
)

## S3 method for class 'matrix'
brulee_linear_reg(
  x,
  y,
  epochs = 20L,
  penalty = 0.001,
  mixture = 0,
  validation = 0.1,
  optimizer = "LBFGS",
  learn_rate = 1,
  momentum = 0,
  batch_size = NULL,
  stop_iter = 5,
  verbose = FALSE,
  ...
)

## S3 method for class 'formula'
brulee_linear_reg(
  formula,
  data,
  epochs = 20L,
  penalty = 0.001,
  mixture = 0,
  validation = 0.1,
optimizer = "LBFGS",
learn_rate = 1,
momentum = 0,
batch_size = NULL,
stop_iter = 5,
verbose = FALSE,
...)

## S3 method for class 'recipe'
brulee_linear_reg(
x,
data,
epochs = 20L,
penalty = 0.001,
mixture = 0,
validation = 0.1,
optimizer = "LBFGS",
learn_rate = 1,
momentum = 0,
batch_size = NULL,
stop_iter = 5,
verbose = FALSE,
...
)

Arguments

x       Depending on the context:
   • A data frame of predictors.
   • A matrix of predictors.
   • A recipe specifying a set of preprocessing steps created from recipes::recipe().

The predictor data should be standardized (e.g. centered or scaled).

... Options to pass to the learning rate schedulers via set_learn_rate(). For example, the reduction or steps arguments to schedule_step() could be passed here.

y       When x is a data frame or matrix, y is the outcome specified as:
   • A data frame with 1 numeric column.
   • A matrix with 1 numeric column.
   • A numeric vector.

ePOCHS An integer for the number of epochs of training.

penalty The amount of weight decay (i.e., L2 regularization).
mixture Proportion of Lasso Penalty (type: double, default: 0.0). A value of mixture = 1 corresponds to a pure lasso model, while mixture = 0 indicates ridge regression (a.k.a weight decay).

validation The proportion of the data randomly assigned to a validation set.
optimizer  The method used in the optimization procedure. Possible choices are 'LBFGS' and 'SGD'. Default is 'LBFGS'.
learn_rate  A positive number that controls the initial rapidity that the model moves along the descent path. Values around 0.1 or less are typical.
momentum  A positive number usually on \([0.50, 0.99]\) for the momentum parameter in gradient descent. (\texttt{optimizer = "SGD"} only)
batch_size  An integer for the number of training set points in each batch. (\texttt{optimizer = "SGD"} only)
stop_iter  A non-negative integer for how many iterations with no improvement before stopping.
verbose  A logical that prints out the iteration history.
formula  A formula specifying the outcome term(s) on the left-hand side, and the predictor term(s) on the right-hand side.
data  When a \texttt{recipe} or \texttt{formula} is used, \texttt{data} is specified as:
   - A \texttt{data frame} containing both the predictors and the outcome.

Details

This function fits a linear combination of coefficients and predictors to model the numeric outcome. The training process optimizes the mean squared error loss function.

The function internally standardizes the outcome data to have mean zero and a standard deviation of one. The prediction function creates predictions on the original scale.

By default, training halts when the validation loss increases for at least \texttt{step_iter} iterations. If \texttt{validation = 0} the training set loss is used.

The predictors data should all be numeric and encoded in the same units (e.g. standardized to the same range or distribution). If there are factor predictors, use a recipe or formula to create indicator variables (or some other method) to make them numeric. Predictors should be in the same units before training.

The model objects are saved for each epoch so that the number of epochs can be efficiently tuned. Both the \texttt{coef()} and \texttt{predict()} methods for this model have an \texttt{epoch} argument (which defaults to the epoch with the best loss value).

The use of the L1 penalty (a.k.a. the lasso penalty) does \textit{not} force parameters to be strictly zero (as it does in packages such as \texttt{glmnet}). The zeroing out of parameters is a specific feature the optimization method used in those packages.

Value

A \texttt{brulee_linear_reg} object with elements:
   - \texttt{models_obj}: a serialized raw vector for the torch module.
   - \texttt{estimates}: a list of matrices with the model parameter estimates per epoch.
   - \texttt{best_epoch}: an integer for the epoch with the smallest loss.
   - \texttt{loss}: A vector of loss values (MSE) at each epoch.
   - \texttt{dim}: A list of data dimensions.
- `y_stats`: A list of summary statistics for numeric outcomes.
- `parameters`: A list of some tuning parameter values.
- `blueprint`: The hardhat blueprint data.

See Also

`predict.brulee_linear_reg()`, `coef.brulee_linear_reg()`, `autoplot.brulee_linear_reg()`

Examples

```r
if (torch::torch_is_installed()) {
  ## -----------------------------------------------------------------------------
  data(ames, package = "modeldata")
  ames$Sale_Price <- log10(ames$Sale_Price)
  set.seed(122)
  in_train <- sample(1:nrow(ames), 2000)
  ames_train <- ames[ in_train,]
  ames_test <- ames[-in_train,]

  # Using matrices
  set.seed(1)
  brulee_linear_reg(x = as.matrix(ames_train[, c("Longitude", "Latitude")]),
                   y = ames_train$Sale_Price,
                   penalty = 0.10, epochs = 1, batch_size = 64)

  # Using recipe
  library(recipes)
  ames_rec <- recipe(Sale_Price ~ Bldg_Type + Neighborhood + Year_Built + Gr_Liv_Area +
                     Full_Bath + Year_Sold + Lot_Area + Central_Air + Longitude + Latitude,
                     data = ames_train) %>%
  # Transform some highly skewed predictors
  step_BoxCox(Lot_Area, Gr_Liv_Area) %>%
  # Lump some rarely occurring categories into "other"
  step_other(Neighborhood, threshold = 0.05) %>%
  # Encode categorical predictors as binary.
  step_dummy(all_nominal_predictors(), one_hot = TRUE) %>%
  # Add an interaction effect:
  step_interact(~ starts_with("Central_Air") : Year_Built) %>%
  step_zv(all_predictors()) %>%
  step_normalize(all_numeric_predictors())

  set.seed(2)
  fit <- brulee_linear_reg(ames_rec, data = ames_train,
                           epochs = 5, batch_size = 32)
```
library(ggplot2)

predict(fit, ames_test) %>%
  bind_cols(ames_test) %>%
ggplot(aes(x = .pred, y = Sale_Price)) +
geom_abline(col = "green") +
geom_point(alpha = .3) +
lims(x = c(4, 6), y = c(4, 6)) +
  coord_fixed(ratio = 1)

library(yardstick)
predict(fit, ames_test) %>%
  bind_cols(ames_test) %>%
rmse(Sale_Price, .pred)

brulee_logistic_reg

Fit a logistic regression model

Description

brulee_logistic_reg() fits a model.

Usage

brulee_logistic_reg(x, ...)

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

## S3 method for class 'data.frame'
brulee_logistic_reg(
  x,
  y,
  epochs = 20L,
  penalty = 0.001,
  mixture = 0,
  validation = 0.1,
  optimizer = "LBFGS",
  learn_rate = 1,
  momentum = 0,
  batch_size = NULL,
class_weights = NULL,
stop_iter = 5,
verbose = FALSE,
...
)

## S3 method for class 'matrix'
brulee_logistic_reg(
x,
y,
epochs = 20L,
penalty = 0.001,
mixture = 0,
validation = 0.1,
optimizer = "LBFGS",
learn_rate = 1,
momentum = 0,
batch_size = NULL,
class_weights = NULL,
stop_iter = 5,
verbose = FALSE,
...
)

## S3 method for class 'formula'
brulee_logistic_reg(
  formula,
data,
epochs = 20L,
penalty = 0.001,
mixture = 0,
validation = 0.1,
optimizer = "LBFGS",
learn_rate = 1,
momentum = 0,
batch_size = NULL,
class_weights = NULL,
stop_iter = 5,
verbose = FALSE,
...
)

## S3 method for class 'recipe'
brulee_logistic_reg(
x,
data,
epochs = 20L,
penalty = 0.001,
mixture = 0, 
validation = 0.1, 
optimizer = "LBFGS",
learn_rate = 1, 
momentum = 0, 
batch_size = NULL, 
class_weights = NULL, 
stop_iter = 5, 
verbose = FALSE,
... 
)

Arguments

x Depending on the context:

- A data frame of predictors.
- A matrix of predictors.
- A recipe specifying a set of preprocessing steps created from recipes::recipe().

The predictor data should be standardized (e.g. centered or scaled).

... Options to pass to the learning rate schedulers via set_learn_rate(). For example, the reduction or steps arguments to schedule_step() could be passed here.

y When x is a data frame or matrix, y is the outcome specified as:

- A data frame with 1 factor column (with two levels).
- A matrix with 1 factor column (with two levels).
- A factor vector (with two levels).

epochs An integer for the number of epochs of training.

penalty The amount of weight decay (i.e., L2 regularization).

mixture Proportion of Lasso Penalty (type: double, default: 0.0). A value of mixture = 1 corresponds to a pure lasso model, while mixture = 0 indicates ridge regression (a.k.a weight decay).

validation The proportion of the data randomly assigned to a validation set.

optimizer The method used in the optimization procedure. Possible choices are 'LBFGS' and 'SGD'. Default is 'LBFGS'.

learn_rate A positive number that controls the rapidity that the model moves along the descent path. Values around 0.1 or less are typical. (optimizer = "SGD" only)

momentum A positive number usually on [0.50, 0.99] for the momentum parameter in gradient descent. (optimizer = "SGD" only)

batch_size An integer for the number of training set points in each batch. (optimizer = "SGD" only)

class_weights Numeric class weights (classification only). The value can be:

- A named numeric vector (in any order) where the names are the outcome factor levels.
• An unnamed numeric vector assumed to be in the same order as the outcome factor levels.
• A single numeric value for the least frequent class in the training data and all other classes receive a weight of one.

**stop_iter**
A non-negative integer for how many iterations with no improvement before stopping.

**verbose**
A logical that prints out the iteration history.

**formula**
A formula specifying the outcome term(s) on the left-hand side, and the predictor term(s) on the right-hand side.

**data**
When a **recipe** or **formula** is used, data is specified as:
• A **data frame** containing both the predictors and the outcome.

### Details
This function fits a linear combination of coefficients and predictors to model the log odds of the classes. The training process optimizes the cross-entropy loss function (a.k.a Bernoulli loss).

By default, training halts when the validation loss increases for at least **step_iter** iterations. If validation = 0 the training set loss is used.

The predictors data should all be numeric and encoded in the same units (e.g. standardized to the same range or distribution). If there are factor predictors, use a recipe or formula to create indicator variables (or some other method) to make them numeric. Predictors should be in the same units before training.

The model objects are saved for each epoch so that the number of epochs can be efficiently tuned. Both the **coef()** and **predict()** methods for this model have an epoch argument (which defaults to the epoch with the best loss value).

The use of the L1 penalty (a.k.a. the lasso penalty) does **not** force parameters to be strictly zero (as it does in packages such as **glmnet**). The zeroing out of parameters is a specific feature the optimization method used in those packages.

### Value
A **brulee_logistic_reg** object with elements:
• **models_obj**: a serialized raw vector for the torch module.
• **estimates**: a list of matrices with the model parameter estimates per epoch.
• **best_epoch**: an integer for the epoch with the smallest loss.
• **loss**: A vector of loss values (MSE for regression, negative log-likelihood for classification) at each epoch.
• **dim**: A list of data dimensions.
• **parameters**: A list of some tuning parameter values.
• **blueprint**: The hardhat blueprint data.

### See Also
predict.brulee_logistic_reg(), coef.brulee_logistic_reg(), autoplot.brulee_logistic_reg()
### brulee_mlp

#### Examples

```r
if (torch::torch_is_installed()) {

library(recipes)
library(yardstick)

### # increase # epochs to get better results

data(cells, package = "modeldata")

set.seed(122)
in_train <- sample(1:nrow(cells), 1000)
cells_train <- cells[in_train,]
cells_test <- cells[-in_train,]

# Using matrices
set.seed(1)
brulee_logistic_reg(x = as.matrix(cells_train[, c("fiber_width_ch_1", "width_ch_1")]),
y = cells_train$class,
penalty = 0.10, epochs = 3)

# Using recipe
library(recipes)

library(recipes)

cells_rec <-
recipe(class ~ ., data = cells_train) %>%
  # Transform some highly skewed predictors
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_pca(all_numeric_predictors(), num_comp = 10)

set.seed(2)
fit <- brulee_logistic_reg(cells_rec, data = cells_train,
penalty = .01, epochs = 5)

fit

autoplot(fit)

library(yardstick)
predict(fit, cells_test, type = "prob") %>%
bind_cols(cells_test) %>%
roc_auc(class, .pred_PS)
}
```

---

**brulee_mlp**  
*Fit neural networks*
Description

brulee_mlp() fits neural network models using stochastic gradient descent. Multiple layers can be used.

Usage

brulee_mlp(x, ...)

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

## S3 method for class 'data.frame'
brulee_mlp(
  x,
  y,
  epochs = 100L,
  hidden_units = 3L,
  activation = "relu",
  penalty = 0.001,
  mixture = 0,
  dropout = 0,
  validation = 0.1,
  optimizer = "LBFGS",
  learn_rate = 0.01,
  rate_schedule = "none",
  momentum = 0,
  batch_size = NULL,
  class_weights = NULL,
  stop_iter = 5,
  verbose = FALSE,
  ...
)

## S3 method for class 'matrix'
brulee_mlp(
  x,
  y,
  epochs = 100L,
  hidden_units = 3L,
  activation = "relu",
  penalty = 0.001,
  mixture = 0,
  dropout = 0,
  validation = 0.1,
  optimizer = "LBFGS",
  learn_rate = 0.01,
  rate_schedule = "none",
  momentum = 0,
brulee_mlp

batch_size = NULL,
class_weights = NULL,
stop_iter = 5,
verbose = FALSE,
...
)

## S3 method for class 'formula'
brulee_mlp(
  formula,
data,
epochs = 100L,
hidden_units = 3L,
activation = "relu",
penalty = 0.001,
mixture = 0,
dropout = 0,
validation = 0.1,
optimizer = "LBFGS",
learn_rate = 0.01,
ratio_schedule = "none",
momentum = 0,
batch_size = NULL,
class_weights = NULL,
stop_iter = 5,
verbose = FALSE,
...
)

## S3 method for class 'recipe'
brulee_mlp(
  x,
data,
epochs = 100L,
hidden_units = 3L,
activation = "relu",
penalty = 0.001,
mixture = 0,
dropout = 0,
validation = 0.1,
optimizer = "LBFGS",
learn_rate = 0.01,
ratio_schedule = "none",
momentum = 0,
batch_size = NULL,
class_weights = NULL,
stop_iter = 5,
verbose = FALSE,
Arguments

\(x\)

Depending on the context:

- A **data frame** of predictors.
- A **matrix** of predictors.
- A **recipe** specifying a set of preprocessing steps created from `recipes::recipe()`.

The predictor data should be standardized (e.g. centered or scaled).

\(y\)

When \(x\) is a **data frame** or **matrix**, \(y\) is the outcome specified as:

- A **data frame** with 1 column (numeric or factor).
- A **matrix** with numeric column (numeric or factor).
- A **vector** (numeric or factor).

**epochs**

An integer for the number of epochs of training.

**hidden_units**

An integer for the number of hidden units, or a vector of integers. If a vector of integers, the model will have \(\text{length(hidden_units)}\) layers each with \(\text{hidden_units}[i]\) hidden units.

**activation**

A string for the activation function. Possible values are "relu", "elu", "tanh", and "linear". If `hidden_units` is a vector, `activation` can be a character vector with length equals to \(\text{length(hidden_units)}\) specifying the activation for each hidden layer.

**penalty**

The amount of weight decay (i.e., L2 regularization).

**mixture**

Proportion of Lasso Penalty (type: double, default: 0.0). A value of mixture = 1 corresponds to a pure lasso model, while mixture = 0 indicates ridge regression (a.k.a weight decay).

**dropout**

The proportion of parameters set to zero.

**validation**

The proportion of the data randomly assigned to a validation set.

**optimizer**

The method used in the optimization procedure. Possible choices are 'LBFGS' and 'SGD'. Default is 'LBFGS'.

**learn_rate**

A positive number that controls the initial rapidity that the model moves along the descent path. Values around 0.1 or less are typical.

**rate_schedule**

A single character value for how the learning rate should change as the optimization proceeds. Possible values are "none" (the default), "decay_time", "decay_expo", "cyclic" and "step". See `schedule_decay_time()` for more details.

**momentum**

A positive number usually on \([0.50, 0.99]\) for the momentum parameter in gradient descent. (\(\text{optimizer = "SGD"}\) only)

**batch_size**

An integer for the number of training set points in each batch. (\(\text{optimizer = "SGD"}\) only)
**class_weights**  Numeric class weights (classification only). The value can be:
- A named numeric vector (in any order) where the names are the outcome factor levels.
- An unnamed numeric vector assumed to be in the same order as the outcome factor levels.
- A single numeric value for the least frequent class in the training data and all other classes receive a weight of one.

**stop_iter**  A non-negative integer for how many iterations with no improvement before stopping.

**verbose**  A logical that prints out the iteration history.

**formula**  A formula specifying the outcome term(s) on the left-hand side, and the predictor term(s) on the right-hand side.

**data**  When a *recipe* or *formula* is used, data is specified as:
- A **data frame** containing both the predictors and the outcome.

**Details**

This function fits feed-forward neural network models for regression (when the outcome is a number) or classification (a factor). For regression, the mean squared error is optimized and cross-entropy is the loss function for classification.

When the outcome is a number, the function internally standardizes the outcome data to have mean zero and a standard deviation of one. The prediction function creates predictions on the original scale.

By default, training halts when the validation loss increases for at least `stop_iter` iterations. If `validation = 0` the training set loss is used.

The *predictors* data should all be numeric and encoded in the same units (e.g. standardized to the same range or distribution). If there are factor predictors, use a recipe or formula to create indicator variables (or some other method) to make them numeric. Predictors should be in the same units before training.

The model objects are saved for each epoch so that the number of epochs can be efficiently tuned. Both the `coef()` and `predict()` methods for this model have an epoch argument (which defaults to the epoch with the best loss value).

The use of the L1 penalty (a.k.a. the lasso penalty) does not force parameters to be strictly zero (as it does in packages such as glmnet). The zeroing out of parameters is a specific feature the optimization method used in those packages.

**Learning Rates:**
The learning rate can be set to constant (the default) or dynamically set via a learning rate scheduler (via the `rate_schedule`). Using `rate_schedule = 'none'` uses the `learn_rate` argument. Otherwise, any arguments to the schedulers can be passed via ....

**Value**

A `brulee_mlp` object with elements:
- `models_obj`: a serialized raw vector for the torch module.
• estimates: a list of matrices with the model parameter estimates per epoch.
• best_epoch: an integer for the epoch with the smallest loss.
• loss: A vector of loss values (MSE for regression, negative log-likelihood for classification) at each epoch.
• dim: A list of data dimensions.
• y_stats: A list of summary statistics for numeric outcomes.
• parameters: A list of some tuning parameter values.
• blueprint: The hardhat blueprint data.

See Also

predict.brulee_mlp(), coef.brulee_mlp(), autoplot.brulee_mlp()

Examples

if (torch::torch_is_installed()) {

## -----------------------------------------------------------------------------
# regression examples (increase # epochs to get better results)

data(ames, package = "modeldata")
ames$Sale_Price <- log10(ames$Sale_Price)
set.seed(122)
in_train <- sample(1:nrow(ames), 2000)
ames_train <- ames[in_train,]
ames_test <- ames[-in_train,]

# Using matrices
set.seed(1)
fit <-
  brulee_mlp(x = as.matrix(ames_train[, c("Longitude", "Latitude")]),
y = ames_train$Sale_Price, penalty = 0.10)

# Using recipe
library(recipes)
ames_rec <-
  recipe(Sale_Price ~ Bldg_Type + Neighborhood + Year_Built + Gr_Liv_Area +
  Full_Bath + Year_Sold + Lot_Area + Central_Air + Longitude + Latitude,
data = ames_train) %>%
# Transform some highly skewed predictors
  step_BoxCox(Lot_Area, Gr_Liv_Area) %>%
# Lump some rarely occurring categories into “other”
  step_other(Neighborhood, threshold = 0.05) %>%
# Encode categorical predictors as binary.
  step_dummy(all_nominal_predictors(), one_hot = TRUE) %>
# Add an interaction effect:
step_interact(~ starts_with("Central_Air"):Year_Built) %>%
step_zv(all_predictors()) %>%
step_normalize(all_numeric_predictors())

set.seed(2)
fit <- brulee_mlp(ames_rec, data = ames_train, hidden_units = 20,
dropout = 0.05, rate_schedule = "cyclic", step_size = 4)
fit

autoplot(fit)

library(ggplot2)
predict(fit, ames_test) %>%
bind_cols(ames_test) %>%
ggplot(aes(x = .pred, y = Sale_Price)) +
geom_abline(col = "green") +
geom_point(alpha = .3) +
lims(x = c(4, 6), y = c(4, 6)) +
coord_fixed(ratio = 1)

library(yardstick)
predict(fit, ames_test) %>%
bind_cols(ames_test) %>%
rmse(Sale_Price, .pred)

# classification

library(dplyr)
library(ggplot2)
data("parabolic", package = "modeldata")

set.seed(1)
in_train <- sample(1:nrow(parabolic), 300)
parabolic_tr <- parabolic[sub.in_train,]
parabolic_te <- parabolic[!in_train,]

set.seed(2)
cls_fit <- brulee_mlp(class ~ ., data = parabolic_tr, hidden_units = 2,
epochs = 200L, learn_rate = 0.1, activation = "elu",
penalty = 0.1, batch_size = 2^8)
autoplot(cls_fit)

grid_points <- seq(-4, 4, length.out = 100)
grid <- expand.grid(X1 = grid_points, X2 = grid_points)
predict(cls_fit, grid, type = "prob") %>%
bind_cols(grid) %>>%

brulee_multinomial_reg

Fit a multinomial regression model

Description

brulee_multinomial_reg() fits a model.

Usage

brulee_multinomial_reg(x, ...)

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

## S3 method for class 'data.frame'
brulee_multinomial_reg(
x,
y,
epochs = 20L,
penalty = 0.001,
mixture = 0,
validation = 0.1,
opimizer = "LBFGS",
learn_rate = 1,
momentum = 0,
batch_size = NULL,
class_weights = NULL,
stop_iter = 5,
verbose = FALSE,
...
)

## S3 method for class 'matrix'
brulee_multinomial_reg(
x,
y,
epochs = 20L,
penalty = 0.001,
mixture = 0,
brulee_multinomial_reg

validation = 0.1,
optimizer = "LBFGS",
learn_rate = 1,
momentum = 0,
batch_size = NULL,
class_weights = NULL,
stop_iter = 5,
verbose = FALSE,
...
)

## S3 method for class 'formula'
brulee_multinomial_reg(
  formula,
data,
epochs = 20L,
penalty = 0.001,
mixture = 0,
validation = 0.1,
optimizer = "LBFGS",
learn_rate = 1,
momentum = 0,
batch_size = NULL,
class_weights = NULL,
stop_iter = 5,
verbose = FALSE,
...
)

## S3 method for class 'recipe'
brulee_multinomial_reg(
x,
data,
epochs = 20L,
penalty = 0.001,
mixture = 0,
validation = 0.1,
optimizer = "LBFGS",
learn_rate = 1,
momentum = 0,
batch_size = NULL,
class_weights = NULL,
stop_iter = 5,
verbose = FALSE,
...
)
Arguments

\(x\)
- Depending on the context:
  - A **data frame** of predictors.
  - A **matrix** of predictors.
  - A **recipe** specifying a set of preprocessing steps created from `recipes::recipe()`.
- The predictor data should be standardized (e.g. centered or scaled).

\(y\)
- When \(x\) is a **data frame** or **matrix**, \(y\) is the outcome specified as:
  - A **data frame** with 1 factor column (with three or more levels).
  - A **matrix** with 1 factor column (with three or more levels).
  - A **factor vector** (with three or more levels).

\(epochs\)
- An integer for the number of epochs of training.

\(penalty\)
- The amount of weight decay (i.e., L2 regularization).

\(mixture\)
- Proportion of Lasso Penalty (type: double, default: 0.0). A value of mixture = 1 corresponds to a pure lasso model, while mixture = 0 indicates ridge regression (a.k.a weight decay).

\(validation\)
- The proportion of the data randomly assigned to a validation set.

\(optimizer\)
- The method used in the optimization procedure. Possible choices are ‘LBFGS’ and ‘SGD’. Default is ‘LBFGS’.

\(learn_rate\)
- A positive number that controls the rapidity that the model moves along the descent path. Values around 0.1 or less are typical. (optimizer = “SGD” only)

\(momentum\)
- A positive number usually on \([0.50, 0.99]\) for the momentum parameter in gradient descent. (optimizer = “SGD” only)

\(batch_size\)
- An integer for the number of training set points in each batch. (optimizer = “SGD” only)

\(class_weights\)
- Numeric class weights (classification only). The value can be:
  - A named numeric vector (in any order) where the names are the outcome factor levels.
  - An unnamed numeric vector assumed to be in the same order as the outcome factor levels.
  - A single numeric value for the least frequent class in the training data and all other classes receive a weight of one.

\(stop_iter\)
- A non-negative integer for how many iterations with no improvement before stopping.

\(verbose\)
- A logical that prints out the iteration history.

\(formula\)
- A formula specifying the outcome term(s) on the left-hand side, and the predictor term(s) on the right-hand side.

\(data\)
- When a **recipe** or **formula** is used, data is specified as:
  - A **data frame** containing both the predictors and the outcome.
Details

This function fits a linear combination of coefficients and predictors to model the log of the class probabilities. The training process optimizes the cross-entropy loss function.

By default, training halts when the validation loss increases for at least step_iter iterations. If validation = 0 the training set loss is used.

The predictors data should all be numeric and encoded in the same units (e.g. standardized to the same range or distribution). If there are factor predictors, use a recipe or formula to create indicator variables (or some other method) to make them numeric. Predictors should be in the same units before training.

The model objects are saved for each epoch so that the number of epochs can be efficiently tuned. Both the coef() and predict() methods for this model have an epoch argument (which defaults to the epoch with the best loss value).

The use of the L1 penalty (a.k.a. the lasso penalty) does not force parameters to be strictly zero (as it does in packages such as glmnet). The zeroing out of parameters is a specific feature the optimization method used in those packages.

Value

A brulee_multinomial_reg object with elements:

- models_obj: a serialized raw vector for the torch module.
- estimates: a list of matrices with the model parameter estimates per epoch.
- best_epoch: an integer for the epoch with the smallest loss.
- loss: A vector of loss values (MSE for regression, negative log-likelihood for classification) at each epoch.
- dim: A list of data dimensions.
- parameters: A list of some tuning parameter values.
- blueprint: The hardhat blueprint data.

See Also

predict.brulee_multinomial_reg(), coef.brulee_multinomial_reg(), autoplot.brulee_multinomial_reg()

Examples

```r
if (torch::torch_is_installed()) {
  library(recipes)
  library(yardstick)

  data(penguins, package = "modeldata")

  penguins <- penguins %>% na.omit()

  set.seed(122)
  in_train <- sample(1:nrow(penguins), 200)
```
penguins_train <- penguins[ in_train,]
penguins_test  <- penguins[-in_train,]

rec <- recipe(island ~ ., data = penguins_train) %>%
  step_dummy(species, sex) %>%
  step_normalize(all_predictors())

set.seed(3)
fit <- brulee_multinomial_reg(rec, data = penguins_train, epochs = 5)
fit

predict(fit, penguins_test) %>%
  bind_cols(penguins_test) %>%
  conf_mat(island, .pred_class)

---

matrix_to_dataset  
Convert data to torch format

Description
For an x/y interface, matrix_to_dataset() converts the data to proper encodings then formats the results for consumption by torch.

Usage
matrix_to_dataset(x, y)

Arguments
x      A numeric matrix of predictors.
y      A vector. If regression than y is numeric. For classification, it is a factor.

Details
Missing values should be removed before passing data to this function.

Value
An R6 index sampler object with classes "training_set", "dataset", and "R6".

Examples
if (torch::torch_is_installed()) {
  matrix_to_dataset(as.matrix(mtcars[, -1]), mtcars$mpg)
}
predict.brulee_linear_reg

Predict from a brulee_linear_reg

Description

Predict from a brulee_linear_reg

Usage

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

Arguments

object A brulee_linear_reg object.

new_data A data frame or matrix of new predictors.

type A single character. The type of predictions to generate. Valid options are:

• "numeric" for numeric predictions.

epoch An integer for the epoch to make predictions. If this value is larger than the maximum number that was fit, a warning is issued and the parameters from the last epoch are used. If left NULL, the epoch associated with the smallest loss is used.

... Not used, but required for extensibility.

Value

A tibble of predictions. The number of rows in the tibble is guaranteed to be the same as the number of rows in new_data.

Examples

if (torch::torch_is_installed()) {

data(ames, package = "modeldata")

ames$Sale_Price <- log10(ames$Sale_Price)

set.seed(1)
in_train <- sample(!:nrow(ames), 2000)
ames_train <- ames[ in_train,]
ames_test  <- ames[-in_train,]

# Using recipe
library(recipes)
ames_rec <-
  recipe(Sale_Price ~ Longitude + Latitude, data = ames_train) %>%
  step_normalize(all_numeric_predictors())

set.seed(2)
fit <- brulee_linear_reg(ames_rec, data = ames_train,
  epochs = 50, batch_size = 32)

predict(fit, ames_test)

---

### predict.brulee_logistic_reg

Predict from a `brulee_logistic_reg`

#### Description

Predict from a `brulee_logistic_reg`

#### Usage

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

#### Arguments

- **object**
  - A `brulee_logistic_reg` object.

- **new_data**
  - A data frame or matrix of new predictors.

- **type**
  - A single character. The type of predictions to generate. Valid options are:
    - "class" for hard class predictions
    - "prob" for soft class predictions (i.e., class probabilities)

- **epoch**
  - An integer for the epoch to make predictions. If this value is larger than the maximum number that was fit, a warning is issued and the parameters from the last epoch are used. If left `NULL`, the epoch associated with the smallest loss is used.

- **...**
  - Not used, but required for extensibility.

#### Value

A tibble of predictions. The number of rows in the tibble is guaranteed to be the same as the number of rows in `new_data`. 
Examples

```r
if (torch::torch_is_installed()) {

library(recipes)
library(yardstick)

data(penguins, package = "modeldata")
penguins <- penguins %>% na.omit()

set.seed(122)
in_train <- sample(1:nrow(penguins), 200)
penguins_train <- penguins[, in_train]
penguins_test <- penguins[-in_train]

rec <- recipe(sex ~ ., data = penguins_train) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_numeric_predictors())

set.seed(3)
fit <- brulee_logistic_reg(rec, data = penguins_train, epochs = 5)
fit

predict(fit, penguins_test)

predict(fit, penguins_test, type = "prob") %>%
  bind_cols(penguins_test) %>%
  roc_curve(sex, .pred_female) %>%
  autoplot()
}
```

---

**predict.brulee_mlp**  
*Predict from a brulee_mlp*

### Description

Predict from a brulee_mlp

### Usage

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

### Arguments

- **object**  
  A brulee_mlp object.
new_data A data frame or matrix of new predictors.
type A single character. The type of predictions to generate. Valid options are:
  • "numeric" for numeric predictions.
  • "class" for hard class predictions
  • "prob" for soft class predictions (i.e., class probabilities)
epoch An integer for the epoch to make predictions. If this value is larger than the
  maximum number that was fit, a warning is issued and the parameters from the
  last epoch are used. If left NULL, the epoch associated with the smallest loss is
  used.
... Not used, but required for extensibility.

Value
A tibble of predictions. The number of rows in the tibble is guaranteed to be the same as the number
of rows in new_data.

Examples

```r
if (torch::torch_is_installed()) {
  # regression example:
  data(ames, package = "modeldata")

  ames$Sale_Price <- log10(ames$Sale_Price)

  set.seed(1)
  in_train <- sample(1:nrow(ames), 2000)
  ames_train <- ames[in_train,]
  ames_test <- ames[-in_train,]

  # Using recipe
  library(recipes)

  ames_rec <-
    recipe(Sale_Price ~ Longitude + Latitude, data = ames_train) %>%
      step_normalize(all_numeric_predictors())

  set.seed(2)
  fit <- brulee_mlp(ames_rec, data = ames_train, epochs = 50, batch_size = 32)

  predict(fit, ames_test)
}
```
### predict.brulee_multinomial_reg

**Predict from a brulee_multinomial_reg**

#### Description

Predict from a brulee_multinomial_reg

#### Usage

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

#### Arguments

- `object`: A brulee_multinomial_reg object.
- `new_data`: A data frame or matrix of new predictors.
- `type`: A single character. The type of predictions to generate. Valid options are:
  - "class" for hard class predictions
  - "prob" for soft class predictions (i.e., class probabilities)
- `epoch`: An integer for the epoch to make predictions. If this value is larger than the maximum number that was fit, a warning is issued and the parameters from the last epoch are used. If left NULL, the epoch associated with the smallest loss is used.
- `...`: Not used, but required for extensibility.

#### Value

A tibble of predictions. The number of rows in the tibble is guaranteed to be the same as the number of rows in new_data.

#### Examples

```r
if (torch::torch_is_installed()) {

library(recipes)
library(yardstick)

data(penguins, package = "modeldata")

penguins <- penguins %>% na.omit()

set.seed(122)
in_train <- sample(1:nrow(penguins), 200)
penguins_train <- penguins[ in_train,]
penguins_test  <- penguins[!in_train,]
```
rec <- recipe(island ~ ., data = penguins_train) %>%
  step_dummy(species, sex) %>%
  step_normalize(all_numeric_predictors())

set.seed(3)
fit <- brulee_multinomial_reg(rec, data = penguins_train, epochs = 5)
fitted

predict(fit, penguins_test) %>%
  bind_cols(penguins_test) %>%
  conf_mat(island, .pred_class)

schedule_decay_time

Change the learning rate over time

Description

Learning rate schedulers alter the learning rate to adjust as training proceeds. In most cases, the
learning rate decreases as epochs increase. The schedule_*() functions are individual schedulers
and set_learn_rate() is a general interface.

Usage

schedule_decay_time(epoch, initial = 0.1, decay = 1)
schedule_decay_expo(epoch, initial = 0.1, decay = 1)
schedule_step(epoch, initial = 0.1, reduction = 1/2, steps = 5)
schedule_cyclic(epoch, initial = 0.001, largest = 0.1, step_size = 5)
set_learn_rate(epoch, learn_rate, type = "none", ...)

Arguments

epoch An integer for the number of training epochs (zero being the initial value).
initial A positive numeric value for the starting learning rate.
decay A positive numeric constant for decreasing the rate (see Details below).
reduction A positive numeric constant stating the proportional decrease in the learning rate
  occurring at every steps epochs.
steps The number of epochs before the learning rate changes.
largest The maximum learning rate in the cycle.
step_size The half-length of a cycle.
learn_rate  A constant learning rate (when no scheduler is used),

type  A single character value for the type of scheduler. Possible values are: "decay_time", "decay_expo", "none", "cyclic", and "step".

Arguments to pass to the individual scheduler functions (e.g. reduction).

Details

The details for how the schedulers change the rates:

- `schedule_decay_time()`: \( rate(\text{epoch}) = \frac{\text{initial}}{1 + \text{decay} \times \text{epoch}} \)
- `schedule_decay_expo()`: \( rate(\text{epoch}) = \text{initial} \exp(-\text{decay} \times \text{epoch}) \)
- `schedule_step()`: \( rate(\text{epoch}) = \text{initial} \times \text{reduction}^{\text{floor}(\text{epoch}/\text{steps})} \)
- `schedule_cyclic()`: \( \text{cycle} = \text{floor}(1+(\text{epoch}/2/\text{stepsize})), x = \text{abs}((\text{epoch}/\text{stepsize})-(2 \times \text{cycle}) + 1), \) and \( rate(\text{epoch}) = \text{initial} + (\text{largest} - \text{initial}) \times \text{max}(0, 1 - x) \)

Value

A numeric value for the updated learning rate.

See Also

`brulee_mlp()`

Examples

```r
library(ggplot2)
library(dplyr)
library(purrr)

iters <- 0:50

bind_rows(
  tibble(epoch = iters, rate = map_dbl(iters, schedule_decay_time), type = "decay_time"),
  tibble(epoch = iters, rate = map_dbl(iters, schedule_decay_expo), type = "decay_expo"),
  tibble(epoch = iters, rate = map_dbl(iters, schedule_step), type = "step"),
  tibble(epoch = iters, rate = map_dbl(iters, schedule_cyclic), type = "cyclic")
) %>%
  ggplot(aes(epoch, rate)) +
  geom_line() +
  facet_wrap(~ type)
#
# Use with neural network
```
autoplot.brulee_linear_reg
    (brulee-autoplot), 2
autoplot.brulee_linear_reg(), 8
autoplot.brulee_logistic_reg
    (brulee-autoplot), 2
autoplot.brulee_logistic_reg(), 12
autoplot.brulee_mlp (brulee-autoplot), 2
autoplot.brulee_mlp(), 18
autoplot.brulee_multinomial_reg
    (brulee-autoplot), 2
autoplot.brulee_multinomial_reg(), 23
brulee-autoplot, 2
brulee-coefs, 3
brulee_linear_reg, 4
brulee_logistic_reg, 9
brulee_mlp, 13
brulee_mlp(), 17
brulee_multinomial_reg, 20

coef(), 7, 12, 17, 23
coef.brulee_linear_reg (brulee-coefs), 3
coef.brulee_linear_reg(), 8
coef.brulee_logistic_reg
    (brulee-coefs), 3
coef.brulee_logistic_reg(), 12
coef.brulee_mlp (brulee-coefs), 3
coef.brulee_mlp(), 18
coef.brulee_multinomial_reg
    (brulee-coefs), 3
coef.brulee_multinomial_reg(), 23

matrix_to_dataset, 24

predict(), 7, 12, 17, 23
predict.brulee_linear_reg, 25
predict.brulee_linear_reg(), 8
predict.brulee_logistic_reg, 26
predict.brulee_logistic_reg(), 12
predict.brulee_mlp, 27

recipes::recipe(), 6, 11, 16, 22

schedule_cyclic (schedule_decay_time), 30
schedule_decay_expo
    (schedule_decay_time), 30
schedule_decay_time, 30
schedule_decay_time(), 16
schedule_step (schedule_decay_time), 30
schedule_step(), 6, 11, 16, 22
set_learn_rate (schedule_decay_time), 30
set_learn_rate(), 6, 11, 16, 22, 30