Package 'densemlp'

November 3, 2026

Title Dense Neural Networks for Tabular Regression, Classification and Survival

Version 0.9.0

Description Dense feed-forward neural networks (multilayer perceptrons) for tabular regression, classification and survival analysis, with a formula or x/y interface. Supports residual and gated hidden blocks, batch normalization, per-layer dropout, learned cross-feature interactions, exponential moving-average weights, learning-rate schedules, internal bootstrap ensembles, case weights (for example importance weights under covariate shift) and Adam optimization. Survival outcomes are trained with either a batch-wise Breslow-tie Cox partial likelihood or a discrete-time inverse-probability-of-censoring-weighted integrated Brier score. The numerical kernels are implemented natively in C++ via 'RcppArmadillo', with no external deep learning framework dependency (no 'torch' / 'libtorch'). Companion helpers provide k-fold cross-validation, hyperparameter search and task-aware evaluation metrics.

URL https://CRAN.R-project.org/package=densemlp

BugReports https://github.com/ielbadisy/densemlp/issues

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Imports graphics, parallel, Rcpp, RhpcBLASctl, stats, utils

LinkingTo Rcpp, RcppArmadillo

Suggests knitr, rmarkdown, survival, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation yes

Author Imad El Badisy [aut, cre]

Maintainer Imad El Badisy <elbadisyimad@gmail.com>

Repository CRAN

Date/Publication 2026-11-03 00:00:02 UTC

Contents

cv_densemlp 2
densemlp 3
densemlp_integrated_brier_score 8
densemlp_metrics 8
group_effects 9
perm_importance 10
plot.densemlp 11
plot.densemlp_importance 11
plot_history 12
predict.densemlp 12
tune_densemlp 13

Index **15**

`cv_densemlp` *Cross-validate a densemlp model*

Description

Fits `densemlp()` on each of `folds` training splits and evaluates it on the held-out fold via `densemlp_metrics()`.

Usage

```
cv_densemlp(
  x,
  y,
  task = c("auto", "regression", "binary", "multiclass", "survival", "count"),
  folds = 5L,
  seed = 1L,
  ncores = 1L,
  verbose = FALSE,
  weights = NULL,
  ...
)
```

Arguments

`x`	Predictor data.frame or matrix.
`y`	Outcome vector.
`task`	`"auto"` infers the task from `y`, as in `densemlp()`. Pass `"survival"` explicitly for a survival outcome.
`folds`	Number of cross-validation folds.
`seed`	Random seed for fold assignment; fold `k` fits with `seed = seed + k`.
`ncores`	Number of cores used to fit folds in parallel (see `densemlp()`'s `ncores`).

verbose	Print per-fold progress.
weights	Optional non-negative case weights, one per row. Each fold is fitted with its training rows' weights (see densemlp()) and scored with weighted metrics on its held-out rows (see densemlp_metrics()), which gives importance-weighted cross-validation under covariate shift.
...	Additional arguments passed to densemlp() for every fold (e.g. hidden_units, epochs, lr).

Value

A list of class densemlp_cv with fold_metrics (one row per fold), summary (mean and SD per metric across folds), and task.

Examples

```
set.seed(1)
x <- data.frame(a = rnorm(60), b = rnorm(60))
y <- x$a - 0.5 * x$b + rnorm(60, sd = 0.1)
cv <- cv_densemlp(x, y, folds = 3, epochs = 20, hidden_units = c(8))
cv$summary
```

---

densemlp *Fit a fast dense multilayer perceptron*

Description

A compact feedforward network for regression and classification on tabular data. Forward propagation, backpropagation, and Adam optimization are all implemented natively in C++ via RcppArmadillo – no torch / libtorch dependency.

Usage

```
densemlp(
  x = NULL,
  y = NULL,
  task = c("auto", "regression", "binary", "multiclass", "survival", "count"),
  loss = c("cox", "brier"),
  n_bins = 10L,
  hidden_units = c(32, 16),
  epochs = 100L,
  batch_size = 32L,
  lr = 0.001,
  validation = 0.2,
  early_stopping = TRUE,
  patience = 10L,
  min_delta = 0,
  min_epochs = max(10L, floor(epochs * 0.2)),
```

```
  residual = FALSE,
  gated = FALSE,
  dropout = 0,
  batch_norm = TRUE,
  input_projection = NULL,
  interaction = FALSE,
  ema_decay = 0,
  ensemble = 1L,
  ensemble_bootstrap = TRUE,
  lr_schedule = c("none", "cosine", "step"),
  seed = 1L,
  verbose = FALSE,
  ncores = 1L,
  formula = NULL,
  data = NULL,
  offset = NULL,
  groups = NULL,
  group_penalty = 0,
  weights = NULL
)

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

Arguments

x	A densemlp object.
y	Outcome (x/y interface): numeric for regression, a factor/character (2 levels for binary, 3+ for multiclass) for classification, numeric proportions in [0, 1] for task = "binary" (a fractional / quasi-binomial outcome), non-negative counts for task = "count", or, for task = "survival", a survival::Surv() object or a two-column matrix/data.frame giving (time, event) (event coded 1 = event, 0 = censored).
task	"auto" infers the task from y (a survival::Surv() response, from either interface, is always detected as "survival"); otherwise one of "regression", "binary", "multiclass", "survival", "count" (Poisson log-likelihood with a log link; never inferred, so it must be requested).
loss	For task = "survival" only: "cox" (batch-wise Breslow-tie Cox partial likelihood, a single linear risk score) or "brier" (IPCW integrated Brier score over a discrete-time grid of n_bins hazard outputs; see the "Survival" section below). Ignored otherwise.
n_bins	For task = "survival", loss = "brier" only: number of discrete-time bins (quantile cutpoints of the observed follow-up times).
hidden_units	Integer vector of hidden layer sizes. An empty vector (integer(0)) gives a linear model (a GLM fitted by stochastic gradient descent).
epochs	Maximum number of training epochs (an upper bound when early_stopping is used).

batch_size — Mini-batch size.

lr — Adam learning rate.

validation — Validation fraction in `[0, 1)` held out for early stopping, or an integer vector of at least two row indices to hold out (for example a split by cluster or by time). Set to `0` to disable (trains for the full `epochs`, no BN running-stat evaluation set). Row indices cannot be combined with `ensemble > 1`.

early_stopping — Logical; stop once validation loss stops improving. Ignored if `validation = 0`.

patience — Number of non-improving epochs to wait before stopping.

min_delta — Minimum validation loss improvement to reset patience.

min_epochs — Minimum number of epochs before early stopping can trigger. Defaults to `max(10, floor(epochs * 0.2))`.

residual — Logical; add a residual skip to every hidden block. A learned linear projection is used when the block dimensions differ.

gated — Logical; use a learned sigmoid gate in every hidden block.

dropout — Dropout probability, either one value or one per hidden layer. Values must be in `[0, 1)`.

batch_norm — Logical; apply batch normalization inside every hidden block (`Linear -> BatchNorm -> ReLU`). When `FALSE`, hidden blocks are `Linear -> ReLU` with no normalization and no learned BN affine parameters.

input_projection — Optional positive integer. When set, a plain linear layer (no activation, no batch normalization) maps the encoded predictors to `input_projection` dimensions before the first hidden block. `NULL` (default) disables it. Cannot be combined with `interaction`.

interaction — Logical; prepend an efficient learned cross-feature layer that models explicit second-order interactions in `O(p)` parameters.

ema_decay — Exponential moving-average decay for model parameters. Set to `0` to disable; values such as `0.99` enable EMA evaluation and best-epoch restoration.

ensemble — Number of internally fitted members whose predictions are averaged. `1` fits a single model and preserves the standard behavior.

ensemble_bootstrap — Logical; bootstrap rows independently for each member when `ensemble > 1`.

lr_schedule — Learning-rate schedule: `"none"`, cosine annealing over `epochs`, or `"step"` decay by 0.5 every `max(5, floor(epochs / 3))` epochs.

seed — Integer seed.

verbose — Logical; print training/validation loss every 10 epochs.

ncores — Number of cores used to fit ensemble members in parallel when `ensemble > 1` (via `parallel::mclapply` on Unix-alikes, serially on Windows). Ignored when `ensemble = 1`.

formula — A formula, e.g. `y ~ .` or, for survival, `survival::Surv(time, status) ~ .`, as an alternative to `x`/`y`. Use either `formula`/`data` or `x`/`y`, not both. A formula may be given as the first positional argument – `densemlp(y ~ ., df)`, `densemlp(formula = y ~ ., data = df)` and `densemlp(x, y)` all work.

data	A data.frame used with formula.
offset	Optional fixed term added to the linear predictor (the output before its activation), on the scale of the outcome for regression and of the link otherwise (log for "count", logit for "binary"). A numeric vector with one value per row, or for "multiclass" a matrix with one column per class. Not available for task = "survival".
groups	Optional grouping factor(s): a vector, or a list or data.frame of vectors, each with one value per row. Every level of every grouping factor gets its own learned intercept on the linear predictor (one per class for multiclass), trained jointly with the network. This gives fixed-effect style cluster or period intercepts, as in panel data. Not available for task = "survival".
group_penalty	Non-negative ridge penalty on the group intercepts, added to each mini-batch loss as group_penalty times the mean squared intercept of each grouping factor. Either one value for all grouping factors or one value per factor. 0 (default) leaves them unpenalized.
weights	Optional non-negative case weights, one per row. Each row's contribution to the loss (training and validation) is multiplied by its weight, so the network minimizes a weighted empirical risk (only relative weights matter: weights and c * weights give the same fit), for example importance weights p_target(x) / p_train(x) under covariate shift, survey weights, or inverse-probability weights. Training weights are rescaled to mean one (the learning rate keeps its meaning), batch normalization uses weighted batch statistics, the validation loss used for early stopping is a weighted mean, and survival uses a weighted Breslow partial likelihood or a weighted IPCW Brier score (with a weighted Kaplan-Meier censoring estimate). Numeric inputs (and a regression outcome) are standardized with weighted means and standard deviations, so integer weights behave like repeated rows (up to the n - 1 denominator of the standard deviations). Rows with weight 0 do not contribute. NULL (default) fits unweighted.
...	Unused.

Details

Hidden layers are Linear -> BatchNorm -> ReLU -> optional gate -> optional dropout -> optional residual (batch statistics during training, running mean/var at prediction time, exponential decay 0.9), with He initialization for the linear weights. Set batch_norm = FALSE to drop the normalization step, leaving Linear -> ReLU hidden blocks. Residual blocks use a learned linear projection when dimensions differ. With input_projection = k, a bare linear layer maps the encoded inputs to k dimensions before the first hidden block (mutually exclusive with interaction). The optional interaction layer is a one-layer cross network initialized as the identity. With ema_decay > 0, validation and final predictions use moving-average parameters. With ensemble > 1, fully fitted internal members are trained with successive seeds and optionally bootstrapped rows, and their response probabilities/predictions are averaged transparently. The output layer is plain Linear -> task activation (no BN): linear (regression, MSE loss), sigmoid (binary, binary cross-entropy), softmax (multiclass, categorical cross-entropy), exponential (count, Poisson log-likelihood), or a linear risk score (survival, Cox partial likelihood); the first four share the output-gradient simplification dZ = (yhat - y) / n (times the case weight when weights is given), while survival uses its own closed-form Cox gradient (see the "Survival" section below). With validation > 0 and early_stopping = TRUE

(both on by default), the parameters (weights, biases, and BN affine/running-stat parameters) from the best validation epoch are restored at the end.

`offset` and `groups` both enter the linear predictor additively, after the output layer and before its activation, so the fitted model is `g(E[y]) = f(x) + offset + sum_k alpha_k[group_k]`, where `f` is the network and the `alpha_k` are the per-level intercepts (see `group_effects()`).

Value

A `densemlp` object.

Survival

`task = "survival"` supports two losses, both trained batch-wise against a `(time, event)` outcome:

- `loss = "cox"` (the default) trains a single linear risk score using a batch-wise Breslow-tie Cox partial log-likelihood – the same "risk set = current mini-batch" approximation used by DeepSurv/pycox, which turns Cox regression into an ordinary per-batch loss compatible with masked-free Adam training, BN, and early stopping unchanged. For a batch sorted by descending time, with each observation's risk set approximated by the sorted prefix of observations with a time at least as large, the loss is the negative mean, over events, of the risk score minus the log of the cumulative sum of exponentiated risk scores over its risk set – computed via a single cumulative sum with a closed-form O(n) gradient (no autodiff).
- `loss = "brier"` trains a discrete-time hazard head with `n_bins` outputs (a quantile grid of the observed follow-up times) directly against the IPCW (Graf et al.) integrated Brier score. Bin `k`'s sigmoid output is a conditional hazard, and the survival probability at that bin is the running product of one minus every hazard up to and including it, which is monotonically non-increasing by construction. Each bin's Brier term is inverse-probability-of-censoring weighted using a Kaplan-Meier estimate of the *censoring* distribution fit once on the training data; subjects censored before a bin are excluded from that bin's term, per Graf's correction. The gradient is closed-form (no autodiff): a bin's survival probability depends on every hazard logit at or before it, so each bin's Brier-score gradient is distributed back across all of them.

`predict(fit, newdata, type = "response")` returns a linear risk score for either loss (for `"brier"`, the negative log of the predicted survival probability at the final time bin, so higher is still riskier); with `loss = "brier"`, `type = "survival"` additionally returns the full `n_bins`-column survival-probability matrix. `densemlp_metrics()` reports Harrell's concordance index for `task = "survival"` regardless of loss.

Examples

```
set.seed(1)
x <- data.frame(a = rnorm(100), b = rnorm(100))
y <- x$a - 0.5 * x$b + rnorm(100, sd = 0.1)
fit <- densemlp(x, y, epochs = 50, hidden_units = c(16))
predict(fit, x[1:5, ])
```

densemlp_integrated_brier_score *Integrated Brier score for a fitted Brier-loss survival densemlp model*

Description

The IPCW (Graf et al.) integrated Brier score, evaluated at the model's own time grid (`object$survival_breaks`), using a Kaplan-Meier estimate of the censoring distribution fit on `y` (i.e. on whatever data is passed in – pass the held-out set's own outcome for an honest out-of-sample estimate). Lower is better; this is exactly the objective `densemlp()` minimizes when fit with `task = "survival", loss = "brier"`.

Usage

```
densemlp_integrated_brier_score(object, newdata, y)
```

Arguments

`object`	A `densemlp` object fit with `task = "survival", loss = "brier"`.
`newdata`	Predictor data to evaluate on.
`y`	The corresponding survival outcome (`survival::Surv()` or a two-column `(time, event)` matrix/data.frame).

Value

A single numeric integrated Brier score.

densemlp_metrics *Prediction metrics for a fitted densemlp model*

Description

Prediction metrics for a fitted densemlp model

Usage

```
densemlp_metrics(truth, estimate, task, prob = NULL, weights = NULL)
```

Arguments

`truth`	Observed outcome values, or, for `task = "survival"`, a `survival::Surv()` object or a two-column `(time, event)` matrix/data.frame.
`estimate`	Predicted values (`type = "response"` for regression and survival, `type = "class"` for classification).
`task`	`"regression"`, `"binary"`, `"multiclass"`, `"survival"`, or `"count"` (scored like regression).

prob	Optional matrix of class probabilities (classification only), used to compute macro_auc and log_loss.
weights	Optional non-negative case weights, one per observation. When given, every metric is its weighted version (weighted means, weighted AUC and concordance counting each pair as the product of its two weights), e.g. an importance-weighted estimate of target-population performance under covariate shift.

Value

A named list of metrics: rmse, nrmse, rsq for regression, accuracy, balanced_accuracy, macro_auc, log_loss for classification, or concordance (Harrell's C-index) for survival.

---

group_effects *Extract the learned group intercepts*

Description

Returns the per-level intercepts that a model fitted with groups adds to its linear predictor, on the scale of the outcome for regression and of the link otherwise.

Usage

```
group_effects(object, ...)

## S3 method for class 'densemlp'
group_effects(object, ...)
```

Arguments

object	A fitted densemlp object fitted with groups.
...	Unused.

Value

A named list with one entry per grouping factor: a named numeric vector of intercepts (one per level), or for multiclass a matrix with one row per level and one column per class. For an ensemble, the members' intercepts are averaged.

perm_importance *Permutation variable importance for a fitted densemlp model*

Description

Model-agnostic permutation importance: for each predictor, the column is randomly shuffled and the drop in predictive performance (relative to the unpermuted baseline) is recorded. Larger values mean the model relied more on that predictor.

Usage

```
perm_importance(object, new_data, truth, metric = NULL, seed = object$seed)

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

Arguments

object	A fitted densemlp object (single model; ensembles are not supported).
new_data	Evaluation predictor data.
truth	Ground-truth outcome for new_data: a numeric vector (regression), a factor/character (classification), or a survival::Surv() / two-column (time, event) object (survival).
metric	Metric name understood by densemlp_metrics(). Defaults to "rmse" (regression), "accuracy" (classification) or "concordance" (survival).
seed	Random seed used for the column shuffles.
x	A densemlp_importance object.
...	Unused.

Value

A densemlp_importance object: a list with data (a data frame of feature / importance, ordered by decreasing importance), metric, baseline and task.

Examples

```
set.seed(1)
x <- data.frame(a = rnorm(120), b = rnorm(120), c = rnorm(120))
y <- x$a - 0.5 * x$b + rnorm(120, sd = 0.1)
fit <- densemlp(x, y, epochs = 40, hidden_units = c(16))
perm_importance(fit, x, y)
```

plot.densemlp *Plot training history*

Description

Plot training history

Usage

```
## S3 method for class 'densemlp'
plot(x, ...)
```

Arguments

x	A fitted densemlp object (single model; ensembles are not supported since members don't share an epoch axis).
...	Additional arguments passed to graphics::matplot().

Value

x, invisibly.

plot.densemlp_importance *Plot permutation importance*

Description

Plot permutation importance

Usage

```
## S3 method for class 'densemlp_importance'
plot(x, top = 20L, ...)
```

Arguments

x	A densemlp_importance object.
top	Number of top features to show.
...	Passed to graphics::barplot().

Value

x, invisibly.

plot_history *Plot the training history of a fitted densemlp model*

Description

A named wrapper around the `plot.densemlp()` method: draws the per-epoch training and validation loss curves.

Usage

```
plot_history(object, ...)
```

Arguments

`object`	A fitted `densemlp` object (single model).
`...`	Passed to `graphics::matplot()`.

Value

`object`, invisibly.

Examples

```
set.seed(1)
x <- data.frame(a = rnorm(80), b = rnorm(80))
y <- x$a - 0.5 * x$b + rnorm(80, sd = 0.1)
fit <- densemlp(x, y, epochs = 40, hidden_units = c(16))
plot_history(fit)
```

predict.densemlp *Predict from a fitted densemlp model*

Description

Predict from a fitted densemlp model

Usage

```
## S3 method for class 'densemlp'
predict(
  object,
  newdata,
  type = c("response", "class", "prob", "survival"),
  offset = NULL,
  groups = NULL,
  ...
)
```

Arguments

object	A fitted densemlp object.
newdata	New predictor data, in the same representation used to fit.
type	"response" (regression: unscaled prediction; binary: predicted probability of the second factor level; count: predicted mean; survival: a linear risk score, higher = riskier, for either loss), "class" (classification only: predicted factor label), "prob" (classification only: a matrix of class probabilities), or "survival" (survival with loss = "brier" only: the full n_bins-column survival-probability matrix).
offset	Offset for newdata, in the same form as in densemlp(). NULL (default) uses zero. Supply it whenever the model was fitted with an offset.
groups	Grouping factor(s) for newdata, in the same form as in densemlp(). Levels not seen in training, and every row when groups is NULL, get a zero intercept (a population-level prediction).
...	Unused.

Value

A numeric vector ("response"), a factor ("class"), or a matrix ("prob" or "survival").

---

tune_densemlp *Tune a densemlp model over a hyperparameter grid*

Description

Fits densemlp() for every combination in grid (each repeated repeats times with successive seeds), ranks candidates by their best internal validation loss, and optionally refits the best configuration on the full data.

Usage

```
tune_densemlp(
  x,
  y,
  task = c("auto", "regression", "binary", "multiclass", "survival", "count"),
  grid = NULL,
  validation = 0.2,
  seed = 1L,
  repeats = 3L,
  ncores = 1L,
  verbose = FALSE,
  refit = TRUE,
  weights = NULL
)
```

Arguments

x	Predictor data.frame or matrix.
y	Outcome vector.
task	"auto" infers the task from y, as in densemlp().
grid	A named list of candidate values. Supported names: hidden_units (a list of integer vectors), dropout (a list of numeric vectors, recycled to hidden_units length), residual, gated, ema_decay, lr_schedule, epochs, batch_size, lr. Any name omitted falls back to a single-value default. interaction is intentionally not tunable here: it is numerically unstable on small/wide data (see package NEWS) and is left at FALSE.
validation	Validation fraction used for every candidate fit.
seed	Base random seed.
repeats	Number of repeated seeds per candidate.
ncores	Number of cores used to fit candidates in parallel (see densemlp()'s ncores).
verbose	Print per-candidate progress.
refit	Refit the best configuration on the supplied data.
weights	Optional non-negative case weights, one per row, passed to every densemlp() fit; candidates are then ranked by the weighted validation loss.

Value

A list of class densemlp_tuned with results (one row per candidate, ranked best first by mean validation loss), best_config, and, when refit = TRUE, best_fit.

Examples

```
set.seed(1)
x <- data.frame(a = rnorm(80), b = rnorm(80))
y <- x$a - 0.5 * x$b + rnorm(80, sd = 0.1)
tuned <- tune_densemlp(
  x, y, repeats = 1,
  grid = list(hidden_units = list(c(8), c(16, 8)), epochs = c(20))
)
tuned$best_config
```

Index

cv_densemlp, 2

densemlp, 3
densemlp(), *2, 3*, *8*, *13, 14*
densemlp_integrated_brier_score, 8
densemlp_metrics, 8
densemlp_metrics(), *2, 3*, *7*, *10*

graphics::barplot(), *11*
graphics::matplot(), *11, 12*
group_effects, 9
group_effects(), *7*

perm_importance, 10
plot.densemlp, 11
plot.densemlp(), *12*
plot.densemlp_importance, 11
plot_history, 12
predict.densemlp, 12
print.densemlp *(*densemlp*)*, 3
print.densemlp_importance
*(*perm_importance*)*, 10

survival::Surv(), *4*

tune_densemlp, 13