Package 'survalis'

May 9, 2026

Type Package

Title Interpretable Survival Machine Learning Framework

Version 0.7.1

Author Imad El Badisy [aut, cre]

Maintainer Imad El Badisy <elbadisyimad@gmail.com>

Description A modular toolkit for interpretable survival machine learning with a unified interface for fitting, prediction, evaluation, and interpretation.
It includes semiparametric, parametric, tree-based, ensemble, boosting, kernel, and deep-learning survival learners, together with benchmarking, scoring, calibration, and model-agnostic interpretation utilities.
Representative methodological anchors include Cox (1972) <doi:10.1111/j.2517-6161.1972.tb00899.x>, Royston and Parmar (2002) <doi:10.1002/sim.1203>, Ishwaran et al. (2008) <doi:10.1214/08-AOAS169>, Jaeger et al. (2019) <doi:10.1214/19-AOAS1261>, Harrell et al. (1982) <doi:10.1001/jama.1982.03320430047030>, Graf et al. (1999) <doi:10.1002/(SICI)1097-0258(19990915/30)18:17/18%3C2529::AID-SIM274%3E3.0.CO;2-5>,
Friedman (2001) <doi:10.1214/aos/1013203451>, Apley and Zhu (2020) <doi:10.1111/rssb.12377>, and Lundberg and Lee (2017) <https://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions>, and other related methods for survival modeling, prediction, and interpretation.

License MIT + file LICENSE

URL https://github.com/ielbadisy/survalis

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

Encoding UTF-8

LazyData true

Depends R (>= 4.1)

Imports survival, ggplot2, functionals, nnls, rpart, tibble, rsample, aftgee, aorsf, bnnSurvival, pec, party, ranger, survdnn, survivalsvm, randomForestSRC, xgboost, BART, flexsurv, glmnet, mboost, rstpm2, timereg, partykit, gower, pracma, torch, data.table, dplyr, glue, cli, purrr, rlang, tidyr

Suggests testthat (>= 3.0.0), knitr, rmarkdown, roxygen2, covr, stats, utils

RoxygenNote 7.3.3

Config/testthat/edition 3

NeedsCompilation no

Repository CRAN

Date/Publication 2026-04-23 20:20:02 UTC

Contents

`auc_survmat` *Time-Dependent AUC from a Survival-Probability Matrix*

Description

Computes a cumulative/dynamic time-dependent AUC using predicted survival probabilities at a specified time point (or the last column if `t_star` is `NULL`). Cases are subjects with an observed event by `t_star`; controls are subjects known to survive beyond `t_star`. Subjects censored before `t_star` are handled through IPCW weighting.

Usage

```
auc_survmat(object, predicted, t_star = NULL)
```

Arguments

`object`	A `Surv` object of length n.
`predicted`	An `n x k` matrix or data frame of survival probabilities with columns named `"t=<time>"`.
`t_star`	Optional numeric time at which to evaluate AUC; if omitted, the rightmost column of `predicted` is used.

Details

Risk scores are defined as `1 - S(t)` at the chosen time. The AUC is computed over case-control pairs using inverse-probability-of-censoring weights for cases and partial credit (0.5) for ties.

Value

A named numeric scalar: `"auc"`.

Examples

```
y <- survival::Surv(time = veteran$time, event = veteran$status)
sp <- matrix(
  stats::plogis(scale(veteran$karno)),
  ncol = 1,
  dimnames = list(NULL, "t=100")
)
auc_survmat(y, predicted = sp, t_star = 100)
```

---

benchmark_default_survlearners

Benchmark Multiple Survival Learners (Cross-Validation Wrapper)

Description

Runs `cv_survlearner()` for a set of learner names (e.g., `"ranger"`, `"coxph"`) by dynamically dispatching `fit_<learner>` and `predict_<learner>` functions. Returns the row-bound CV results across all requested learners.

Usage

```
benchmark_default_survlearners(
  formula,
  data,
  learners,
  times,
  metrics = c("cindex", "ibs"),
  folds = 5,
  seed = 123,
  ncores = 1,
  verbose = FALSE,
  suppress_errors = TRUE,
  ...
)
```

Arguments

formula	A survival formula of the form `Surv(time, status) ~ x1 + x2 + ...`.
data	A data frame containing the variables in `formula`.
learners	Character vector of learner ids (without prefixes), e.g. `c("ranger","coxph","glmnet")`. For each id, `fit_<id>` and `predict_<id>` must exist.
times	Numeric vector of evaluation time points for survival predictions.
metrics	Character vector of metrics to compute in CV (e.g., `"cindex"`, `"ibs"`, `"iae"`, `"ise"`). The metric semantics are those of `cv_survlearner()`.
folds	Integer number of CV folds (default `5`).
seed	Integer random seed for fold generation.
ncores	Integer; number of CPU cores passed to `cv_survlearner()` for fold-level parallel mapping (default `1`).
verbose	Logical; if `TRUE`, prints progress per learner.
suppress_errors	Logical; if `TRUE` (default) errors from individual learners are caught and reported via `warning()` and benchmarking continues. If `FALSE`, the first error is thrown.
...	Additional arguments forwarded to each learner's `fit_*()`.

Details

Learners are run independently using identical CV splits and scoring settings. Any learner whose `fit_*()` or `predict_*()` function is missing will be skipped with a warning. At least one learner must complete successfully or an error is raised.

Value

A data frame of CV results (as returned by `cv_survlearner()`) with an extra column `learner` identifying the source learner.

See Also

`cv_survlearner()`, `plot_benchmark()`, `summarise_benchmark()`

Examples

```
res <- benchmark_default_survlearners(
  Surv(time, status) ~ age + karno + trt,
  data = veteran,
  learners = c("coxph", "rpart"),
  times = c(80, 160),
  metrics = c("cindex", "ibs"),
  folds = 2,
  seed = 1
)
head(res)
```

benchmark_tuned_survlearners *Benchmark Tuned Survival Learners with Nested Cross-Validation*

Description

Runs nested cross-validation for one or more learners that expose `tune_<learner>()`, `fit_<learner>()`, and `predict_<learner>()`. The outer folds estimate performance, while each inner training split is tuned using the learner's existing `tune_*()` implementation only on the outer training data.

Usage

```
benchmark_tuned_survlearners(
  formula,
  data,
  learners,
  times,
  metrics = c("cindex", "ibs"),
  outer_folds = 5,
  inner_folds = 5,
  seed = 123,
  inner_ncores = 1,
  learner_args = list(),
  refit_final = FALSE,
  verbose = FALSE,
  suppress_errors = TRUE,
  ...
)
```

Arguments

`formula`	A survival formula of the form `Surv(time, status) ~ x1 + x2 + ...`.
`data`	A data frame containing the variables in `formula`.
`learners`	Character vector of learner ids (without prefixes), e.g. `c("ranger", "glmnet")`.
`times`	Numeric vector of evaluation time points for survival predictions.
`metrics`	Character vector of metrics to optimize and report.
`outer_folds`	Integer number of outer CV folds used for performance estimation.
`inner_folds`	Integer number of inner CV folds used by each learner's tuning routine.
`seed`	Integer random seed used for outer and inner resampling.
`inner_ncores`	Integer; number of CPU cores passed to each learner's inner `cv_survlearner()` calls during tuning (default 1).
`learner_args`	Optional named list of learner-specific arguments. Each entry can be either a list of tuning arguments passed to `tune_<learner>()`, or a list with `tune` and/or `fit` components. The `fit` component is forwarded only when refitting the selected hyperparameters on an outer training split or on the full dataset.

refit_final	Logical; if TRUE, tunes each learner on the full dataset and refits a final model with the selected hyperparameters.
verbose	Logical; if TRUE, prints progress per learner and outer fold.
suppress_errors	Logical; if TRUE (default), errors from individual learners or outer folds are caught and reported via warning().
...	Additional arguments passed to each learner's tune_*() function.

Value

A list of class "nested_surv_benchmark" with components outer_results, outer_summary, selected_params, final_models, and settings.

See Also

benchmark_default_survlearners, cv_survlearner

Examples

```
res <- benchmark_tuned_survlearners(
  Surv(time, status) ~ age + karno + trt,
  data = veteran,
  learners = c("ranger", "glmnet"),
  times = c(100, 200),
  outer_folds = 3,
  inner_folds = 2
)
res$outer_summary
```

---

best_survlearner *Select the Best Survival Learner by a Given Metric*

Description

Extracts the top-performing learner(s) under a chosen metric from benchmark results, using the average value across folds.

Usage

```
best_survlearner(benchmark_results, metric, maximize = NULL)
```

Arguments

benchmark_results	A data frame with columns learner, metric, and value.
metric	Character name of the metric to optimize (e.g., "cindex", "ibs"). Must exist in benchmark_results$metric.
maximize	Logical; whether to maximize the metric. If NULL (default), the function maximizes for concordance-like metrics ("cindex") and minimizes for error-like metrics ("ibs", "brier", "iae", "ise", "ece").

Value

A tibble with columns learner, metric, and the selected average value for the best learner(s). Ties are returned as multiple rows.

See Also

benchmark_default_survlearners(), summarise_benchmark()

Examples

```
res <- tibble::tibble(
  learner = c("coxph", "coxph", "rpart", "rpart"),
  metric = c("cindex", "ibs", "cindex", "ibs"),
  value = c(0.64, 0.19, 0.60, 0.23)
)
best_survlearner(res, metric = "cindex")
best_survlearner(res, metric = "ibs")
```

---

brier *Brier Score with IPCW for a Single Time Point*

Description

Computes the inverse-probability-of-censoring weighted (IPCW) Brier score at a single time t_star.

Usage

```
brier(object, pre_sp, t_star)
```

Arguments

object	A Surv object.
pre_sp	Numeric vector of predicted survival probabilities $S(t^* \mid x_i)$.
t_star	Numeric evaluation time.

Details

The censoring distribution $G(t)$ is estimated via Kaplan-Meier on `1 - status`. Observed events before `t_star` contribute $S(t_i)^2/G(t_i)$; those at risk at `t_star` contribute $(1 - S(t^*))^2/G(t^*)$. Returns `NA` if $G(t^*)$ is undefined or zero.

Value

A named numeric scalar: `"brier"`.

Examples

```
y <- survival::Surv(time = veteran$time, event = veteran$status)
pre_sp <- stats::plogis(scale(veteran$karno))
brier(y, pre_sp = pre_sp, t_star = 100)
```

`cindex_survmat` *Concordance Index from a Survival-Probability Matrix*

Description

Computes Harrell's concordance index using predicted survival probabilities at a specified time point (or the last column if `t_star` is `NULL`).

Usage

```
cindex_survmat(object, predicted, t_star = NULL)
```

Arguments

`object`	A `Surv` object of length n.
`predicted`	An `n x k` matrix or data frame of survival probabilities with columns named `"t=<time>"`.
`t_star`	Optional numeric time at which to evaluate the c-index; if omitted, the rightmost column of `predicted` is used.

Details

Risk scores are defined as `1 - S(t)` at the chosen time. Ties receive partial credit (0.5). Pairs not comparable due to censoring are excluded.

Value

A named numeric scalar: `"C index"`.

Examples

```
y <- survival::Surv(time = veteran$time, event = veteran$status)
sp <- matrix(
  stats::plogis(scale(veteran$karno)),
  ncol = 1,
  dimnames = list(NULL, "t=100")
)
cindex_survmat(y, predicted = sp, t_star = 100)
```

`compute_ale` *Accumulated Local Effects (ALE) for Survival Models*

Description

Computes ALE curves for a numeric (continuous) feature with respect to survival probabilities at one or more evaluation times. ALE summarizes the average *local* effect of changing a feature within small intervals, is robust to correlated features, and is centered to have mean zero.

Usage

```
compute_ale(model, newdata, feature, times, grid.size = 20)
```

Arguments

`model`	An `mlsurv_model` created by a `fit_*()` function. Must include a valid `learner` so that `predict_<learner>()` can be dispatched.
`newdata`	Data frame used to compute ALE (typically the training set or a representative sample).
`feature`	Single **numeric/continuous** feature name for which to compute ALE. Categorical features are not supported here (use PDP/ICE).
`times`	Numeric vector of time points at which to evaluate survival probabilities.
`grid.size`	Integer number of quantile cut points used to build the ALE grid (default 20). The algorithm uses quantiles of `newdata[[feature]]`.

Details

For consecutive quantile bins $[z_k, z_{k+1}]$ of the target feature, ALE integrates the *local* change in the model prediction when moving the feature from z_k to z_{k+1} while holding all other features at their observed values, and then accumulates these differences across bins. For survival models, predictions are survival probabilities at `times`. The returned ALE curves are centered (mean zero across the grid) per time.

Value

A list with:

ale A data frame with columns `feature_value` and one column per time (`"t=<time>"`) containing centered ALE effects.

integrated If multiple times were provided, a data frame with columns `feature_value` and `integrated_ale` equal to the mean of per-time ALE effects across `times`; otherwise `NULL`.

See Also

`plot_ale()`, `compute_pdp()`

Examples

```
mod <- fit_coxph(Surv(time, status) ~ age + karno + celltype, data = veteran)
ale_res <- compute_ale(
  model = mod,
  newdata = veteran,
  feature = "karno",
  times = c(80, 160),
  grid.size = 8
)
head(ale_res$ale)
```

---

compute_calibration *Calibration of Survival Predictions at a Single Time Point*

Description

Computes a nonparametric calibration curve for a survival model at one evaluation time by binning predicted survival probabilities and comparing bin-wise means to Kaplan-Meier-based observed survival, with bootstrap CIs.

Usage

```
compute_calibration(
  model,
  data,
  time,
  status,
  eval_time,
  n_bins = 10,
  n_boot = 100,
  seed = 123,
  learner_name = NULL
)
```

Arguments

model	An mlsurv_model fitted via a fit_*() function; must include a valid learner so the corresponding predict_<learner>() can be dispatched, and (ideally) a $data field for level alignment.
data	A data frame with predictors and survival outcome columns.
time	Survival time; either a numeric vector of the same length as nrow(data) or a single string giving the column name in data.
status	Event indicator; either a numeric/logical vector or a single string giving the column name in data. Events should be coded 1, censoring 0.
eval_time	Single numeric time at which to assess calibration.
n_bins	Integer number of quantile-based bins used to group predictions.
n_boot	Integer number of bootstrap resamples for confidence intervals.
seed	Integer seed for reproducibility of binning/bootstrap.
learner_name	Optional character override for labeling the learner in downstream plots (defaults to model$learner).

Details

Predicted survival at eval_time is obtained from the appropriate predict_<learner>(). Predictions are split into n_bins quantile bins. For each bin, the function reports: mean predicted survival, observed survival at eval_time from a Kaplan-Meier fit on the bin's rows, and bootstrap percentile (2.5%, 97.5%) CIs on the observed survival computed by resampling rows with replacement.

Value

A list with components:

calibration_table A data frame with columns bin, mean_pred_surv, observed_surv, lower_ci, upper_ci.

eval_time The evaluation time used.

n_bins Number of bins.

n_boot Number of bootstrap resamples.

learner The learner label (from learner_name or model$learner).

See Also

plot_calibration()

Examples

```
mod <- fit_coxph(Surv(time, status) ~ age + karno + celltype, data = veteran)
calib <- compute_calibration(
  model = mod,
  data = veteran,
  time = "time",
  status = "status",
```

```
  eval_time = 80,
  n_bins = 4,
  n_boot = 5,
  seed = 1
)
head(calib$calibration_table)
```

`compute_counterfactual` *Compute individual counterfactual changes to increase survival*

Description

For a single individual (one-row `newdata`), propose feature changes that maximize survival probability at a target time, subject to optional per-feature change costs and bounds inferred from the training data in `model$data`.

Usage

```
compute_counterfactual(
  model,
  newdata,
  times,
  target_time,
  features_to_change = NULL,
  grid.size = 100,
  max.change = NULL,
  cost_penalty = 0.01
)
```

Arguments

`model`	A fitted survival model (e.g., an `mlsurv_model`) that was trained with a data frame stored in `model$data`, and (ideally) names of the outcome variables in `model$time` and `model$status`.
`newdata`	A data frame with exactly one row representing the individual.
`times`	Numeric vector of time points used for prediction. Required unless the model's predict function can infer times; used together with `target_time`.
`target_time`	Numeric scalar time at which to optimize survival. If missing and `times` is provided, the median of `times` is used. If `times` is missing but `target_time` is provided, `times` is set to `target_time`.
`features_to_change`	Optional character vector of feature names allowed to change. Defaults to all predictors (non-outcome columns).
`grid.size`	Integer grid size for numeric features (default 100).

max.change	Optional named list of numeric bounds for per-feature absolute change, e.g., list(age = 5).
cost_penalty	Numeric penalty weight applied to magnitude of change (default 0.01).

Details

For each candidate feature, the function sweeps over plausible values (numeric grid between observed min/max; all other levels for categorical), predicts survival at target_time, and reports the best penalized gain relative to the original value. Survival predictions are obtained via a corresponding predict_* function inferred from model$learner (e.g., predict_coxph for learner = "coxph").

Value

A data.frame with one row per feature considered and columns: feature, original_value, suggested_value, survival_gain, change_cost, penalized_gain.

Examples

```
df <- veteran
df$A <- df$trt
mod <- fit_coxph(survival::Surv(time, status) ~ A + age + karno, data = df)

cf <- compute_counterfactual(
  model = mod,
  newdata = df[1, , drop = FALSE],
  times = c(50, 100, 150),
  target_time = 100,
  features_to_change = c("A", "age", "karno"),
  grid.size = 10,
  cost_penalty = 0.02
)
head(cf)
```

---

compute_interactions *Compute Feature Interactions for Survival Predictions*

Description

Estimates global and time-varying interaction strengths of model predictions, using a Friedman-H style decomposition adapted to survival partial dependence. Works with any mlsurv_model that has a matching predict_*() method returning survival probabilities.

Usage

```
compute_interactions(
  model,
  data,
  times,
  target_time = NULL,
  features = NULL,
  type = c("1way", "heatmap", "time"),
  grid.size = 30,
  batch.size = 100
)
```

Arguments

model	An mlsurv_model created via a fit_*() function; must include a valid learner so the appropriate predict_<learner>() can be dispatched.
data	Data frame used to probe the model (typically training data).
times	Numeric vector of evaluation times used for prediction.
target_time	Single time at which to quantify interactions for type = "1way" or type = "heatmap". Required for these types.
features	Optional character vector of feature names to evaluate. Defaults to all predictors in data except the outcome columns.
type	One of: • "1way" – per-feature Friedman-H interaction strength vs. all others at target_time; • "heatmap" – pairwise feature interaction matrix at target_time; • "time" – per-feature interaction strength across times.
grid.size	Integer; number of random grid values / replicates used for Monte Carlo marginalization (default 30).
batch.size	Reserved for future batching support (currently unused).

Details

For a target time t^*, let $f(x)$ be the predicted survival probability. For feature j, we approximate a decomposition:

$$f(x) \approx f_j(x_j) + f_{-j}(x_{-j})$$

by Monte Carlo marginalization over subsets of features using random sampling from the empirical distribution in data. The reported interaction strength is:

$$H_j(t^*) = \sqrt{\frac{\sum(f(x) - \{\tilde{f}_j(x_j) + \tilde{f}_{-j}(x_{-j})\})^2}{\sum f(x)^2}},$$

clipped to [0, 1]. Pairwise heatmaps are computed analogously for (j, k).

Larger values indicate stronger non-additivity (interaction) involving the feature(s). The "time" mode repeats the computation across all times to show dynamics.

Value

A data frame whose structure depends on `type`:

- `"1way"`: columns `feature`, `interaction`.
- `"heatmap"`: columns `feature1`, `feature2`, `interaction` (symmetric with zeros on the diagonal).
- `"time"`: columns `feature`, `time`, `interaction`.

References

Friedman, J. H., and Popescu, B. E. (2008). Predictive learning via rule ensembles. *Annals of Applied Statistics*. (Friedman's H interaction measure.)

Examples

```
mod <- fit_coxph(Surv(time, status) ~ age + karno + trt, data = veteran)
times <- c(80, 160)
compute_interactions(
  model = mod,
  data = veteran,
  times = times,
  target_time = 80,
  features = c("age", "karno"),
  type = "1way",
  grid.size = 6
)
```

`compute_pdp` *Partial Dependence and ICE for Survival Predictions*

Description

Computes partial dependence (PDP) and/or individual conditional expectation (ICE) curves of predicted survival probabilities for a single feature at one or more evaluation times. Works with any learner fitted via `fit_*()` that exposes a matching `predict_*()` method returning survival probabilities.

Usage

```
compute_pdp(model, data, feature, times, method = "pdp+ice", grid.size = 20)
```

Arguments

model	An mlsurv_model created by a fit_*() function; must contain a valid learner so the appropriate predict_<learner>() can be dispatched.
data	A data frame used to construct PDP/ICE profiles (typically the training data).
feature	Character scalar; the feature name to analyze (numeric or categorical).
times	Numeric vector of evaluation times at which survival probabilities are computed.
method	One of "pdp", "ice", or "pdp+ice" (default). Controls which profiles are produced.
grid.size	Integer number of grid points for numeric features (default 20). Ignored for categorical features (levels are used).

Details

For numeric features, a regular grid over the observed range is used; for categorical features, all observed levels are used. For each grid value, predictions are made for every row of data with the feature forced to the grid value, yielding ICE curves per row and PDP as the average across rows.

If multiple times are supplied, outputs are stacked in long format with a time column; an additional integrated PDP is computed via trapezoidal rule when more than one time is provided.

Value

A list with elements:

results Long data frame with columns: surv_prob, time, type (pdp/ice), .id (row id for ICE), and the analyzed feature.

pdp_integrated (Optional) Data frame with feature and integrated_surv (time-integrated PDP), present only if length(times) > 1 and PDP was requested.

Examples

```
mod <- fit_coxph(Surv(time, status) ~ age + karno + trt, data = veteran)
pdp_age <- compute_pdp(
  model = mod,
  data = veteran,
  feature = "age",
  times = c(80, 160),
  method = "pdp+ice",
  grid.size = 8
)
head(pdp_age$results)
```

compute_shap *Compute local SHAP-like contributions for survival predictions*

Description

Estimates per-feature contributions (à la SHAP) to the predicted survival probability for a **single** observation at one or more time points. For each time, the method samples random feature orders, marginalizes future features with values from a baseline dataset, and accumulates the marginal effects of adding each feature back.

Usage

```
compute_shap(
  model,
  newdata,
  baseline_data,
  times,
  sample.size = 100,
  aggregate = FALSE,
  method = c("meanabs", "integral")
)
```

Arguments

model	A fitted survival model produced by fit_*() that exposes model$learner (e.g., "coxph") and model$data (the training frame).
newdata	A data frame with **exactly one row** (the instance to explain).
baseline_data	A data frame to sample background values from (typically the training data used to fit model).
times	Numeric vector of evaluation times (same scale as the outcome).
sample.size	Integer, number of random feature orderings to sample per time (default 100).
aggregate	Logical; if TRUE, aggregate contributions across times into a single value per feature (see method).
method	Character; aggregation method if aggregate=TRUE: "meanabs" (mean absolute contribution) or "integral" (trapezoidal integral over time). Default "meanabs".

Details

The prediction function is inferred from model$learner as predict_<learner> and called with signature predict_fun(model, newdata, times = times). Factor levels in newdata are harmonized to those in model$data.

Value

If aggregate = FALSE: a data frame with columns feature, phi, and time (one row per feature per time). If aggregate = TRUE: a data frame with columns feature and phi (one row per feature), with attribute "shap_method" set to the aggregation used.

See Also

plot_shap(), the various predict_* methods (e.g. predict_coxph())

Examples

```
mod <- fit_coxph(survival::Surv(time, status) ~ age + karno + celltype, data = veteran)

shap_td <- compute_shap(
  model         = mod,
  newdata       = veteran[100, , drop = FALSE],
  baseline_data = veteran,
  times         = c(100, 200),
  sample.size   = 5,
  aggregate     = FALSE
)
head(shap_td)

shap_meanabs <- compute_shap(
  model         = mod,
  newdata       = veteran[100, , drop = FALSE],
  baseline_data = veteran,
  times         = c(100, 200),
  sample.size   = 5,
  aggregate     = TRUE,
  method        = "meanabs"
)
head(shap_meanabs)
```

---

compute_surrogate *Local Surrogate Explanation for Survival Predictions (LIME-style)*

Description

Builds a sparse, locally weighted linear surrogate model to explain a fitted survival model's prediction at a specific target time for a single instance. Categorical features are binarized relative to the instance of interest; local weights are computed from feature-space proximity (Gower by default); and a penalized (lasso) or unpenalized linear model is fit to approximate the model's predicted survival probability at target_time.

Usage

```
compute_surrogate(
  model,
  newdata,
  baseline_data,
  times,
  target_time,
```

```
  k = 5,
  dist.fun = "gower",
  gower.power = 5,
  kernel.width = NULL,
  penalized = TRUE,
  exclude = NULL
)
```

Arguments

model	An mlsurv_model fitted via fit_*(), containing a valid $learner so that predict_<learner> can be dispatched.
newdata	A one-row data frame: the instance to explain (must have the same predictors as baseline_data).
baseline_data	A data frame used to define the local neighborhood and to fit the surrogate (typically the model's training data).
times	Numeric vector of times passed to the prediction function; must include target_time (nearest value is used).
target_time	Numeric time at which to explain the prediction.
k	Desired number of non-zero coefficients for the penalized surrogate (used only when penalized = TRUE); acts as a target sparsity.
dist.fun	Distance function for locality weighting. Default "gower" (via **gower**); otherwise any method accepted by dist (requires kernel.width).
gower.power	Power applied to $(1-\text{gower\_dist})$ when dist.fun = "gower" to control locality sharpness (default 5).
kernel.width	Positive numeric bandwidth used for non-Gower kernels (Gaussian weighting on pairwise distances).
penalized	Logical; if TRUE (default) fits a lasso surrogate via **glmnet**; otherwise fits a weighted least squares lm().
exclude	Optional character vector of column names to exclude from the surrogate (in addition to survival outcome columns).

Details

Target: The surrogate approximates $S(t^* \mid x)$ returned by the underlying model at target_time. The nearest column in times is used.

Weights: Locality weights are computed from distances between baseline_data rows and newdata. With "gower", weights are $(1-\text{gower\_dist})^{\text{gower.power}}$; otherwise Gaussian $\exp\{-d^2/\text{kernel.width}^2\}^{1/2}$.

Sparsity: When penalized=TRUE, a **glmnet** path is fit and the solution with degrees of freedom closest to k is selected (preferring exactly k if available).

Value

A data frame with one row per selected feature containing:

feature Feature name (after recoding).

feature_value Value of the feature in `newdata`.

effect Local contribution $\beta_j \cdot x_j$ at `target_time`.

target_time The explanation time (copied for plotting).

Rows are ordered by decreasing $|effect|$.

Examples

```
mod <- fit_coxph(Surv(time, status) ~ age + karno + celltype, data = veteran)
local_expl <- compute_surrogate(
  model = mod,
  newdata = veteran[2, , drop = FALSE],
  baseline_data = veteran,
  times = c(80, 160),
  target_time = 80,
  k = 3
)
head(local_expl)
```

`compute_tree_surrogate` *Compute Tree-Based Surrogate Model for Survival Predictions*

Description

Fits decision tree surrogate models to approximate the predictions of a fitted survival model at one or more evaluation times. This allows users to gain interpretable, rule-based approximations of complex survival models.

Usage

```
compute_tree_surrogate(model, data, times, minsplit = 10, cp = 0.01)
```

Arguments

`model`	A fitted survival model object created with a `fit_*()` function. Must include a valid `learner` field corresponding to a `predict_*()` method.
`data`	A data frame containing predictor variables (and optional survival outcome columns).
`times`	A numeric vector of evaluation times at which to approximate model predictions. Must contain at least one value.
`minsplit`	Minimum number of observations required to attempt a split in the surrogate tree. Passed to `rpart::rpart.control()`.
`cp`	Complexity parameter for the surrogate tree. Passed to `rpart::rpart.control()`.

Details

For each evaluation time, the function:

1. Predicts survival probabilities from the fitted model.
2. Excludes survival outcome columns (`time`, `status`, `event`) from the predictors.
3. Fits a decision tree to approximate the predicted probabilities.
4. Computes the R between the model predictions and the surrogate predictions.
5. Counts the number of splits per feature.

If multiple `times` are provided, results are stored for each time point.

Value

An object of class `"tree_surrogate"`, containing:

- `times`: the evaluation times.
- `results`: a list with one element per time, each containing:
  - `tree`: the fitted `rpart` object.
  - `r_squared`: the R of the surrogate model vs. the original predictions.
  - `split_count`: a table of feature split counts.
- `dynamic`: logical indicating if more than one time was used.

Examples

```
mod_ranger <- fit_ranger(Surv(time, status) ~ age + karno + celltype, data = veteran)
tree_ranger <- compute_tree_surrogate(
  model = mod_ranger,
  data = veteran,
  times = c(100, 200, 300)
)
```

---

compute_varimp *Permutation variable importance for survival models*

Description

Estimates feature importance by measuring the change in a survival metric after permuting each feature.

Usage

```
compute_varimp(
  model,
  times,
  metric = "ibs",
  n_repetitions = 10,
  seed = NULL,
  subset = NULL,
  importance_type = c("delta", "mean")
)
```

Arguments

model	An mlsurv_model-like object with fields data, formula, and learner.
times	Numeric vector of evaluation times.
metric	Character string, e.g. "ibs" or "cindex".
n_repetitions	Integer; number of permutations per feature.
seed	Optional integer seed for reproducibility.
subset	Optional row indices or logical vector to subset model$data.
importance_type	One of "delta" (default) or "mean"; see Details.

Details

For each feature, rows are permuted n_repetitions times, predictions are recomputed, and the chosen metric is compared to the baseline (unpermuted) value. The scaled_importance rescales values to sum to 100%.

Value

A data.frame with columns:

- feature: feature name,
- value: importance value (change in metric),
- scaled_importance: percent-scaled importance (see Details).

Examples

```
mod <- fit_coxph(survival::Surv(time, status) ~ age + karno + celltype, data = veteran)
imp <- compute_varimp(
  model = mod,
  times = 80,
  metric = "brier",
  n_repetitions = 3,
  seed = 1,
  subset = 40
)
head(imp)
```

cv_plot *Boxplot of Cross-Validation Metric Distributions*

Description

Visualizes per-fold metric values from `cv_survlearner` using a boxplot with jittered points.

Usage

```
cv_plot(cv_results)
```

Arguments

`cv_results`	A tibble/data frame as returned by `cv_survlearner()`.

Value

A **ggplot2** object.

Examples

```
cv_results <- tibble::tibble(
  metric = c("cindex", "cindex", "ibs", "ibs"),
  value = c(0.62, 0.66, 0.19, 0.21)
)
cv_plot(cv_results)
```

cv_summary *Summarize Cross-Validation Results*

Description

Produces mean, standard deviation, standard error, and 95\ for each metric returned by `cv_survlearner`.

Usage

```
cv_summary(cv_results)
```

Arguments

`cv_results`	A tibble/data frame as returned by `cv_survlearner()`.

Value

A tibble with columns: `metric`, `mean`, `sd`, `n`, `se`, `lower`, `upper`.

Examples

```
cv_results <- tibble::tibble(
  metric = c("cindex", "cindex", "ibs", "ibs"),
  value = c(0.62, 0.66, 0.19, 0.21)
)
cv_summary(cv_results)
```

---

cv_survlearner *Cross-Validate a Survival Learner (fold-mapped with* fmapn*)*

Description

Runs k-fold cross-validation for any pair of fit_fun/pred_fun that follow the package's learner contracts, and returns tidy per-fold metric values. Fold iteration is handled by functionals::fmapn() with optional parallel execution and a progress bar.

Usage

```
cv_survlearner(
  formula,
  data,
  fit_fun,
  pred_fun,
  times,
  metrics = c("cindex", "ibs"),
  folds = 5,
  seed = 123,
  verbose = FALSE,
  ncores = 1,
  pb = interactive(),
  ...
)
```

Arguments

formula	A survival formula Surv(time, status) ~ predictors.
data	A data frame containing all variables in formula.
fit_fun	Function with signature fit_fun(formula, data, ...) that returns an mlsurv_model.
pred_fun	Function with signature pred_fun(object, newdata, times, ...) returning a survival-probability matrix/data frame (columns named "t=<time>").
times	Numeric vector of evaluation times (passed to pred_fun).
metrics	Character vector of metrics to compute. Supported: "cindex", "brier" (single time), "ibs", "iae", "ise", "ece" (single time).
folds	Integer; number of folds (default 5).
seed	Integer random seed for reproducibility (default 123).

verbose	Logical; print row-dropping due to missingness (default FALSE).
ncores	Integer; number of CPU cores for fmapn parallel mapping (default 1). Set > 1 to enable parallelism.
pb	Logical; show a progress bar during fold mapping (default interactive()).
...	Additional arguments forwarded to fit_fun.

Details

The routine:

1. Validates Surv(...) on the LHS and warns against using . in formulas.
2. Drops rows with missing values in any variables referenced by formula.
3. Supports Surv(time, status == k) by recoding the status to 0/1.
4. Builds stratified v-folds on the status indicator (**rsample**).
5. For each fold: fits on the analysis set, predicts on the assessment set, and computes metrics.

Fold iteration is performed via functionals::fmapn(), which preserves per-fold identifiers (id, fold) and returns a list ready for dplyr::bind_rows().

Value

A tibble with columns: splits (rsample split object), id, fold, metric, and value.

See Also

fmapn

Examples

```
cv_results <- cv_survlearner(
  formula = Surv(time, status) ~ age + karno,
  data = veteran,
  fit_fun = fit_coxph,
  pred_fun = predict_coxph,
  times = c(80, 160),
  metrics = c("cindex", "ibs"),
  folds = 2,
  seed = 1
)
```

`cv_survmetalearner` *Cross-Validate a Stacked Survival Meta-Learner*

Description

Performs v-fold cross-validation for the NNLS stacking meta-learner over a fixed set of base learners and their predictions. On each fold, stacking weights are learned on the analysis set and evaluated on the assessment set.

Usage

```
cv_survmetalearner(
  formula,
  data,
  times,
  base_models,
  base_preds,
  folds = 5,
  metrics = c("cindex", "ibs"),
  seed = 123,
  verbose = TRUE
)
```

Arguments

`formula`	A survival formula `Surv(time, status) ~ predictors`.
`data`	A data frame containing all variables referenced in `formula`.
`times`	Numeric vector of evaluation times for stacking and scoring.
`base_models`	A named list of fitted base learner models (used to predict on assessment folds and for the final refit on full data).
`base_preds`	A named list of *training-set* prediction matrices (rows align with `data`, columns align with `times`); names must match `base_models`.
`folds`	Integer; number of CV folds (default `5`).
`metrics`	Character vector of metrics to compute (default `c("cindex","ibs")`). Supported: `"cindex"`, `"brier"` (single time), `"ibs"`, `"iae"`, `"ise"`, `"ece"` (single time).
`seed`	Integer random seed (default `123`).
`verbose`	Logical; if `TRUE`, prints fold progress (default `TRUE`).

Details

For each fold: (1) subset `base_preds` to the analysis indices; (2) learn time-specific NNLS weights with `fit_survmetalearner`;

(3) predict stacked survival on the assessment set with `predict_survmetalearner`; (4) compute requested metrics. After CV, a final meta-learner is fit on the full data.

Value

An object of class "cv_survmetalearner_result" with components:

model Final "survmetalearner" fit on all data.

cv_results Per-fold metric values (tibble).

summary Fold-aggregated mean and sd by metric (tibble).

folds, metrics CV settings.

See Also

fit_survmetalearner, predict_survmetalearner, plot_survmetalearner_weights

Examples

```
form <- Surv(time, status) ~ age + karno + trt
times <- c(80, 160)
mod_cox <- fit_coxph(form, data = veteran)
mod_rpart <- fit_rpart(form, data = veteran)
base_models <- list(coxph = mod_cox, rpart = mod_rpart)
base_preds  <- list(
  coxph = predict_coxph(mod_cox, veteran, times),
  rpart = predict_rpart(mod_rpart, veteran, times)
)

cv_res <- cv_survmetalearner(
  formula = form, data = veteran, times = times,
  base_models = base_models, base_preds = base_preds,
  folds = 2, metrics = c("cindex", "ibs"), seed = 1, verbose = FALSE
)

cv_res$summary
plot_survmetalearner_weights(cv_res$model)
```

---

ece_survmat *Expected Calibration Error (ECE) at a Single Time Point*

Description

Computes a binned Expected Calibration Error at time t_star, using quantile-based bins of predicted survival probabilities.

Usage

```
ece_survmat(object, sp_matrix, t_star, n_bins = 10L, p = 1, weighted = TRUE)
```

Arguments

object	A Surv object.
sp_matrix	Matrix or data frame of survival probabilities with rows = subjects and (optionally) columns named "t=<time>".
t_star	Numeric evaluation time (must be a single value).
n_bins	Integer number of quantile bins (default 10).
p	Power for the L^p error (default 1 gives absolute error).
weighted	Logical; if TRUE (default) weights bin errors by n_k / N where n_k is the bin size.

Details

Let $\hat{S}_k(t^*)$ be the mean predicted survival in bin k and $\tilde{S}_k(t^*)$ the Kaplan-Meier survival at t^* in that bin. The (weighted) ECE is

$$\mathrm{ECE}(t^*) = \left(\sum_k w_k \, |\hat{S}_k(t^*) - \tilde{S}_k(t^*)|^p \right)^{1/p}$$

with $w_k = n_k / N$.

Value

A named numeric scalar: "ece".

Examples

```
y <- survival::Surv(
  time = c(1, 2, 3, 4, 6, 7, 8, 9),
  event = c(1, 1, 0, 1, 0, 1, 1, 0)
)
sp <- cbind("t=5" = c(0.15, 0.20, 0.35, 0.40, 0.55, 0.65, 0.75, 0.80))
ece_survmat(y, sp_matrix = sp, t_star = 5, n_bins = 4)
```

---

fit_aalen *Fit an Additive Hazards (Aalen) Model*

Description

Fits an additive hazards regression model using **timereg**'s aalen and returns an mlsurv_model compatible with the **survalis** workflow.

Usage

```
fit_aalen(formula, data, max.time = NULL, n.sim = 0, resample.iid = 1)
```

Arguments

formula	A survival formula Surv(time, status) ~ predictors.
data	A data frame containing the variables in formula.
max.time	Optional maximum follow-up time used by the fitting routine.
n.sim	Integer; number of simulations for variance estimation (default 0).
resample.iid	Integer; indicator for iid resampling (passed to aalen).

Details

The Aalen model assumes an additive hazard:

$$\lambda(t \mid X) = \beta_0(t) + X^\top \beta(t),$$

with nonparametric cumulative coefficient functions.

Value

A list of class "mlsurv_model" with elements: model, learner="aalen", engine="timereg", formula, data, time, status.

Examples

```
mod_aalen <- fit_aalen(
  Surv(time, status) ~ trt + karno + age,
  data = veteran
)
head(predict_aalen(mod_aalen, newdata = veteran[1:5, ], times = c(50, 100, 150)))
```

---

fit_aftgee — *Fit an Accelerated Failure Time Model Using Generalized Estimating Equations*

Description

Fits an accelerated failure time (AFT) model via **aftgee**, which implements GEE methodology for censored survival data. Returns an mlsurv_model-compatible object used throughout the package.

Usage

```
fit_aftgee(formula, data, corstr = "independence")
```

Arguments

formula	A survival formula of the form Surv(time, status) ~ predictors.
data	A data.frame containing the variables in the model.
corstr	Working correlation structure; one of "independence", "exchangeable", or "ar1". Default: "independence".

Details

The AFT model assumes $\log(T) = X\beta + \epsilon$, where T is survival time, X is the covariate matrix, and ϵ is an error term whose distribution determines the baseline survival. **aftgee** estimates β using GEE. In this wrapper we set `id = NULL`, assuming one observation per subject.

Value

An object of class `mlsurv_model` with components:

- `model`: the fitted `aftgee` model.
- `learner`: `"aftgee"`.
- `formula`: the survival formula.
- `data`: the training data.

References

Jin, Z., Lin, D. Y., Wei, L. J., & Ying, Z. (2003). Rank-based inference for the accelerated failure time model. *Biometrika*, 90(2), 341-353.

Examples

```
mod <- fit_aftgee(
  Surv(time, status) ~ trt + karno + age,
  data = veteran
)
head(predict_aftgee(mod, newdata = veteran[1:5, ], times = c(20, 60, 120)))
```

`fit_bart` *Fit a Bayesian Additive Regression Trees (BART) Survival Model*

Description

Fits a right-censored survival model using **BART**'s `surv.bart` and returns an `mlsurv_model` compatible with the **survalis** pipeline.

Usage

```
fit_bart(formula, data, K = 3, ...)
```

Arguments

formula	A survival formula of the form `Surv(time, status) ~ predictors`. Only right-censored responses are supported.
data	A `data.frame` containing variables referenced in `formula`.
K	Integer; number of internal time grid intervals used by the BART survival engine (default 3). Larger values yield a finer internal evaluation grid.
...	Further args to BART::surv.bart

Details

The response must be of class `Surv(..., type = "right")`. The fitted object stores the engine's internal evaluation times in `$eval_times` (from `bart_fit$times`) for downstream prediction alignment.

Value

An object of class `mlsurv_model` with elements:

model Fitted `BART::surv.bart` object.

learner `"bart"`.

formula, data Original inputs.

eval_times Engine's internal time grid used for prediction.

See Also

predict_bart, surv.bart

Examples

```
ex_data <- veteran[1:40, c("time", "status", "age", "karno", "celltype")]
mod_bart <- fit_bart(
  Surv(time, status) ~ age + karno + celltype,
  data = ex_data,
  K = 1,
  ntree = 5,
  ndpost = 20,
  nskip = 5,
  mc.cores = 1,
  seed = 42
)
```

`fit_blackboost` *Fit a Componentwise Gradient Boosted Cox Model (blackboost)*

Description

Fits a survival boosting model using **mboost**'s `blackboost` with a Cox proportional hazards loss (`mboost::CoxPH()`) and shallow tree base-learners (**partykit**). Returns an `mlsurv_model` compatible with the `survalis` pipeline.

Usage

```
fit_blackboost(
  formula,
  data,
  weights = NULL,
  mstop = 100,
  nu = 0.1,
  minsplit = 10,
  minbucket = 4,
  maxdepth = 2,
  ...
)
```

Arguments

`formula`	A survival formula of the form `Surv(time, status) ~ predictors`.
`data`	A `data.frame` containing all variables referenced in `formula`.
`weights`	Optional case weights passed to `mboost::blackboost()`.
`mstop`	Integer; number of boosting iterations (stopping iteration). Default `100`.
`nu`	Learning rate/shrinkage in `(0,1]`; default `0.1`.
`minsplit`	minbucket, maxdepth Tree control parameters passed via `partykit::ctree_control()` (defaults: `10`, `4`, `2`).
`minbucket`	Minimum bucket size per tree node.
`maxdepth`	Maximum tree depth.
`...`	Additional arguments forwarded to `mboost::boost_control()` and/or `mboost::blackboost()`.

Details

The base-learner is a conditional inference tree controlled by `partykit::ctree_control()`. The loss is the partial likelihood for Cox PH via `mboost::CoxPH()`. Use `mstop` and `nu` to control complexity.

Value

An object of class `mlsurv_model` with elements:

model Fitted `mboost` model.

learner `"blackboost"`.

formula, data, time, status Original inputs/metadata.

References

Bühlmann P, Hothorn T (2007). Boosting algorithms: regularization, prediction and model fitting. *Statistical Science*.

Examples

```
mod <- fit_blackboost(
  Surv(time, status) ~ age + karno + celltype,
  data = veteran
)
head(predict_blackboost(mod, newdata = veteran[1:5, ], times = c(100, 200)))
```

`fit_bnnsurv` *Fit a kNN–Ensemble Survival Model (bnnSurvival)*

Description

Fits an ensemble of k–nearest neighbour survival learners via **bnnSurvival**. The fitted object is standardized to the `mlsurv_model` contract used in **survalis**.

Usage

```
fit_bnnsurv(
  formula,
  data,
  k = 5,
  num_base_learners = 10,
  num_features_per_base_learner = NULL,
  metric = "mahalanobis",
  weighting_function = function(x) x * 0 + 1,
  replace = TRUE,
  sample_fraction = NULL
)
```

Arguments

formula	A survival formula of the form `Surv(time, status) ~ predictors`.
data	A data frame containing all variables referenced in `formula`.
k	Integer, number of neighbours for each base learner. Default `20`.
num_base_learners	Integer, number of base learners in the ensemble. Default `10`.
num_features_per_base_learner	Integer or `NULL`. If not `NULL`, each base learner samples this many features. Passed to the engine.
metric	Character distance metric for neighbour search (for example, `"mahalanobis"`). Passed to the engine.
weighting_function	Function used to weight neighbours. Defaults to a constant weighting `function(x) x * 0 + 1`. Passed to the engine.
replace	Logical; sample with replacement when drawing observations for a base learner. Passed to the engine.
sample_fraction	Optional numeric in `(0, 1]`; fraction of rows used by each base learner. Passed to the engine.

Details

The native engine returns full survival curves on an internal time grid. See `predict_bnnsurv` for how these are post–processed and (optionally) interpolated to user–requested times.

Value

An object of class `"mlsurv_model"` with elements:

model The underlying **bnnSurvival** fit.

learner Scalar `"bnnsurv"`.

engine Scalar `"bnnSurvival"`.

formula, data Inputs preserved for downstream use.

time, status Character names of the survival outcome fields.

Engine

Uses **bnnSurvival::bnnSurvival**. This wrapper calls the engine via `requireNamespace("bnnSurvival", quietly = TRUE)` and stores the native model in `$model`.

See Also

`predict_bnnsurv()`, `tune_bnnsurv()`

Examples

```
mod <- fit_bnnsurv(
  Surv(time, status) ~ age + karno + diagtime + prior,
  data = veteran
)
head(predict_bnnsurv(mod, newdata = veteran[1:5, ], times = c(50, 100)))
```

`fit_cforest` *Fit a Conditional Inference Survival Forest*

Description

Fits a survival forest model using the `party::cforest()` implementation of conditional inference trees for right-censored survival data.

Usage

```
fit_cforest(
  formula,
  data,
  teststat = "quad",
  testtype = "Univariate",
  mincriterion = 0,
  ntree = 500,
  mtry = 5,
  replace = TRUE,
  fraction = 0.632,
  ...
)
```

Arguments

`formula`	A `Surv()` survival formula specifying the outcome and predictors.
`data`	A data frame containing the variables in the model.
`teststat`	Character string specifying the test statistic to use (default = `"quad"`). See `party::cforest_control()`.
`testtype`	Character string specifying the type of test (default = `"Univariate"`).
`mincriterion`	Numeric, the value of the test statistic that must be exceeded for a split to be performed (default = `0`).
`ntree`	Integer, number of trees to grow (default = `500`).
`mtry`	Integer, number of variables randomly selected at each node (default = `5`).
`replace`	Logical, whether sampling of cases is with replacement (default = `TRUE`).
`fraction`	Proportion of samples to draw if `replace = FALSE` (default = `0.632`).
`...`	Additional arguments passed to `party::cforest_control()`.

Details

This function wraps `party::cforest()` to fit a conditional inference forest for survival analysis, returning an `mlsurv_model` object compatible with `predict_cforest()` and the `survalis` framework.

Value

An object of class `"mlsurv_model"`, containing:

`model`	The fitted `cforest` model.
`learner`	Character string `"cforest"`.
`formula`	The model formula.
`data`	The training data.

Examples

```
mod <- fit_cforest(Surv(time, status) ~ age + celltype + karno, data = veteran)
head(predict_cforest(mod, newdata = veteran[1:5, ], times = c(100, 200)))
```

`fit_coxph` *Fit a Cox Proportional Hazards Model*

Description

Fits a Cox proportional hazards regression model using the `survival::coxph()` function, and returns an object compatible with the `mlsurv_model` interface.

Usage

```
fit_coxph(formula, data, ...)
```

Arguments

`formula`	A survival formula of the form `Surv(time, status) ~ predictors`.
`data`	A data frame containing the variables in the model.
`...`	Additional arguments passed to `survival::coxph()`.

Details

The fitted object is stored along with metadata such as the learner name (`"coxph"`), original formula, data, and names of the time and status variables. The function requires the **survival** and **pec** packages.

Value

An object of class "mlsurv_model" containing:

- model – the fitted coxph object
- learner – the string "coxph"
- formula – the survival formula used
- data – the training dataset
- time, status – names of the survival outcome variables

Examples

```
mod_cox <- fit_coxph(Surv(time, status) ~ age + karno + celltype, data = veteran)
summary(mod_cox)
```

---

fit_flexsurvreg *Fit a Parametric Survival Regression Model Using flexsurvreg*

Description

This function fits a fully parametric survival regression model using the **flexsurv** package. It supports a variety of parametric distributions (e.g., Weibull, exponential, log-normal) and returns a model object compatible with the mlsurv_model interface for downstream survival predictions.

Usage

```
fit_flexsurvreg(formula, data, dist = "weibull", ...)
```

Arguments

formula	A Surv-based survival formula of the form Surv(time, status) ~ predictors.
data	A data frame containing the variables in the model.
dist	Character string specifying the parametric distribution to use (default = "weibull"). See flexsurvreg for available options.
...	Additional arguments passed to flexsurvreg.

Value

An object of class mlsurv_model containing:

- model - the fitted flexsurvreg model
- learner - string identifier "flexsurvreg"
- formula - the model formula
- data - the training data
- time - the name of the time-to-event column
- status - the name of the event indicator column

Examples

```
mod_flex <- fit_flexsurvreg(Surv(time, status) ~ age + celltype + karno,
                            data = veteran,
                            dist = "weibull")
```

`fit_glmnet` *Fit a Penalized Cox Proportional Hazards Model (glmnet)*

Description

Fits a Cox proportional hazards model with elastic net regularization using **glmnet**. The function wraps `cv.glmnet` to select the optimal penalty parameter λ by cross-validation.

Usage

```
fit_glmnet(formula, data, alpha = 1, ...)
```

Arguments

`formula`	A survival formula of the form `Surv(time, status) ~ predictors`.
`data`	A `data.frame` containing all variables referenced in `formula`.
`alpha`	Numeric value in `[0, 1]` specifying the elastic net mixing parameter: • `1` for LASSO penalty • `0` for ridge penalty • between 0 and 1 for elastic net
`...`	Additional arguments passed to `cv.glmnet`.

Value

An object of class `"mlsurv_model"` containing:

- `model` – the fitted `cv.glmnet` object
- `learner` – the string `"glmnet"`
- `formula`, `data`, `time`, and `status` metadata

See Also

`cv.glmnet`, `predict_glmnet`

Examples

```
mod_glmnet <- fit_glmnet(
  Surv(time, status) ~ age + karno + celltype,
  data = veteran
)

summary(mod_glmnet$model)
```

`fit_orsf` *Fit an Oblique Random Survival Forest (ORSF) Model*

Description

Fits an Oblique Random Survival Forest using the **aorsf** package. This method builds an ensemble of oblique decision trees for survival analysis, where splits are based on linear combinations of features, allowing for improved performance in high-dimensional or correlated feature settings.

Usage

```
fit_orsf(formula, data, ...)
```

Arguments

`formula`	A survival formula of the form `Surv(time, status) ~ predictors`.
`data`	A data frame containing the variables specified in `formula`.
`...`	Additional arguments passed to `aorsf::orsf()`.

Details

ORSF models extend traditional Random Survival Forests by allowing oblique splits, which can improve prediction accuracy when predictors are correlated. Missing data are omitted by default (`na_action = "omit"`).

Value

An object of class `"mlsurv_model"` containing:

- `model`: The fitted **aorsf** ORSF model object.
- `learner`: Character string, always `"orsf"`.
- `formula`: The survival formula used.
- `data`: The training dataset.
- `time`: Name of the survival time variable.
- `status`: Name of the event indicator variable.

References

Jaeger BC, Long DL, Long DM, Sims M, Szychowski JM, Min YI, Bandyopadhyay D. Oblique random survival forests. *Annals of Applied Statistics*. 2019;13(3):1847-1883. doi:10.1214/19-AOAS1261

Examples

```
mod <- fit_orsf(Surv(time, status) ~ age + karno, data = veteran)
summary(mod)
```

`fit_ranger` *Fit a Survival Random Forest Model Using ranger*

Description

This function fits a survival random forest model using the **ranger** package and returns an object compatible with the `mlsurv_model` class.

Usage

```
fit_ranger(formula, data, ...)
```

Arguments

`formula`	A survival formula of the form `Surv(time, status) ~ predictors`.
`data`	A `data.frame` containing the variables in the model.
`...`	Additional arguments passed to `ranger`.

Details

This function wraps `ranger` for survival analysis and stores the result in a standardized `mlsurv_model` object.

Value

An object of class `"mlsurv_model"` containing:

- `model` - the fitted `ranger` model
- `learner` - character string "ranger"
- `formula` - the model formula
- `data` - training data used to fit the model

See Also

`predict_ranger`, `tune_ranger`, `ranger`

Examples

```
mod <- fit_ranger(
  Surv(time, status) ~ age + karno + celltype,
  data = veteran,
  num.trees = 25
)
summary(mod)
```

fit_rpart *Fit a Survival Tree Model using* rpart

Description

Fits a survival tree using the **rpart** package with an exponential splitting rule. This learner is compatible with the mlsurv_model interface and can be used with the cv_survlearner() and tune_*() functions in the **survalis** framework.

Usage

```
fit_rpart(
  formula,
  data,
  minsplit = 20,
  minbucket = round(minsplit/3),
  cp = 0.01,
  maxcompete = 4,
  maxsurrogate = 5,
  usesurrogate = 2,
  xval = 10,
  surrogatestyle = 0,
  maxdepth = 30
)
```

Arguments

formula	A survival formula of the form Surv(time, status) ~ predictors.
data	A data.frame containing the variables in the model.
minsplit	Minimum number of observations that must exist in a node in order for a split to be attempted. Default is 20.
minbucket	Minimum number of observations in any terminal node. Default is round(minsplit / 3).
cp	Complexity parameter for pruning. Default is 0.01.
maxcompete	Number of competitor splits retained in the output. Default is 4.
maxsurrogate	Number of surrogate splits retained in the output. Default is 5.
usesurrogate	How surrogates are used in the splitting process. Default is 2.
xval	Number of cross-validations to perform in rpart. Default is 10.
surrogatestyle	Controls selection of surrogate splits. Default is 0.
maxdepth	Maximum depth of any node of the final tree. Default is 30.

Details

This function fixes the method = "exp" argument to fit an exponential splitting survival tree, which models the hazard function assuming exponential survival within terminal nodes.

Value

An object of class "mlsurv_model" containing:

- model – fitted rpart survival tree object
- learner – "rpart"
- formula, data, time, and status

References

Therneau TM, Atkinson EJ. (2019). *An Introduction to Recursive Partitioning Using the RPART Routines*. Mayo Clinic.

Examples

```
mod_rpart <- fit_rpart(Surv(time, status) ~ age + karno + celltype, data = veteran)
pred_rpart <- predict_rpart(mod_rpart, newdata = veteran[1:5, ], times = c(100, 200, 300))
head(pred_rpart)
```

---

fit_rsf *Fit a Random Survival Forest (RSF) Model*

Description

Fits a Random Survival Forest for right-censored time-to-event data using **randomForestSRC** and returns a standardized mlsurv_model compatible with the survalis framework.

Usage

```
fit_rsf(formula, data, ntree = 500, mtry = NULL, nodesize = 15, ...)
```

Arguments

formula	A survival formula of the form Surv(time, status) ~ predictors.
data	A data.frame containing the variables referenced in formula.
ntree	Integer; number of trees to grow (default: 500).
mtry	Integer or NULL; number of variables randomly selected at each split. If NULL, the engine's default is used.
nodesize	Integer; minimum terminal node size (default: 15).
...	Additional arguments forwarded to rfsrc.

Details

RSF extends random forests to survival data by growing an ensemble of survival trees on bootstrap samples and aggregating survival functions across trees.

Value

An object of class `mlsurv_model`, a named list with elements:

model The fitted `randomForestSRC::rfsrc` object.

learner Character scalar identifying the learner (`"rsf"`).

engine Character scalar naming the engine (`"randomForestSRC"`).

formula The original survival formula.

data The training dataset (or a minimal subset needed for prediction).

time Name of the survival time variable.

status Name of the event indicator (1 = event, 0 = censored).

References

Ishwaran H, Kogalur UB, Blackstone EH, Lauer MS (2008). Random survival forests. *Annals of Applied Statistics*, 2(3):841-860.

Examples

```
mod_rsf <- fit_rsf(Surv(time, status) ~ age + celltype + karno,
                   data = veteran, ntree = 200)

times <- c(100, 200, 300)
pred_probs <- predict_rsf(mod_rsf, newdata = veteran[1:5, ], times = times)
print(round(pred_probs, 3))
```

---

fit_selectcox *Fit a Predictor-Selection Cox Model (pec::selectCox, mlsurv_model-compatible)*

Description

Fits a Cox proportional hazards model with automated predictor selection using **pec**'s `selectCox()`, returning an `mlsurv_model` object compatible with the `survalis` evaluation and cross-validation pipeline.

Usage

```
fit_selectcox(formula, data, rule = "aic")
```

Arguments

`formula`	A survival formula of the form `Surv(time, status) ~ predictors`. The left-hand side must be a `Surv()` object from the **survival** package.
`data`	A `data.frame` containing all variables referenced in `formula`.
`rule`	Selection rule passed to `pec::selectCox()`, e.g., `"aic"` (default) or `"p"`.

Details

This wrapper standardizes the return object to the `mlsurv_model` contract so downstream prediction and evaluation behave consistently across learners.

Value

An object of class `mlsurv_model`, a named list with elements:

model The fitted `pec::selectCox` model.

learner Character scalar identifying the learner (`"selectcox"`).

engine Character scalar naming the engine (`"selectCox"`).

formula The original survival formula.

data The training dataset (or a minimal subset needed for prediction).

rule The selection rule used.

Engine

Uses **pec::selectCox**. Selection can be driven by Akaike's Information Criterion (`rule = "aic"`) or by p-value thresholds (`rule = "p"`), as implemented by **pec**.

References

Mogensen UB, Ishwaran H, Gerds TA (2012). Evaluating Random Forests for Survival Analysis using **pec**. Gerds TA, et al. **pec**: Prediction Error Curves for Survival Models.

See Also

`predict_selectcox()`, `tune_selectcox()`, `pec::selectCox()`

Examples

```
mod <- fit_selectcox(Surv(time, status) ~ age + celltype + karno, data = veteran)
```

`fit_stpm2` *Fit a Flexible Parametric Survival Model (rstpm2, mlsurv_model-compatible)*

Description

Fits a parametric/penalised generalised survival model using **rstpm2**'s `stpm2()`, returning an `mlsurv_model` object that integrates with the `survalis` evaluation and cross-validation pipeline. The baseline hazard is modeled with restricted cubic splines controlled by the degrees of freedom.

Usage

```
fit_stpm2(formula, data, df = 4, ...)
```

Arguments

formula	A survival formula of the form `Surv(time, status) ~ predictors`. The left-hand side must be a `Surv()` object from the **survival** package.
data	A `data.frame` containing all variables referenced in `formula`.
df	Integer degrees of freedom for the restricted cubic spline baseline. Default is `3`.
...	Additional arguments forwarded to `rstpm2::stpm2()`.

Details

This wrapper standardizes the model object to the `mlsurv_model` contract so downstream prediction and evaluation behave consistently across learners.

Value

An object of class `mlsurv_model`, a named list with elements:

model The fitted `rstpm2::stpm2` object.

learner Character scalar identifying the learner (`"stpm2"`).

engine Character scalar naming the engine (`"rstpm2"`).

formula The original survival formula.

data The training dataset (or a minimal subset needed for prediction).

time Name of the survival time variable.

status Name of the event indicator (1 = event, 0 = censored).

Engine

Uses **rstpm2::stpm2**. Spline complexity is governed by `df`; larger values allow more flexibility in the baseline hazard. Additional engine arguments can be passed via ....

References

Royston P, Parmar MKB (2002). Flexible parametric proportional-hazards and proportional-odds models for censored survival data. *Statistics in Medicine*. Lambert PC, Royston P, Crowther MJ (2009). Restricted cubic splines for non-proportional hazards. *Statistics in Medicine*.

Examples

```
mod_stpm2 <- fit_stpm2(Surv(time, status) ~ age + karno + celltype,
                       data = veteran, df = 4)
predict_stpm2(mod_stpm2, newdata = veteran[1:5, ], times = c(100, 200, 300))
summary(mod_stpm2)
```

`fit_survdnn` *Fit a Deep Neural Network Survival Model (mlsurv_model-compatible)*

Description

Fits a deep neural network survival model using **survdnn** (backed by **torch**). Supports the losses currently exposed by **survdnn** (`"cox"`, `"cox_l2"`, `"aft"`, and `"coxtime"`). Returns an `mlsurv_model` object that integrates with the `survalis` evaluation and cross-validation pipeline.

Usage

```
fit_survdnn(
  formula,
  data,
  loss = "cox",
  hidden = c(32L, 32L, 16L),
  activation = "relu",
  lr = 1e-04,
  epochs = 300L,
  optimizer = "adam",
  optim_args = list(),
  verbose = FALSE,
  dropout = 0.3,
  batch_norm = TRUE,
  callbacks = NULL,
  .seed = NULL,
  .device = "auto",
  na_action = "omit",
  ...
)
```

Arguments

`formula`	A survival formula of the form `Surv(time, status) ~ predictors`. The left-hand side must be a `Surv()` object from the `survival` package.
`data`	A `data.frame` containing all variables referenced in `formula`.
`loss`	Loss function name understood by **survdnn**. One of `"cox"`, `"cox_l2"`, `"aft"`, or `"coxtime"`.
`hidden`	Integer vector of hidden layer sizes (e.g., `c(32L, 32L, 16L)`).
`activation`	Activation function name. Supported options depend on the installed **survdnn** version and include `"relu"`, `"leaky_relu"`, `"tanh"`, `"sigmoid"`, `"gelu"`, `"elu"`, and `"softplus"`.
`lr`	Learning rate.
`epochs`	Number of training epochs.

optimizer	Optimizer name. One of "adam", "adamw", "sgd", "rmsprop", or "adagrad".
optim_args	Optional named list of extra optimizer arguments passed to **torch** via **survdnn**.
verbose	Logical; print training progress.
dropout	Numeric dropout rate in [0, 1].
batch_norm	Logical; whether to use batch normalization in hidden layers.
callbacks	Optional list of callback functions used by **survdnn**.
.seed	Optional integer random seed passed through to **survdnn**.
.device	Computation device. One of "auto", "cpu", or "cuda".
na_action	How to handle missing values. One of "omit" or "fail".
...	Additional arguments forwarded to the underlying engine.

Details

Design contract. All fit_*() functions in survalis: (i) return a named list with model, learner, engine, formula, data, time, and status; and (ii) retain information required by predict_*() to build a consistent design for new data.

Value

An object of class mlsurv_model, a named list with elements:

model The underlying fitted **survdnn** model.

learner Character scalar identifying the learner ("survdnn").

engine Character scalar naming the engine ("survdnn").

formula The original survival formula.

data The training dataset (or a minimal subset needed for prediction).

time Name of the survival time variable.

status Name of the event indicator (1 = event, 0 = censored).

Engine

Uses **survdnn::survdnn** with **torch**. The wrapper exposes the core training arguments used by the engine, including optimizer choice, dropout, batch normalization, callbacks, device selection, and missing-value handling.

References

survdnn documentation; **torch** for deep learning in R.

See Also

predict_survdnn(), tune_survdnn()

Examples

```
if (requireNamespace("survdnn", quietly = TRUE) &&
    requireNamespace("torch", quietly = TRUE) &&
    torch::torch_is_installed()) {
  mod <- fit_survdnn(Surv(time, status) ~ age + karno + celltype,
                     data = veteran, loss = "cox", epochs = 50, verbose = FALSE)

  pred <- predict_survdnn(mod, newdata = veteran[1:5, ], times = c(30, 90, 180))
  print(pred)
}
```

`fit_survmetalearner` *Fit a Stacked Survival Meta-Learner (Time-Varying NNLS)*

Description

Learns nonnegative, time-specific stacking weights over a set of base survival learners by solving a nonnegative least squares (NNLS) problem at each requested time t^*. For each t^*, the target is the indicator $Y = I(T > t^*)$; the features are the base learners' predicted survival probabilities $S_\ell(t^* \mid x)$. Weights are constrained to be nonnegative and are normalized to sum to 1 per time point.

Usage

```
fit_survmetalearner(
  base_preds,
  time,
  status,
  times,
  base_models,
  formula,
  data
)
```

Arguments

`base_preds`	A named list of matrices/data frames, one per base learner, each of dimension `n` x `length(times)`, with columns named `"t=<time>"`.
`time`	Numeric vector of observed event/censoring times (length `n`).
`status`	Numeric/binary vector of event indicators (1=event, 0=censor) (length `n`).
`times`	Numeric vector of evaluation times at which to learn weights.
`base_models`	A named list of fitted base learner objects; names must match `names(base_preds)` and the learner names used by `predict_*()`.
`formula`	A `Surv()` formula that was used for the base models (stored for metadata and downstream scoring).
`data`	The training data frame used for the base models (stored for metadata).

Details

For each `t` in `times`, this function fits

$$\min_{w \geq 0} midY - Sw\|_2^2 \quad \text{s.t.} \sum_j w_j = 1$$

where $Y = I(T > t)$ and S is the matrix of base survival probabilities at t. The NNLS solution from **nnls** is renormalized to sum to 1 (if the solution is all zeros, weights remain `NA` for that time).

Value

An object of class `c("mlsurv_model","survmetalearner")` with elements:

weights Matrix `L x T` of nonnegative stacking weights, rows=learners, cols=`"t=<time>"`.

base_models Named list of base learner fits.

base_preds Named list of base prediction matrices on training data.

learners Character vector of learner names (from `names(base_preds)`).

formula, data, time, status Training metadata for scoring/reporting.

learner The string `"survmetalearner"` (for `predict_*` dispatch).

See Also

`predict_survmetalearner`, `plot_survmetalearner_weights`, `cv_survmetalearner`

Examples

```
form <- Surv(time, status) ~ age + karno + trt
times <- c(80, 160)
mod_cox <- fit_coxph(form, data = veteran)
mod_rpart <- fit_rpart(form, data = veteran)
base_models <- list(coxph = mod_cox, rpart = mod_rpart)
base_preds <- list(
  coxph = predict_coxph(mod_cox, newdata = veteran, times = times),
  rpart = predict_rpart(mod_rpart, newdata = veteran, times = times)
)
meta_model <- fit_survmetalearner(
  base_preds = base_preds,
  time = veteran$time,
  status = veteran$status,
  times = times,
  base_models = base_models,
  formula = form,
  data = veteran
)
meta_model$weights
```

`fit_survsvm` *Fit a Survival SVM Model (mlsurv_model-compatible)*

Description

Fits a survival support vector machine using **survivalsvm**. The default setup uses the regression-type loss with quadratic programming optimization and an additive or linear kernel, but all **survivalsvm** options are exposed. Returns an `mlsurv_model` object that integrates with the `survalis` evaluation and cross-validation pipeline.

Usage

```
fit_survsvm(
  formula,
  data,
  type = "regression",
  gamma.mu = 0.1,
  opt.meth = "quadprog",
  kernel = "add_kernel",
  diff.meth = NULL
)
```

Arguments

`formula`	A survival formula of the form `Surv(time, status) ~ predictors`. The left-hand side must be a `Surv()` object from the `survival` package.
`data`	A `data.frame` containing all variables referenced in `formula`.
`type`	SVM loss type used by `survivalsvm` (e.g., `"regression"`).
`gamma.mu`	Regularization parameter for the margin/hinge component.
`opt.meth`	Optimization method (e.g., `"quadprog"`).
`kernel`	Kernel type (e.g., `"lin_kernel"`, `"add_kernel"`).
`diff.meth`	Optional differentiation method passed to the engine.

Details

Design contract. All `fit_*()` functions in `survalis`: (i) return a named list with `model`, `learner`, `engine`, `formula`, `data`, `time`, and `status`; (ii) preserve `terms(formula)` or equivalent for consistent prediction design; and (iii) keep engine arguments required downstream by `predict_*()`.

Value

An object of class `mlsurv_model`, a named list with elements:

model The underlying fitted **survivalsvm** model.

learner Character scalar identifying the learner (`"survsvm"`).

engine Character scalar naming the engine (`"survivalsvm"`).

formula The original survival formula.

data The training dataset (or a minimal subset needed for prediction).

time Name of the survival time variable.

status Name of the event indicator (1 = event, 0 = censored).

Engine

Uses **survivalsvm::survivalsvm**. Some optimization methods (e.g., `"quadprog"`) require additional system dependencies and the **quadprog** package to be installed.

References

Binder H, et al. (2009). Survival Support Vector Machines (and related work). **survivalsvm** package documentation.

See Also

`predict_survsvm()`, `tune_survsvm()`

Examples

```
mod_svm <- fit_survsvm(Surv(time, status) ~ age + celltype + karno,
                           data = veteran,
                           type = "regression",
                           gamma.mu = 0.1,
                           kernel = "lin_kernel")

times <- c(100, 300, 500)
predict_survsvm(mod_svm, newdata = veteran[1:5, ], times = times, dist = "exp")
predict_survsvm(mod_svm, newdata = veteran[1:5, ], times = times, dist = "weibull", shape = 1.5)

cv_results_svm <- cv_survlearner(
  formula = Surv(time, status) ~ age + celltype + karno,
  data = veteran,
  fit_fun = fit_survsvm,
  pred_fun = predict_survsvm,
  times = c(100, 300, 500),
  metrics = c("cindex", "ibs"),
  folds = 5,
  seed = 42,
  gamma.mu = 0.1,
  kernel = "lin_kernel")

print(cv_results_svm)
cv_summary(cv_results_svm)
cv_plot(cv_results_svm)
```

`fit_xgboost` *Fit an XGBoost Survival Model (mlsurv_model-compatible)*

Description

Fits a gradient-boosted tree model for time-to-event outcomes using **xgboost**. Supports `"survival:aft"` (default) and `"survival:cox"`. Returns an `mlsurv_model` object compatible with the `survalis` pipeline.

Usage

```
fit_xgboost(
  formula,
  data,
  booster = "gbtree",
  objective = "survival:aft",
  aft_loss_distribution = "extreme",
  aft_loss_distribution_scale = 1,
  nrounds = 100
)
```

Arguments

`formula`	A survival formula of the form `Surv(time, status) ~ predictors`.
`data`	A `data.frame` containing variables in `formula`.
`booster`	XGBoost booster type (default `"gbtree"`).
`objective`	One of `"survival:aft"` (default) or `"survival:cox"`.
`aft_loss_distribution`	AFT error distribution: `"extreme"` (default), `"logistic"`, or `"normal"`.
`aft_loss_distribution_scale`	Positive numeric scale for the AFT loss (default `1`).
`nrounds`	Integer number of boosting iterations (default `100`).

Details

Design contract: returns a named list with engine metadata and preserves `terms(formula)` so prediction uses the exact same encoding as training.

Value

An object of class `mlsurv_model`:

model Fitted `xgboost` model.

learner `"xgboost"`.

formula, data Original inputs.

time, status Names of the survival time and event variables.

objective Objective used.

dist, scale AFT distribution and scale (populated even if Cox is used).

terms Preserved `terms` for consistent prediction design matrices.

Engine

Uses **xgboost**. For `"survival:aft"`, interval labels are set via `label_lower_bound`/`label_upper_bound` with right-censoring represented by `Inf` on the upper bound. For `"survival:cox"`, labels follow the Cox objective convention.

See Also

`predict_xgboost()`, `tune_xgboost()`

Examples

```
mod_xgb <- fit_xgboost(
  Surv(time, status) ~ age + karno + celltype,
  data = veteran
)
```

`iae_survmat` *Integrated Absolute Error Against Kaplan-Meier*

Description

Computes $\int \mid \bar{S}(t) - \hat{S}_{KM}(t) \mid \, dt$ where $\bar{S}(t)$ is the mean predicted survival across subjects and $\hat{S}_{KM}(t)$ is the Kaplan-Meier estimate. Integration is carried out over KM event times using a left Riemann sum.

Usage

```
iae_survmat(object, sp_matrix, times)
```

Arguments

`object`	A `Surv` object.
`sp_matrix`	Matrix/data frame of survival probabilities (rows = subjects, columns aligned with `times`).
`times`	Numeric vector of times corresponding to the columns of `sp_matrix`.

Value

A named numeric scalar: `"iae"`.

Examples

```
y <- survival::Surv(time = veteran$time, event = veteran$status)
times <- c(60, 120)
lp <- stats::plogis(scale(veteran$karno))
sp <- cbind("t=60" = pmin(1, lp + 0.05), "t=120" = pmax(0, lp - 0.05))
colnames(sp) <- c("t=60", "t=120")
iae_survmat(y, sp_matrix = sp, times = times)
```

---

ibs_survmat *Integrated Brier Score (Discrete Integration)*

Description

Computes the Integrated Brier Score (IBS) over a vector of times, using discrete left Riemann integration of IPCW Brier scores.

Usage

```
ibs_survmat(object, sp_matrix, times)
```

Arguments

`object`	A `Surv` object.
`sp_matrix`	Matrix/data frame of survival probabilities with one column per time in `times`.
`times`	Numeric vector of strictly increasing times (length must equal `ncol(sp_matrix)`).

Value

A named numeric scalar: `"ibs"`.

Examples

```
y <- survival::Surv(time = veteran$time, event = veteran$status)
times <- c(60, 120)
lp <- stats::plogis(scale(veteran$karno))
sp <- cbind("t=60" = pmin(1, lp + 0.05), "t=120" = pmax(0, lp - 0.05))
colnames(sp) <- c("t=60", "t=120")
ibs_survmat(y, sp_matrix = sp, times = times)
```

ise_survmat *Integrated Squared Error Against Kaplan-Meier*

Description

Computes $\int (\bar{S}(t) - \hat{S}_{KM}(t))^2 \, dt$ where $\bar{S}(t)$ is the mean predicted survival and $\hat{S}_{KM}(t)$ is the Kaplan-Meier curve. Integration uses KM event times and a left Riemann sum.

Usage

```
ise_survmat(object, sp_matrix, times)
```

Arguments

object	A Surv object.
sp_matrix	Matrix/data frame of survival probabilities (rows = subjects, columns aligned with times).
times	Numeric vector of times corresponding to the columns of sp_matrix.

Value

A named numeric scalar: "ise".

Examples

```
y <- survival::Surv(time = veteran$time, event = veteran$status)
times <- c(60, 120)
lp <- stats::plogis(scale(veteran$karno))
sp <- cbind("t=60" = pmin(1, lp + 0.05), "t=120" = pmax(0, lp - 0.05))
colnames(sp) <- c("t=60", "t=120")
ise_survmat(y, sp_matrix = sp, times = times)
```

list_interpretability_methods *List interpretability methods available in survalis*

Description

Returns a table mapping each compute_* function to its paired plot_* helper (if any). Methods without a plot helper show NA.

Usage

```
list_interpretability_methods()
```

Value

A tibble with columns `compute`, `plot`, `has_compute`, and `has_plot`.

Examples

```
list_interpretability_methods()
subset(list_interpretability_methods(), is.na(plot))  # methods without plot
```

`list_metrics` *List Available Evaluation Metrics*

Description

Returns a data frame describing the evaluation metrics supported by `survalis` (as used by helpers like `cv_survlearner()` and `score_survmodel()`).

Usage

```
list_metrics()
```

Details

Columns:

- `metric`: short metric name used throughout the package.
- `direction`: whether higher or lower values are better.
- `summary`: brief description.
- `range`: typical value range.

Value

A tibble/data.frame with one row per metric.

Examples

```
list_metrics()
```

list_survlearners *List survival learners available in survalis*

Description

Returns a table of known survival learners, showing the expected `fit_*`, `predict_*`, and (if present) `tune_*` functions, plus booleans indicating which functions are available.

Usage

```
list_survlearners(has_tune = FALSE)
```

Arguments

has_tune	Logical (default FALSE). If TRUE, return only learners that have a corresponding `tune_*` function. If FALSE, return all.

Value

A tibble with columns:

- `learner` – learner id (e.g., "ranger")
- `fit`, `predict`, `tune` – function names (tune may be NA)
- `has_fit`, `has_predict`, `has_tune` – logical flags
- `available` – `has_fit` & `has_predict`

Examples

```
list_survlearners()                  # all learners (default)
list_survlearners(has_tune = TRUE)   # only tunable learners
```

list_tunable_survlearners *List tunable survival learners*

Description

Filters learners to only those that provide a `tune_*` function in addition to `fit_*` and `predict_*`.

Usage

```
list_tunable_survlearners()
```

Value

A base `data.frame` like `list_survlearners()` but containing only rows where `tune` is not NA.

Examples

```
list_tunable_survlearners()
```

`plot_ale` *Plot ALE Curves for Survival Models*

Description

Visualizes ALE results produced by `compute_ale()` either as per-time curves (one curve per evaluation time) or as an integrated curve averaged across times.

Usage

```
plot_ale(
  ale_result,
  feature,
  which = c("per_time", "integrated"),
  smooth = FALSE
)
```

Arguments

`ale_result`	A list returned by `compute_ale()`.
`feature`	Character name of the feature (for axis labeling only).
`which`	Either `"per_time"` to plot one ALE curve per time, or `"integrated"` to plot the time-averaged ALE (requires multiple `times`).
`smooth`	Logical; if `TRUE`, overlays a LOESS smooth on the plotted curve(s).

Details

Per-time plots show how the feature's local effect varies across different evaluation times. The integrated plot summarizes the average effect over the supplied time grid (simple mean across times of the centered ALE values).

Value

A `ggplot2` object.

See Also

`compute_ale()`, `compute_pdp()`, `plot_pdp()`

Examples

```
mod <- fit_coxph(Surv(time, status) ~ age + karno + celltype, data = veteran)
ale_res <- compute_ale(
  model = mod,
  newdata = veteran,
  feature = "karno",
  times = c(80, 160),
  grid.size = 8
)
plot_ale(ale_res, feature = "karno", which = "per_time")
plot_ale(ale_res, feature = "karno", which = "integrated", smooth = TRUE)
```

---

plot_benchmark *Plot Benchmark Distributions Across Learners*

Description

Produces box-and-jitter plots of CV metric values per learner, faceted by metric for quick visual comparison.

Usage

```
plot_benchmark(benchmark_results)
```

Arguments

benchmark_results	A data frame from `benchmark_default_survlearners()` with columns `learner`, `metric`, and `value`.

Value

A **ggplot2** object.

See Also

`benchmark_default_survlearners()`, `summarise_benchmark()`

Examples

```
res <- tibble::tibble(
  learner = c("coxph", "coxph", "rpart", "rpart"),
  metric = c("cindex", "ibs", "cindex", "ibs"),
  value = c(0.64, 0.19, 0.60, 0.23)
)
plot_benchmark(res)
```

`plot_calibration` *Plot Calibration Curve for Survival Predictions*

Description

Produces a calibration plot comparing mean predicted survival (x-axis) to observed survival with bootstrap CIs (y-axis) at a single evaluation time.

Usage

```
plot_calibration(calib_output, smooth = TRUE)
```

Arguments

`calib_output`	Output list returned by `compute_calibration()`.
`smooth`	Logical; if `TRUE`, overlays a LOESS smooth of `observed_surv ~ mean_pred_surv`.

Details

Points above the diagonal indicate underprediction (observed survival higher than predicted), while points below indicate overprediction.

Value

A `ggplot2` object showing bin-wise calibration points, bootstrap error bars, the 45° reference line, and (optionally) a smooth curve.

See Also

`compute_calibration()`

Examples

```
mod <- fit_coxph(Surv(time, status) ~ age + karno + celltype, data = veteran)
calib <- compute_calibration(
  model = mod,
  data = veteran,
  time = "time",
  status = "status",
  eval_time = 80,
  n_bins = 4,
  n_boot = 5,
  seed = 1
)
plot_calibration(calib)
```

plot_counterfactual *Plot Counterfactual Recommendations*

Description

Visualizes counterfactual feature changes returned by compute_counterfactual, ranking recommendations by either raw survival gain or penalized gain.

Usage

```
plot_counterfactual(
  counterfactual_df,
  metric = c("penalized_gain", "survival_gain"),
  top_n = NULL,
  include_negative = FALSE
)
```

Arguments

counterfactual_df	A data frame returned by compute_counterfactual().
metric	Character scalar; one of "penalized_gain" (default) or "survival_gain" indicating which column to plot on the x-axis.
top_n	Optional integer limiting the plot to the top n features after sorting by metric.
include_negative	Logical; if FALSE (default), only positive-gain recommendations are shown.

Value

A **ggplot2** object.

Examples

```
df <- veteran
df$A <- df$trt
mod <- fit_coxph(survival::Surv(time, status) ~ A + age + karno, data = df)
cf <- compute_counterfactual(
  model = mod,
  newdata = df[1, , drop = FALSE],
  times = c(50, 100, 150),
  target_time = 100,
  features_to_change = c("A", "age", "karno"),
  grid.size = 10
)
plot_counterfactual(cf)
```

`plot_interactions` *Plot Interaction Strengths for Survival Models*

Description

Visualizes interaction outputs from `compute_interactions` as (i) a ranked bar chart for one-way interactions, (ii) a pairwise heatmap, or (iii) time-varying interaction trajectories.

Usage

```
plot_interactions(object, type = c("1way", "heatmap", "time"))
```

Arguments

`object`	A data frame returned by `compute_interactions()` whose columns match the requested `type`.
`type`	One of `"1way"`, `"heatmap"`, or `"time"`.

Details

1way: Bars rank features by Friedman-H interaction strength at the target time. **heatmap**: Tiles show pairwise interaction magnitudes (symmetric). **time**: Lines show interaction strength vs. time for each feature.

Value

A **ggplot2** object.

Examples

```
mod <- fit_coxph(Surv(time, status) ~ age + karno + trt, data = veteran)
times <- c(80, 160)
ia <- compute_interactions(
  model = mod,
  data = veteran,
  times = times,
  target_time = 80,
  features = c("age", "karno"),
  type = "1way",
  grid.size = 6
)
plot_interactions(ia, type = "1way")
```

plot_pdp *Plot PDP/ICE Curves for Survival Models*

Description

Plots partial dependence (PDP) and/or individual conditional expectation (ICE) results returned by compute_pdp either per evaluation time or as an integrated PDP over time.

Usage

```
plot_pdp(
  pdp_ice_output,
  feature,
  method = "pdp+ice",
  ids = NULL,
  which = c("per_time", "integrated"),
  alpha_ice = 0.2,
  smooth = FALSE
)
```

Arguments

pdp_ice_output	The list returned by compute_pdp().
feature	Character scalar; the same feature analyzed in compute_pdp().
method	One of "pdp", "ice", or "pdp+ice" indicating which curves to draw in per-time plots (ignored for integrated plots).
ids	Optional vector of row ids (.id) to subset ICE curves for clarity in per-time plots.
which	One of "per_time" (facet by time) or "integrated" (plot the time-integrated PDP if available).
alpha_ice	Alpha transparency for ICE lines/boxes in per-time plots (default 0.2).
smooth	Logical; if TRUE and the feature is numeric, apply a smooth curve (loess) for integrated PDP; otherwise draw a line.

Details

Per-time: For numeric features, draws ICE lines and PDP overlays per time. For categorical features, shows ICE as boxplots per level and PDP as point summaries. **Integrated:** Plots the PDP integrated across time (if provided by compute_pdp()); numeric features can be smoothed with smooth=TRUE.

Value

A **ggplot2** object.

Examples

```
mod <- fit_coxph(Surv(time, status) ~ age + karno + trt, data = veteran)
pdp_age <- compute_pdp(
  model = mod,
  data = veteran,
  feature = "age",
  times = c(80, 160),
  method = "pdp+ice",
  grid.size = 8
)
plot_pdp(pdp_age, feature = "age", which = "per_time")
plot_pdp(pdp_age, feature = "age", which = "integrated", smooth = TRUE)
```

---

plot_shap *Plot SHAP-like contributions for survival models*

Description

Plots time-dependent SHAP estimates (lines over time) or aggregated SHAP (bar chart) returned by `compute_shap()`.

Usage

```
plot_shap(shapley_result, type = c("auto"))
```

Arguments

`shapley_result`	A data frame returned by `compute_shap()`. If it has a `time` column, a time plot is produced; otherwise an aggregated bar plot.
`type`	One of `"auto"` (default), `"time"`, or `"integrated"`. With `"auto"`, the presence of a `time` column determines the plot type.

Value

A **ggplot2** object.

Examples

```
mod <- fit_coxph(survival::Surv(time, status) ~ age + karno + celltype, data = veteran)
shap_td <- compute_shap(mod, veteran[10, , drop = FALSE],
veteran, times = c(50, 100), sample.size = 5)
p1 <- plot_shap(shap_td)      # auto -> time plot

shap_ag <- compute_shap(mod, veteran[10, , drop = FALSE],
veteran, times = c(50, 100),
sample.size = 5, aggregate = TRUE, method = "meanabs")
p2 <- plot_shap(shap_ag)      # auto -> bar plot
```

plot_surrogate *Plot Local Surrogate Explanation*

Description

Visualizes the signed local effects from compute_surrogate as a horizontal bar chart, optionally limiting to the top n contributors.

Usage

```
plot_surrogate(surrogate_df, top_n = NULL)
```

Arguments

surrogate_df	A data frame returned by compute_surrogate().
top_n	Optional integer; if provided, display only the top top_n features by absolute effect.

Value

A **ggplot2** object showing feature contributions $\beta_j \cdot x_j$ (positive/negative) at the target time.

Examples

```
mod <- fit_coxph(Surv(time, status) ~ age + karno + celltype, data = veteran)
local_expl <- compute_surrogate(
  model = mod,
  newdata = veteran[2, , drop = FALSE],
  baseline_data = veteran,
  times = c(80, 160),
  target_time = 80,
  k = 3
)
plot_surrogate(local_expl, top_n = 3)
```

plot_survmat *Plot Predicted Survival Curves from a survmat*

Description

Plots one or more predicted survival curves from a survival-probability matrix. By default, each row of S is shown as an individual curve. If a group vector is supplied, curves are summarized by group using the requested aggregation function.

Usage

```
plot_survmat(
  S,
  times = NULL,
  group = NULL,
  ids = NULL,
  summary_fun = c("mean", "median"),
  show_individual = NULL,
  alpha = 0.2,
  linewidth = 0.7
)
```

Arguments

S	A surv_mat: numeric matrix/data.frame of survival probabilities.
times	Optional numeric vector of time points corresponding to columns of S. If omitted, times are inferred from column names of the form "t=<time>".
group	Optional vector of group labels of length nrow(S). When supplied, grouped summary curves are plotted.
ids	Optional integer vector of row ids to subset before plotting.
summary_fun	Aggregation for grouped curves; one of "mean" (default) or "median".
show_individual	Logical. If NULL (default), individual curves are shown only when group is omitted. If TRUE, individual curves are overlaid beneath the grouped summary curves.
alpha	Alpha transparency for individual curves (default 0.2).
linewidth	Line width for plotted curves (default 0.7).

Value

A **ggplot2** object.

Examples

```
S <- data.frame(`t=1` = c(0.95, 0.90, 0.92),
                `t=2` = c(0.80, 0.70, 0.78),
                `t=3` = c(0.60, 0.45, 0.55),
                check.names = FALSE)
plot_survmat(S)
plot_survmat(S, group = c("A", "B", "A"))
```

plot_survmetalearner_weights *Plot Time-Varying Stacking Weights*

Description

Visualizes the learned nonnegative NNLS stacking weights $w_\ell(t)$ over time for each base learner.

Usage

```
plot_survmetalearner_weights(model)
```

Arguments

model	A "survmetalearner" object from fit_survmetalearner.

Value

A **ggplot2** object showing weight trajectories (one line per learner).

Examples

```
form <- Surv(time, status) ~ age + karno + trt
times <- c(80, 160)
mod_cox <- fit_coxph(form, data = veteran)
mod_rpart <- fit_rpart(form, data = veteran)
base_models <- list(coxph = mod_cox, rpart = mod_rpart)
base_preds <- list(
  coxph = predict_coxph(mod_cox, newdata = veteran, times = times),
  rpart = predict_rpart(mod_rpart, newdata = veteran, times = times)
)
meta_model <- fit_survmetalearner(
  base_preds = base_preds,
  time = veteran$time,
  status = veteran$status,
  times = times,
  base_models = base_models,
  formula = form,
  data = veteran
)
plot_survmetalearner_weights(meta_model)
```

`plot_tree_surrogate` *Plot Tree-Based Surrogate Models or Feature Importances*

Description

Visualizes the results of a `tree_surrogate` object returned by `compute_tree_surrogate()`. Can display either the fitted surrogate trees or aggregated feature importance based on split counts.

Usage

```
plot_tree_surrogate(tree_surrogate, type = c("tree", "importance"), top_n = 10)
```

Arguments

`tree_surrogate`	An object of class `"tree_surrogate"`.
`type`	Character string indicating the type of plot: • `"tree"`: displays the surrogate decision tree(s) for each time. • `"importance"`: displays a bar plot of top features ranked by split count.
`top_n`	Integer, the number of top features to display in the importance plot. Ignored if `type = "tree"`.

Details

- If `type = "tree"`, a separate tree diagram is produced for each evaluation time.
- If `type = "importance"`, feature split counts are summed across all times and plotted as a bar chart.

Requires the `partykit` package for tree plotting and `ggplot2` for importance plotting.

Value

A plot object (for `"importance"`) or printed tree diagrams (for `"tree"`).

Examples

```
plot_tree_surrogate(tree_ranger, type = "tree")
plot_tree_surrogate(tree_ranger, type = "importance", top_n = 5)
```

plot_varimp *Plot Permutation Variable Importance*

Description

Creates a dot plot of permutation-based variable importance, using either the scaled importance (default) or the raw importance column.

Usage

```
plot_varimp(varimp_df, use_scaled = TRUE)
```

Arguments

varimp_df	A data frame as returned by compute_varimp.
use_scaled	Logical; if TRUE (default), plot scaled_importance (percent). If unavailable, falls back to raw importance with a warning.

Value

A **ggplot2** object.

Examples

```
mod <- fit_coxph(survival::Surv(time, status) ~ age + karno + celltype, data = veteran)
imp <- compute_varimp(
  model = mod,
  times = 80,
  metric = "brier",
  n_repetitions = 3,
  seed = 1,
  subset = 40
)
plot_varimp(imp, use_scaled = TRUE)
plot_varimp(imp, use_scaled = FALSE)
```

predict_aalen *Predict Survival from an Aalen Additive Hazards Model*

Description

Computes survival probabilities at specified time points from a model fitted with fit_aalen.

Usage

```
predict_aalen(object, newdata, times)
```

Arguments

object	An "mlsurv_model" returned by fit_aalen.
newdata	A data frame of new observations.
times	Numeric vector of time points at which to evaluate survival probabilities.

Value

A data frame (rows = observations, columns = "t=<time>").

Examples

```
mod <- fit_aalen(Surv(time, status) ~ trt + karno + age, data = veteran, max.time = 600)
head(predict_aalen(mod, newdata = veteran[1:5, ], times = 0:10))
```

---

predict_aftgee *Predict Survival Probabilities from an* aftgee *Model*

Description

Computes survival probabilities $S(t \mid x)$ at specified time points from a fitted AFT model using a log-normal approximation for the error distribution.

Usage

```
predict_aftgee(object, newdata, times = NULL)
```

Arguments

object	An mlsurv_model produced by fit_aftgee.
newdata	A data.frame of predictor values for prediction.
times	Numeric vector of time points at which to estimate survival probabilities. If NULL, a sequence of 20 values from 1 to the maximum observed time in object$data is used.

Details

We use the approximation

$$S(t \mid x) \approx 1 - \Phi\left(\frac{\log t - x^\top \hat{\beta}}{\sigma}\right),$$

where Φ is the standard normal CDF. Here we set $\sigma = 1$ as a simple, distribution-agnostic proxy; this yields a monotone-in-time score useful for benchmarking. For production use, prefer a parametric AFT fit where σ is estimated.

Value

A data.frame of survival probabilities with one row per `newdata` observation and one column per requested time (`"t=<time>"`).

Examples

```
mod <- fit_aftgee(Surv(time, status) ~ trt + karno + age, data = veteran)
predict_aftgee(mod, newdata = veteran[1:5, ], times = c(20, 60, 120))
```

`predict_bart` *Predict Survival Probabilities from a BART Survival Model*

Description

Generates survival probability predictions at requested times for new data using a model fitted by `fit_bart`.

Usage

```
predict_bart(object, newdata, times)
```

Arguments

`object`	An `mlsurv_model` returned by `fit_bart`.
`newdata`	A `data.frame` of new observations for prediction.
`times`	Numeric vector of time points at which to estimate survival probabilities (same scale as the training time).

Details

The BART engine predicts survival on its internal grid `object$eval_times`. Requested `times` are aligned to that grid by nearest-neighbor matching, returning one survival estimate per requested time.

Value

A base `data.frame` with one row per observation in `newdata` and columns named `"t=<time>"` (character), containing survival probabilities in [0, 1].

See Also

`fit_bart`, `surv.bart`

Examples

```
ex_data <- veteran[1:40, c("time", "status", "age", "karno", "celltype")]
mod_bart <- fit_bart(
  Surv(time, status) ~ age + karno + celltype,
  data = ex_data,
  K = 1,
  ntree = 5,
  ndpost = 20,
  nskip = 5,
  mc.cores = 1,
  seed = 42
)
predict_bart(mod_bart, newdata = ex_data[1:5, ], times = c(10, 30, 60))
```

---

predict_blackboost *Predict Survival Probabilities from a blackboost Model*

Description

Generates survival probabilities at requested times from a fitted **mboost** Cox boosting model produced by `fit_blackboost()`.

Usage

```
predict_blackboost(object, newdata, times, ...)
```

Arguments

`object`	An `mlsurv_model` returned by `fit_blackboost()`.
`newdata`	A `data.frame` of new observations for prediction.
`times`	Numeric vector of time points at which to compute survival probabilities.
`...`	Additional arguments forwarded to `mboost::survFit()`.

Details

Predictions use `mboost::survFit()` to obtain a step function \(S(t)\) per observation. If any requested `times` exceed the model's maximum time, the last survival value is carried forward (right-constant). Values are then matched to `times` using stepwise (piecewise-constant) interpolation.

Value

A base `data.frame` with one row per observation in `newdata` and columns named `"t=<time>"` (character), containing survival probabilities in [0, 1].

Examples

```
mod <- fit_blackboost(Surv(time, status) ~ age + karno + celltype, data = veteran)
predict_blackboost(mod, newdata = veteran[1:5, ], times = c(5, 10, 40))
```

predict_bnnsurv *Predict Survival with a bnnSurvival Model*

Description

Generates survival probabilities from an `mlsurv_model` fitted by `fit_bnnsurv`. If `times` is supplied, survival curves are linearly interpolated from the engine's internal time grid to those times.

Usage

```
predict_bnnsurv(object, newdata, times = NULL)
```

Arguments

`object`	A fitted `mlsurv_model` returned by `fit_bnnsurv()`.
`newdata`	A data frame of new observations for prediction.
`times`	Optional numeric vector of evaluation time points. If `NULL` (default), the function returns survival on the engine's native grid.

Details

Internally, predictions are obtained via `bnnSurvival::predict()`. The returned survival matrix is post–processed to enforce monotonicity (cumulative minimum over time). When `times` is provided, values are obtained by `stats::approx()` (linear interpolation, rule = 2).

Value

A base `data.frame` with one row per observation and columns named `"t=<time>"` containing survival probabilities in `[0, 1]`. Probabilities are clipped to `[0, 1]` and made non–increasing over time.

See Also

`fit_bnnsurv()`, `tune_bnnsurv()`

Examples

```
mod <- fit_bnnsurv(Surv(time, status) ~ age + karno + diagtime + prior, data = veteran)
pred <- predict_bnnsurv(mod, newdata = veteran[1:3, ], times = c(50, 100, 200))
pred
```

`predict_cforest` *Predict Survival Probabilities from a Conditional Inference Survival Forest*

Description

Generates predicted survival probabilities at specified time points from a fitted `cforest` survival model.

Usage

```
predict_cforest(object, newdata, times, ...)
```

Arguments

`object`	An `mlsurv_model` object returned by `fit_cforest()`.
`newdata`	A data frame containing the predictor variables for prediction.
`times`	A numeric vector of time points at which to estimate survival probabilities.
`...`	Not used, included for compatibility.

Details

Survival curves are extracted from the fitted `cforest` model and linearly interpolated to the requested time points.

Value

A data frame of survival probabilities with one row per observation in `newdata` and one column per requested time point (named `"t=<time>"`).

Examples

```
mod <- fit_cforest(Surv(time, status) ~ age + celltype + karno, data = veteran)
predict_cforest(mod, newdata = veteran[1:5, ], times = c(100, 200, 300))
```

predict_coxph *Predict Survival Probabilities from a Cox PH Model*

Description

Generates predicted survival probabilities at specified time points for new data, using a fitted Cox proportional hazards model.

Usage

```
predict_coxph(object, newdata, times)
```

Arguments

object	An "mlsurv_model" object returned by fit_coxph().
newdata	A data frame containing the predictor variables for which to compute predictions.
times	A numeric vector of time points at which to evaluate survival probabilities.

Details

Predictions are computed using pec::predictSurvProb(). The output is formatted as a data frame with one row per observation in newdata and one column per time point.

Value

A data frame of survival probabilities with columns named "t=<time>" for each requested time.

Examples

```
mod_cox <- fit_coxph(Surv(time, status) ~ age + karno + celltype, data = veteran)
predict_coxph(mod_cox, newdata = veteran[1:5, ], times = c(100, 200, 300))
```

predict_flexsurvreg *Predict Survival Probabilities from a flexsurvreg Model*

Description

This function generates survival probability predictions at specified time points from a model fitted with fit_flexsurvreg.

Usage

```
predict_flexsurvreg(object, newdata, times, ...)
```

Arguments

object	A fitted mlsurv_model object returned by fit_flexsurvreg.
newdata	A data frame of new observations for which to predict survival probabilities.
times	Numeric vector of time points at which to estimate survival probabilities.
...	Additional arguments passed to predict.flexsurvreg.

Value

A data frame of survival probabilities with one row per observation in newdata and one column per time point. Column names are of the form "t=100", "t=200", etc.

Examples

```
mod_flex <- fit_flexsurvreg(Surv(time, status) ~ age + celltype + karno,
                            data = veteran,
                            dist = "weibull")
predict_flexsurvreg(mod_flex, newdata = veteran[1:5, ],
                    times = c(100, 200, 300))
```

---

predict_glmnet *Predict Survival Probabilities from a Penalized Cox Model (glmnet)*

Description

Predicts survival probabilities at specified time points from a fitted penalized Cox proportional hazards model produced by fit_glmnet.

Usage

```
predict_glmnet(object, newdata, times, ...)
```

Arguments

object	A fitted "mlsurv_model" object from fit_glmnet.
newdata	A data.frame of new observations for prediction.
times	Numeric vector of time points at which to estimate survival probabilities. Must be in the same scale as the training time variable.
...	Not used.

Details

Predictions are computed by:

1. Computing the linear predictors for training and new data at s = "lambda.min".
2. Fitting a Cox PH model on the training linear predictors to estimate the baseline cumulative hazard.
3. Interpolating the baseline hazard at each times and transforming via $S(t \mid x) = \exp(-H_0(t)\exp(\eta))$.

Value

A `data.frame` with one row per observation in `newdata` and one column per requested time point (`"t=<time>"`).

See Also

`fit_glmnet`

Examples

```
mod_glmnet <- fit_glmnet(
  Surv(time, status) ~ age + karno + celltype,
  data = veteran
)

predict_glmnet(
  mod_glmnet,
  newdata = veteran[1:5, ],
  times = c(100, 200, 300)
)
```

`predict_orsf` *Predict Survival Probabilities from an ORSF Model*

Description

Generates survival probability predictions at specified time points from a fitted Oblique Random Survival Forest model.

Usage

```
predict_orsf(object, newdata, times, ...)
```

Arguments

`object`	A fitted ORSF model object from `fit_orsf`.
`newdata`	A data frame of new observations for prediction.
`times`	A numeric vector of time points at which to estimate survival probabilities.
`...`	Additional arguments passed to `predict.aorsf()`.

Details

Predictions are computed using the `pred_type = "surv"` option from **aorsf**, returning estimated survival probabilities for each observation at the specified time points.

Value

A data frame where each row corresponds to an observation in `newdata` and each column corresponds to a requested prediction time (`"t=<time>"`).

Examples

```
mod <- fit_orsf(Surv(time, status) ~ age + karno, data = veteran)
pred <- predict_orsf(mod, newdata = veteran[1:5, ], times = c(100, 200, 300))
head(pred)
```

`predict_ranger` *Predict Survival Probabilities from a ranger Model*

Description

Generates predicted survival probabilities for given time points from a model fitted with `fit_ranger`.

Usage

```
predict_ranger(object, newdata, times)
```

Arguments

`object`	An `"mlsurv_model"` object returned by `fit_ranger`.
`newdata`	A `data.frame` containing the same predictors as the training data.
`times`	Numeric vector of time points at which to estimate survival probabilities.

Details

Predictions are obtained from `predict.ranger` and survival curves are interpolated to match the requested `times`.

Value

A `data.frame` with one row per observation in `newdata` and one column per time point (columns named `"t=<time>"`).

See Also

`fit_ranger`, `tune_ranger`

Examples

```
mod <- fit_ranger(
  Surv(time, status) ~ age + karno + celltype,
  data = veteran,
  num.trees = 25
)
predict_ranger(mod, newdata = veteran[1:5, ], times = c(100, 200, 300))
```

---

predict_rpart *Predict Survival Probabilities from an* rpart *Survival Tree*

Description

Predicts survival probabilities at specified time points from a fitted `rpart` survival tree model, assuming an exponential survival distribution within each terminal node.

Usage

```
predict_rpart(object, newdata, times, ...)
```

Arguments

`object`	An `"mlsurv_model"` object created by `fit_rpart()`.
`newdata`	A `data.frame` containing the predictor variables for prediction.
`times`	Numeric vector of time points at which to compute survival probabilities.
`...`	Additional arguments passed to internal prediction functions.

Details

Predictions are based on converting the predicted mean survival times from the survival tree into survival probabilities under an exponential assumption:

$$S(t) = \exp(-t/\hat{\mu})$$

where $\hat{\mu}$ is the predicted mean survival time from the terminal node.

Value

A `data.frame` with one row per observation in `newdata` and one column per requested time point, containing predicted survival probabilities.

Examples

```
mod_rpart <- fit_rpart(Surv(time, status) ~ age + karno + celltype, data = veteran)
predict_rpart(mod_rpart, newdata = veteran[1:5, ], times = c(100, 200, 300))
```

predict_rsf *Predict Survival Probabilities from an RSF Model*

Description

Generates survival probability predictions for new data using a model fitted by `fit_rsf()`.

Usage

```
predict_rsf(object, newdata, times = NULL, ...)
```

Arguments

`object`	A fitted `mlsurv_model` returned by `fit_rsf()`.
`newdata`	A `data.frame` of new observations for prediction. Must include the same predictor variables used at training time.
`times`	Optional numeric vector of time points at which to return survival probabilities. If `NULL`, predictions are returned at the model's internal time grid.
`...`	Additional arguments forwarded to `predict.rfsrc`.

Details

If `times` is provided, the function aligns predictions by selecting, for each requested time, the closest available time from the RSF prediction object's `time.interest` grid (nearest-neighbor matching).

Value

A base `data.frame` of survival probabilities with one row per observation in `newdata` and columns named `t={time}` (character), containing numeric values in [0, 1].

See Also

`fit_rsf()`, `predict.rfsrc`

Examples

```
mod_rsf <- fit_rsf(Surv(time, status) ~ age + celltype + karno,
                   data = veteran, ntree = 200)

times <- c(100, 200, 300)
pred_probs <- predict_rsf(mod_rsf, newdata = veteran[1:5, ], times = times)
print(round(pred_probs, 3))
```

predict_selectcox *Predict Survival Probabilities with a Selected Cox Model*

Description

Generates survival probabilities at specified time points using a fitted selected-predictor Cox model (`mlsurv_model`) via `pec::predictSurvProb()`.

Usage

```
predict_selectcox(object, newdata, times)
```

Arguments

`object`	A fitted `mlsurv_model` returned by `fit_selectcox()`.
`newdata`	A `data.frame` of new observations for prediction. Must include the same predictor variables used at training time.
`times`	Numeric vector of evaluation time points (same scale as the training survival time). Must be non-negative and finite.

Details

Internally calls `pec::predictSurvProb(object$model, newdata, times)` and renames columns to the standard `t=...` convention used by `survalis`.

Value

A base `data.frame` with one row per observation in `newdata` and columns named `t={time}` (character), containing numeric survival probabilities in [0, 1].

See Also

`fit_selectcox()`, `tune_selectcox()`

Examples

```
mod <- fit_selectcox(Surv(time, status) ~ age + celltype + karno, data = veteran)
predict_selectcox(mod, newdata = veteran[1:5, ], times = c(100, 200, 300))
```

predict_stpm2 *Predict Survival Probabilities with an rstpm2 Model*

Description

Generates survival probabilities at specified time points using a fitted **rstpm2** model wrapped as an `mlsurv_model`.

Usage

```
predict_stpm2(object, newdata, times, ...)
```

Arguments

`object`	A fitted `mlsurv_model` returned by `fit_stpm2()`.
`newdata`	A `data.frame` of new observations for prediction. Must include the same predictor variables used at training time.
`times`	Numeric vector of evaluation time points (same scale as the training survival time). Must be non-negative and finite.
`...`	Additional arguments forwarded to `predict()`.

Details

Internally expands `newdata` over the requested `times` and calls `predict(object$model, type = "surv", newtime = times)`; results are reshaped to a wide format with one column per time point.

Value

A base `data.frame` with one row per observation in `newdata` and columns named `t={time}` (character), containing numeric survival probabilities in [0, 1].

See Also

`fit_stpm2()`

Examples

```
mod_stpm2 <- fit_stpm2(Surv(time, status) ~ age + karno + celltype,
                       data = veteran, df = 4)
predict_stpm2(mod_stpm2, newdata = veteran[1:5, ], times = c(100, 200, 300))
```

predict_survdnn *Predict Survival Probabilities with a DNN Survival Model*

Description

Generates survival probabilities at specified time points using a fitted deep neural network survival model (`mlsurv_model`).

Usage

```
predict_survdnn(
  object,
  newdata,
  times = NULL,
  type = c("survival", "lp", "risk"),
  ...
)
```

Arguments

`object`	A fitted `mlsurv_model` returned by `fit_survdnn()`.
`newdata`	A `data.frame` of new observations for prediction. Must include the same predictor variables used at training time.
`times`	Numeric vector of evaluation time points (same scale as the survival time used at training). Required for `type = "survival"` and `type = "risk"`. Ignored for `type = "lp"`.
`type`	Prediction type. `"survival"` returns survival probabilities, `"lp"` returns the engine's linear predictor / latent score, and `"risk"` returns `1 - S(t)` at a single requested time.
`...`	Additional arguments forwarded to `predict.survdnn()`.

Details

Delegates to the installed **survdnn** prediction method. The default remains `type = "survival"` so the result stays compatible with `cv_survlearner()` and the rest of the `survalis` evaluation pipeline.

Value

- If `type = "survival"`, a base `data.frame` with one row per observation in `newdata` and columns named `t={time}` containing values in `[0, 1]`.
- If `type = "lp"` or `type = "risk"`, a numeric vector.

See Also

`fit_survdnn()`, `tune_survdnn()`

Examples

```
if (requireNamespace("survdnn", quietly = TRUE) &&
    requireNamespace("torch", quietly = TRUE) &&
    torch::torch_is_installed()) {
  mod <- fit_survdnn(Surv(time, status) ~ age + karno + celltype,
                     data = veteran, loss = "cox", epochs = 50, verbose = FALSE)

  pred <- predict_survdnn(mod, newdata = veteran[1:5, ], times = c(30, 90, 180))
  print(pred)
}
```

`predict_survmetalearner` *Predict with a Stacked Survival Meta-Learner*

Description

Produces stacked survival probabilities by combining base learner predictions via the time-specific weights learned by `fit_survmetalearner`.

Usage

```
predict_survmetalearner(model, newdata, times)
```

Arguments

`model`	A `"survmetalearner"` object returned by `fit_survmetalearner()`.
`newdata`	A data frame of new observations for prediction.
`times`	Numeric vector of evaluation times (must be a subset of the times used to train the meta-learner).

Details

For each base learner listed in `model$learners`, the corresponding `predict_<learner>` function is called to obtain $S_\ell(t \mid x)$. Stacked predictions are computed as $\hat{S}(t \mid x) = \sum_\ell w_\ell(t) S_\ell(t \mid x)$, where $w_\ell(t)$ are the learned nonnegative weights for time t.

Value

A data frame with one row per observation and one column per requested time (columns named `"t=<time>"`), containing stacked survival probabilities.

See Also

`fit_survmetalearner`, `plot_survmetalearner_weights`

Examples

```
form <- Surv(time, status) ~ age + karno + trt
times <- c(80, 160)
mod_cox <- fit_coxph(form, data = veteran)
mod_rpart <- fit_rpart(form, data = veteran)
base_models <- list(coxph = mod_cox, rpart = mod_rpart)
base_preds <- list(
  coxph = predict_coxph(mod_cox, newdata = veteran, times = times),
  rpart = predict_rpart(mod_rpart, newdata = veteran, times = times)
)
meta_model <- fit_survmetalearner(
  base_preds = base_preds,
  time = veteran$time,
  status = veteran$status,
  times = times,
  base_models = base_models,
  formula = form,
  data = veteran
)
predict_survmetalearner(meta_model, newdata = veteran[1:3, ], times = times)
```

---

predict_survsvm *Predict Survival Probabilities with Survival SVM*

Description

Generates survival probabilities at specified times from a fitted survival SVM (`mlsurv_model`). Since many SVM variants output a single predicted time (or rank), this function maps predicted times to survival curves using either an Exponential or Weibull parametric assumption.

Usage

```
predict_survsvm(object, newdata, times, dist = "exp", shape = 1)
```

Arguments

`object`	A fitted `mlsurv_model` returned by `fit_survsvm()`.
`newdata`	A `data.frame` of new observations for prediction. Must include the same predictor variables used at training time.
`times`	Numeric vector of evaluation time points (same scale as the survival time used at training). Must be non-negative and finite.
`dist`	Parametric family used to map predicted times to survival curves: `"exp"` (exponential) or `"weibull"`.
`shape`	Weibull shape parameter (used only if `dist = "weibull"`).

Details

Parametric mapping. Let $\hat{T}$ be the predicted time from the SVM model (per row). For a requested evaluation time t:

- Exponential: $S(t) = \exp(-t/\hat{T})$
- Weibull: $S(t) = \exp\{-(t/\hat{T})^{\kappa}\}$, where κ is `shape`

This mapping provides calibrated-looking survival probabilities but is a modeling assumption external to the SVM fit; verify adequacy in practice.

Value

A base `data.frame` with one row per observation in `newdata` and columns named `t={time}` (character), containing numeric values in [0, 1].

See Also

`fit_survsvm()`, `tune_survsvm()`

Examples

```
mod_svm <- fit_survsvm(Surv(time, status) ~ age + celltype + karno,
                           data = veteran,
                           type = "regression",
                           gamma.mu = 0.1,
                           kernel = "lin_kernel")

times <- c(100, 300, 500)
predict_survsvm(mod_svm, newdata = veteran[1:5, ], times = times, dist = "exp")
predict_survsvm(mod_svm, newdata = veteran[1:5, ], times = times, dist = "weibull", shape = 1.5)
```

`predict_xgboost` *Predict Survival with XGBoost*

Description

Generates predictions from an XGBoost survival `mlsurv_model`. If `times` is `NULL`, returns the raw linear predictor. If `times` is provided, returns survival probabilities computed via the AFT mapping using `object$dist` and `object$scale`.

Usage

```
predict_xgboost(object, newdata, times = NULL)
```

Arguments

object	A fitted mlsurv_model returned by fit_xgboost().
newdata	A data.frame of new observations for prediction.
times	Optional numeric vector of evaluation time points (same scale as training time). If NULL, returns the linear predictor.

Details

AFT mapping with $q = (\log t - \eta)/\sigma$:

- Normal: $S(t) = 1 - \Phi(q)$
- Extreme (Gumbel): $S(t) = \exp\{-\exp(q)\}$
- Logistic: $S(t) = 1/(1 + \exp(q))$

Note: If the model was trained with objective = "survival:cox", survival probabilities are still computed using the supplied AFT distribution/scale stored in the object; interpret with caution.

Value

- If times is NULL: a numeric vector of linear predictors (one per row of newdata).
- If times is provided: a base data.frame of survival probabilities with columns named "t=<time>" and row names "ID_<i>".

See Also

fit_xgboost(), tune_xgboost()

Examples

```
mod_xgb <- fit_xgboost(Surv(time, status) ~ age + karno + celltype,
                       data = veteran, nrounds = 20)
predict_xgboost(mod_xgb, newdata = veteran[1:5, ], times = c(100, 200, 300))
```

---

score_survmodel *Score a Fitted Survival Model on Its Training Data*

Description

Computes one or more performance metrics for a fitted mlsurv_model, predicting on the training data with the model's corresponding predict_* function.

Usage

```
score_survmodel(
  model,
  times,
  metrics = c("cindex", "ibs", "brier", "iae", "ise")
)
```

Arguments

model	An object of class "mlsurv_model".
times	Numeric vector of evaluation times. For "brier", must be a single time.
metrics	Character vector of metrics to compute. Supported: "cindex", "auc", "ibs", "brier", "iae", "ise", "ece".

Details

The function constructs the appropriate predict_* function name from model$learner, predicts survival probabilities on model$data, builds a Surv object from model$formula, and computes the metrics. If "brier" is requested with multiple times, an error is thrown.

Value

A tibble with columns metric and value.

Examples

```
fitted_model <- fit_coxph(Surv(time, status) ~ age + karno + trt, data = veteran)
score_survmodel(
  fitted_model,
  times = c(80, 160),
  metrics = c("cindex", "ibs")
)
```

---

summarise_benchmark *Summarise Benchmark Results (Mean SD with Wald CI)*

Description

Aggregates cross-validated benchmark results by learner and metric, reporting mean, standard deviation, standard error, and an approximate 95% Wald confidence interval.

Usage

```
summarise_benchmark(benchmark_results)
```

Arguments

benchmark_results
: A data frame produced by benchmark_default_survlearners(), containing at least learner, metric, and value.

Value

A tibble with columns learner, metric, mean, sd, n, se, lower, upper.

See Also

benchmark_default_survlearners(), plot_benchmark(), summarize_benchmark_results()

Examples

```
res <- tibble::tibble(
  learner = c("coxph", "coxph", "rpart", "rpart"),
  metric = c("cindex", "ibs", "cindex", "ibs"),
  value = c(0.64, 0.19, 0.60, 0.23)
)
summarise_benchmark(res)
```

---

summarize_benchmark_results

Compact Table of Mean SD by Learner and Metric

Description

Creates a wide table summarizing each learner's performance as mean sd per metric, suitable for reporting.

Usage

```
summarize_benchmark_results(results, digits = 3)
```

Arguments

results	A data frame with columns learner, metric, and value (e.g., output of benchmark_default_survlearr
digits	Integer number of decimal places in the formatted summary (default 3).

Value

A wide tibble with one row per learner and one column per metric, containing formatted strings "mean sd".

See Also

summarise_benchmark(), benchmark_default_survlearners()

Examples

```
res <- tibble::tibble(
  learner = c("coxph", "coxph", "rpart", "rpart"),
  metric = c("cindex", "ibs", "cindex", "ibs"),
  value = c(0.64, 0.19, 0.60, 0.23)
)
summarize_benchmark_results(res, digits = 2)
```

---

summary.mlsurv_model *Summarize an* mlsurv_model

Description

Produces a compact, human-readable overview of a fitted survival learner that follows the `mlsurv_model` contract (e.g., objects returned by `fit_*()` in this toolkit). The summary prints the learner id, engine, original formula, and basic data characteristics (sample size, predictor names, time range, event rate). A structured list is returned invisibly for programmatic use.

Usage

```
## S3 method for class 'mlsurv_model'
summary(object, ...)
```

Arguments

`object`	An object of class `"mlsurv_model"` produced by a `fit_*()` function.
`...`	Ignored; included for S3 signature compatibility.

Details

This method relies on the presence of the fields `learner`, `formula`, and `data` stored in the fitted object (the standard `mlsurv_model` contract). Output printing uses **cli** if available.

Value

Invisibly returns a list of class `"summary.mlsurv_model"` containing:

learner Character id of the learner (e.g., `"ranger"`, `"coxph"`).

engine Underlying package/engine used to fit the model.

formula The original survival formula.

data_summary List with `observations`, `predictors`, `time_range`, and `event_rate`.

See Also

`fit_coxph()`, other `fit_*()` learners returning `mlsurv_model` objects.

Examples

```
mod <- fit_coxph(Surv(time, status) ~ age + trt + celltype, data = veteran)
summary(mod)

s <- summary(mod)  # capture the structured result invisibly
str(s)
```

`survmat_to_chf` *Convert a survival-probability matrix (survmat) to cumulative hazard*

Description

Uses the identity $\Lambda(t \mid x) = -\log S(t \mid x)$ to compute cumulative hazards for each observation and time point.

Usage

```
survmat_to_chf(S, eps = 1e-12)
```

Arguments

`S`	A `surv_mat`: numeric matrix/data.frame of survival probabilities.
`eps`	Numeric in (0,1). Stabilizer to avoid `log(0)`. Values of `S` are clamped to $[\varepsilon, 1-\varepsilon]$ before logging. Default is `1e-12`.

Value

A numeric matrix with the same dimensions as `S`, containing cumulative hazards $\Lambda(t \mid x)$.

Examples

```
S <- data.frame(`t=1` = c(0.9, 0.8), `t=2` = c(0.7, 0.6))
survmat_to_chf(S)
```

`survmat_to_haz` *Convert a survival-probability matrix (survmat) to hazards on a time grid*

Description

Computes *individual-specific* (i.e., conditional on covariates) hazards from predicted survival curves $S(t \mid x_i)$ on a discrete time grid.

Usage

```
survmat_to_haz(S, times, eps = 1e-12, t0 = 0)
```

Arguments

`S`	A `surv_mat`: numeric matrix/data.frame of survival probabilities.
`times`	Numeric vector of time points corresponding to the columns of `S`. Must be strictly increasing and non-negative.
`eps`	Numeric in (0,1). Stabilizer to avoid `log(0)` and division by zero; survival probabilities are clamped to $[\varepsilon, 1-\varepsilon]$. Default is `1e-12`.
`t0`	Numeric scalar; left boundary for the first interval. Default is `0`. If your grid already starts at 0 and you want the first interval to be (0, t1], keep `t0 = 0`.

Details

This function returns a *piecewise-constant hazard* per interval:

$$h_{i,j} \approx \frac{\Lambda_i(t_j) - \Lambda_i(t_{j-1})}{t_j - t_{j-1}} = \frac{\log S_i(t_{j-1}) - \log S_i(t_j)}{t_j - t_{j-1}}.$$

with $S_i(0) = 1$ and $\Lambda_i(t) = -\log S_i(t)$.

Value

A numeric matrix of hazards with the same dimensions as `S`. Each column corresponds to the interval ending at `times[j]`.

Examples

```
times <- c(1, 2, 3)
S <- matrix(c(0.9, 0.8, 0.7,
              0.95,0.9,0.85), nrow = 2, byrow = TRUE)
colnames(S) <- paste0("t=", times)
H <- survmat_to_haz(S, times)
H
```

`survmat_to_quantile` *Compute a survival-time quantile from a survival-probability matrix (survmat) with grid-based approach*

Description

Returns the smallest grid time t such that $S(t) \leq 1 - p$. This is a right-continuous approximation on the provided time grid.

Usage

```
survmat_to_quantile(S, times, p = 0.5)
```

Arguments

`S`	A `surv_mat`: numeric matrix/data.frame of survival probabilities.
`times`	Numeric vector of time points corresponding to columns of `S`. Must be strictly increasing and non-negative.
`p`	Probability in (0,1). Default is `0.5`.

Details

For `p = 0.5`, this yields a grid-based **median survival time**.

Value

Numeric vector of quantile times (length = `nrow(S)`). Returns `NA` if the threshold is not crossed on the grid (e.g., survival stays above $1 - p$ over all `times`).

Examples

```
times <- c(1, 2, 3)
S <- matrix(c(0.9, 0.6, 0.4,
              0.95,0.9,0.85), nrow = 2, byrow = TRUE)
colnames(S) <- paste0("t=", times)
survmat_to_quantile(S, times, p = 0.5)  # median on the grid
```

`survmat_to_rmst` *Compute restricted mean survival time (RMST) from a survival-probability matrix (survmat)*

Description

Computes $\mathrm{RMST}(\tau) = \int_0^\tau S(t)\,dt$ for each observation, approximated via the trapezoidal rule on the provided time grid.

Usage

```
survmat_to_rmst(S, times, tau = max(times))
```

Arguments

`S`	A `surv_mat`: numeric matrix/data.frame of survival probabilities.
`times`	Numeric vector of time points corresponding to columns of `S`. Must be strictly increasing and non-negative.
`tau`	Positive numeric scalar; upper integration limit. Default is `max(times)`.

Details

If the grid does not include `0`, the function prepends $(0, S(0) = 1)$ to correctly integrate from time 0.

Value

Numeric vector of RMST values (length = `nrow(S)`).

Examples

```
times <- c(1, 2, 3)
S <- matrix(c(0.9, 0.8, 0.7,
              0.95,0.9,0.85), nrow = 2, byrow = TRUE)
colnames(S) <- paste0("t=", times)
survmat_to_rmst(S, times, tau = 3)
```

tune_bart *Tune BART Survival Hyperparameters (Cross-Validation)*

Description

Cross-validates BART survival models over a user-supplied grid and selects the best configuration by the primary metric. Optionally refits the best model on the full dataset.

Usage

```
tune_bart(
  formula,
  data,
  times,
 param_grid = expand.grid(K = c(3, 5), ntree = c(50, 100), power = c(2, 2.5), base =
    c(0.75, 0.95)),
  metrics = c("cindex", "ibs"),
  folds = 5,
  seed = 123,
  ncores = 1,
  refit_best = FALSE
)
```

Arguments

formula	A survival formula of the form `Surv(time, status) ~ predictors`.
data	A `data.frame` with variables referenced in `formula`.
times	Numeric vector of evaluation time points.
param_grid	A `data.frame` (e.g., from `expand.grid()`) of candidate hyperparameters (e.g., `K`, `ntree`, `power`, `base`).
metrics	Character vector of evaluation metrics (default `c("cindex","ibs")`). The first entry is used as the primary selection metric.
folds	Integer; number of cross-validation folds (default `5`).
seed	Integer random seed for reproducibility (default `123`).
ncores	Integer number of CPU cores passed to `cv_survlearner` for fold evaluation (default `1`).
refit_best	Logical; if `TRUE`, refits the best configuration on the full data and returns it. If `FALSE`, returns a results table.

Details

Internally calls `cv_survlearner()` with `fit_bart()`/`predict_bart()` so tuning mirrors the production prediction path.

Value

If `refit_best = FALSE`, a `data.frame` (class `"tuned_surv"`) sorted by the primary metric with one row per grid combination. If `refit_best = TRUE`, a fitted `mlsurv_model` returned by `fit_bart`.

See Also

`fit_bart`, `predict_bart`, `surv.bart`

Examples

```
grid <- expand.grid(K = c(3), ntree = c(50), power = c(2), base = c(0.75, 0.95))
res <- tune_bart(
  formula = Surv(time, status) ~ age + karno + celltype,
  data = veteran,
  param_grid = grid,
  times = c(10, 60),
  refit_best = FALSE
)
print(res)

mod_bart <- tune_bart(
  formula = Surv(time, status) ~ age + karno + celltype,
  data = veteran,
  param_grid = grid,
  times = c(10, 60),
  refit_best = TRUE
)
summary(mod_bart)
```

---

tune_blackboost *Tune blackboost Hyperparameters (Cross-Validation)*

Description

Cross-validates **mboost** Cox boosting models over a hyperparameter grid and selects the best configuration according to the primary metric. Optionally refits the best model on the full dataset.

Usage

```
tune_blackboost(
  formula,
  data,
  times,
 param_grid = expand.grid(mstop = c(50, 100, 200), nu = c(0.05, 0.1), maxdepth = c(2,
    3)),
  metrics = c("cindex", "ibs"),
```

```
  folds = 5,
  seed = 123,
  ncores = 1,
  refit_best = FALSE,
  ...
)
```

Arguments

formula	A survival formula of the form `Surv(time, status) ~ predictors`.
data	A `data.frame` containing variables referenced in `formula`.
times	Numeric vector of evaluation time points (same scale as the training time).
param_grid	A `data.frame` (e.g., from `expand.grid()`) with columns such as `mstop`, `nu`, and `maxdepth`.
metrics	Character vector of metrics to compute (e.g., `"cindex"`, `"ibs"`). The first entry is the primary selection metric.
folds	Integer; number of CV folds. Default `5`.
seed	Integer random seed for reproducibility. Default `123`.
ncores	Integer number of CPU cores passed to `cv_survlearner` for fold evaluation (default `1`).
refit_best	Logical; if `TRUE`, refits the best configuration on the full data and returns it. Default `FALSE`.
...	Additional arguments forwarded to `fit_blackboost()`.

Details

Internally calls `cv_survlearner()` with `fit_blackboost()`/`predict_blackboost()` so tuning mirrors the production path. Typical grids vary `mstop`, `nu`, and tree `maxdepth`.

Value

If `refit_best = FALSE`, a `data.frame` (class `"tuned_surv"`) of grid results with metric columns, sorted by the primary metric. If `refit_best = TRUE`, a fitted `mlsurv_model` from `fit_blackboost()` using the selected hyperparameters.

Examples

```
res_blackboost <- tune_blackboost(
  formula = Surv(time, status) ~ age + karno + celltype,
  data = veteran,
  times = c(5, 10, 40),
  param_grid = expand.grid(
    mstop = c(100, 200),
    nu = c(0.05, 0.1),
    maxdepth = c(2, 3)
  ),
  metrics = c("cindex", "ibs"),
```

```
  folds = 3
)
print(res_blackboost)

mod_blackboost_best <- tune_blackboost(
  formula = Surv(time, status) ~ age + karno + celltype,
  data = veteran,
  times = c(5, 10, 40),
  param_grid = expand.grid(
    mstop = c(100, 200),
    nu = c(0.05, 0.1),
    maxdepth = c(2, 3)
  ),
  metrics = c("cindex", "ibs"),
  folds = 3,
  refit_best = TRUE
)
summary(mod_blackboost_best)
```

---

tune_bnnsurv *Tune bnnSurvival Hyperparameters (Cross-Validation)*

Description

Cross-validates `fit_bnnsurv` over a user-supplied grid and aggregates metrics (for example, `"cindex"`, `"ibs"`). Optionally refits and returns the best configuration.

Usage

```
tune_bnnsurv(
  formula,
  data,
  times,
 param_grid = expand.grid(k = c(2, 3), num_base_learners = c(30, 50), sample_fraction =
    c(0.5, 1), stringsAsFactors = FALSE),
  metrics = c("cindex", "ibs"),
  folds = 5,
  seed = 123,
  ncores = 1,
  refit_best = FALSE
)
```

Arguments

`formula`	A survival formula of the form `Surv(time, status) ~ predictors`.
`data`	Training data frame.
`times`	Numeric vector of evaluation time points used during CV.

param_grid	A data frame (for example from expand.grid()) with columns k, num_base_learners, and sample_fraction. Defaults cover a small illustrative search.
metrics	Character vector of metric names to compute and summarize. The first entry is treated as the primary ranking metric.
folds	Integer number of cross-validation folds. Default 5.
seed	Integer random seed for reproducibility.
ncores	Integer number of CPU cores passed to cv_survlearner for fold evaluation (default 1).
refit_best	Logical; if TRUE, refits the best configuration on the full data and returns that model. Otherwise returns the results table.

Details

Evaluation is performed by cv_survlearner() using fit_bnnsurv() and predict_bnnsurv() to match production code paths. Results are ordered by the first entry in metrics.

Value

If refit_best = FALSE, a tibble with one row per configuration containing metrics and the tuning parameters, with a failed flag for combinations that errored during CV. If refit_best = TRUE, an mlsurv_model refit at the selected hyperparameters.

See Also

fit_bnnsurv(), predict_bnnsurv(), cv_survlearner()

Examples

```
grid <- expand.grid(
  k = c(2, 3),
  num_base_learners = c(5, 10),
  sample_fraction = c(0.5, 1),
  stringsAsFactors = FALSE
)

res <- tune_bnnsurv(
  formula = Surv(time, status) ~ age + karno + diagtime + prior,
  data = veteran,
  times = c(100, 200),
  param_grid = grid,
  refit_best = FALSE
)
res

best <- tune_bnnsurv(
  formula = Surv(time, status) ~ age + karno + diagtime + prior,
  data = veteran,
  times = c(100, 200),
  param_grid = grid,
```

```
  refit_best = TRUE
)
head(predict_bnnsurv(best, newdata = veteran[1:5, ], times = c(50, 100)))
```

`tune_cforest` *Tune a Conditional Inference Survival Forest*

Description

Performs cross-validated hyperparameter tuning for a conditional inference survival forest using the `fit_cforest()` and `predict_cforest()` functions.

Usage

```
tune_cforest(
  formula,
  data,
  times,
  param_grid = expand.grid(ntree = c(100, 300), mtry = c(2, 3), mincriterion = c(0,
    0.95), fraction = c(0.5, 0.632)),
  metrics = c("cindex", "ibs"),
  folds = 5,
  seed = 123,
  ncores = 1,
  refit_best = FALSE,
  ...
)
```

Arguments

`formula`	A `Surv()` survival formula specifying the outcome and predictors.
`data`	A data frame containing the variables in the model.
`times`	A numeric vector of time points at which to evaluate performance.
`param_grid`	A data frame or list specifying the grid of hyperparameter values to evaluate. Columns should include `ntree`, `mtry`, `mincriterion`, and `fraction`.
`metrics`	A character vector of performance metrics to compute (default = `c("cindex", "ibs")`).
`folds`	Integer, number of cross-validation folds (default = 5).
`seed`	Integer, random seed for reproducibility.
`ncores`	Integer number of CPU cores passed to `cv_survlearner` for fold evaluation (default 1).
`refit_best`	Logical, if `TRUE` refits a model with the best-performing hyperparameters and returns it; otherwise returns a summary table of results.
`...`	Additional arguments passed to `fit_cforest()`.

Details

Cross-validation is performed using `cv_survlearner()` and results are sorted in descending order of the first metric specified in `metrics`.

Value

If `refit_best = FALSE`, a data frame of mean cross-validation scores for each hyperparameter combination (class `"tuned_surv"`). If `refit_best = TRUE`, an `mlsurv_model` object fitted with the optimal hyperparameters.

Examples

```
grid <- expand.grid(
  ntree = c(50, 100),
  mtry = c(2, 4),
  mincriterion = c(0, 0.95),
  fraction = c(0.632)
)
res <- tune_cforest(Surv(time, status) ~ age + celltype + karno,
  data = veteran,
  times = c(100, 200),
  param_grid = grid,
  folds = 3
)
print(res)
```

---

tune_flexsurvreg *Tune Parametric Survival Models with flexsurvreg*

Description

Performs hyperparameter tuning for `fit_flexsurvreg` over a grid of parametric distributions using cross-validation. Returns either a summary table of performance metrics or a refitted best model.

Usage

```
tune_flexsurvreg(
  formula,
  data,
  times,
  param_grid = c("weibull", "exponential", "lognormal"),
  metrics = c("cindex", "ibs"),
  folds = 5,
  seed = 123,
  ncores = 1,
  refit_best = FALSE,
  ...
)
```

Arguments

formula	A Surv-based survival formula.
data	A data frame containing the variables in the model.
times	Numeric vector of time points at which to evaluate performance.
param_grid	Character vector of distributions to test (default = c("weibull", "exponential", "lognormal")).
metrics	Character vector of performance metrics to compute (default = c("cindex", "ibs")).
folds	Integer, number of cross-validation folds (default = 5).
seed	Random seed for reproducibility.
ncores	Integer number of CPU cores passed to cv_survlearner for fold evaluation (default 1).
refit_best	Logical; if TRUE, refits the best-performing model on the full dataset and returns it. If FALSE, returns the tuning results.
...	Additional arguments passed to fit_flexsurvreg.

Value

If refit_best = FALSE, returns a data frame of tuning results with one row per distribution and columns for each metric. If refit_best = TRUE, returns the best mlsurv_model object.

Examples

```
res <- tune_flexsurvreg(Surv(time, status) ~ age + karno + celltype,
                        data = veteran,
                        param_grid = c("weibull", "exponential", "lognormal"),
                        times = c(100, 200, 300))
print(res)

best_mod <- tune_flexsurvreg(Surv(time, status) ~ age + karno + celltype,
                             data = veteran,
                             param_grid = c("weibull", "exponential", "lognormal"),
                             times = c(100, 200, 300),
                             refit_best = TRUE)
summary(best_mod)
```

---

tune_glmnet *Tune Penalized Cox Proportional Hazards Model via Cross-Validation*

Description

Performs hyperparameter tuning for penalized Cox models (**glmnet**) over a grid of alpha values using cross-validation on one or more metrics. Optionally refits the best model on the full dataset.

Usage

```
tune_glmnet(
  formula,
  data,
  times,
  param_grid = c(alpha = seq(0, 1, by = 0.25)),
  metrics = c("cindex", "ibs"),
  folds = 5,
  seed = 123,
  ncores = 1,
  refit_best = FALSE,
  ...
)
```

Arguments

formula	A survival formula of the form Surv(time, status) ~ predictors.
data	A data.frame containing all variables referenced in formula.
times	Numeric vector of time points at which to evaluate performance.
param_grid	A data.frame or list specifying candidate alpha values. Typically created with expand.grid.
metrics	Character vector of performance metrics to compute. The first entry is used as the primary selection metric.
folds	Integer; number of cross-validation folds. Default is 5.
seed	Integer random seed for reproducibility. Default is 123.
ncores	Integer number of CPU cores passed to cv_survlearner for fold evaluation (default 1).
refit_best	Logical; if TRUE, fits the best configuration on the full dataset and returns the fitted model. If FALSE (default), returns a results table.
...	Additional arguments passed to fit_glmnet.

Value

If refit_best = FALSE, returns a data.frame (class "tuned_surv") with hyperparameters and metric values for each grid combination, sorted by the primary metric. If refit_best = TRUE, returns a fitted "mlsurv_model" from fit_glmnet.

See Also

fit_glmnet, predict_glmnet

Examples

```
param_grid <- expand.grid(alpha = seq(0, 1, by = 0.25))

res_glmnet <- tune_glmnet(
```

```
  formula = Surv(time, status) ~ age + karno + celltype,
  data = veteran,
  times = c(100, 200, 300),
  param_grid = param_grid,
  metrics = c("cindex", "ibs"),
  folds = 3
)
print(res_glmnet)

mod_glmnet_best <- tune_glmnet(
  formula = Surv(time, status) ~ age + karno + celltype,
  data = veteran,
  times = c(100, 200, 300),
  param_grid = param_grid,
  refit_best = TRUE
)

summary(mod_glmnet_best)
```

---

tune_orsf *Tune Oblique Random Survival Forests (ORSF) via Cross-Validation*

Description

Performs grid-search hyperparameter tuning for **aorsf** models using cross-validation and selects the best configuration by the primary metric. Optionally refits the best model on the full dataset.

Usage

```
tune_orsf(
  formula,
  data,
  times,
 param_grid = expand.grid(n_tree = c(100, 300), mtry = c(2, 3), min_events = c(5, 10)),
  metrics = c("cindex", "ibs"),
  folds = 5,
  seed = 123,
  ncores = 1,
  refit_best = FALSE,
  ...
)
```

Arguments

formula	A survival formula of the form `Surv(time, status) ~ predictors`.
data	A `data.frame` containing all variables referenced in `formula`.

times	Numeric vector of evaluation time points (same scale as the training survival time). Must be positive and finite.
param_grid	A data.frame (e.g., from expand.grid()) of candidate hyperparameters. Typical columns include: • n_tree: number of trees (e.g., c(100, 300)) • mtry: number of variables tried at each split • min_events: minimum number of events required to attempt a split
metrics	Character vector of metrics to evaluate/optimize (e.g., "cindex", "ibs"). The first entry is used as the primary selection metric.
folds	Integer; number of cross-validation folds. Default is 5.
seed	Integer random seed for reproducibility. Default is 123.
ncores	Integer number of CPU cores passed to cv_survlearner for fold evaluation (default 1).
refit_best	Logical; if TRUE, refits the best configuration on the full data and returns it. Default is FALSE.
...	Additional arguments forwarded to the underlying engine via fit_orsf.

Details

Internally calls cv_survlearner() with fit_orsf / predict_orsf so tuning uses the exact same code paths as production. The min_events column of param_grid is passed to the engine as n_split (minimum events for a candidate split).

Value

If refit_best = FALSE, a data.frame (class "tuned_surv") with one row per grid combination and columns for hyperparameters and metrics, ordered by the first metric. If refit_best = TRUE, a fitted mlsurv_model from fit_orsf using the best settings.

See Also

fit_orsf, predict_orsf, **aorsf**

Examples

```
res_orsf <- tune_orsf(
  formula    = Surv(time, status) ~ age + karno,
  data       = veteran,
  times      = c(100, 200, 300),
  param_grid = expand.grid(
    n_tree     = c(100, 200),
    mtry       = c(1, 2),
    min_events = c(5, 10)
  ),
  metrics    = c("cindex", "ibs"),
  folds      = 2
)
```

```
print(res_orsf)

mod_orsf_best <- tune_orsf(
  formula    = Surv(time, status) ~ age + karno,
  data       = veteran,
  times      = c(100, 200, 300),
  param_grid = expand.grid(
    n_tree     = c(100, 200),
    mtry       = c(1, 2),
    min_events = c(5, 10)
  ),
  metrics    = c("cindex", "ibs"),
  folds      = 2,
  refit_best = TRUE
)

summary(mod_orsf_best)
```

---

tune_ranger *Hyperparameter Tuning for ranger Survival Models*

Description

Performs grid search tuning of a survival random forest model using **ranger** over a set of hyperparameter combinations.

Usage

```
tune_ranger(
  formula,
  data,
  times,
 param_grid = expand.grid(num.trees = c(100, 300), mtry = c(1, 2, 3), min.node.size =
    c(3, 5)),
  metrics = c("cindex", "ibs"),
  folds = 5,
  seed = 123,
  ncores = 1,
  refit_best = FALSE,
  ...
)
```

Arguments

formula	A survival formula of the form `Surv(time, status) ~ predictors`.
data	A `data.frame` containing the variables in the model.

times	Numeric vector of time points at which to evaluate performance.
param_grid	A data.frame of hyperparameter combinations. Must contain columns num.trees, mtry, and min.node.size.
metrics	Character vector of evaluation metrics (e.g., "cindex", "ibs").
folds	Number of cross-validation folds.
seed	Random seed for reproducibility.
ncores	Integer number of CPU cores passed to cv_survlearner for fold evaluation (default 1).
refit_best	Logical; if TRUE, refits the model using the best parameters.
...	Additional arguments passed to fit_ranger.

Details

Uses cv_survlearner to perform cross-validation for each parameter combination. If refit_best = TRUE, the function returns the best-fitting model; otherwise, it returns a tuning results table.

Value

If refit_best = FALSE, returns a data.frame of tuning results sorted by the primary metric. If refit_best = TRUE, returns an "mlsurv_model" object with the best parameters and an attribute "tuning_results".

See Also

fit_ranger, predict_ranger

Examples

```
mod_ranger_best <- tune_ranger(
  formula = Surv(time, status) ~ age + karno + celltype,
  data = veteran,
  times = c(80, 160),
  param_grid = expand.grid(
    num.trees = c(25),
    mtry = c(2),
    min.node.size = c(5)
  ),
  metrics = c("cindex", "ibs"),
  folds = 2,
  refit_best = TRUE
)
summary(mod_ranger_best)
```

tune_rpart *Tune a Survival Tree Model (*rpart*) via Cross-Validation*

Description

Performs hyperparameter tuning for the rpart survival tree model using cross-validation, returning either a table of results or the best fitted model.

Usage

```
tune_rpart(
  formula,
  data,
  times,
 param_grid = expand.grid(minsplit = c(10, 20), cp = c(0.001, 0.01), maxdepth = c(10,
    30)),
  metrics = c("cindex", "ibs"),
  folds = 5,
  seed = 123,
  ncores = 1,
  refit_best = TRUE,
  ...
)
```

Arguments

formula	A survival formula of the form Surv(time, status) ~ predictors.
data	A data.frame containing the variables in the model.
times	Numeric vector of time points for evaluation.
param_grid	A data.frame or expand.grid() result specifying hyperparameter combinations. Must include minsplit, cp, and maxdepth.
metrics	Character vector of evaluation metrics to compute (e.g., "cindex", "ibs").
folds	Number of cross-validation folds. Default is 5.
seed	Random seed for reproducibility.
ncores	Integer number of CPU cores passed to cv_survlearner for fold evaluation (default 1).
refit_best	Logical; if TRUE, the best hyperparameter combination is refitted on the full dataset.
...	Additional arguments passed to fit_rpart().

Value

If refit_best = FALSE, returns a tibble summarizing mean CV performance for each parameter combination. If refit_best = TRUE, returns an "mlsurv_model" object for the best parameters, with a "tuning_results" attribute.

Examples

```
res_rpart <- tune_rpart(
  formula = Surv(time, status) ~ age + karno + celltype,
  data = veteran,
  times = c(100, 200, 300),
  param_grid = expand.grid(
    minsplit = c(10, 20),
    cp = c(0.001, 0.01),
    maxdepth = c(10, 30)
  ),
  metrics = c("cindex", "ibs"),
  folds = 3,
  seed = 42,
  refit_best = FALSE
)

print(res_rpart)

mod_rpart_best <- tune_rpart(
  formula = Surv(time, status) ~ age + karno + celltype,
  data = veteran,
  times = c(100, 200, 300),
  param_grid = expand.grid(
    minsplit = c(10, 20),
    cp = c(0.001, 0.01),
    maxdepth = c(10, 30)
  ),
  metrics = c("cindex", "ibs"),
  folds = 3,
  seed = 42,
  refit_best = TRUE
)
```

---

tune_rsf *Tune Random Survival Forest Hyperparameters (Cross-Validation)*

Description

Cross-validates RSF models over a specified parameter grid and selects the best configuration according to the primary metric. Optionally refits the best model on the full dataset.

Usage

```
tune_rsf(
  formula,
  data,
  times,
 param_grid = expand.grid(ntree = c(200, 500), mtry = c(1, 2, 3), nodesize = c(5, 15)),
```

```
  metrics = c("cindex", "ibs"),
  folds = 5,
  seed = 123,
  ncores = 1,
  refit_best = FALSE,
  ...
)
```

Arguments

formula	A survival formula of the form `Surv(time, status) ~ predictors`.
data	A `data.frame` containing all variables referenced in `formula`.
times	Numeric vector of evaluation time points (same scale as the survival time used at training). Must be non-negative and finite.
param_grid	A data frame (e.g., from `expand.grid()`) containing columns for RSF hyperparameters such as `ntree`, `mtry`, and `nodesize`.
metrics	Character vector of metrics to evaluate/optimize (e.g., `"cindex"`, `"ibs"`). The first entry is the primary selection metric.
folds	Integer; number of cross-validation folds.
seed	Integer random seed for reproducibility.
ncores	Integer number of CPU cores passed to `cv_survlearner` for fold evaluation (default 1).
refit_best	Logical; if `TRUE`, refits the best configuration on the full data and returns it, with tuning results attached.
...	Additional arguments forwarded to the underlying engine where applicable.

Details

Internally calls `cv_survlearner()` with `fit_rsf()`/`predict_rsf()` so tuning uses the same code paths as production. Typical grids vary `ntree`, `mtry`, and `nodesize`.

Value

If `refit_best = FALSE`, a `data.frame` (class `"tuned_surv"`) of grid results with metric columns and hyperparameters, ordered by the first metric. If `refit_best = TRUE`, a fitted `mlsurv_model` from `fit_rsf()` with attribute `"tuning_results"` containing the full grid results.

See Also

`fit_rsf()`, `predict_rsf()`

Examples

```
mod_rsf_best <- tune_rsf(
  formula = Surv(time, status) ~ age + celltype + karno,
  data = veteran,
```

```
  times = c(100, 200, 300),
  param_grid = expand.grid(
    ntree = c(200, 500),
    mtry = c(1, 2, 3),
    nodesize = c(5, 15)
  ),
  metrics = c("cindex", "ibs"),
  folds = 3,
  refit_best = TRUE
)

summary(mod_rsf_best)
```

`tune_selectcox` *Tune SelectCox Rule (Cross-Validation)*

Description

Cross-validates **pec**'s `selectCox()` across one or more selection rules and selects the best configuration by the primary metric. Optionally refits the best rule on the full dataset.

Usage

```
tune_selectcox(
  formula,
  data,
  times,
  rules = c("aic", "p"),
  metrics = c("cindex", "ibs"),
  folds = 5,
  seed = 123,
  ncores = 1,
  refit_best = TRUE,
  ...
)
```

Arguments

`formula`	A survival formula of the form `Surv(time, status) ~ predictors`. The left-hand side must be a `Surv()` object from the **survival** package.
`data`	A `data.frame` containing all variables referenced in `formula`.
`times`	Numeric vector of evaluation time points (same scale as the training survival time). Must be non-negative and finite.
`rules`	Character vector of selection rules to compare (e.g., `c("aic","p")`).

metrics	Character vector of metrics to evaluate/optimize (e.g., "cindex", "ibs", "ise"). The first entry is the primary selection metric.
folds	Number of cross-validation folds.
seed	Integer random seed for reproducibility.
ncores	Integer number of CPU cores passed to cv_survlearner for fold evaluation (default 1).
refit_best	Logical; if TRUE, refits the best rule on the full data and returns it as an mlsurv_model (with tuning results attached).
...	Additional arguments forwarded to the underlying engine where applicable.

Details

Evaluation. Internally calls cv_survlearner() with fit_selectcox()/predict_selectcox() to ensure tuning uses the same code paths as production. Rules typically include "aic" and/or "p".

Value

If refit_best = FALSE, a data.frame (class "tuned_surv") with a row per rule and metric columns, ordered by the first metric. If refit_best = TRUE, a fitted mlsurv_model from fit_selectcox() with attribute "tuning_results" containing the full results.

See Also

fit_selectcox(), predict_selectcox(), pec::selectCox()

Examples

```
res_selectcox <- tune_selectcox(
  formula = Surv(time, status) ~ age + karno + celltype,
  data = veteran,
  times = c(100, 200, 300),
  rules = c("aic", "p"),
  metrics = c("cindex", "ibs", "ise"),
  folds = 3,
  refit_best = FALSE
)

print(res_selectcox)
class(res_selectcox)

mod_selectcox <- tune_selectcox(
  formula = Surv(time, status) ~ age + karno + celltype,
  data = veteran,
  times = c(100, 200, 300),
  rules = c("aic", "p"),
  metrics = c("cindex", "ibs", "ise"),
  folds = 3,
  refit_best = TRUE
)
```

```
summary(mod_selectcox)
```

---

tune_survdnn *Tune Deep Neural Network Survival Models (Cross-Validation)*

Description

Cross-validates **survdnn**-based models over a user-specified grid and selects the best configuration by the primary metric. Optionally refits the best model on the full dataset.

Usage

```
tune_survdnn(
  formula,
  data,
  times,
  param_grid = list(hidden = list(c(32, 16), c(64, 32, 16)), lr = c(0.001, 5e-04),
    activation = c("relu", "gelu"), epochs = c(100, 200), loss = c("cox", "aft"),
     optimizer = "adam", dropout = c(0.1, 0.3), batch_norm = c(TRUE)),
  metrics = c("cindex", "ibs"),
  folds = 3,
  seed = 42,
  ncores = 1,
  refit_best = FALSE,
  ...
)
```

Arguments

formula	A survival formula of the form `Surv(time, status) ~ predictors`. The left-hand side must be a `Surv()` object from the `survival` package.
data	A `data.frame` containing all variables referenced in `formula`.
times	Numeric vector of evaluation time points (same scale as the survival time used at training). Must be non-negative and finite.
param_grid	A named list of candidate hyperparameters. Typical entries: `hidden` (list of integer vectors), `lr` (numeric), `activation` (character), `epochs` (integer), `loss` (character), `optimizer` (character), `dropout` (numeric), and `batch_norm` (logical).
metrics	Character vector of metrics to evaluate/optimize (e.g., `"cindex"`, `"ibs"`). The first entry is used as the primary selection metric.
folds	Number of cross-validation folds.
seed	Integer random seed for reproducibility.
ncores	Integer number of CPU cores passed to `cv_survlearner` for fold evaluation (default 1).

refit_best	Logical; if TRUE, refits the best configuration on the full data and returns it as the result (with tuning attributes).
...	Additional arguments forwarded to the underlying engine where applicable.

Details

Evaluation. Internally calls cv_survlearner() with fit_survdnn()/predict_survdnn() so tuning uses the same code paths as production. Hyperparameters are combined via tidyr::crossing() and each row is passed through to fit_survdnn(), so the grid can include any supported engine argument exposed by this wrapper.

Value

If refit_best = FALSE, a data.frame (class "tuned_surv") with hyperparameter settings and metric columns, ordered by the first metric. If refit_best = TRUE, a fitted mlsurv_model from fit_survdnn() with attribute "tuning_results" containing the full grid results.

See Also

fit_survdnn(), predict_survdnn()

Examples

```
if (requireNamespace("survdnn", quietly = TRUE) &&
    requireNamespace("torch", quietly = TRUE) &&
    torch::torch_is_installed()) {
  grid <- list(
    hidden = list(c(16), c(32, 16)),
    lr = c(1e-4, 5e-4),
    activation = c("relu", "tanh"),
    epochs = c(300),
    loss = c("cox", "coxtime"),
    optimizer = "adam",
    dropout = c(0.1, 0.3)
  )

  mod <- tune_survdnn(
    formula = Surv(time, status) ~ age + karno + celltype,
    data = veteran,
    times = c(90),
    metrics = c("cindex", "ibs"),
    param_grid = grid,
    refit_best = TRUE
  )

  summary(mod)
}
```

tune_survsvm *Tune Survival SVM Hyperparameters (Cross-Validation)*

Description

Cross-validates Survival SVM models over a user-specified grid and selects the best configuration based on the chosen metric. Optionally refits the best model on the full dataset.

Usage

```
tune_survsvm(
  formula,
  data,
  times,
  metrics = "cindex",
  param_grid,
  folds = 5,
  seed = 42,
  ncores = 1,
  refit_best = FALSE,
  dist = "exp",
  shape = 1
)
```

Arguments

formula	A survival formula of the form `Surv(time, status) ~ predictors`. The left-hand side must be a `Surv()` object from the `survival` package.
data	A `data.frame` containing all variables referenced in `formula`.
times	Numeric vector of evaluation time points (same scale as the survival time used at training). Must be non-negative and finite.
metrics	Character vector of metrics to evaluate/optimize (e.g., `"cindex"`, `"ibs"`). The first entry is the primary selection metric.
param_grid	A named list of candidate hyperparameters; typical entries include `gamma.mu` and `kernel`. Use `list(gamma.mu = c(...), kernel = c(...))`.
folds	Number of cross-validation folds.
seed	Integer random seed for reproducibility.
ncores	Integer number of CPU cores passed to `cv_survlearner` for fold evaluation (default 1).
refit_best	Logical; if `TRUE`, refits the best configuration on the full data and returns it as the result (with tuning attributes).
dist	Parametric mapping for predictions during CV: `"exp"` or `"weibull"`.
shape	Weibull shape parameter if `dist = "weibull"`.

Details

Evaluation. Internally calls `cv_survlearner()` with `fit_survsvm()`/`predict_survsvm()` to keep code paths consistent with production usage. The prediction step applies the specified `dist`/shape mapping to convert predicted times into survival probabilities at `times`.

Value

If `refit_best = FALSE`, a `data.frame` (class `"tune_surv"`) of grid results with metric columns and tuning parameters. If `refit_best = TRUE`, a fitted `mlsurv_model` (class augmented with `"tune_surv"`) with the full results attached in `attr(, "tuning_results")`.

See Also

`fit_survsvm()`, `predict_survsvm()`

Examples

```
grid <- list(
  gamma.mu = c(0.01, 0.1),
  kernel = c("lin_kernel", "add_kernel")
)

res_svm <- tune_survsvm(
  formula = Surv(time, status) ~ age + celltype + karno,
  data = veteran,
  times = c(100, 300, 500),
  metrics = c("cindex", "ibs"),
  param_grid = grid,
  folds = 3,
  refit_best = TRUE
)

summary(res_svm)

res_svm <- tune_survsvm(
  formula = Surv(time, status) ~ age + celltype + karno,
  data = veteran,
  times = c(100, 300, 500),
  metrics = c("cindex", "ibs"),
  param_grid = grid,
  folds = 3,
  refit_best = FALSE
)

res_svm
```

tune_xgboost *Tune XGBoost Survival Hyperparameters (Cross-Validation)*

Description

Cross-validates XGBoost survival models over a user-specified grid and returns a results table with metric summaries per configuration. Any row that errors during CV is marked `failed = TRUE`.

Usage

```
tune_xgboost(
  formula,
  data,
  times,
 param_grid = expand.grid(nrounds = c(50, 100), max_depth = c(3, 6), eta = c(0.01, 0.1),
   aft_loss_distribution = c("extreme", "logistic"), aft_loss_distribution_scale =
     c(0.5, 1), objective = "survival:aft", stringsAsFactors = FALSE),
  metrics = c("cindex", "ibs"),
  folds = 5,
  seed = 123,
  ncores = 1
)
```

Arguments

`formula`	A survival formula of the form `Surv(time, status) ~ predictors`.
`data`	A `data.frame` containing variables referenced in `formula`.
`times`	Numeric vector of evaluation time points.
`param_grid`	A `data.frame` (e.g., from `expand.grid()`) with columns: `nrounds`, `max_depth`, `eta`, `aft_loss_distribution`, `aft_loss_distribution_scale`, and `objective`. Values are passed through to `fit_xgboost` and ultimately to **xgboost**.
`metrics`	Character vector of metrics to evaluate (e.g., `"cindex"`, `"ibs"`). The first entry is treated as the primary selection metric for ordering.
`folds`	Integer number of CV folds.
`seed`	Integer random seed for reproducibility.
`ncores`	Integer number of CPU cores passed to `cv_survlearner` for fold evaluation (default 1).

Details

Internally calls `cv_survlearner` with `fit_xgboost` / `predict_xgboost`. Any configuration that errors (e.g., due to invalid parameters or data issues) is recorded with `failed = TRUE` and omitted from metric summarization.

Value

A `tibble` with one row per grid configuration, containing:

nrounds, max_depth, eta, aft_loss_distribution, aft_loss_distribution_scale, objective The grid values.

failed Logical; `TRUE` if the configuration errored.

metric columns One column per entry in `metrics` (when available).

The table is arranged by the first metric in `metrics` (ascending, as implemented).

See Also

`fit_xgboost`, `predict_xgboost`

Examples

```
grid <- expand.grid(
  nrounds = c(20, 40),
  max_depth = c(2, 3),
  eta = c(0.1, 0.2),
  aft_loss_distribution = c("extreme", "logistic"),
  aft_loss_distribution_scale = c(0.5, 1),
  objective = "survival:aft",
  stringsAsFactors = FALSE
)

res_xgb <- tune_xgboost(
  formula = survival::Surv(time, status) ~ age + karno + celltype,
  data    = survival::veteran,
  times   = c(100, 200),
  metrics = c("cindex", "ibs"),
  param_grid = grid,
  folds   = 2,
  seed    = 123
)
head(res_xgb)
```

---

veteran *Veteran's Administration Lung Cancer Trial Data*

Description

This is the `veteran` dataset originally from the **survival** package, containing data from a randomized trial of lung cancer treatments.

Usage

```
veteran
```

Format

A data frame with 137 observations and 8 variables:

trt Treatment: 1=standard, 2=test

celltype Cell type: squamous, smallcell, adeno, large

time Survival time in days

status Censoring status: 1=dead, 0=alive

karno Karnofsky performance score (higher = better)

diagtime Months from diagnosis to randomization

age Age in years

prior Prior therapy: 0=no, 10=yes

Source

survival package, originally from Kalbfleisch and Prentice (1980) *The Statistical Analysis of Failure Time Data*.

Examples

```
head(veteran)
summary(veteran$time)
```

Index