Package 'tidyclust'

June 28, 2026

Title A Common API to Clustering

Version 0.3.2

Description A common interface to specifying clustering models, in the same style as 'parsnip'. Creates unified interface across different functions and computational engines.

License MIT + file LICENSE

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

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

Depends R (>= 4.1)

Imports cli (>= 3.0.0), lifecycle, dials (>= 1.3.0), dplyr (>= 1.0.9), flexclust (>= 1.3-6), generics (>= 0.1.2), glue (>= 1.6.2), hardhat (>= 1.0.0), mclust, modelenv (>= 0.2.0), parsnip (>= 1.0.2), philentropy (>= 0.9.0), prettyunits (>= 1.1.0), purrr (>= 1.0.0), rlang (>= 1.0.6), rsample (>= 1.0.0), stats, tibble (>= 3.1.0), tidyr (>= 1.2.0), tune (>= 2.1.0), utils, vctrs (>= 0.5.0)

Suggests butcher, cluster, ClusterR, clustMixType (>= 0.3-5), covr, dbscan, future, future.apply, klaR, knitr, LPCM, meanShiftR, mirai (>= 1.0.0), modeldata (>= 1.0.0), RcppHungarian, recipes (>= 1.0.0), rmarkdown, testthat (>= 3.0.0), withr, workflows (>= 1.1.2)

Config/Needs/website pkgdown, tidymodels, tidyverse, palmerpenguins, patchwork, ggforce, tidyverse/tidytemplate, mvtnorm

Config/testthat/edition 3

Config/usethis/last-upkeep 2025-04-24

Encoding UTF-8

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Emil Hvitfeldt [aut, cre] (ORCID:
<https://orcid.org/0000-0002-0679-1945>),
Kelly Bodwin [aut],
Posit Software, PBC [cph, fnd] (ROR: <https://ror.org/03wc8by49>)

Maintainer Emil Hvitfeldt <emil.hvitfeldt@posit.co>

Repository CRAN

Date/Publication 2026-06-28 21:30:02 UTC

Contents

`augment.cluster_fit` *Augment data with predictions*

Description

`augment()` will add column(s) for predictions to the given data.

Usage

```
## S3 method for class 'cluster_fit'
augment(x, new_data, ...)
```

Arguments

`x`	A `cluster_fit` object produced by `fit.cluster_spec()` or `fit_xy.cluster_spec()`.
`new_data`	A data frame or matrix.
`...`	Not currently used.

Details

For partition models, a `.pred_cluster` column is added.

Preprocessing with workflows:

When `x` is a fitted `workflows::workflow()` that includes a recipe, the recipe transformations are applied to `new_data` before predicting. The returned tibble contains the **original** (untransformed) `new_data` plus the `.pred_cluster` column, so the data is not altered by preprocessing.

Value

A tibble containing `new_data` with a `.pred_cluster` column appended giving the cluster assignment for each row.

Examples

```
kmeans_spec <- k_means(num_clusters = 5) |>
  set_engine("stats")

kmeans_fit <- fit(kmeans_spec, ~., mtcars)

kmeans_fit |>
  augment(new_data = mtcars)

# With a workflow that includes a recipe
library(recipes)
library(workflows)

rec <- recipe(~., data = mtcars) |>
  step_normalize(all_predictors())

wf_fit <- workflow() |>
  add_recipe(rec) |>
  add_model(kmeans_spec) |>
  fit(data = mtcars)

# Returns original (untransformed) data with .pred_cluster appended
augment(wf_fit, new_data = mtcars)
```

---

axe-cluster_fit *Axing a cluster_fit.*

Description

cluster_fit objects are created from the tidyclust package.

Usage

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

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

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

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

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

Arguments

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

Value

Axed cluster_fit object.

Examples

```
k_fit <- k_means(num_clusters = 3) |>
  parsnip::set_engine("stats") |>
  fit(~., data = mtcars)

butcher::butcher(k_fit)
```

`bandwidth` *Bandwidth*

Description

The kernel bandwidth used by mean shift to estimate the local density gradient. Smaller values yield more clusters, while larger values merge them.

Usage

```
bandwidth(range = c(0.01, 1), trans = NULL)
```

Arguments

`range`	A two-element vector holding the *defaults* for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the *transformed units*.
`trans`	A `trans` object from the `scales` package, such as `scales::transform_log10()` or `scales::transform_reciprocal()`. If not provided, the default is used which matches the units used in `range`. If no transformation, `NULL`.

Details

Used in `tidyclust::mean_shift()` models. The scale on which the bandwidth is interpreted depends on the engine, since some engines rescale predictors internally before applying the kernel.

Value

A `dials` parameter object for use with `tune::tune_grid()` and related functions.

Examples

```
bandwidth()
```

cluster_fit *Model Fit Object Information*

Description

An object with class "cluster_fit" is a container for information about a model that has been fit to the data.

Details

The following model types are implemented in tidyclust:

- K-Means in `k_means()`
- Hierarchical (Agglomerative) Clustering in `hier_clust()`

The main elements of the object are:

- `spec`: A `cluster_spec` object.
- `fit`: The object produced by the fitting function.
- `preproc`: This contains any data-specific information required to process new a sample point for prediction. For example, if the underlying model function requires arguments `x` and the user passed a formula to `fit`, the `preproc` object would contain items such as the terms object and so on. When no information is required, this is `NA`.

As discussed in the documentation for `cluster_spec`, the original arguments to the specification are saved as quosures. These are evaluated for the `cluster_fit` object prior to fitting. If the resulting model object prints its call, any user-defined options are shown in the call preceded by a tilde (see the example below). This is a result of the use of quosures in the specification.

This class and structure is the basis for how **tidyclust** stores model objects after seeing the data and applying a model.

cluster_metric_set *Combine metric functions*

Description

`cluster_metric_set()` allows you to combine multiple metric functions together into a new function that calculates all of them at once.

Usage

```
cluster_metric_set(...)
```

Arguments

`...`	The bare names of the functions to be included in the metric set. These functions must be cluster metrics such as `sse_total()`, `sse_ratio()`, or `silhouette_avg()`.

Details

All functions must be cluster metrics. To include a metric that wraps a built-in metric with custom arguments, such as `silhouette_avg()` with a non-default `dist_fun`, first wrap it with `new_cluster_metric()` so that it carries the `cluster_metric` class. See the examples in `new_cluster_metric()`.

Value

A `cluster_metric_set()` object, combining the use of all input metrics.

See Also

`new_cluster_metric()`

`cluster_spec` *Model Specification Information*

Description

An object with class "cluster_spec" is a container for information about a model that will be fit.

Details

The following model types are implemented in tidyclust:

- K-Means in `k_means()`
- Hierarchical (Agglomerative) Clustering in `hier_clust()`

The main elements of the object are:

- `args`: A vector of the main arguments for the model. The names of these arguments may be different from their counterparts n the underlying model function. For example, for a `k_means()` model, the argument name for the number of clusters are called "num_clusters" instead of "k" to make it more general and usable across different types of models (and to not be specific to a particular model function). The elements of `args` can `tune()` with the use in `tune_cluster()`.

For more information see https://www.tidymodels.org/start/tuning/. If left to their defaults (`NULL`), the arguments will use the underlying model functions default value. As discussed below, the arguments in `args` are captured as quosures and are not immediately executed.

- `...`: Optional model-function-specific parameters. As with `args`, these will be quosures and can be `tune()`.
- `mode`: The type of model, such as "partition". Other modes will be added once the package adds more functionality.
- `method`: This is a slot that is filled in later by the model's constructor function. It generally contains lists of information that are used to create the fit and prediction code as well as required packages and similar data.
- `engine`: This character string declares exactly what software will be used. It can be a package name or a technology type.

This class and structure is the basis for how **tidyclust** stores model objects prior to seeing the data.

Argument Details

An important detail to understand when creating model specifications is that they are intended to be functionally independent of the data. While it is true that some tuning parameters are *data dependent*, the model specification does not interact with the data at all.

For example, most R functions immediately evaluate their arguments. For example, when calling `mean(dat_vec)`, the object `dat_vec` is immediately evaluated inside of the function.

`tidyclust` model functions do not do this. For example, using

```
k_means(num_clusters = ncol(mtcars) / 5)
```

does not execute `ncol(mtcars) / 5` when creating the specification. This can be seen in the output:

```
> k_means(num_clusters = ncol(mtcars) / 5)
K Means Cluster Specification (partition)

Main Arguments:
  num_clusters = ncol(mtcars)/5

Computational engine: stats
```

The model functions save the argument *expressions* and their associated environments (a.k.a. a quosure) to be evaluated later when either `fit.cluster_spec()` or `fit_xy.cluster_spec()` are called with the actual data.

The consequence of this strategy is that any data required to get the parameter values must be available when the model is fit. The two main ways that this can fail is if:

1. The data have been modified between the creation of the model specification and when the model fit function is invoked.
2. If the model specification is saved and loaded into a new session where those same data objects do not exist.

The best way to avoid these issues is to not reference any data objects in the global environment but to use data descriptors such as `.cols()`. Another way of writing the previous specification is

```
k_means(num_clusters = .cols() / 5)
```

This is not dependent on any specific data object and is evaluated immediately before the model fitting process begins.

One less advantageous approach to solving this issue is to use quasiquotation. This would insert the actual R object into the model specification and might be the best idea when the data object is small. For example, using

```
k_means(num_clusters = ncol(!!mtcars) - 1)
```

would work (and be reproducible between sessions) but embeds the entire mtcars data set into the `num_clusters` expression:

```
> k_means(num_clusters = ncol(!!mtcars) / 5)
K Means Cluster Specification (partition)

Main Arguments:
  num_clusters = ncol(structure(list(mpg = c(21, 21, 22.8, 21.4, 18.7,<snip>

Computational engine: stats
```

However, if there were an object with the number of columns in it, this wouldn't be too bad:

```
> num_clusters_val <- ncol(mtcars) / 5
> num_clusters_val
[1] 10
> k_means(num_clusters = !!num_clusters_val)
K Means Cluster Specification (partition)

Main Arguments:
  num_clusters = 2.2
```

More information on quosures and quasiquotation can be found at https://adv-r.hadley.nz/quasiquotation.html.

`contr_one_hot` *One-hot contrast matrix*

Description

A re-export of `hardhat::contr_one_hot()` for use with `indicators = "one_hot"`.

Usage

```
contr_one_hot(n, contrasts = TRUE, sparse = FALSE)
```

Arguments

`n`	A vector of character factor levels (of length >=1) or the number of unique levels (>= 1).
`contrasts`	This argument is for backwards compatibility and only the default of `TRUE` is supported.
`sparse`	This argument is for backwards compatibility and only the default of `FALSE` is supported.

cut_height *Cut Height*

Description

Used in most `tidyclust::hier_clust()` models.

Usage

```
cut_height(range = c(0, dials::unknown()), trans = NULL)
```

Arguments

`range`	A two-element vector holding the *defaults* for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the *transformed units*.
`trans`	A `trans` object from the `scales` package, such as `scales::transform_log10()` or `scales::transform_reciprocal()`. If not provided, the default is used which matches the units used in `range`. If no transformation, `NULL`.

Value

A `dials` parameter object for use with `tune::tune_grid()` and related functions.

Examples

```
cut_height()
```

db_clust *Density-Based Spatial Clustering of Applications with Noise (DBSCAN)*

Description

`db_clust` defines a model that fits clusters based on areas with observations that are densely packed together using the DBSCAN algorithm

There are multiple implementations for this model, and the implementation is chosen by setting the model engine. The engine-specific pages for this model are listed below.

- dbscan
- hdbscan

Usage

```
db_clust(
  mode = "partition",
  engine = "dbscan",
  radius = NULL,
  min_points = NULL
)
```

Arguments

mode	A single character string for the type of model. The only possible value for this model is "partition".
engine	A single character string specifying what computational engine to use for fitting. The engine for this model is "dbscan".
radius	Positive double, Radius drawn around points to determine core-points and cluster assignments (required).
min_points	Positive integer, Minimum number of connected points required to form a core-point, including the point itself (required).

Details

What does it mean to predict?:
To predict the cluster assignment for a new observation, we determine if a point is within the radius of a core point. If so, we predict the same cluster as the core point. If not, we predict the observation to be an outlier.

Value

A db_clust cluster specification.

Examples

```
# Show all engines
modelenv::get_from_env("db_clust")

db_clust()
```

---

extract-tidyclust *Extract elements of a tidyclust model object*

Description

These functions extract various elements from a clustering object. If they do not exist yet, an error is thrown.

- extract_fit_engine() returns the engine specific fit embedded within a tidyclust model fit. For example, when using k_means() with the "lm" engine, this returns the underlying kmeans object.
- extract_parameter_set_dials() returns a set of dials parameter objects.

Usage

```
## S3 method for class 'cluster_fit'
extract_fit_engine(x, ...)

## S3 method for class 'cluster_spec'
extract_parameter_set_dials(x, ...)
```

Arguments

x	A cluster_fit object or a cluster_spec object.
...	Not currently used.

Details

Extracting the underlying engine fit can be helpful for describing the model (via `print()`, `summary()`, `plot()`, etc.) or for variable importance/explainers.

However, users should not invoke the `predict()` method on an extracted model. There may be preprocessing operations that `tidyclust` has executed on the data prior to giving it to the model. Bypassing these can lead to errors or silently generating incorrect predictions.

Good:

```
tidyclust_fit |> predict(new_data)
```

Bad:

```
tidyclust_fit |> extract_fit_engine() |> predict(new_data)
```

Value

The extracted value from the tidyclust object, x, as described in the description section.

Examples

```
kmeans_spec <- k_means(num_clusters = 2)
kmeans_fit <- fit(kmeans_spec, ~., data = mtcars)

extract_fit_engine(kmeans_fit)
```

`extract_centroids` *Extract clusters from model*

Description

When applied to a fitted cluster specification, returns a tibble with cluster location. When such locations doesn't make sense for the model, a mean location is used.

Usage

```
extract_centroids(object, ...)
```

Arguments

`object`	An fitted `cluster_spec` object.
`...`	Other arguments passed to methods. Using the `prefix` allows you to change the prefix in the levels of the factor levels. Using `labels` allows you to provide a character vector of cluster labels, overriding `prefix`.

Details

Some model types such as K-means as seen in `k_means()` stores the centroid in the object itself. leading the use of this function to act as an simple extract. Other model types such as Hierarchical (Agglomerative) Clustering as seen in `hier_clust()`, are fit in such a way that the number of clusters can be determined at any time after the fit. Setting the `num_clusters` or `cut_height` in this function will be used to determine the clustering when reported.

Further more, some models like `hier_clust()`, doesn't have a notion of "centroids". The mean of the observation within each cluster assignment is returned as the centroid.

The ordering of the clusters is such that the first observation in the training data set will be in cluster 1, the next observation that doesn't belong to cluster 1 will be in cluster 2, and so on and forth. As the ordering of clustering doesn't matter, this is done to avoid identical sets of clustering having different labels if fit multiple times.

Related functions:

`extract_centroids()` is a part of a trio of functions doing similar things:

- `extract_cluster_assignment()` returns the cluster assignments of the training observations
- `extract_centroids()` returns the location of the centroids
- `predict()` returns the cluster a new observation belongs to

Value

A `tibble::tibble()` with 1 row for each centroid and their position. `.cluster` denotes the cluster name for the centroid. The remaining variables match variables passed into model.

See Also

extract_cluster_assignment() predict.cluster_fit()

Examples

```
set.seed(1234)
kmeans_spec <- k_means(num_clusters = 5) |>
  set_engine("stats")

kmeans_fit <- fit(kmeans_spec, ~., mtcars)

kmeans_fit |>
  extract_centroids()

kmeans_fit |>
  extract_centroids(labels = c("A", "B", "C", "D", "E"))

# Some models such as `hier_clust()` fits in such a way that you can specify
# the number of clusters after the model is fit.
# A Hierarchical (Agglomerative) Clustering method doesn't technically have
# clusters, so the center of the observation within each cluster is returned
# instead.
hclust_spec <- hier_clust() |>
  set_engine("stats")

hclust_fit <- fit(hclust_spec, ~., mtcars)

hclust_fit |>
  extract_centroids(num_clusters = 2)

hclust_fit |>
  extract_centroids(cut_height = 250)
```

---

extract_cluster_assignment

Extract cluster assignments from model

Description

When applied to a fitted cluster specification, returns a tibble with cluster assignments of the data used to train the model.

Usage

```
extract_cluster_assignment(object, ...)
```

Arguments

object	An fitted cluster_spec object.
...	Other arguments passed to methods. Using the prefix allows you to change the prefix in the levels of the factor levels. Using labels allows you to provide a character vector of cluster labels, overriding prefix.

Details

Some model types such as K-means as seen in k_means() stores the cluster assignments in the object itself. leading the use of this function to act as an simple extract. Other model types such as Hierarchical (Agglomerative) Clustering as seen in hier_clust(), are fit in such a way that the number of clusters can be determined at any time after the fit. Setting the num_clusters or cut_height in this function will be used to determine the clustering when reported.

The ordering of the clusters is such that the first observation in the training data set will be in cluster 1, the next observation that doesn't belong to cluster 1 will be in cluster 2, and so on and forth. As the ordering of clustering doesn't matter, this is done to avoid identical sets of clustering having different labels if fit multiple times.

Related functions:

extract_cluster_assignment() is a part of a trio of functions doing similar things:

- extract_cluster_assignment() returns the cluster assignments of the training observations
- extract_centroids() returns the location of the centroids
- predict() returns the cluster a new observation belongs to

Value

A tibble::tibble() with 1 column named .cluster. This tibble will correspond the the training data set.

See Also

extract_centroids() predict.cluster_fit()

Examples

```
kmeans_spec <- k_means(num_clusters = 5) |>
  set_engine("stats")

kmeans_fit <- fit(kmeans_spec, ~., mtcars)

kmeans_fit |>
  extract_cluster_assignment()

kmeans_fit |>
  extract_cluster_assignment(prefix = "C_")

kmeans_fit |>
  extract_cluster_assignment(labels = c("A", "B", "C", "D", "E"))
```

```
# Some models such as `hier_clust()` fits in such a way that you can specify
# the number of clusters after the model is fit
hclust_spec <- hier_clust() |>
  set_engine("stats")

hclust_fit <- fit(hclust_spec, ~., mtcars)

hclust_fit |>
  extract_cluster_assignment(num_clusters = 2)

hclust_fit |>
  extract_cluster_assignment(cut_height = 250)
```

`extract_fit_summary` *S3 method to get fitted model summary info depending on engine*

Description

S3 method to get fitted model summary info depending on engine

Usage

```
extract_fit_summary(object, ...)
```

Arguments

`object`	a fitted `cluster_spec` object
`...`	other arguments passed to methods

Details

The elements `cluster_names` and `cluster_assignments` will be factors.

Value

A list with various summary elements

Examples

```
kmeans_spec <- k_means(num_clusters = 5) |>
  set_engine("stats")

kmeans_fit <- fit(kmeans_spec, ~., mtcars)

kmeans_fit |>
  extract_fit_summary()
```

finalize_model_tidyclust *Splice final parameters into objects*

Description

[Deprecated]

These functions are deprecated. Please use `tune::finalize_model()` and `tune::finalize_workflow()` instead, which now support `cluster_spec` objects natively.

Usage

```
finalize_model_tidyclust(x, parameters)

finalize_workflow_tidyclust(x, parameters)
```

Arguments

`x`	A recipe, `parsnip` model specification, or workflow.
`parameters`	A list or 1-row tibble of parameter values. Note that the column names of the tibble should be the `id` fields attached to `tune()`. For example, in the `Examples` section below, the model has `tune("K")`. In this case, the parameter tibble should be "K" and not "neighbors".

Value

An updated version of `x`.

Examples

```
kmeans_spec <- k_means(num_clusters = tune())
best_params <- data.frame(num_clusters = 5)

# Old:
finalize_model_tidyclust(kmeans_spec, best_params)

# New:
tune::finalize_model(kmeans_spec, best_params)
```

`fit.cluster_spec` *Fit a Model Specification to a Data Set*

Description

`fit()` and `fit_xy()` take a model specification, translate_tidyclust the required code by substituting arguments, and execute the model fit routine.

Usage

```
## S3 method for class 'cluster_spec'
fit(object, formula, data, control = control_cluster(), ...)

## S3 method for class 'cluster_spec'
fit_xy(object, x, case_weights = NULL, control = control_cluster(), ...)
```

Arguments

`object`	An object of class `cluster_spec` that has a chosen engine (via `set_engine()`).
`formula`	An object of class `formula` (or one that can be coerced to that class): a symbolic description of the model to be fitted.
`data`	Optional, depending on the interface (see Details below). A data frame containing all relevant variables (e.g. predictors, case weights, etc). Note: when needed, a *named argument* should be used.
`control`	A named list with elements `verbosity` and `catch`. See `control_cluster()`.
`...`	Not currently used; values passed here will be ignored. Other options required to fit the model should be passed using `set_engine()`.
`x`	A matrix, sparse matrix, or data frame of predictors. Only some models have support for sparse matrix input. See `modelenv::get_encoding()` for details. `x` should have column names.
`case_weights`	An optional classed vector of numeric case weights. This must return `TRUE` when `hardhat::is_case_weights()` is run on it. See `hardhat::frequency_weights()` and `hardhat::importance_weights()` for examples.

Details

`fit()` and `fit_xy()` substitute the current arguments in the model specification into the computational engine's code, check them for validity, then fit the model using the data and the engine-specific code. Different model functions have different interfaces (e.g. formula or x/y) and these functions translate_tidyclust between the interface used when `fit()` or `fit_xy()` was invoked and the one required by the underlying model.

When possible, these functions attempt to avoid making copies of the data. For example, if the underlying model uses a formula and `fit()` is invoked, the original data are references when the model is fit. However, if the underlying model uses something else, such as x/y, the formula is

evaluated and the data are converted to the required format. In this case, any calls in the resulting model objects reference the temporary objects used to fit the model.

If the model engine has not been set, the model's default engine will be used (as discussed on each model page). If the `verbosity` option of `control_cluster()` is greater than zero, a warning will be produced.

If you would like to use an alternative method for generating contrasts when supplying a formula to `fit()`, set the global option `contrasts` to your preferred method. For example, you might set it to: `options(contrasts = c(unordered = "contr.helmert", ordered = "contr.poly"))`. See the help page for `stats::contr.treatment()` for more possible contrast types.

Value

A `cluster_fit` object that contains several elements:

- `spec`: The model specification object (`object` in the call to `fit`)
- `fit`: when the model is executed without error, this is the model object. Otherwise, it is a `try-error` object with the error message.
- `preproc`: any objects needed to convert between a formula and non-formula interface (such as the `terms` object)

The return value will also have a class related to the fitted model (e.g. `"_kmeans"`) before the base class of `"cluster_fit"`.

A fitted `cluster_fit` object.

See Also

`set_engine()`, `control_cluster()`, `cluster_spec`, `cluster_fit`

Examples

```
library(dplyr)

kmeans_mod <- k_means(num_clusters = 5)

using_formula <-
  kmeans_mod |>
  set_engine("stats") |>
  fit(~., data = mtcars)

using_x <-
  kmeans_mod |>
  set_engine("stats") |>
  fit_xy(x = mtcars)

using_formula
using_x
```

get_centroid_dists *Computes distance from observations to centroids*

Description

Computes distance from observations to centroids

Usage

```
get_centroid_dists(
  new_data,
  centroids,
  dist_fun = function(x, y) {
     philentropy::dist_many_many(x, y, method =
    "euclidean")
 }
)
```

Arguments

new_data	A data frame
centroids	A data frame where each row is a centroid.
dist_fun	A function of the form `function(x, y)` that takes two matrices (centroids and observations) and returns a distance matrix. Defaults to `philentropy::dist_many_many` with Euclidean distance. See `philentropy::getDistMethods()` for a list of supported methods, and `vignette("tuning_and_metrics", package = "tidyclust")` for usage examples.

glance.cluster_fit *Construct a single row summary "glance" of a model, fit, or other object*

Description

This method glances the model in a tidyclust model object, if it exists.

Usage

```
## S3 method for class 'cluster_fit'
glance(x, ...)
```

Arguments

x	model or other R object to convert to single-row data frame
...	other arguments passed to methods

Value

A one-row tibble with model-level summary statistics such as total within-cluster sum of squares, between-cluster sum of squares, and number of iterations. Support depends on the underlying engine.

Examples

```
# glance() support depends on the underlying engine.
## Not run:
kmeans_fit <- k_means(num_clusters = 3) |>
  set_engine("stats") |>
  fit(~., mtcars)

glance(kmeans_fit)

## End(Not run)
```

---

gm_clust *Gaussian Mixture Models (GMM)*

Description

`gm_clust` defines a model that fits clusters based on fitting a specified number of multivariate Gaussian distributions (MVG) to the data.

There are multiple implementations for this model, and the implementation is chosen by setting the model engine. The engine-specific pages for this model are listed below.

- mclust

Usage

```
gm_clust(
  mode = "partition",
  engine = "mclust",
  num_clusters = NULL,
  circular = TRUE,
  shared_size = TRUE,
  zero_covariance = TRUE,
  shared_orientation = TRUE,
  shared_shape = TRUE
)
```

Arguments

`mode`	A single character string for the type of model. The only possible value for this model is "partition".
`engine`	A single character string specifying what computational engine to use for fitting. The engine for this model is `"mclust"`.

num_clusters	Positive integer, number of clusters in model (required).
circular	Boolean, whether or not to fit circular MVG distributions for each cluster. Default TRUE.
shared_size	Boolean, whether each cluster MVG should have the same size/volume. Default TRUE.
zero_covariance	Boolean, whether or not to assign covariances of 0 for each MVG. Default TRUE.
shared_orientation	Boolean, whether each cluster MVG should have the same orientation. Default TRUE.
shared_shape	Boolean, whether each cluster MVG should have the same shape. Default TRUE.

Details

What does it mean to predict?:
To predict the cluster assignment for a new observation, we determine which cluster a point has the highest probability of belonging to.

Value

A gm_clust cluster specification.

Examples

```
# Show all engines
modelenv::get_from_env("gm_clust")

gm_clust()
```

---

gm_clust_params *Gaussian mixture covariance structure parameters*

Description

Logical flags controlling the covariance structure of cluster Gaussians fit by tidyclust::gm_clust() with the mclust engine. See gm_clust() for descriptions.

Usage

```
circular(values = c(TRUE, FALSE))

zero_covariance(values = c(TRUE, FALSE))

shared_orientation(values = c(TRUE, FALSE))

shared_shape(values = c(TRUE, FALSE))

shared_size(values = c(TRUE, FALSE))
```

Arguments

values	A vector of possible values (c(TRUE, FALSE) by default).

Value

A dials parameter object for use with tune::tune_grid() and related functions.

Examples

```
circular()
zero_covariance()
shared_orientation()
shared_shape()
shared_size()
```

---

hier_clust *Hierarchical (Agglomerative) Clustering*

Description

hier_clust() defines a model that fits clusters based on a distance-based dendrogram

There are different ways to fit this model, and the method of estimation is chosen by setting the model engine. The engine-specific pages for this model are listed below.

- stats

Usage

```
hier_clust(
  mode = "partition",
  engine = "stats",
  num_clusters = NULL,
  cut_height = NULL,
  linkage_method = "complete",
  dist_fun = NULL
)
```

Arguments

mode	A single character string for the type of model. The only possible value for this model is "partition".
engine	A single character string specifying what computational engine to use for fitting. Possible engines are listed below. The default for this model is "stats".
num_clusters	Positive integer, number of clusters in model (optional).
cut_height	Positive double, height at which to cut dendrogram to obtain cluster assignments (only used if num_clusters is NULL)

`linkage_method` the agglomeration method to be used. This should be (an unambiguous abbreviation of) one of `"ward.D"`, `"ward.D2"`, `"single"`, `"complete"`, `"average"` (= UPGMA), `"mcquitty"` (= WPGMA), `"median"` (= WPGMC) or `"centroid"` (= UPGMC).

`dist_fun` A function for calculating the distance between observations. Defaults to `philentropy::distance` which supports numerous distance metrics via its `method` argument. The function should accept a matrix or data frame and return a square numeric matrix or an object coercible to one via `stats::as.dist()`. See `silhouette()` for further details.

Details

What does it mean to predict?:

To predict the cluster assignment for a new observation, we find the closest cluster. How we measure "closeness" is dependent on the specified type of linkage in the model:

- *single linkage*: The new observation is assigned to the same cluster as its nearest observation from the training data.
- *complete linkage*: The new observation is assigned to the cluster with the smallest maximum distances between training observations and the new observation.
- *average linkage*: The new observation is assigned to the cluster with the smallest average distances between training observations and the new observation.
- *centroid method*: The new observation is assigned to the cluster with the closest centroid, as in prediction for k_means.
- *Ward's method*: The new observation is assigned to the cluster with the smallest increase in **error sum of squares (ESS)** due to the new addition. The ESS is computed as the sum of squared distances between observations in a cluster, and the centroid of the cluster.

Note that these heuristics for assigning new observations to existing clusters are approximations. For most linkage methods, the predictions on training data may not match the cluster assignments from `extract_cluster_assignment()`. This is because `extract_cluster_assignment()` uses `cutree()` to cut the fitted dendrogram, which assigns clusters based on the dendrogram structure rather than proximity to existing cluster members. Observations on the boundary between clusters may therefore be assigned to different clusters by the two methods.

Value

A `hier_clust` cluster specification.

Examples

```
# Show all engines
modelenv::get_from_env("hier_clust")

hier_clust()
```

k_means *K-Means*

Description

k_means() defines a model that fits clusters based on distances to a number of centers. This definition doesn't just include K-means, but includes models like K-prototypes.

There are different ways to fit this model, and the method of estimation is chosen by setting the model engine. The engine-specific pages for this model are listed below.

- stats: Classical K-means
- ClusterR: Classical K-means
- klaR: K-Modes
- clustMixType: K-prototypes

Usage

```
k_means(mode = "partition", engine = "stats", num_clusters = NULL)
```

Arguments

mode	A single character string for the type of model. The only possible value for this model is "partition".
engine	A single character string specifying what computational engine to use for fitting. Possible engines are listed below. The default for this model is "stats".
num_clusters	Positive integer, number of clusters in model.

Details

What does it mean to predict?:

For a K-means model, each cluster is defined by a location in the predictor space. Therefore, prediction in tidyclust is defined by calculating which cluster centroid an observation is closest too.

Value

A k_means cluster specification.

Examples

```
# Show all engines
modelenv::get_from_env("k_means")

k_means()
```

linkage_method *The agglomeration Linkage method*

Description

The agglomeration Linkage method

Usage

```
linkage_method(values = values_linkage_method)

values_linkage_method
```

Arguments

values	A character string of possible values. See linkage_methods in examples below.

Details

This parameter is used in tidyclust models for hier_clust().

Value

A dials parameter object for use with tune::tune_grid() and related functions.

Examples

```
values_linkage_method
linkage_method()
```

mean_shift *Mean Shift Clustering*

Description

mean_shift() defines a model that fits clusters by iteratively shifting observations toward regions of high density, with the number of clusters determined automatically from the data.

There are different implementations for this model, and the implementation is chosen by setting the model engine. The engine-specific pages for this model are listed below.

- LPCM
- meanShiftR

Usage

```
mean_shift(mode = "partition", engine = "LPCM", bandwidth = NULL)
```

Arguments

mode	A single character string for the type of model. The only possible value for this model is "partition".
engine	A single character string specifying what computational engine to use for fitting. The default engine for this model is "LPCM".
bandwidth	Positive double, kernel bandwidth controlling the size of the neighborhood used to compute the density estimate (required).

Details

What does it mean to predict?:
To predict the cluster assignment for a new observation, the mean shift procedure is run from the new point until it converges to a mode. The observation is then assigned to the cluster of the nearest discovered training mode.

Value

A mean_shift cluster specification.

Examples

```
# Show all engines
modelenv::get_from_env("mean_shift")

mean_shift()
```

---

min_grid.cluster_spec *Determine the minimum set of model fits*

Description

Determine the minimum set of model fits

Usage

```
## S3 method for class 'cluster_spec'
min_grid(x, grid, ...)
```

Arguments

x	A cluster specification.
grid	A tibble with tuning parameter combinations.
...	Not currently used.

Value

A tibble with the minimum tuning parameters to fit and an additional list column with the parameter combinations used for prediction.

min_points *Minimum number of points*

Description

The minimum number of connected points required to form a core point in density-based clustering. Used in `tidyclust::db_clust()` with the `dbscan` and `hdbscan` engines.

Usage

```
min_points(range = c(2L, 20L), trans = NULL)
```

Arguments

`range`	A two-element vector holding the *defaults* for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the *transformed units*.
`trans`	A `trans` object from the `scales` package, such as `scales::transform_log10()` or `scales::transform_reciprocal()`. If not provided, the default is used which matches the units used in `range`. If no transformation, `NULL`.

Value

A `dials` parameter object for use with `tune::tune_grid()` and related functions.

Examples

```
min_points()
```

new_cluster_metric *Construct a new clustering metric function*

Description

This function provides a convenient wrapper to create the one type of metric function used in tidyclust: clustering metrics. It adds a metric-specific class to `fn`. These features are used by `cluster_metric_set()` and by `tune_cluster()` when tuning.

Use `new_cluster_metric()` when you want to author your own clustering metric, for example to call `silhouette_avg()` with a non-default `dist_fun`. A plain function cannot be passed to `cluster_metric_set()` directly; it must first be wrapped with `new_cluster_metric()` so that it carries the `cluster_metric` class.

Usage

```
new_cluster_metric(fn, direction)
```

Arguments

fn	A function. It should take `object` and `new_data` as its first two arguments and return a single-row tibble, as produced by the built-in metrics such as `silhouette_avg()`.
direction	A string. One of: • `"maximize"` • `"minimize"` • `"zero"`

Value

A `cluster_metric` object.

See Also

`cluster_metric_set()`

Examples

```
# Author a custom metric that uses a non-default distance function. Here we
# use the average silhouette with Chebyshev (L-infinity) distance.
linf_dist <- function(x) philentropy::distance(x, method = "chebyshev")

linf_silhouette_avg <- new_cluster_metric(
  function(object, new_data = NULL, ...) {
    silhouette_avg(object, new_data = new_data, dist_fun = linf_dist, ...)
  },
  direction = "maximize"
)

# The custom metric can now be combined with others in a metric set.
cluster_metric_set(linf_silhouette_avg, sse_ratio)
```

---

predict.cluster_fit *Model predictions*

Description

Apply to a model to create different types of predictions. `predict()` can be used for all types of models and uses the "type" argument for more specificity.

Usage

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

## S3 method for class 'cluster_fit'
predict_raw(object, new_data, opts = list(), ...)
```

Arguments

object	An object of class cluster_fit.
new_data	A rectangular data object, such as a data frame.
type	A single character value or NULL. Possible values are "cluster", or "raw". When NULL, predict() will choose an appropriate value based on the model's mode.
opts	A list of optional arguments to the underlying predict function that will be used when type = "raw". The list should not include options for the model object or the new data being predicted.
...	Optional arguments passed to the underlying predict function. Use prefix to change the prefix in the cluster factor levels (default: "Cluster_"). Use labels to supply a character vector of cluster labels, which overrides prefix.

Details

If "type" is not supplied to predict(), then a choice is made:

- type = "cluster" for clustering models

predict() is designed to provide a tidy result (see "Value" section below) in a tibble output format.

The ordering of the clusters is such that the first observation in the training data set will be in cluster 1, the next observation that doesn't belong to cluster 1 will be in cluster 2, and so on and forth. As the ordering of clustering doesn't matter, this is done to avoid identical sets of clustering having different labels if fit multiple times.

What does it mean to predict?:

Prediction is not always formally defined for clustering models. Therefore, each cluster_spec method will have their own section on how "prediction" is interpreted, and done if implemented.

Related functions:

predict() when used with tidyclust objects is a part of a trio of functions doing similar things:

- extract_cluster_assignment() returns the cluster assignments of the training observations
- extract_centroids() returns the location of the centroids
- predict() returns the cluster a new observation belongs to

Value

With the exception of type = "raw", the results of predict.cluster_fit() will be a tibble as many rows in the output as there are rows in new_data and the column names will be predictable.

For clustering results the tibble will have a .pred_cluster column.

Using type = "raw" with predict.cluster_fit() will return the unadulterated results of the prediction function.

When the model fit failed and the error was captured, the predict() function will return the same structure as above but filled with missing values. This does not currently work for multivariate models.

See Also

extract_cluster_assignment() extract_centroids()

Examples

```
kmeans_spec <- k_means(num_clusters = 5) |>
  set_engine("stats")

kmeans_fit <- fit(kmeans_spec, ~., mtcars)

kmeans_fit |>
  predict(new_data = mtcars)

# Some models such as `hier_clust()` fits in such a way that you can specify
# the number of clusters after the model is fit
hclust_spec <- hier_clust() |>
  set_engine("stats")

hclust_fit <- fit(hclust_spec, ~., mtcars)

hclust_fit |>
  predict(new_data = mtcars[4:6, ], num_clusters = 2)

hclust_fit |>
  predict(new_data = mtcars[4:6, ], cut_height = 250)
```

---

prep_data_dist *Prepares data and distance matrices for metric calculation*

Description

Prepares data and distance matrices for metric calculation

Usage

```
prep_data_dist(
  object,
  new_data = NULL,
  dists = NULL,
  dist_fun = philentropy::distance
)
```

Arguments

object	A fitted cluster_spec object.
new_data	A dataset to calculate predictions on. If NULL, the trained cluster assignments from the fitted object are used.

dists	A distance matrix for the data. If NULL, distance is computed on new_data using the stats::dist() function.
dist_fun	A function of the form function(x) that takes a data frame or matrix and returns a dist object. Defaults to philentropy::distance with Euclidean distance. See philentropy::getDistMethods() for a list of supported methods, and vignette("tuning_and_metrics", package = "tidyclust") for usage examples.

Value

A list

---

radius *Radius*

Description

The radius used by density-based clustering to determine core points and cluster assignments. Used in tidyclust::db_clust() with the dbscan engine.

Usage

```
radius(range = c(0, dials::unknown()), trans = NULL)
```

Arguments

range	A two-element vector holding the *defaults* for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the *transformed units*.
trans	A trans object from the scales package, such as scales::transform_log10() or scales::transform_reciprocal(). If not provided, the default is used which matches the units used in range. If no transformation, NULL.

Value

A dials parameter object for use with tune::tune_grid() and related functions.

Examples

```
radius()
```

`reconcile_clusterings_mapping` *Relabels clusters to match another cluster assignment*

Description

When forcing one-to-one, the user needs to decide what to prioritize:

- "accuracy": optimize raw count of all observations with the same label across the two assignments
- "precision": optimize the average percent of each alt cluster that matches the corresponding primary cluster

Usage

```
reconcile_clusterings_mapping(
  primary,
  alternative,
  one_to_one = TRUE,
  optimize = "accuracy"
)
```

Arguments

`primary`	A vector containing cluster labels, to be matched
`alternative`	Another vector containing cluster labels, to be changed
`one_to_one`	Boolean; should each alt cluster match only one primary cluster?
`optimize`	One of "accuracy" or "precision"; see description.

Details

Retains the cluster labels of the primary assignment, and relabel the alternate assignment to match as closely as possible. The user must decide whether clusters are forced to be "one-to-one"; that is, are we allowed to assign multiple labels from the alternate assignment to the same primary label?

Cluster labels are arbitrary — two clusterings of the same data may agree on the groups but use different label names (e.g. "Dog" vs "Apple" for the same cluster). `reconcile_clusterings_mapping()` is useful when you want to compare two clusterings, for example:

- Comparing cluster assignments across cross-validation folds.
- Checking stability of a clustering algorithm across different random seeds.
- Aligning predicted clusters on new data with the original training labels.

Value

A tibble with 3 columns; `primary`, `alt`, `alt_recoded`

Examples

```
factor1 <- c("Apple", "Apple", "Carrot", "Carrot", "Banana", "Banana")
factor2 <- c("Dog", "Dog", "Cat", "Dog", "Fish", "Fish")
reconcile_clusterings_mapping(factor1, factor2)

factor1 <- c("Apple", "Apple", "Carrot", "Carrot", "Banana", "Banana")
factor2 <- c("Dog", "Dog", "Cat", "Dog", "Fish", "Parrot")
reconcile_clusterings_mapping(factor1, factor2, one_to_one = FALSE)
```

---

set_args.cluster_spec *Change arguments of a cluster specification*

Description

Change arguments of a cluster specification

Usage

```
## S3 method for class 'cluster_spec'
set_args(object, ...)
```

Arguments

object	A model specification.
...	One or more named model arguments.

Value

An updated cluster_spec object.

---

set_engine.cluster_spec *Change engine of a cluster specification*

Description

Change engine of a cluster specification

Usage

```
## S3 method for class 'cluster_spec'
set_engine(object, engine, ...)
```

Arguments

object	A model specification.
engine	A character string for the software that should be used to fit the model. This is highly dependent on the type of model (e.g. linear regression, random forest, etc.).
...	Any optional arguments associated with the chosen computational engine. These are captured as quosures and can be tuned with tune().

Value

An updated cluster_spec object.

---

set_mode.cluster_spec *Change mode of a cluster specification*

Description

Change mode of a cluster specification

Usage

```
## S3 method for class 'cluster_spec'
set_mode(object, mode, ...)
```

Arguments

object	A model specification.
mode	A character string for the model type (e.g. "classification" or "regression")
...	One or more named model arguments.

Value

An updated cluster_spec object.

`silhouette` *Measures silhouette between clusters*

Description

Measures silhouette between clusters

Usage

```
silhouette(
  object,
  new_data = NULL,
  dists = NULL,
  dist_fun = philentropy::distance
)
```

Arguments

`object`	A fitted tidyclust model
`new_data`	A dataset to predict on. If `NULL`, uses trained clustering.
`dists`	A distance matrix. Used if `new_data` is `NULL`.
`dist_fun`	A function of the form `function(x)` that takes a data frame or matrix and returns a `dist` object. Defaults to `philentropy::distance` with Euclidean distance. See `philentropy::getDistMethods()` for a list of supported methods, and `vignette("tuning_and_metrics", package = "tidyclust")` for usage examples.

Details

`silhouette_avg()` is the corresponding cluster metric function that returns the average of the values given by `silhouette()`.

Value

A tibble giving the silhouette for each observation.

Examples

```
kmeans_spec <- k_means(num_clusters = 5) |>
  set_engine("stats")

kmeans_fit <- fit(kmeans_spec, ~., mtcars)

dists <- mtcars |>
  as.matrix() |>
  dist()

silhouette(kmeans_fit, dists = dists)
```

`silhouette_avg` *Measures average silhouette across all observations*

Description

Measures average silhouette across all observations

Usage

```
silhouette_avg(object, ...)

## S3 method for class 'cluster_spec'
silhouette_avg(object, ...)

## S3 method for class 'cluster_fit'
silhouette_avg(object, new_data = NULL, dists = NULL, dist_fun = NULL, ...)

## S3 method for class 'workflow'
silhouette_avg(object, new_data = NULL, dists = NULL, dist_fun = NULL, ...)

silhouette_avg_vec(
  object,
  new_data = NULL,
  dists = NULL,
  dist_fun = philentropy::distance,
  ...
)
```

Arguments

`object`	A fitted kmeans tidyclust model
`...`	Other arguments passed to methods.
`new_data`	A dataset to predict on. If `NULL`, uses trained clustering.
`dists`	A distance matrix. Used if `new_data` is `NULL`.
`dist_fun`	A function of the form `function(x)` that takes a data frame or matrix and returns a `dist` object. Defaults to `philentropy::distance` with Euclidean distance. See `philentropy::getDistMethods()` for a list of supported methods, and `vignette("tuning_and_metrics", package = "tidyclust")` for usage examples.

Details

Not to be confused with `silhouette()` that returns a tibble with silhouette for each observation. The silhouette coefficient ranges from -1 to 1, where values close to 1 indicate well-separated clusters. This metric has `direction = "maximize"`, so `tune::select_best()` and `tune::show_best()` will return models with the highest silhouette values.

Value

A double; the average silhouette.

See Also

Other cluster metric: sse_ratio(), sse_total(), sse_within_total()

Examples

```
kmeans_spec <- k_means(num_clusters = 5) |>
  set_engine("stats")

kmeans_fit <- fit(kmeans_spec, ~., mtcars)

dists <- mtcars |>
  as.matrix() |>
  dist()

silhouette_avg(kmeans_fit, dists = dists)

silhouette_avg_vec(kmeans_fit, dists = dists)
```

---

sse_ratio *Compute the ratio of the WSS to the total SSE*

Description

Compute the ratio of the WSS to the total SSE

Usage

```
sse_ratio(object, ...)

## S3 method for class 'cluster_spec'
sse_ratio(object, ...)

## S3 method for class 'cluster_fit'
sse_ratio(object, new_data = NULL, dist_fun = NULL, ...)

## S3 method for class 'workflow'
sse_ratio(object, new_data = NULL, dist_fun = NULL, ...)

sse_ratio_vec(
  object,
  new_data = NULL,
  dist_fun = function(x, y) {
     philentropy::dist_many_many(x, y, method =
    "euclidean")
```

```
 },
  ...
)
```

Arguments

object	A fitted kmeans tidyclust model
...	Other arguments passed to methods.
new_data	A dataset to predict on. If NULL, uses trained clustering.
dist_fun	A function of the form function(x, y) that takes two matrices (centroids and observations) and returns a distance matrix. Defaults to philentropy::dist_many_many with Euclidean distance. See philentropy::getDistMethods() for a list of supported methods, and vignette("tuning_and_metrics", package = "tidyclust") for usage examples.

Value

A tibble with 3 columns; .metric, .estimator, and .estimate.

See Also

Other cluster metric: silhouette_avg(), sse_total(), sse_within_total()

Examples

```
kmeans_spec <- k_means(num_clusters = 5) |>
  set_engine("stats")

kmeans_fit <- fit(kmeans_spec, ~., mtcars)

sse_ratio(kmeans_fit)

sse_ratio_vec(kmeans_fit)
```

---

sse_total *Compute the total sum of squares*

Description

Compute the total sum of squares

Usage

```
sse_total(object, ...)

## S3 method for class 'cluster_spec'
sse_total(object, ...)

## S3 method for class 'cluster_fit'
sse_total(object, new_data = NULL, dist_fun = NULL, ...)

## S3 method for class 'workflow'
sse_total(object, new_data = NULL, dist_fun = NULL, ...)

sse_total_vec(
  object,
  new_data = NULL,
  dist_fun = function(x, y) {
     philentropy::dist_many_many(x, y, method =
    "euclidean")
 },
  ...
)
```

Arguments

object	A fitted kmeans tidyclust model
...	Other arguments passed to methods.
new_data	A dataset to predict on. If NULL, uses trained clustering.
dist_fun	A function of the form function(x, y) that takes two matrices (centroids and observations) and returns a distance matrix. Defaults to philentropy::dist_many_many with Euclidean distance. See philentropy::getDistMethods() for a list of supported methods, and vignette("tuning_and_metrics", package = "tidyclust") for usage examples.

Value

A tibble with 3 columns; .metric, .estimator, and .estimate.

See Also

Other cluster metric: silhouette_avg(), sse_ratio(), sse_within_total()

Examples

```
kmeans_spec <- k_means(num_clusters = 5) |>
  set_engine("stats")

kmeans_fit <- fit(kmeans_spec, ~., mtcars)
```

```
sse_total(kmeans_fit)

sse_total_vec(kmeans_fit)
```

`sse_within` *Calculates Sum of Squared Error in each cluster*

Description

Calculates Sum of Squared Error in each cluster

Usage

```
sse_within(
  object,
  new_data = NULL,
  dist_fun = function(x, y) {
     philentropy::dist_many_many(x, y, method =
    "euclidean")
 }
)
```

Arguments

`object`	A fitted kmeans tidyclust model
`new_data`	A dataset to predict on. If `NULL`, uses trained clustering.
`dist_fun`	A function of the form `function(x, y)` that takes two matrices (centroids and observations) and returns a distance matrix. Defaults to `philentropy::dist_many_many` with Euclidean distance. See `philentropy::getDistMethods()` for a list of supported methods, and `vignette("tuning_and_metrics", package = "tidyclust")` for usage examples.

Details

`sse_within_total()` is the corresponding cluster metric function that returns the sum of the values given by `sse_within()`.

Value

A tibble with two columns, the cluster name and the SSE within that cluster.

Examples

```
kmeans_spec <- k_means(num_clusters = 5) |>
  set_engine("stats")

kmeans_fit <- fit(kmeans_spec, ~., mtcars)

sse_within(kmeans_fit)
```

`sse_within_total` *Compute the sum of within-cluster SSE*

Description

Compute the sum of within-cluster SSE

Usage

```
sse_within_total(object, ...)

## S3 method for class 'cluster_spec'
sse_within_total(object, ...)

## S3 method for class 'cluster_fit'
sse_within_total(object, new_data = NULL, dist_fun = NULL, ...)

## S3 method for class 'workflow'
sse_within_total(object, new_data = NULL, dist_fun = NULL, ...)

sse_within_total_vec(
  object,
  new_data = NULL,
  dist_fun = function(x, y) {
     philentropy::dist_many_many(x, y, method =
    "euclidean")
 },
  ...
)
```

Arguments

`object`	A fitted kmeans tidyclust model
`...`	Other arguments passed to methods.
`new_data`	A dataset to predict on. If `NULL`, uses trained clustering.
`dist_fun`	A function of the form `function(x, y)` that takes two matrices (centroids and observations) and returns a distance matrix. Defaults to `philentropy::dist_many_many` with Euclidean distance. See `philentropy::getDistMethods()` for a list of supported methods, and `vignette("tuning_and_metrics", package = "tidyclust")` for usage examples.

Details

Not to be confused with `sse_within()` that returns a tibble with within-cluster SSE, one row for each cluster.

Value

A tibble with 3 columns; `.metric`, `.estimator`, and `.estimate`.

See Also

Other cluster metric: `silhouette_avg()`, `sse_ratio()`, `sse_total()`

Examples

```
kmeans_spec <- k_means(num_clusters = 5) |>
  set_engine("stats")

kmeans_fit <- fit(kmeans_spec, ~., mtcars)

sse_within_total(kmeans_fit)

sse_within_total_vec(kmeans_fit)
```

---

tidy.cluster_fit *Turn a tidyclust model object into a tidy tibble*

Description

This method tidies the model in a tidyclust model object, if it exists.

Usage

```
## S3 method for class 'cluster_fit'
tidy(x, ...)
```

Arguments

`x`	An object to be converted into a tidy `tibble::tibble()`.
`...`	Additional arguments to tidying method.

Value

A tibble with one row per cluster. Columns depend on the underlying engine but typically include `.cluster` and cluster-level summary statistics such as centroid coordinates or cluster size.

Examples

```
# tidy() support depends on the underlying engine. For the stats engine,
# broom must be installed.
## Not run:
kmeans_fit <- k_means(num_clusters = 3) |>
  set_engine("stats") |>
  fit(~., mtcars)
```

```
tidy(kmeans_fit)

hclust_fit <- hier_clust(num_clusters = 3) |>
  set_engine("stats") |>
  fit(~., mtcars)

tidy(hclust_fit)

## End(Not run)
```

`translate_tidyclust` *Resolve a Model Specification for a Computational Engine*

Description

`translate_tidyclust()` will translate_tidyclust a model specification into a code object that is specific to a particular engine (e.g. R package). It translate tidyclust generic parameters to their counterparts.

Usage

```
translate_tidyclust(x, ...)

## Default S3 method:
translate_tidyclust(x, engine = x$engine, ...)
```

Arguments

`x`	A model specification.
`...`	Not currently used.
`engine`	The computational engine for the model (see `?set_engine`).

Details

`translate_tidyclust()` produces a *template* call that lacks the specific argument values (such as `data`, etc). These are filled in once `fit()` is called with the specifics of the data for the model. The call may also include `tune()` arguments if these are in the specification. To handle the `tune()` arguments, you need to use the tune package. For more information see https://www.tidymodels.org/start/tuning/

It does contain the resolved argument names that are specific to the model fitting function/engine.

This function can be useful when you need to understand how `tidyclust` goes from a generic model specific to a model fitting function.

Note: this function is used internally and users should only use it to understand what the underlying syntax would be. It should not be used to modify the cluster specification.

Value

Prints translated code.

`tune_cluster` *Model tuning via grid search*

Description

`tune_cluster()` computes a set of performance metrics for a pre-defined set of tuning parameters that correspond to a cluster model or recipe across one or more resamples of the data.

Usage

```
tune_cluster(object, ...)

## S3 method for class 'cluster_spec'
tune_cluster(
  object,
  preprocessor,
  resamples,
  ...,
  param_info = NULL,
  grid = 10,
  metrics = NULL,
  control = tune::control_grid()
)

## S3 method for class 'workflow'
tune_cluster(
  object,
  resamples,
  ...,
  param_info = NULL,
  grid = 10,
  metrics = NULL,
  control = tune::control_grid()
)
```

Arguments

`object`	A `tidyclust` model specification or a `workflows::workflow()`.
`...`	Not currently used.
`preprocessor`	A traditional model formula or a recipe created using `recipes::recipe()`.
`resamples`	An `rset()` object.

param_info	A dials::parameters() object or NULL. If none is given, a parameters set is derived from other arguments. Passing this argument can be useful when parameter ranges need to be customized.
grid	A data frame of tuning combinations or a positive integer. The data frame should have columns for each parameter being tuned and rows for tuning parameter candidates. An integer denotes the number of candidate parameter sets to be created automatically.
metrics	A cluster_metric_set() or NULL.
control	An object used to modify the tuning process. Defaults to tune::control_grid().

Value

An updated version of resamples with extra list columns for .metrics and .notes (optional columns are .predictions and .extracts). .notes contains warnings and errors that occur during execution. The .notes column is a tibble with columns location, type, note, and trace. The trace column contains rlang::trace_back() objects for errors and warnings, which can be useful for debugging.

Choosing metrics

The metrics argument accepts a cluster_metric_set(). If NULL, the default metrics are sse_within_total() and sse_total().

Common metrics and their interpretation:

- sse_within_total(): Total within-cluster sum of squares. Lower values indicate tighter, more compact clusters. Use the "elbow method" — plot this against num_clusters and look for where the improvement flattens.
- sse_ratio(): Ratio of within-cluster SS to total SS. Lower is better (more variance explained by the clustering).
- silhouette_avg(): Average silhouette width (range -1 to 1). Higher values indicate better-separated clusters. Values above 0.5 are generally considered good.

After tuning, use these functions to inspect results:

- tune::collect_metrics(): All metrics for every parameter combination.
- tune::show_best(): Top N parameter combinations for a given metric.
- tune::select_best(): Single best parameter combination.

Configuration column

The .config column in the results follows the pattern pre{num}_mod{num}_post{num}. The numbers encode which combination of preprocessor, model, and postprocessor parameters was used. A value of 0 means that element was not tuned. For example, pre0_mod2_post0 means the preprocessor was not tuned and this is the second model parameter combination.

Parallel processing

Parallel processing is supported via the `future` and `mirai` packages. To enable parallelism, set up a `future` plan or `mirai` daemons before calling `tune_cluster()`:

```
# Using future
library(future)
plan(multisession, workers = 4)
res <- tune_cluster(wflow, resamples = folds, grid = grid)
plan(sequential)

# Using mirai
library(mirai)
daemons(4)
res <- tune_cluster(wflow, resamples = folds, grid = grid)
daemons(0)
```

See tune::parallelism for more details.

Examples

```
library(recipes)
library(rsample)
library(workflows)
library(tune)

rec_spec <- recipe(~., data = mtcars) |>
  step_normalize(all_numeric_predictors()) |>
  step_pca(all_numeric_predictors())

kmeans_spec <- k_means(num_clusters = tune())

wflow <- workflow() |>
  add_recipe(rec_spec) |>
  add_model(kmeans_spec)

grid <- tibble(num_clusters = 1:3)

set.seed(4400)
folds <- vfold_cv(mtcars, v = 2)

res <- tune_cluster(
  wflow,
  resamples = folds,
  grid = grid
)
res

collect_metrics(res)
```

`update.db_clust` *Update a cluster specification*

Description

If parameters of a cluster specification need to be modified, `update()` can be used in lieu of recreating the object from scratch.

Usage

```
## S3 method for class 'db_clust'
update(
  object,
  parameters = NULL,
  radius = NULL,
  min_points = NULL,
  fresh = FALSE,
  ...
)

## S3 method for class 'gm_clust'
update(
  object,
  parameters = NULL,
  num_clusters = NULL,
  circular = NULL,
  zero_covariance = NULL,
  shared_orientation = NULL,
  shared_shape = NULL,
  shared_size = NULL,
  fresh = FALSE,
  ...
)

## S3 method for class 'hier_clust'
update(
  object,
  parameters = NULL,
  num_clusters = NULL,
  cut_height = NULL,
  linkage_method = NULL,
  dist_fun = NULL,
  fresh = FALSE,
  ...
)

## S3 method for class 'k_means'
```

```
update(object, parameters = NULL, num_clusters = NULL, fresh = FALSE, ...)

## S3 method for class 'mean_shift'
update(object, parameters = NULL, bandwidth = NULL, fresh = FALSE, ...)
```

Arguments

object	A cluster specification.
parameters	A 1-row tibble or named list with *main* parameters to update. Use **either** parameters **or** the main arguments directly when updating. If the main arguments are used, these will supersede the values in parameters. Also, using engine arguments in this object will result in an error.
radius	Positive double, Radius drawn around points to determine core-points and cluster assignments (required).
min_points	Positive integer, Minimum number of connected points required to form a core-point, including the point itself (required).
fresh	A logical for whether the arguments should be modified in-place or replaced wholesale.
...	Not used for update().
num_clusters	Positive integer, number of clusters in model.
circular	Boolean, whether or not to fit circular MVG distributions for each cluster. Default TRUE.
zero_covariance	Boolean, whether or not to assign covariances of 0 for each MVG. Default TRUE.
shared_orientation	Boolean, whether each cluster MVG should have the same orientation. Default TRUE.
shared_shape	Boolean, whether each cluster MVG should have the same shape. Default TRUE.
shared_size	Boolean, whether each cluster MVG should have the same size/volume. Default TRUE.
cut_height	Positive double, height at which to cut dendrogram to obtain cluster assignments (only used if num_clusters is NULL)
linkage_method	the agglomeration method to be used. This should be (an unambiguous abbreviation of) one of "ward.D", "ward.D2", "single", "complete", "average" (= UPGMA), "mcquitty" (= WPGMA), "median" (= WPGMC) or "centroid" (= UPGMC).
dist_fun	A function for calculating the distance between observations. Defaults to philentropy::distance which supports numerous distance metrics via its method argument. The function should accept a matrix or data frame and return a square numeric matrix or an object coercible to one via stats::as.dist(). See silhouette() for further details.
bandwidth	Positive double, kernel bandwidth controlling the size of the neighborhood used to compute the density estimate (required).

Value

An updated cluster specification.

Examples

```
kmeans_spec <- k_means(num_clusters = 5)
kmeans_spec
update(kmeans_spec, num_clusters = 1)
update(kmeans_spec, num_clusters = 1, fresh = TRUE)

param_values <- tibble::tibble(num_clusters = 10)

kmeans_spec |> update(param_values)
```

Index