

# Package ‘mlr3inferr’

May 8, 2026

**Title** Inference on the Generalization Error

**Version** 0.2.1

**Description** Confidence interval and resampling methods for inference on the generalization error.

**License** LGPL-3

**URL** <https://mlr3inferr.mlr-org.com>,  
<https://github.com/mlr-org/mlr3inferr>

**BugReports** <https://github.com/mlr-org/mlr3inferr/issues>

**Depends** mlr3 (>= 1.1.0), R (>= 3.1.0)

**Imports** checkmate, data.table, future, lgr, mlr3measures, mlr3misc, paradox, R6, withr

**Suggests** rpart, testthat (>= 3.0.0)

**Config/testthat/edition** 3

**Encoding** UTF-8

**NeedsCompilation** no

**RoxygenNote** 7.3.2.9000

**Collate** 'MeasureAbstractCi.R' 'aaa.R' 'MeasureCi.R' 'MeasureCiConZ.R'  
'MeasureCiCorT.R' 'MeasureCiHoldout.R' 'MeasureCiNestedCV.R'  
'MeasureCiWaldCV.R' 'ResamplingNestedCV.R'  
'ResamplingPairedSubsampling.R' 'bibentries.R' 'zzz.R'

**Author** Sebastian Fischer [cre, aut] (ORCID:  
<<https://orcid.org/0000-0002-9609-3197>>),  
Hannah Schulz-Kümpel [aut] (ORCID:  
<<https://orcid.org/0000-0003-3972-8392>>)

**Maintainer** Sebastian Fischer <sebf.fischer@gmail.com>

**Repository** CRAN

**Date/Publication** 2025-11-26 13:50:02 UTC

## Contents

mlr3inferr-package . . . . .	2
mlr_measures_abstract_ci . . . . .	3
mlr_measures_ci . . . . .	5
mlr_measures_ci.con_z . . . . .	6
mlr_measures_ci.cor_t . . . . .	7
mlr_measures_ci.holdout . . . . .	8
mlr_measures_ci.ncv . . . . .	9
mlr_measures_ci.wald_cv . . . . .	10
mlr_resamplings_ncv . . . . .	12
mlr_resamplings_paired_subsampling . . . . .	13
<b>Index</b>	<b>16</b>

---

mlr3inferr-package	<i>mlr3inferr: Inference on the Generalization Error</i>
--------------------	--

---

## Description

Confidence interval and resampling methods for inference on the generalization error.

## Author(s)

**Maintainer:** Sebastian Fischer <sebf.fischer@gmail.com> ([ORCID](#))

Authors:

- Hannah Schulz-Kümpel <hannah.kuempel@stat.uni-muenchen.de> ([ORCID](#))

## See Also

Useful links:

- <https://mlr3inferr.mlr-org.com>
- <https://github.com/mlr-org/mlr3inferr>
- Report bugs at <https://github.com/mlr-org/mlr3inferr/issues>

---

mlr\_measures\_abstract\_ci

*Abstract Class for Confidence Intervals*


---

## Description

Base class for confidence interval measures. See section *Inheriting* on how to add a new method.

## Details

The aggregator of the wrapped measure is ignored, as the inheriting CI dictates how the point estimate is constructed. If a measure for which to calculate a CI has `$obs_loss` but also a `$trafo`, (such as RMSE), the delta method is used to obtain confidence intervals.

## Parameters

- `alpha` :: numeric(1)  
The desired alpha level. This is initialized to `$0.05$`.
- `within_range` :: logical(1)  
Whether to restrict the confidence interval within the range of possible values. This is initialized to TRUE.

## Inheriting

To define a new CI method, inherit from the abstract base class and implement the private method: `ci: function(tbl: data.table, rr: ResampleResult, param_vals: named list()) -> numeric(3)`. If `requires_obs_loss` is set to TRUE, `tbl` contains the columns `loss`, `row_id` and `iteration`, which are the pointwise loss, Otherwise, `tbl` contains the result of `rr$score()` with the name of the loss column set to "loss". the identifier of the observation and the resampling iteration. It should return a vector containing the estimate, lower and upper boundary in that order.

In case the confidence interval is not of the form  $(\text{estimate}, \text{estimate} - z * \text{se}, \text{estimate} + z * \text{se})$

it is also necessary to implement the private method: `.trafo: function(ci: numeric(3), measure: Measure) -> numeric(3)`

Which receives a confidence interval for a pointwise loss (e.g. squared-error) and transforms it according to the transformation `measure$trafo` (e.g. `sqrt` to go from `mse` to `rmse`).

## Super class

`mlr3::Measure -> MeasureAbstractCi`

## Public fields

`resamplings` (character())

On which resampling classes this method can operate.

`measure` ([Measure](#))

## Methods

### Public methods:

- [MeasureAbstractCi\\$new\(\)](#)
- [MeasureAbstractCi\\$aggregate\(\)](#)
- [MeasureAbstractCi\\$clone\(\)](#)

**Method** `new()`: Creates a new instance of this R6 class.

*Usage:*

```
MeasureAbstractCi$new(
  measure = NULL,
  param_set = ps(),
  packages = character(),
  resamplings,
  label,
  delta_method = FALSE,
  requires_obs_loss = TRUE,
  man = NA
)
```

*Arguments:*

`measure` ([Measure](#))

The measure for which to calculate a confidence interval. Must have `$obs_loss`.

`param_set` ([ParamSet](#))

Set of hyperparameters.

`packages` (`character()`)

Set of required packages. A warning is signaled by the constructor if at least one of the packages is not installed, but loaded (not attached) later on-demand via [requireNamespace\(\)](#).

`resamplings` (`character()`)

To which resampling classes this measure can be applied.

`label` (`character(1)`)

Label for the new instance.

`delta_method` (`logical(1)`)

Whether to use the delta method for measures (such RMSE) that have a trafo.

`requires_obs_loss` (`logical(1)`)

Whether the inference method requires a pointwise loss function.

`man` (`character(1)`)

Manual page.

**Method** `aggregate()`: Obtain a point estimate, as well as lower and upper CI boundary.

*Usage:*

```
MeasureAbstractCi$aggregate(rr)
```

*Arguments:*

`rr` ([ResampleResult](#))

The resample result.

*Returns:* `named numeric(3)`

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
MeasureAbstractCi$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

mlr_measures_ci	<i>Default CI Method</i>
-----------------	--------------------------

---

## Description

For certain resampling methods, there are default confidence interval methods. See `mlr3::mlr_reflections$default_ci_m` for a selection. This measure will select the appropriate CI method depending on the class of the used [Resampling](#).

## Parameters

Only those from [MeasureAbstractCi](#).

## Super classes

```
mlr3::Measure -> mlr3inferr::MeasureAbstractCi -> Measure
```

## Methods

### Public methods:

- [MeasureCi\\$new\(\)](#)
- [MeasureCi\\$aggregate\(\)](#)
- [MeasureCi\\$clone\(\)](#)

**Method** `new()`: Creates a new instance of this [R6](#) class.

*Usage:*

```
MeasureCi$new(measure)
```

*Arguments:*

`measure` ([Measure](#) or `character(1)`)  
A measure of ID of a measure.

**Method** `aggregate()`: Obtain a point estimate, as well as lower and upper CI boundary.

*Usage:*

```
MeasureCi$aggregate(rr)
```

*Arguments:*

`rr` ([ResampleResult](#))  
Resample result.

*Returns:* `named numeric(3)`

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
MeasureCi$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
rr = resample(tsk("sonar"), lrn("classif.featureless"), rsmp("holdout"))
rr$aggregate(msr("ci", "classif.acc"))
# is the same as:
rr$aggregate(msr("ci.holdout", "classif.acc"))
```

---

mlr\_measures\_ci.con\_z *Conservative-Z CI*

---

## Description

The conservative-z confidence intervals based on the [ResamplingPairedSubsampling](#). Because the variance estimate is obtained using only  $n / 2$  observations, it tends to be conservative. This inference method can also be applied to non-decomposable losses.

## Point Estimation

For the point estimation, only the first `repeats_out` resampling iterations will be used, as the other resampling iterations are only used to estimate the variance. This is respected when calling `$aggregate()` using a standard (non-CI) measure.

## Parameters

Only those from [MeasureAbstractCi](#).

## Super classes

```
mlr3::Measure -> mlr3inferr::MeasureAbstractCi -> MeasureCiConZ
```

## Methods

### Public methods:

- [MeasureCiConZ\\$new\(\)](#)
- [MeasureCiConZ\\$clone\(\)](#)

**Method** `new()`: Creates a new instance of this R6 class.

*Usage:*

```
MeasureCiConZ$new(measure)
```

*Arguments:*

measure ([Measure](#) or character(1))  
A measure of ID of a measure.

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

```
MeasureCiConZ$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

## References

Nadeau, Claude, Bengio, Yoshua (1999). "Inference for the generalization error." *Advances in neural information processing systems*, **12**.

## Examples

```
ci_conz = msr("ci.con_z", "classif.acc")
ci_conz
```

---

mlr\_measures\_ci.cor\_t *Corrected-T CI*

---

## Description

Corrected-T confidence intervals based on [ResamplingSubsampling](#). A heuristic factor is applied to correct for the dependence between the iterations. The confidence intervals tend to be liberal. This inference method can also be applied to non-decomposable losses.

## Parameters

Only those from [MeasureAbstractCi](#).

## Super classes

```
mlr3::Measure -> mlr3infe::MeasureAbstractCi -> MeasureCiCorrectedT
```

## Methods

### Public methods:

- [MeasureCiCorrectedT\\$new\(\)](#)
- [MeasureCiCorrectedT\\$clone\(\)](#)

**Method** new(): Creates a new instance of this [R6](#) class.

*Usage:*

```
MeasureCiCorrectedT$new(measure)
```

*Arguments:*

measure ([Measure](#) or character(1))  
 A measure of ID of a measure.

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

```
MeasureCiCorrectedT$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

## References

Nadeau, Claude, Bengio, Yoshua (1999). "Inference for the generalization error." *Advances in neural information processing systems*, **12**.

## Examples

```
m_cort = msr("ci.cor_t", "classif.acc")
m_cort
rr = resample(
  tsk("sonar"),
  lrn("classif.featureless"),
  rsmp("subsampling", repeats = 10)
)
rr$aggregate(m_cort)
```

---

```
mlr_measures_ci.holdout
```

*Holdout CI*

---

## Description

Standard holdout CI. This inference method can only be applied to decomposable losses.

## Parameters

Only those from [MeasureAbstractCi](#).

## Super classes

```
mlr3::Measure -> mlr3infern::MeasureAbstractCi -> MeasureCiHoldout
```

## Methods

### Public methods:

- [MeasureCiHoldout\\$new\(\)](#)
- [MeasureCiHoldout\\$clone\(\)](#)

**Method** `new()`: Creates a new instance of this [R6](#) class.

*Usage:*

```
MeasureCiHoldout$new(measure)
```

*Arguments:*

measure ([Measure](#) or `character(1)`)  
A measure of ID of a measure.

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
MeasureCiHoldout$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

## Examples

```
ci_ho = msr("ci.holdout", "classif.acc")
ci_ho
rr = resample(tsk("sonar"), lrn("classif.featureless"), rsmp("holdout"))
rr$aggregate(ci_ho)
```

---

mlr\_measures\_ci.ncv    *Nested CV CI*

---

## Description

Confidence Intervals based on [ResamplingNestedCV](#), including bias-correction. This inference method can only be applied to decomposable losses.

## Point Estimation

The point estimate uses a bias correction term as described in Bates et al. (2024). Therefore, the results of directly applying a measure `$aggregate(msr(<key>))` will be different from the point estimate of `$aggregate(msr("ci", <key>))`, where the point estimate is obtained by averaging over the outer CV results.

## Parameters

Those from [MeasureAbstractCi](#), as well as:

- `bias :: logical(1)`  
Whether to do bias correction. This is initialized to TRUE. If FALSE, the outer iterations are used for the point estimate and no bias correction is applied.

**Super classes**

```
mlr3::Measure -> mlr3infeer::MeasureAbstractCi -> MeasureCiNestedCV
```

**Methods****Public methods:**

- `MeasureCiNestedCV$new()`
- `MeasureCiNestedCV$clone()`

**Method** `new()`: Creates a new instance of this [R6](#) class.

*Usage:*

```
MeasureCiNestedCV$new(measure)
```

*Arguments:*

measure ([Measure](#) or `character(1)`)  
A measure of ID of a measure.

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
MeasureCiNestedCV$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

**References**

Bates, Stephen, Hastie, Trevor, Tibshirani, Robert (2024). “Cross-validation: what does it estimate and how well does it do it?” *Journal of the American Statistical Association*, **119**(546), 1434–1445.

**Examples**

```
ci_ncv = msr("ci.ncv", "classif.acc")
ci_ncv
```

---

```
mlr_measures_ci.wald_cv
```

*Cross-Validation CI*

---

**Description**

Confidence intervals for cross-validation. The method is asymptotically exact for the so called *Test Error* as defined by Bayle et al. (2020). For the (expected) risk, the confidence intervals tend to be too liberal. This inference method can only be applied to decomposable losses.

## Parameters

Those from [MeasureAbstractCi](#), as well as:

- `variance` :: "all-pairs" or "within-fold"  
How to estimate the variance. The results tend to be very similar.

## Super classes

`mlr3::Measure` -> `mlr3inferred::MeasureAbstractCi` -> `MeasureCiWaldCV`

## Methods

### Public methods:

- `MeasureCiWaldCV$new()`
- `MeasureCiWaldCV$clone()`

**Method** `new()`: Creates a new instance of this R6 class.

*Usage:*

```
MeasureCiWaldCV$new(measure)
```

*Arguments:*

`measure` ([Measure](#) or `character(1)`)  
A measure of ID of a measure.

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
MeasureCiWaldCV$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## References

Bayle, Pierre, Bayle, Alexandre, Janson, Lucas, Mackey, Lester (2020). "Cross-validation confidence intervals for test error." *Advances in Neural Information Processing Systems*, **33**, 16339–16350.

## Examples

```
m_walddcv = msr("ci.wald_cv", "classif.ce")
m_walddcv
rr = resample(tsk("sonar"), lrn("classif.featureless"), rsmp("cv"))
rr$aggregate(m_walddcv)
```

---

mlr\_resamplings\_ncv *Nested Cross-Validation*

---

## Description

This implements the Nested CV resampling procedure by Bates et al. (2024).

## Point Estimation

When calling `$aggregate()` on a resample result obtained using this resampling method, only the outer resampling iterations will be used, as they have a smaller bias. See section "Point Estimation" of [MeasureCINestedCV](#).

## Parameters

- `folds :: integer(1)`  
The number of folds. This is initialized to 5.
- `repeats :: integer(1)`  
The number of repetitions. This is initialized to 10.

## Super class

`mlr3::Resampling` -> `ResamplingNestedCV`

## Active bindings

`iters (integer(1))`  
The total number of resampling iterations.

## Methods

### Public methods:

- `ResamplingNestedCV$new()`
- `ResamplingNestedCV$unflatten()`
- `ResamplingNestedCV$clone()`

**Method** `new()`: Creates a new instance of this R6 class.

*Usage:*

```
ResamplingNestedCV$new()
```

**Method** `unflatten()`: Convert a resampling iteration to a more useful representation. For outer resampling iterations, `inner` is NA.

*Usage:*

```
ResamplingNestedCV$unflatten(iter)
```

*Arguments:*

`iter` (integer(1))  
The iteration.

*Returns:* list(rep, outer, inner)

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
ResamplingNestedCV$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## References

Bates, Stephen, Hastie, Trevor, Tibshirani, Robert (2024). “Cross-validation: what does it estimate and how well does it do it?” *Journal of the American Statistical Association*, **119**(546), 1434–1445.

## Examples

```
ncv = rsmp("ncv", folds = 3, repeats = 10L)
ncv
rr = resample(tsk("mtcars"), lrn("regr.featureless"), ncv)
```

---

mlr\_resamplings\_paired\_subsampling  
*Paired Subsampling*

---

## Description

Paired Subsampling to enable inference on the generalization error.

## Details

The first `repeats_in` iterations are a standard [ResamplingSubsampling](#) and should be used to obtain a point estimate of the generalization error. The remaining iterations should be used to estimate the standard error. Here, the data is divided `repeats_out` times into two equally sized disjunct subsets, to each of which subsampling which, a subsampling with `repeats_in` repetitions is applied. See the `$unflatten(iter)` method to map the iterations to this nested structure.

## Point Estimation

When calling `$aggregate()` on a resample result obtained using this resampling method, only the first `repeats_out` iterations will be used. See section "Point Estimation" of [MeasureCiConZ](#).

**Parameters**

- `repeats_in :: integer(1)`  
The inner repetitions.
- `repeats_out :: integer(1)`  
The outer repetitions.
- `ratio :: numeric(1)`  
The proportion of data to use for training.

**Super class**

`mlr3::Resampling` -> `ResamplingPairedSubsampling`

**Active bindings**

`iters (integer(1))`  
The total number of resampling iterations.

**Methods****Public methods:**

- `ResamplingPairedSubsampling$new()`
- `ResamplingPairedSubsampling$unflatten()`
- `ResamplingPairedSubsampling$clone()`

**Method** `new()`: Creates a new instance of this R6 class.

*Usage:*

`ResamplingPairedSubsampling$new()`

**Method** `unflatten()`: Unflatten the resampling iteration into a more informative representation:

- `inner`: The subsampling iteration
- `outer`: NA for the first `repeats_in` iterations. Otherwise it indicates the outer iteration of the paired subsamplings.
- `partition`: NA for the first `repeats_in` iterations. Otherwise it indicates whether the subsampling is applied to the first or second partition Of the two disjoint halves.

*Usage:*

`ResamplingPairedSubsampling$unflatten(iter)`

*Arguments:*

`iter (integer(1))`  
Resampling iteration.

*Returns:* `list(outer, partition, inner)`

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

`ResamplingPairedSubsampling$clone(deep = FALSE)`

*Arguments:*

`deep` Whether to make a deep clone.

## **References**

Nadeau, Claude, Bengio, Yoshua (1999). "Inference for the generalization error." *Advances in neural information processing systems*, **12**.

## **Examples**

```
pw_subs = rsmpl("paired_subsampling")  
pw_subs
```

# Index

Measure, [3–5](#), [7–11](#)  
MeasureAbstractCi, [5–9](#), [11](#)  
MeasureAbstractCi  
    ([mlr\\_measures\\_abstract\\_ci](#)), [3](#)  
MeasureCi ([mlr\\_measures\\_ci](#)), [5](#)  
MeasureCiConZ, [13](#)  
MeasureCiConZ ([mlr\\_measures\\_ci.con\\_z](#)), [6](#)  
MeasureCiCorrectedT  
    ([mlr\\_measures\\_ci.cor\\_t](#)), [7](#)  
MeasureCiHoldout  
    ([mlr\\_measures\\_ci.holdout](#)), [8](#)  
MeasureCiNestedCV, [12](#)  
MeasureCiNestedCV  
    ([mlr\\_measures\\_ci.ncv](#)), [9](#)  
MeasureCiWaldCV  
    ([mlr\\_measures\\_ci.wald\\_cv](#)), [10](#)  
mlr3::Measure, [3](#), [5–8](#), [10](#), [11](#)  
mlr3::Resampling, [12](#), [14](#)  
mlr3infeer ([mlr3infeer-package](#)), [2](#)  
mlr3infeer-package, [2](#)  
mlr3infeer::MeasureAbstractCi, [5–8](#), [10](#),  
    [11](#)  
mlr\_measures\_abstract\_ci, [3](#)  
mlr\_measures\_ci, [5](#)  
mlr\_measures\_ci.con\_z, [6](#)  
mlr\_measures\_ci.cor\_t, [7](#)  
mlr\_measures\_ci.holdout, [8](#)  
mlr\_measures\_ci.ncv, [9](#)  
mlr\_measures\_ci.wald\_cv, [10](#)  
mlr\_resamplings\_ncv, [12](#)  
mlr\_resamplings\_paired\_subsampling, [13](#)  
  
ParamSet, [4](#)  
  
R6, [4–7](#), [9–12](#), [14](#)  
requireNamespace(), [4](#)  
ResampleResult, [4](#), [5](#)  
Resampling, [5](#)  
ResamplingNestedCV, [9](#)  
ResamplingNestedCV  
    ([mlr\\_resamplings\\_ncv](#)), [12](#)  
ResamplingPairedSubsampling, [6](#)  
ResamplingPairedSubsampling  
    ([mlr\\_resamplings\\_paired\\_subsampling](#)),  
    [13](#)  
ResamplingSubsampling, [7](#), [13](#)