

Package ‘mlr3fairness’

June 24, 2025

Type Package

Title Fairness Auditing and Debiasing for 'mlr3'

Version 0.4.0

Description Integrates fairness auditing and bias mitigation methods for the 'mlr3' ecosystem. This includes fairness metrics, reporting tools, visualizations and bias mitigation techniques such as ``Reweighting'' described in 'Kamiran, Calders' (2012) <[doi:10.1007/s10115-011-0463-8](https://doi.org/10.1007/s10115-011-0463-8)> and ``Equalized Odds'' described in 'Hardt et al.' (2016) <<https://papers.nips.cc/paper/2016/file/9d2682367c3935defcb1f9e247a97c0d-Paper.pdf>>. Integration with 'mlr3' allows for auditing of ML models as well as convenient joint tuning of machine learning algorithms and debiasing methods.

License LGPL-3

URL <https://mlr3fairness.mlr-org.com>,
<https://github.com/mlr-org/mlr3fairness>

BugReports <https://github.com/mlr-org/mlr3fairness/issues>

Depends R (>= 3.4.0)

Imports checkmate, data.table (>= 1.13.6), ggplot2, mlr3 (>= 0.23.0.9000), mlr3learners, mlr3measures, mlr3misc, mlr3pipelines, paradox, R6 (>= 2.4.1), rlang

Suggests cccp, CVXR, pagedown, fairml, future, iml, kableExtra, knitr, linprog, lgr, mlr3viz, patchwork, ranger, rmarkdown, rpart, testthat (>= 3.1.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel false

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2
NeedsCompilation no
Author Florian Pfisterer [cre, aut] (ORCID: <https://orcid.org/0000-0001-8867-762X>),
Wei Siyi [aut],
Michel Lang [aut] (ORCID: <https://orcid.org/0000-0001-9754-0393>)
Maintainer Florian Pfisterer <pfistererf@googlemail.com>
Repository CRAN
Date/Publication 2025-06-24 12:40:02 UTC

Contents

adult	2
compare_metrics	4
compas	5
compute_metrics	7
fairness_accuracy_tradeoff	8
fairness_prediction_density	10
fairness_tensor	11
groupdiff_tau	12
groupwise_metrics	13
MeasureFairness	14
MeasureFairnessComposite	16
MeasureFairnessConstraint	17
MeasureSubgroup	19
mlr_learners_fairness	20
mlr_measures_fairness	21
mlr_measures_positive_probability	22
mlr_pipeops_equalized_odds	23
mlr_pipeops_explicit_pta	25
mlr_pipeops_reweighing	27
report_datasheet	30
report_fairness	31
report_modelcard	32
task_summary	33
Index	34

adult	<i>Adult Dataset</i>
-------	----------------------

Description

Dataset used to predict whether income exceeds \$50K/yr based on census data. Also known as "Census Income" dataset Train dataset contains 13 features and 30178 observations. Test dataset contains 13 features and 15315 observations. Target column is "target": A binary factor where 1: <=50K and 2: >50K for annual income. The column "sex" is set as protected attribute.

Derived tasks

- `adult_train`: Original train split for the adult task available at UCI.
- `adult_test`: Original test split for the adult task available at UCI.

Using Adult - Known Problems

The adult dataset has several known limitations such as its age, limited documentation, and out-dated feature encodings (Ding et al., 2021). Furthermore, the selected threshold (income $\leq 50K$) has strong implications on the outcome of analysis, such that "In many cases, the \$50k threshold understates and misrepresents the broader picture" (Ding et al., 2021). As a result, conclusions w.r.t. real-world implications are severely limited.

We decide to replicate the dataset here, as it is a widely used benchmark dataset and it can still serve this purpose.

Pre-processing

- `fnlwgt` Remove final weight, which is the number of people the census believes the entry represents
- `native-country` Remove Native Country, which is the country of origin for an individual
- Rows containing NA in `workclass` and `occupation` have been removed.
- Pre-processing inspired by article: @url <https://cseweb.ucsd.edu/classes/sp15/cse190-c/reports/sp15/048.pdf>

Metadata

- (integer) `age`: The age of the individuals
- (factor) `workclass`: A general term to represent the employment status of an individual
- (factor) `education`: The highest level of education achieved by an individual.
- (integer) `education_num`: the highest level of education achieved in numerical form.
- (factor) `marital_status`: marital status of an individual.
- (factor) `occupation`: the general type of occupation of an individual
- (factor) `relationship`: whether the individual is in a relationship-
- (factor) `race`: Descriptions of an individual's race
- (factor) `sex`: the biological sex of the individual
- (integer) `capital-gain`: capital gains for an individual
- (integer) `capital-loss`: capital loss for an individual
- (integer) `hours-per-week`: the hours an individual has reported to work per week
- (factor) `target`: whether or not an individual makes more than \$50,000 annually

Source

Dua, Dheeru, Graff, Casey (2017). "UCI Machine Learning Repository." <http://archive.ics.uci.edu/ml/>. Ding, Frances, Hardt, Moritz, Miller, John, Schmidt, Ludwig (2021). "Retiring adult: New datasets for fair machine learning." In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

Examples

```
library("mlr3")
data("adult_test", package = "mlr3fairness")
data("adult_train", package = "mlr3fairness")
```

compare_metrics	<i>Compare different metrics</i>
-----------------	----------------------------------

Description

Compare learners with respect to one or multiple metrics. Metrics can but be but are not limited to fairness metrics.

Usage

```
compare_metrics(object, ...)
```

Arguments

- | | |
|--------|---|
| object | (mlr3::PredictionClassif mlr3::BenchmarkResult mlr3::ResampleResult)
The object to create a plot for. |
| | <ul style="list-style-type: none"> If provided a (mlr3::PredictionClassif). Then the visualization will compare the fairness metrics among the binary level from protected field through bar plots. If provided a (mlr3::ResampleResult). Then the visualization will generate the boxplots for fairness metrics, and compare them among the binary level from protected field. If provided a (mlr3::BenchmarkResult). Then the visualization will generate the boxplots for fairness metrics, and compare them among both the binary level from protected field and the models implemented. |
| ... | The arguments to be passed to methods, such as: <ul style="list-style-type: none"> fairness_measures (list of mlr3::Measure)
The fairness measures that will be evaluated on object, could be single mlr3::Measure or list of mlr3::Measures. Default measure set to be <code>msr("fairness.acc")</code>. task (mlr3::TaskClassif)
The data task that contains the protected column, only required when object is (mlr3::PredictionClassif). |

Value

A 'ggplot2' object.

Protected Attributes

The protected attribute is specified as a `col_role` in the corresponding [mlr3::Task\(\)](#):

```
<Task>$col_roles$pta = "name_of_attribute"
```

This also allows specifying more than one protected attribute, in which case fairness will be considered on the level of intersecting groups defined by all columns selected as a predicted attribute.

Examples

```
library("mlr3")
library("mlr3learners")

# Setup the Fairness Measures and tasks
task = tsk("adult_train")$filter(1:500)
learner = lrn("classif.ranger", predict_type = "prob")
learner$train(task)
predictions = learner$predict(task)
design = benchmark_grid(
  tasks = task,
  learners = lrns(c("classif.ranger", "classif.rpart"),
    predict_type = "prob", predict_sets = c("train", "test")),
  resamplings = rsmpls("cv", folds = 3)
)

bmr = benchmark(design)
fairness_measure = msr("fairness.tpr")
fairness_measures = msrs(c("fairness.tpr", "fairness.fnr", "fairness.acc"))

# Predictions
compare_metrics(predictions, fairness_measure, task)
compare_metrics(predictions, fairness_measures, task)

# BenchmarkResult and ResamplingResult
compare_metrics(bmr, fairness_measure)
compare_metrics(bmr, fairness_measures)
```

compas

COMPAS Dataset

Description

The COMPAS dataset includes the processed COMPAS data between 2013-2014. The data cleaning process followed the guidance in the original COMPAS repo. Contains 6172 observations and 14 features. The target column could either be "is_recid" or "two_year_recid", but often "two_year_recid" is preferred. The column "sex" is set as protected attribute, but more often "race" is used.

Derived tasks:

- compas : A classification task for the [compas](#) data set with the protected attribute 'sex'.
- compas_race_binary : A classification task for the [compas](#) data set with the protected attribute 'race'. The observations have been filtered, keeping only observations with race "Caucasian" and "African-American". The protected attribute has been set to "race".

Format

`R6::R6Class` inheriting from `mlr3::TaskClassif`.

`R6::R6Class` inheriting from `mlr3::TaskClassif`.

Using COMPAS - Known Problems

The COMPAS dataset was collected as part of the ProPublica analysis of machine bias in criminal sentencing. It is important to note, that using COMPAS is generally discouraged for the following reasons:

- The prediction task derived from this dataset has little connection to actually relevant tasks in the context of risk assessment instruments.
- Collected data and labels suffer from disparate measurement bias.

The dataset should therefore not be used to benchmark new fairness algorithms or measures. For a more in-depth treatment, see Bao et al., 2021: It's COMPASlicated: The Messy Relationship between RAI Datasets and Algorithmic Fairness Benchmarks. We replicate the dataset here to raise awareness for this issue. Furthermore, similar issues exist across a wide variety of datasets widely used in the context of fairness auditing and we, therefore, consider issues, e.g. derived from disparate measurement bias an important issue in the context of fairness audits.

Pre-processing

- Identifying columns are removed
- Removed the outliers for `abs(days_b_screening_arrest) >= 30`.
- Removed observations where `is_recid != -1`.
- Removed observations where `c_charge_degree != "O"`.
- Removed observations where `score_text != 'N/A'`.
- Factorize the features that are categorical.
- Add length of stay (`c_jail_out - c_jail_in`) in the dataset.
- Pre-processing Resource: @url <https://github.com/propublica/compas-analysis/blob/master/Compas%20Analysis.i>

Note: The 'is_recid' column was removed as it's a target column.

Metadata

- (integer) age: The age of defendants.
- (factor) c_charge_degree : The charge degree of defendants. F: Felony M: Misdemeanor
- (factor) race: The race of defendants.
- (factor) age_cat: The age category of defendants.
- (factor) score_text: The score category of defendants.
- (factor) sex: The sex of defendants.
- (integer) priors_count: The prior criminal records of defendants.
- (integer) days_b_screening_arrest: The count of days between screening date and (original) arrest date. If they are too far apart, that may indicate an error. If the value is negative, that indicate the screening date happened before the arrest date.

- (integer) decile_score: Indicate the risk of recidivism (Min=1, Max=10)
- (factor) two_year_recid: Binary variable indicate whether defendant is rearrested at within two years.
- (numeric) length_of_stay: The count of days stay in jail.

Construction

```
mlr_tasks$get("compas")
tsk("compas")

mlr_tasks$get("compas_race_binary")
tsk("compas_race_binary")
```

Source

ProPublica Analysis: <https://github.com/propublica/compas-analysis>

Bao, Michelle, Zhou, Angela, Zottola, A S, Brubach, Brian, Desmarais, Sarah, Horowitz, Seth A, Lum, Kristian, Venkatasubramanian, Suresh (2021). "It's COMPASlicated: The Messy Relationship between RAI Datasets and Algorithmic Fairness Benchmarks." In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

Examples

```
library("mlr3")
data("compas", package = "mlr3fairness")
```

compute_metrics	<i>Compute metrics for non-mlr3 predictions.</i>
-----------------	--

Description

Allows computing metrics for predictions that do not stem from mlr3, and were e.g. being made by models outside of mlr3. Currently only classif and regr - style predictions are supported.

Usage

```
compute_metrics(data, target, protected_attribute, prediction, metrics = NULL)
```

Arguments

data	(data.table) The dataset used for predicting.
target	(character) The name of the target variable. Must be available in data.
protected_attribute	(character) The name(s) of the protected attributes(s). Must be available in data.

prediction	(vector) A vector containing predictions.
metrics	(Metriclist) (List of) mlr3 metrics to apply.

Protected Attributes

The protected attribute is specified as a `col_role` in the corresponding `mlr3::Task()`:

```
<Task>$col_roles$pta = "name_of_attribute"
```

This also allows specifying more than one protected attribute, in which case fairness will be considered on the level of intersecting groups defined by all columns selected as a predicted attribute.

Examples

```
library("mlr3")
# Get adult data as a data.table
train = tsk("adult_train")$data()
mod = rpart::rpart(target ~ ., train)

# Predict on test data
test = tsk("adult_test")$data()
yhat = predict(mod, test, type = "vector")

# Convert to a factor with the same levels
yhat = as.factor(yhat)
levels(yhat) = levels(test$target)

compute_metrics(
  data = test,
  target = "target",
  prediction = yhat,
  protected_attribute = "sex",
  metrics = msr("fairness.acc")
)
```

fairness_accuracy_tradeoff

Plot Fairness Accuracy Trade-offs

Description

Provides visualization wrt. trade-offs between fairness and accuracy metrics across learners and resampling iterations. This can assist in gauging the optimal model from a set of options along with estimates of variance (through individual resampling iterations).

Usage

```
fairness_accuracy_tradeoff(object, ...)
```


Arguments

- object** ([mlr3::PredictionClassif](#) | [mlr3::BenchmarkResult](#) | [mlr3::ResampleResult](#))
The binary class prediction object that will be evaluated.
- If provided a [mlr3::PredictionClassif](#). Then only one point will indicate the accuracy and fairness metrics for the current predictions. Requires also passing a [mlr3::Task](#).
 - If provided a [mlr3::ResampleResult](#). Then the plot will compare the accuracy and fairness metrics for the same model, but different resampling iterations as well as the aggregate indicated by a cross.
 - If provided a [mlr3::BenchmarkResult](#). Then the plot will compare the accuracy and fairness metrics for all models and all resampling iterations. Points are colored according to the learner_id and faceted by task_id. The aggregated score is indicated by a cross.
- ...** Arguments to be passed to methods. Such as:
- **fairness_measure** ([mlr3::Measure](#))
The fairness measures that will be evaluated. Default measure set to be `msr("fairness.fpr")`
 - **accuracy_measure** ([mlr3::Measure](#))
The accuracy measure that will be evaluated. Default measure set to be `msr("classif.acc")`.
 - **task** ([mlr3::TaskClassif](#))
The data task that contains the protected column, only required when the class of object is ([mlr3::PredictionClassif](#))

Value

A 'ggplot2' object.

Protected Attributes

The protected attribute is specified as a `col_role` in the corresponding [mlr3::Task\(\)](#):

```
<Task>$col_roles$pta = "name_of_attribute"
```

This also allows specifying more than one protected attribute, in which case fairness will be considered on the level of intersecting groups defined by all columns selected as a predicted attribute.

Examples

```
library("mlr3")
library("mlr3learners")
library("ggplot2")

# Setup the Fairness measure and tasks
task = tsk("adult_train")$filter(1:500)
learner = lrn("classif.ranger", predict_type = "prob")
fairness_measure = msr("fairness.tpr")

# Example 1 - A single prediction
learner$train(task)
predictions = learner$predict(task)
fairness_accuracy_tradeoff(predictions, fairness_measure, task = task)
```

```
# Example2 - A benchmark
design = benchmark_grid(
  tasks = task,
  learners = lrns(c("classif.featureless", "classif.rpart"),
    predict_type = "prob", predict_sets = c("train", "test")),
  resamplings = rsmpls("cv", folds = 2)
)
bmr = benchmark(design)
fairness_accuracy_tradeoff(bmr, fairness_measure)
```

fairness_prediction_density

Probability Density Plot

Description

Visualizes per-subgroup densities across learners, task and class. The plot is a combination of boxplot and violin plot. The y-axis shows the levels in protected columns. And the x-axis shows the predicted probability. The title for the plot will demonstrate which class for predicted probability.

Usage

```
fairness_prediction_density(object, ...)
```

Arguments

object	(mlr3::PredictionClassif mlr3::ResampleResult mlr3::BenchmarkResult) The binary class prediction object that will be evaluated. If mlr3::PredictionClassif , a mlr3::Task is required.
...	The arguments to be passed to methods, such as: <ul style="list-style-type: none"> task (mlr3::TaskClassif) The data task that contains the protected column. type character The plot type. Either violin or density.

Value

A 'ggplot2' object.

Protected Attributes

The protected attribute is specified as a `col_role` in the corresponding [mlr3::Task\(\)](#):

```
<Task>$col_roles$pta = "name_of_attribute"
```

This also allows specifying more than one protected attribute, in which case fairness will be considered on the level of intersecting groups defined by all columns selected as a predicted attribute.

Examples

```
library("mlr3")
library("mlr3learners")

task = tsk("adult_train")$filter(1:500)
learner = lrn("classif.rpart", predict_type = "prob", cp = 0.001)
learner$train(task)

# For prediction
predictions = learner$predict(task)
fairness_prediction_density(predictions, task)

# For resampling
rr = resample(task, learner, rsmp("cv"))
fairness_prediction_density(rr)
```

fairness_tensor	<i>Compute the Fairness Tensor given a Prediction and a Task</i>
-----------------	--

Description

A fairness tensor is a list of groupwise confusion matrices.

Usage

```
fairness_tensor(object, normalize = "all", ...)

## S3 method for class 'data.table'
fairness_tensor(object, normalize = "all", task, ...)

## S3 method for class 'PredictionClassif'
fairness_tensor(object, normalize = "all", task, ...)

## S3 method for class 'ResampleResult'
fairness_tensor(object, normalize = "all", ...)
```

Arguments

object	(data.table::data.table() mlr3::PredictionClassif mlr3::ResampleResult) A data.table with columns truth and prediction, a mlr3::PredictionClassif or a mlr3::ResampleResult .
normalize	(character) How should the fairness tensor be normalized? "all" normalizes entries by dividing by dataset size, "group" normalizes entries by dividing by group size and "none" does not conduct any normalization at all.
...	any Currently not used.

task (mlr3::TaskClassif)
 A mlr3::TaskClassif. Needs col_role "pta" to be set.

Value

list() of confusion matrix for every group in "pta".

Protected Attributes

The protected attribute is specified as a col_role in the corresponding mlr3::Task():

```
<Task>$col_roles$pta = "name_of_attribute"
```

This also allows specifying more than one protected attribute, in which case fairness will be considered on the level of intersecting groups defined by all columns selected as a predicted attribute.

Examples

```
library("mlr3")
task = tsk("compas")
prediction = lrn("classif.rpart")$train(task)$predict(task)
fairness_tensor(prediction, task = task)
```

groupdiff_tau

Groupwise Operations

Description

groupdiff_tau() computes $\min(x/y, y/x)$, i.e. the smallest symmetric ratio between x and y that is smaller than 1. If x is a vector, the symmetric ratio between all elements in x is computed.

groupdiff_absdiff() computes $\max(\text{abs}(x - y, y - x))$, i.e. the smallest absolute difference between x and y . If x is a vector, the symmetric absolute difference between all elements in x is computed.

Usage

```
groupdiff_tau(x)
```

```
groupdiff_absdiff(x)
```

```
groupdiff_diff(x)
```

Arguments

x (numeric())
 Measured performance in group 1, 2, ...

Value

A single numeric.

Protected Attributes

The protected attribute is specified as a `col_role` in the corresponding `mlr3::Task()`:

```
<Task>$col_roles$pta = "name_of_attribute"
```

This also allows specifying more than one protected attribute, in which case fairness will be considered on the level of intersecting groups defined by all columns selected as a predicted attribute.

Examples

```
groupdiff_tau(1:3)
groupdiff_diff(1:3)
groupdiff_absdiff(1:3)
```

groupwise_metrics	Evaluate a metric on each protected subgroup in a task.
-------------------	---

Description

Instantiates one new measure per protected attribute group in a task. Each metric is then evaluated only on predictions made for the given specific subgroup.

Usage

```
groupwise_metrics(base_measure, task, intersect = TRUE)
```

Arguments

base_measure	(Measure()) The base metric evaluated within each subgroup.
task	<code>mlr3::Task</code> <code>mlr3::Task()</code> to instantiate measures for.
intersect	logical Should multiple pta groups be intersected? Defaults to TRUE. Only relevant if more than one pta columns are provided.

Value

list
List of `mlr3::Measures`.

See Also

[MeasureSubgroup](#)

Examples

```
library("mlr3")
t = tsk("compas")
l = lrn("classif.rpart")
m = groupwise_metrics(msr("classif.acc"), t)
l$train(t)$predict(t)$score(m, t)
```

MeasureFairness

Base Measure for Fairness

Description

This measure extends `mlr3::Measure()` with statistical group fairness: A common approach to quantifying a model's fairness is to compute the difference between a protected and an unprotected group according w.r.t. some performance metric, e.g. classification error (see `mlr3::msr()` with "classif.ce") or false positive rate (see `mlr3::msr()` with "classif.fpr"). The operation for comparison (e.g., difference or quotient) can be specified using the operation parameter, e.g. `groupdiff_absdiff()` or `groupdiff_tau()`.

Composite measures encompassing multiple fairness metrics can be built using [MeasureFairness-Composite](#).

Some popular predefined measures can be found in the [dictionary mlr3::mlr_measures](#).

Protected Attributes

The protected attribute is specified as a `col_role` in the corresponding `mlr3::Task()`:

```
<Task>$col_roles$pta = "name_of_attribute"
```

This also allows specifying more than one protected attribute, in which case fairness will be considered on the level of intersecting groups defined by all columns selected as a predicted attribute.

Super class

```
mlr3::Measure -> MeasureFairness
```

Public fields

```
base_measure (mlr3::Measure())
```

The base measure to be used by the fairness measures, e.g. `mlr3::msr()` with "classif.fpr" for the false positive rate.

```
operation (function())
```

The operation used to compute the difference. A function with args 'x' and 'y' that returns a single value. Defaults to `abs(x - y)`.

Methods

Public methods:

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

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

Usage:

```
MeasureFairness$new(
  id = NULL,
  base_measure,
  operation = groupdiff_absdiff,
  minimize = TRUE,
  range = c(-Inf, Inf)
)
```

Arguments:

`id` (character)

The measure's id. Set to 'fairness.<base_measure_id>' if omitted.

`base_measure` (`Measure()`)

The base metric evaluated within each subgroup.

`operation` (function)

The operation used to compute the difference. A function that returns a single value given input: computed metric for each subgroup. Defaults to [groupdiff_absdiff](#).

`minimize` (`logical()`)

Should the measure be minimized? Defaults to TRUE.

`range` (`numeric(2)`)

Range of the resulting measure. Defaults to `c(-Inf, Inf)`.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

[MeasureFairnessComposite](#)

Examples

```
library("mlr3")
# Create MeasureFairness to measure the Predictive Parity.
t = tsk("adult_train")
learner = lrn("classif.rpart", cp = .01)
learner$train(t)
measure = msr("fairness", base_measure = msr("classif.ppv"))
predictions = learner$predict(t)
predictions$score(measure, task = t)
```

MeasureFairnessComposite

Composite Fairness Measure

Description

Computes a composite measure from multiple fairness metrics and aggregates them using `aggfun` (defaulting to `mean()`).

Protected Attributes

The protected attribute is specified as a `col_role` in the corresponding `mlr3::Task()`:

```
<Task>$col_roles$pta = "name_of_attribute"
```

This also allows specifying more than one protected attribute, in which case fairness will be considered on the level of intersecting groups defined by all columns selected as a predicted attribute.

Super class

```
mlr3::Measure -> MeasureFairnessComposite
```

Methods

Public methods:

- `MeasureFairnessComposite$new()`
- `MeasureFairnessComposite$clone()`

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

Usage:

```
MeasureFairnessComposite$new(
  id = NULL,
  measures,
  aggfun = function(x) mean(x),
  operation = groupdiff_absdiff,
  minimize = TRUE,
  range = c(-Inf, Inf)
)
```

Arguments:

`id` (character(1))

Id of the measure. Defaults to the concatenation of ids in measure.

`measures` (list of `MeasureFairness`)

List of fairness measures to aggregate.

`aggfun` (function())

Aggregation function used to aggregate results from respective measures. Defaults to `sum`.

`operation (function())`

The operation used to compute the difference. A function that returns a single value given input: computed metric for each subgroup. Defaults to `groupdiff_absdiff`. See `MeasureFairness` for more information.

`minimize (logical(1))`

Should the measure be minimized? Defaults to `TRUE`.

`range (numeric(2))`

Range of the resulting measure. Defaults to `c(-Inf, Inf)`.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
library("mlr3")
# Equalized Odds Metric
MeasureFairnessComposite$new(measures = msrs(c("fairness.fpr", "fairness.tpr")))

# Other metrics e.g. based on negative rates
MeasureFairnessComposite$new(measures = msrs(c("fairness.fnr", "fairness.tnr")))
```

MeasureFairnessConstraint

Fairness Constraint Measure

Description

This measure allows constructing for 'constraint' measures of the following form:

$$\min_{\text{performances}} \text{subject to fairness} < \epsilon$$

Protected Attributes

The protected attribute is specified as a `col_role` in the corresponding `mlr3::Task()`:

```
<Task>$col_roles$pta = "name_of_attribute"
```

This also allows specifying more than one protected attribute, in which case fairness will be considered on the level of intersecting groups defined by all columns selected as a predicted attribute.

Super class

```
mlr3::Measure -> MeasureFairnessConstraint
```

Public fields

performance_measure (Measure())
 The performance measure to be used.

fairness_measure (Measure())
 The fairness measure to be used.

epsilon (numeric)
 Deviation from perfect fairness that is allowed.

Methods**Public methods:**

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

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

Usage:

```
MeasureFairnessConstraint$new(
  id = NULL,
  performance_measure,
  fairness_measure,
  epsilon = 0.01,
  range = c(-Inf, Inf)
)
```

Arguments:

id (character)
 The measure's id. Set to 'fairness.<base_measure_id>' if omitted.

performance_measure (Measure())
 The measure used to measure performance (e.g. accuracy).

fairness_measure (Measure())
 The measure used to measure fairness (e.g. equalized odds).

epsilon (numeric)
 Allowed divergence from perfect fairness. Initialized to 0.01.

range (numeric)
 Range of the resulting measure. Defaults to c(-Inf, Inf).

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

See Also

mlr_measures_fairness

Examples

```
# Accuracy subject to equalized odds fairness constraint:
library("mlr3")
t = tsk("adult_train")
learner = lrn("classif.rpart", cp = .01)
learner$train(t)
measure = msr("fairness.constraint", id = "acc_tpr", msr("classif.acc"), msr("fairness.tpr"))
predictions = learner$predict(t)
predictions$score(measure, task = t)
```

MeasureSubgroup	<i>Evaluate a metric on a subgroup</i>
-----------------	--

Description

Allows for calculation of arbitrary `mlr3::Measure()`s on a selected sub-group.

Super class

`mlr3::Measure` -> MeasureSubgroup

Public fields

`base_measure` (`mlr3::Measure()`)
 The base measure to be used by the fairness measures, e.g. `mlr3::msr()` with "classif.fpr" for the false positive rate.

`subgroup` (`character`)(`integer`)
 Subgroup identifier.

`intersect` (`logical`)
 Should groups be intersected?

Methods**Public methods:**

- `MeasureSubgroup$new()`
- `MeasureSubgroup$clone()`

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

Usage:

`MeasureSubgroup$new(id = NULL, base_measure, subgroup, intersect = TRUE)`

Arguments:

`id` (`character`)
 The measure's id. Set to 'fairness.<base_measure_id>' if omitted.

`base_measure` (`Measure()`)
 The measure used to measure fairness.

subgroup (character)|(integer)
 Subgroup identifier. Either value for the protected attribute or position in task\$levels.

intersect [logical](#)
 Should multiple pta groups be intersected? Defaults to TRUE. Only relevant if more than one pta columns are provided.

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

See Also

[MeasureFairness](#), [groupwise_metrics](#)

Examples

```
library("mlr3")
# Create MeasureFairness to measure the Predictive Parity.
t = tsk("adult_train")
learner = lrn("classif.rpart", cp = .01)
learner$train(t)
measure = msr("subgroup", base_measure = msr("classif.acc"), subgroup = "Female")
predictions = learner$predict(t)
predictions$score(measure, task = t)
```

mlr_learners_fairness *Fair Learners in mlr3*

Description

Fair Learners in mlr3

Usage

```
mlr_learners_fairness
```

Format

An object of class `data.table` (inherits from `data.frame`) with 5 rows and 3 columns.

Value

A `data.table` containing an overview of available fair learners.

Predefined measures

mlr3fairness comes with a set of predefined 'fair learners' listed below:

key	package	reference
regr.fairfrm	fairml	Scutari et al., 2021
classif.fairfgrm	fairml	Scutari et al., 2021
regr.fairzlm	fairml	Zafar et al., 2019
classif.fairzlm	fairml	Zafar et al., 2019
regr.fairnclm	fairml	Komiyama et al., 2018

Protected Attributes

The protected attribute is specified as a `col_role` in the corresponding `mlr3::Task()`:

```
<Task>$col_roles$pta = "name_of_attribute"
```

This also allows specifying more than one protected attribute, in which case fairness will be considered on the level of intersecting groups defined by all columns selected as a predicted attribute.

Examples

```
# example code

library("mlr3")
# Available learners:
mlr_learners_fairness
```

mlr_measures_fairness *Fairness Measures in mlr3*

Description

Fairness Measures in mlr3

Usage

```
mlr_measures_fairness
```

Format

An object of class `data.table` (inherits from `data.frame`) with 18 rows and 2 columns.

Value

A `data.table` containing an overview of available fairness metrics.

Predefined measures

mlr3fairness comes with a set of predefined fairness measures as listed below. For full flexibility, **MeasureFairness** can be used to construct classical group fairness measures based on a difference between a performance metrics across groups by combining a performance measure with an operation for measuring differences. Furthermore **MeasureSubgroup** can be used to measure performance in a given subgroup, or alternatively `groupwise_metrics(measure, task)` to instantiate a measure for each subgroup in a **mlr3::Task**.

key	description
fairness.acc	Absolute differences in accuracy across groups
fairness.mse	Absolute differences in mean squared error across groups
fairness.fnr	Absolute differences in false negative rates across groups
fairness.fpr	Absolute differences in false positive rates across groups
fairness.tnr	Absolute differences in true negative rates across groups
fairness.tpr	Absolute differences in true positive rates across groups
fairness.npv	Absolute differences in negative predictive values across groups
fairness.ppv	Absolute differences in positive predictive values across groups
fairness.fomr	Absolute differences in false omission rates across groups
fairness.fp	Absolute differences in false positives across groups
fairness.tp	Absolute differences in true positives across groups
fairness.tn	Absolute differences in true negatives across groups
fairness.fn	Absolute differences in false negatives across groups
fairness.cv	Difference in positive class prediction, also known as Calders-Wevers gap or demographic parity
fairness.eod	Equalized Odds: Mean of absolute differences between true positive and false positive rates across groups
fairness.pp	Predictive Parity: Mean of absolute differences between ppv and npv across groups
fairness.acc_eod=.05	Accuracy under equalized odds < 0.05 constraint
fairness.acc_ppv=.05	Accuracy under ppv difference < 0.05 constraint

Examples

```
library("mlr3")
# Predefined measures:
mlr_measures_fairness$key
```

```
mlr_measures_positive_probability
      Positive Probability Measure
```

Description

Return the probability of a positive prediction, often known as 'Calders-Wevers' gap. This is defined as count of positive predictions divided by the number of observations.

Super class

```
mlr3::Measure -> MeasurePositiveProbability
```

Methods**Public methods:**

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

Method `new()`: Initialize a Measure Positive Probability Object

Usage:

`MeasurePositiveProbability$new()`

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

Usage:

`MeasurePositiveProbability$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Examples

```
library("mlr3")
# Create Positive Probability Measure
t = tsk("adult_train")
learner = lrn("classif.rpart", cp = .01)
learner$train(t)
measure = msr("classif.pp")
predictions = learner$predict(t)
predictions$score(measure, task = t)
```

mlr_pipeops_equalized_odds

Equalized Odds Debiasing

Description

Fairness post-processing method to achieve equalized odds fairness. Works by randomly flipping a subset of predictions with pre-computed probabilities in order to satisfy equalized odds constraints. NOTE: Carefully assess the correct privileged group.

Format

[R6::R6Class](#) object inheriting from [mlr3pipelines::PipeOpTaskPreproc](#)/ [mlr3pipelines::PipeOp](#).

Construction

`PipeOpEOd*$new(id = "eod", param_vals = list())`

- `id` (`character(1)`).
- `param_vals` (`list()`)

Input and Output Channels

Input and output channels are inherited from `mlr3pipelines::PipeOpTaskPreproc`. Instead of a `mlr3::Task`, a `mlr3::TaskClassif` is used as input and output during training and prediction.

The output during training is the input `mlr3::Task`. The output during prediction is a `mlr3::PredictionClassif` with partially flipped predictions.

State

The `$state` is a named list with the `$state` elements inherited from `mlr3pipelines::PipeOpTaskPreproc`.

Parameters

- `alpha` (numeric): A number between 0 (no debiasing) and 1 (full debiasing). Controls the debiasing strength by multiplying the flipping probabilities with `alpha`.
- `privileged` (character): The privileged group.

Fields

Only fields inherited from `mlr3pipelines::PipeOpTaskPreproc/mlr3pipelines::PipeOp`.

Methods

Methods inherited from `mlr3pipelines::PipeOpTaskPreproc/mlr3pipelines::PipeOp`.

Super class

`mlr3pipelines::PipeOp` -> `PipeOpE0d`

Methods

Public methods:

- `PipeOpE0d$new()`
- `PipeOpE0d$clone()`

Method `new()`: Creates a new instance of this `R6::R6Class` R6 class.

Usage:

```
PipeOpE0d$new(id = "E0d", param_vals = list())
```

Arguments:

`id` character

The PipeOps identifier in the PipeOps library.

`param_vals` list

The parameter values to be set. See Parameters.

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

Usage:

```
PipeOpE0d$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

- Hardt M, Price E, Srebro N (2016). “Equality of Opportunity in Supervised Learning.” In *Advances in Neural Information Processing Systems*, volume 29, 3315–3323. <https://papers.nips.cc/paper/2016/file/9d2682367c3935defcb1f9e247a97c0d-Paper.pdf>.
- Pleiss, Geoff, Raghavan, Manish, Wu, Felix, Kleinberg, Jon, Weinberger, Q K (2017). “On Fairness and Calibration.” In Guyon I, Luxburg UV, Bengio S, Wallach H, Fergus R, Vishwanathan S, Garnett R (eds.), *Advances in Neural Information Processing Systems*, volume 30. <https://proceedings.neurips.cc/paper/2017/file/b8b9c74ac526fffb2d39ab038d1cd7-Paper.pdf>.

See Also

<https://mlr3book.mlr-org.com/list-pipeops.html>

Other PipeOps: [mlr_pipeops_explicit_pta](#), [mlr_pipeops_reweighing](#)

Examples

```
library("mlr3")
library("mlr3pipelines")

eod = po("E0d")
learner_po = po("learner_cv",
  learner = lrn("classif.rpart"),
  resampling.method = "insample"
)

task = tsk("compas")
graph = learner_po %>>% eod
glrn = GraphLearner$new(graph)
glrn$train(task)

# On a Task
glrn$predict(task)

# On newdata
glrn$predict_newdata(task$data(cols = task$feature_names))
```

```
mlr_pipeops_explicit_pta
```

```
  PipeOpExplicitPta
```

Description

Turns the column with column role ‘pta’ into an explicit separate column prefixed with “*.internal_pta*”. This keeps it from getting changed or adapted by subsequent pipelines that operate on the feature pta.

Format

[R6::R6Class](#) object inheriting from [mlr3pipelines::PipeOpTaskPreproc/mlr3pipelines::PipeOp](#).

Construction

```
PipeOpExplicitPta$new(id = "reweighing", param_vals = list())
```

- `id` (`character(1)`).
- `param_vals` (`list()`)

Input and Output Channels

Input and output channels are inherited from [mlr3pipelines::PipeOpTaskPreproc](#). Instead of a [mlr3::Task](#), a [mlr3::TaskClassif](#) is used as input and output during training and prediction.

The output during training is the input [mlr3::Task](#) with added weights column according to target class. The output during prediction is the unchanged input.

State

The `$state` is a named `list` with the `$state` elements inherited from [mlr3pipelines::PipeOpTaskPreproc](#).

Parameters

The `PipeOp` does not have any hyperparameters.

Internals

Copies the existing `pta` column to a new column.

Fields

Only fields inherited from [mlr3pipelines::PipeOpTaskPreproc/mlr3pipelines::PipeOp](#).

Methods

Methods inherited from [mlr3pipelines::PipeOpTaskPreproc/mlr3pipelines::PipeOp](#).

Super classes

```
mlr3pipelines::PipeOp -> mlr3pipelines::PipeOpTaskPreproc -> PipeOpExplicitPta
```

Methods**Public methods:**

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

Method `new()`: Creates a new instance of this [R6::R6Class](#) R6 class.

Usage:

```
PipeOpExplicitPta$new(id = "explicit_pta", param_vals = list())
```

Arguments:

id character

The PipeOps identifier in the PipeOps library.

param_vals list

The parameter values to be set. See Parameters.

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

See Also

<https://mlr3book.mlr-org.com/list-pipeops.html>

Other PipeOps: [mlr_pipeops_equalized_odds](#), [mlr_pipeops_reweighing](#)

Examples

```
library("mlr3")
library("mlr3pipelines")
epta = po("explicit_pta")
new = epta$train(list(tsk("adult_train")))
```

mlr_pipeops_reweighing

Reweighting to balance disparate impact metric

Description

Adjusts class balance and protected group balance in order to achieve fair(er) outcomes.

Format

R6::R6Class object inheriting from [mlr3pipelines::PipeOpTaskPreproc/mlr3pipelines::PipeOp](#).

PipeOpReweightingWeights

Adds a class weight column to the [mlr3::Task](#) that different [mlr3::Learners](#) may be using. In case initial weights are present, those are multiplied with new weights. Caution: Only fairness tasks are supported. Which means tasks need to have protected field. `tskcol_rolespta`.

PipeOpReweightingOversampling

Oversamples a [mlr3::Task](#) for more balanced ratios in subgroups and protected groups. Can be used if a learner does not support weights. Caution: Only fairness tasks are supported. Which means tasks need to have protected field. `tskcol_rolespta`.

Construction

```
PipeOpReweighing$new(id = "reweighing", param_vals = list())
```

- `id` (`character(1)`).
- `param_vals` (`list()`)

Input and Output Channels

Input and output channels are inherited from `mlr3pipelines::PipeOpTaskPreproc`. Instead of a `mlr3::Task`, a `mlr3::TaskClassif` is used as input and output during training and prediction.

The output during training is the input `mlr3::Task` with added weights column according to target class. The output during prediction is the unchanged input.

State

The `$state` is a named list with the `$state` elements inherited from `mlr3pipelines::PipeOpTaskPreproc`.

Parameters

- `alpha` (`numeric()`): A number between 0 (no debiasing) and 1 (full debiasing).

Internals

Introduces, or overwrites, the "weights" column in the `mlr3::Task`. However, the `mlr3::Learner` method needs to respect weights for this to have an effect.

The newly introduced column is named `reweighing.WEIGHTS`; there will be a naming conflict if this column already exists and is *not* a weight column itself.

Fields

Only fields inherited from `mlr3pipelines::PipeOpTaskPreproc/mlr3pipelines::PipeOp`.

Methods

Methods inherited from `mlr3pipelines::PipeOpTaskPreproc/mlr3pipelines::PipeOp`.

Super classes

```
mlr3pipelines::PipeOp -> mlr3pipelines::PipeOpTaskPreproc -> PipeOpReweighingWeights
```

Methods**Public methods:**

- `PipeOpReweighingWeights$new()`
- `PipeOpReweighingWeights$clone()`

Method `new()`: Creates a new instance of this `R6::R6Class` R6 class.

Usage:

```
PipeOpReweightingWeights$new(id = "reweighing_wts", param_vals = list())
```

Arguments:

id character

The PipeOps identifier in the PipeOps library.

param_vals list

The parameter values to be set.

- alpha: controls the proportion between initial weight (1 if non existing) and reweighing weight. Defaults to 1. Here is how it works: $\text{new_weight} = (1 - \alpha) * 1 + \alpha \times \text{reweighing_weight}$ $\text{final_weight} = \text{old_weight} * \text{new_weight}$

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Super classes

```
mlr3pipelines::PipeOp -> mlr3pipelines::PipeOpTaskPreproc -> PipeOpReweightingOversampling
```

Methods

Public methods:

- `PipeOpReweightingOversampling$new()`
- `PipeOpReweightingOversampling$clone()`

Method new():

Usage:

```
PipeOpReweightingOversampling$new(id = "reweighing_os", param_vals = list())
```

Arguments:

id 'character'

The PipeOp's id.

param_vals 'list'

A list of parameter values.

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

References

Kamiran, Faisal, Calders, Toon (2012). "Data preprocessing techniques for classification without discrimination." *Knowledge and Information Systems*, **33**(1), 1–33.

See Also

<https://mlr3book.mlr-org.com/list-pipeops.html>

Other PipeOps: [mlr_pipeops_equalized_odds](#), [mlr_pipeops_explicit_pta](#)

Examples

```
library("mlr3")
library("mlr3pipelines")

reweighing = po("reweighing_wts")
learner_po = po("learner", learner = lrn("classif.rpart"))

data = tsk("adult_train")
graph = reweighing %>% learner_po
glrn = GraphLearner$new(graph)
glrn$train(data)
tem = glrn$predict(data)
tem$confusion
```

report_datasheet

Create a Datasheet for Documenting a Dataset

Description

Creates a new **rmarkdown** template with a skeleton questionnaire for dataset documentation. Uses the awesome markdown template created by Chris Garbin [from Github](#).

Usage

```
report_datasheet(filename = "datasheet.Rmd", edit = FALSE, build = FALSE)
```

Arguments

filename	(character(1)) File path or name for new file that should be created.
edit	(logical(1)) TRUE to edit the template immediately.
build	(logical(1)) Should the report be built after creation? Initialized to FALSE.

Value

Invisibly returns the path to the newly created file(s).

References

Gebu, Timnit, Morgenstern, Jamie, Vecchione, Briana, Vaughan, Wortman J, Wallach, Hanna, III D, Hal, Crawford, Kate (2018). “Datasheets for datasets.” *arXiv preprint arXiv:1803.09010*.

See Also

Other fairness_reports: [report_fairness\(\)](#), [report_modelcard\(\)](#)

Examples

```
report_file = tempfile()
report_datasheet(report_file)
```

report_fairness	Create a Fairness Report
-----------------	--------------------------

Description

Creates a new **rmarkdown** template with a skeleton of reported metrics and visualizations. Uses the awesome markdown template created by Chris Garbin [from Github](#).

Usage

```
report_fairness(
  filename = "fairness_report.Rmd",
  objects,
  edit = FALSE,
  check_objects = FALSE,
  build = FALSE
)
```

Arguments

filename	(character(1)) File path or name for new file that should be created.
objects	(list()) A named list of objects required for the fairness report. Objects are saved as <name>.rds in the new folder created for the report. <ul style="list-style-type: none"> task :: The mlr3::Task a report should be created for. resample_result :: A mlr3::ResampleResult result to be analyzed. ... :: any other objects passed on for the report.
edit	(logical(1)) TRUE to edit the template immediately.

check_objects	(logical(1)) Should items in objects be checked? If FALSE, no checks on object are performed.
build	(logical(1)) Should the report be built after creation? Initialized to FALSE.

Value

Invisibly returns the path to the newly created file(s).

See Also

Other fairness_reports: [report_datasheet\(\)](#), [report_modelcard\(\)](#)

Examples

```
library("mlr3")
report_file = tempfile()
task = tsk("compas")
learner = lrn("classif.rpart", predict_type = "prob")
rr = resample(task, learner, rsmpl("cv", folds = 3L))
report_fairness(report_file, list(task = task, resample_result = rr))
```

report_modelcard	<i>Create a Modelcard</i>
------------------	---------------------------

Description

Creates a new **markdown** template with a skeleton questionnaire for a model card. Uses the awesome markdown template created by Chris Garbin [from Github](#).

Usage

```
report_modelcard(filename = "modelcard.Rmd", edit = FALSE, build = FALSE)
```

Arguments

filename	(character(1)) File path or name for new file that should be created.
edit	(logical(1)) TRUE to edit the template immediately.
build	(logical(1)) Should the report be built after creation? Initialized to FALSE.

Value

Invisibly returns the path to the newly created file(s).

References

Mitchell, Margaret, Wu, Simone, Zaldivar, Andrew, Barnes, Parker, Vasserman, Lucy, Hutchinson, Ben, Spitzer, Elena, Raji, Deborah I, Gebru, Timnit (2019). “Model cards for model reporting.” In *Proceedings of the conference on fairness, accountability, and transparency*, 220–229.

See Also

Other fairness_reports: [report_datasheet\(\)](#), [report_fairness\(\)](#)

Examples

```
report_file = tempfile()
report_modelcard(report_file)
```

task_summary	<i>Task summary for fairness report</i>
--------------	---

Description

Create the general task documentation in a dataframe for fairness report. The information includes

- Audit Date
- Task Name
- Number of observations
- Number of features
- Target Name
- Feature Names
- The Protected Attribute

Usage

```
task_summary(task)
```

Arguments

task [mlr3::Task](#)

Value

data.frame containing the reported information

Examples

```
library("mlr3")
task_summary(tsk("adult_train"))
```

Index

- * **PipeOps**
 - mlr_pipeops_equalized_odds, 23
 - mlr_pipeops_explicit_pta, 25
 - mlr_pipeops_reweighing, 27
- * **datasets**
 - mlr_learners_fairness, 20
 - mlr_measures_fairness, 21
- * **data**
 - adult, 2
 - compas, 5
- * **fairness_reports**
 - report_datasheet, 30
 - report_fairness, 31
 - report_modelcard, 32
- adult, 2
- adult_test(adult), 2
- adult_train(adult), 2
- character, 10
- compare_metrics, 4
- Compas(compas), 5
- compas, 5, 5
- compute_metrics, 7
- data.table::data.table(), 11
- dictionary, 14
- fairness_accuracy_tradeoff, 8
- fairness_prediction_density, 10
- fairness_tensor, 11
- groupdiff_absdiff, 15
- groupdiff_absdiff(groupdiff_tau), 12
- groupdiff_absdiff(), 14
- groupdiff_diff(groupdiff_tau), 12
- groupdiff_tau, 12
- groupdiff_tau(), 14
- groupwise_metrics, 13, 20
- logical, 13, 20
- mean(), 16
- MeasureFairness, 14, 16, 20, 22
- MeasureFairnessComposite, 14, 15, 16
- MeasureFairnessConstraint, 17
- MeasurePositiveProbability
 - (mlr_measures_positive_probability), 22
- MeasureSubgroup, 13, 19, 22
- mlr3::BenchmarkResult, 4, 9, 10
- mlr3::Learner, 27, 28
- mlr3::Measure, 4, 9, 13, 14, 16, 17, 19, 22
- mlr3::Measure(), 14, 19
- mlr3::mlr_measures, 14
- mlr3::msr(), 14, 19
- mlr3::PredictionClassif, 4, 9–11, 24
- mlr3::ResampleResult, 4, 9–11, 31
- mlr3::Task, 9, 10, 13, 22, 24, 26–28, 31, 33
- mlr3::Task(), 4, 8–10, 12–14, 16, 17, 21
- mlr3::TaskClassif, 4, 6, 9, 10, 12, 24, 26, 28
- mlr3pipelines::PipeOp, 23, 24, 26–29
- mlr3pipelines::PipeOpTaskPreproc, 23, 24, 26–29
- mlr_learners_fairness, 20
- mlr_measures_fairness, 21
- mlr_measures_positive_probability, 22
- mlr_pipeops_equalized_odds, 23, 27, 30
- mlr_pipeops_explicit_pta, 25, 25, 30
- mlr_pipeops_reweighing, 25, 27, 27
- msr(classif.acc), 9
- PipeOpEOd(mlr_pipeops_equalized_odds), 23
- PipeOpExplicitPta
 - (mlr_pipeops_explicit_pta), 25
- PipeOpReweighingOversampling
 - (mlr_pipeops_reweighing), 27
- PipeOpReweighingWeights
 - (mlr_pipeops_reweighing), 27
- R6, 15, 16, 18, 19

R6::R6Class, [6](#), [23](#), [24](#), [26–28](#)
report_datasheet, [30](#), [32](#), [33](#)
report_fairness, [31](#), [31](#), [33](#)
report_modelcard, [31](#), [32](#), [32](#)

task_summary, [33](#)