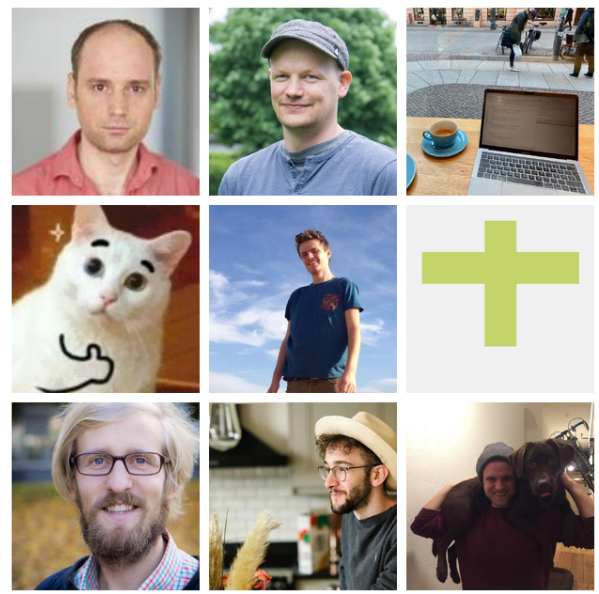
Modern Machine Learning in R

https://mlr-org.com/

https://github.com/mlr-org



Michel Lang, Bernd Bischl, Jakob Richter, Martin Binder, Marc Becker, Patrick Schratz, Raphael Sonabend, Lennart Schneider and more!

April 14, 2021

Intro

STRUCTURE

- Motivation: Why do we want to use mlr3?
- Key principles of mlr3
- The mlr3verse:
  - mlr3pipelines: preprocessing and combination
  - mlr3tuning: hyperparameter tuning
  - mlr3fselect: feature selection
  - mlr3proba: survival time prediction

WHAT ARE YOU SUPPOSED TO TAKE AWAY

- Know the most important building blocks in mlr3
- Have an overview of most technical possibilities in mlr3
- Know how the different mlr3verse packages interact
- Know where to find help

MOTIVATION: WHY DO WE WANT TO USE MLR3?

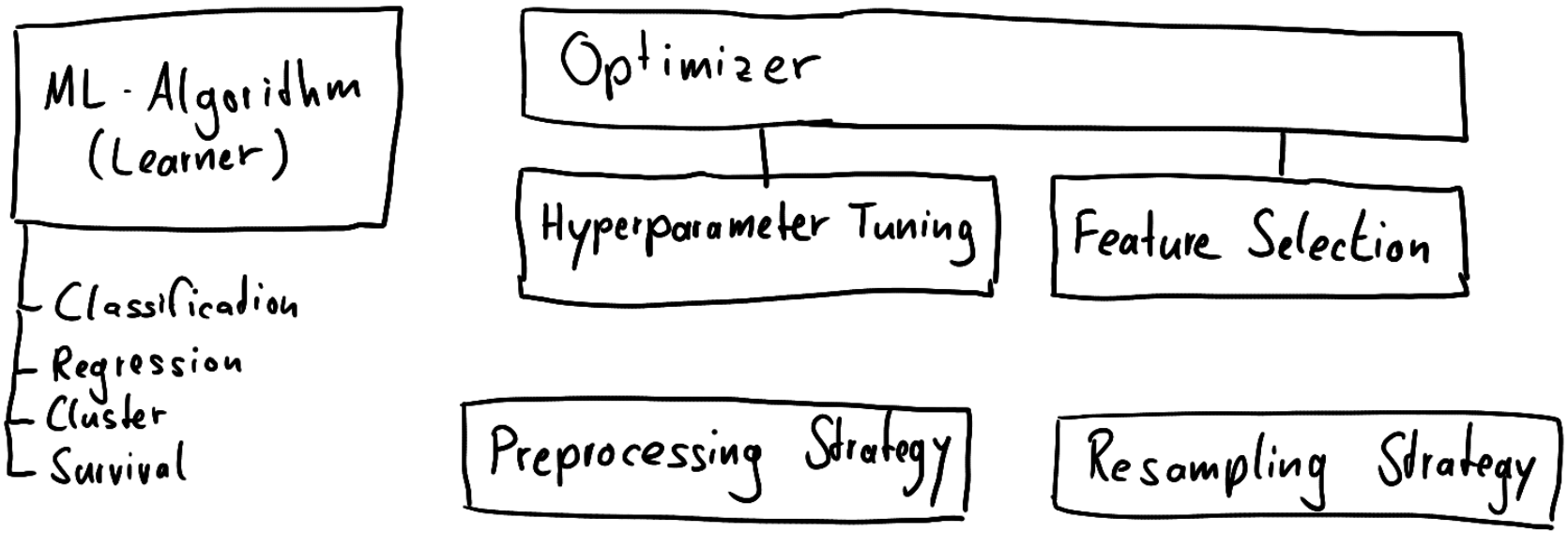
As researchers we want to efficiently benchmark

- many methods
- on many datasets.

And: We want our methods to be easily available for a wide audience.

Note: mlr3 is designed to be extendable by other packages!

METHODS?


ML - Algorithm (Learner)
Classification
Regression
Cluster
Survival
Optimizer
Hyperparameter Tuning
Feature Selection
Preprocessing Strategy
Resampling Strategy

MOTIVATION: MAKING BENCHMARKS EASY!

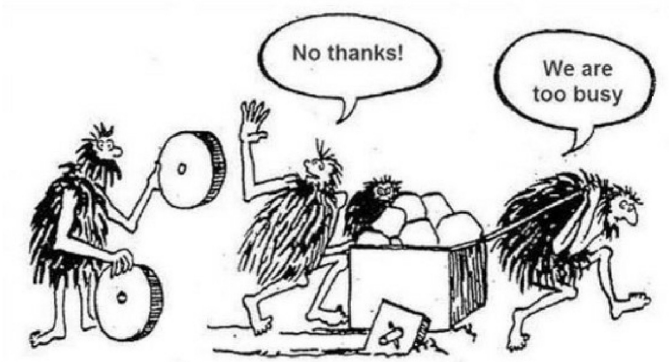
By unifying

- interface to train and predict methods,
- interface to learner's hyperparameters,
- interface to optimizers,
- resampling,
- preprocessing independently from the data,
- parallelization, and
- error handling

methods can be used interchangeably and can be easily benchmarked.

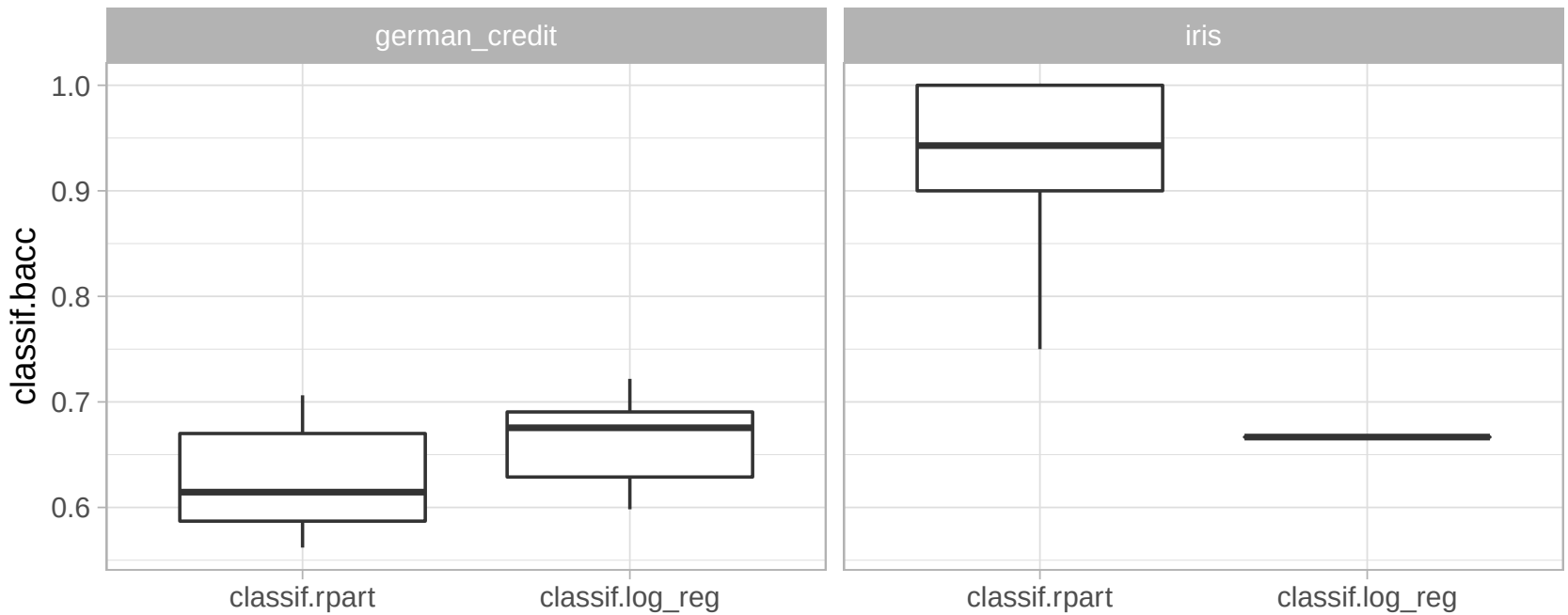
IS IT WORTH TO "LEARN" MLR3

- Avoid making mistakes by relying on tested functionality
  - predefined performance measures
  - resampling
  - ...
- Easily scale up your benchmark
  - integrated parallelization
  - benchmarking functions
  - soon on clusters: `batchtools` + `mlr3batchmark`
- New methods can be easily integrated in the mlr3verse



MLR3: A SHORT EXAMPLE

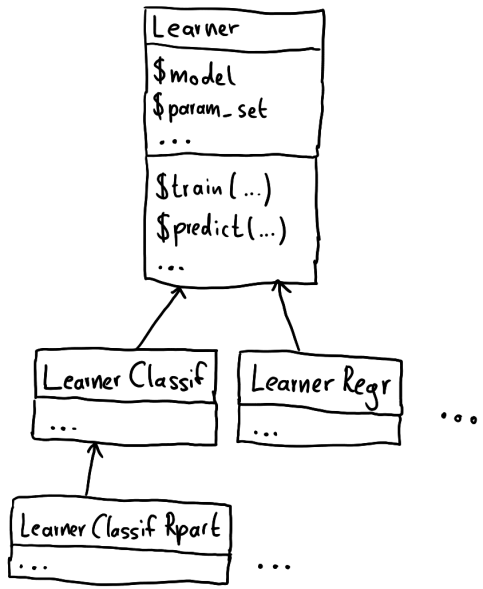
```
library(mlr3verse)
tasks = list(
  as_task_classif(iris, target = "Species"), # task from df
  tsk("german_credit") # example task
)
learners = lrns(c("classif.rpart", "classif.log_reg"))
bmg = benchmark_grid(tasks, learners, rsmp("cv"))
bmr = benchmark(bmg)
autoplot(bmr, measure = msr("classif.bacc")) # balanced accuracy
```



Principles of mlr3

MLR3 PHILOSOPHY

- Overcome limitations of S3 with the help of **R6**
  - Truly object-oriented: data and methods live in the same object
  - Make use of inheritance
  - Make slight use of reference semantics
- Embrace **data.table**, both for arguments and internally
  - Fast operations for tabular data
  - List columns to arrange complex objects in tabular structure
- Be **light on dependencies**:
  - `R6`, `data.table`, `lgr`, `...`
  - Plus some of our own packages (`backports`, `checkmate`, `...`)
  - Special packages are loaded from mlr3 extension libraries



MLR3: OBJECTS AND FUNCTIONS

User created objects:

- Tasks: data + meta information
- Learner: ml algorithm + hyperparameter + model
- Measure: formula + meta information
- Resampling: strategy (+ indices)

Further objects:

- Prediction, ResampleResult, BenchmarkResult, ...

Functions to create objects:

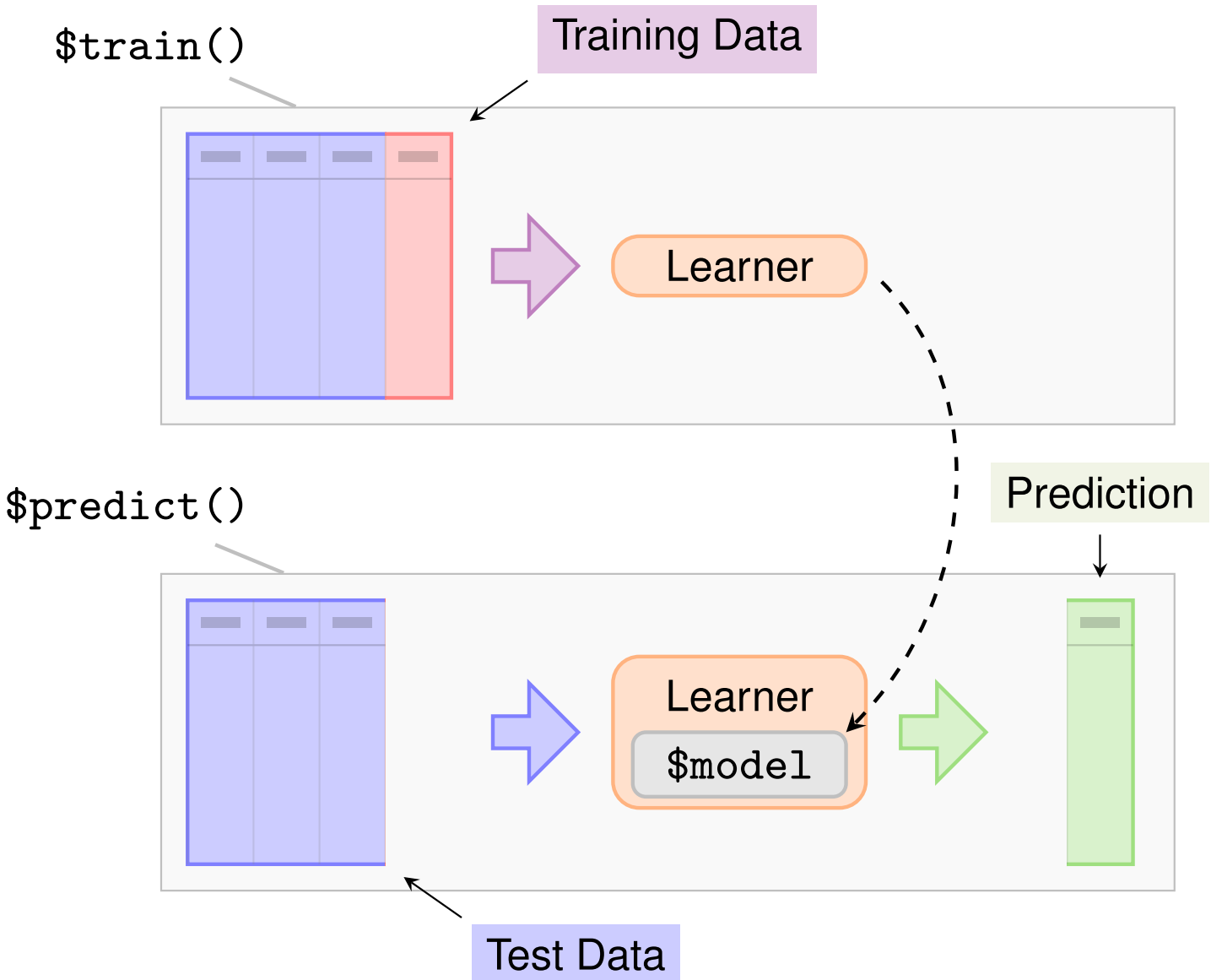
- `tsk()`, `as_task_*()`: Task
- `lrn()`, `lrns()`: Learners
- `msr()`, `msrs()`: Measures
- `rsmp()`: Resampling strategies

Hint: calling e.g. just `lrns()` prints all available learners in the `mlr_learners` dictionary. Dictionaries can get populated by add-on packages (e.g. `mlr3extralearners`)

Functions:

- `resample()`
- `benchmark_grid()` + `benchmark()`
- `future::plan()`: enables parallelization
- `mlr3viz::autoplot()`: visualizes mlr3 objects

LEARNING ALGORITHMS



TRAIN, PREDICT, SCORE

```
task = as_task_classif(iris, target = "Species")
```

Objects have *fields* that contain information about the object.

```
c(task$nrow, task$ncol)

#> [1] 150   5
```

Some fields of R6-objects may be *"Active Bindings"*. Internally they are realized as functions that are called whenever the value is set or retrieved.

```
task$nrow = 3

#> Error:  Field/Binding is read-only
```

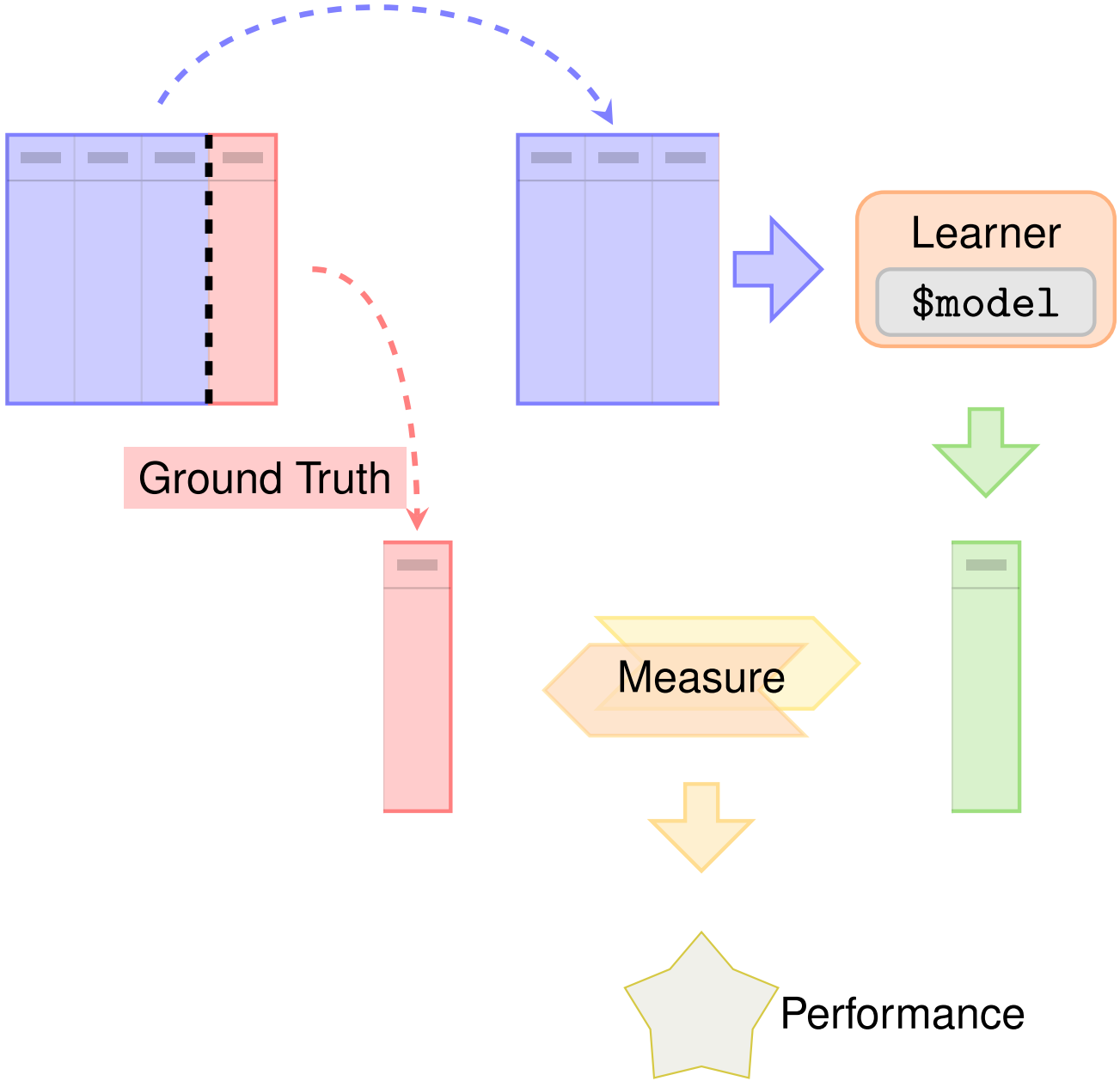
Others can be overwritten:

```
learner = lrn("classif.fnn") # fast k nearest neighbors
learner$param_set

#> <ParamSet>
#>           id    class lower upper nlevels default  value
#>       <char>   <char> <num> <num>   <num>  <list> <list>
#> 1:         k ParamInt     1   Inf     Inf       1
#> 2: algorithm ParamFct    NA    NA       3 kd_tree

learner$param_set$values$k = 4
```

TRAIN, PREDICT, SCORE

```
str(learner$model)

#>  NULL

learner$train(task, row_ids = (1:75) * 2 - 1)
str(learner$model)

#> List of 2
#>  $ train:Classes 'data.table' and 'data.frame':
75 obs. of  4 variables:
#>   ..$ Petal.Length: num [1:75] 1.4 1.3 1.4 1.4 1.4 1.5 1.4 1.2 1.3 1.7 ...
#>   ..$ Petal.Width : num [1:75] 0.2 0.2 0.2 0.3 0.2 0.2 0.1 0.2 0.4 0.3 ...
#>   ..$ Sepal.Length: num [1:75] 5.1 4.7 5 4.6 4.4 5.4 4.8 5.8 5.4 5.7 ...
#>   ..$ Sepal.Width : num [1:75] 3.5 3.2 3.6 3.4 2.9 3.7 3 4 3.9 3.8 ...
#>   ..- attr(*, ".internal.selfref")=<externalptr>
#>  $ cl    : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 .

pred = learner$predict(task, row_ids = (1:75) * 2)
```

PERFORMANCE EVALUATION

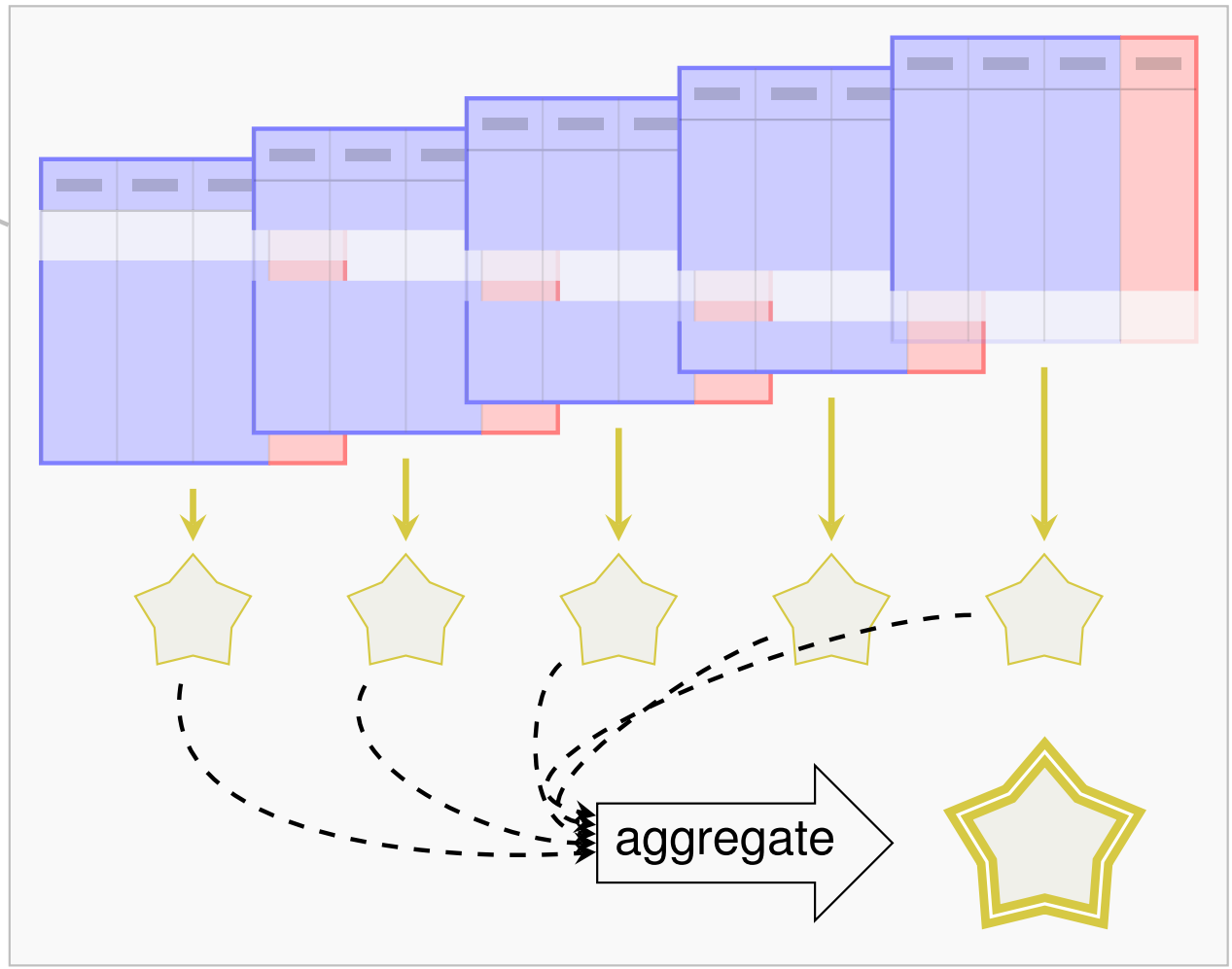


TRAIN, PREDICT, SCORE

```
pred$confusion

#>             truth
#> response     setosa versicolor virginica
#>   setosa         25          0         0
#>   versicolor      0         24         3
#>   virginica       0          1        22

pred$score()

#> classif.ce
#>      0.053

pred$score(msr("classif.bacc"))

#> classif.bacc
#>         0.95

head(as.data.table(pred), 4)

#>    row_ids  truth response
#>      <int> <fctr>   <fctr>
#> 1:       2 setosa   setosa
#> 2:       4 setosa   setosa
#> 3:       6 setosa   setosa
#> 4:       8 setosa   setosa
```

RESAMPLING

`resample()`



RESAMPLING

- Resample description: How to split the data

```
cv5 = rsmp("cv", folds = 5)
```

- Use the `resample()` function for resampling:

```
task = tsk("iris")
learner = lrn("classif.rpart")
rr = resample(task, learner, cv5, store_models = TRUE)
```

(`store_models = TRUE` so we can access models in `rr` later.)

- We get a `ResamplingResult` object:

```
print(rr)
#> <ResampleResult> of 5 iterations
#> * Task: iris
#> * Learner: classif.rpart
#> * Warnings: 0 in 0 iterations
#> * Errors: 0 in 0 iterations
```

RESAMPLING

- Predictions of individual folds

```
predictions = rr$predictions()
predictions[[1]]
```

```
#> <PredictionClassif> for 30 observations:
#>     row_ids     truth  response
#>           5    setosa    setosa
#>          12    setosa    setosa
#>          14    setosa    setosa
#> ---
#>         142 virginica virginica
#>         144 virginica virginica
#>         149 virginica virginica
```

RESAMPLING

- Predictions of individual folds

```
predictions = rr$predictions()
predictions[[1]]
```

```
#> <PredictionClassif> for 30 observations:
#>     row_ids     truth  response
#>           5    setosa    setosa
#>          12    setosa    setosa
#>          14    setosa    setosa
#> ---
#>         142 virginica virginica
#>         144 virginica virginica
#>         149 virginica virginica
```

- Score of individual folds

```
scores = rr$score()
scores[1:3, c("iteration", "classif.ce")]
```

```
#>    iteration classif.ce
#>        <int>      <num>
#> 1:         1      0.033
#> 2:         2      0.067
#> 3:         3      0.033
```

RESAMPLING

- Access to models of individual folds (only if `$store_models = TRUE`)

```
rr$learners[[1]]$importance()
#>  Petal.Width Petal.Length Sepal.Length  Sepal.Width
#>           71           65           43           24
```

RESAMPLING

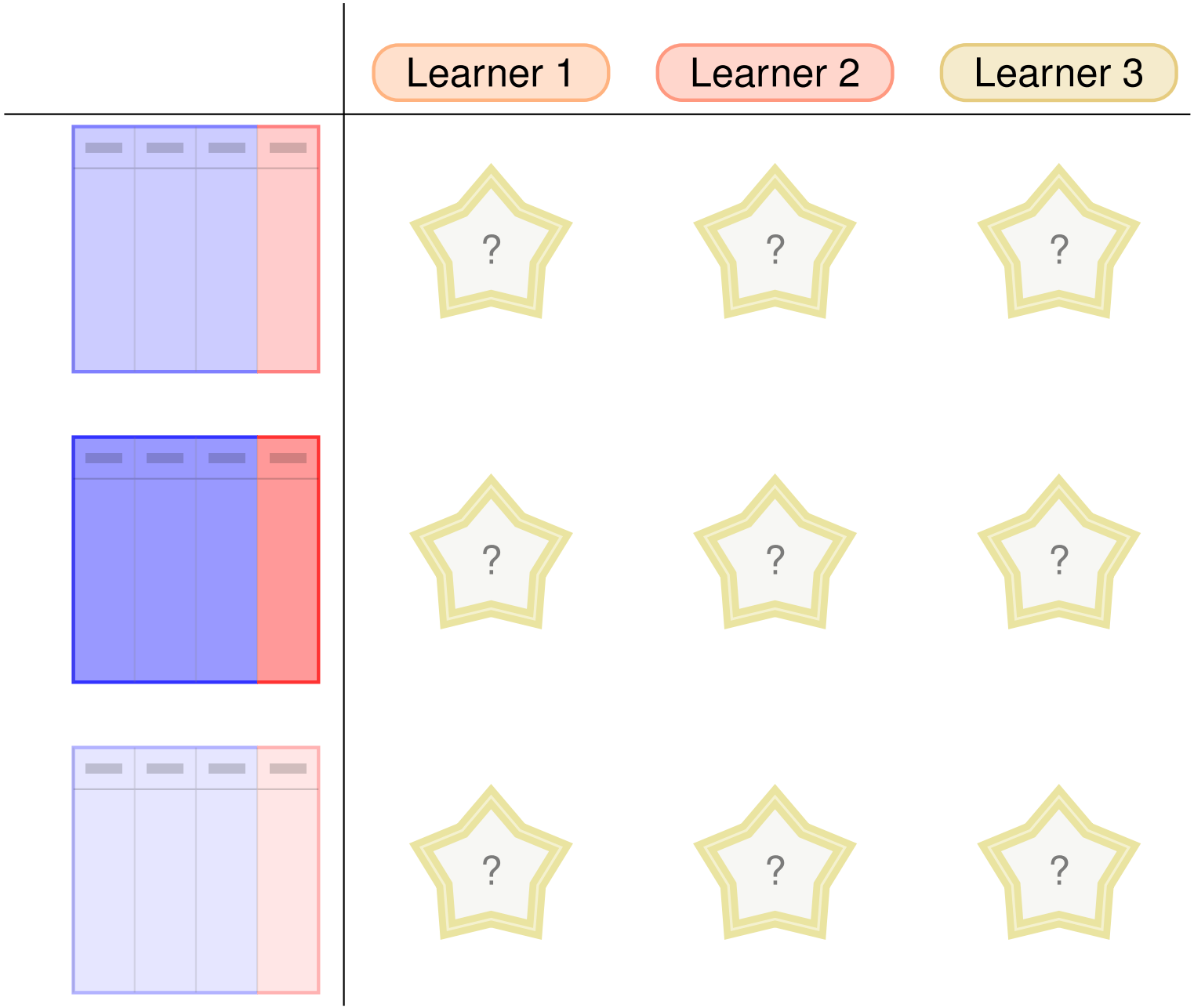
- Access to models of individual folds (only if `$store_models = TRUE`)

```
rr$learners[[1]]$importance()
#>  Petal.Width Petal.Length Sepal.Length  Sepal.Width
#>           71           65           43           24
```

- Aggregate over multiple folds:

```
sapply(rr$learners, function(x) x$importance()) %>%
  apply(1, mean)
#>  Petal.Width Petal.Length Sepal.Length  Sepal.Width
#>           71           65           44           28
```

PERFORMANCE COMPARISON: BENCHMARK



PERFORMANCE COMPARISON

- Multiple Learners, multiple Tasks:

```
library("mlr3learners")
learners = list(lrn("classif.rpart"), lrn("classif.kknn"))
tasks = list(tsk("iris"), tsk("sonar"), tsk("wine"))
```

- Set up the *design* and execute benchmark:

```
design = benchmark_grid(tasks, learners, cv5)
bmr = benchmark(design)
```

- We get a `BenchmarkResult` object which shows that `kknn` outperforms `rpart`:

```
bmr_ag = bmr$aggregate()
bmr_ag[, c("task_id", "learner_id", "classif.ce")]
#>    task_id    learner_id classif.ce
#>     <char>        <char>      <num>
#> 1:    iris classif.rpart      0.053
#> 2:    iris  classif.kknn      0.040
#> 3:   sonar classif.rpart      0.317
#> 4:   sonar  classif.kknn      0.158
#> 5:    wine classif.rpart      0.124
#> 6:    wine  classif.kknn      0.040
```

BENCHMARK RESULT

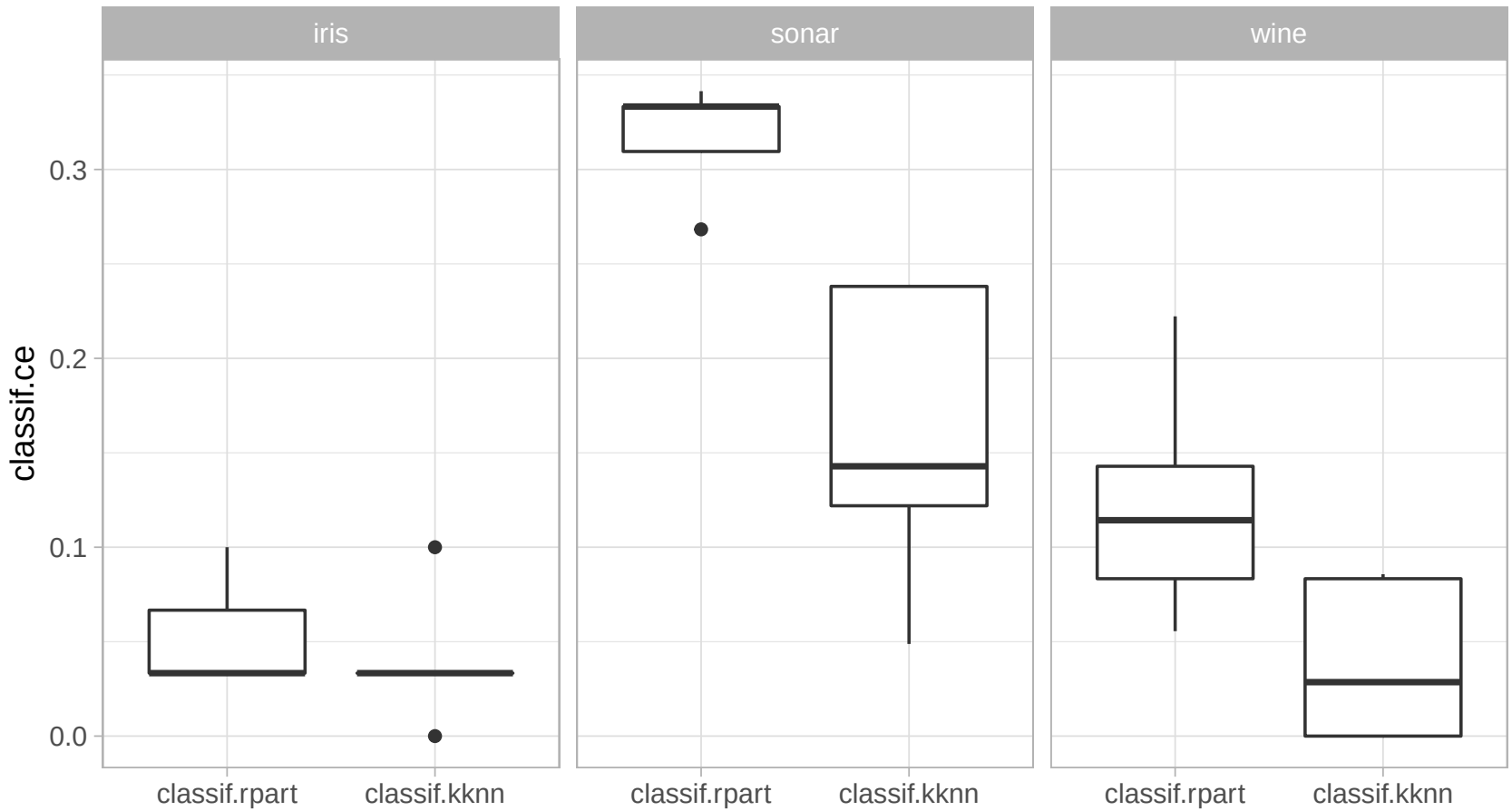
What exactly is a `BenchmarkResult` object?
Just like `Prediction` and `ResamplingResult`!

- Table representation using `as.data.table()`
- Active bindings and functions that make information easily accessible

BENCHMARK RESULT

The `mlr3viz` package contains `autoplot()` functions for many mlr3 objects

```
library(mlr3viz)
autoplot(bmr)
```



BENCHMARK RESULT

Many objects can be transformed into a `data.table()`.

```
as.data.table(bmr)[1:4, -1]

#                 task                     learner         resampling
#               <list>                      <list>             <list>
# 1: <TaskClassif[46]> <LearnerClassifRpart[34]> <ResamplingCV[19]>
# 2: <TaskClassif[46]> <LearnerClassifRpart[34]> <ResamplingCV[19]>
# 3: <TaskClassif[46]> <LearnerClassifRpart[34]> <ResamplingCV[19]>
# 4: <TaskClassif[46]> <LearnerClassifRpart[34]> <ResamplingCV[19]>
#    iteration              prediction
#        <int>                  <list>
# 1:         1 <PredictionClassif[19]>
# 2:         2 <PredictionClassif[19]>
# 3:         3 <PredictionClassif[19]>
# 4:         4 <PredictionClassif[19]>
```

CONTROL OF EXECUTION

Parallelization

```
future::plan("multicore")
```

- runs each resampling iteration as a job
- also allows nested resampling

Encapsulation

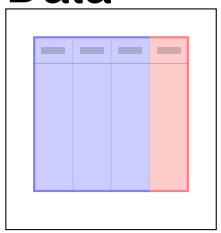
```
learner$encapsulate = c(train = "callr", predict = "callr")
```

- Spawns a separate R process to train the learner
- Learner may segfault without tearing down the session
- Logs are captured
- Possibilty to have a fallback to create predictions

MLR3: SHORT RECAP

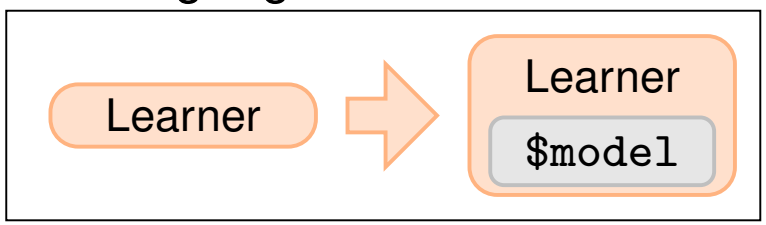
Ingredients:

Data



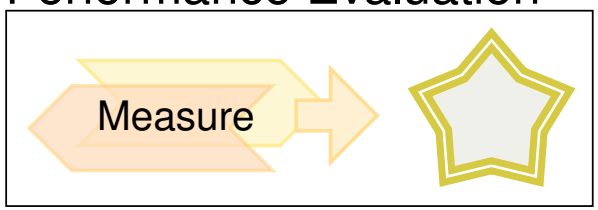
`TaskClassif`,
`TaskRegr`,
`tsk()`

Learning Algorithms



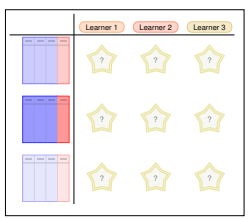
`lrn()` ⇒ `Learner`,
↪`Learner$train()`,
↪`Learner$predict()` ⇒ `Prediction`

Performance Evaluation



`rsmp()` ⇒ `Resampling`,
`msr()` ⇒ `Measure`,
`resample()` ⇒ `ResamplingResult`,
↪ `ResamplingResult$score()`,
↪ `ResamplingResult$aggregate()`

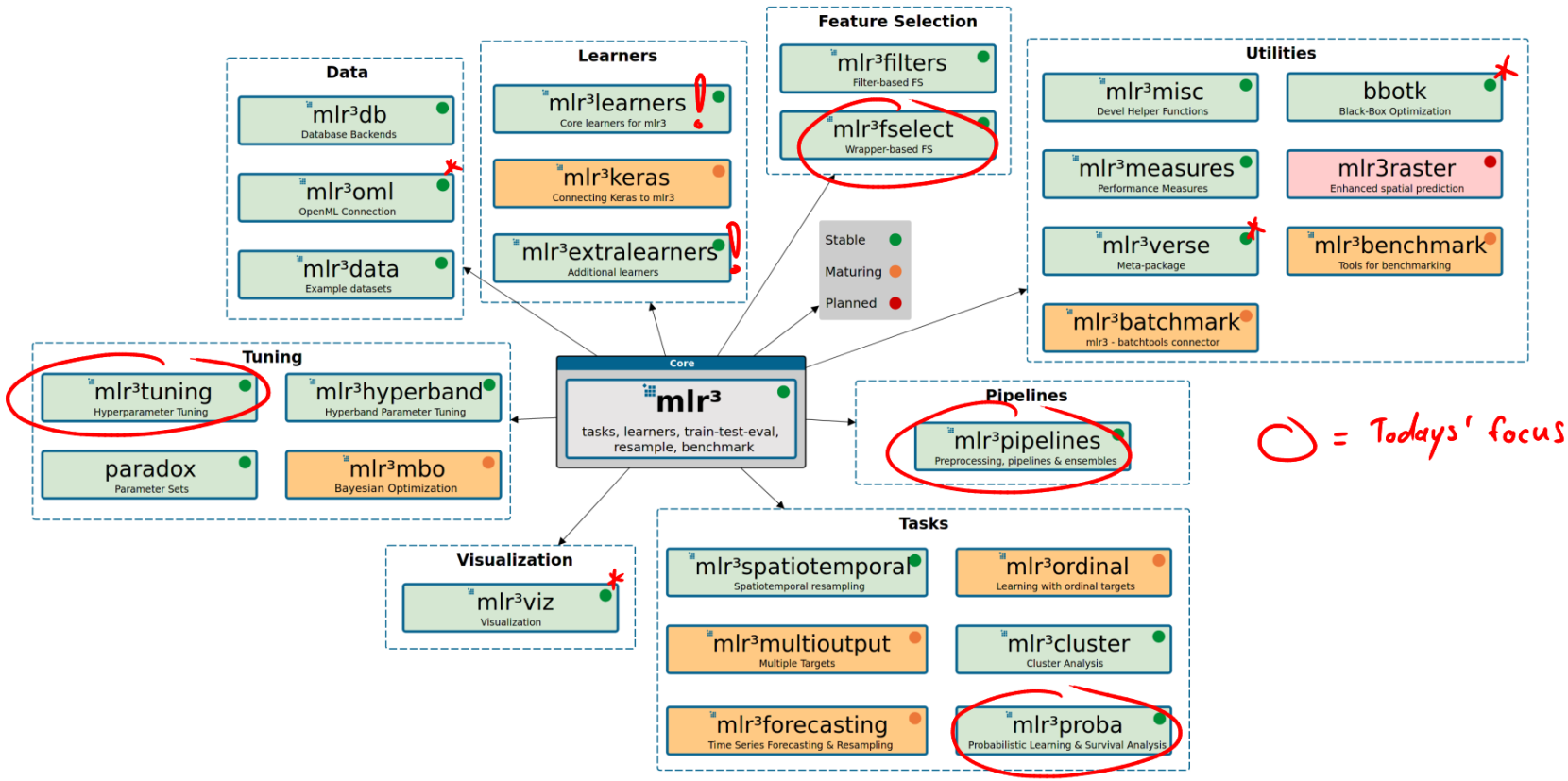
Performance Comparison



`benchmark_grid()`,
`benchmark()` ⇒ `BenchmarkResult`

mlr3verse

THE MLR3VERSE



mlr3pipelines

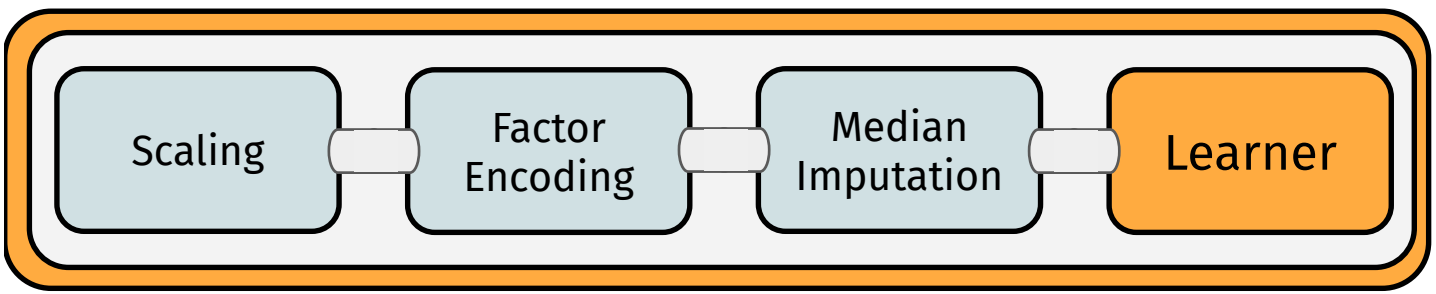
MLR3PIPELINES

Main author: Martin Binder (LMU)

Machine Learning Workflows:

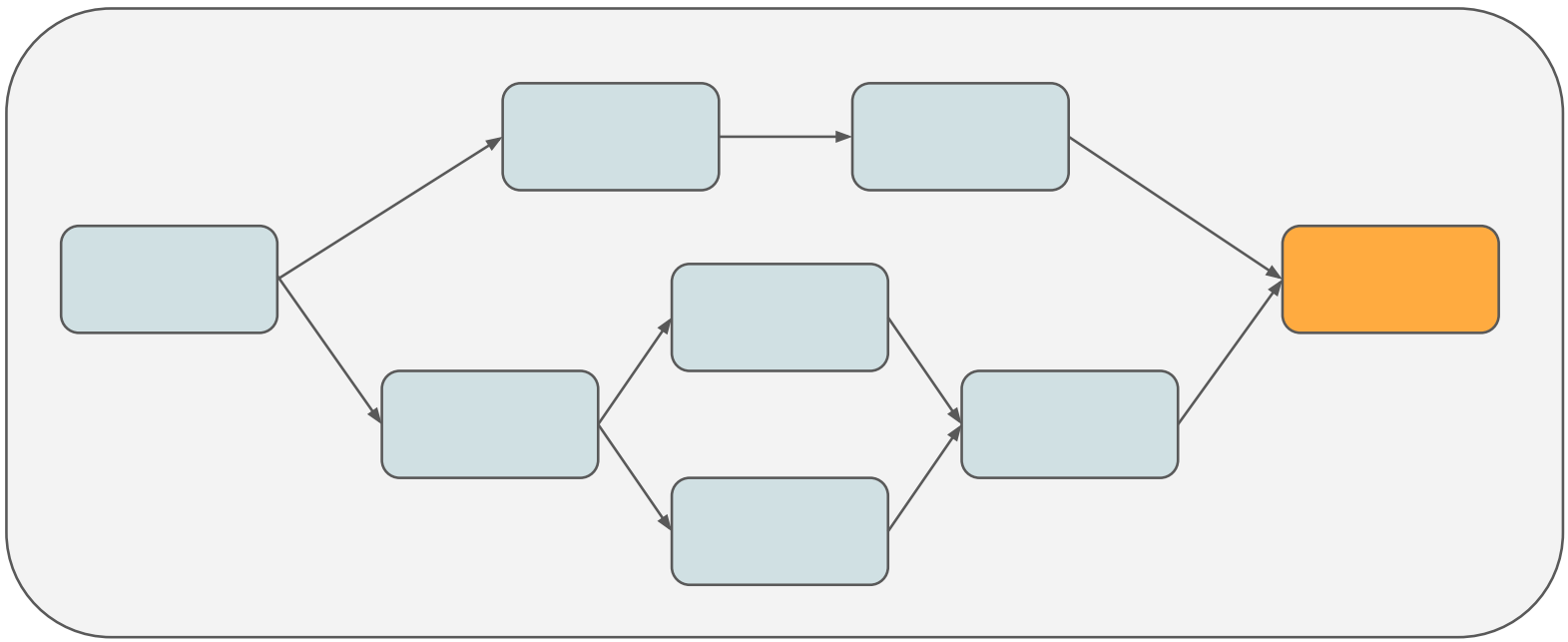
- **Preprocessing**: Feature extraction, feature selection, missing data imputation,...
- **Ensemble methods**: Model averaging, model stacking
- `mlr3`: modular model fitting

$\Rightarrow$ `mlr3pipelines`: modular ML workflows



MACHINE LEARNING WORKFLOWS

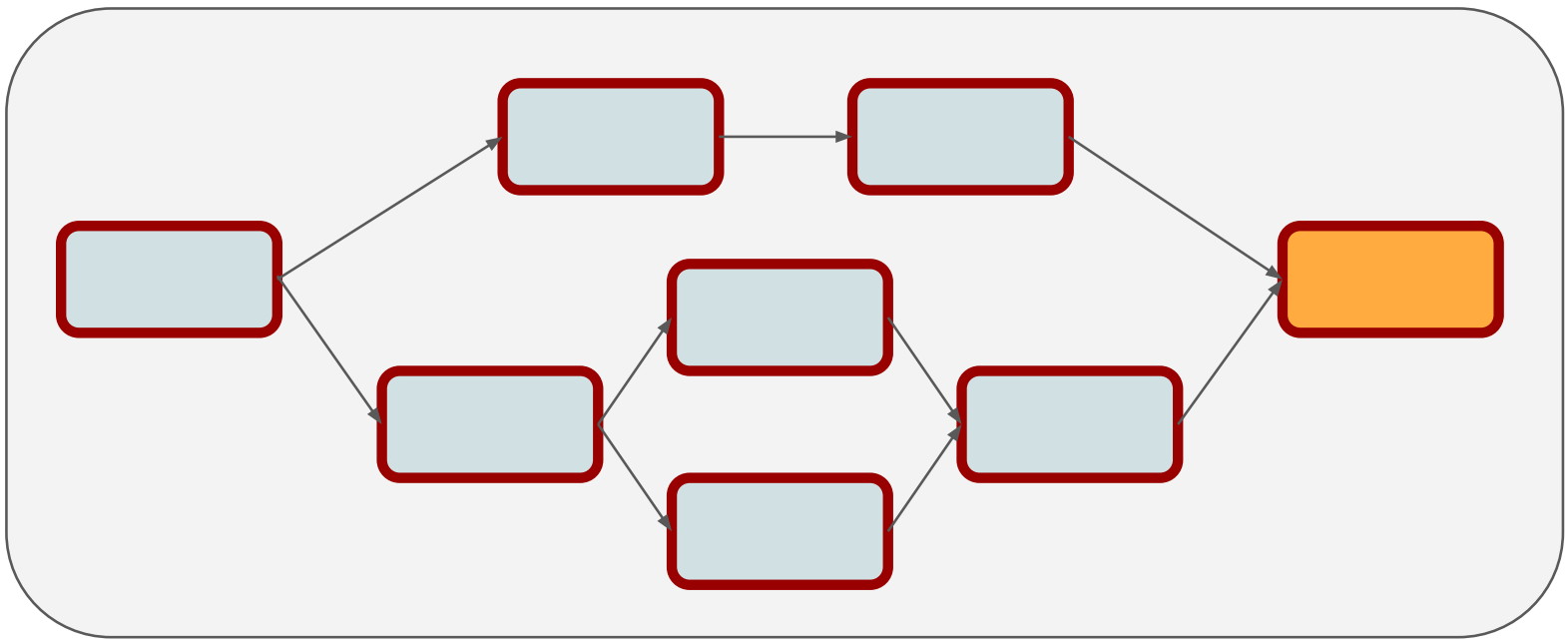
– what do they look like?



MACHINE LEARNING WORKFLOWS

– what do they look like?

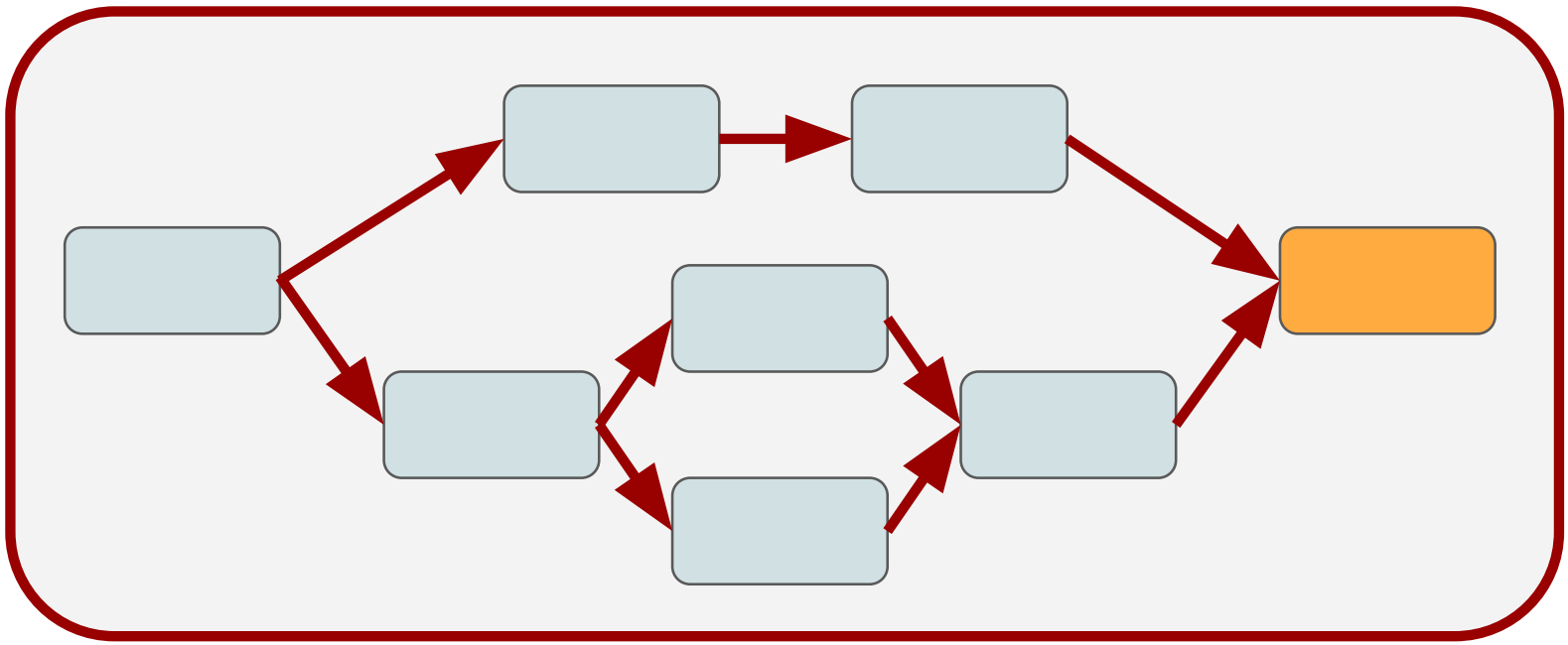
- **Building blocks**: *what* is happening? $\rightarrow$ `PipeOp`



MACHINE LEARNING WORKFLOWS

– what do they look like?

- **Building blocks**: *what* is happening? $\rightarrow$ `PipeOp`
- **Structure**: in what *sequence* is it happening? $\rightarrow$ `Graph`

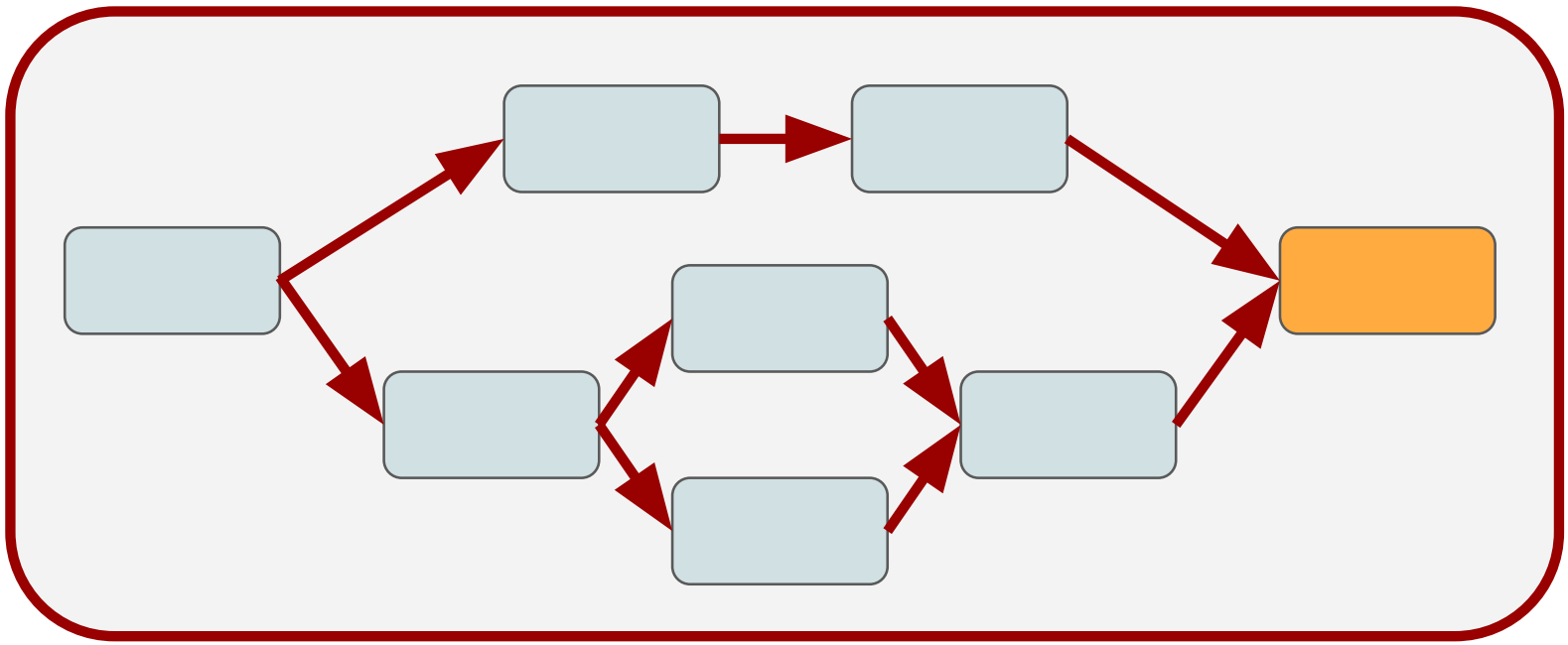


MACHINE LEARNING WORKFLOWS

– what do they look like?

- **Building blocks**: *what* is happening? $\rightarrow$ `PipeOp`
- **Structure**: in what *sequence* is it happening? $\rightarrow$ `Graph`

$\Rightarrow$ `Graph`: `PipeOps` as **nodes** with **edges** (data flow) between them

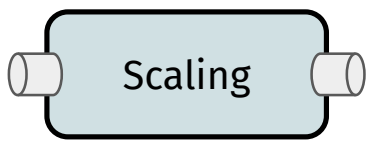


mlr3pipelines: PipeOps

THE BUILDING BLOCKS

PipeOp: Single Unit of Data Operation

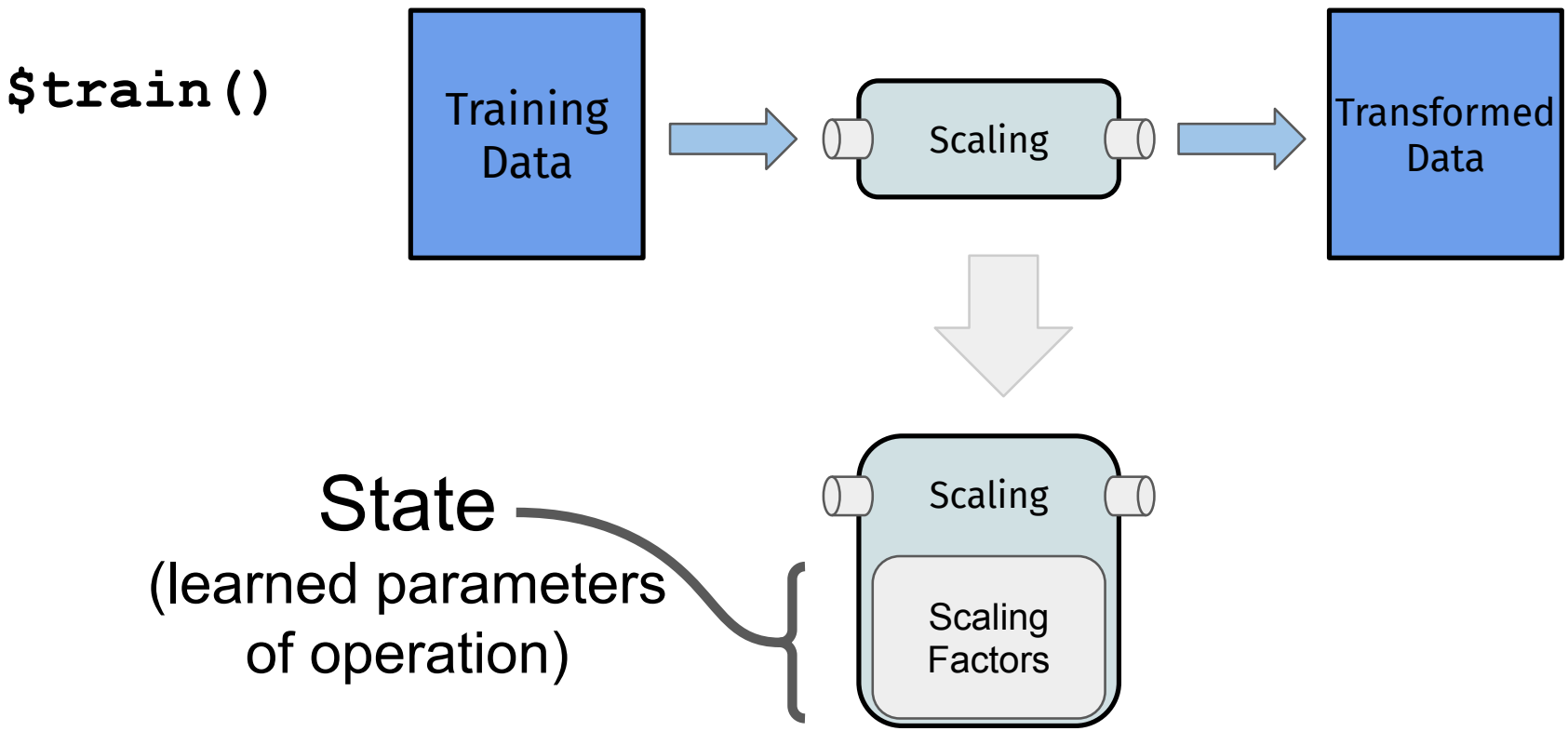
- `pip = po("scale")` to construct



THE BUILDING BLOCKS

PipeOp: Single Unit of Data Operation

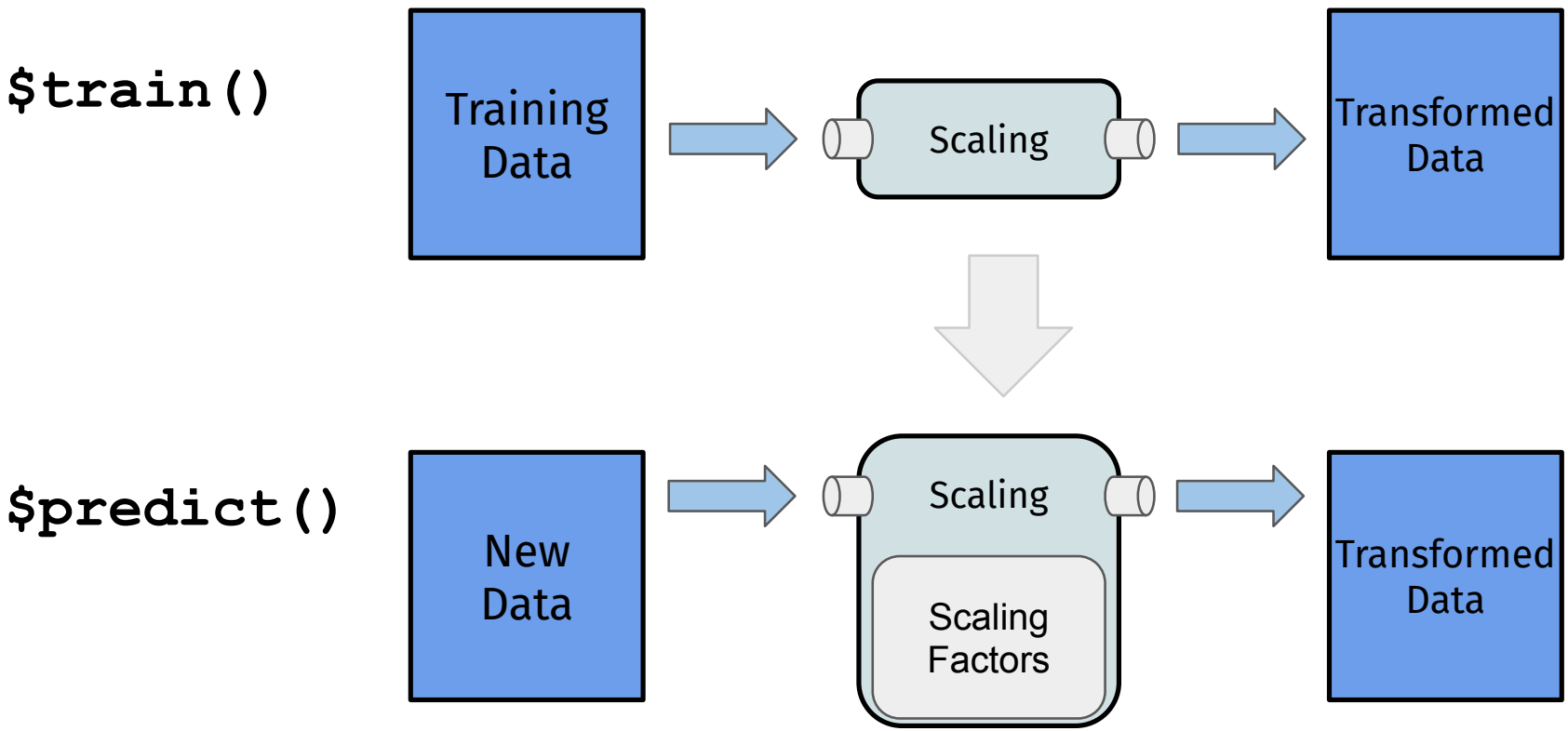
- `pip = po("scale")` to construct
- `pip$train()`: process data and create `pip$state`



THE BUILDING BLOCKS

PipeOp: Single Unit of Data Operation

- `pip = po("scale")` to construct
- `pip$train()`: process data and create `pip$state`
- `pip$predict()`: process data depending on the `pip$state`



THE BUILDING BLOCKS

```
po = po("scale")
trained = po$train(list(task))
trained$output$head(3)

#>    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#>     <fctr>        <num>       <num>        <num>       <num>
#> 1:  setosa         -1.3        -1.3         -0.9        1.02
#> 2:  setosa         -1.3        -1.3         -1.1       -0.13
#> 3:  setosa         -1.4        -1.3         -1.4        0.33
```

THE BUILDING BLOCKS

```
po = po("scale")
trained = po$train(list(task))
trained$output$head(3)

#>    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#>     <fctr>        <num>       <num>        <num>       <num>
#> 1:  setosa         -1.3        -1.3         -0.9        1.02
#> 2:  setosa         -1.3        -1.3         -1.1       -0.13
#> 3:  setosa         -1.4        -1.3         -1.4        0.33
```

```
head(po$state, 2)

#> $center
#> Petal.Length  Petal.Width Sepal.Length  Sepal.Width
#>          3.8          1.2          5.8          3.1
#>
#> $scale
#> Petal.Length  Petal.Width Sepal.Length  Sepal.Width
#>         1.77         0.76         0.83         0.44
```

THE BUILDING BLOCKS

```
po = po("scale")
trained = po$train(list(task))
trained$output$head(3)

#>    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#>     <fctr>        <num>       <num>        <num>       <num>
#> 1:  setosa         -1.3        -1.3         -0.9        1.02
#> 2:  setosa         -1.3        -1.3         -1.1       -0.13
#> 3:  setosa         -1.4        -1.3         -1.4        0.33
```

```
smalltask = task$clone()
smalltask = smalltask$filter(1:3)
pred = po$predict(list(smalltask))
pred$output$data()

#>    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#>     <fctr>        <num>       <num>        <num>       <num>
#> 1:  setosa         -1.3        -1.3         -0.9        1.02
#> 2:  setosa         -1.3        -1.3         -1.1       -0.13
#> 3:  setosa         -1.4        -1.3         -1.4        0.33
```

PIPEOPS SO FAR

```
mlr_pipeops$keys()
```

```
#>  [1] "boxcox"            "branch"            "chunk"
#>  [4] "classbalancing"    "classifavg"        "classweights"
#>  [7] "colapply"          "collapsefactors"   "colroles"
#> [10] "compose_crank"     "compose_distr"     "compose_probregr"
#> [13] "copy"              "crankcompose"      "datefeatures"
#> [16] "distrcompose"      "encode"            "encodeimpact"
#> [19] "encodelmer"        "featureunion"      "filter"
#> [22] "fixfactors"        "histbin"           "ica"
#> [25] "imputeconstant"    "imputehist"        "imputelearner"
#> [28] "imputemean"        "imputemedian"      "imputemode"
#> [31] "imputeoor"         "imputesample"      "kernelpca"
#> [34] "learner"           "learner_cv"        "missind"
#> [37] "modelmatrix"       "multiplicityexply" "multiplicityimply"
#> [40] "mutate"            "nmf"               "nop"
#> [43] "ovrsplit"          "ovrunite"          "pca"
#> [46] "proxy"             "quantilebin"       "randomprojection"
#> [49] "randomresponse"    "regravg"           "removeconstants"
#> [52] "renamecolumns"     "replicate"         "scale"
#> [55] "scalemaxabs"       "scalerange"        "select"
#> [58] "smote"             "spatialsign"       "subsample"
#> [...]
```

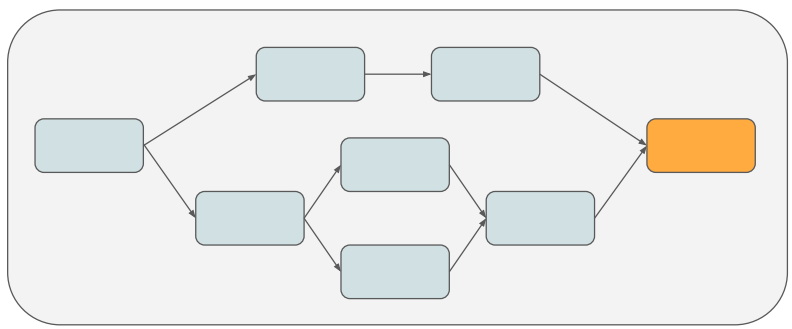
PIPEOPS

Linear PipeOps

- Simple data preprocessing operations (scaling, Box Cox, Yeo Johnson, PCA, ICA)
- Missing value imputation (sampling, mean, median, mode, new level, ...)
- Feature selection (by name, by type, using filter methods)
- Categorical data encoding (one-hot, treatment, impact)
- Sampling (subsampling for speed, sampling for class balance)
- Text processing
- Date processing

Complex PipeOps

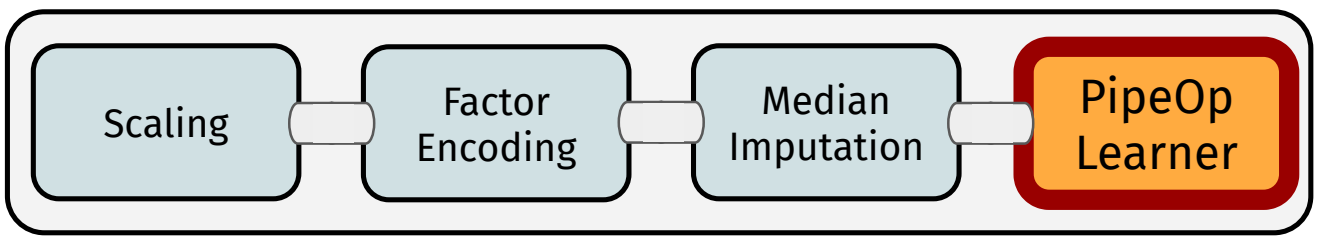
- Ensemble methods on Predictions (weighted average, possibly learned weights)
- Branching (simultaneous branching, alternative branching)
- Combination of data: `featureunion`



LEARNERS AND GRAPHS

PipeOpLearner

- `Learner` as a `PipeOp`
- Fits a model, output is `Prediction`



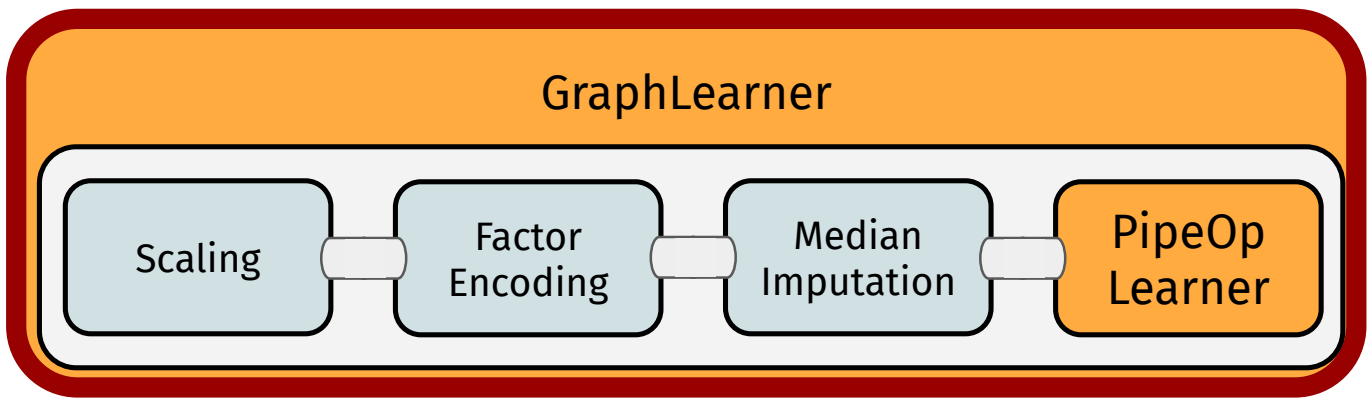
LEARNERS AND GRAPHS

PipeOpLearner

- `Learner` as a `PipeOp`
- Fits a model, output is `Prediction`

GraphLearner

- `Graph` as a `Learner`
- All benefits of `mlr3`: **resampling**, **tuning**, **nested resampling**, . . .

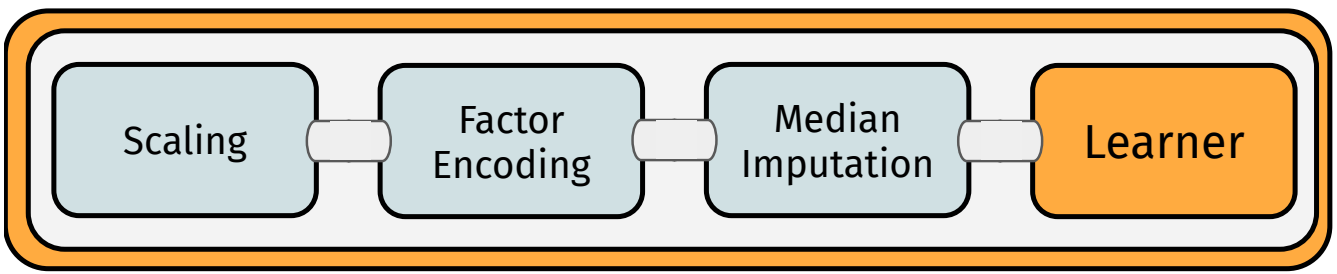


mlr3pipelines: Linear Pipelines

MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

```
graph_pp = po("scale") %>>%
  po("encode") %>>%
  po("imputemedian") %>>%
  lrn("classif.rpart")
```

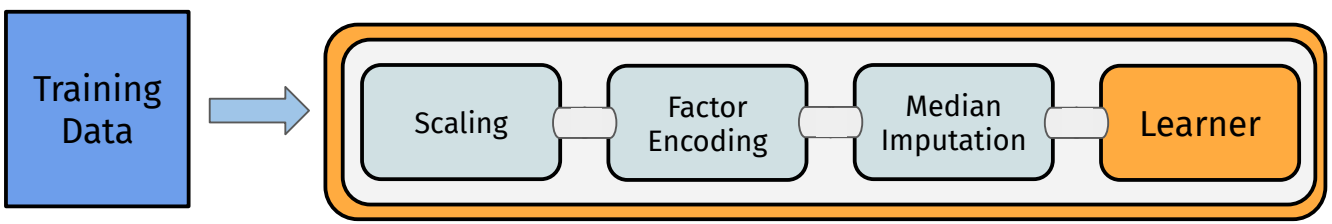


MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

- `train()`ing: Data propagates and creates `$states`

```
glrn = GraphLearner$new(graph_pp)
glrn$train(task)
```

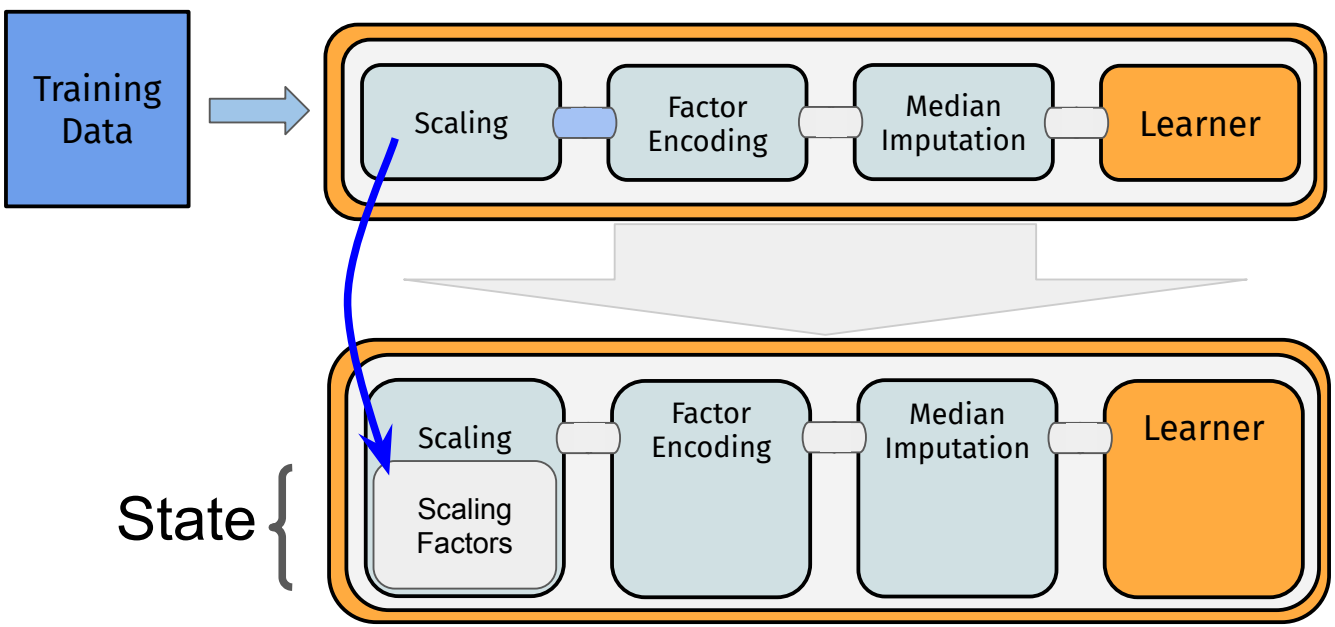


MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

- `train()`ing: Data propagates and creates `$states`

```
glrn = GraphLearner$new(graph_pp)
glrn$train(task)
```

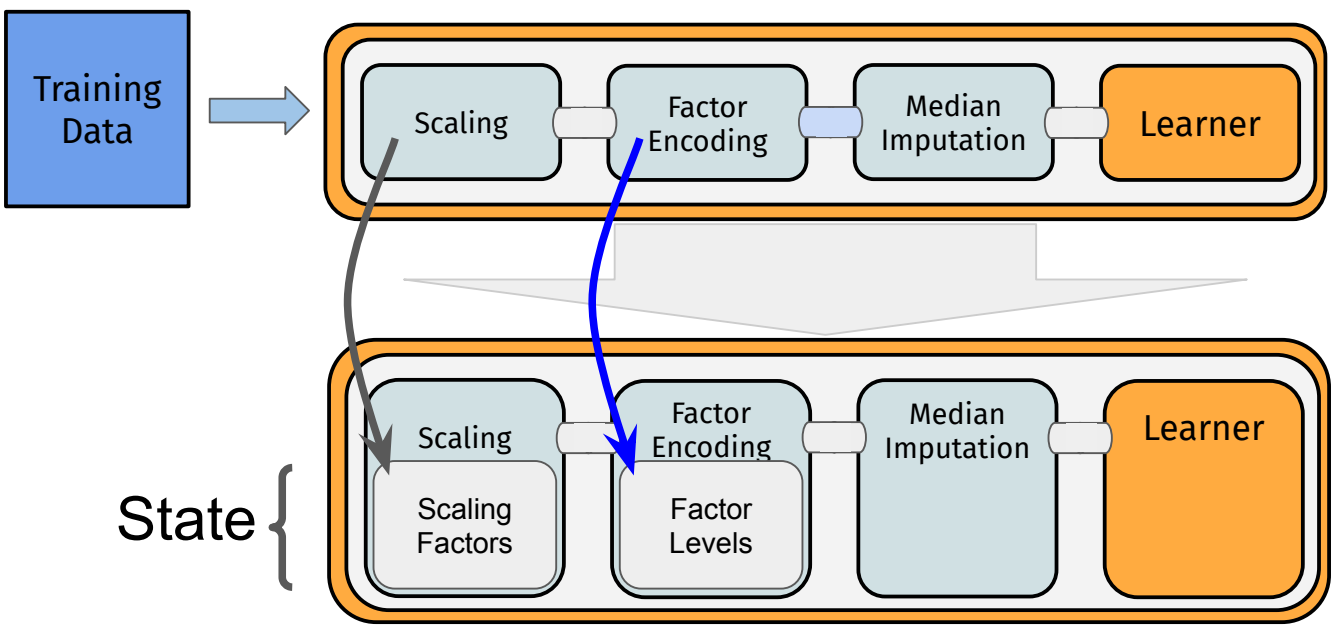


MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

- `train()`ing: Data propagates and creates `$states`

```
glrn = GraphLearner$new(graph_pp)
glrn$train(task)
```

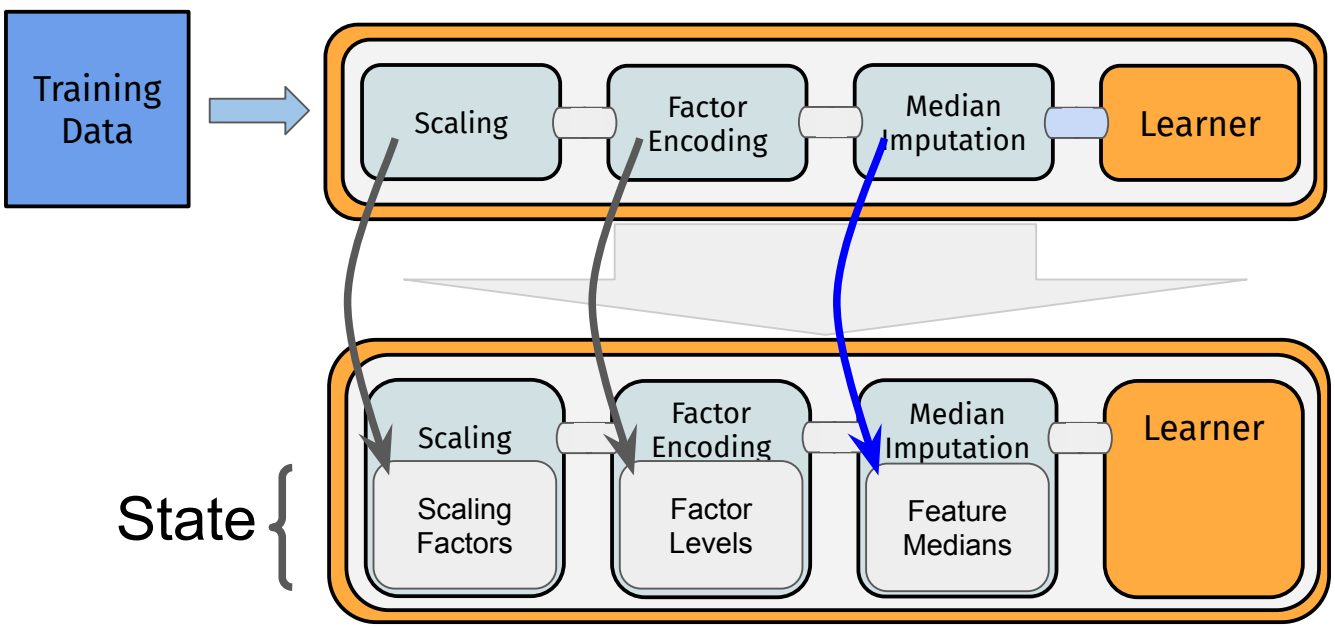


MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

- `train()`ing: Data propagates and creates `$states`

```
glrn = GraphLearner$new(graph_pp)
glrn$train(task)
```

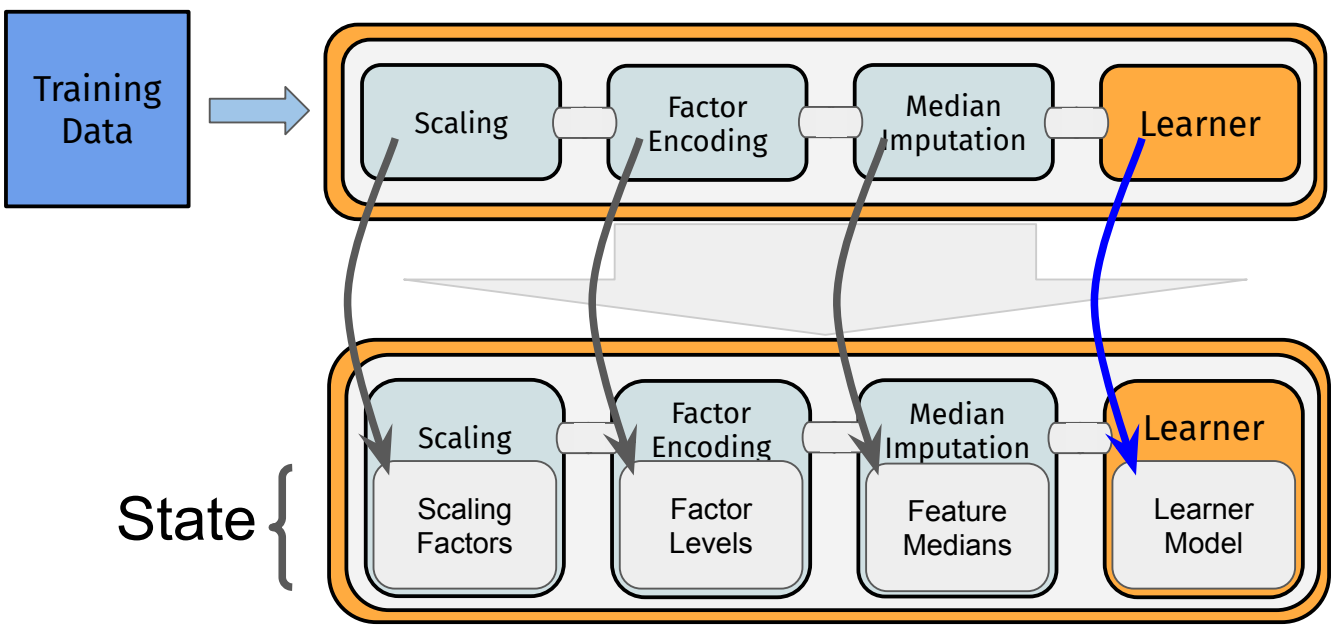


MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

- `train()`ing: Data propagates and creates `$states`

```
glrn = GraphLearner$new(graph_pp)
glrn$train(task)
```

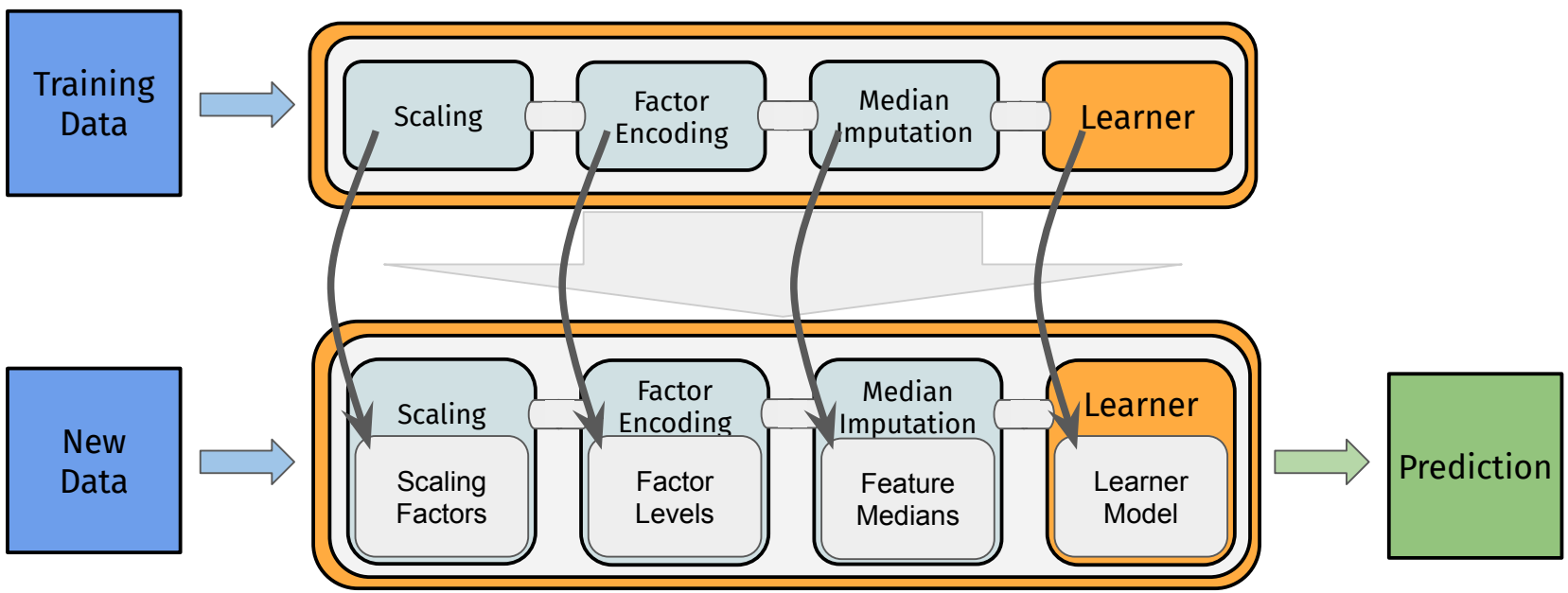


MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

- `train()`ing: Data propagates and creates `$states`
- `predict()`tion: Data propagates, uses `$states`

```
glrn$predict(task)
```



MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline `scale %>>% encode %>>% impute %>>% rpart`

- Setting / retrieving parameters: `$param_set`

```
graph_pp$pipeops$scale$param_set$values$center = FALSE
```

MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline `scale %>>% encode %>>% impute %>>% rpart`

- Setting / retrieving parameters: `$param_set`

```
graph_pp$pipeops$scale$param_set$values$center = FALSE
```

- Retrieving state: `$state` of individual `PipeOps` (*after* `$train()`)

```
graph_pp$pipeops$scale$state$scale
#> Petal.Length  Petal.Width Sepal.Length  Sepal.Width
#>          4.2          1.4          5.9          3.1
```

MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline `scale %>>% encode %>>% impute %>>% rpart`

- Setting / retrieving parameters: `$param_set`

```
graph_pp$pipeops$scale$param_set$values$center = FALSE
```

- Retrieving state: `$state` of individual `PipeOps` (*after* `$train()`)

```
graph_pp$pipeops$scale$state$scale
#> Petal.Length  Petal.Width Sepal.Length  Sepal.Width
#>          4.2          1.4          5.9          3.1
```

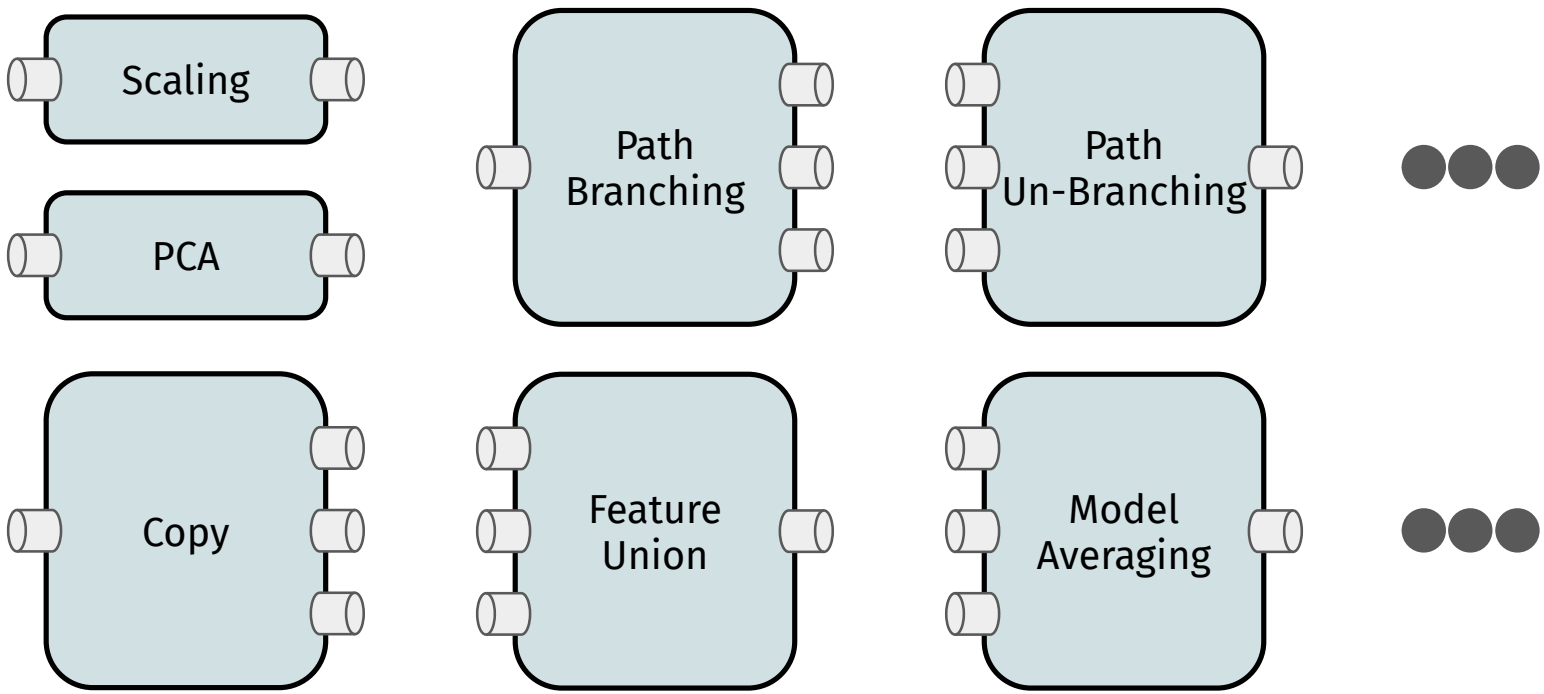
- Retrieving intermediate results: `$.result` (set debug option before)

```
graph_pp$pipeops$scale$.result[[1]]$head(3)
#>    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#>     <fctr>        <num>       <num>        <num>       <num>
#> 1:  setosa         0.34        0.14         0.86        1.13
#> 2:  setosa         0.34        0.14         0.83        0.97
#> 3:  setosa         0.31        0.14         0.79        1.03
```

mlr3pipelines: Nonlinear Pipelines

PIPEOPS WITH MULTIPLE INPUTS / OUTPUTS

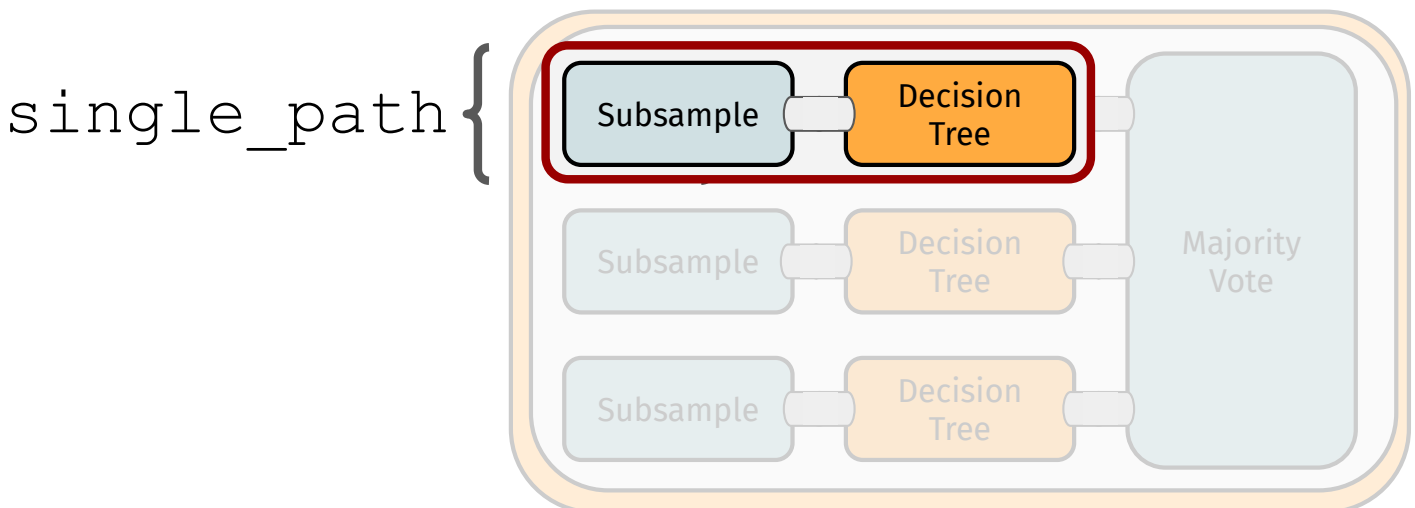
PipeOps with multiple inputs or multiple outputs:



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

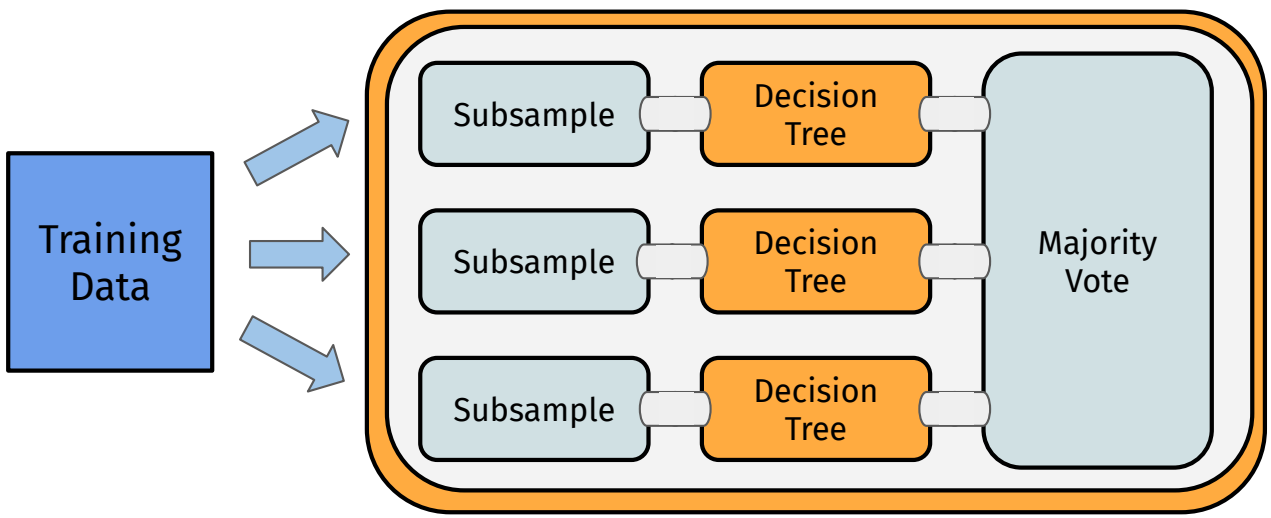
```
single_path = po("subsample") %>>% lrn("classif.rpart")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

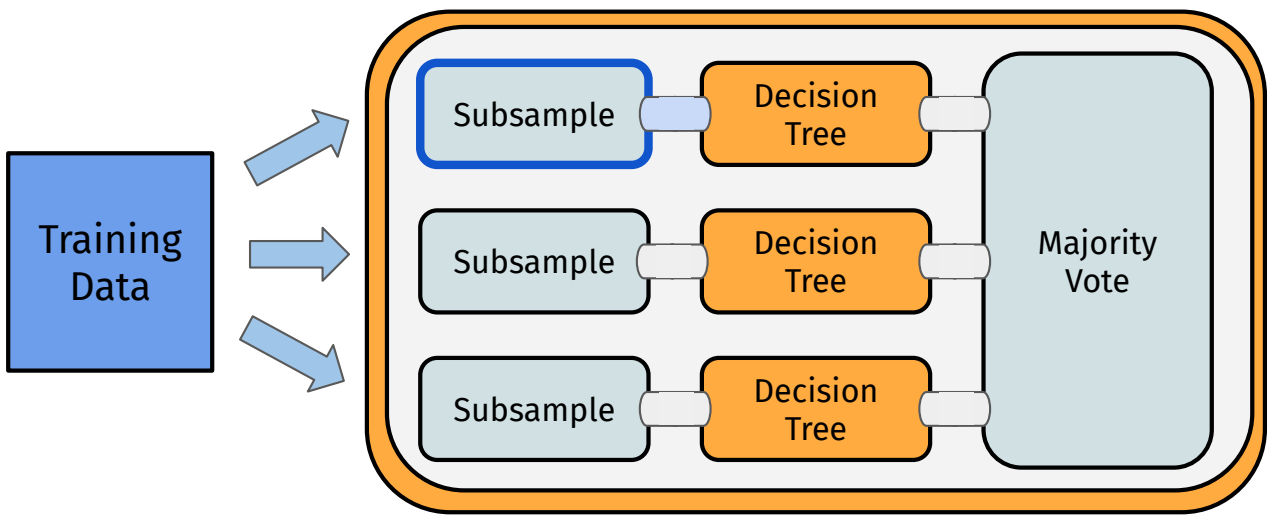
```
single_path = po("subsample") %>>% lrn("classif.rpart")

graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

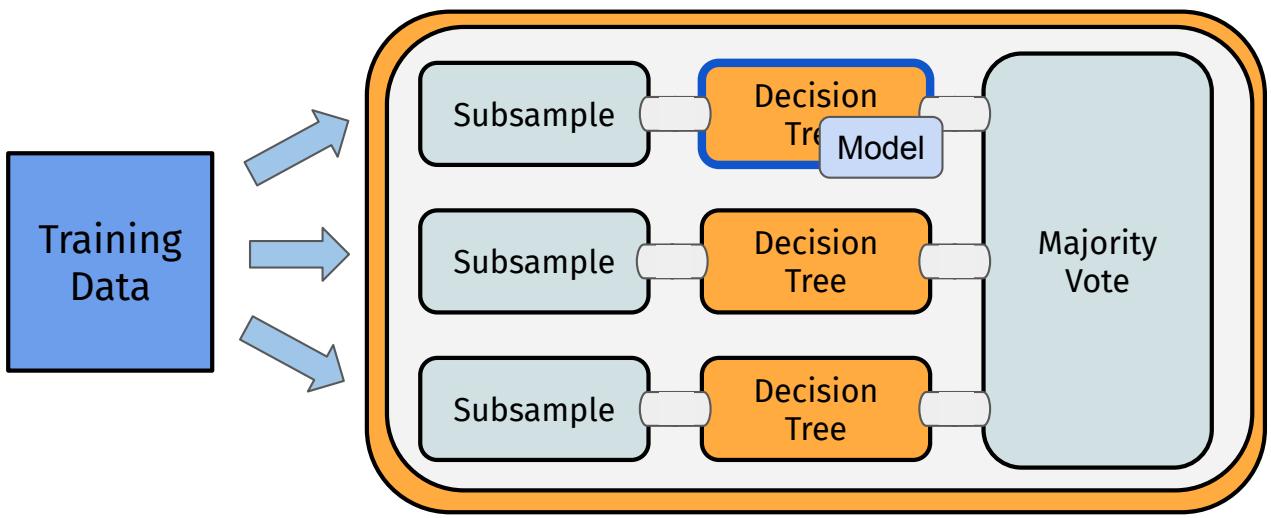
```
single_path = po("subsample") %>>% lrn("classif.rpart")

graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

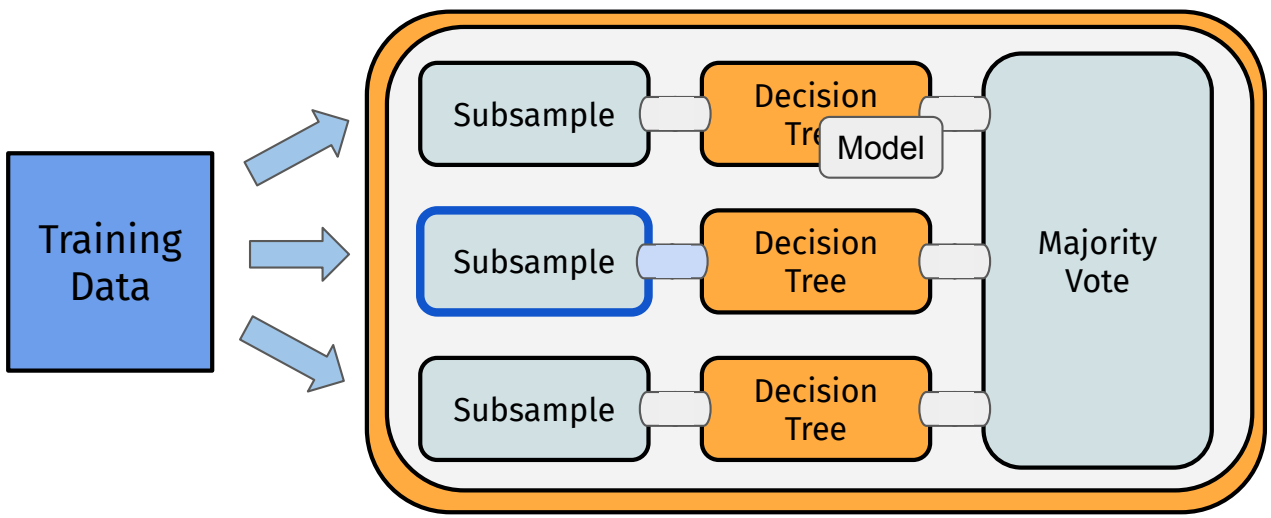
```
single_path = po("subsample") %>>% lrn("classif.rpart")

graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

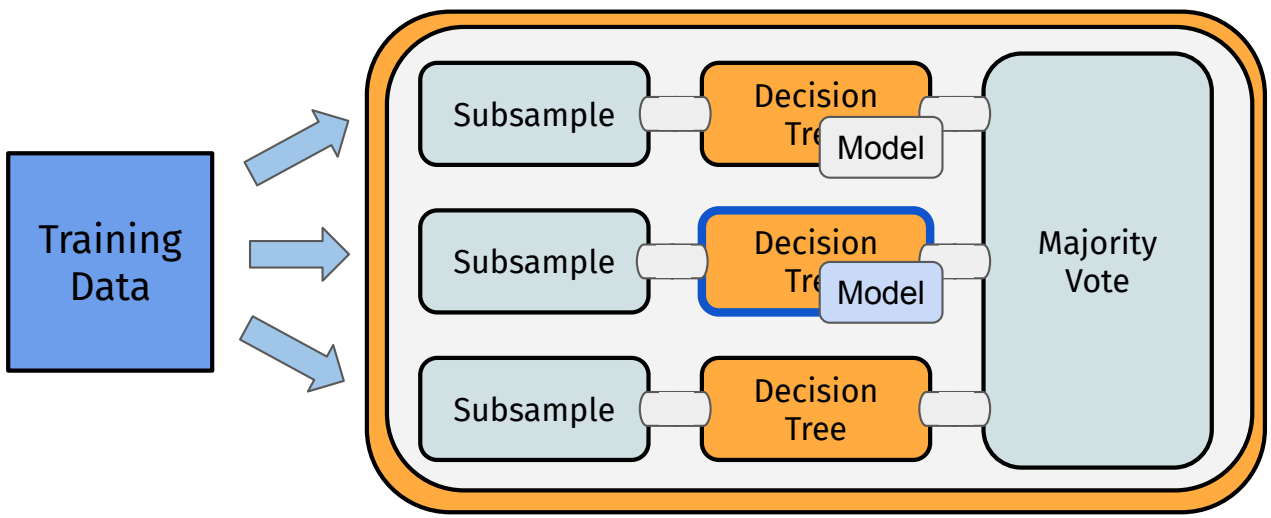
```
single_path = po("subsample") %>>% lrn("classif.rpart")

graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

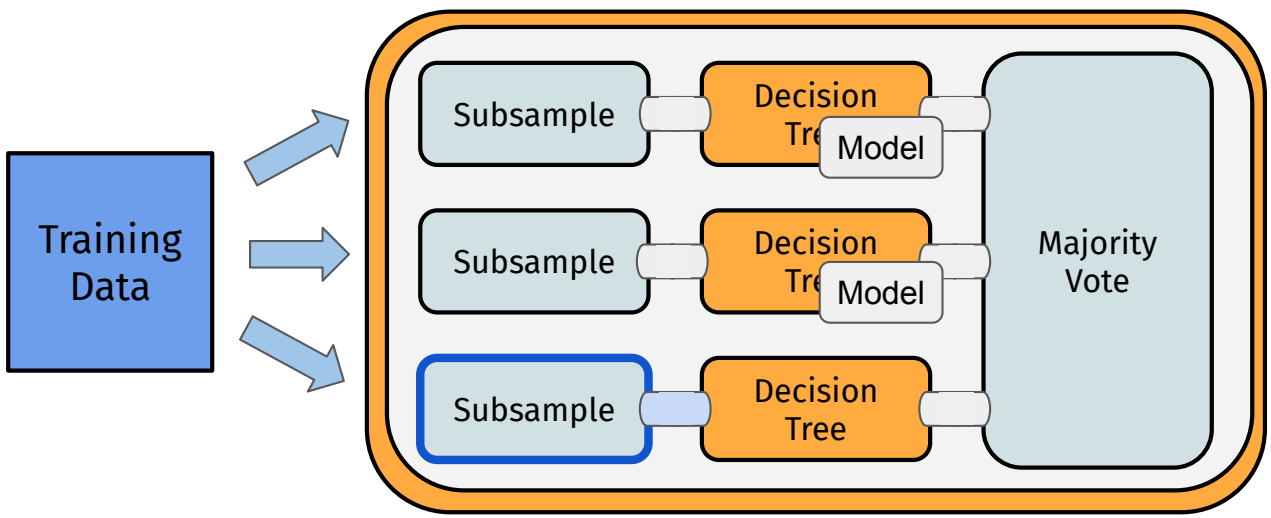
```
single_path = po("subsample") %>>% lrn("classif.rpart")

graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

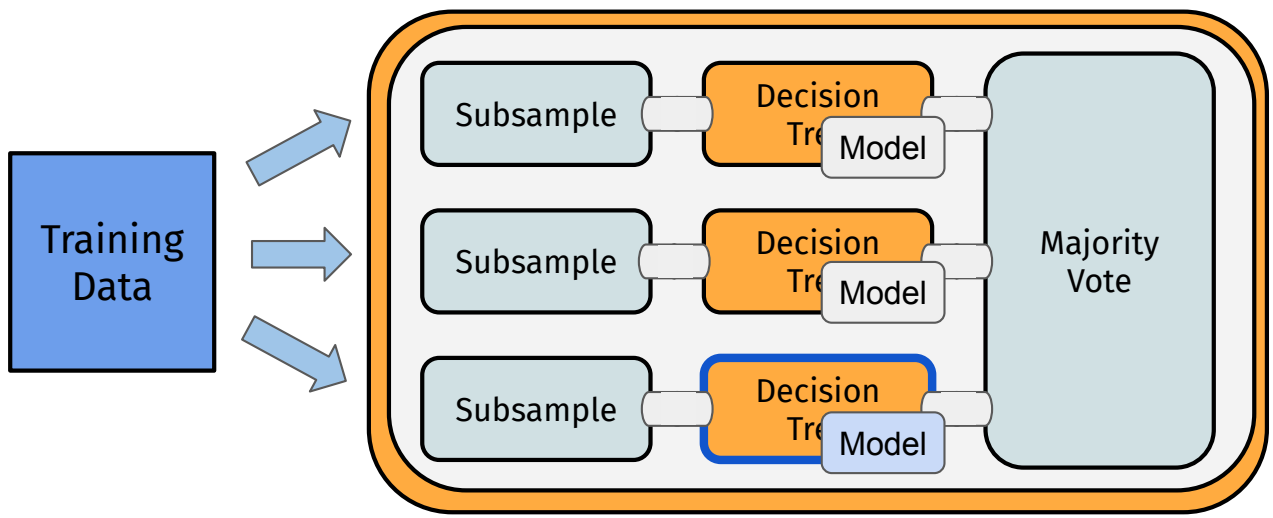
```
single_path = po("subsample") %>>% lrn("classif.rpart")

graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

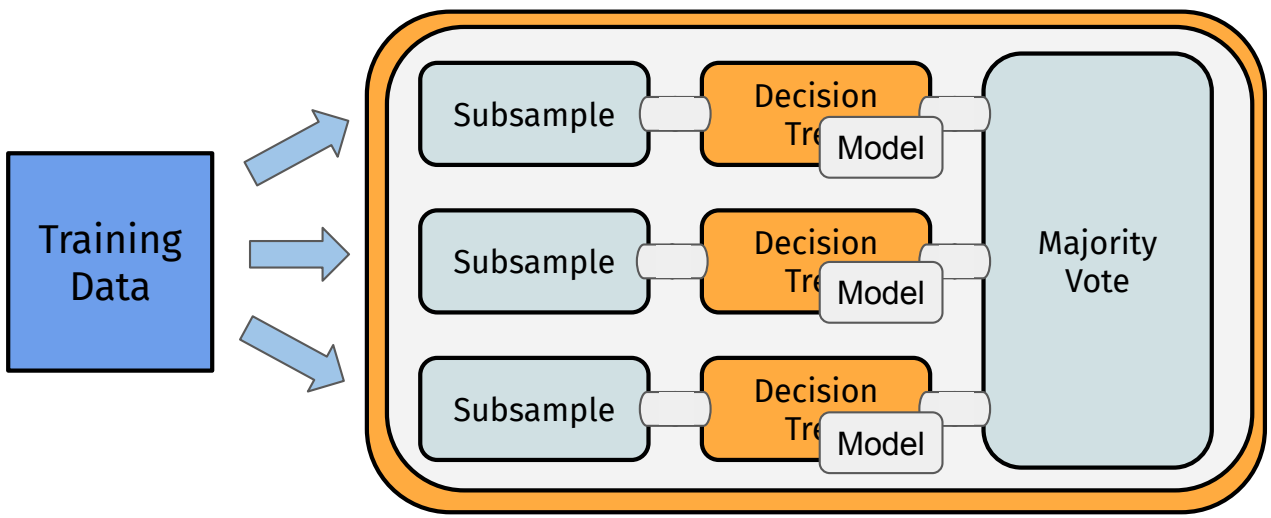
```
single_path = po("subsample") %>>% lrn("classif.rpart")

graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

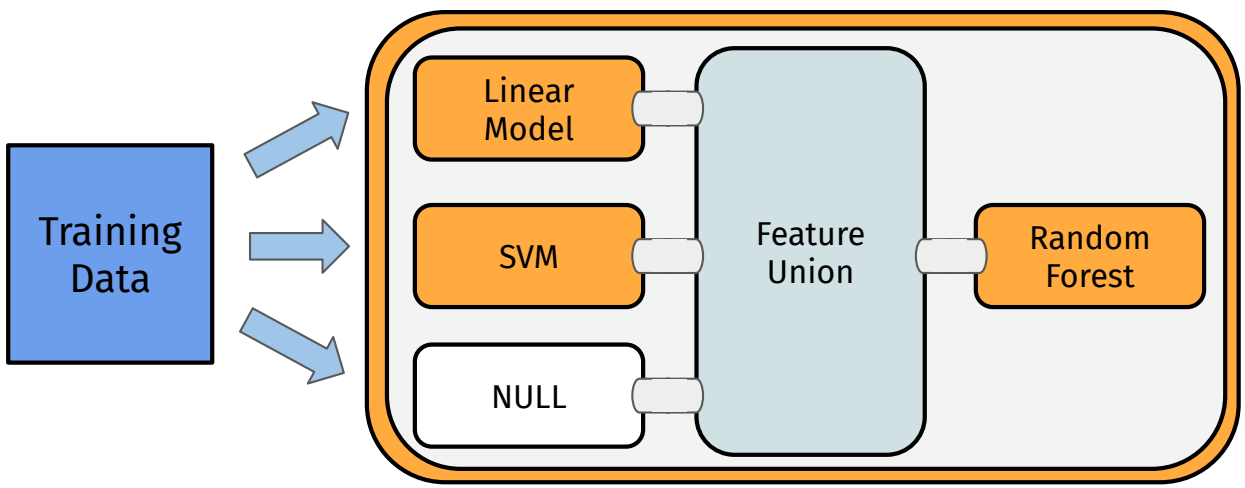
```
single_path = po("subsample") %>>% lrn("classif.rpart")

graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Stacking

```
graph_stack = gunion(list(
      po("learner_cv", learner = lrn("regr.lm")),
      po("learner_cv", learner = lrn("regr.svm")),
      po("nop"))) %>>%
  po("featureunion") %>>%
  lrn("regr.ranger")
```

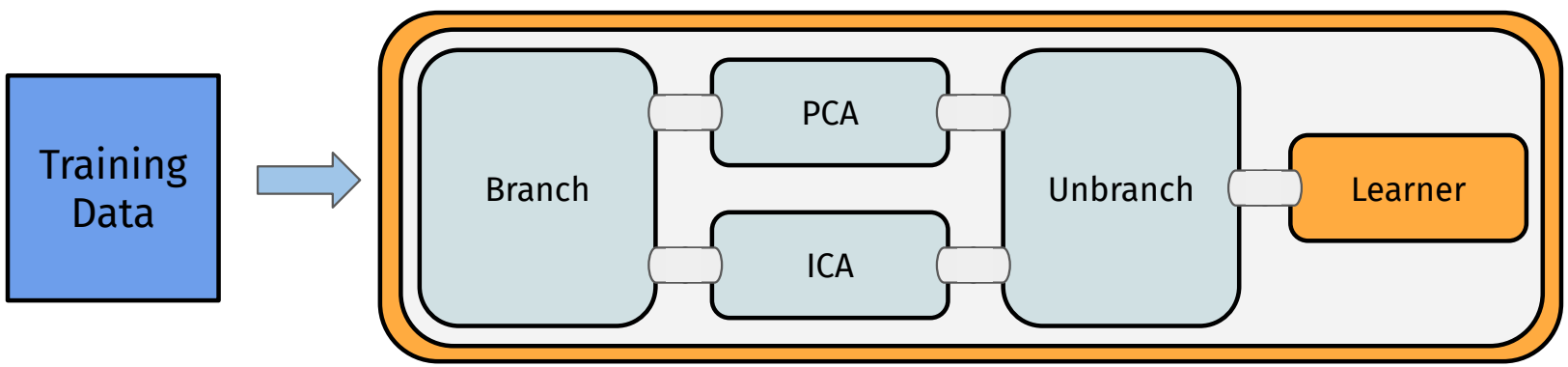


MLR3PIPELINES IN ACTION

Branching

```
graph_branch = ppl("branch", list(
  pca = po("pca"),
  ica = po("ica"))) %>>%
  lrn("classif.kknn")
```

Execute only one of several alternative paths

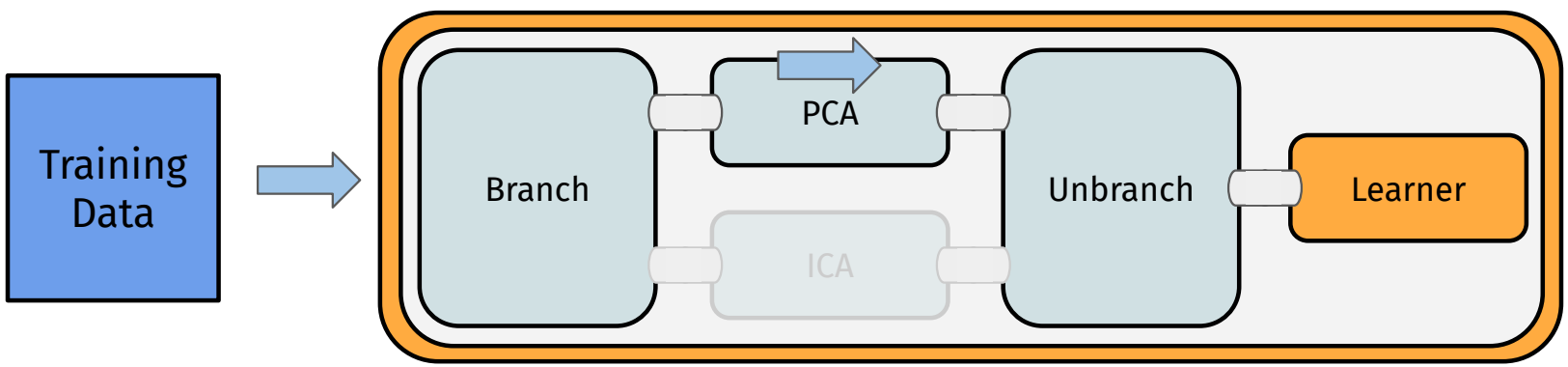


MLR3PIPELINES IN ACTION

Branching

```
graph_branch = ppl("branch", list(
  pca = po("pca"),
  ica = po("ica"))) %>>%
  lrn("classif.kknn")
```

```
> graph_branch$pipeops$branch$
    param_set$values$selection = "pca"
```

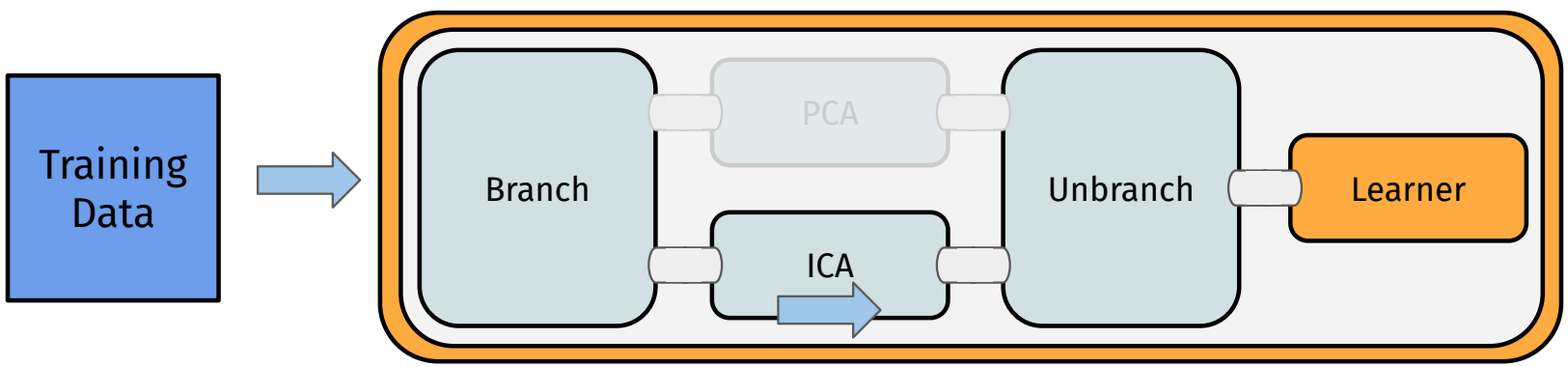


MLR3PIPELINES IN ACTION

Branching

```
graph_branch = ppl("branch", list(
  pca = po("pca"),
  ica = po("ica"))) %>>%
  lrn("classif.kknn")
```

```
> graph_branch$pipeops$branch$
    param_set$values$selection = "ica"
```



mlr3pipelines: Targeting Columns

TARGETING COLUMNS

Two ways of restricting actions to individual columns:

- Individual `PipeOps`: `affect_columns` parameter
- Subgraphs, `po("select")`, and `po("featureunion")`

$\Rightarrow$ Both make use of column `Selectors`

Suppose we only want PCA on some columns of our data:

```
task$data(1:9)
```

```
#>    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#>     <fctr>        <num>       <num>        <num>       <num>
#> 1:  setosa          1.4         0.2          5.1         3.5
#> 2:  setosa          1.4         0.2          4.9         3.0
#> 3:  setosa          1.3         0.2          4.7         3.2
#> 4:  setosa          1.5         0.2          4.6         3.1
#> 5:  setosa          1.4         0.2          5.0         3.6
#> 6:  setosa          1.7         0.4          5.4         3.9
#> 7:  setosa          1.4         0.3          4.6         3.4
#> 8:  setosa          1.5         0.2          5.0         3.4
#> 9:  setosa          1.4         0.2          4.4         2.9
```

TARGETING COLUMNS

Two ways of restricting actions to individual columns:

- Individual `PipeOps`: `affect_columns` parameter
- Subgraphs, `po("select")`, and `po("featureunion")`

$\Rightarrow$ Both make use of column `Selectors`

Using `affect_columns`:

```
sel = selector_grep("^Sepal")

partial_pca = po("pca", affect_columns = sel)

result = partial_pca$train(list(task))

result[[1]]$data(1:3)

#>    Species   PC1    PC2 Petal.Length Petal.Width
#>     <fctr> <num>  <num>        <num>       <num>
#> 1:  setosa -0.78  0.378          1.4         0.2
#> 2:  setosa -0.94 -0.137          1.4         0.2
#> 3:  setosa -1.15  0.045          1.3         0.2
```

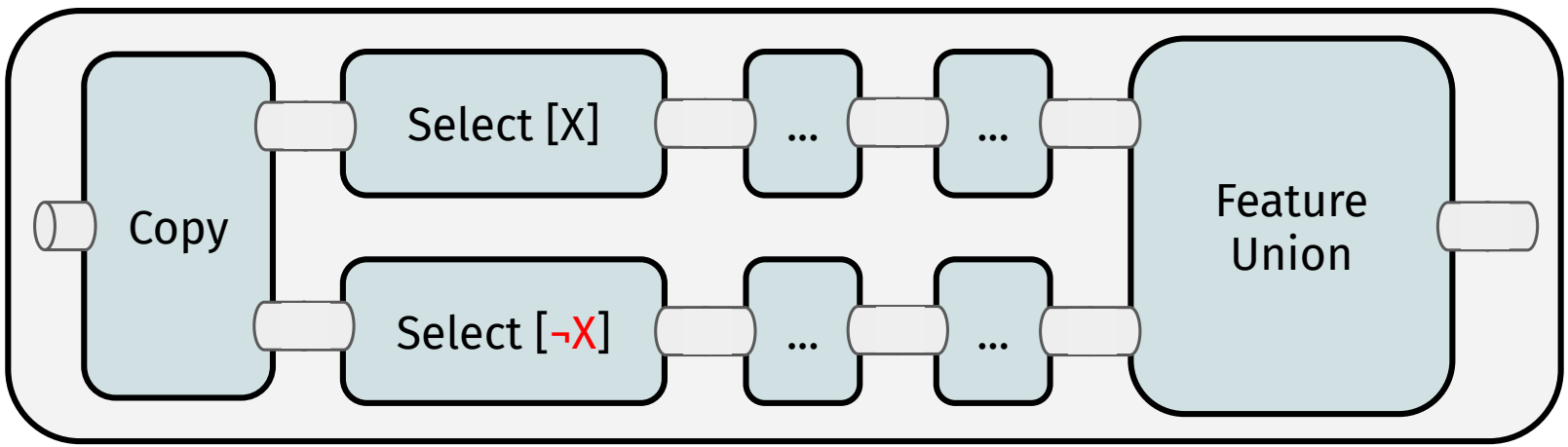
TARGETING COLUMNS

Two ways of restricting actions to individual columns:

- Individual `PipeOps`: `affect_columns` parameter
- Subgraphs, `po("select")`, and `po("featureunion")`

$\Rightarrow$ Both make use of column `Selectors`

Using `po("select")`:



TARGETING COLUMNS

Two ways of restricting actions to individual columns:

- Individual `PipeOps`: `affect_columns` parameter
- Subgraphs, `po("select")`, and `po("featureunion")`

$\Rightarrow$ Both make use of column `Selectors`

Using `po("select")`:

```
sel = selector_grep("^Sepal")
selcomp = selector_invert(sel)

partial_pca = gunion(list(
  po("select", selector = sel) %>>% po("pca"),
  po("select", selector = selcomp, id = "select2"))) %>>%
  po("featureunion")

partial_pca$train(task)[[1]]$data(1:3)

#>    Species   PC1    PC2 Petal.Length Petal.Width
#>     <fctr> <num>  <num>        <num>       <num>
#> 1:  setosa -0.78  0.378          1.4         0.2
#> 2:  setosa -0.94 -0.137          1.4         0.2
#> 3:  setosa -1.15  0.045          1.3         0.2
```

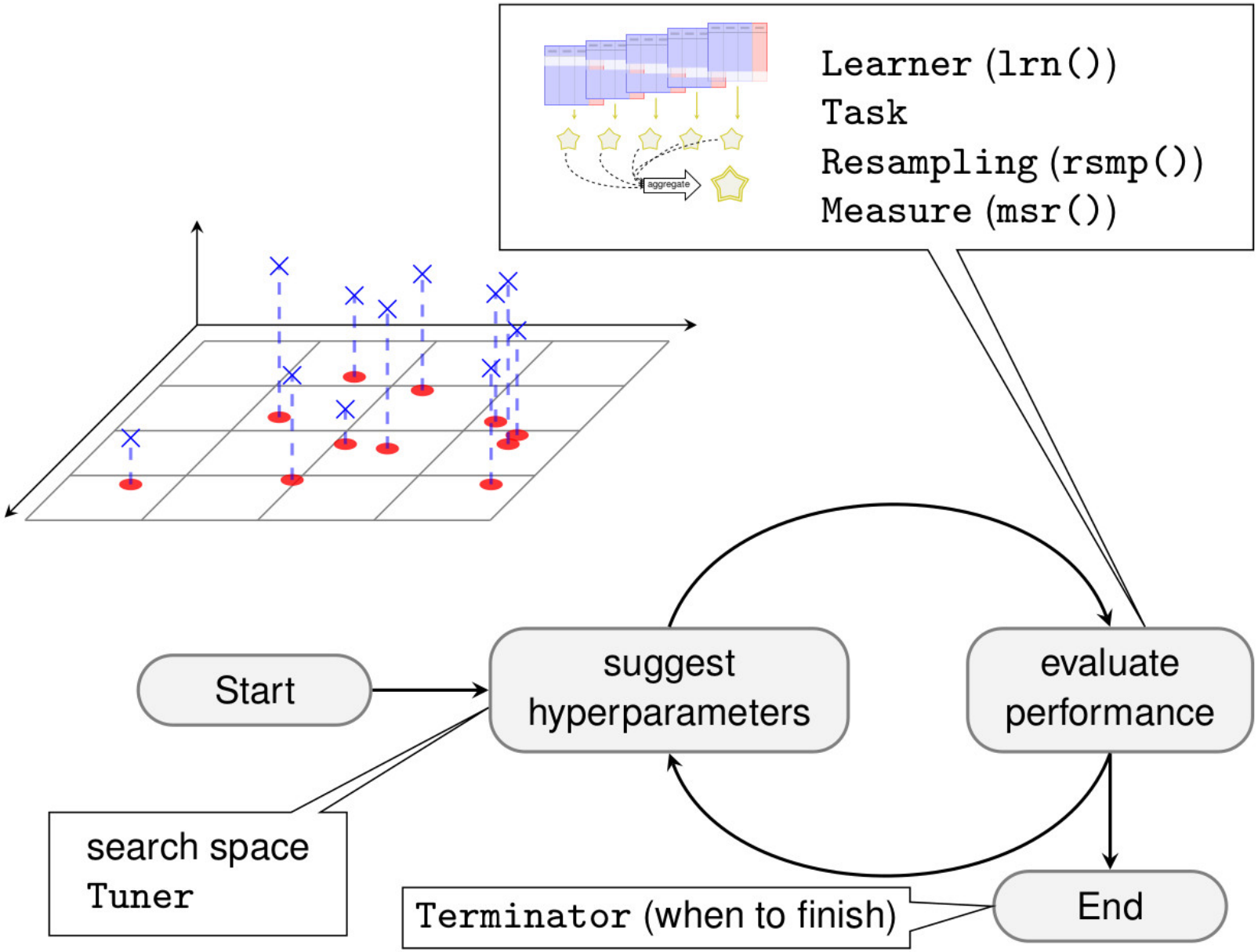
MLR3PIPELINES: SHORT RECAP

`mlr3pipelines` overview:

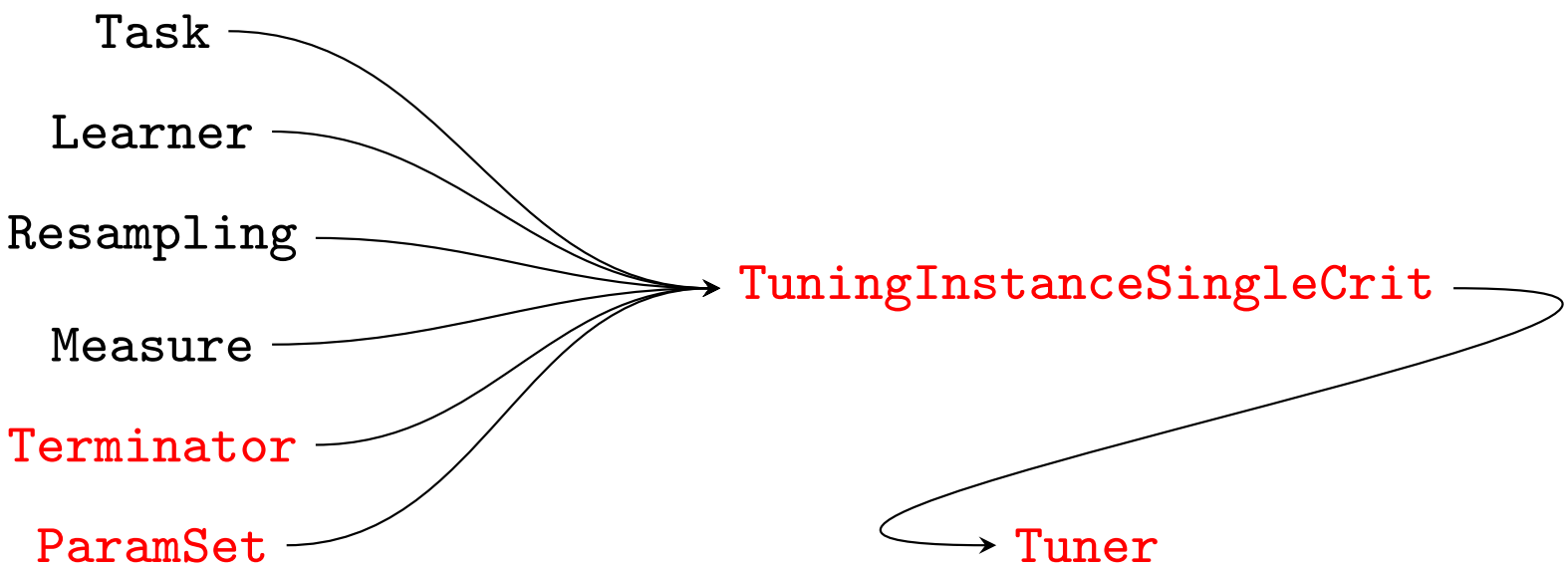
- Construct a `PipeOp` using `po()`
- Use `Graph` operators to connect them
  - %>>%—chain operations
  - `gunion()`—put operations in parallel
  - `pipeline_greplicate()`—put many copies of an operation in parallel
- Train/predict with the `PipeOp` or `Graph` using `$train()`/`$predict()`
- Inspect the trained state through `$state`
- Encapsulate the `Graph` in a `GraphLearner` for resampling, benchmarking, and tuning

mlr3tunig

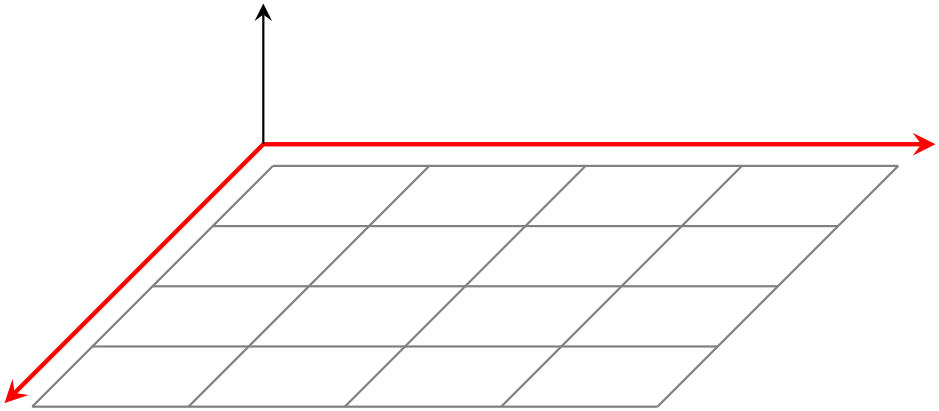
MLR3TUNIG


Learner (lrn())
Task
Resampling (rsmp())
Measure (msr())
aggregate
Start
suggest hyperparameters
evaluate performance
search space
Tuner
Terminator (when to finish)
End

OBJECTS IN TUNING



SEARCH SPACE



```
ps(id1 = domain1, id2 = domain2, ...)
```

Numerical parameter	`p_dbl(lower, upper)`
Integer parameter	`p_int(lower, upper)`
Discrete parameter	`p_fct(levels)`
Logical parameter	`p_lgl()`

```
library("paradox")
searchspace_knn = ps(
  k = p_int(lower = 1, upper = 20)
)
```

TERMINATION

- Tuning needs a *termination condition*: when to finish
- `trm()` to create `Terminator` objects

```
as.data.table(trm())
```

```
#>                 key
#>              <char>
#> 1:       clock_time
#> 2:            combo
#> 3:            evals
#> 4:             none
#> 5:     perf_reached
#> 6:         run_time
#> 7:       stagnation
#> 8: stagnation_batch
```

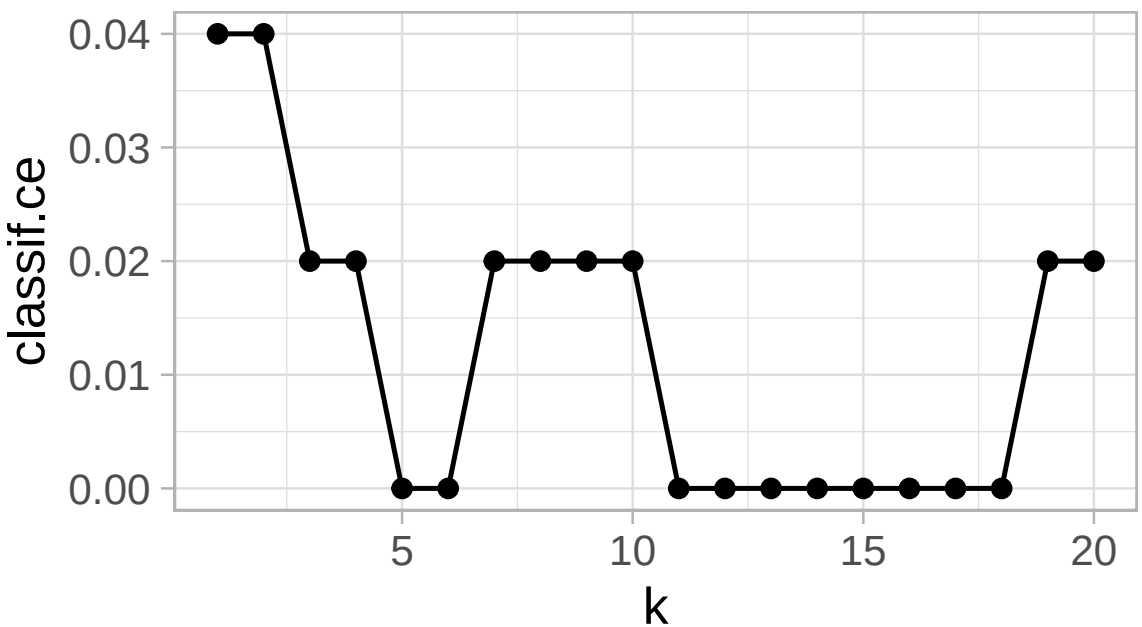
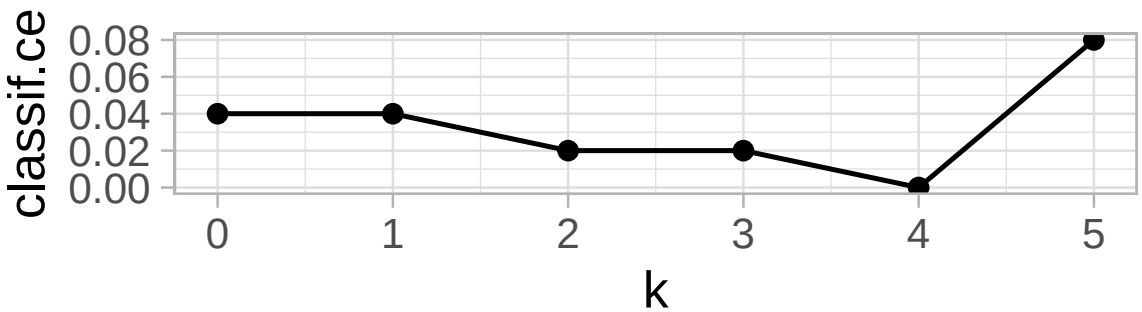
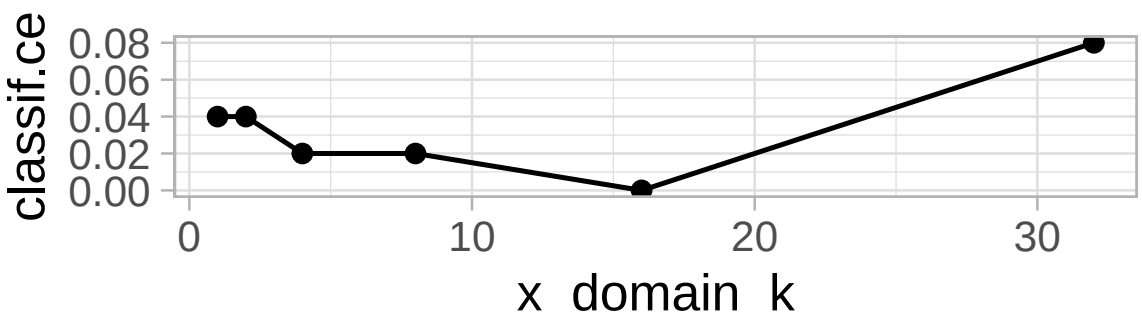
```
trm("evals", n_evals = 20)
```

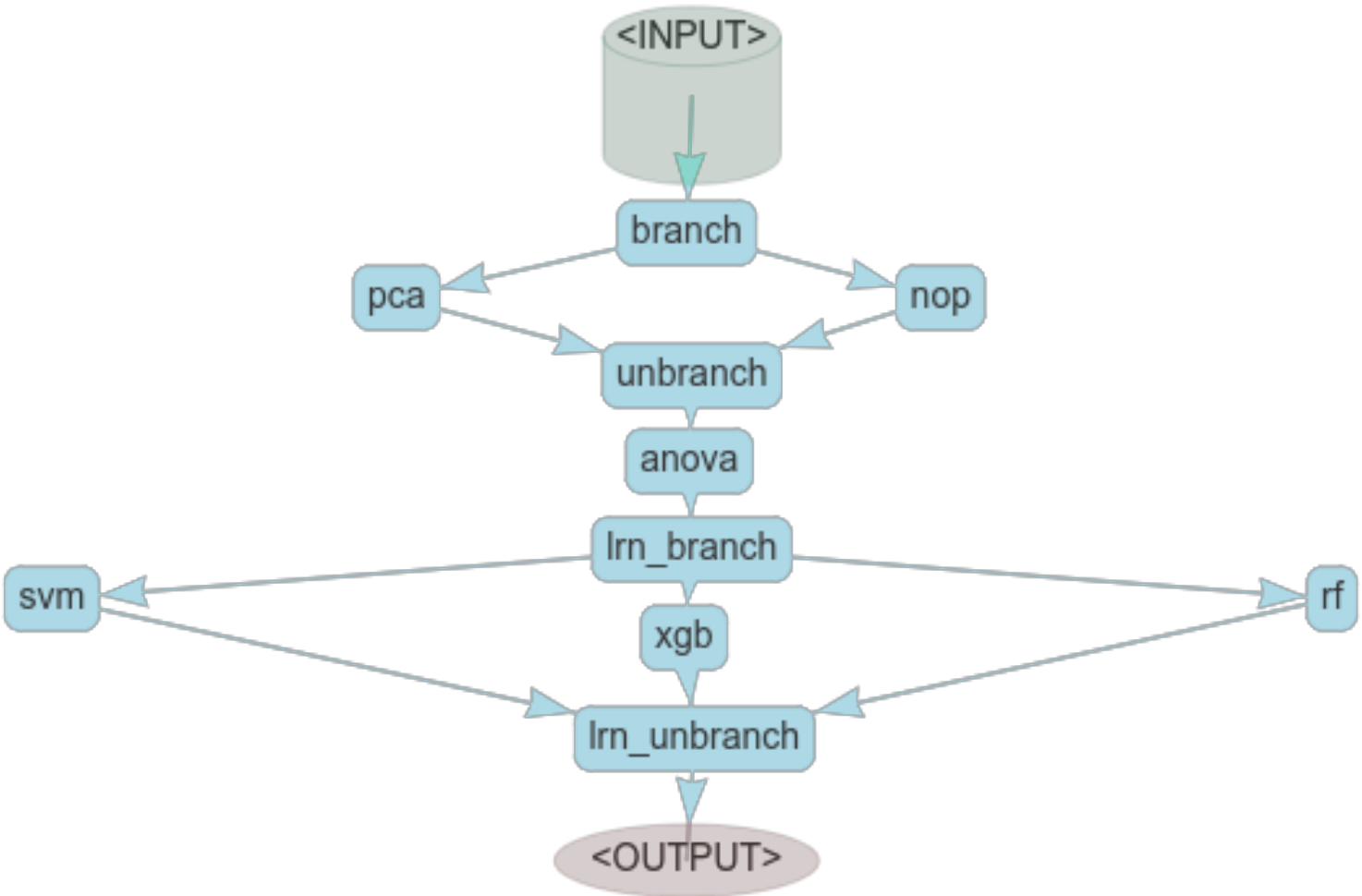
```
#> <TerminatorEvals>
#> * Parameters: n_evals=20
```

TUNING METHOD

- need to choose a *tuning method*
- `tnr()` to create `Tuner` objects
- ```
  as.data.table(tnr())
  #>              key
  #>           <char>
  #> 1:          cmaes
  #> 2: design_points
  #> 3:          gensa
  #> 4:    grid_search
  #> 5:         nloptr
  #> 6: random_search
  ```
  ```

TUNING RESULTS

```
inst = TuningInstanceSingleCrit$new(task = tsk("iris"),
  learner = lrn("classif.kknn", kernel = "rectangular"),
  resampling = rsmp("holdout"), measure = msr("classif.ce"),
  terminator = trm("none"), search_space = searchspace_knn)
gsearch = tnr("grid_search", resolution = 20)
gsearch$optimize(inst)

#>        k learner_param_vals  x_domain classif.ce
#>    <int>             <list>    <list>      <num>
#> 1:    11          <list[2]> <list[1]>          0

ggplot(as.data.table(inst$archive), aes(x = k, y = classif.ce)) +
  geom_line() + geom_point()
```



TUNING WITH TRAFO

```
searchspace_knn = ps(
  "k" = p_int(lower = 0, upper = 5, trafo = function(x) 2^x)
)
inst$search_space = searchspace_knn
inst$archive$clear()
gsearch$optimize(inst)
ggplot(as.data.table(inst$archive), aes(x = k, y = classif.ce)) +
  geom_line() + geom_point()
```




```
ggplot(as.data.table(inst$archive), aes(x = x_domain_k, y = classif.ce)) +
  geom_line() + geom_point()
```



SELF-TUNING LEARNER

The `AutoTuner` ...

- optimizes hyperparameters for given `measure`, `resampling`.
- train model on complete data with optimal hyperparameters

```
optlrn = AutoTuner$new(
  learner = lrn("classif.kknn", kernel = "rectangular"),
  resampling = rsmp("holdout"), measure = msr("classif.ce"),
  terminator = trm("none"), tuner = tnr("grid_search", resolution = 10)
  search_space = searchspace_knn)
rr = resample(task = tsk("iris"), learner = optlrn,
  resampling = rsmp("holdout"), store_models = TRUE)

#> INFO  [12:15:56.915] [bbotk] Starting to optimize 1 parameter(s) wit
#> INFO  [12:15:56.943] [bbotk] Evaluating 1 configuration(s)
#> INFO  [12:15:57.037] [bbotk] Result of batch 1:
#> INFO  [12:15:57.039] [bbotk]  k classif.ce
#> INFO  [12:15:57.039] [bbotk]  5       0.18 feeb3181-c148-4fc3-8be8-3
#> INFO  [12:15:57.040] [bbotk] Evaluating 1 configuration(s)
#> INFO  [12:15:57.104] [bbotk] Result of batch 2:
#> INFO  [12:15:57.105] [bbotk]  k classif.ce
#> INFO  [12:15:57.105] [bbotk]  4      0.091 d641e535-c647-49ab-a380-c
#> INFO  [12:15:57.107] [bbotk] Evaluating 1 configuration(s)
#> INFO  [12:15:57.229] [bbotk] Result of batch 3:
#> INFO  [12:15:57.230] [bbotk] k classif.ce
```

PIPELINES TUNING

- Works **exactly** as in basic `mlr3` / `mlr3tuning`
- `PipeOps` have *hyperparameters* (using `paradox` pkg)
- `Graphs` have hyperparameters of all components *combined*
- $\Rightarrow$ Joint **tuning** and nested CV of complete graph

```
p1 = ppl("branch", list(
  "pca" = po("pca"),
  "nothing" = po("nop")
))
p2 = flt("anova")
p3 = ppl("branch", list(
  "svm" = lrn("classif.svm", id = "svm", kernel = "radial",
    type = "C-classification"),
  "xgb" = lrn("classif.xgboost", id = "xgb"),
  "rf" = lrn("classif.ranger",  id = "rf")
), prefix_branchops = "lrn_")
gr = p1 %>>% p2 %>>% p3
glrn = GraphLearner$new(gr)
```

PIPELINES TUNING



PIPELINES TUNING

```
ps = ps(
  branch.selection = p_fct(levels = c("pca", "nothing")),
  anova.filter.frac = p_dbl(lower = 0.1, upper = 1),
  lrn_branch.selection = p_fct(levels = c("svm", "xgb", "rf")),
  rf.mtry = p_int(lower = 1L, upper = 20L),
  xgb.nrounds = p_int(lower = 1, upper = 500),
  svm.cost = p_dbl(lower = -12, upper = 4),
  svm.gamma = p_dbl(lower = -12, upper = -1))
ps$add_dep("rf.mtry", "lrn_branch.selection", CondEqual$new("rf"))
ps$add_dep("xgb.nrounds", "lrn_branch.selection", CondEqual$new("xgb"))
ps$add_dep("svm.cost", "lrn_branch.selection", CondEqual$new("svm"))
ps$add_dep("svm.gamma", "lrn_branch.selection", CondEqual$new("svm"))
ps$trafo = function(x, param_set) {
  if (x$lrn_branch.selection == "svm") {
    x$svm.cost = 2^x$svm.cost; x$svm.gamma = 2^x$svm.gamma
  }
  return(x)
}
inst = TuningInstanceSingleCrit$new(tsk("sonar"), glrn, rsmp("cv", folds=3),
  msr("classif.ce"), ps, trm("evals", n_evals = 10))
tnr("random_search")$optimize(inst)
```

mlr3fselect

MLR3FSELECT

Main author: Martin Becker (LMU)

Feature selection methods:

- Filtering: `mlr3filters`

```
po("filter", filter = mlr3filters::flt("variance"), filter.frac = 0.5)
```

- Embedded Methods

```
if("selected_features" %in% learner$properties)
  learner$selected_features()
```

- **Wrapper Methods:** `mlr3fselect`

FEATURE SELECTION OPTIMIZE

Feature selection is a combinatorial optimization problem.
→ Usage very similar to `mlr3tuning`.

- need to choose a *feature selection* method (basically combinatorial optimizers)
- `fs()` to create `FSelector` objects
- ```
  as.data.table(fs())
  #>                  key
  #>               <char>
  #> 1:     design_points
  #> 2: exhaustive_search
  #> 3:    genetic_search
  #> 4:     random_search
  #> 5:              rfe
  #> 6:       sequential
  ```
  ```

MLR3FSELECT

```
library(mlr3fselect)
inst = FSelectInstanceSingleCrit$new(
  task = tsk("pima"),
  learner = lrn("classif.rpart"),
  resampling = rsmp("holdout"), measure = msr("classif.ce"),
  terminator = trm("evals", n_evals = 20))
fsel = fs("random_search")
fsel$optimize(inst)

#>       age glucose insulin   mass pedigree pregnant pressure triceps
#>    <lgcl>  <lgcl>  <lgcl> <lgcl>   <lgcl>   <lgcl>   <lgcl>  <lgcl>
#> 1:   TRUE    TRUE    TRUE  FALSE     TRUE     TRUE     TRUE   FALSE
#>                                        features classif.ce
#>                                          <list>      <num>
#> 1: age,glucose,insulin,pedigree,pregnant,pressure       0.27
```

Remember: `inst$archive` contains all evaluations.

mlr3proba

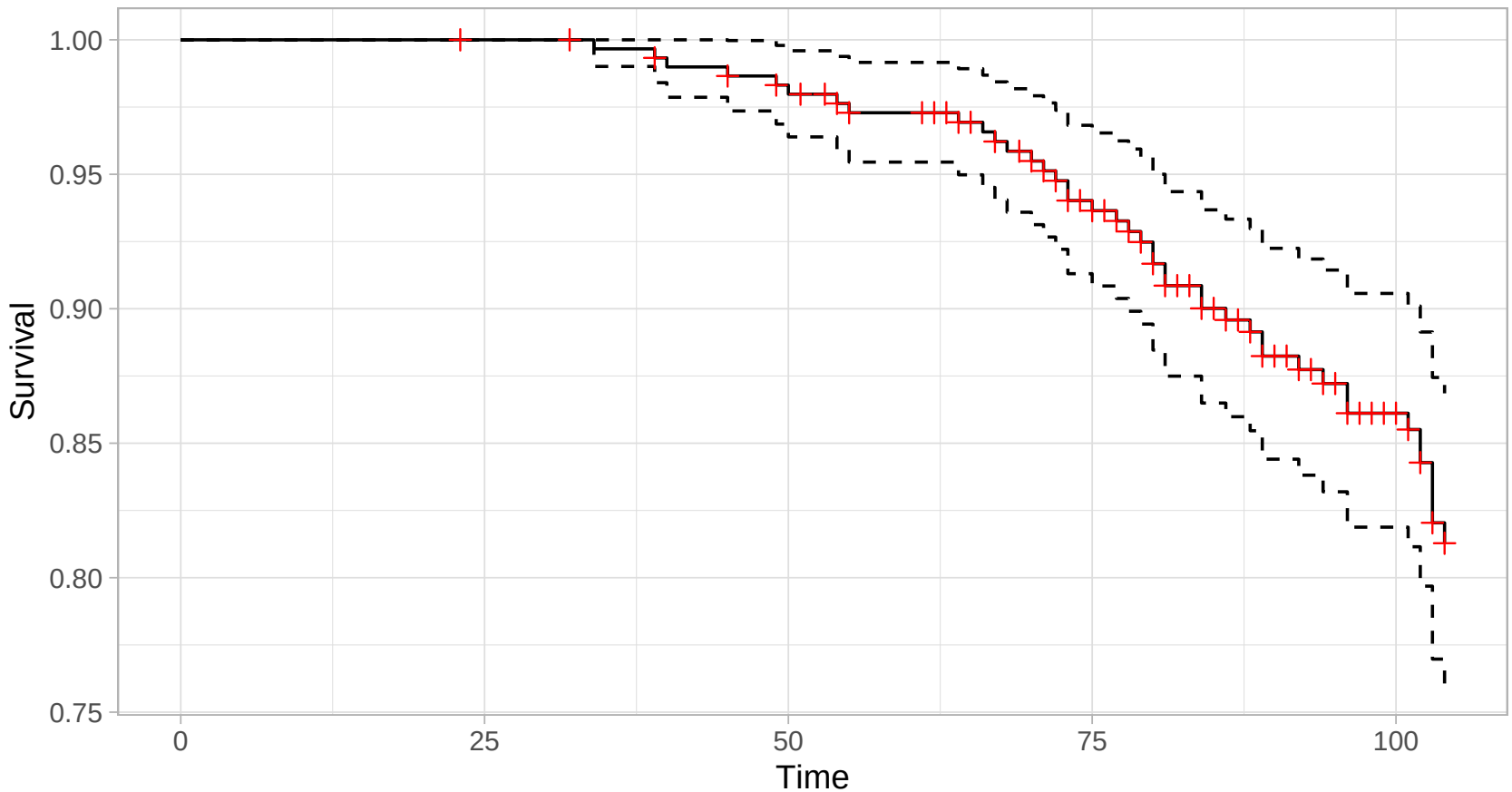
MLR3PROBA

Main author: Raphael Sonabend (UCL)

- Survival Task
  - `type`: right, left, interval cens. (...)
  - `time`: event time (right cens.) / starting time (interval cens.)
  - `time2`: ending Time (interval cens.)
  - `event`:
    - 0 no event, 1 event (right cens.)
    - 0 right cens., 1 event, 2 left cens., 3 interval cens. (interval cens.)
- Survival Learners
  Prediction types
  - prognostic index (continuous rank, `crank`)
  - probabilistic (distribution, `distr`)
  - linear predictor (`lp`)
- Survival Measures

MLR3PROBA: EXAMPLE

```
library("mlr3proba")
task = as_task_surv(
  survival::rats, id = "rats_rcens", time = "time",
  event = "status", type = "right")
autoplot(task)
```



MLR3PROBA: LEARNERS

```
as.data.table(lrns()) %>%
  .[grepl("surv", key),.(key, properties, predict_types)] %>%
  head(12)

#>                    key properties  predict_types
#>                 <char>     <list>          <list>
#>  1:     surv.akritas                 crank,distr
#>  2:  surv.blackboost    weights distr,crank,lp
#>  3:     surv.cforest    weights    distr,crank
#>  4:    surv.coxboost    weights distr,crank,lp
#>  5:       surv.coxph    weights distr,crank,lp
#>  6:     surv.coxtime                 crank,distr
#>  7:       surv.ctree    weights    distr,crank
#>  8: surv.cv_coxboost    weights distr,crank,lp
#>  9:   surv.cv_glmnet    weights       crank,lp
#> 10:     surv.deephit                 crank,distr
#> 11:    surv.deepsurv                 crank,distr
#> 12:     surv.dnnsurv                 crank,distr
```

MLR3PROBA: MEASURES

```
as.data.table(msrs()) %>%
  .[task_type == "surv", .(key, predict_type)] %>% head(15)

#>                     key predict_type
#>                  <char>       <char>
#>  1:          surv.brier        distr
#>  2:   surv.calib_alpha        distr
#>  3:    surv.calib_beta           lp
#>  4: surv.chambless_auc           lp
#>  5:        surv.cindex        crank
#>  6:          surv.graf        distr
#>  7:      surv.hung_auc           lp
#>  8:    surv.intlogloss        distr
#>  9:       surv.logloss        distr
#> 10:           surv.mae     response
#> 11:           surv.mse     response
#> 12:     surv.nagelk_r2           lp
#> 13:   surv.oquigley_r2           lp
#> 14:           surv.rmse     response
#> 15:        surv.schmid        distr
```

MLR3PROBA: EXAMPLE

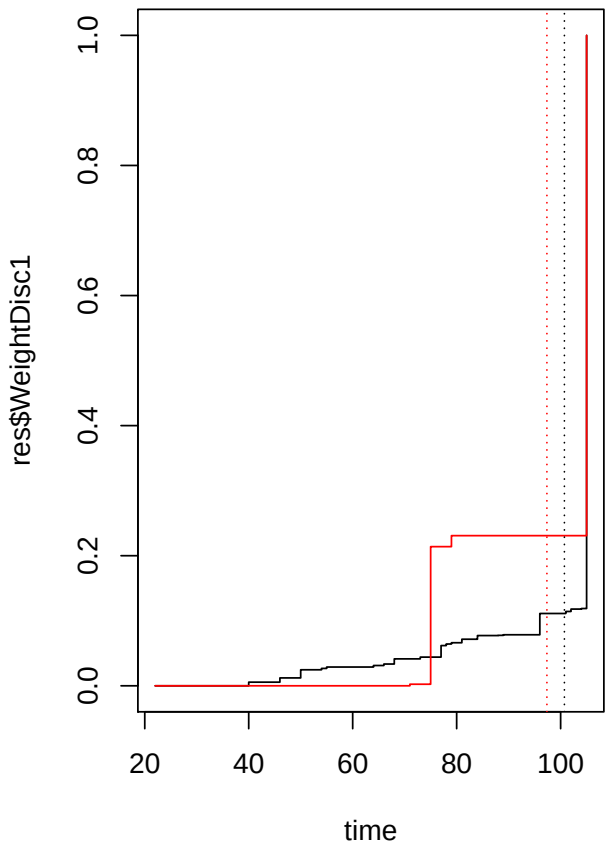
We can calculate *cindex* for `predict_type = "distr"` as negative *estimated survival expectation*.

```
task = tsk("rats")$select(c("litter", "rx")) #remove factor column
learners = lrns(c("surv.coxph", "surv.ranger", "surv.glmnet"))
bmr = benchmark(benchmark_grid(task, learners, rsmp("cv", folds = 3)))
bmr$aggregate(msr("surv.cindex"))

#>       nr      resample_result task_id  learner_id resampling_id
#>    <int>               <list>  <char>      <char>        <char>
#> 1:     1 <ResampleResult[21]>    rats  surv.coxph            cv
#> 2:     2 <ResampleResult[21]>    rats surv.ranger            cv
#> 3:     3 <ResampleResult[21]>    rats surv.glmnet            cv
#>    iters surv.harrell_c
#>    <int>          <num>
#> 1:     3           0.60
#> 2:     3           0.63
#> 3:     3           0.61

bmr$aggregate(msr("surv.brier")) # does not work bc. of glmnet

#> Error in assertThat(object, inherits(object, "Distribution"),
errormsg):  NA is not an R6 Distribution object
```

MLR3PROBA: DISTRIBUTIONS

Predictions are based on `distr6::VectorDistribution` objects:

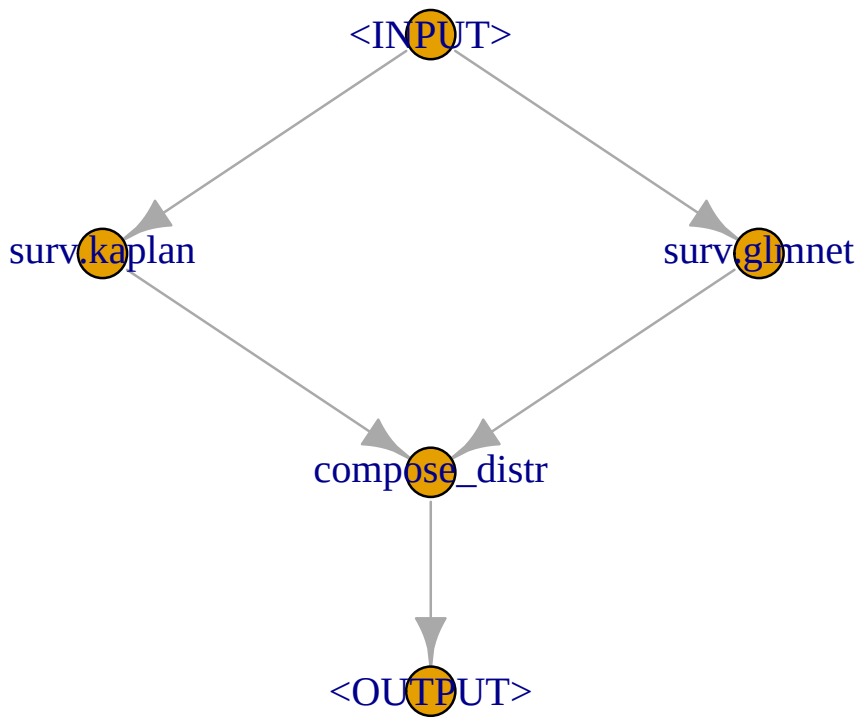
```
tsk_rats = tsk("rats")
learn = lrn("surv.ranger")
pred_ids = c(50, 300)
time = seq(22, 105)
learn$train(tsk_rats, row_ids =
  setdiff(1:300, pred_ids))
pred = learn$predict(tsk_rats,
  row_ids = pred_ids)
res = pred$distr$cdf(time)
plot(time, res$WeightDisc1, type = "s")
lines(time, res$WeightDisc2, type = "s",
  col = "red")
abline(v = -pred$crank,
  col = c("black", "red"), lty = 3)
```



MLR3PROBA: COMPOSITOR

We can calculate *Brier score* (Integrated Graf Score) for `predict_type = "crank"` by combining Kaplan–Meier estimator with "`crank` only" Learner using a predefined pipeline:

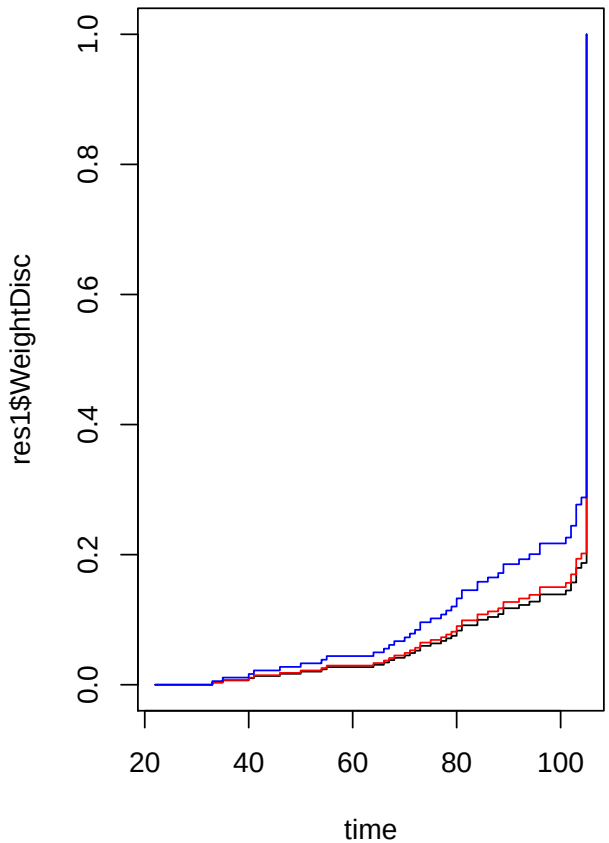
```
glm_distr = ppl("distrcompositor", learner = lrn("surv.glmnet"),
  estimator = "kaplan", form = "ph")
plot(glm_distr)
```



MLR3PROBA: COMPOSITOR

"Scale" Kaplan-Meier according to `cranks` from `surv.glmnet`:

```
lrn1 = lrn("surv.kaplan")
lrn2 = as_learner(glm_distr)
lrn1$train(task); lrn2$train(task)
# Kaplan-Meier the same for all
pred1 = lrn1$predict(task, row_ids = 1)
pred2 = lrn2$predict(task, row_ids = pred_ids)
res1 = pred1$distr$cdf(time)
res2 = pred2$distr$cdf(time)
plot(time, res1$WeightDisc, type = "s")
lines(time, res2$WeightDisc1, type = "s",
  col = "red")
lines(time, res2$WeightDisc2, type = "s",
  col = "blue")
```



MLR3PROBA: EXAMPLE

```
glm_distr = ppl("distrcompositor", learner = lrn("surv.glmnet"),
  estimator = "kaplan", form = "ph")
learners[[3]] = as_learner(glm_distr)
learners[[4]] = lrn("surv.kaplan")
bmr = benchmark(benchmark_grid(task, learners, rsmp("cv", folds = 3)))
bmr$aggregate(msr("surv.brier"))[,.(learner_id, surv.graf)]

#>                                learner_id surv.graf
#>                                    <char>     <num>
#> 1:                              surv.coxph     0.072
#> 2:                             surv.ranger     0.074
#> 3: surv.kaplan.surv.glmnet.compose_distr     0.073
#> 4:                             surv.kaplan     0.071

bmr$aggregate(msr("surv.cindex"))[,.(learner_id, surv.harrell_c)]

#>                                learner_id surv.harrell_c
#>                                    <char>          <num>
#> 1:                              surv.coxph           0.58
#> 2:                             surv.ranger           0.59
#> 3: surv.kaplan.surv.glmnet.compose_distr           0.58
#> 4:                             surv.kaplan           0.50
```

MLR3PROBA: MORE

Not covered here:

- Other methods to obtain `crank` from `distr` prediction using `PipeOpCrankCompositor`
- Transform `PredictionRegr` to `PredictionSurv` using `PipeOpPredRegrSurv`
- Average `PredictionSurv` using `PipeOpSurvAvg`
- ...

How to get Help

HOW TO GET HELP

- Where to start?
  - Check these slides
  - **Check the mlr3book https://mlr3book.mlr-org.com**
- Get help for R6 objects?
  1. Find out what kind of R6 object you have:

```
class(bmr)
#> [1] "BenchmarkResult" "R6"
```

  2. Go to the corresponding help page:

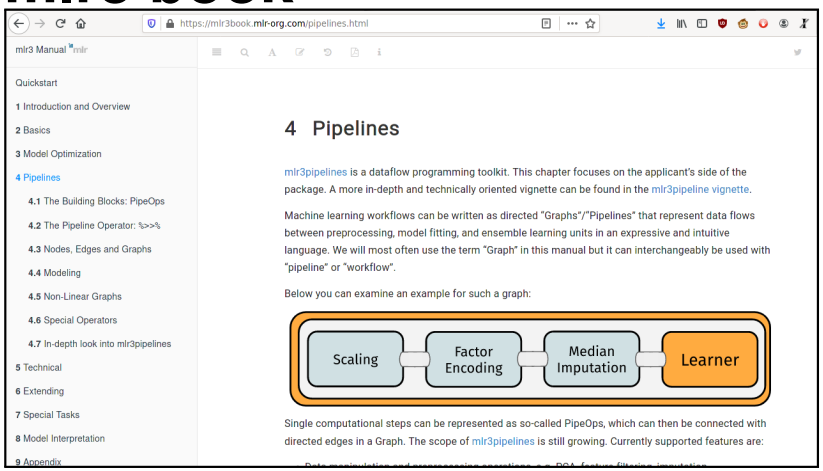
```
?BenchmarkResult
```

  Open the corresponding man page with

```
learner$help()
```

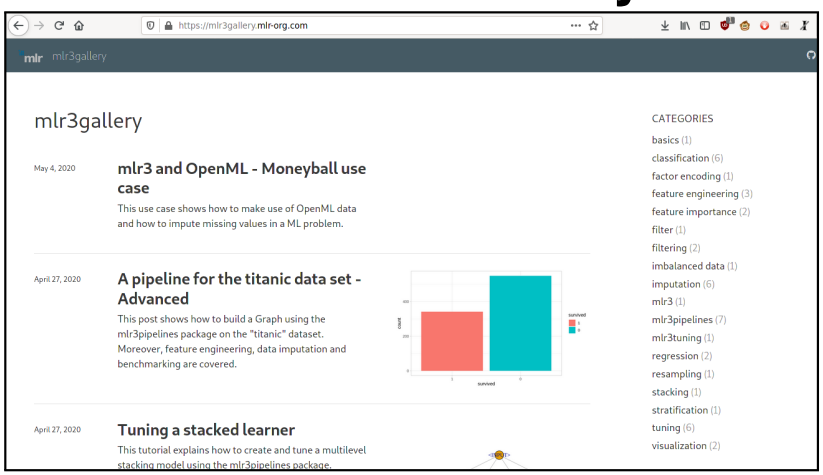
MLR3 RESOURCES

mlr3 book



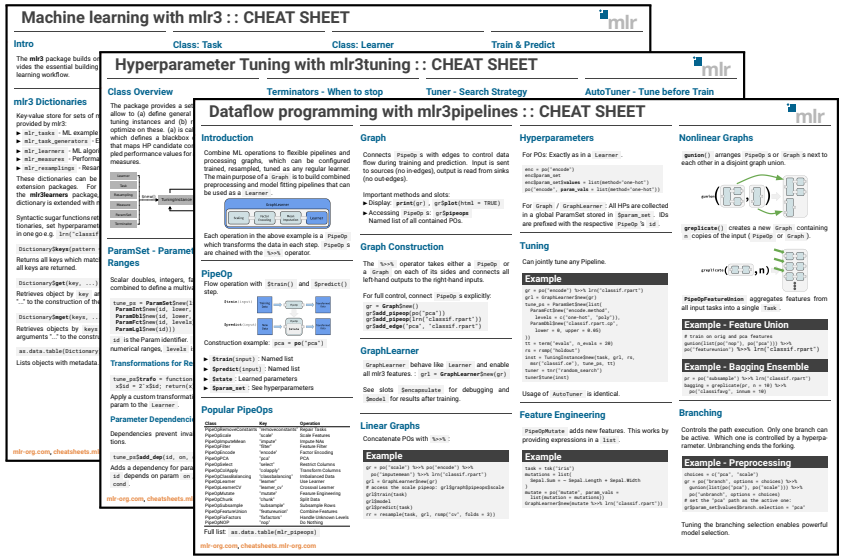
https://mlr3book.mlr-org.com/

mlr3 Use Case Gallery



https://mlr3gallery.mlr-org.com/

Cheat Sheets



https://cheatsheets.mlr-org.com/

More:

- Stackoverflow: https://stackoverflow.com/tags/mlr3
- Mattermost channel: https://lmmisld-lmu-stats-slds.srv.mwn.de/mlr_invite/
- GitHub Issue in one of the projects: https://github.com/mlr-org/

OUTLOOK

What was not covered today?

- Error handling (fallback learners)
- Database backends
- Cost-sensitive classification
- Model interpretation (interpretable machine learning)
- ...

What is to come?

- `mlr3pipelines`: caching, parallelization
- Forecasting via `mlr3forecast`
- Deep Learning via `mlr3keras`
- More optimizers via `mlr3mbo`, `miesmuschl`

Thanks! Please ask questions!