LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

mlr3 and ML pipelines

`https://tinyurl.com/rxaxe3x`

Bernd Bischl, Michel Lang, Martin Binder, and many others

Department of Statistics – LMU Munich

November 15, 2019

MLR2

- Unified interface to machine learning algorithms in R with a common infrastructure for ML tasks
- 89 classification, 59 regression, 12 survival, 10 clustering, 3 multi-label learners
- Added value: benchmarking, feature selection through wrappers, cost-sensitive classification, hyperparameter tuning...

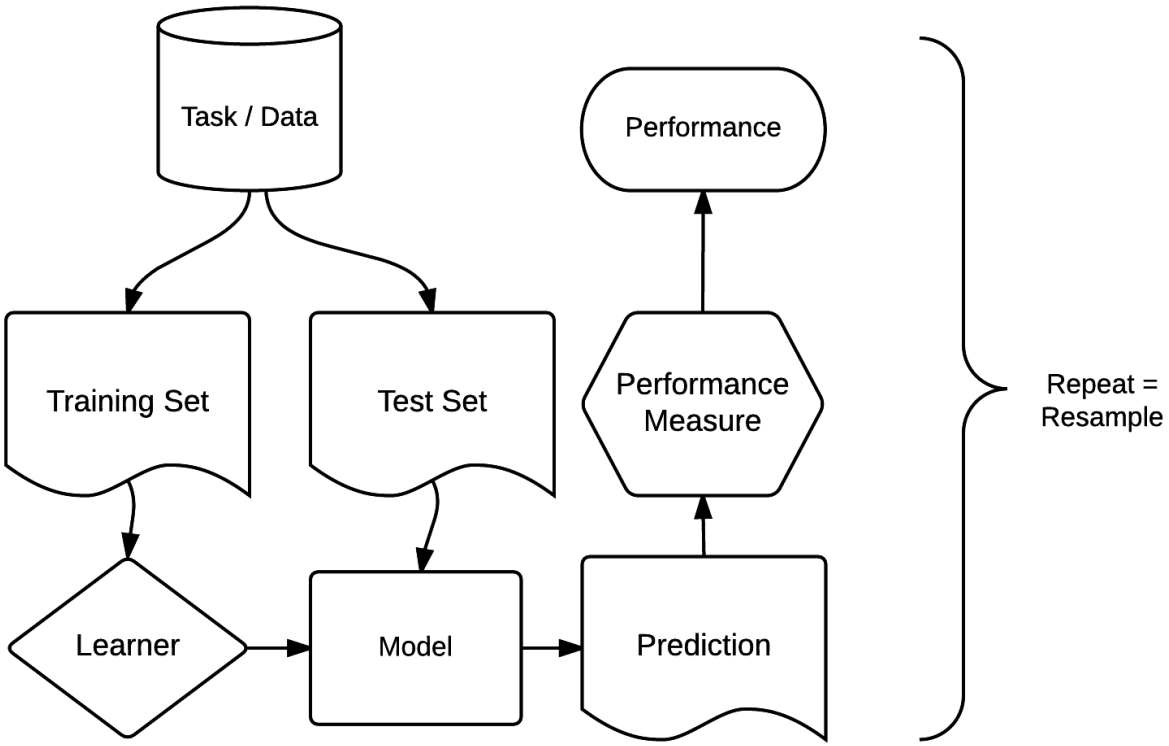
MLR3: THE NEXT GENERATION

- mlr is a large monolithic package which integrated hundreds of other R packages
  - Difficult to maintain and extend
  - Hundreds of dependencies to install
  - Changes in other learners (without tests) broke tests in mlr and prevented releases
  - Long build and test times
  - R ecosystem evolved
- mlr3 completely redesigned and reimplemented from the ground up as ecosystem of modular small packages
- mlr now in maintenance-only mode

MLR3 DESIGN PRINCIPLES

- Overcome limitations of S3 with the help of **R6**
  - Truly object-oriented (OO): data and methods together
  - Inheritance
  - Reference semantics
- Embrace **data.table**, both for arguments and for internal data structures
  - Fast operations for tabular data
  - Better support for list columns to arrange complex objects in a tabular structure
  - Reference semantics
- Be **light on dependencies**. Direct and recursive dependencies:
  - `R6`, `data.table`, `digest`, `lgr`, `uuid`
  - Some self-maintained packages (`backports`, `checkmate`, `...`)

MLR3 BUILDING BLOCKS



EXAMPLE: TRAIN + PREDICT

```
task = TaskClassif$new("iris", iris, target = "Species")
learner = lrn("classif.rpart")

learner$train(task, 1:120)
learner$model

#> n= 120
#>
#> node), split, n, loss, yval, (yprob)
#>       * denotes terminal node
#>
#> 1) root 120 70 setosa (0.41667 0.41667 0.16667)
#>   2) Petal.Length< 2.45 50  0 setosa (1.00000 0.00000 0.00000) *
#>   3) Petal.Length>=2.45 70 20 versicolor (0.00000 0.71429 0.28571)
#>     6) Petal.Length< 4.95 49  1 versicolor (0.00000 0.97959 0.02041) *
#>     7) Petal.Length>=4.95 21  2 virginica (0.00000 0.09524 0.90476) *
```

EXAMPLE: TRAIN + PREDICT

```
p = learner$predict(task, 121:150)
head(as.data.table(p), 3)

#>    row_id     truth   response
#> 1:    121 virginica  virginica
#> 2:    122 virginica versicolor
#> 3:    123 virginica  virginica

p$confusion

#>             truth
#> response     setosa versicolor virginica
#>   setosa          0          0         0
#>   versicolor      0          0         5
#>   virginica       0          0        25

p$score(msr("classif.acc"))

#> classif.acc
#>      0.8333
```

EXAMPLE: RESAMPLE

```
r = resample(tsk("iris"), lrn("classif.rpart"), rsmp("cv", folds = 3))
r$score()

#>             task task_id                 learner    learner_id
#> 1: <TaskClassif>    iris <LearnerClassifRpart> classif.rpart
#> 2: <TaskClassif>    iris <LearnerClassifRpart> classif.rpart
#> 3: <TaskClassif>    iris <LearnerClassifRpart> classif.rpart
#>        resampling resampling_id iteration prediction classif.ce
#> 1: <ResamplingCV>            cv         1     <list>       0.08
#> 2: <ResamplingCV>            cv         2     <list>       0.08
#> 3: <ResamplingCV>            cv         3     <list>       0.06

r$aggregate()

#> classif.ce
#>    0.07333
```

EXAMPLE: BENCHMARKING

```
grid = benchmark_grid(
  tasks = tsk("iris"),
  learners = lrns(c("classif.featureless", "classif.rpart")),
  resamplings = rsmp("cv", folds = 3)
)
bmr = benchmark(grid)
aggr = bmr$aggregate(msr("classif.acc"))
aggr[, c("learner_id", "classif.acc")]

#>            learner_id classif.acc
#> 1: classif.featureless     0.2667
#> 2:       classif.rpart     0.9333
```

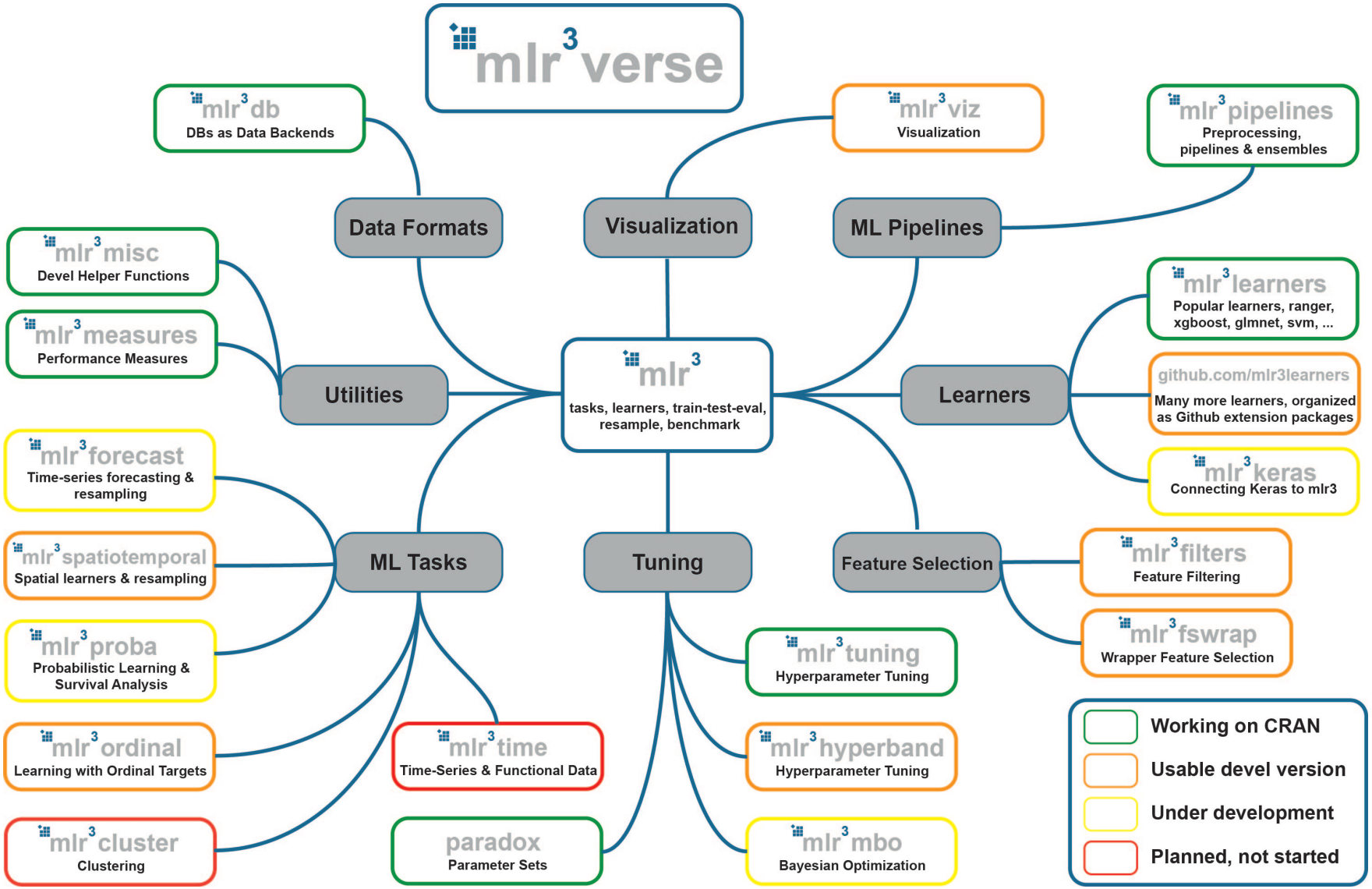
PARALLELIZATION

- Parallelization via `future`:

```
future::plan("multicore")
benchmark(design) # now parallel
```

- Backends: isolated R session (`callr`), multicore, socket clusters, MPI clusters, HPCs (`batchtools`), ...
- Each train-predict step is a single job

MLR3 ECOSYSTEM



mlr3tuning

TUNING

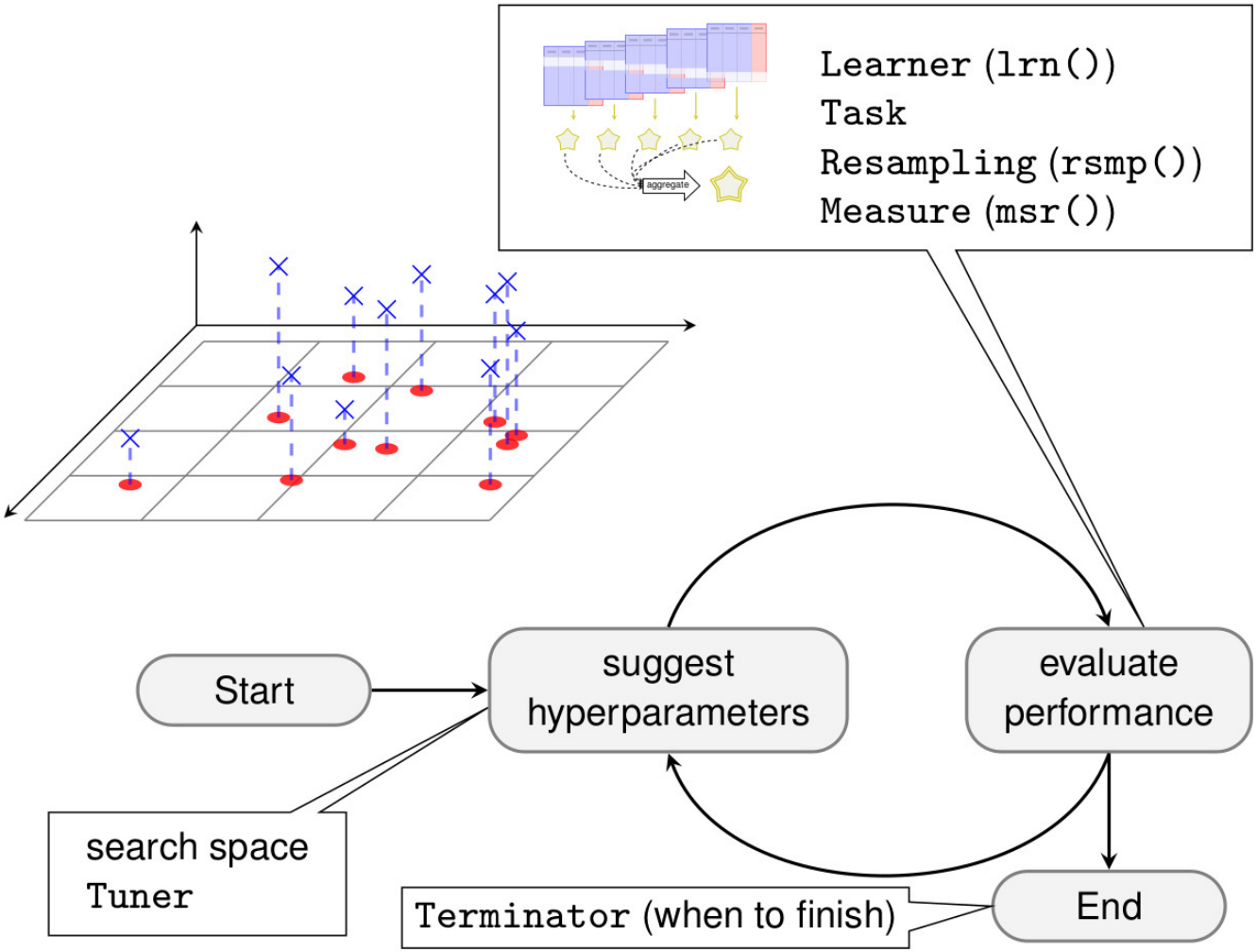
- Behavior of most methods depends on *hyperparameters*
- We want to choose them so our algorithm performs well
- Good hyperparameters are data-dependent

$\Rightarrow$ We do *black box optimization* ("Try stuff and see what works")

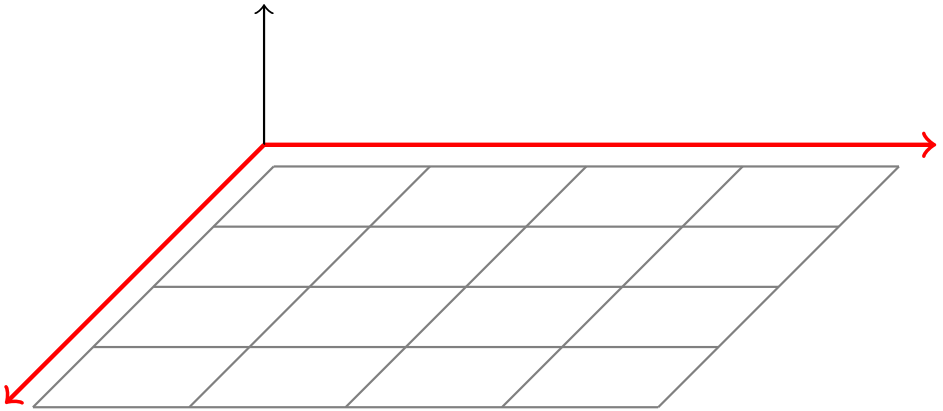
Tuning toolbox for `mlr3`:

```
library("mlr3tuning")
```

TUNING


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

SEARCH SPACE



```
ParamSet$new(list(param1, param2, ...))
```

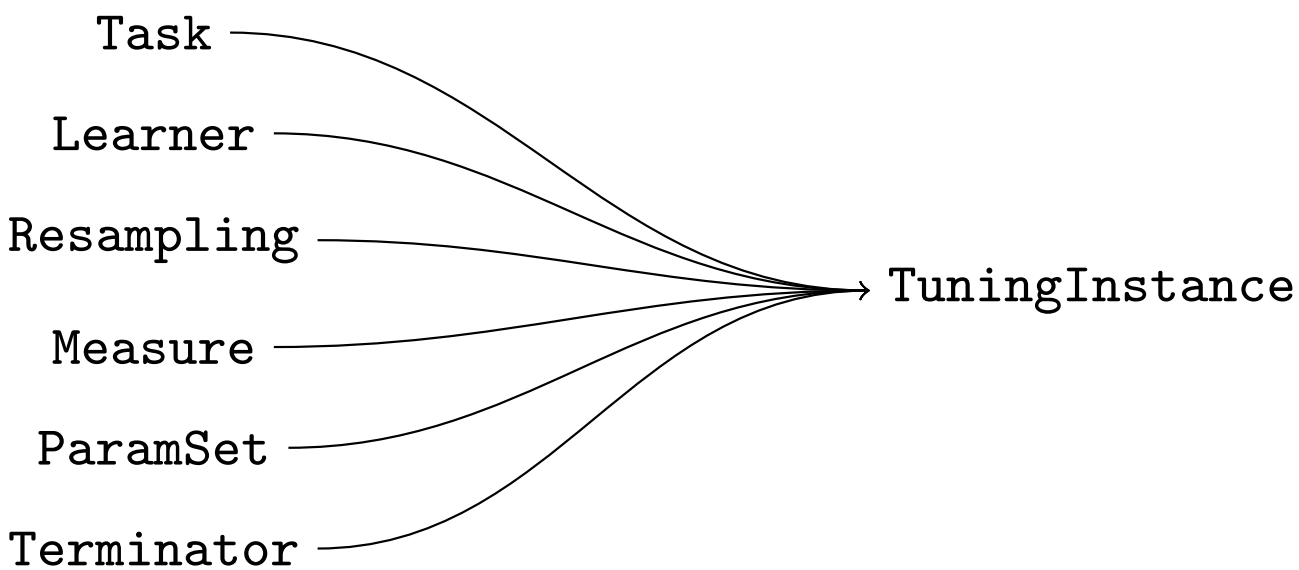
Numerical parameter	`ParamDbl$new(id, lower, upper)`
Integer parameter	`ParamInt$new(id, lower, upper)`
Discrete parameter	`ParamFct$new(id, levels)`
Logical parameter	`ParamLgl$new(id)`
Untyped parameter	`ParamUty$new(id)`

```
library("paradox")
searchspace_knn = ParamSet$new(list(
  ParamInt$new("k", 1, 20)
))
```

TUNING AN INSTANCE

TuningInstance defines

- Black-box objective function
- Search space
- Termination criterion



TUNING AN INSTANCE

```
inst = TuningInstance$new(
  tsk("iris"), lrn("classif.kknn", kernel="rectangular"),
  rsmp("cv"), msr("classif.ce"),
  searchspace_knn, term("evals", n_evals=20))
rsearch = tnr("random_search")
rsearch$tune(inst)
inst$result[c(1,3)]

#> $tune_x
#> $tune_x$k
#> [1] 15
#>
#>
#> $perf
#> classif.ce
#>    0.03333
```

TUNING AN INSTANCE

The instance stores the same data structure as `mlr3::benchmark`

```
as.data.table(inst$bmr)[1:2,]

#>                                    uhash          task
#> 1: 08981c6c-a7e6-447d-a523-289312efbc18 <TaskClassif>
#> 2: 08981c6c-a7e6-447d-a523-289312efbc18 <TaskClassif>
#>                 learner     resampling iteration prediction
#> 1: <LearnerClassifKKNN> <ResamplingCV>         1     <list>
#> 2: <LearnerClassifKKNN> <ResamplingCV>         2     <list>

inst$archive()[1:2, ]

#>    nr batch_nr  resample_result task_id   learner_id resampling_id
#> 1:  1        1 <ResampleResult>    iris classif.kknn            cv
#> 2:  2        2 <ResampleResult>    iris classif.kknn            cv
#>    iters params tune_x warnings errors classif.ce
#> 1:    10 <list> <list>        0      0    0.04667
#> 2:    10 <list> <list>        0      0    0.06667
```

NESTED RESAMPLING

- Need to perform nested resampling to estimate tuned learner performance

$\Rightarrow$ Treat tuning as if it were a `Learner`!

  - Training:
    1. Tune model using (inner) resampling
    2. Train final model with best parameters on all (i.e. outer resampling) data
  - Predicing: Just use final model

- **AutoTuner**

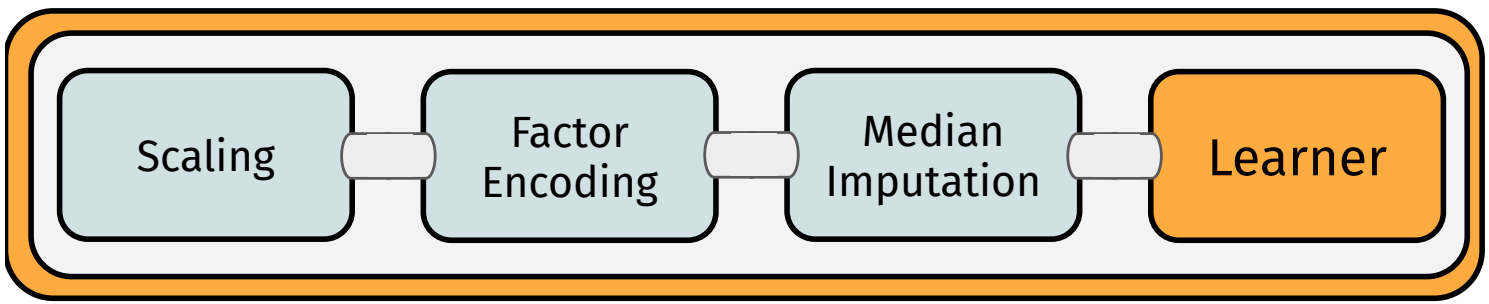
```
optlrn = AutoTuner$new(lrn("classif.kknn", kernel="rectangular"),
  rsmp("cv"), msr("classif.ce"), searchspace_knn,
  term("evals", n_evals = 20), tnr("random_search"))
resample(tsk("iris"), optlrn, rsmp("cv"))
```

mlr3pipelines

MACHINE LEARNING WORKFLOWS

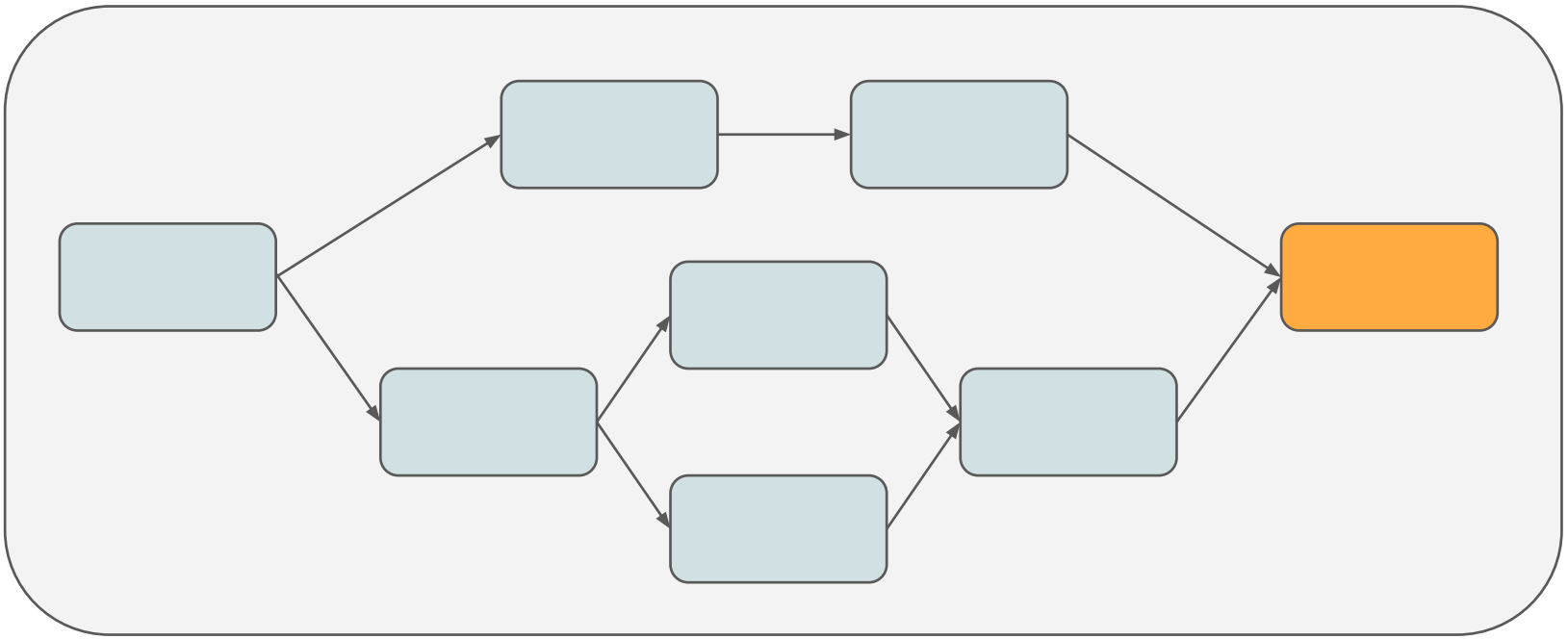
- **Preprocessing**: Feature extraction, feature selection, missing data imputation,...
- **Ensemble methods**: Model averaging, model stacking
- `mlr3`: modular model fitting

$\Rightarrow$ `mlr3pipelines`: modular ML workflows



MACHINE LEARNING WORKFLOWS

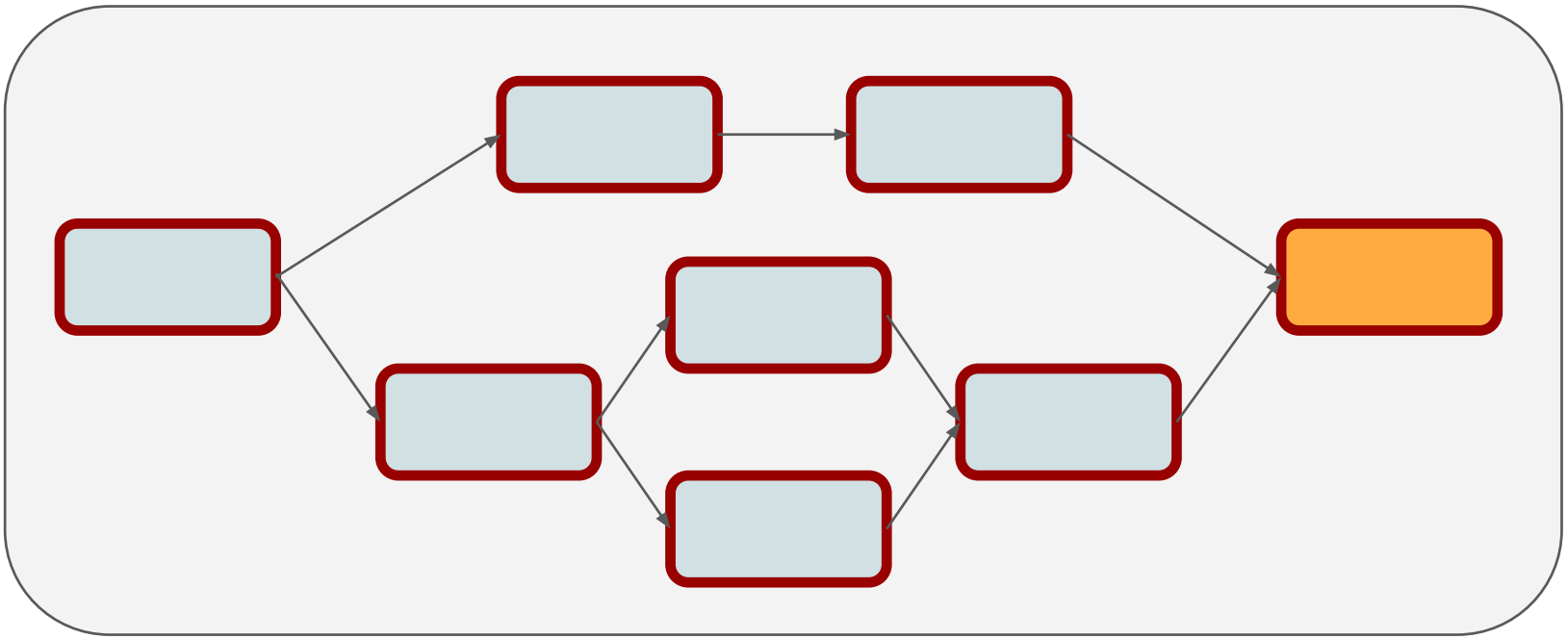
– what do they look like?



MACHINE LEARNING WORKFLOWS

– what do they look like?

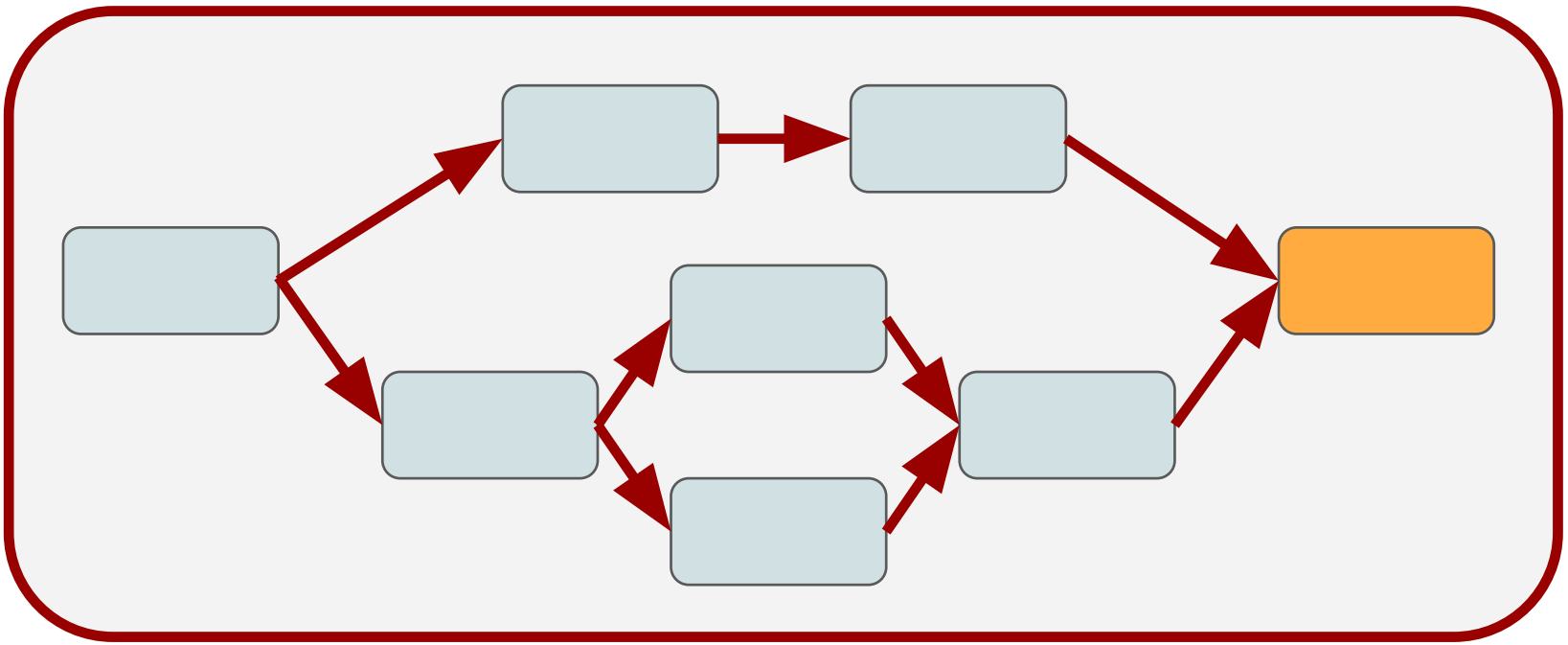
- **Building blocks**: *what* is happening? $\rightarrow$ `PipeOp`



MACHINE LEARNING WORKFLOWS

– what do they look like?

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

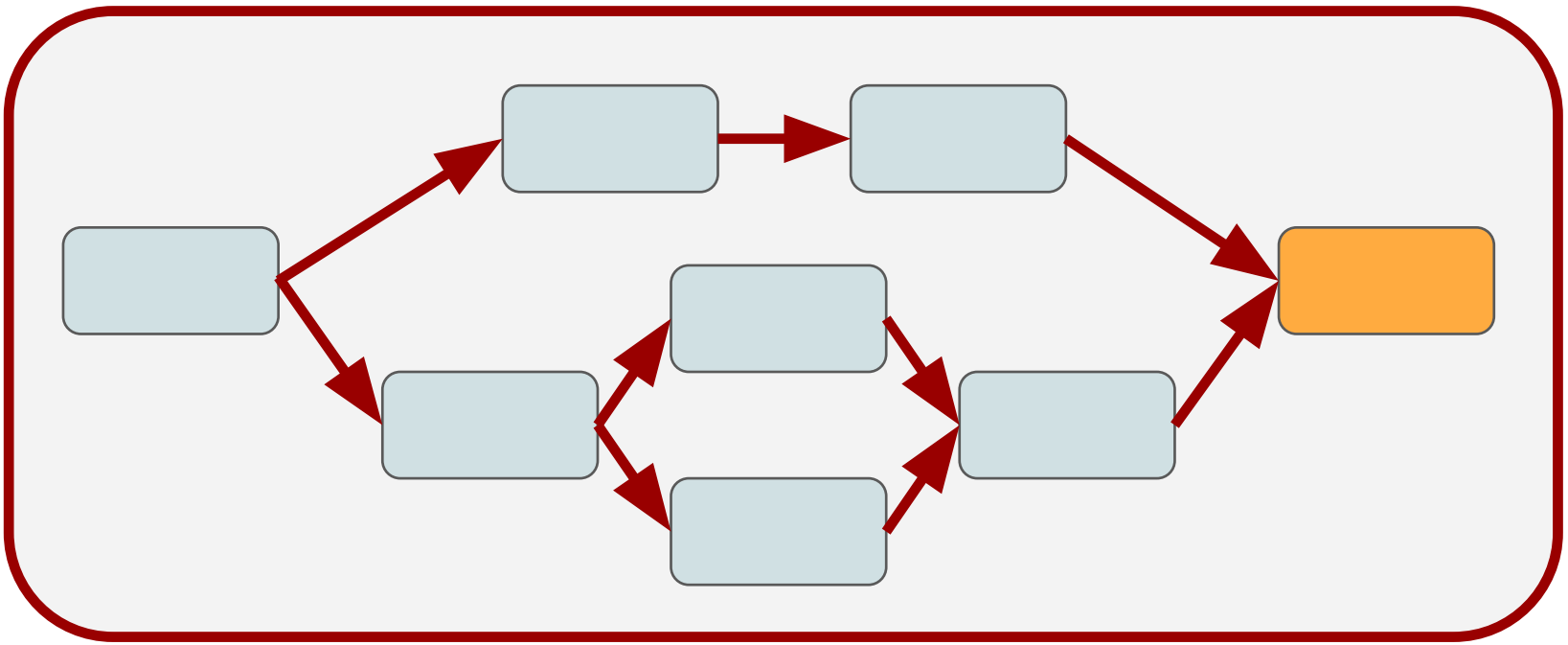


MACHINE LEARNING WORKFLOWS

– 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



PipeOps

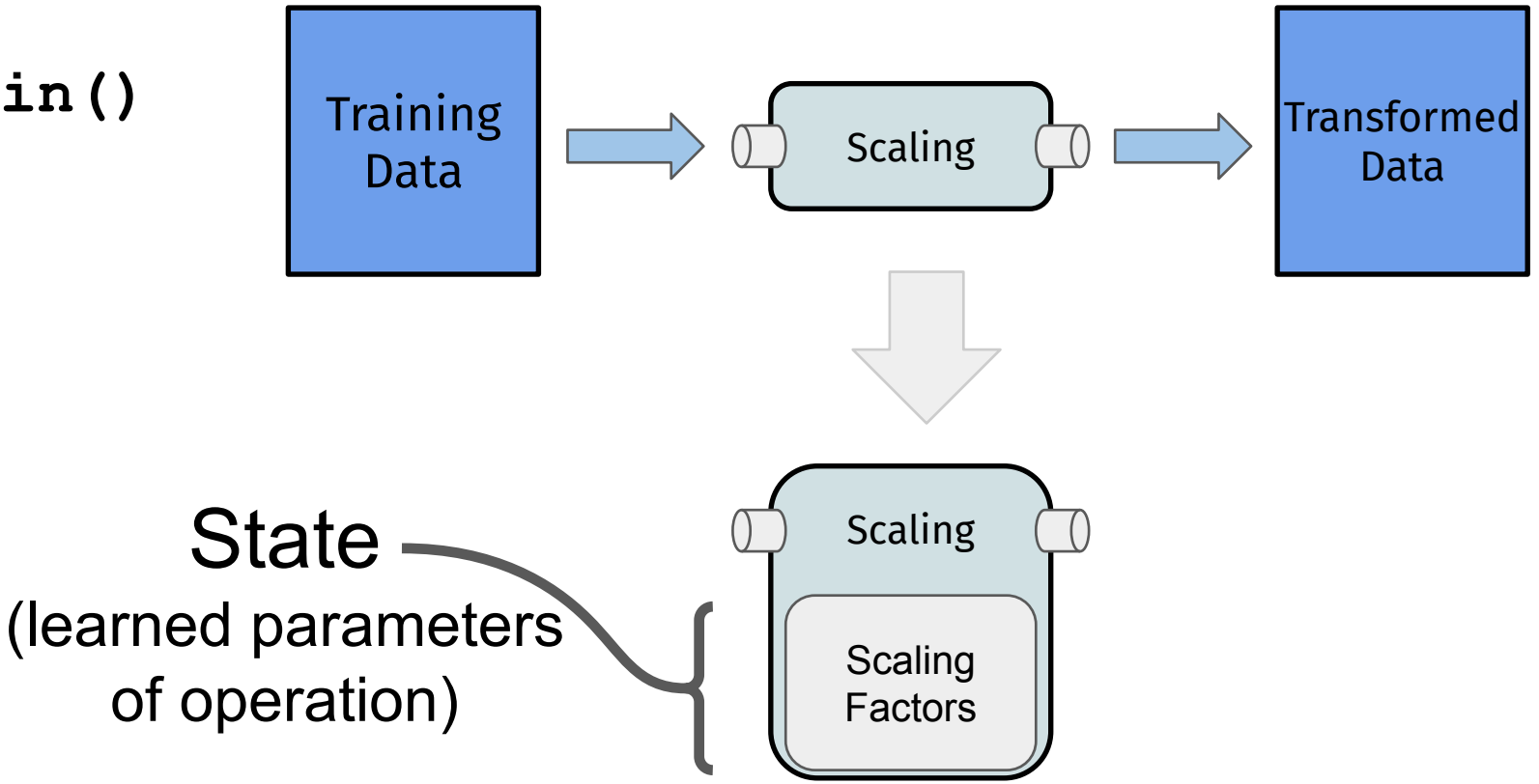
PIPEOP: SINGLE UNIT OF DATA OPERATION

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

PIPEOP: SINGLE UNIT OF DATA OPERATION

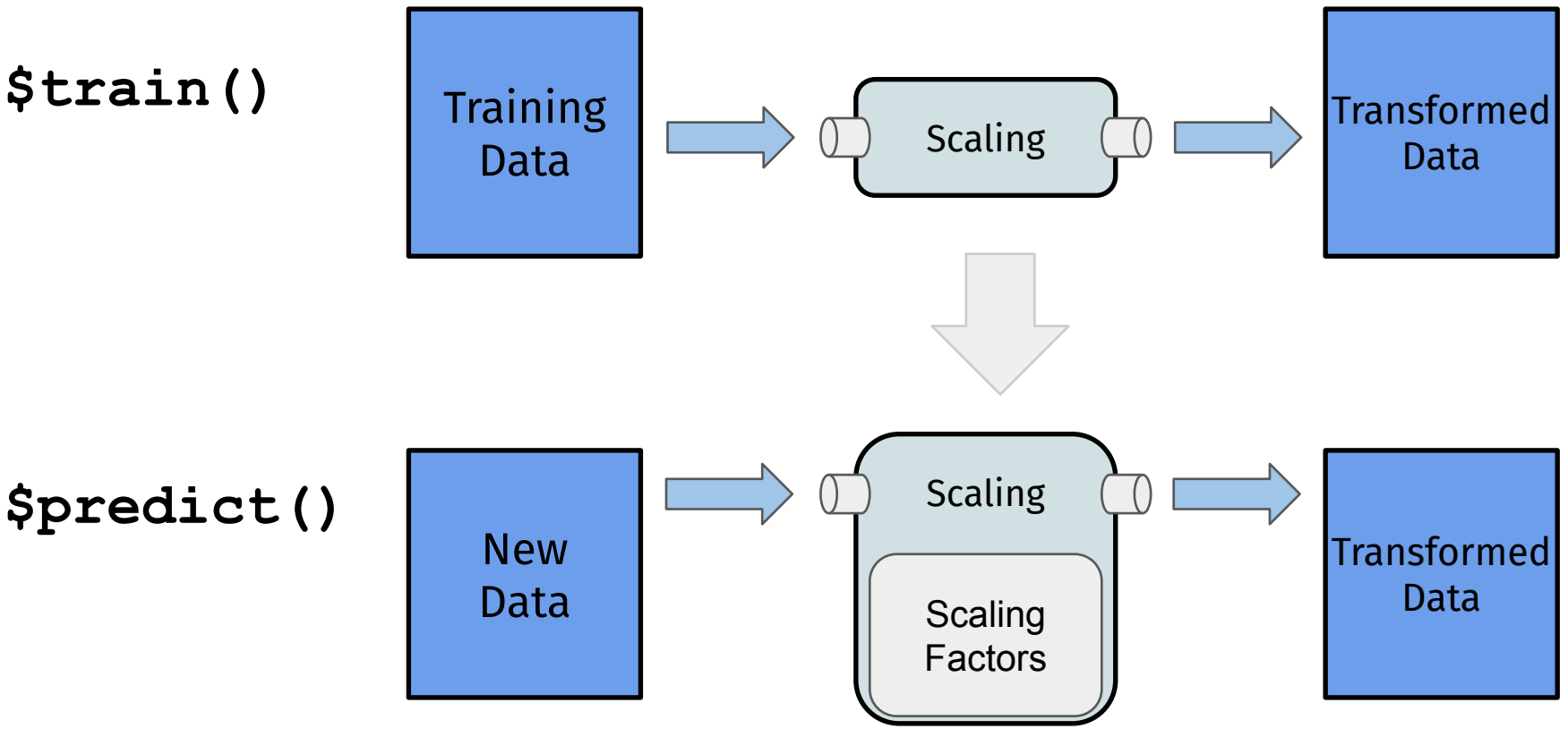
`pip$train()`: process data and create `pip$state`

`$train()`



PIPEOP: SINGLE UNIT OF DATA OPERATION

`pip$predict()`: process data depending on the `pip$state`



PIPEOP: SINGLE UNIT OF DATA OPERATION

```
po = po("scale")

trained = po$train(list(task))

trained[[1]]$head(3)

#>    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#> 1:  setosa       -1.336      -1.311       -0.8977      1.0156
#> 2:  setosa       -1.336      -1.311       -1.1392     -0.1315
#> 3:  setosa       -1.392      -1.311       -1.3807      0.3273
```

PIPEOP: SINGLE UNIT OF DATA OPERATION

```
po = po("scale")

trained = po$train(list(task))

trained[[1]]$head(3)

#>    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#> 1:  setosa       -1.336      -1.311       -0.8977      1.0156
#> 2:  setosa       -1.336      -1.311       -1.1392     -0.1315
#> 3:  setosa       -1.392      -1.311       -1.3807      0.3273
```

```
head(po$state, 2)

#> $center
#> Petal.Length  Petal.Width Sepal.Length  Sepal.Width
#>        3.758        1.199        5.843        3.057
#>
#> $scale
#> Petal.Length  Petal.Width Sepal.Length  Sepal.Width
#>       1.7653       0.7622       0.8281       0.4359
```

PIPEOP: SINGLE UNIT OF DATA OPERATION

```
po = po("scale")

trained = po$train(list(task))

trained[[1]]$head(3)

#>    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#> 1:  setosa       -1.336      -1.311       -0.8977      1.0156
#> 2:  setosa       -1.336      -1.311       -1.1392     -0.1315
#> 3:  setosa       -1.392      -1.311       -1.3807      0.3273
```

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

#>    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#> 1:  setosa       -1.336      -1.311       -0.8977      1.0156
#> 2:  setosa       -1.336      -1.311       -1.1392     -0.1315
#> 3:  setosa       -1.392      -1.311       -1.3807      0.3273
```

LIST OF PIPEOPS

Included

- Simple preprocessors (scaling, Box-Cox, Yeo-Johnson, PCA, ICA)
- NA imputation (constant, hist-sampling, model-based, dummies)
- Feature filtering (by name, by type, statistical filters)
- Categorical data encoding (one-hot, treatment, impact)
- Combination of data: `featureunion`
- Sampling (subsampling for speed, sampling for class balance)
- Ensembling of predictions (weighted average, optimized weights)
- Branching (simultaneous branching, alternative branching)

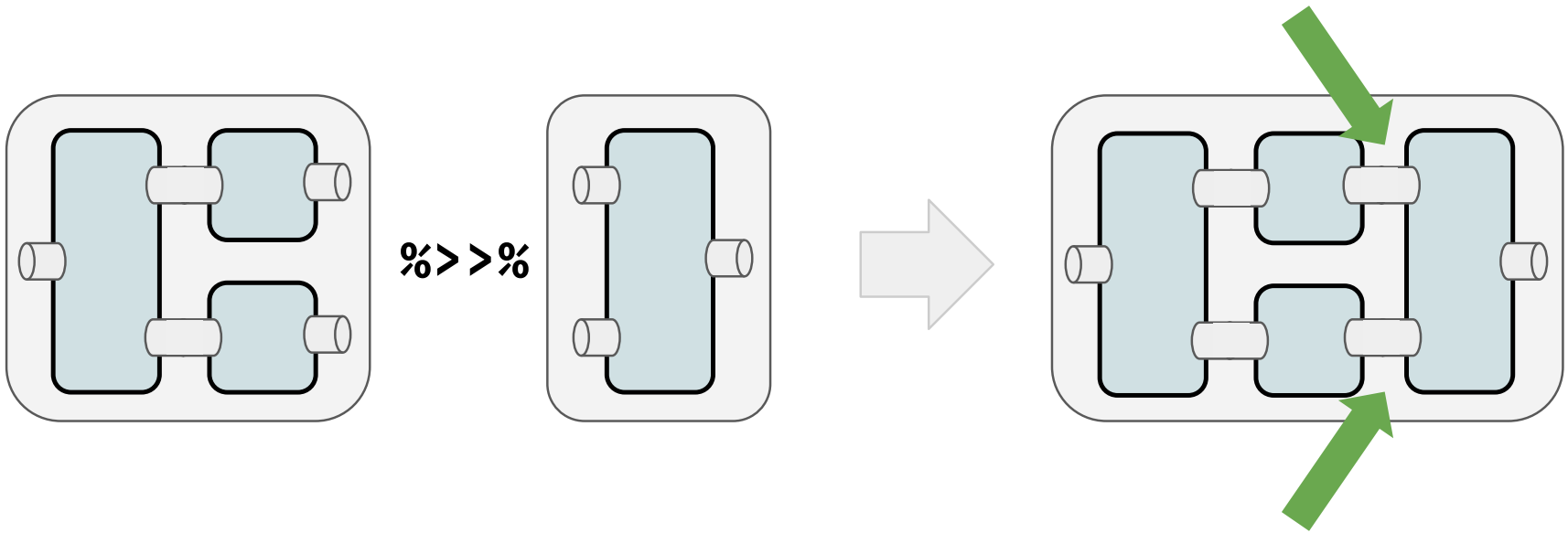
Planned

- Outlier detection
- Text processing
- Time series and spatio-temporal data
- Multi-output and ordinal targets

Graph Operations

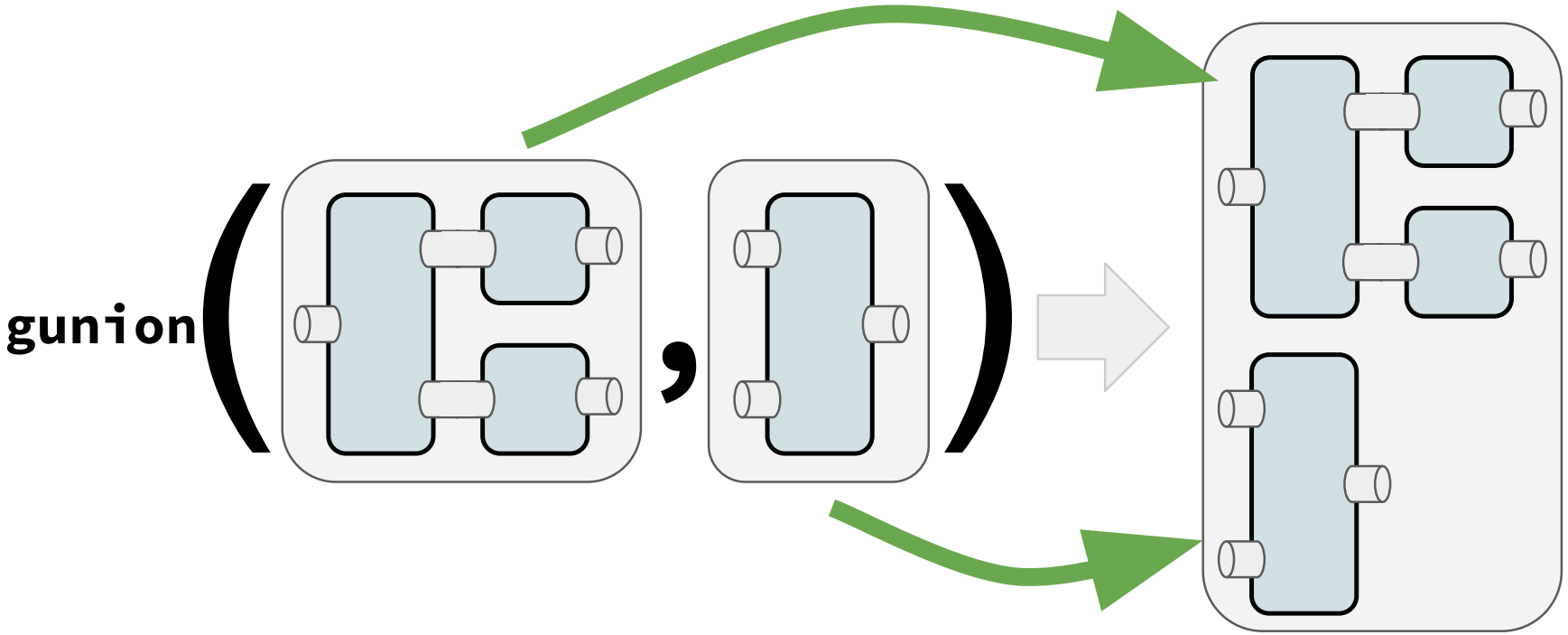
GRAPH OPERATIONS

%>>% concatenates `Graphs` and `PipeOps`



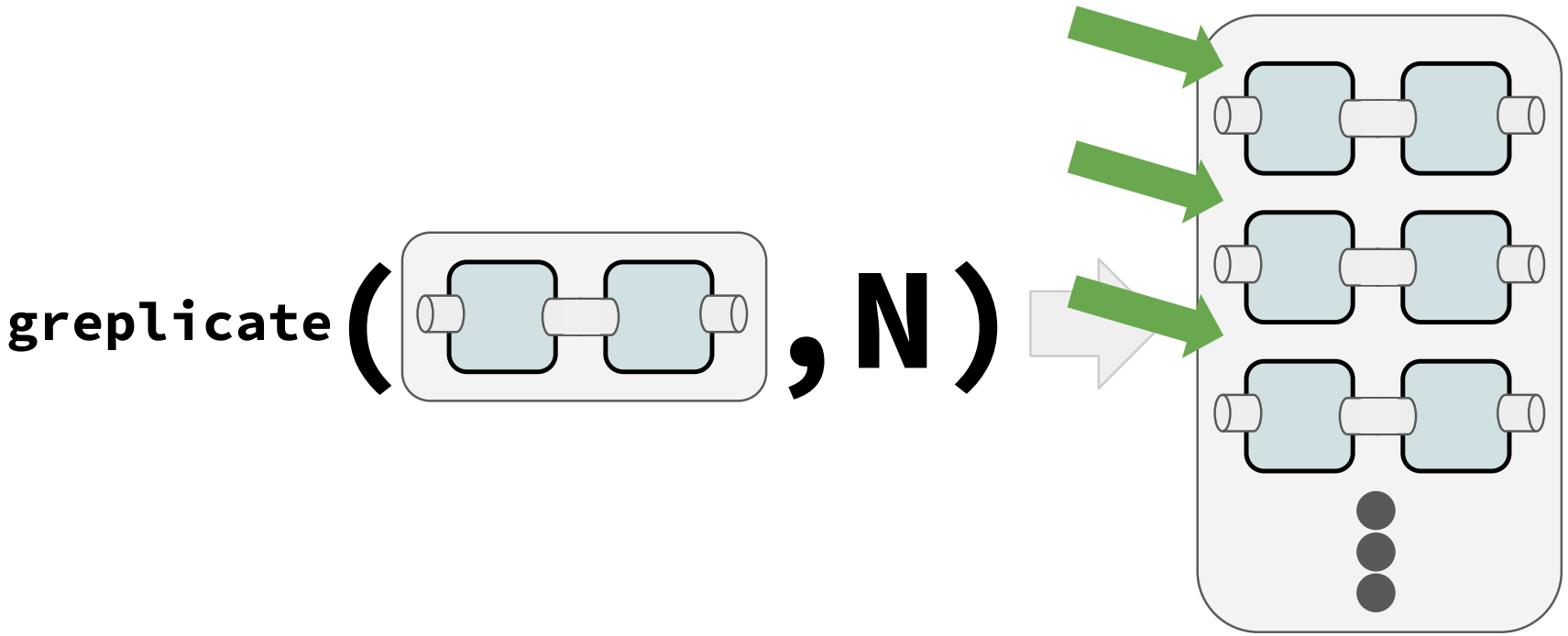
GRAPH OPERATIONS

`gunion()` unites `Graphs` and `PipeOps`



GRAPH OPERATIONS

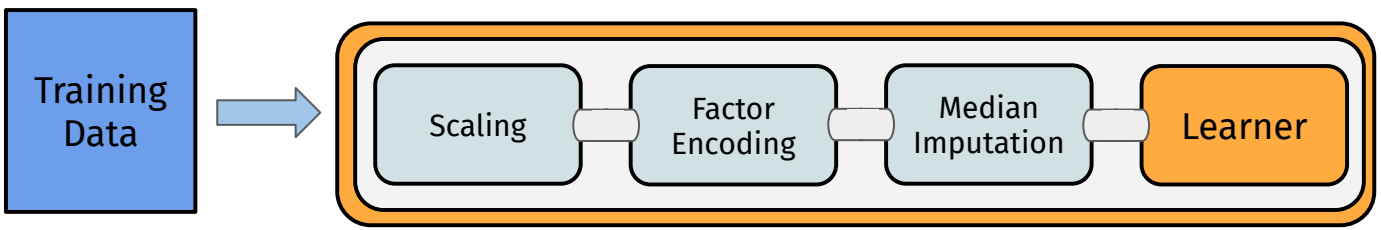
`greplicate()` unites copies of `Graphs` and `PipeOps`



Linear Pipelines

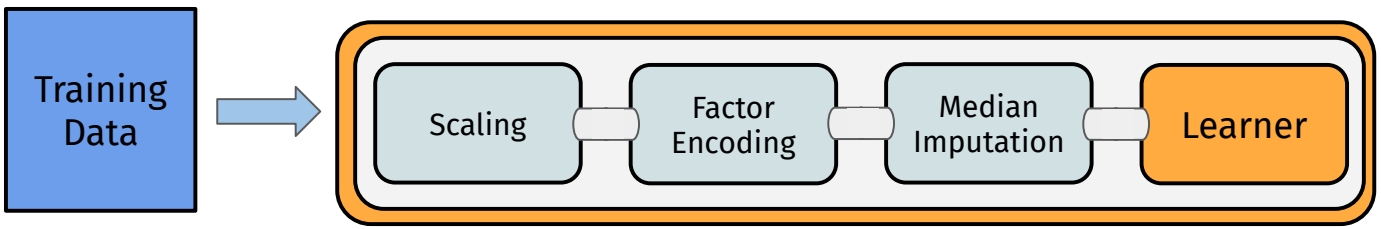
LINEAR PREPROCESSING

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



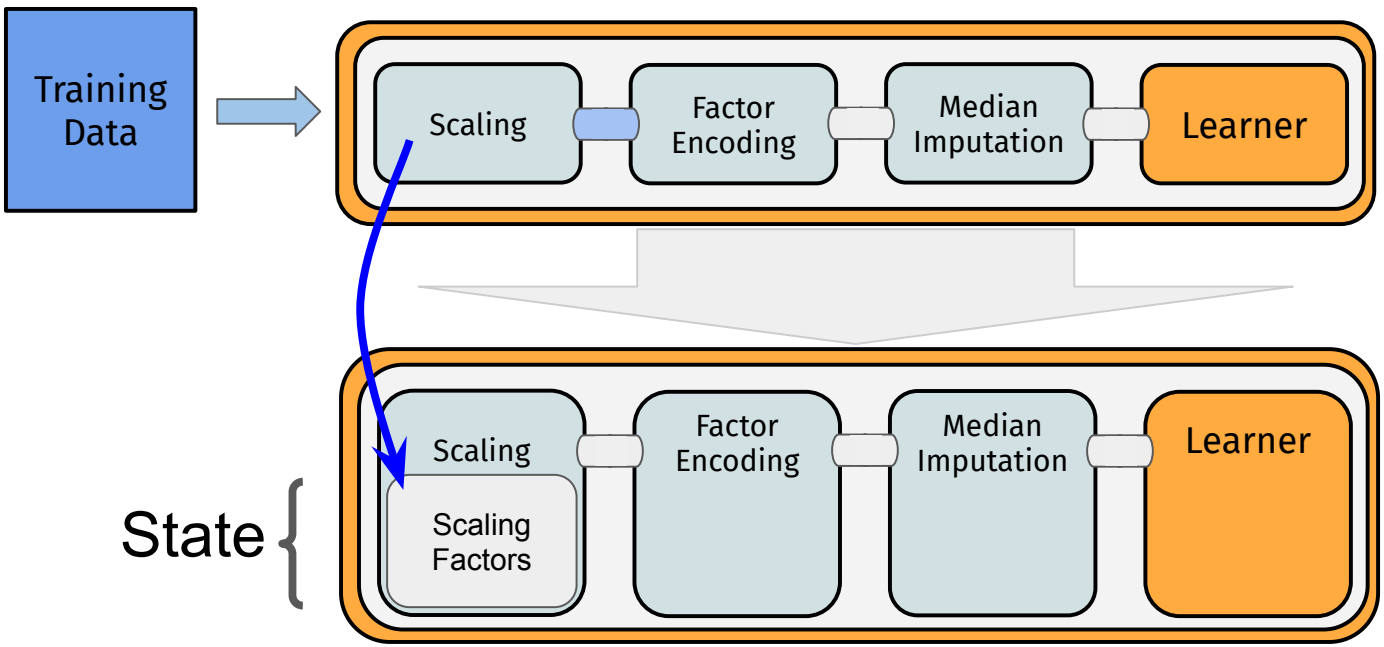
LINEAR PREPROCESSING

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



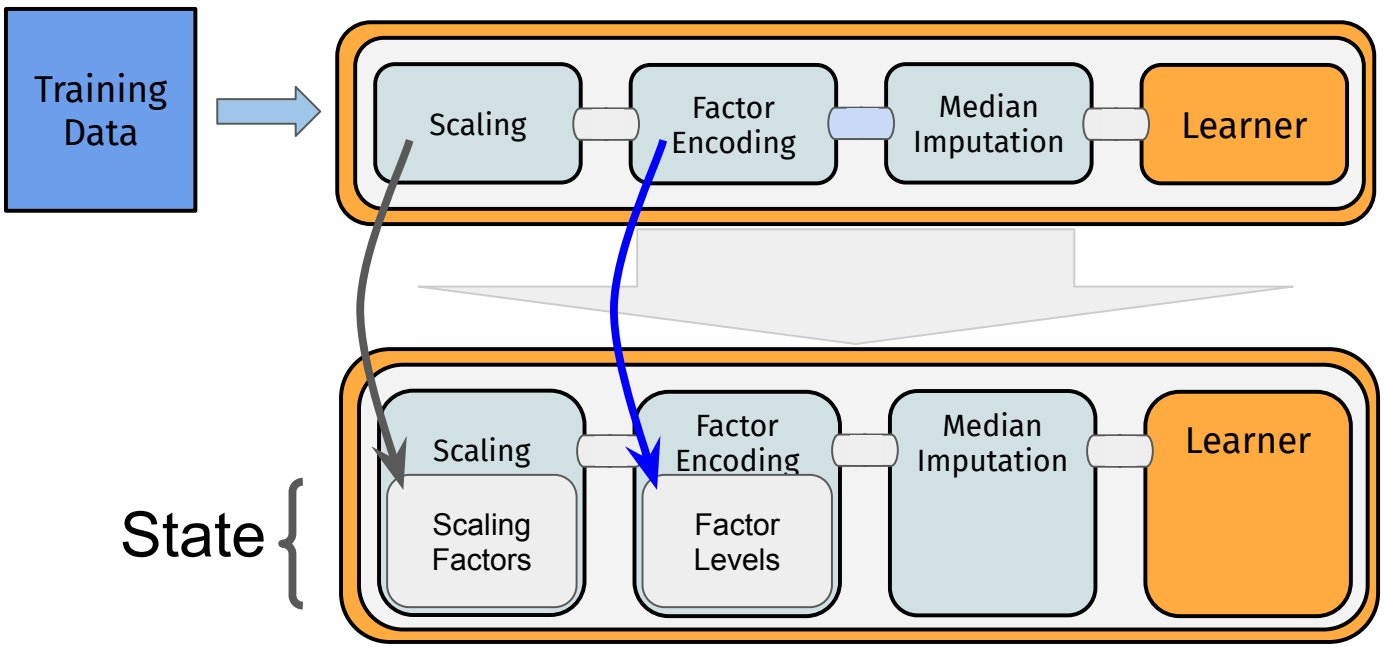
LINEAR PREPROCESSING

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



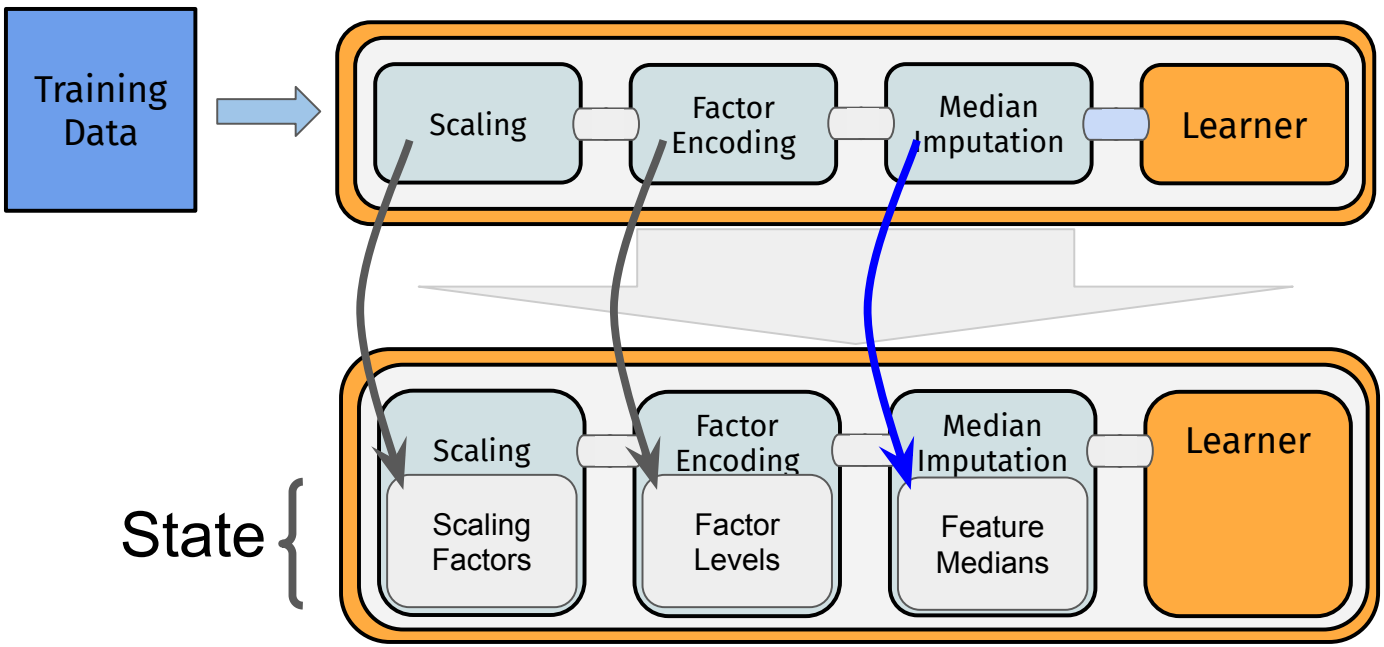
LINEAR PREPROCESSING

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



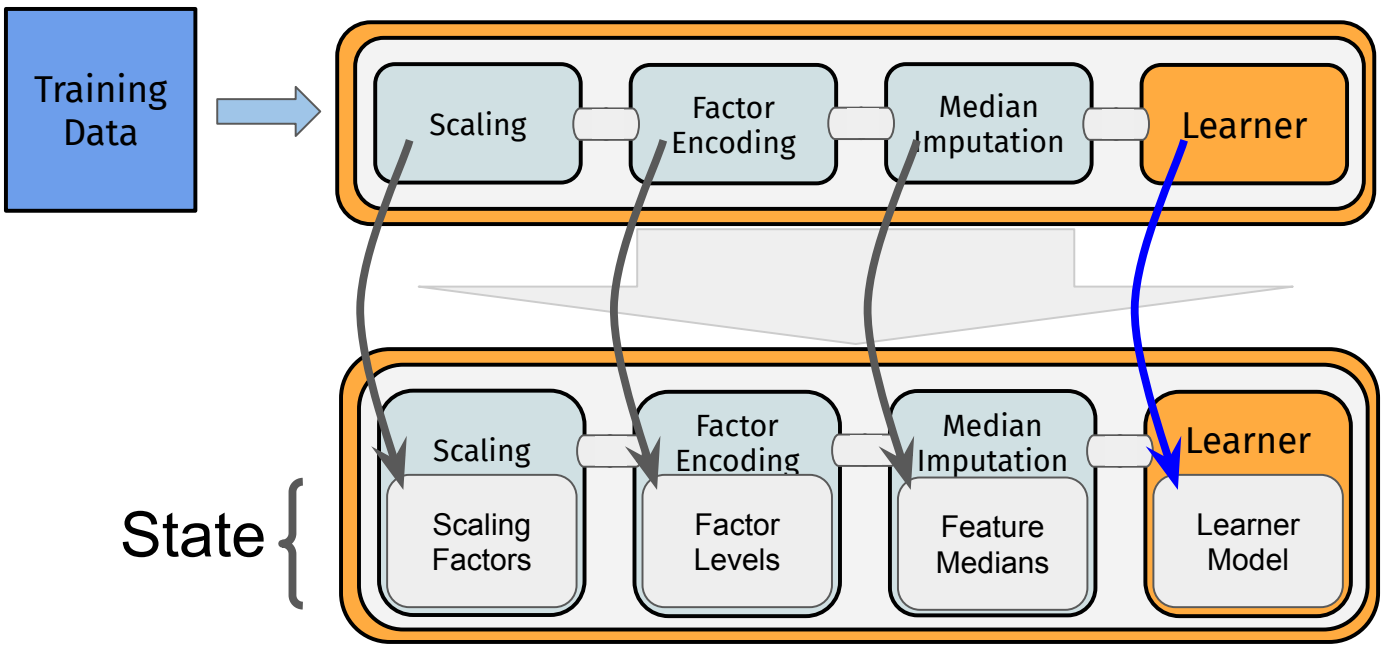
LINEAR PREPROCESSING

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



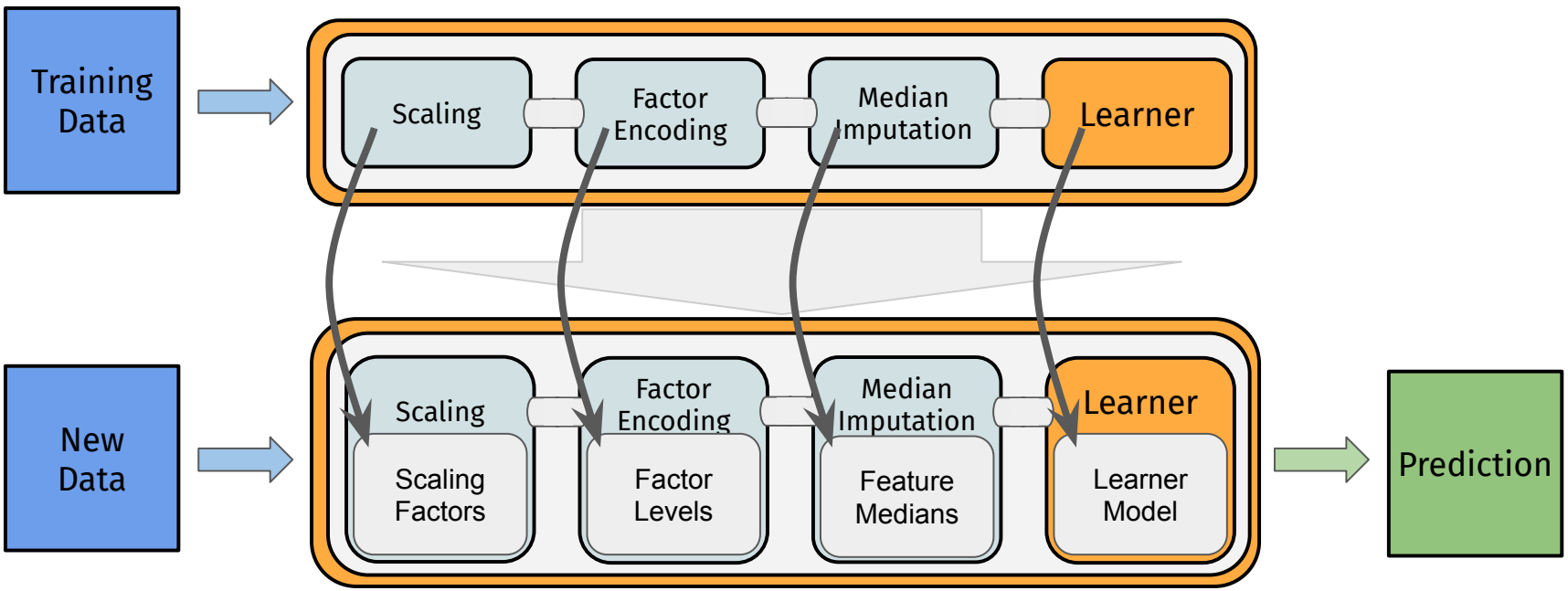
LINEAR PREPROCESSING

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



LINEAR PREPROCESSING

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



LINEAR PREPROCESSING

`scale %>>% encode %>>% impute %>>% rpart`

- Setting / retrieving parameters: `$param_set`

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

LINEAR PREPROCESSING

`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
#>        4.163        1.424        5.921        3.098
```

LINEAR PREPROCESSING

`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()`)

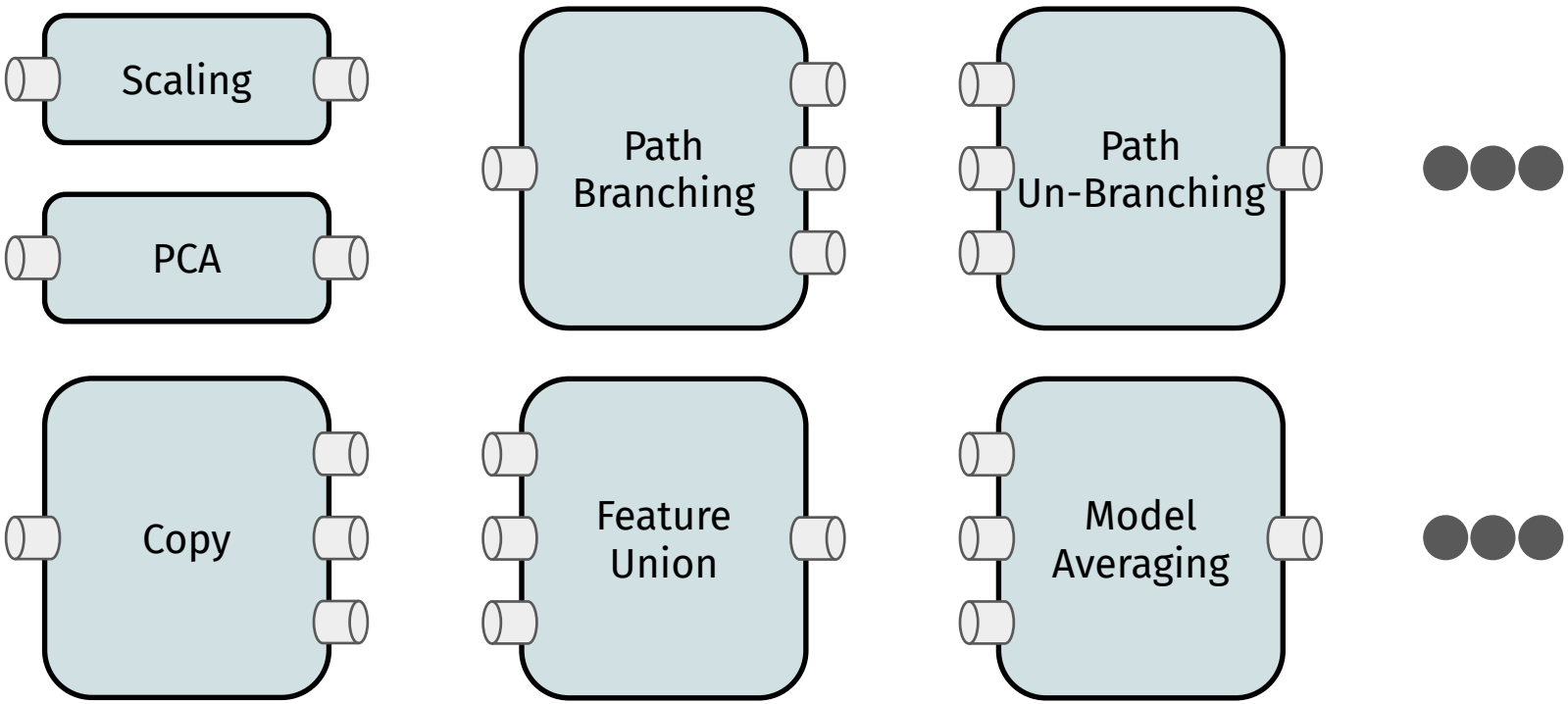
```
graph_pp$pipeops$scale$state$scale
#> Petal.Length  Petal.Width Sepal.Length  Sepal.Width
#>        4.163        1.424        5.921        3.098
```

- 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
#> 1:  setosa       0.3363      0.1404       0.8613      1.1296
#> 2:  setosa       0.3363      0.1404       0.8275      0.9682
#> 3:  setosa       0.3122      0.1404       0.7938      1.0328
```

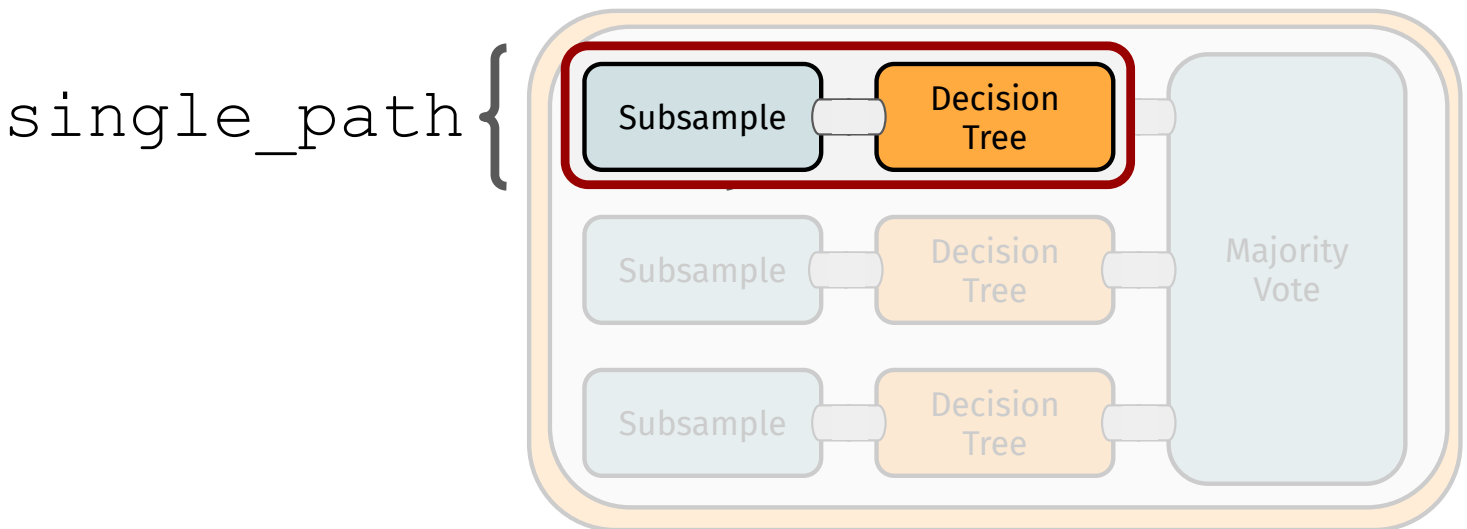
Nonlinear Pipelines

PIPEOPS WITH MULTIPLE INPUTS / OUTPUTS



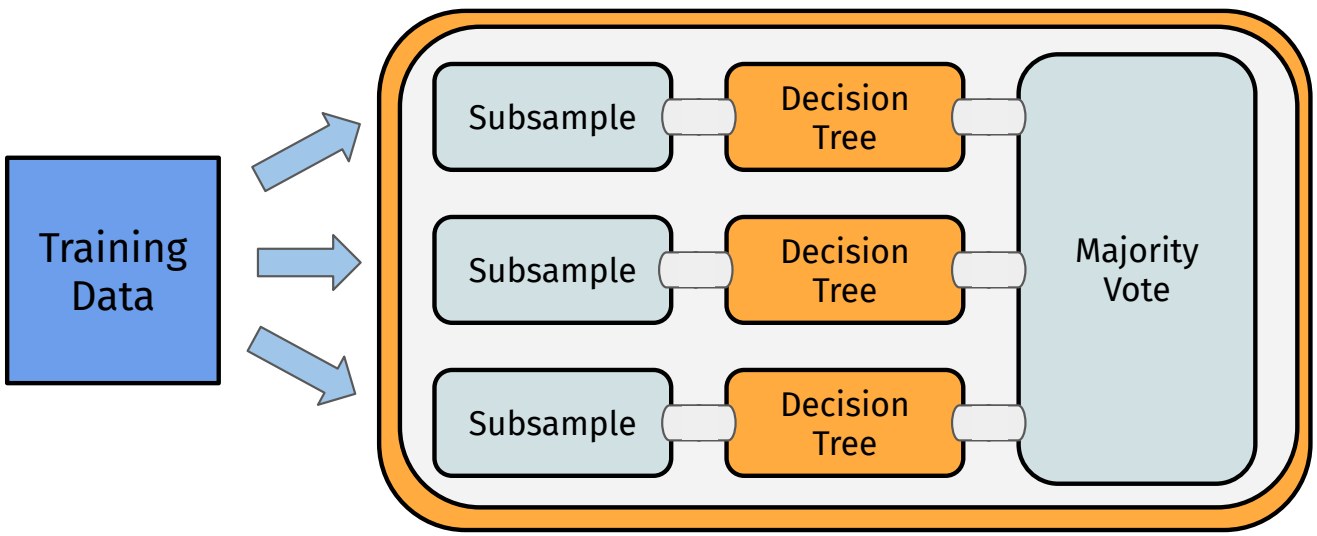
ENSEMBLE METHOD: BAGGING

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



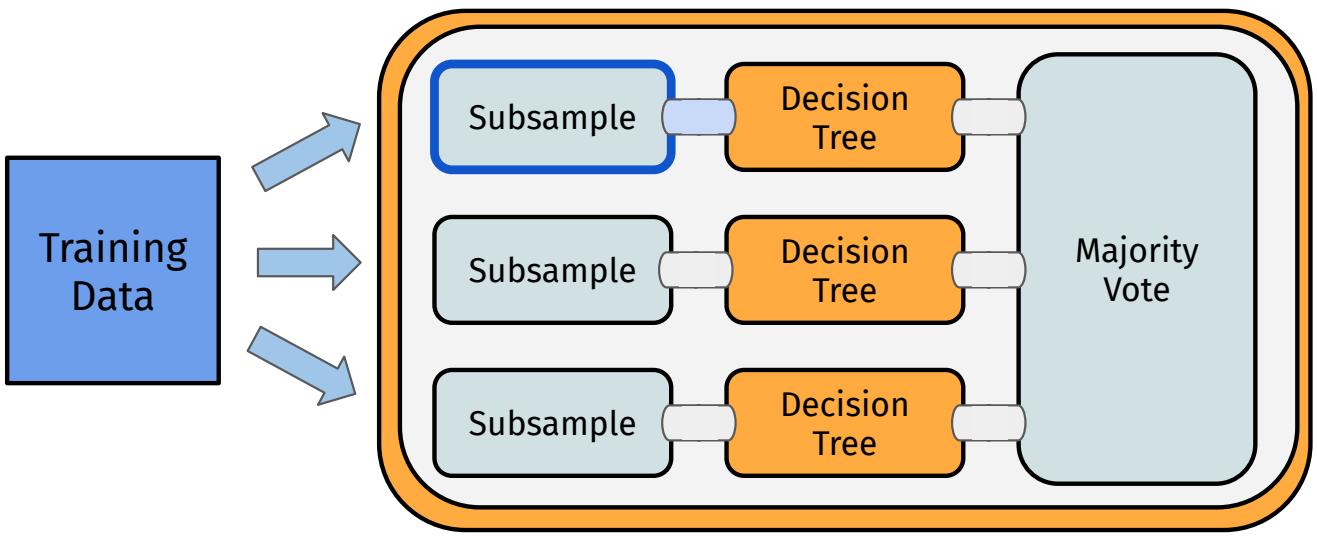
ENSEMBLE METHOD: BAGGING

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

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



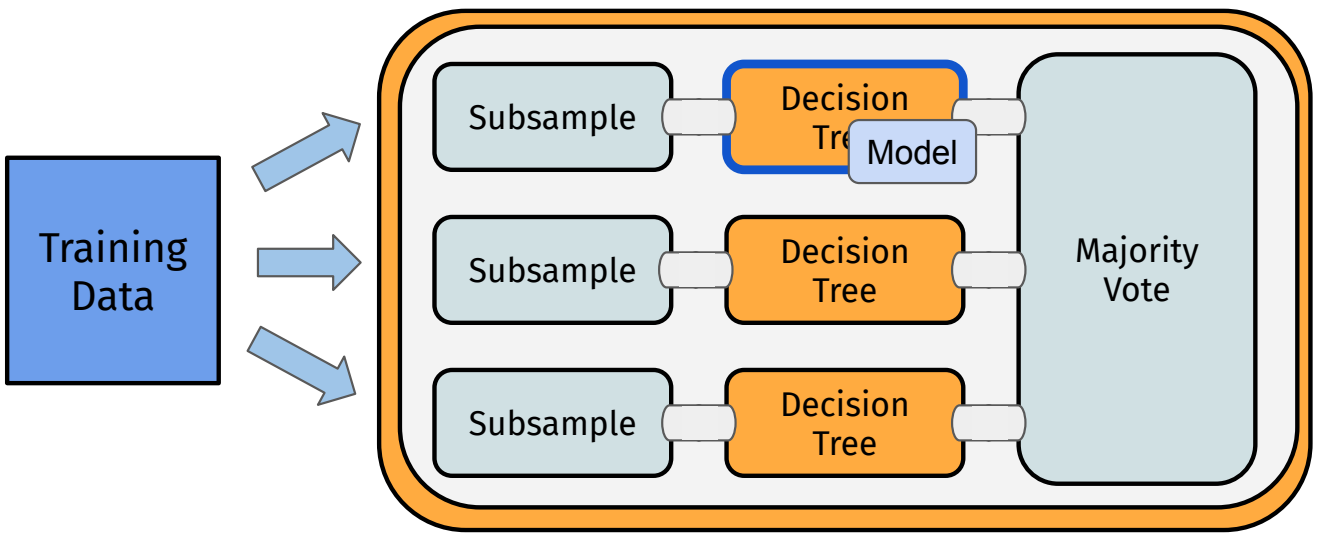
ENSEMBLE METHOD: BAGGING

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

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



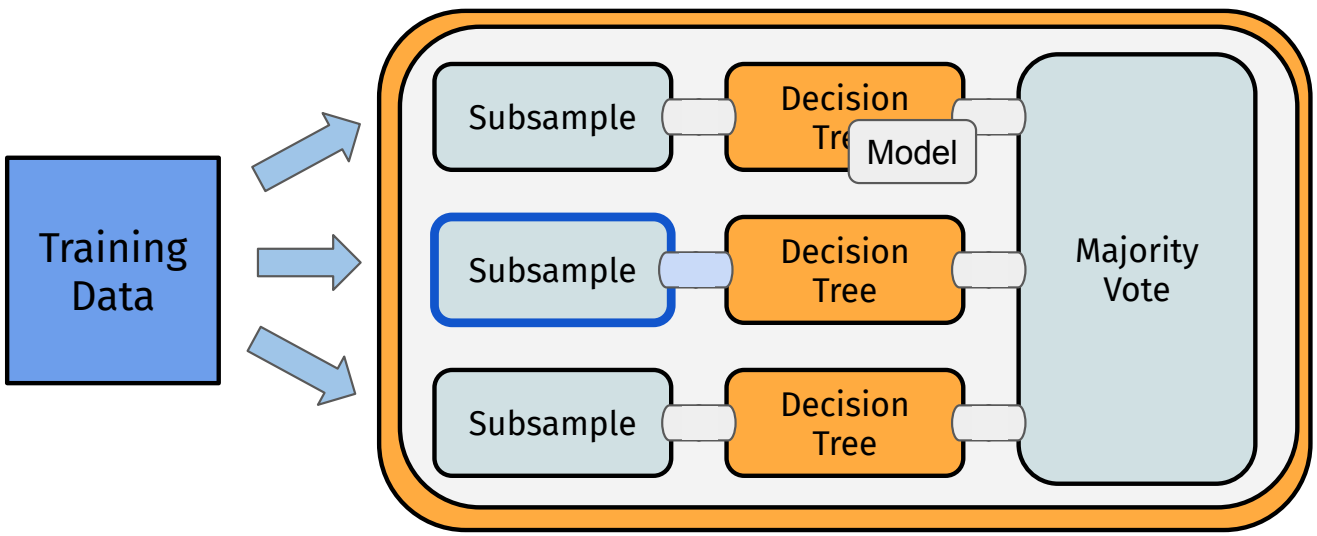
ENSEMBLE METHOD: BAGGING

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

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



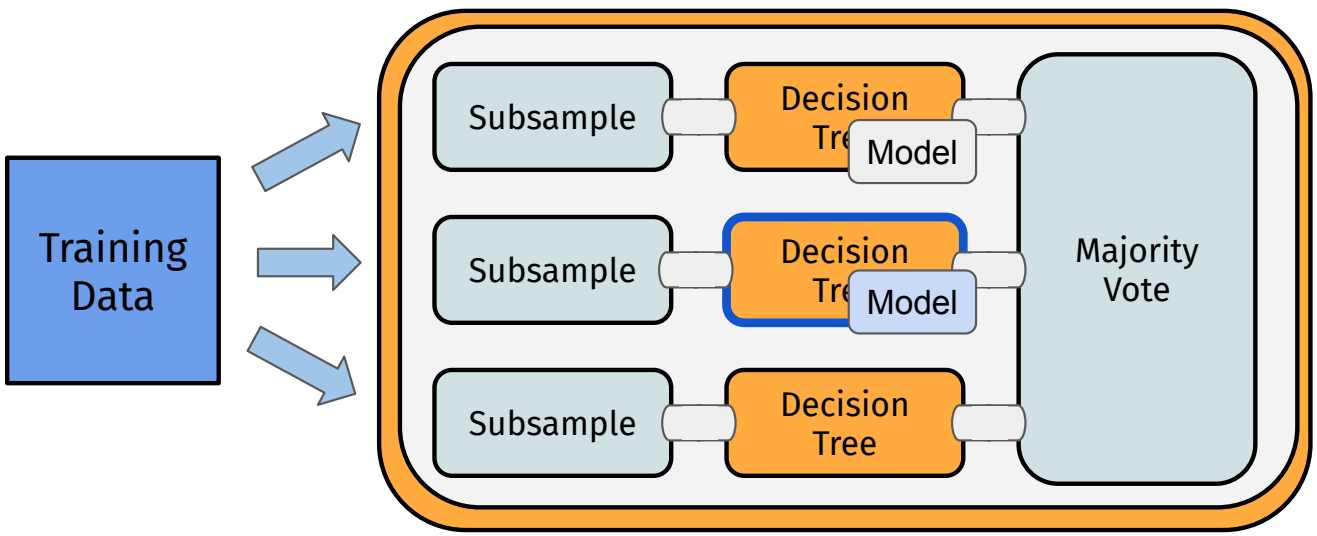
ENSEMBLE METHOD: BAGGING

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

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



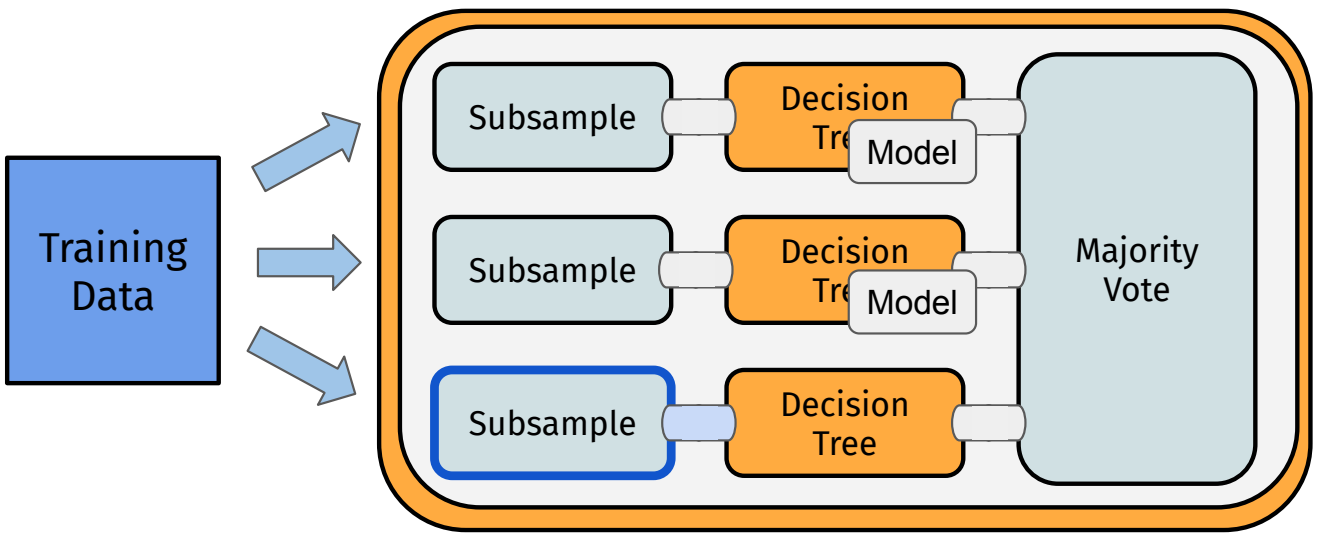
ENSEMBLE METHOD: BAGGING

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

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



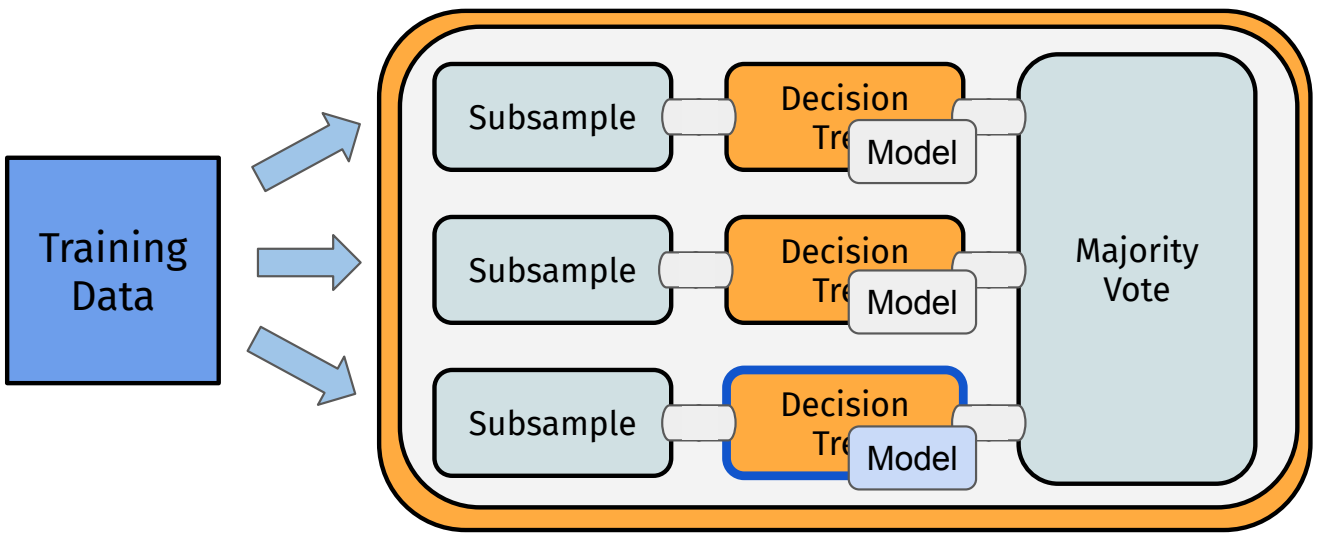
ENSEMBLE METHOD: BAGGING

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

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



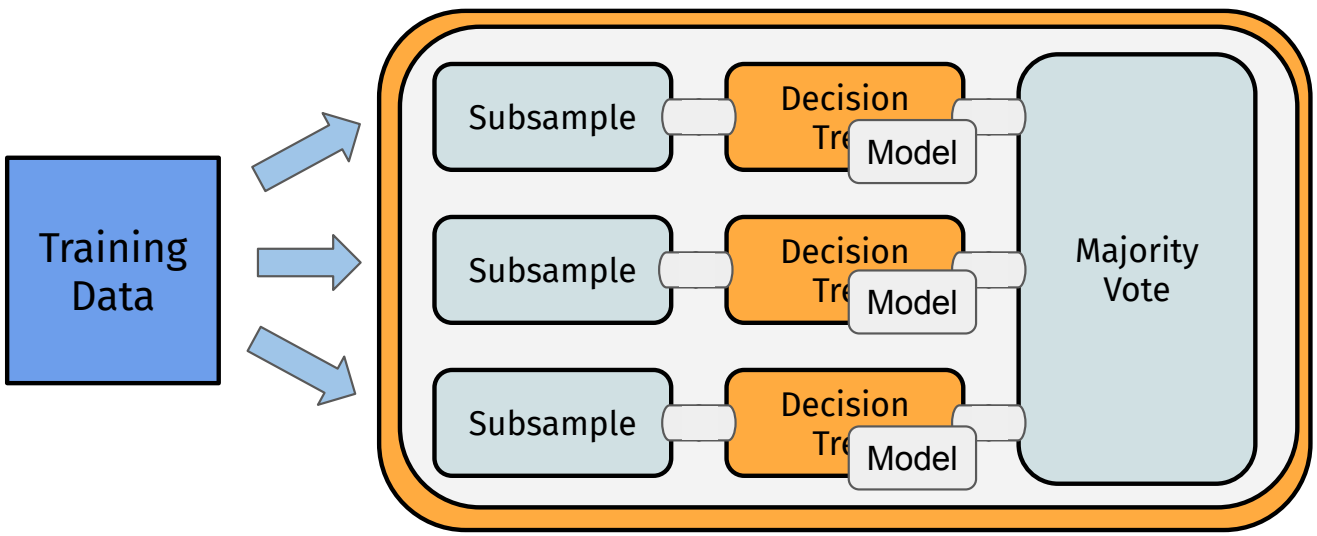
ENSEMBLE METHOD: BAGGING

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

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



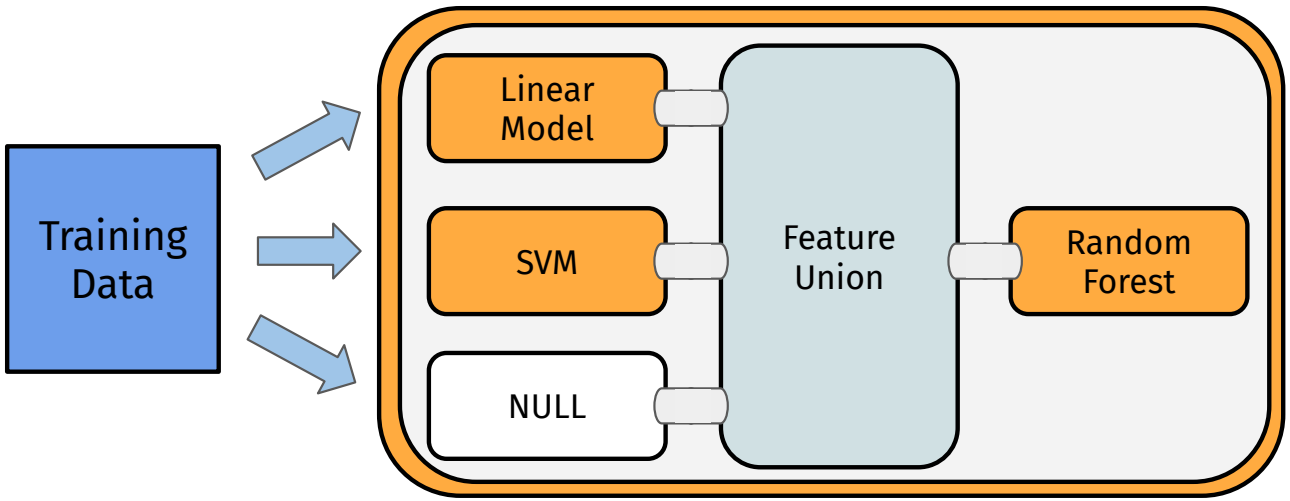
ENSEMBLE METHOD: BAGGING

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

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



ENSEMBLE METHOD: STACKING

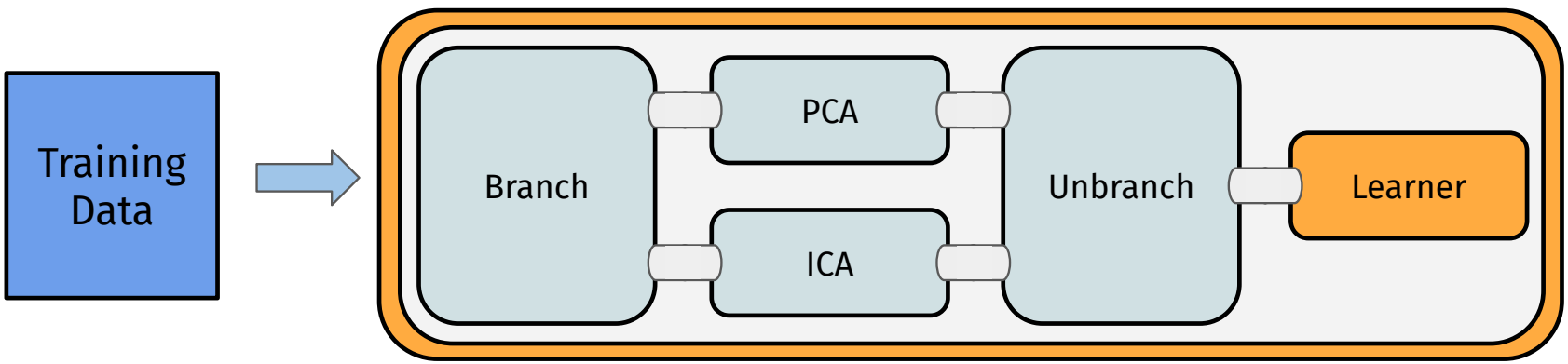
```
graph_stack = gunion(list(
      po("learner_cv", learner = lrn("regr.lm")),
      po("learner_cv", learner = lrn("regr.svm")),
      po("nop"))) %>>%
  po("featureunion") %>>%
  lrn("regr.ranger")
```



BRANCHING

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

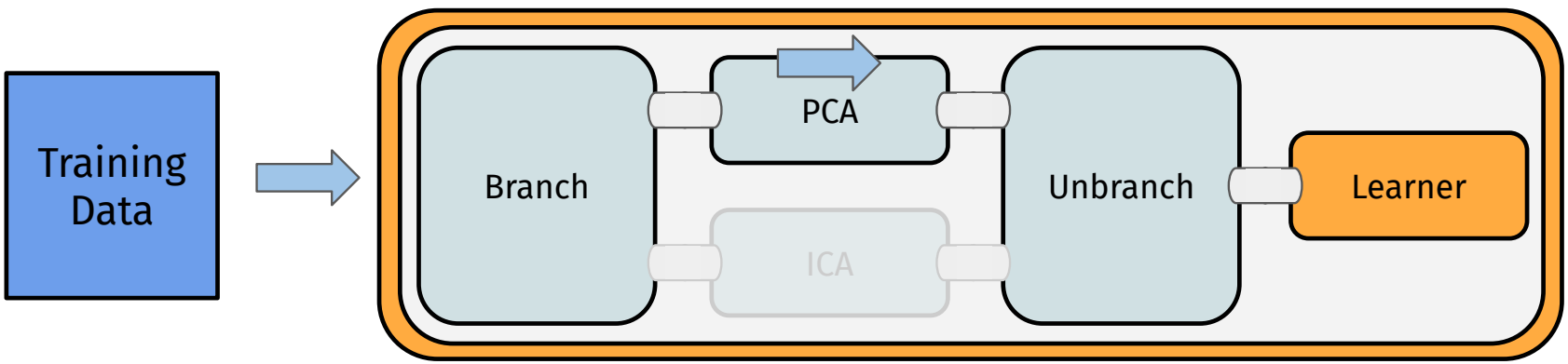
Execute only one of several alternative paths



BRANCHING

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

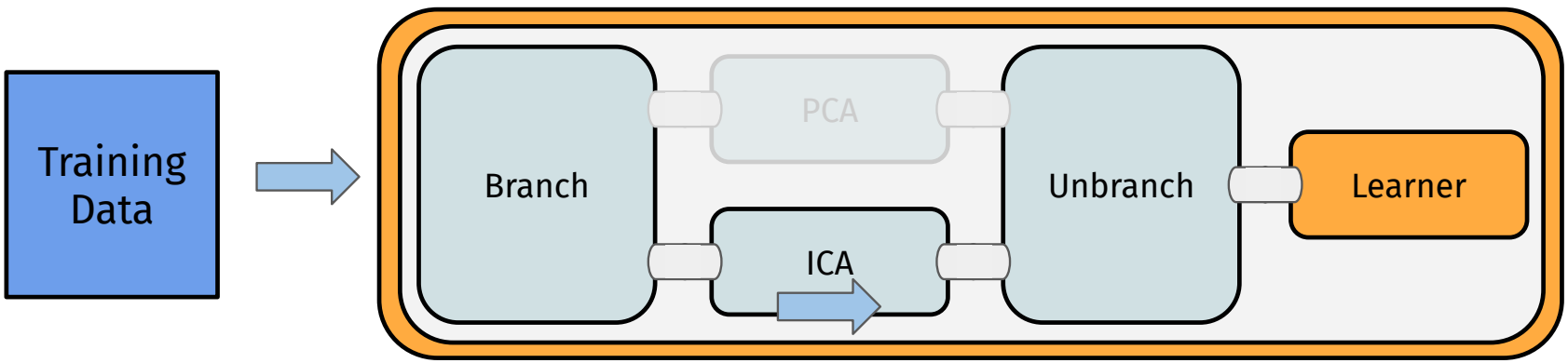
```
> graph_branch$pipeops$branch$
    param_set$values$selection = "pca"
```



BRANCHING

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

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



Targeting Columns

RESTRICT PIPEOPS TO COLS WITH 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
#> 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
```

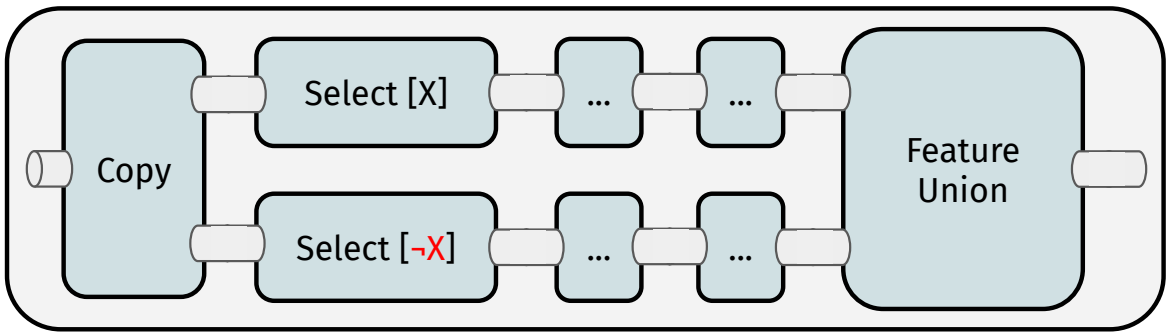
RESTRICT PIPEOPS TO COLS WITH SELECTORS

Option 1: `PipeOps` `affect_columns` parameter

```
my_pca = po("pca", affect_columns = selector_grep("^Sepal"))

result = my_pca$train(list(task))

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

#>    Species     PC1      PC2 Petal.Length Petal.Width
#> 1:  setosa -0.7781  0.37813          1.4         0.2
#> 2:  setosa -0.9351 -0.13701          1.4         0.2
#> 3:  setosa -1.1513  0.04534          1.3         0.2
```

RESTRICT PIPEOPS TO COLS WITH SELECTORS

Option 2: Use `po("select")`

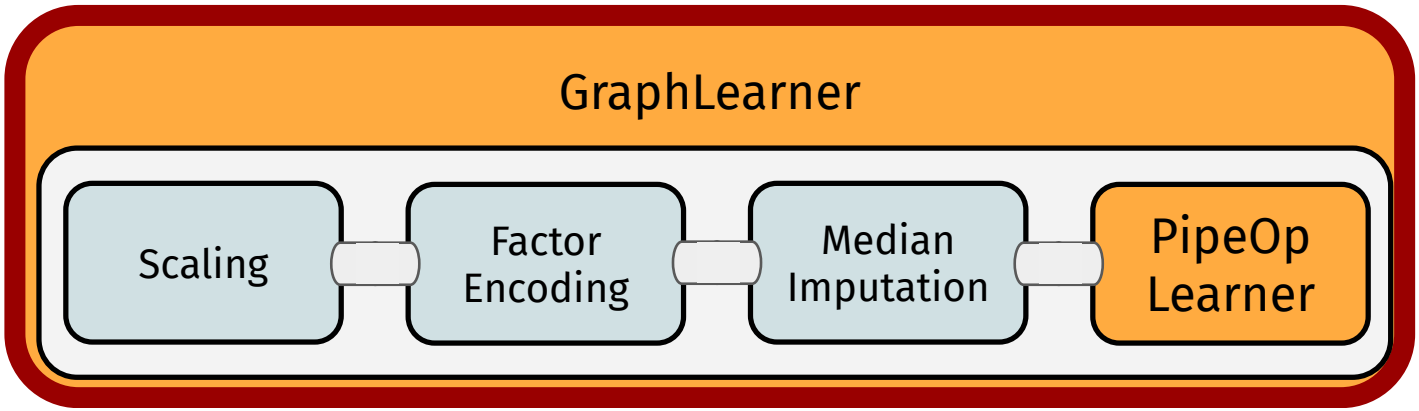



```
sel1 = selector_grep("^Sepal")
sel2 = selector_invert(sel1)

my_pca = gunion(list(
  po("select", selector = sel1) %>>% po("pca"),
  po("select", selector = sel2, id = "select2")
)) %>>% po("featureunion")

my_pca$train(task)[[1]]
```

Connecting Pipelines to mlr3: Resampling, Hyperparameters, Tuning and Nested CV

GRAPHLEARNER

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




```
graph_pp = po("scale") %>>% po("encode") %>>%
  po("imputemedian") %>>% lrn("classif.rpart")
glrn = GraphLearner$new(graph_pp)
glrn$train(task)
glrn$predict(task)
resample(task, glrn, rsmp("cv", folds = 3))
```

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

```
library("paradox") ; library("mlr3tuning"); library("mlr3filters")
glrn = po("pca") %>>%  flt("anova") %>>% lrn("classif.svm")
ps = ParamSet$new(list(
  ParamDbl$new("anova.filter.frac", lower = 0.1, upper = 1),
  ParamDbl$new("classif.svm.cost", lower = -10, upper = -10),
  ParamDbl$new("classif.svm.gamma", lower = -10, upper = -10)
))
ps$trafo = function(x, param_set) {
  x$classif.svm.cost = 2^x$classif.svm.cost
  x$classif.svm.gamma = 2^x$classif.svm.gamma
  return(x)
}
inst = TuningInstance$new(tsk("sonar"), glrn, rsmp("cv"),
  msr("classif.ce"), ps, term("evals", n_evals = 10))
tnr("random_search")$tune(inst)
```

OUTLOOK

- Release mlr3book version 0.1
- mlr3 e-learning platform with videos and usecases
- Improve pipelines a bit
- Maybe a bit more syntactic sugar
- Nearly done:`mlr3proba`, `mlr3hyperband`, `mlr3ordinal`
- I will work on: `mlr3mbo` for Bayesian optimization

Thanks! Please ask questions!