



Machine Learning Pipelines in R

mlr3pipelines

Department of Statistics – LMU Munich



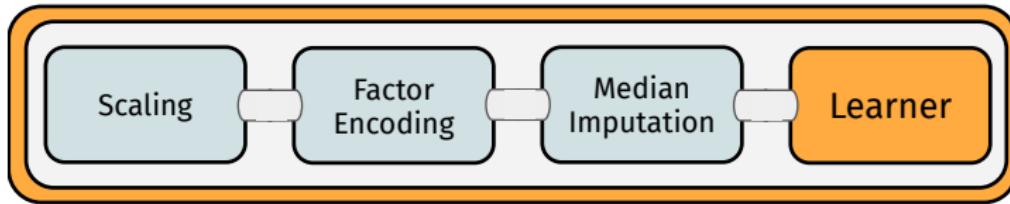
Intro

MLR3PIPELINES

Machine Learning Workflows:

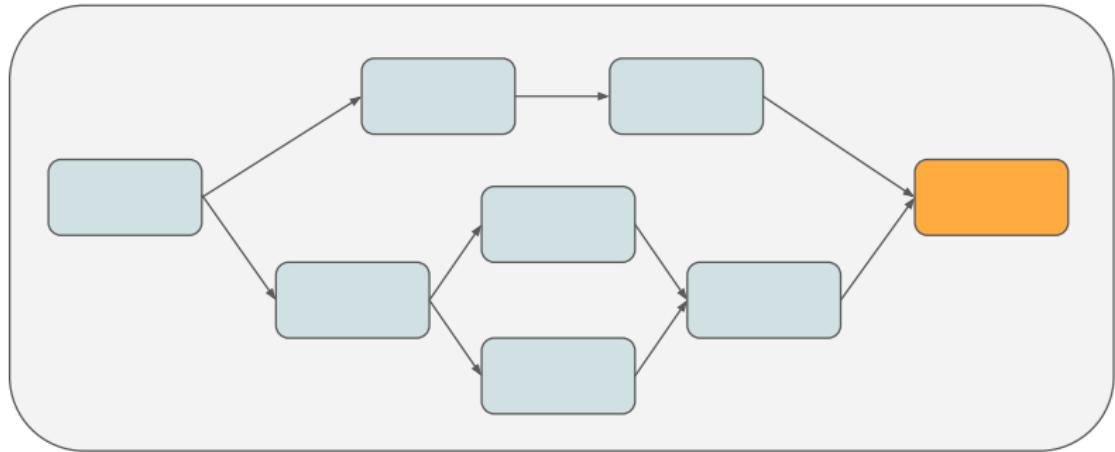
- **Preprocessing:** Feature extraction, feature selection, missing data imputation,...
- **Ensemble methods:** Model averaging, model stacking
- **mlr3:** modular model fitting

⇒ **mlr3pipelines:** modular ML workflows



MACHINE LEARNING WORKFLOWS

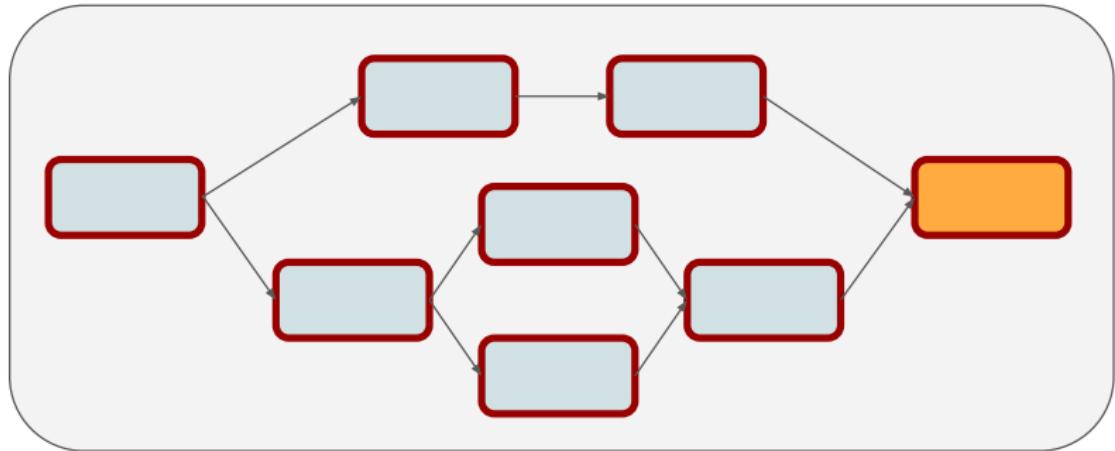
– what do they look like?



MACHINE LEARNING WORKFLOWS

– what do they look like?

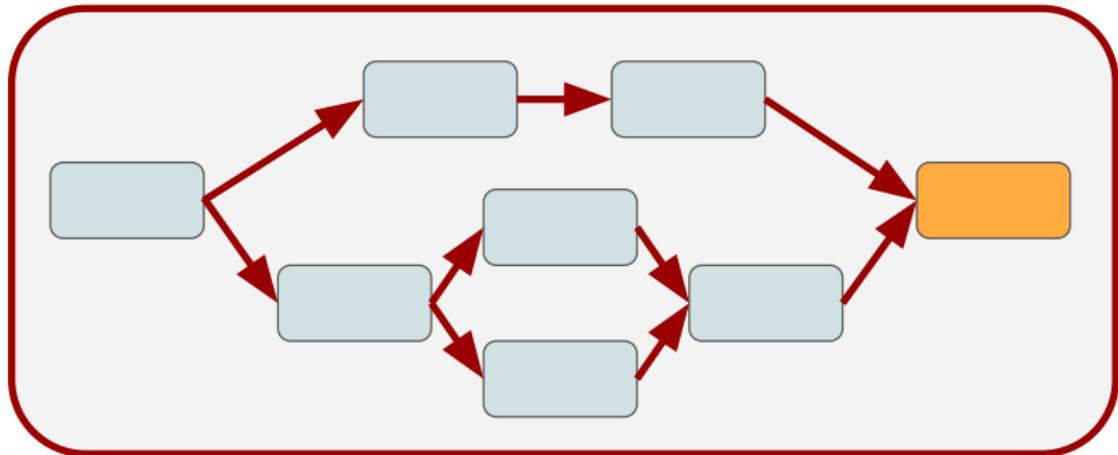
- **Building blocks:** *what is happening? → PipeOp*



MACHINE LEARNING WORKFLOWS

– what do they look like?

- **Building blocks:** *what is happening?* → PipeOp
- **Structure:** *in what sequence is it happening?* → Graph



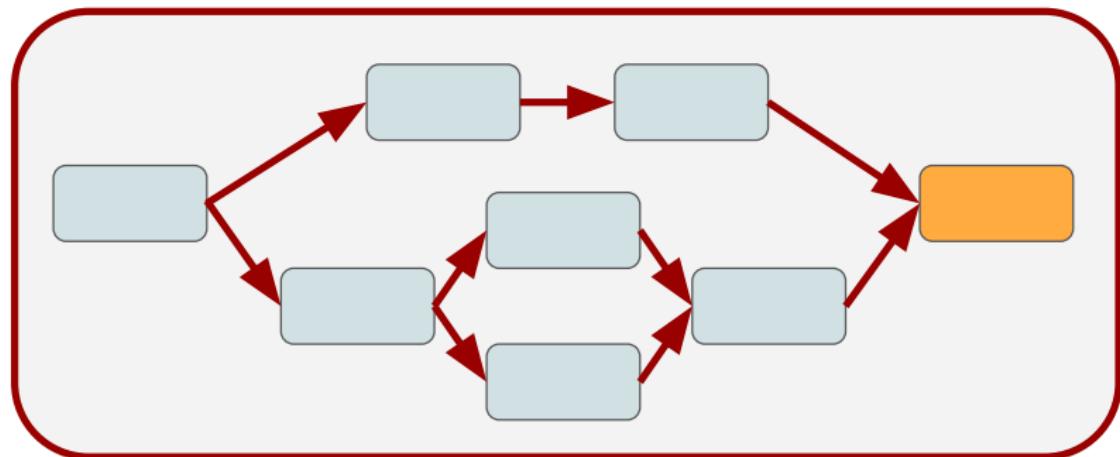
MACHINE LEARNING WORKFLOWS

– what do they look like?

- **Building blocks:** *what is happening?* → PipeOp

- **Structure:** *in what sequence is it happening?* → Graph

⇒ Graph: PipeOps as **nodes** with **edges** (data flow) between them

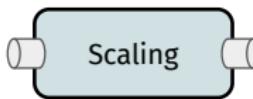


PipeOps

THE BUILDING BLOCKS

PipeOp: Single Unit of Data Operation

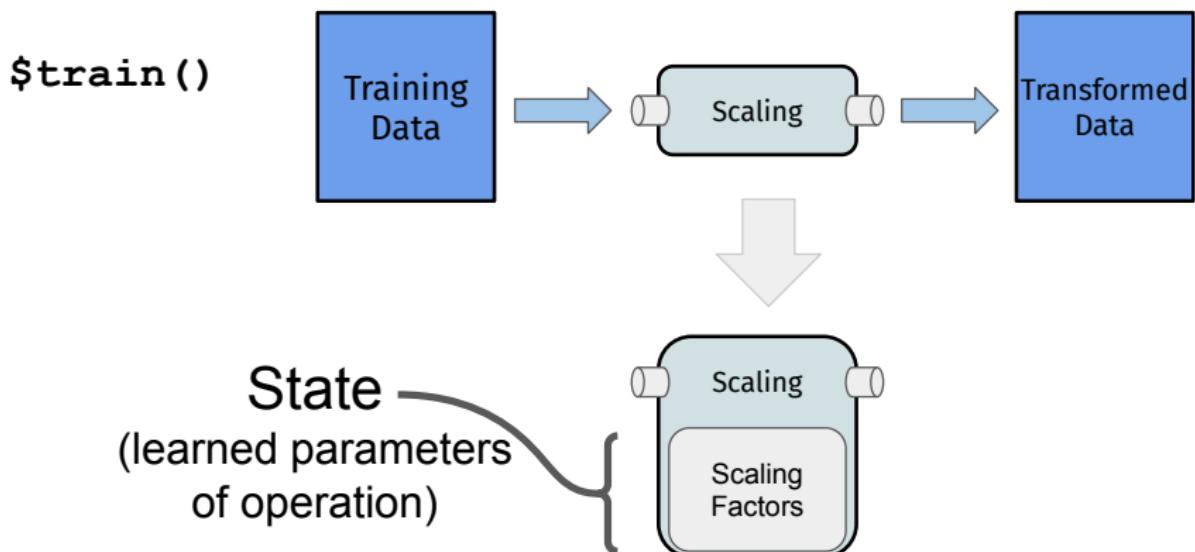
- `pip = po("scale")` to construct



THE BUILDING BLOCKS

PipeOp: Single Unit of Data Operation

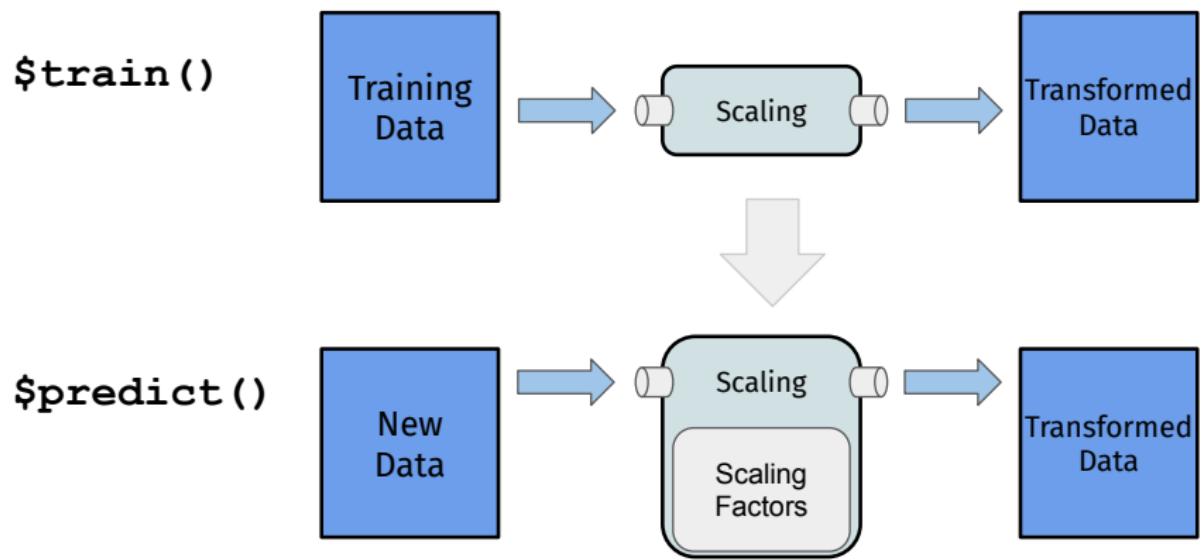
- `pip = po("scale")` to construct
- `pip$train()`: process data and create `pip$state`



THE BUILDING BLOCKS

PipeOp: Single Unit of Data Operation

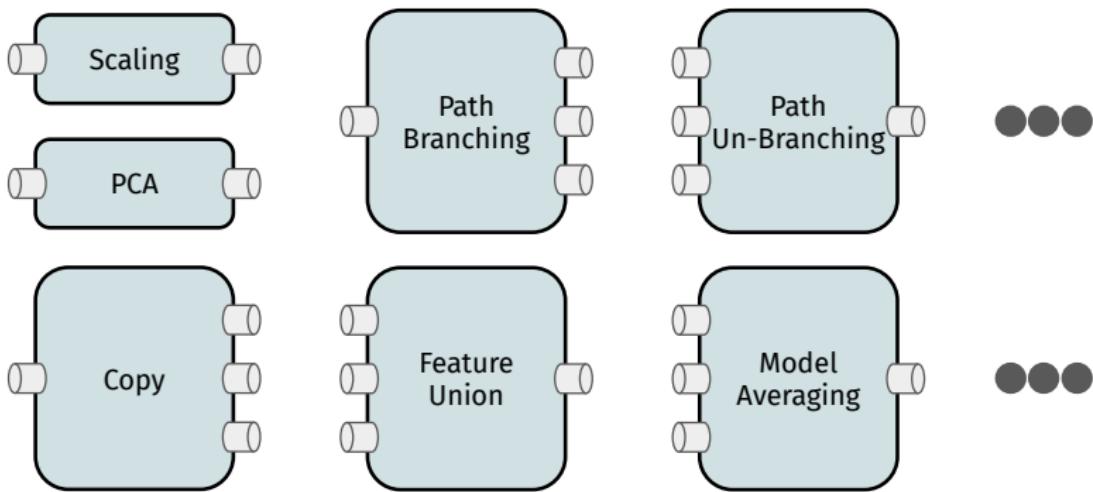
- `pip = po("scale")` to construct
- `pip$train()`: process data and create `pip$state`
- `pip$predict()`: process data depending on the `pip$state`



THE BUILDING BLOCKS

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`
- Multiple inputs or multiple outputs



THE BUILDING BLOCKS

```
po = po("scale")

trained = po$train(list(task))

trained[[1]]$head(3)

#>   Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#> 1: setosa    -1.335752   -1.311052   -0.8976739   1.0156020
#> 2: setosa    -1.335752   -1.311052   -1.1392005  -0.1315388
#> 3: setosa    -1.392399   -1.311052   -1.3807271   0.3273175
```

THE BUILDING BLOCKS

```
po = po("scale")

trained = po$train(list(task))

trained[[1]]$head(3)

#>   Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#> 1: setosa    -1.335752   -1.311052   -0.8976739   1.0156020
#> 2: setosa    -1.335752   -1.311052   -1.1392005  -0.1315388
#> 3: setosa    -1.392399   -1.311052   -1.3807271   0.3273175
```

```
head(po$state, 2)

#> $center
#> Petal.Length  Petal.Width Sepal.Length  Sepal.Width
#>      3.758000     1.199333     5.843333     3.057333
#>
#> $scale
#> Petal.Length  Petal.Width Sepal.Length  Sepal.Width
#>      1.7652982    0.7622377     0.8280661     0.4358663
```

THE BUILDING BLOCKS

```
po = po("scale")

trained = po$train(list(task))

trained[[1]]$head(3)

#>   Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#> 1: setosa    -1.335752   -1.311052   -0.8976739   1.0156020
#> 2: setosa    -1.335752   -1.311052   -1.1392005  -0.1315388
#> 3: setosa    -1.392399   -1.311052   -1.3807271   0.3273175
```

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

#>   Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#> 1: setosa    -1.335752   -1.311052   -0.8976739   1.0156020
#> 2: setosa    -1.335752   -1.311052   -1.1392005  -0.1315388
#> 3: setosa    -1.392399   -1.311052   -1.3807271   0.3273175
```

PIPEOPS SO FAR

```
as.data.table(mlr_pipeops)
```

#>	key	packages	tags ...
#> 1	boxcox	bestNormalize	data transform
#> 2	branch		meta
#> 3	chunk		meta
#> 4	classbalancing		imbalanced data, data transform
#> 5	classifavg	stats	ensemble
#> 6	classweights		imbalanced data, data transform
#> 7	colapply		data transform
#> 8	collapsefactors		data transform
#> 9	copy		meta
#> 10	datefeatures		data transform
#> 11	encode	stats	encode, data transform
#> 12	encodeimpact		encode, data transform
#> 13	encodelmer	lme4, nloptr	encode, data transform
#> 14	featureunion		ensemble
#> 15	filter		feature selection, data transform
#> 16	fixfactors		robustify, data transform
#> 17	histbin	graphics	data transform
#> [...]			

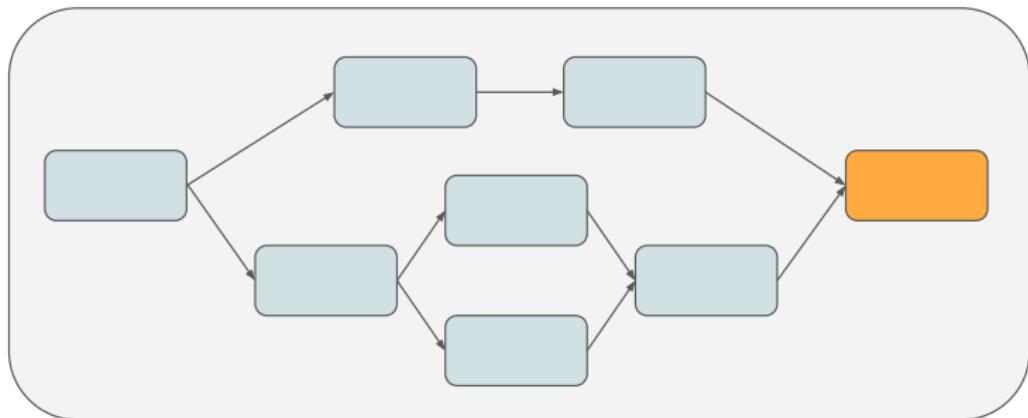
PIPEOPS SO FAR AND PLANNED

- 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)
- Ensemble methods on Predictions (weighted average, possibly learned weights)
- Branching (simultaneous branching, alternative branching)
- Combination of data: `featureunion`
- Text processing
- Date processing
- Time series and spatio-temporal data (*planned*)
- Multi-output and ordinal targets (*planned*)
- Outlier detection (*planned*)

Graph Operations

THE STRUCTURE

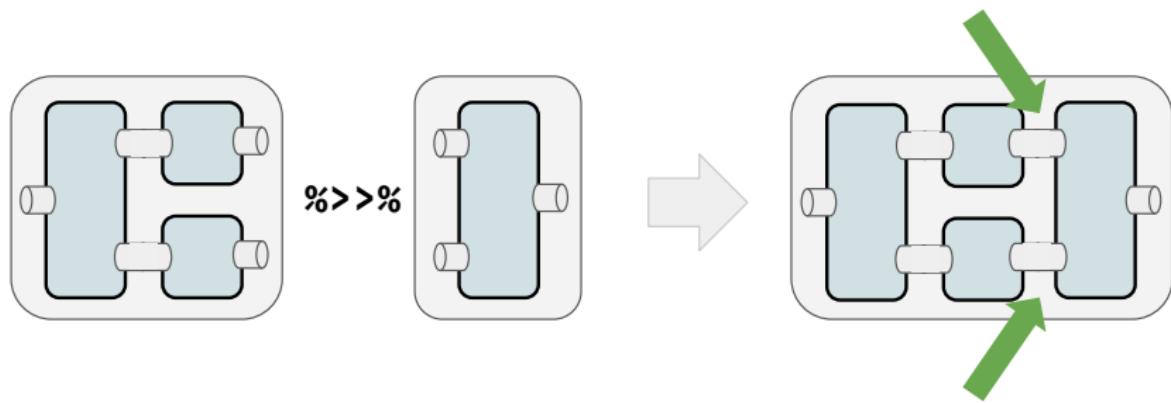
Graph Operations



THE STRUCTURE

Graph Operations

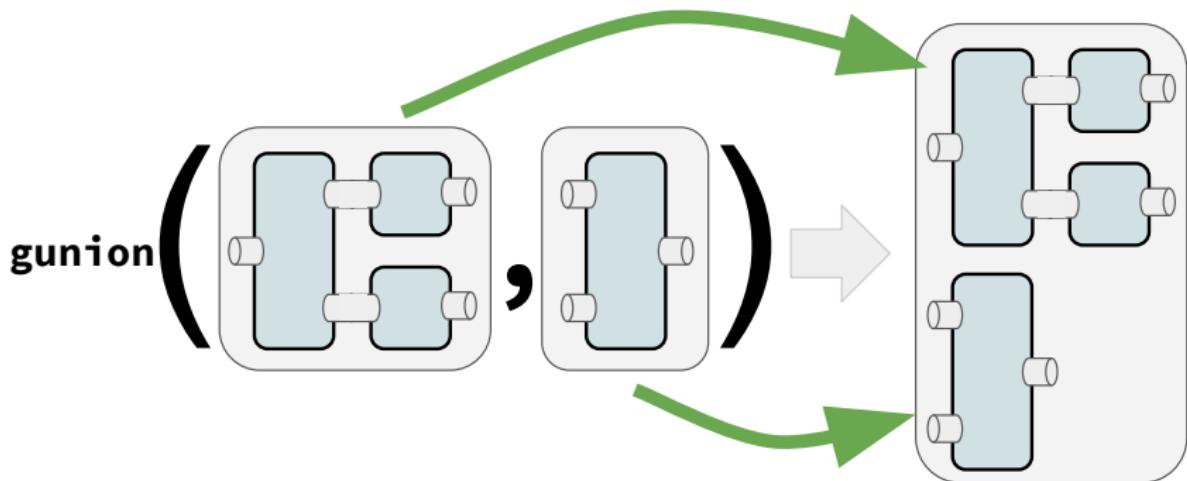
- The `%>>%`-operator concatenates Graphs and PipeOps



THE STRUCTURE

Graph Operations

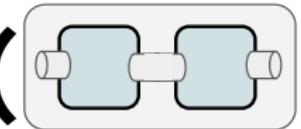
- The `%>>%`-operator concatenates Graphs and PipeOps
- The `gunion()`-function unites Graphs and PipeOps

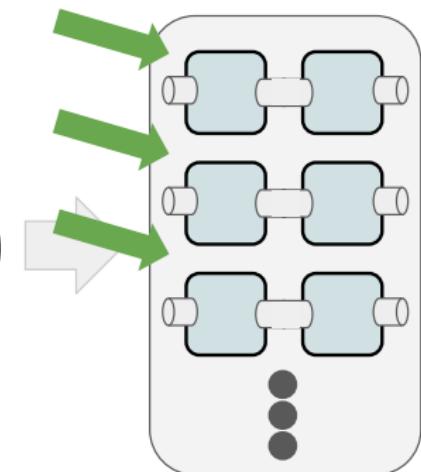


THE STRUCTURE

Graph Operations

- The `%>>%`-operator concatenates Graphs and PipeOps
- The `gunion()`-function unites Graphs and PipeOps
- The `pipeline_greplicate()`-function unites copies of Graphs and PipeOps

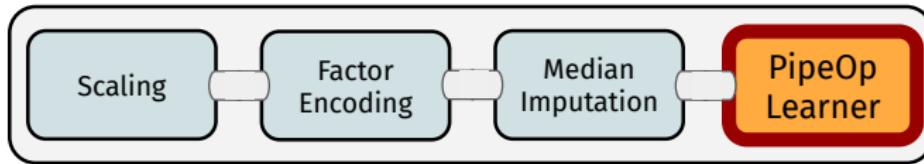
`pipeline_
greplicate (`  `, N)`



LEARNERS AND GRAPHS

PipeOpLearner

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



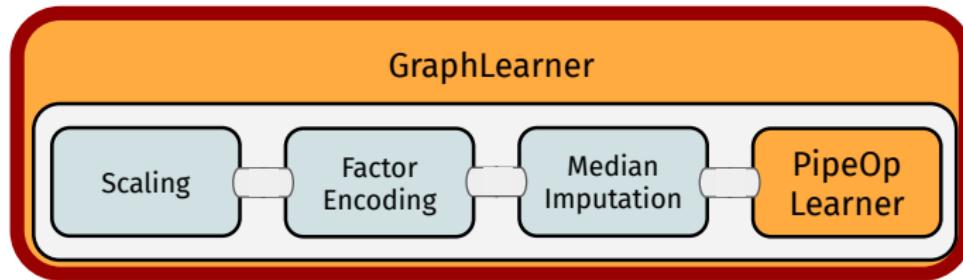
LEARNERS AND GRAPHS

PipeOpLearner

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

GraphLearner

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

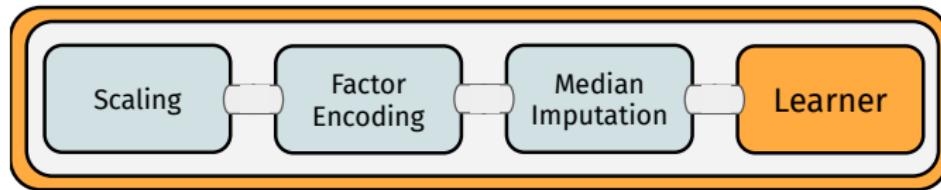


Linear Pipelines

MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

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

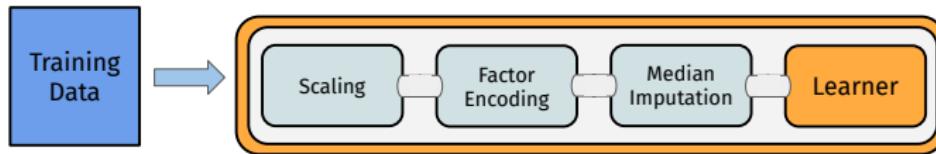


MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

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

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

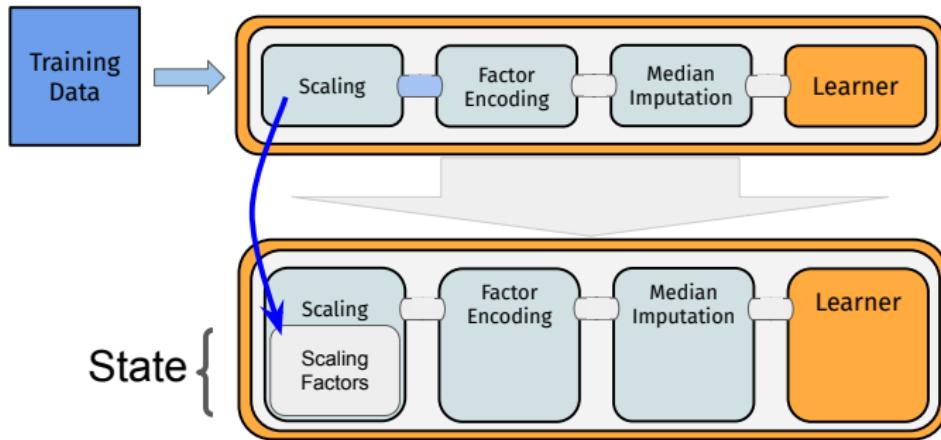


MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

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

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

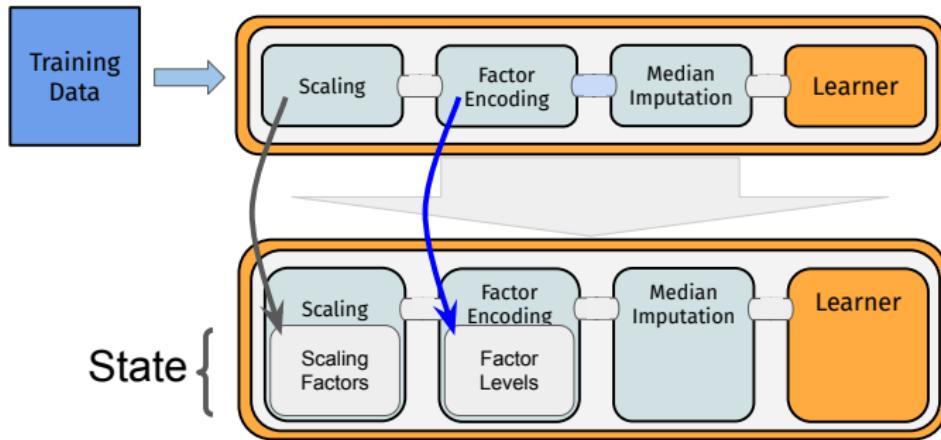


MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

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

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

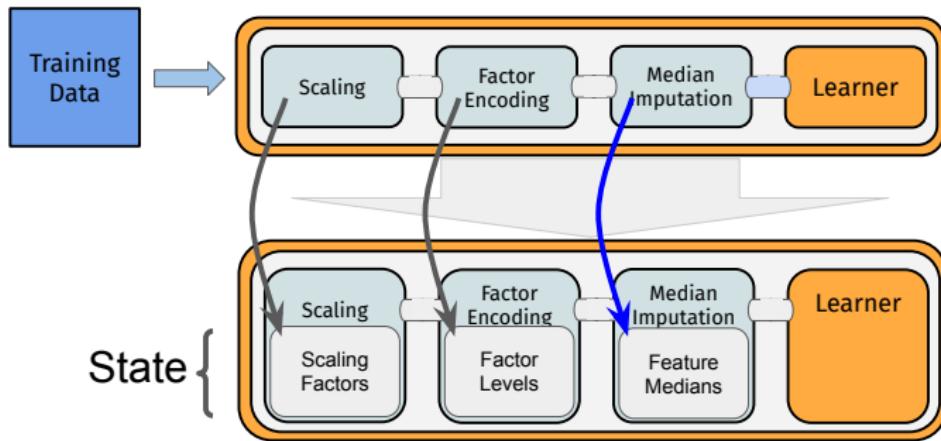


MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

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

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

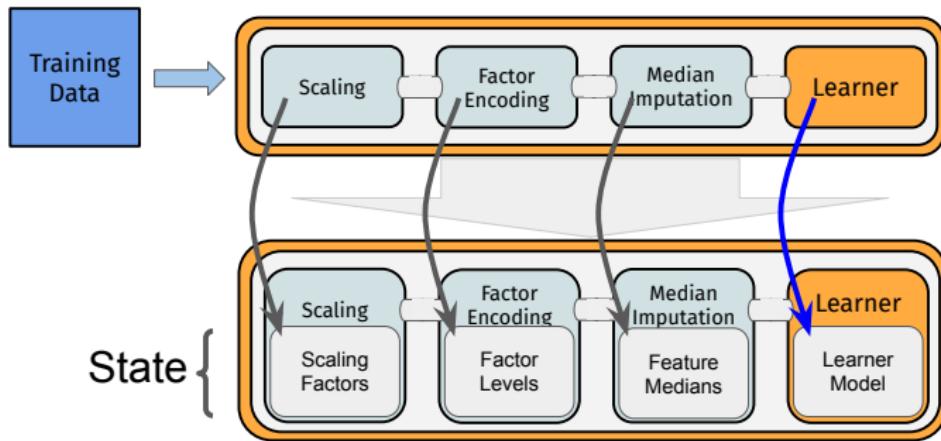


MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

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

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

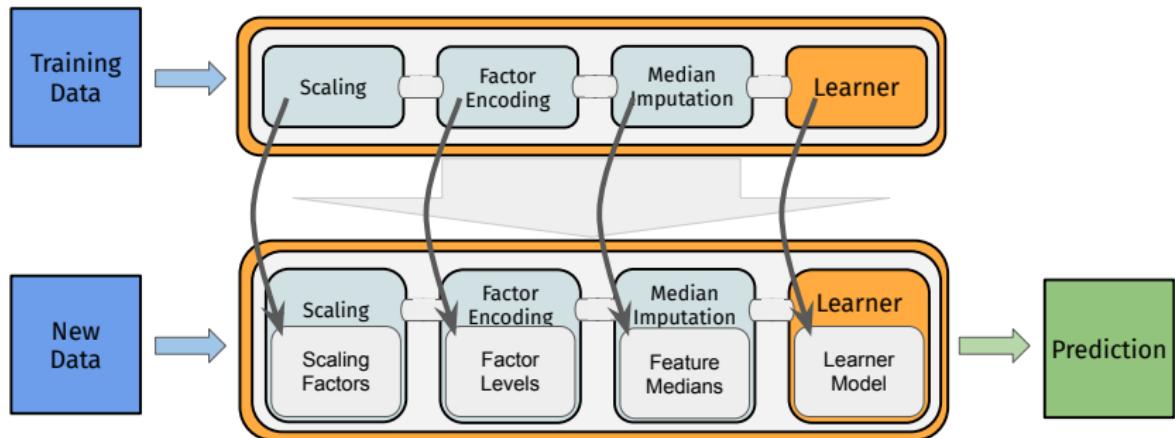


MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

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

```
glnr$predict(task)
```



MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

- 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  
#>     4.163367     1.424451     5.921098     3.098387
```

MLR3PIPELINES IN ACTION

Linear Preprocessing Pipeline

- Setting / retrieving parameters: `$param_set`

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

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

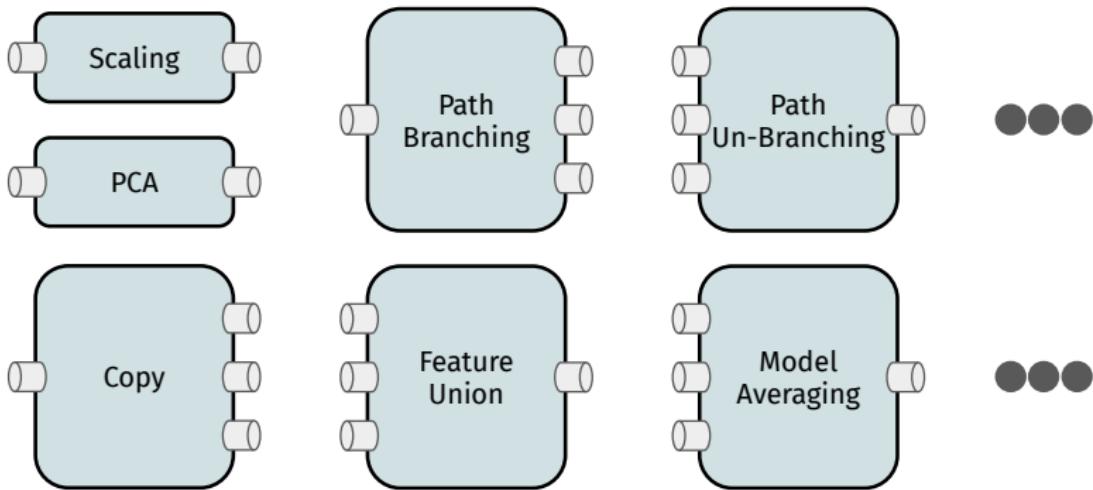
```
graph_pp$pipeops$scale$state$scale  
#> Petal.Length  Petal.Width Sepal.Length  Sepal.Width  
#>     4.163367    1.424451    5.921098    3.098387
```

- 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    0.3362663    0.140405    0.8613268   1.1296201  
#> 2:  setosa    0.3362663    0.140405    0.8275493   0.9682458  
#> 3:  setosa    0.3122473    0.140405    0.7937718   1.0327956
```

Nonlinear Pipelines

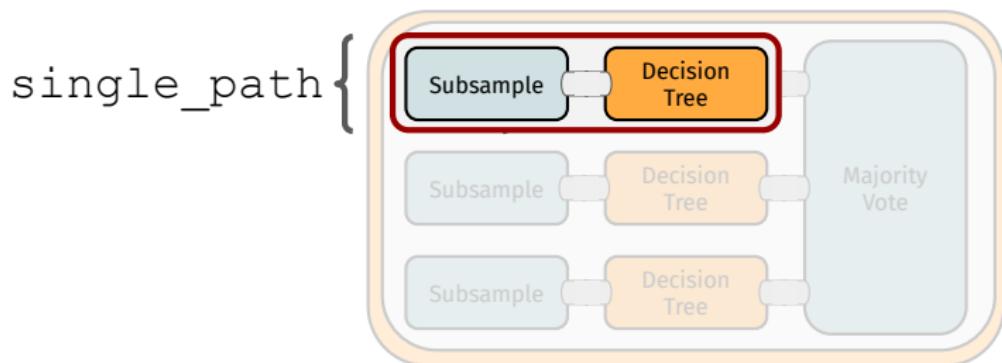
PIPEOPS WITH MULTIPLE INPUTS / OUTPUTS



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

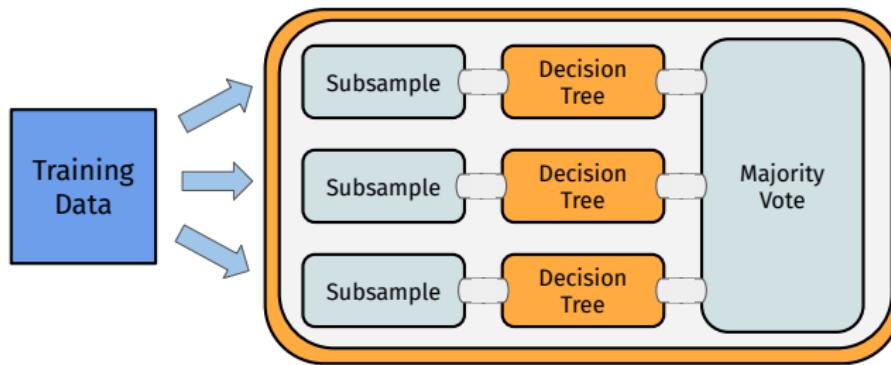
```
single_path = po("subsample") %>>% lrn("classif.rpart")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

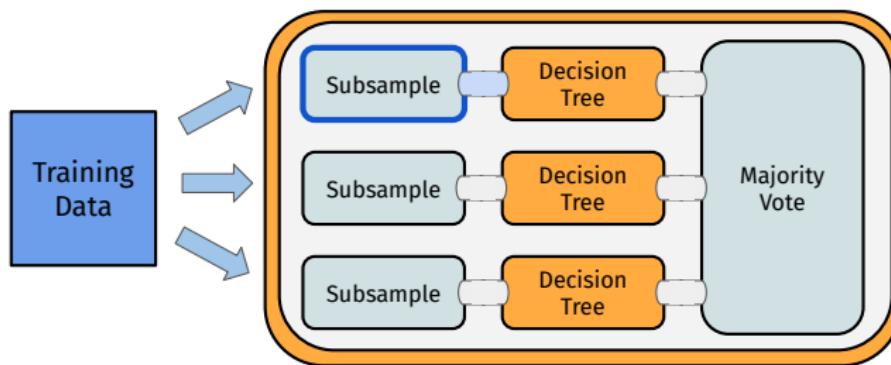
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

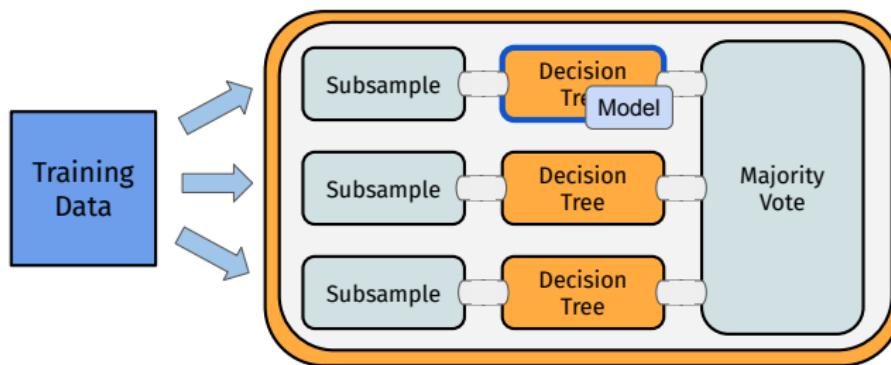
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

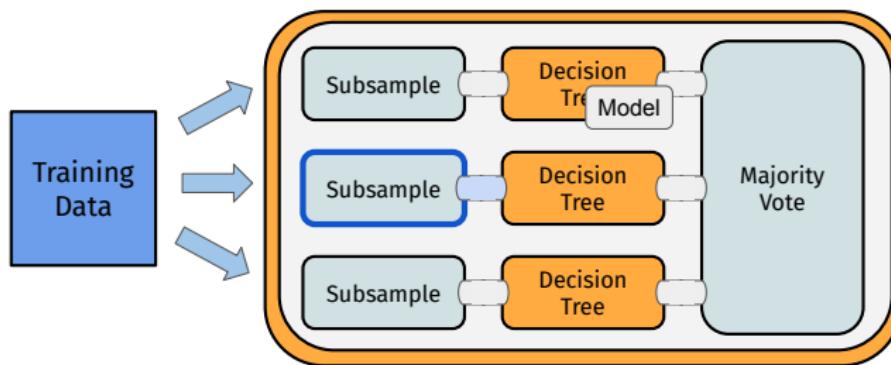
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

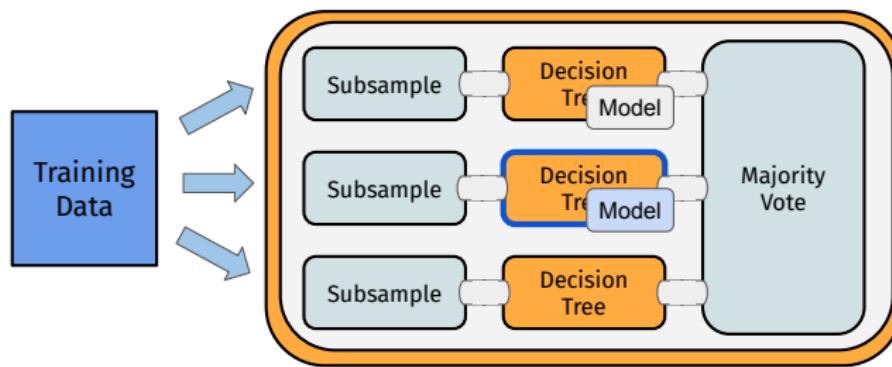
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

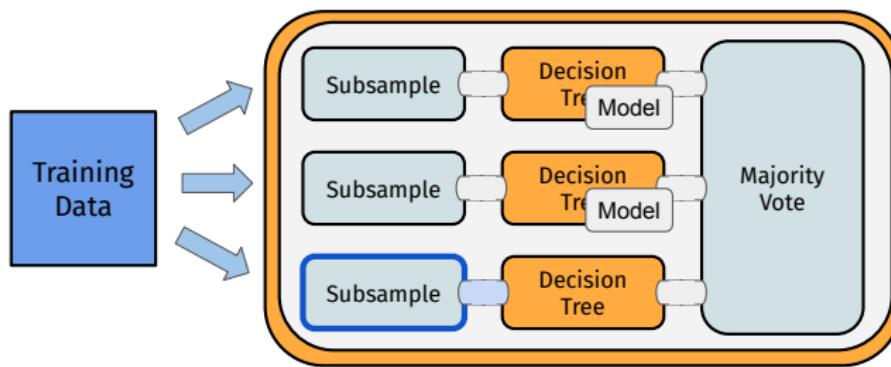
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

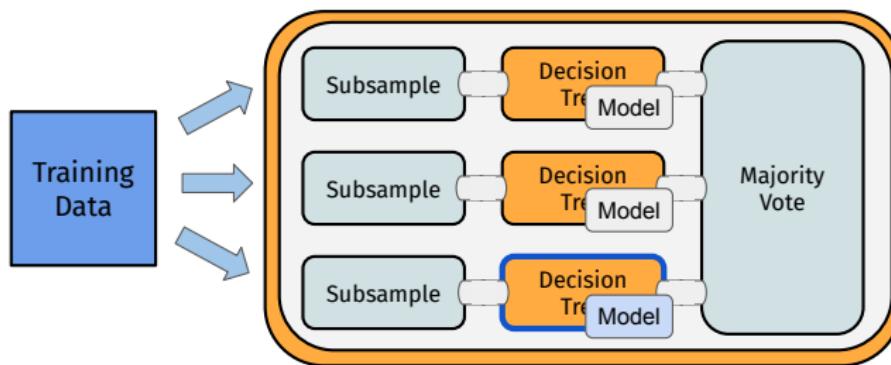
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

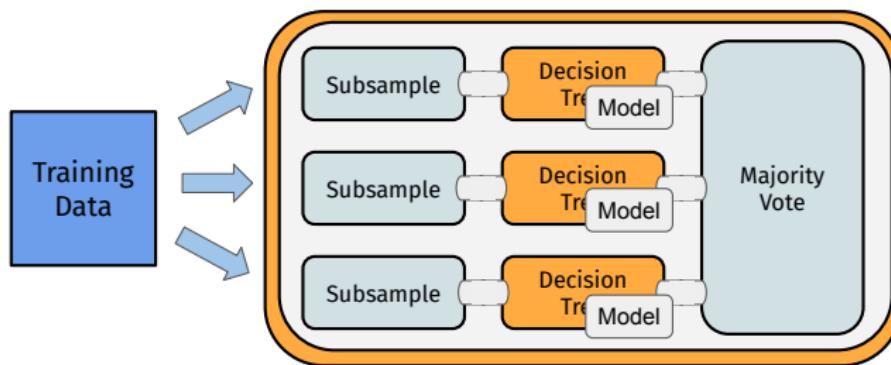
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



MLR3PIPELINES IN ACTION

Ensemble Method: Bagging

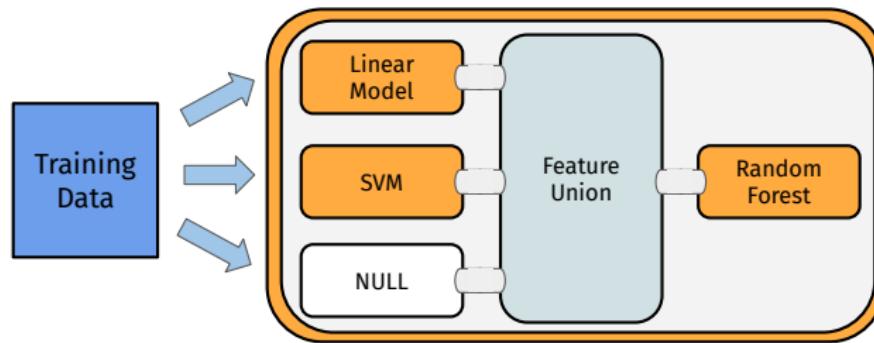
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
  po("classifavg")
```



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")
```

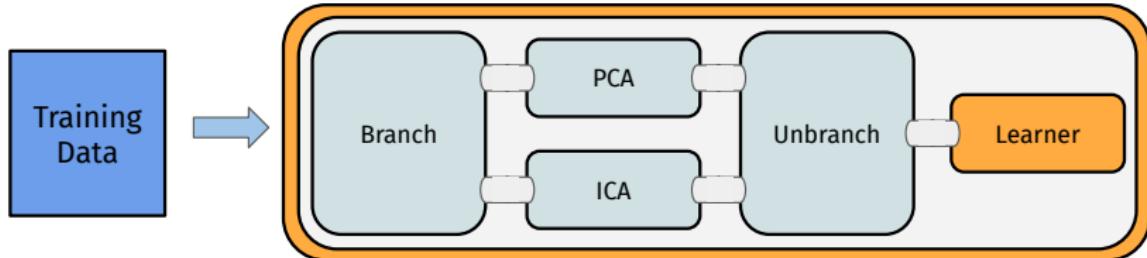


MLR3PIPELINES IN ACTION

Branching

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

Execute only one of several alternative paths

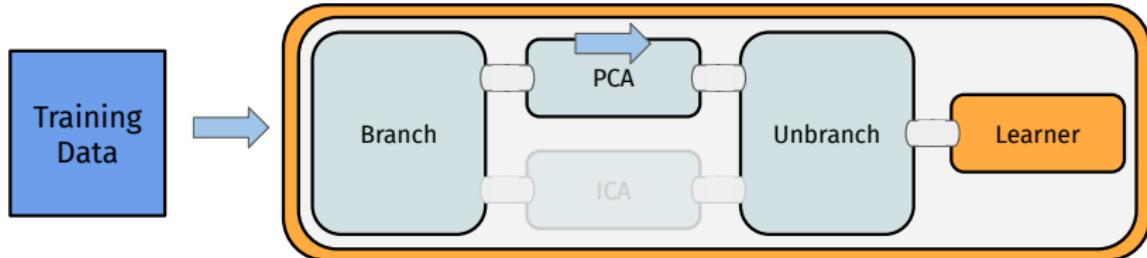


MLR3PIPELINES IN ACTION

Branching

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

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

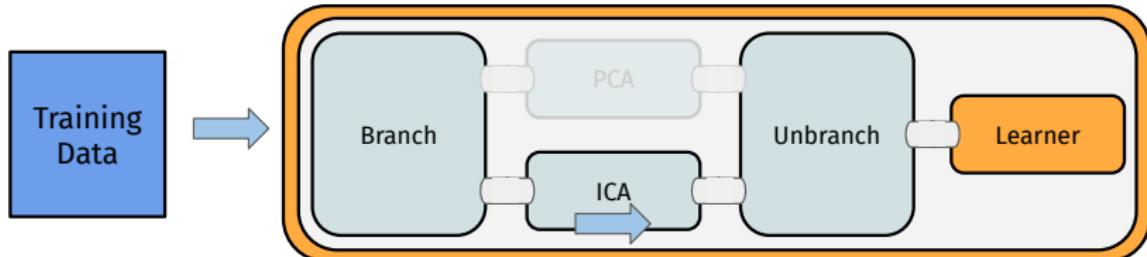


MLR3PIPELINES IN ACTION

Branching

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

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



Targeting Columns

TARGETING COLUMNS

Two ways of restricting actions to individual columns:

- Individual PipeOps: `affect_columns` parameter
 - Subgraphs, `po("select")`, and `po("featureunion")`
- ⇒ 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
#> 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")`
 - ⇒ Both make use of column Selectors

Using `affect_columns`:

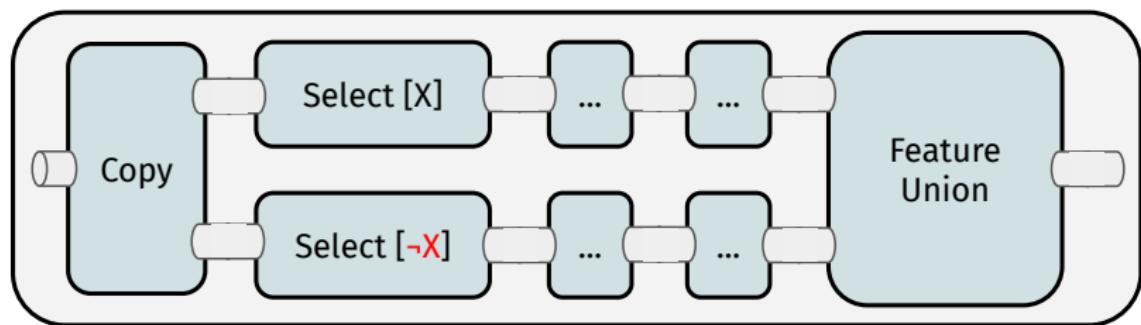
```
sel = selector_grep("^Sepal")  
  
partial_pca = po("pca", affect_columns = sel)  
  
result = partial_pca$train(list(task))  
  
result[[1]]$data(1:3)  
  
#>   Species       PC1       PC2 Petal.Length Petal.Width  
#> 1:  setosa -0.7781478  0.37813255      1.4        0.2  
#> 2:  setosa -0.9350903 -0.13700728      1.4        0.2  
#> 3:  setosa -1.1513076  0.04533873      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")`
 - ⇒ 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")`
 - ⇒ 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
#> 1:  setosa -0.7781478  0.37813255      1.4        0.2
#> 2:  setosa -0.9350903 -0.13700728      1.4        0.2
#> 3:  setosa -1.1513076  0.04533873      1.3        0.2
```

“Pipelines” Dictionary & Short Form

“PIPELINES” DICTIONARY & SHORT FORM

Many frequently used *patterns* for pipelines

- Making Learners robust to bad data (imputation + feature encoding + ...)
- Bagging
- Branching

“PIPELINES” DICTIONARY & SHORT FORM

Many frequently used *patterns* for pipelines

- Making Learners robust to bad data (imputation + feature encoding + ...)
- Bagging
- Branching

Collection of these is in `mlr3pipelines`

```
head(as.data.table(mlr_pipeops), 5)[, list(key, input.num, output.num)]
```

```
#>           key input.num output.num
#> 1:      boxcox      1         1
#> 2:     branch      1        NA
#> 3:      chunk      1        NA
#> 4: classbalancing      1         1
#> 5:    classifavg     NA         1
```

“PIPELINES” DICTIONARY & SHORT FORM

Many frequently used *patterns* for pipelines

- Making Learners robust to bad data (imputation + feature encoding + ...)
- Bagging
- Branching

Collection of these is in `mlr3pipelines`

`po()` accesses the `mlr_pipeops` “Dictionary”.

```
pca = po("pca")
pca

#> PipeOp: <pca> (not trained)
#> values: <list()>
#> Input channels <name [train type, predict type]>:
#>   input [Task,Task]
#> Output channels <name [train type, predict type]>:
#>   output [Task,Task]
```

AutoML with mlr3pipelines

AUTOML <3 PIPELINES

- AutoML: Automatic Machine Learning

AUTOML <3 PIPELINES

- AutoML: Automatic Machine Learning
- Let the algorithm make decisions about
 - ➊ *what learner* to use,
 - ➋ *what preprocessing* to use, and
 - ➌ *what hyperparameters* to use.

AUTOML <3 PIPELINES

- AutoML: Automatic Machine Learning
- Let the algorithm make decisions about
 - ➊ *what learner* to use,
 - ➋ *what preprocessing* to use, and
 - ➌ *what hyperparameters* to use.
- (1) and (2) are decisions about *graph structure* in `mlr3pipelines`

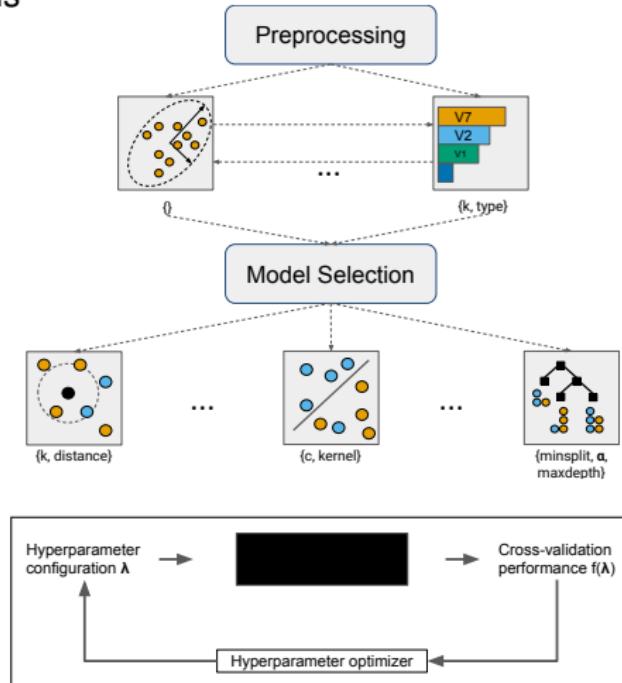
AUTOML <3 PIPELINES

- AutoML: Automatic Machine Learning
 - Let the algorithm make decisions about
 - ➊ *what learner* to use,
 - ➋ *what preprocessing* to use, and
 - ➌ *what hyperparameters* to use.
 - (1) and (2) are decisions about *graph structure* in `mlr3pipelines`
- ⇒ The problem reduces to **pipelines + parameter tuning**

AUTOML WITH MLR3PIPELINES

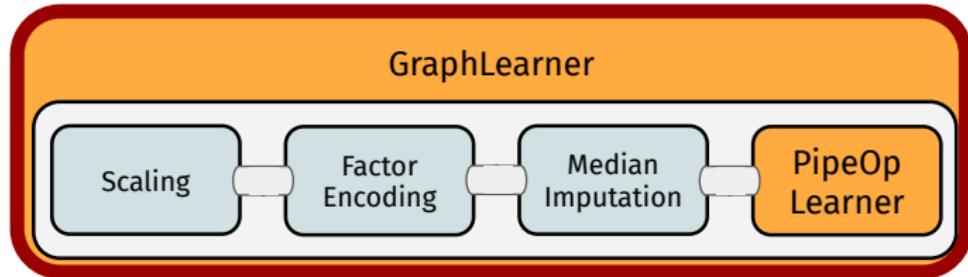
AutoML in a Nutshell

- Preprocessing steps
- ML Algorithms
- Tuner



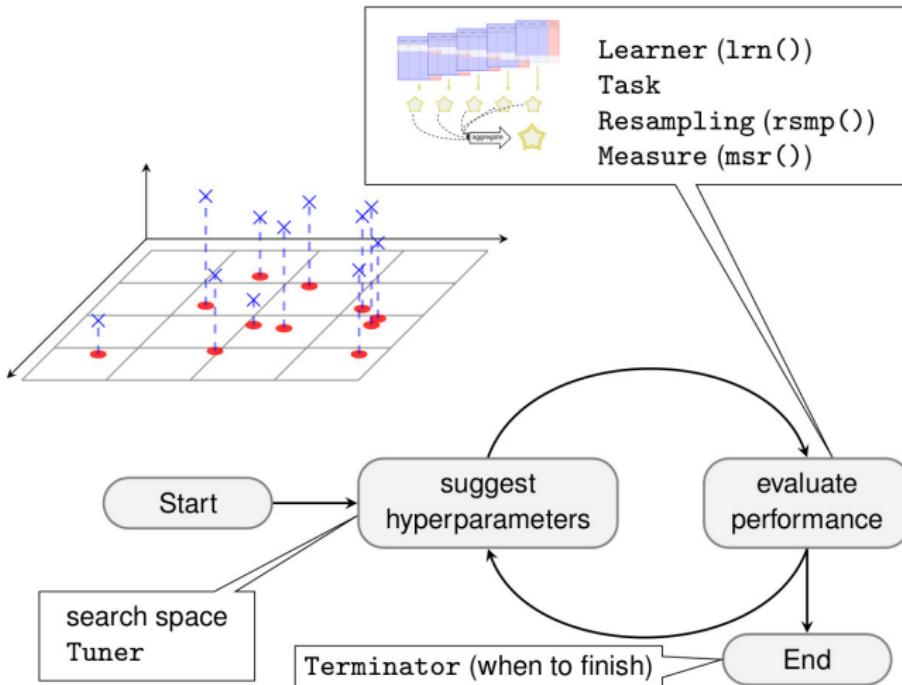
GRAPHLEARNER

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



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

TUNING

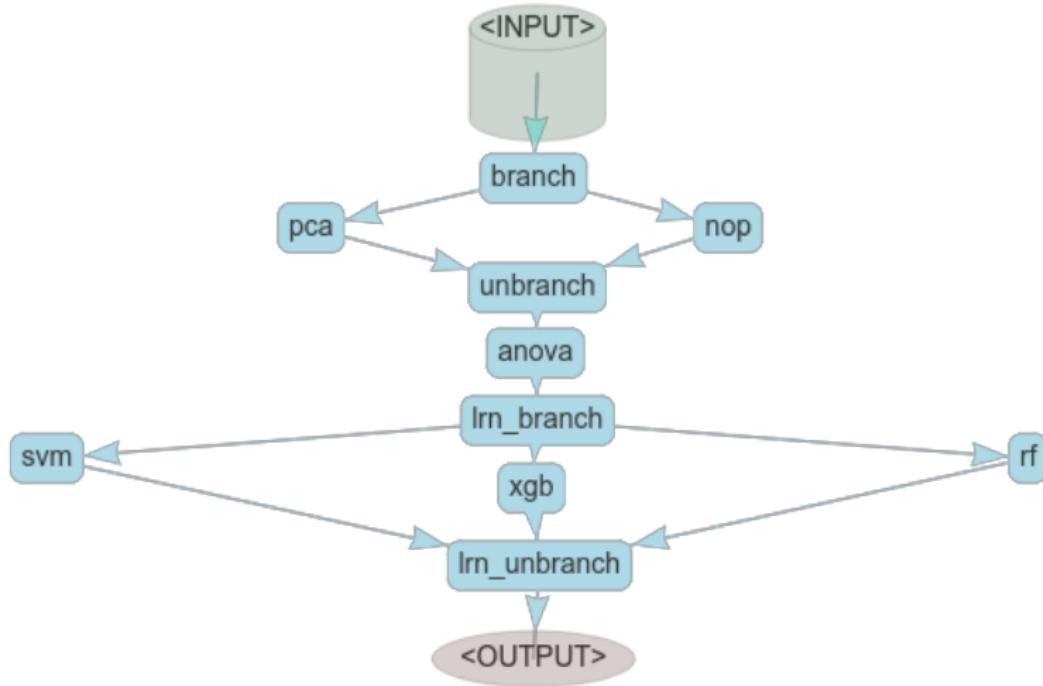


PIPELINES TUNING

- Works **exactly** as in basic `mlr3` / `mlr3tuning`
- PipeOps have *hyperparameters* (using `paradox` pkg)
- Graphs have hyperparameters of all components *combined*
- ⇒ 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 = ParamSet$new(list(
  ParamFct$new("branch.selection", levels = c("pca", "nothing")),
  ParamDbl$new("anova.filter.frac", lower = 0.1, upper = 1),
  ParamFct$new("lrn_branch.selection", levels = c("svm", "xgb", "rf")),
  ParamInt$new("rf.mtry", lower = 1L, upper = 20L),
  ParamInt$new("xgb.nrounds", lower = 1, upper = 500),
  ParamDbl$new("svm.cost", lower = -12, upper = 4),
  ParamDbl$new("svm.gamma", 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$new(tsk("sonar"), glnr, rsmp("cv", folds=3),
  msr("classif.ce"), ps, trm("evals", n_evals = 10))
tnr("random_search")$optimize(inst)
```