



LUDWIG-  
MAXIMILIANS-  
UNIVERSITÄT  
MÜNCHEN

# Machine Learning Pipelines in R

## mlr3pipelines

Department of Statistics – LMU Munich

November 07, 2019



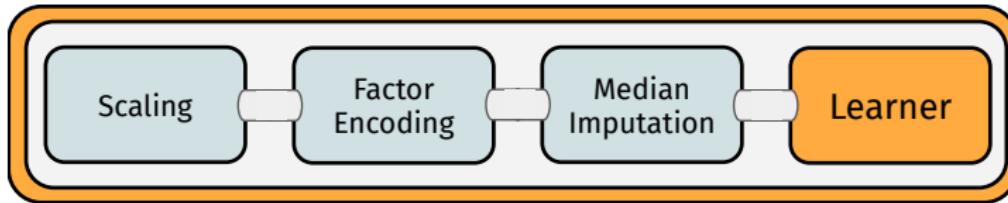
# Intro

# MLR3PIPELINES

## Machine Learning Workflows:

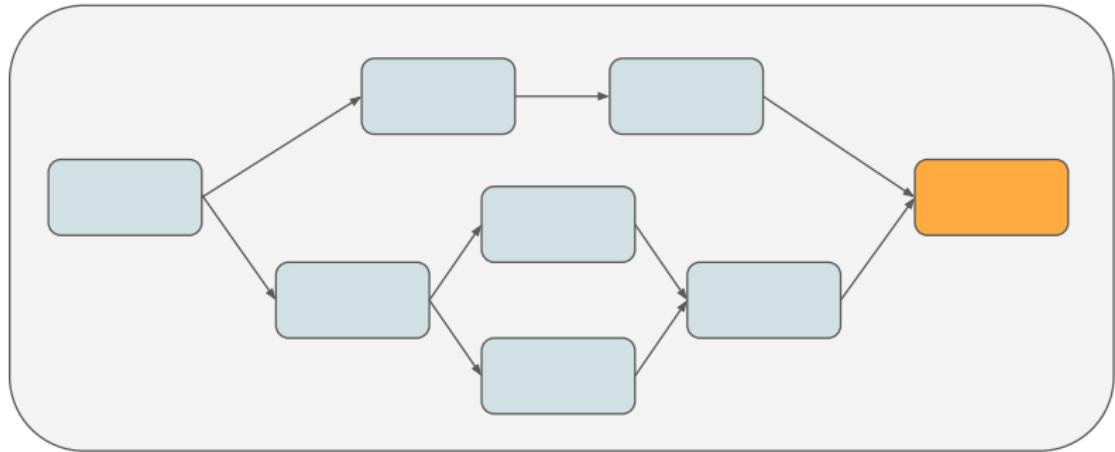
- **Preprocessing:** Feature extraction, feature selection, missing data imputation,...
- **Ensemble methods:** Model averaging, model stacking
- **mlr3:** modular model fitting

→ `mlr3pipelines`: modular ML workflows



# MACHINE LEARNING WORKFLOWS

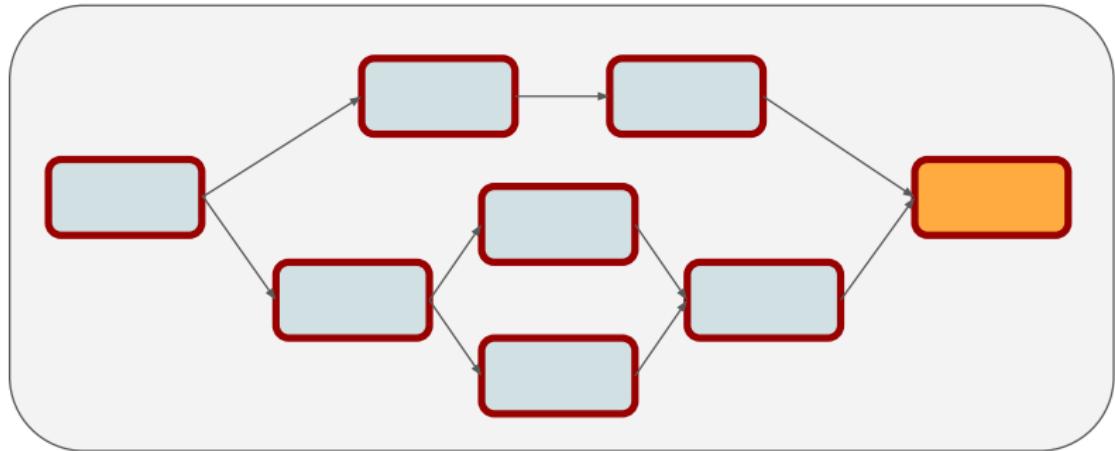
– what do they look like?



# MACHINE LEARNING WORKFLOWS

– what do they look like?

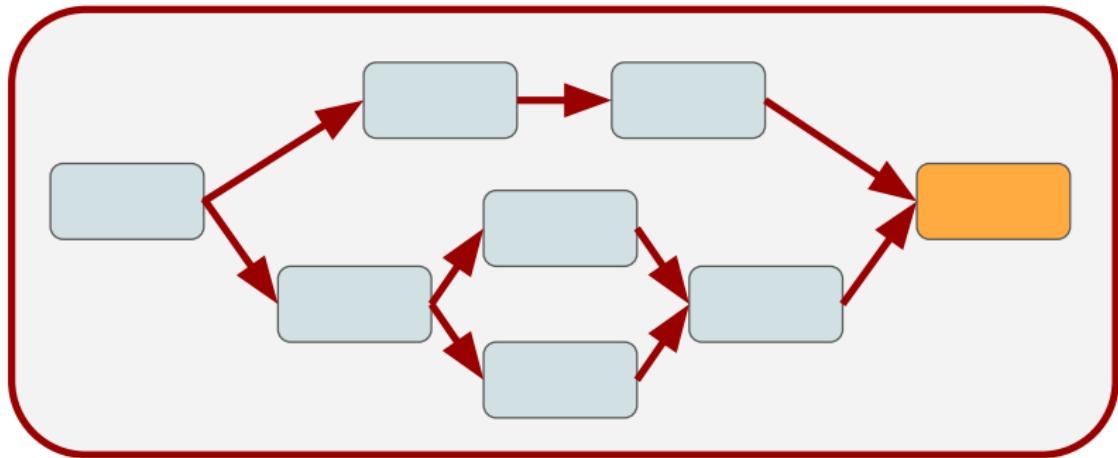
- **Building blocks:** *what is happening? → PipeOp*



# MACHINE LEARNING WORKFLOWS

– what do they look like?

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



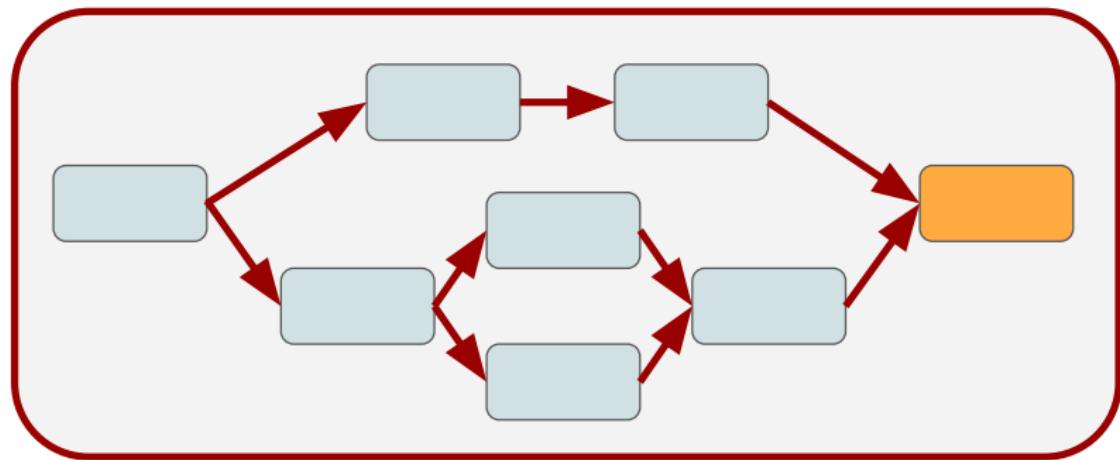
# MACHINE LEARNING WORKFLOWS

– 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

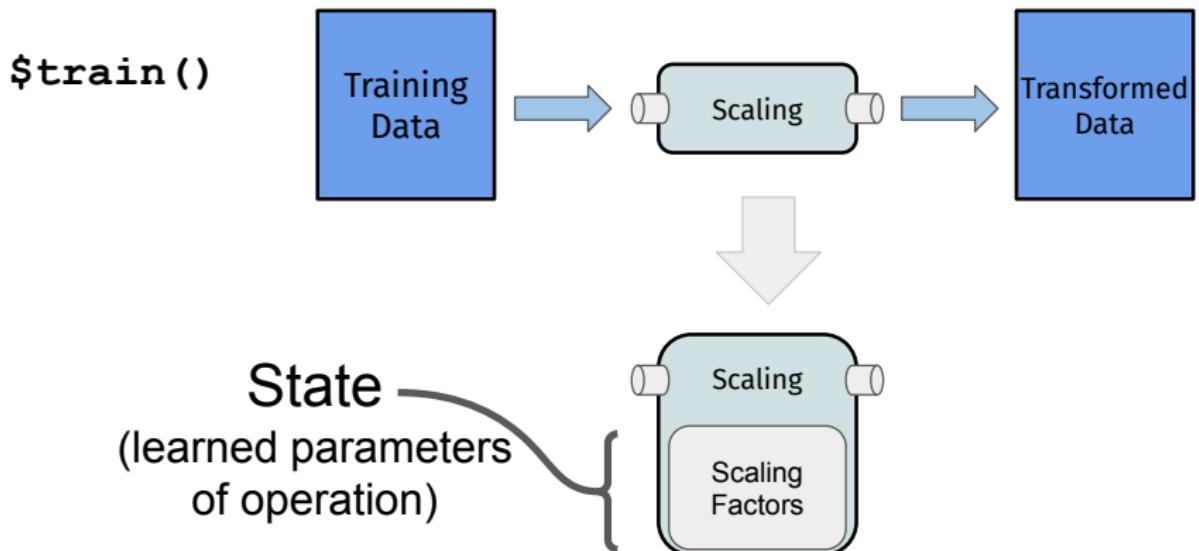
- `pip = po("scale")` to construct



# THE BUILDING BLOCKS

## PipeOp: Single Unit of Data Operation

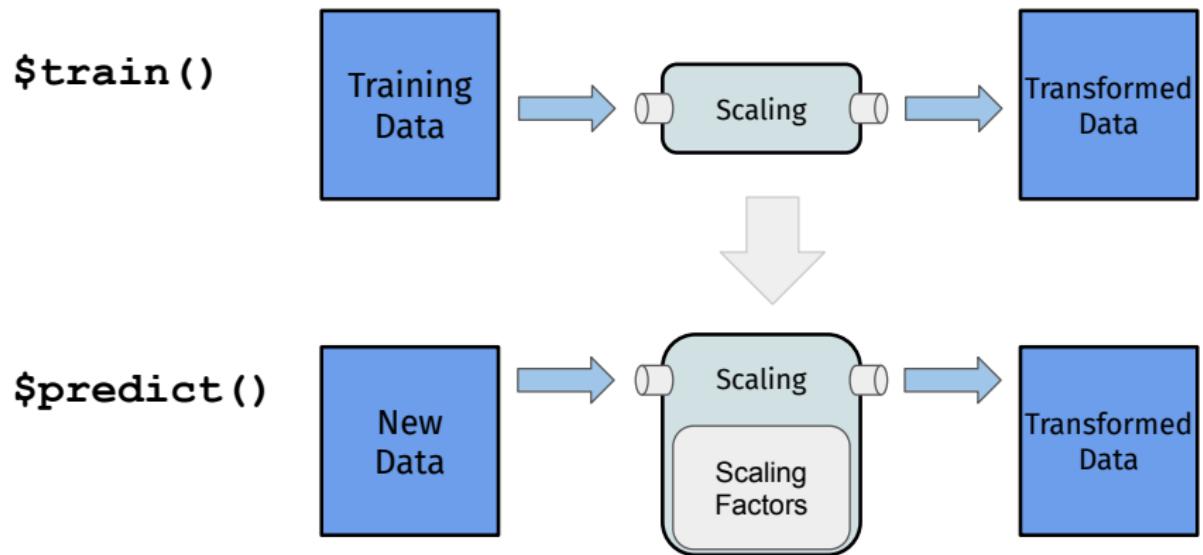
- `pip = po("scale")` to construct
- `pip$train()`: process data and create `pip$state`



# THE BUILDING BLOCKS

## PipeOp: Single Unit of Data Operation

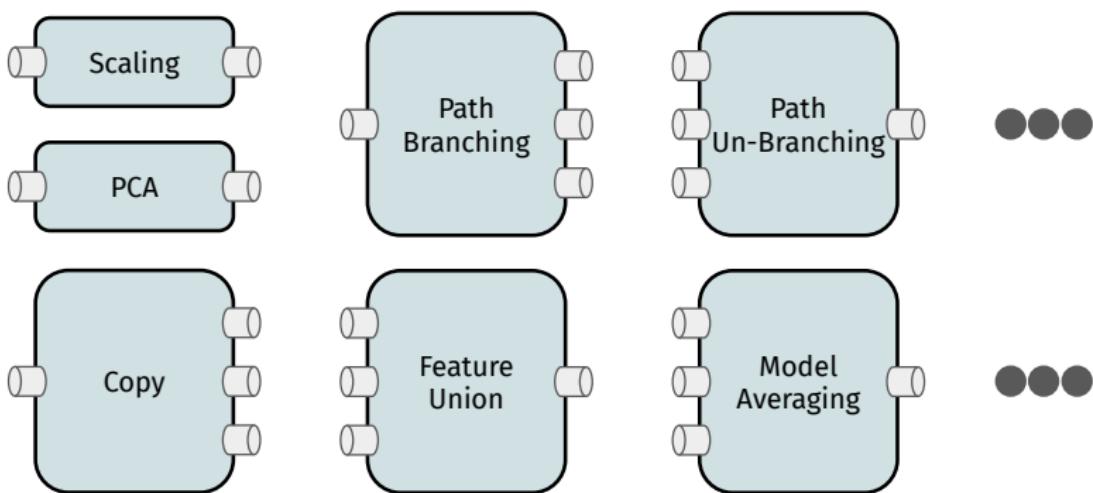
- `pip = po("scale")` to construct
- `pip$train()`: process data and create `pip$state`
- `pip$predict()`: process data depending on the `pip$state`



# 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 input.num output.num ...
#> 1    boxcox bestNormalize       1          1
#> 2    branch                         1        NA
#> 3    chunk                           1        NA
#> 4 classbalancing                   1          1
#> 5 classifavg           stats       NA          1
#> 6 classweights                    1          1
#> 7 colapply                         1          1
#> 8 collapsefactors                  1          1
#> 9   copy                            1        NA
#> 10 encode           stats       1          1
#> 11 encodeimpact                   1          1
#> 12 encodelmer lme4, nloptr       1          1
#> 13 featureunion                   NA          1
#> 14 filter                          1          1
#> 15 fixfactors                     1          1
#> 16 histbin           graphics       1          1
#> 17     ica            fastICA       1          1
#> [...]
```

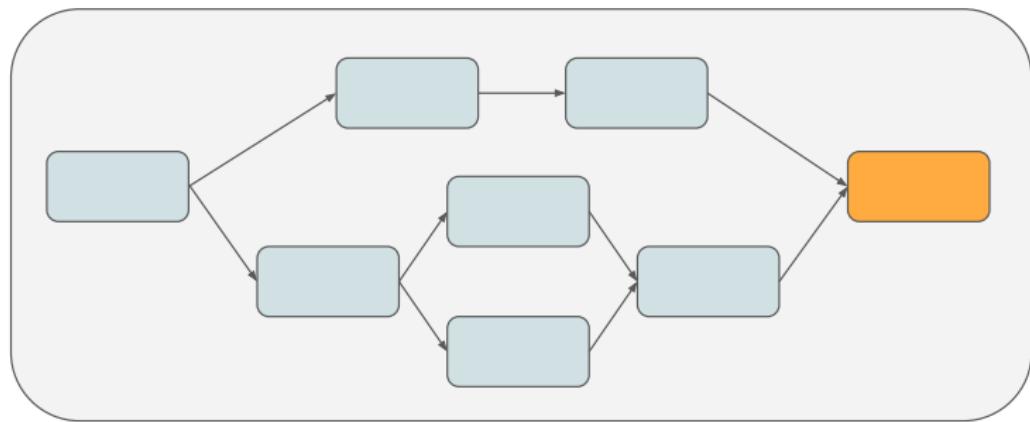
# PIPEOPS SO FAR AND PLANNED

- Simple data preprocessing operations (scaling, Box Cox, Yeo Johnson, PCA, ICA)
- Missing value imputation (sampling, median, as 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 (*planned*)
- Time series and spatio-temporal data (*planned*)
- Multi-output and ordinal targets (*planned*)
- Outlier detection (*planned*)

# Graph Operations

# THE STRUCTURE

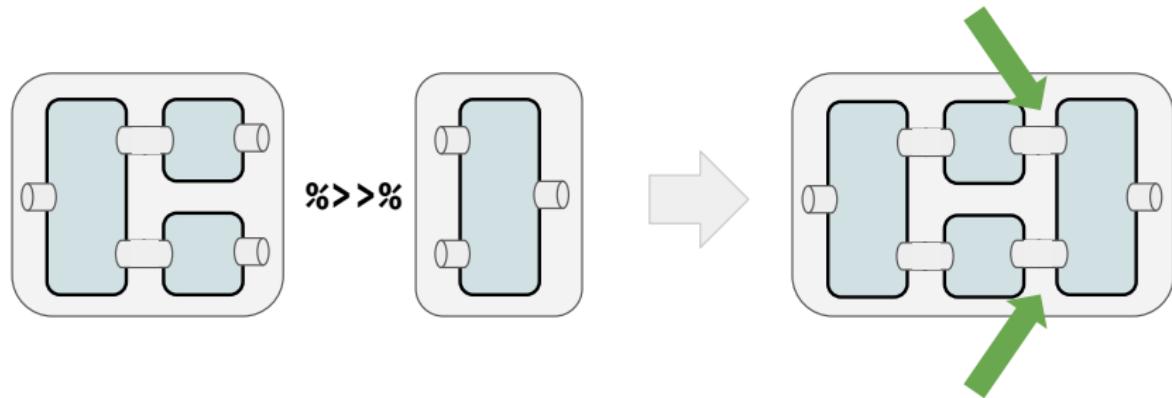
## Graph Operations



# THE STRUCTURE

## Graph Operations

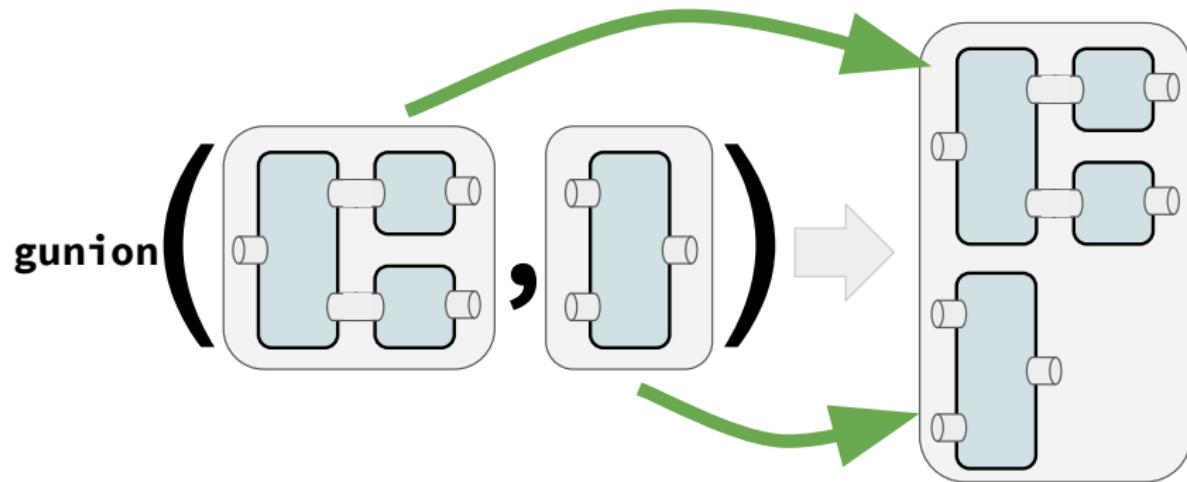
- The `%>>%`-operator concatenates Graphs and PipeOps



# THE STRUCTURE

## Graph Operations

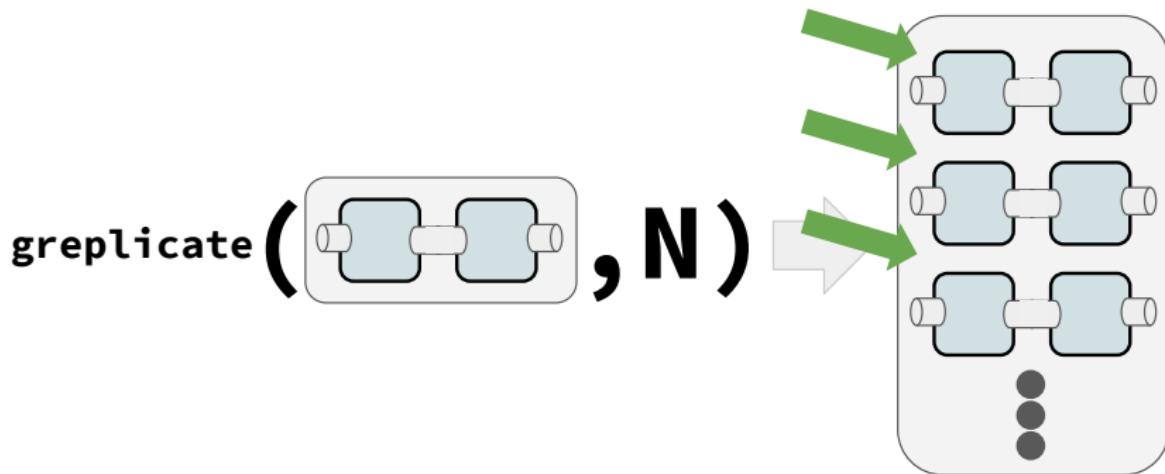
- The `%>>%`-operator concatenates Graphs and PipeOps
- The `gunion()`-function unites Graphs and PipeOps



# THE STRUCTURE

## Graph Operations

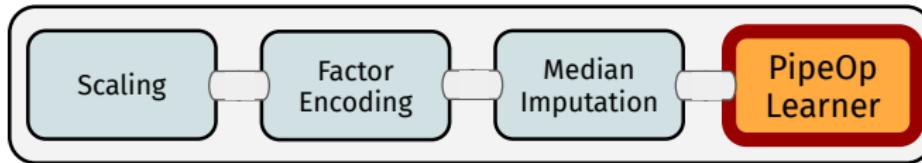
- The `%>>%`-operator concatenates Graphs and PipeOps
- The `gunion()`-function unites Graphs and PipeOps
- The `grePLICATE()`-function unites copies of Graphs and PipeOps



# LEARNERS AND GRAPHS

## PipeOpLearner

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



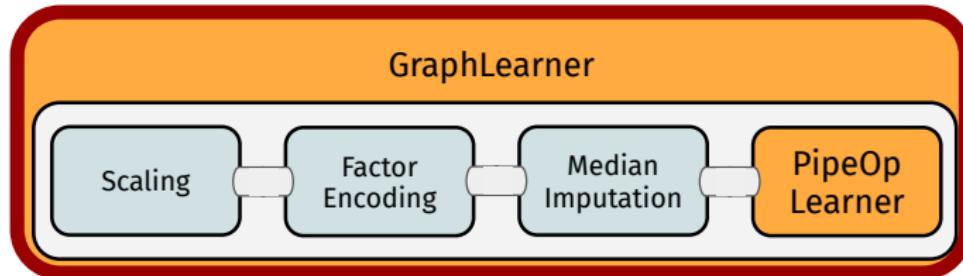
# LEARNERS AND GRAPHS

## PipeOpLearner

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

## GraphLearner

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

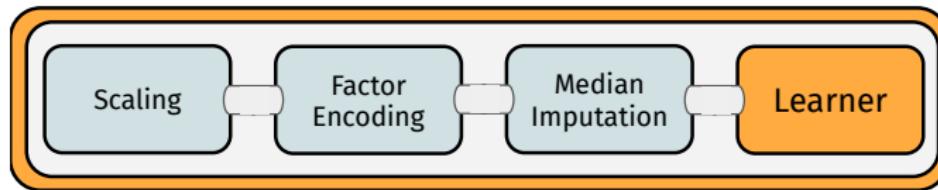


# **Linear Pipelines**

# MLR3PIPELINES IN ACTION

## Linear Preprocessing Pipeline

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

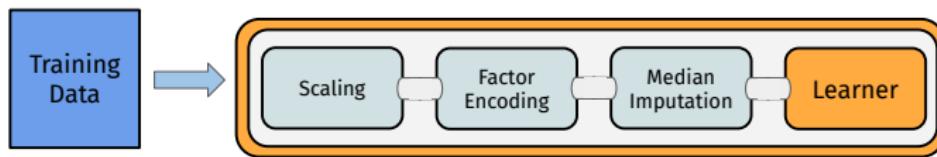


# MLR3PIPELINES IN ACTION

## Linear Preprocessing Pipeline

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

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

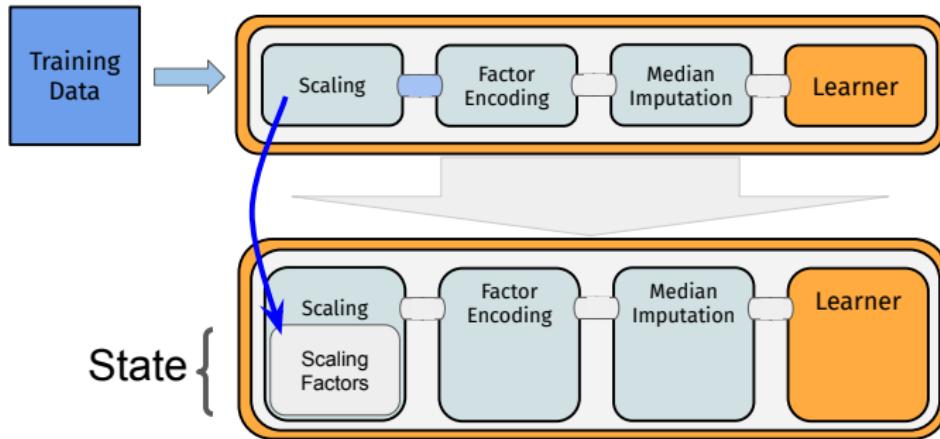


# MLR3PIPELINES IN ACTION

## Linear Preprocessing Pipeline

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

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

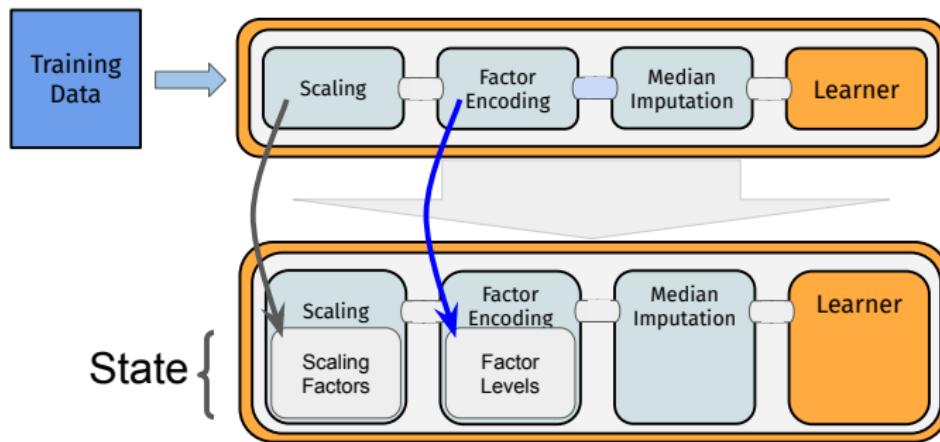


# MLR3PIPELINES IN ACTION

## Linear Preprocessing Pipeline

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

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

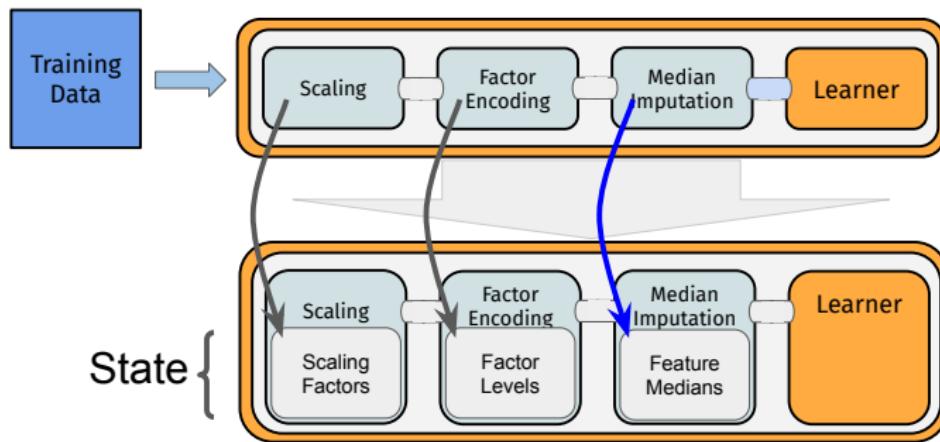


# MLR3PIPELINES IN ACTION

## Linear Preprocessing Pipeline

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

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

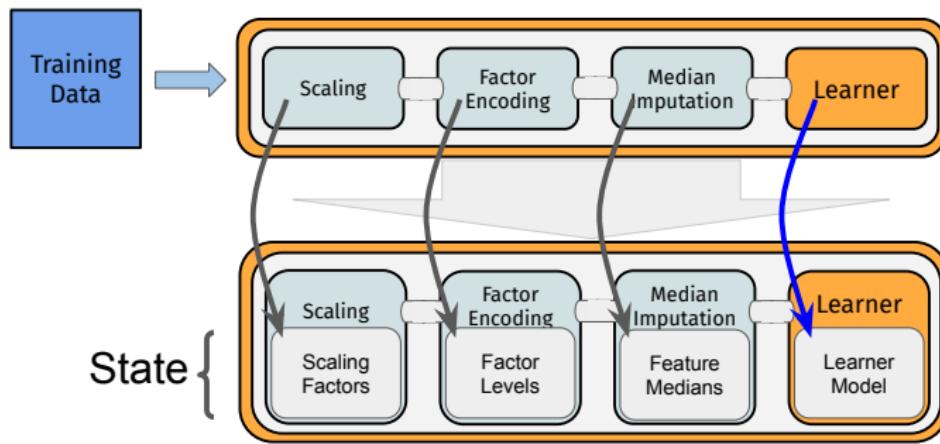


# MLR3PIPELINES IN ACTION

## Linear Preprocessing Pipeline

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

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

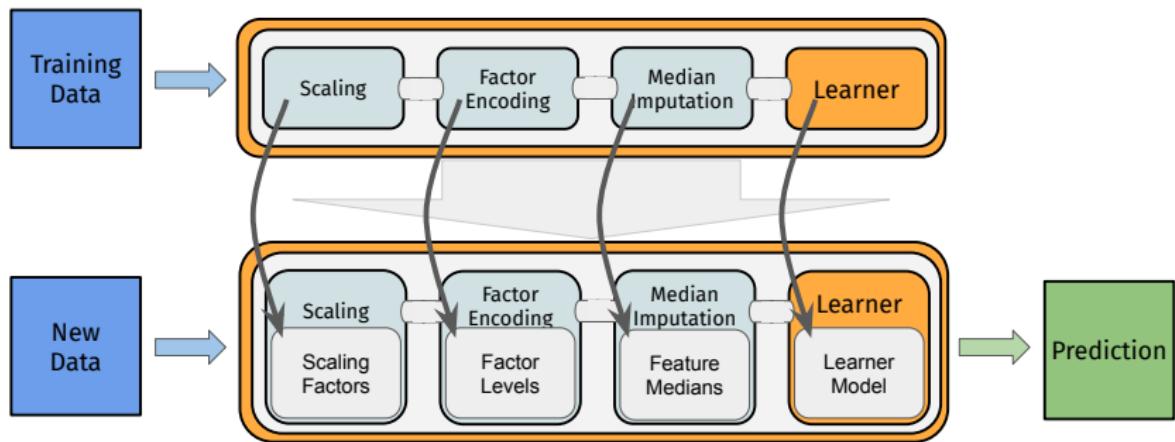


# MLR3PIPELINES IN ACTION

## Linear Preprocessing Pipeline

- `train()`ing: Data propagates and creates `$states`
- `predict()`ition: Data propagates, uses `$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  
#>     4.163367      1.424451      5.921098      3.098387
```

# 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
```

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

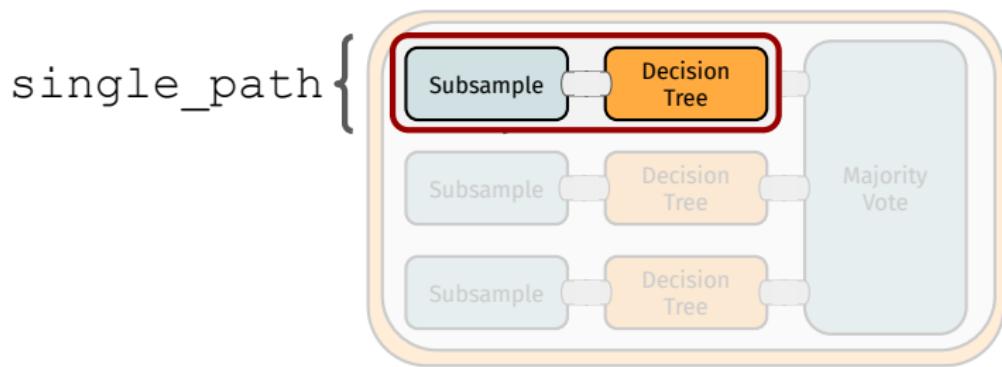
```
graph_pp$pipeops$scale$.result[[1]]$head(3)  
#>   Species Petal.Length Petal.Width Sepal.Length Sepal.Width  
#> 1:  setosa    0.3362663    0.140405    0.8613268    1.1296201  
#> 2:  setosa    0.3362663    0.140405    0.8275493    0.9682458  
#> 3:  setosa    0.3122473    0.140405    0.7937718    1.0327956
```

# **Nonlinear Pipelines**

# MLR3PIPELINES IN ACTION

## Ensemble Method: Bagging

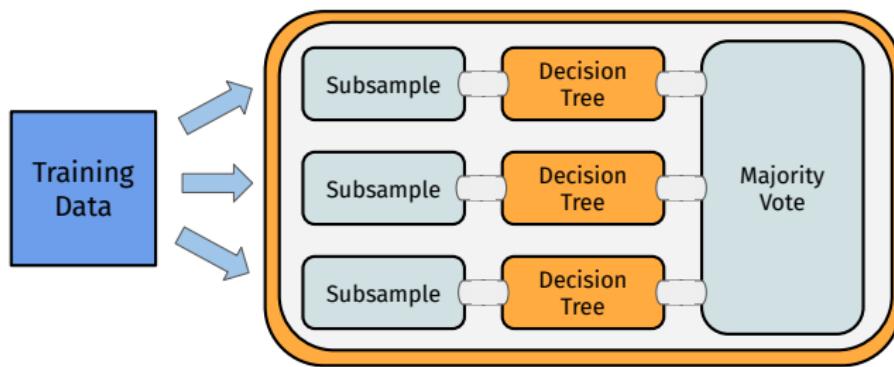
```
single_path = po("subsample") %>>% lrn("classif.rpart")
```



# MLR3PIPELINES IN ACTION

## Ensemble Method: Bagging

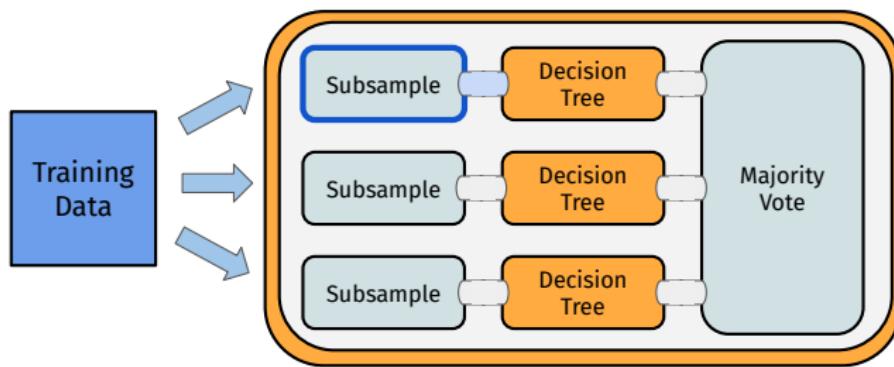
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = grepligate(single_path, n = 3) %>>%
  po("classifavg")
```



# MLR3PIPELINES IN ACTION

## Ensemble Method: Bagging

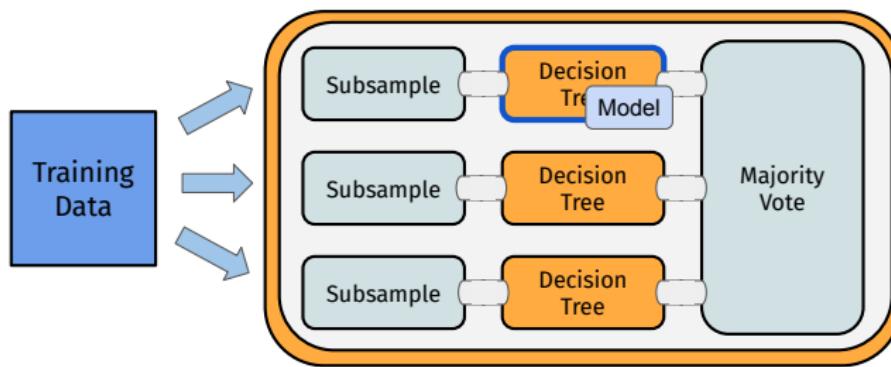
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = grepligate(single_path, n = 3) %>>%
  po("classifavg")
```



# MLR3PIPELINES IN ACTION

## Ensemble Method: Bagging

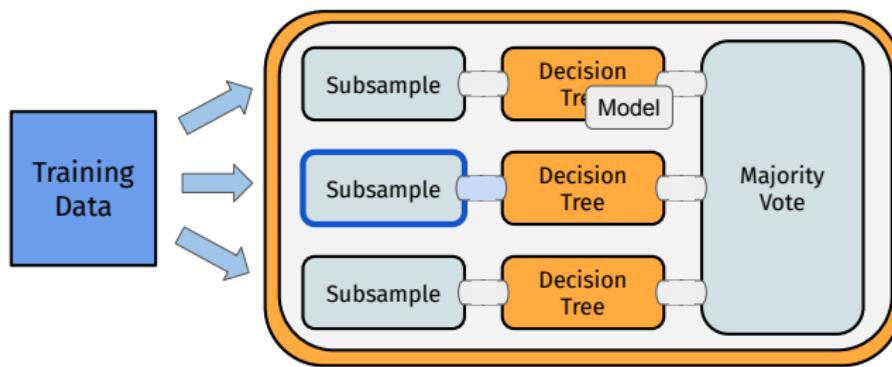
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = grepligate(single_path, n = 3) %>>%
  po("classifavg")
```



# MLR3PIPELINES IN ACTION

## Ensemble Method: Bagging

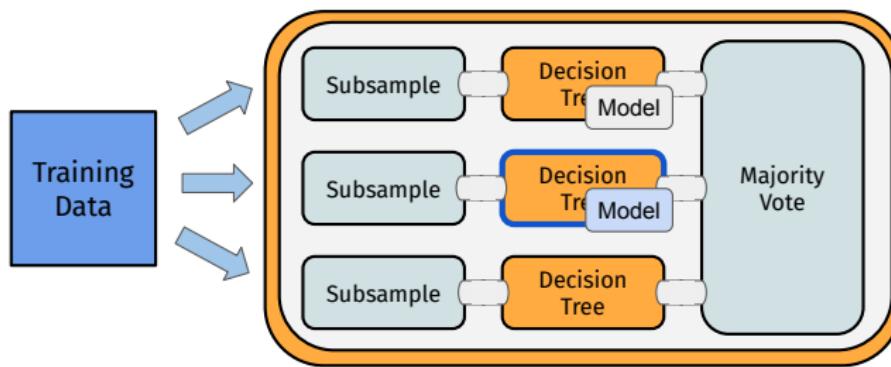
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = grepligate(single_path, n = 3) %>>%
  po("classifavg")
```



# MLR3PIPELINES IN ACTION

## Ensemble Method: Bagging

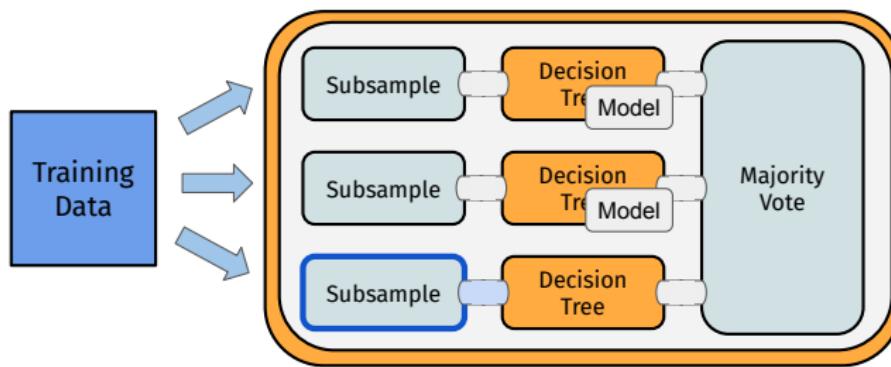
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = grepligate(single_path, n = 3) %>>%
  po("classifavg")
```



# MLR3PIPELINES IN ACTION

## Ensemble Method: Bagging

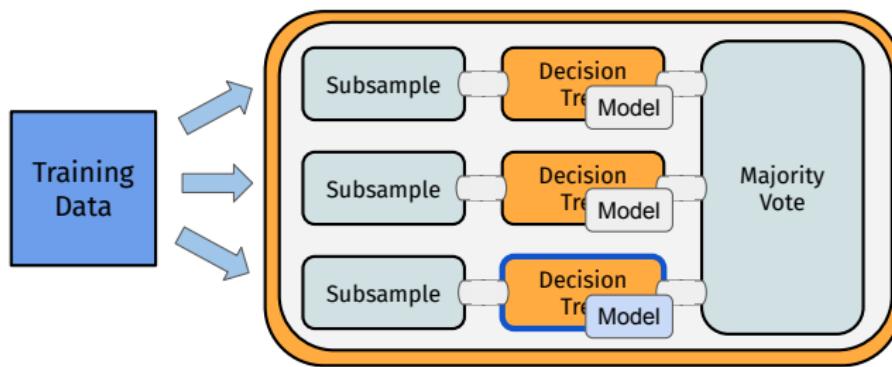
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = grepligate(single_path, n = 3) %>>%
  po("classifavg")
```



# MLR3PIPELINES IN ACTION

## Ensemble Method: Bagging

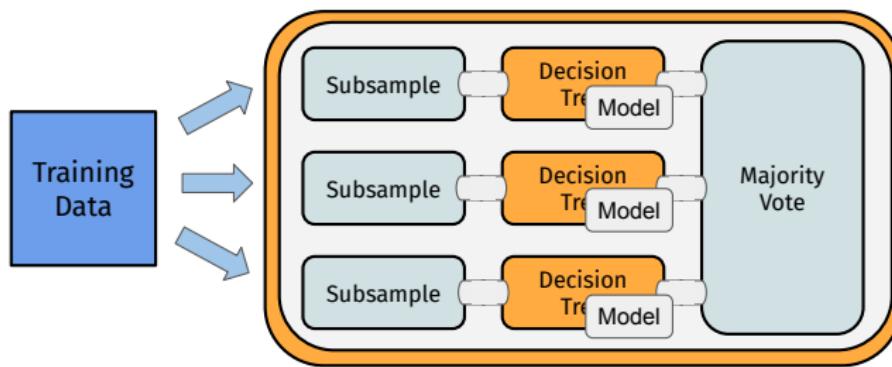
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = grepligate(single_path, n = 3) %>>%
  po("classifavg")
```



# MLR3PIPELINES IN ACTION

## Ensemble Method: Bagging

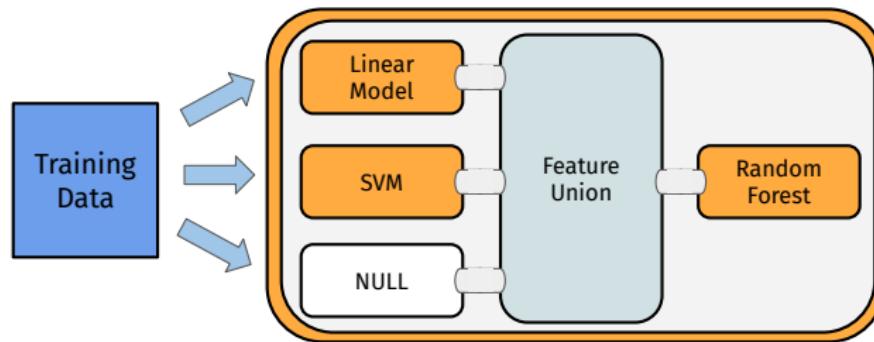
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = grepligate(single_path, n = 3) %>>%
  po("classifavg")
```



# 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")
```

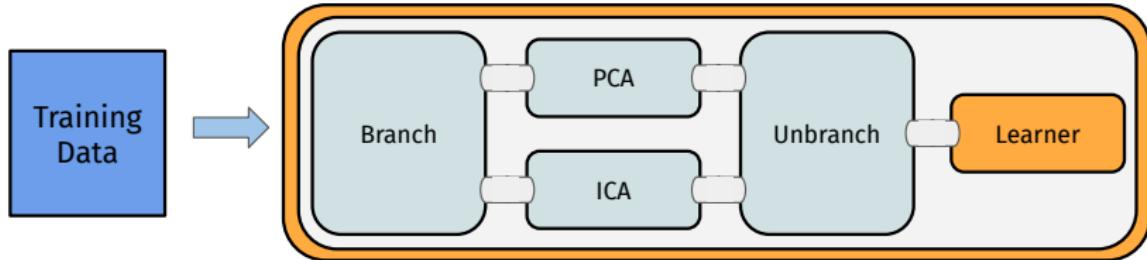


# MLR3PIPELINES IN ACTION

## Branching

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

Execute only one of several alternative paths

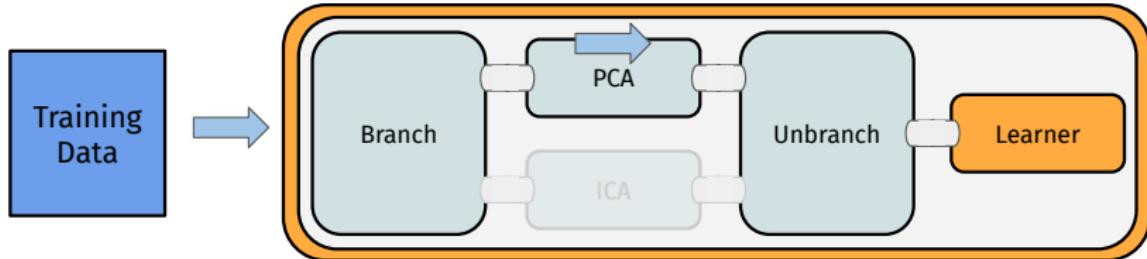


# MLR3PIPELINES IN ACTION

## 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"
```

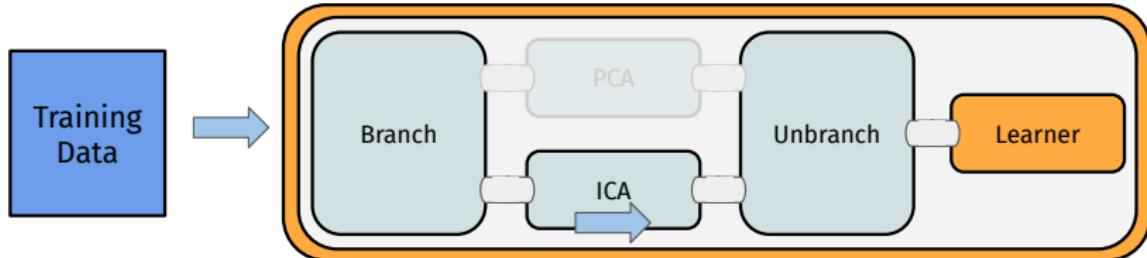


# MLR3PIPELINES IN ACTION

## 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

# 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)

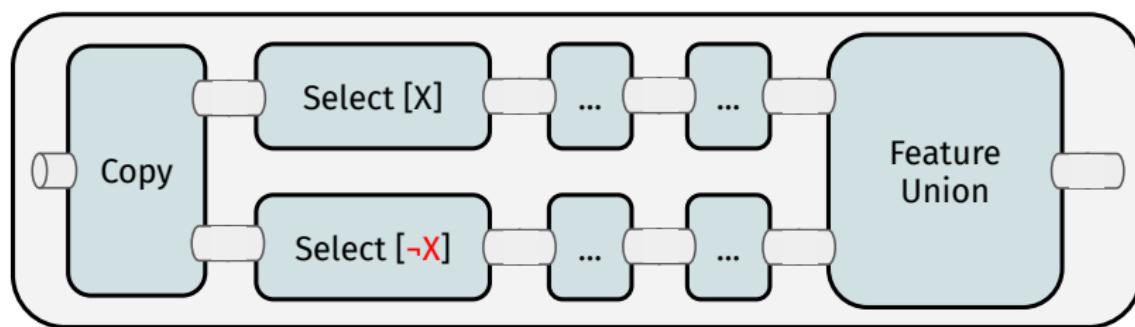
#>   Species       PC1        PC2 Petal.Length Petal.Width
#> 1:  setosa -0.7781478  0.37813255      1.4        0.2
#> 2:  setosa -0.9350903 -0.13700728      1.4        0.2
#> 3:  setosa -1.1513076  0.04533873      1.3        0.2
```

# 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
```

# **Hyperparameters and Tuning**

# HYPERPARAMETERS AND TUNING

- PipeOps have *hyperparameters* (using paradox pkg)
- Graphs have hyperparameters of all components *combined*
- ⇒ simultaneous **Tuning** of Learner and preprocessing (mlr3tuning package)

```
library("paradox") ; library("mlr3tuning")
glrn = po("scale") %>>% lrn("classif.rpart")
ps = ParamSet$new(list(
  ParamLgl$new("scale.scale"),
  ParamInt$new("classif.rpart.minsplit", 1, 20)
))
inst = TuningInstance$new(task, glrn,
  rsmp("cv"), msr("classif.ce"), ps,
  term("evals", n_evals = 10))

tnr("random_search")$tune(inst)

inst$result
```

# **Outro**

# MLR3PIPELINES

mlr3pipelines overview:

- Construct a PipeOp using `po()`
- Use Graph operators to connect them
  - `%>>%`—chain operations
  - `gunion()`—put operations in parallel
  - `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