Pipelines and AutoML with mlr3

`https://tinyurl.com/mlr3pipelines`

Department of Statistics – LMU Munich

May 28, 2020

SO YOU WANT TO DO ML IN R

- `R` gives you access to many machine learning methods
- . . . but without a unified interface
- . . . resampling and performance evaluation are cumbersome

SO YOU WANT TO DO ML IN R

- `R` gives you access to many machine learning methods
- ... but without a unified interface
- ... resampling and performance evaluation are cumbersome

mlr3 provides an interface to several machine learning algorithms for *training*, *predicting*, *resampling*, *tuning*, *benchmarks* and more.

SO YOU WANT TO DO ML IN R

- R gives you access to many machine learning methods
- ... but without a unified interface
- ... resampling and performance evaluation are cumbersome

mlr3 provides an interface to several machine learning algorithms for *training*, *predicting*, *resampling*, *tuning*, *benchmarks* and more.

Example:

```
data = tsk("iris")
algo = lrn("classif.ranger")
algo$train(data)
```

SO YOU WANT TO DO ML IN R

- R gives you access to many machine learning methods
- ... but without a unified interface
- ... resampling and performance evaluation are cumbersome

mlr3 provides an interface to several machine learning algorithms for *training*, *predicting*, *resampling*, *tuning*, *benchmarks* and more.

Example:

```
data = tsk("iris")
algo = lrn("classif.ranger")
algo$train(data)

algo$predict_newdata(data.frame(
  Sepal.Length = 4, Sepal.Width = 4,
  Petal.Length = 2, Petal.Width = 0.4
))

#> <PredictionClassif> for 1 observations:
#>  row_id truth response
#>       1  <NA>   setosa
```

SO YOU WANT TO DO ML IN R

- R gives you access to many machine learning methods
- ...but without a unified interface
- ...resampling and performance evaluation are cumbersome

mlr3 provides an interface to several machine learning algorithms for *training*, *predicting*, *resampling*, *tuning*, *benchmarks* and more.

Example:

```r
data = tsk("iris")
algo = lrn("classif.ranger")
rr = resample(data, algo, rsmp("cv"))
rr$aggregate(msr("classif.acc"))

#> classif.acc
#>        0.96
```

SO YOU WANT TO DO ML IN R

- R gives you access to many machine learning methods
- ... but without a unified interface
- ... resampling and performance evaluation are cumbersome

mlr3 provides an interface to several machine learning algorithms for *training*, *predicting*, *resampling*, *tuning*, *benchmarks* and more.

Example:

```
design = benchmark_grid(
  tasks = list(tsk("iris"), tsk("german_credit")),
  learners = list(lrn("classif.ranger"), lrn("classif.rpart")),
  resamplings = list(rsmp("cv"))
)
bmr = benchmark(design)
bmr$aggregate(msr("classif.acc"))[,
  .(task_id, learner_id, classif.acc)]

#>          task_id     learner_id classif.acc
#> 1:          iris classif.ranger   0.9600000
#> 2:          iris  classif.rpart   0.9466667
#> 3: german_credit classif.ranger   0.7720000
#> 4: german_credit  classif.rpart   0.7310000
```

MLR3 PHILOSOPHY

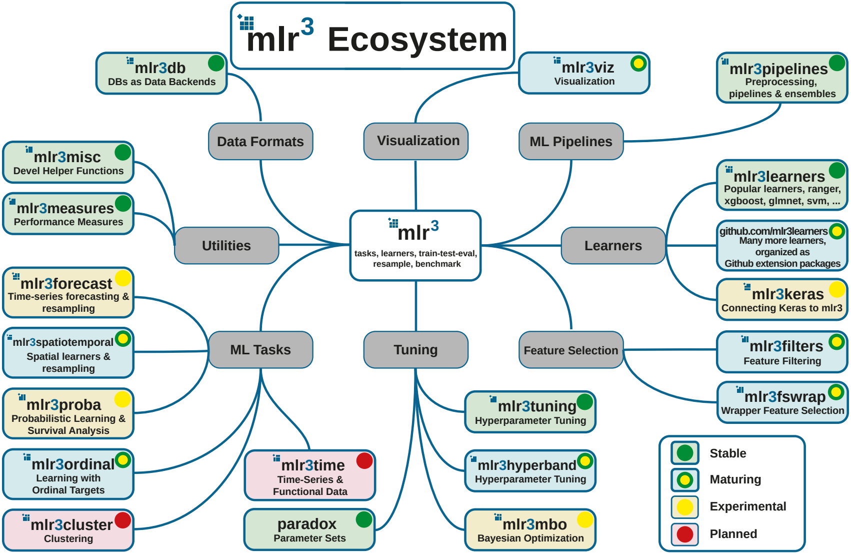
- Overcome limitations of S3 with the help of **R6**
  - Truly object-oriented: data and methods live in the same object
  - Make use of inheritance
  - Reference semantics

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
  - Reference semantics
- Embrace **data.table**, both for arguments and internally
  - Fast operations for tabular data
  - List columns to arrange complex objects in tabular structure

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
  - 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`, `Metrics`, `lgr`, `uuid`, `mlbench`, `digest`
  - Plus some of our own packages (`backports`, `checkmate`, ...)

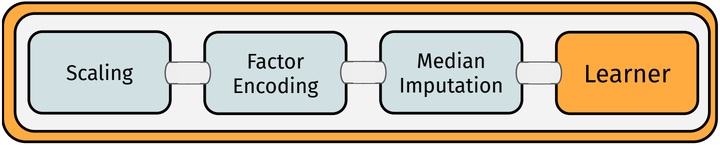

mlr3 Ecosystem
mlr3db
DBs as Data Backends
mlr3viz
Visualization
mlr3pipelines
Preprocessing, pipelines & ensembles
Data Formats
Visualization
ML Pipelines
mlr3misc
Devel Helper Functions
mlr3measures
Performance Measures
mlr3learners
Popular learners, ranger, xgboost, glmnet, svm, ...
Utilities
mlr3
tasks, learners, train-test-eval, resample, benchmark
Learners
github.com/mlr3learners
Many more learners, organized as Github extension packages
mlr3forecast
Time-series forecasting & resampling
mlr3keras
Connecting Keras to mlr3
mlr3spatiotemporal
Spatial learners & resampling
ML Tasks
Tuning
Feature Selection
mlr3filters
Feature Filtering
mlr3proba
Probabilistic Learning & Survival Analysis
mlr3tuning
Hyperparameter Tuning
mlr3fswrap
Wrapper Feature Selection
mlr3ordinal
Learning with Ordinal Targets
mlr3time
Time-Series & Functional Data
mlr3hyperband
Hyperparameter Tuning
Stable
Maturing
Experimental
Planned
mlr3cluster
Clustering
paradox
Parameter Sets
mlr3mbo
Bayesian Optimization

mlr3pipelines

MACHINE LEARNING WORKFLOWS

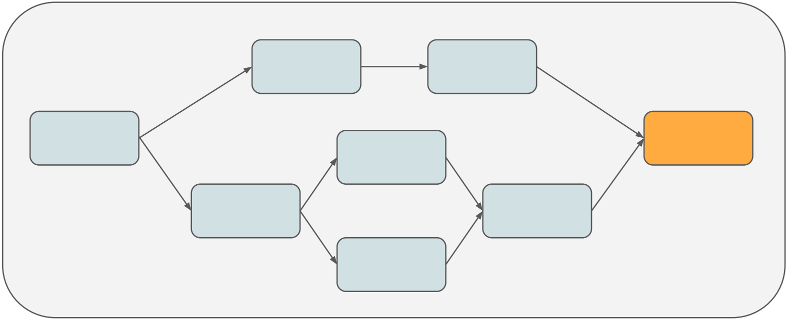
- **Preprocessing**: Feature extraction, feature selection, missing data imputation,. . .
- **Ensemble methods**: Model averaging, model stacking
- `mlr3`: modular model fitting

⇒ `mlr3pipelines`: modular ML workflows



MACHINE LEARNING WORKFLOWS

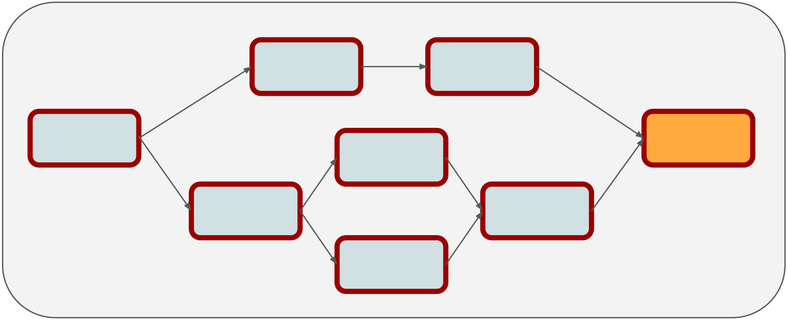
– what do they look like?



MACHINE LEARNING WORKFLOWS

– what do they look like?

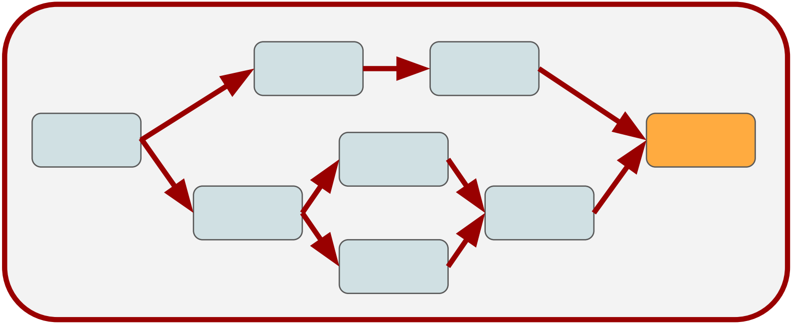
- **Building blocks**: *what* is happening? → `PipeOp`



MACHINE LEARNING WORKFLOWS

– what do they look like?

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

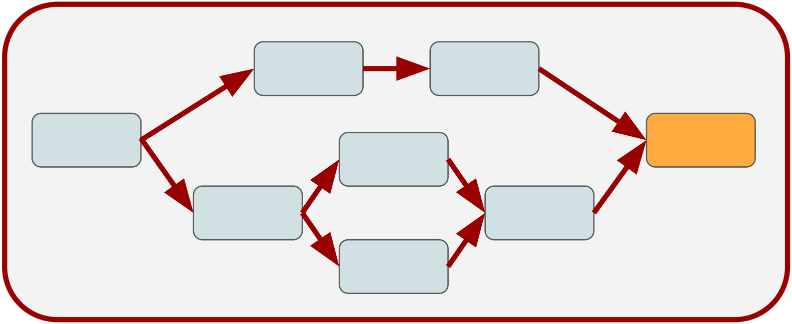


MACHINE LEARNING WORKFLOWS

– what do they look like?

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

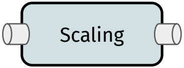
$\Rightarrow$ `Graph`: `PipeOps` as **nodes** with **edges** (data flow) between them



PipeOps

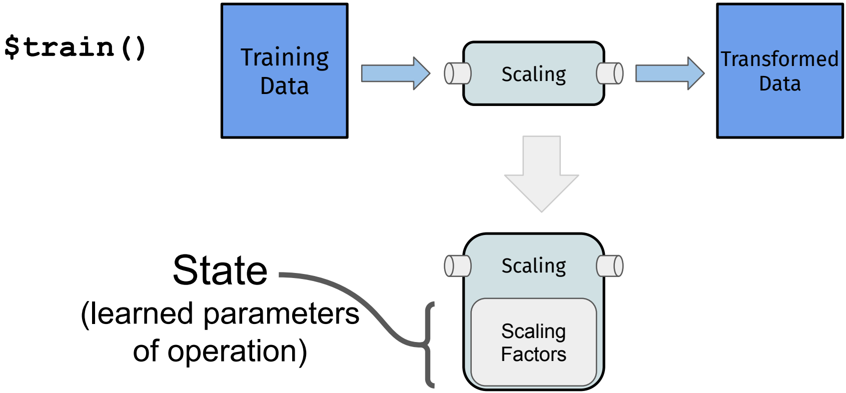
PIPEOP: SINGLE UNIT OF DATA OPERATION

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



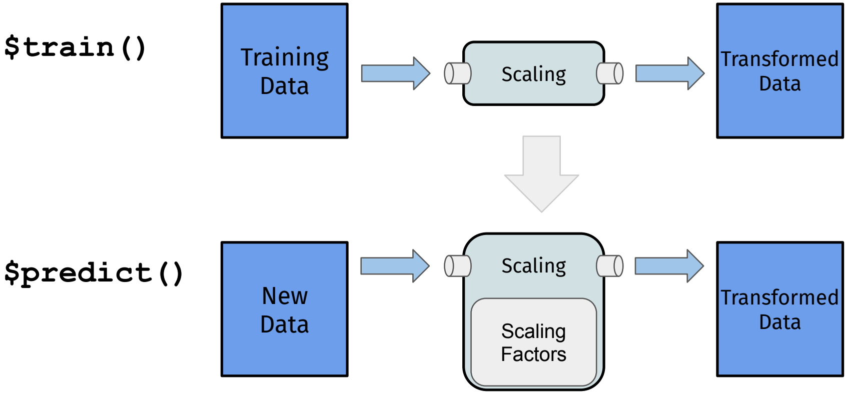
PIPEOP: SINGLE UNIT OF DATA OPERATION

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



PIPEOP: SINGLE UNIT OF DATA OPERATION

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



PIPEOP: SINGLE UNIT OF DATA OPERATION

```r
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
```

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    -1.335752   -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
```

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    -1.335752   -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
```

LIST OF PIPEOPS

Included

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

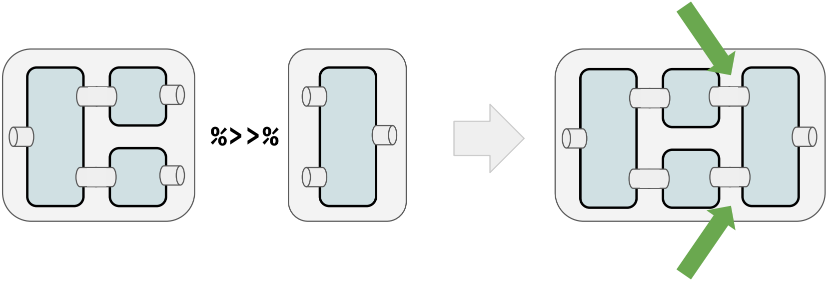
Planned

- Time series and spatio-temporal data
- Multi-output and ordinal targets

Graph Operations

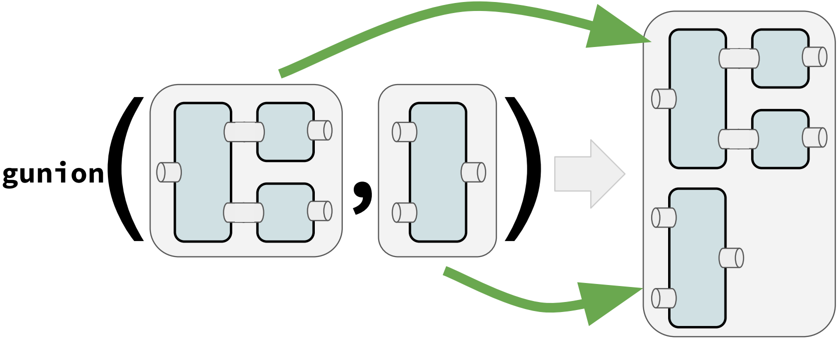
GRAPH OPERATIONS

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



GRAPH OPERATIONS

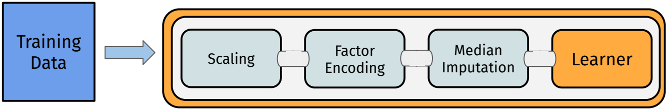
`gunion()` unites `Graphs` and `PipeOps`



Linear Pipelines

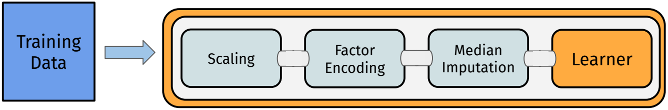
LINEAR PREPROCESSING

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



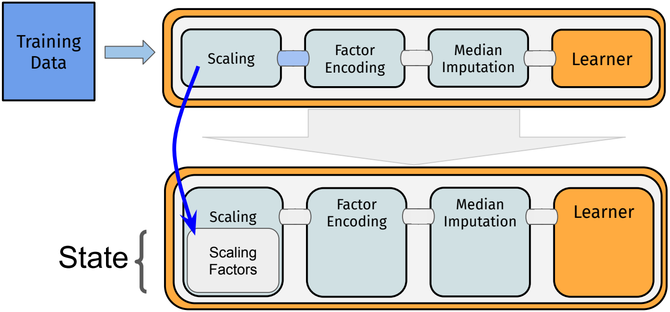
LINEAR PREPROCESSING

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



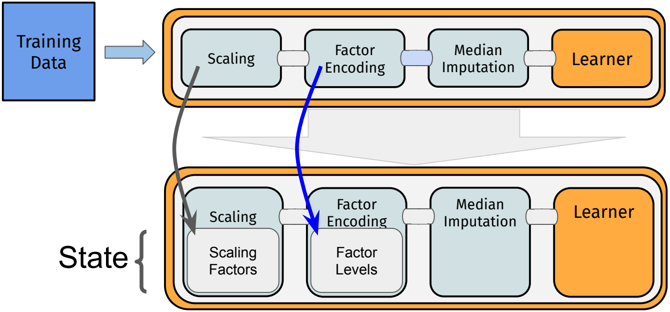
LINEAR PREPROCESSING

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



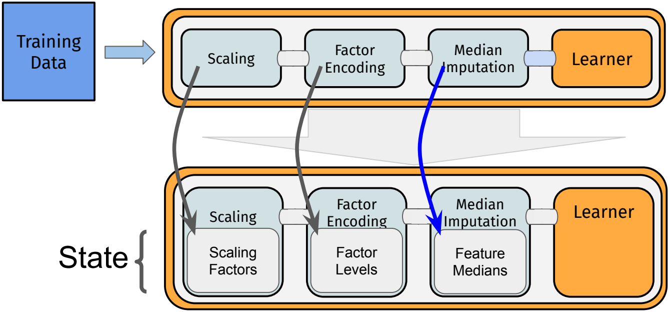
LINEAR PREPROCESSING

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



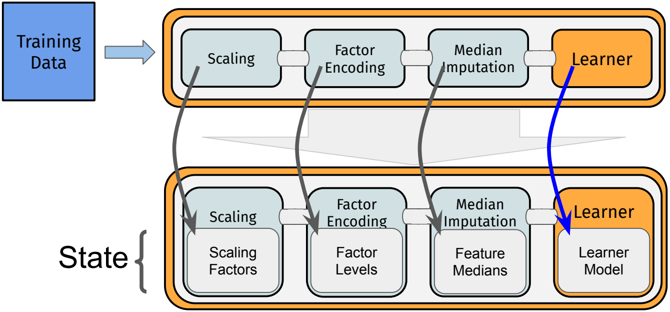
LINEAR PREPROCESSING

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



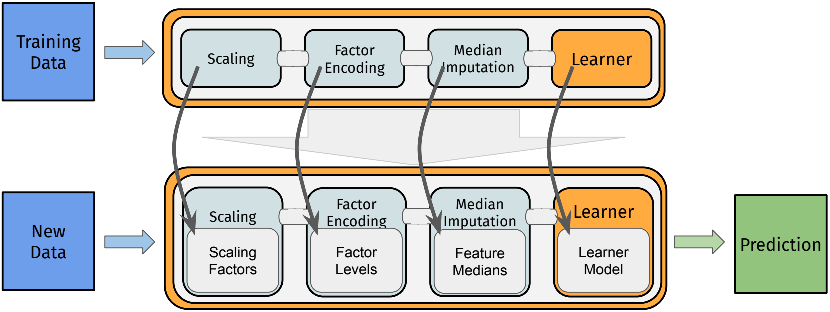
LINEAR PREPROCESSING

- `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
#>     4.163367     1.424451     5.921098     3.098387
```

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()`)

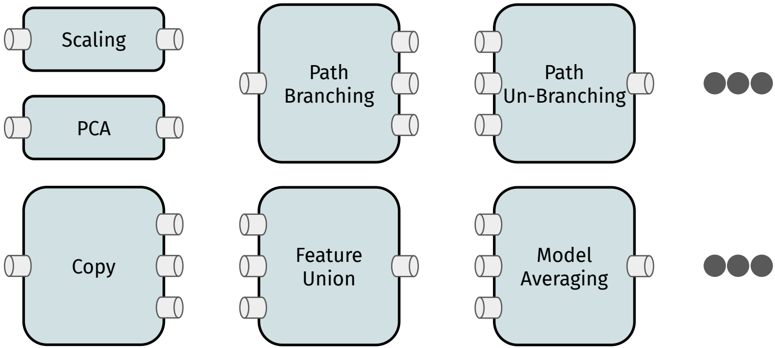
```
graph_pp$pipeops$scale$state$scale
#> Petal.Length  Petal.Width Sepal.Length  Sepal.Width
#>     4.163367     1.424451     5.921098     3.098387
```

- 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
```

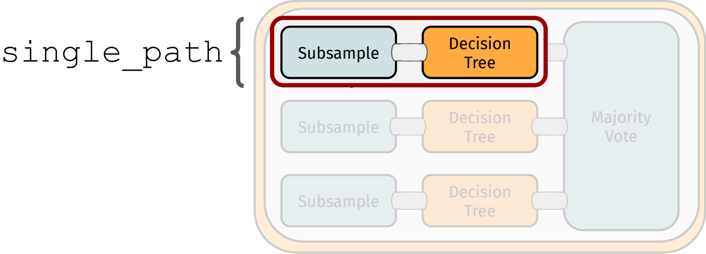
Nonlinear Pipelines

PIPEOPS WITH MULTIPLE INPUTS / OUTPUTS



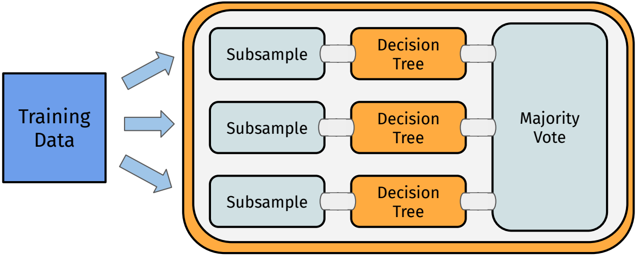
ENSEMBLE METHOD: BAGGING

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



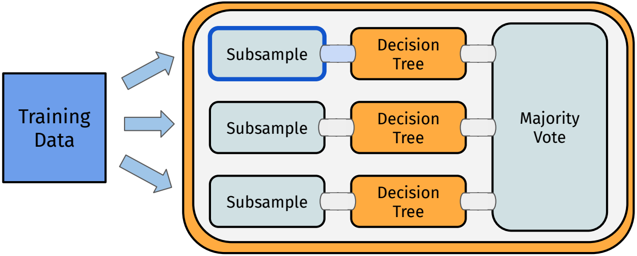
ENSEMBLE METHOD: BAGGING

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

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



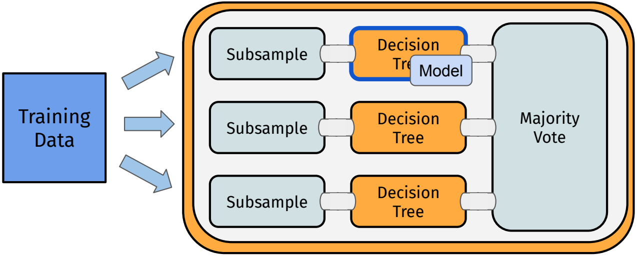
ENSEMBLE METHOD: BAGGING

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

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



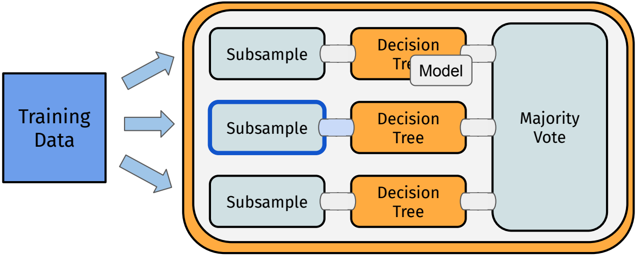
ENSEMBLE METHOD: BAGGING

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

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



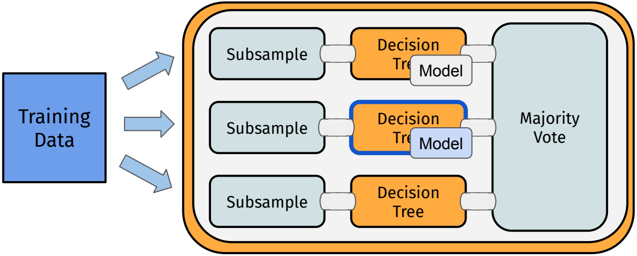
ENSEMBLE METHOD: BAGGING

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

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



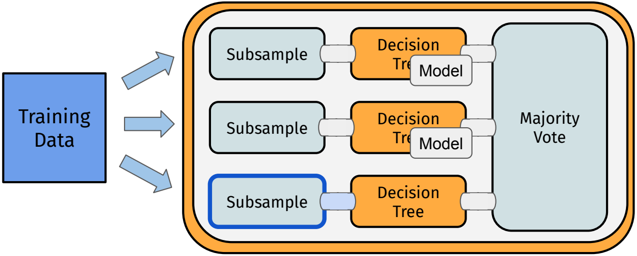
ENSEMBLE METHOD: BAGGING

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

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



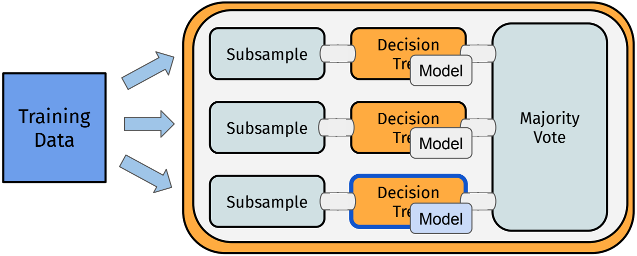
ENSEMBLE METHOD: BAGGING

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

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



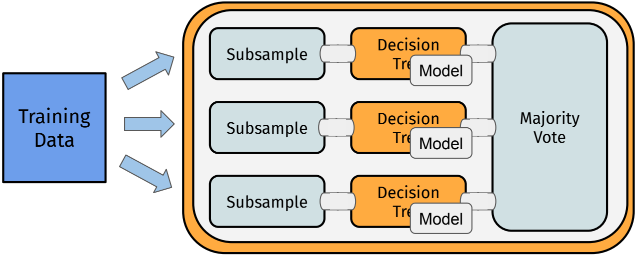
ENSEMBLE METHOD: BAGGING

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

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



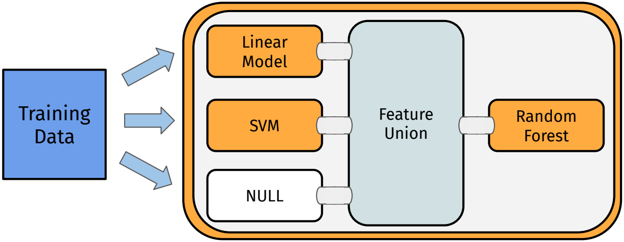
ENSEMBLE METHOD: BAGGING

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

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



ENSEMBLE METHOD: STACKING

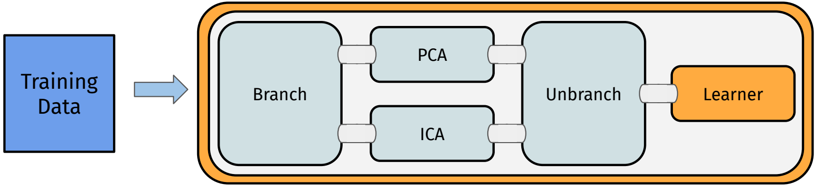
```
graph_stack = gunion(list(
      po("learner_cv", learner = lrn("regr.lm")),
      po("learner_cv", learner = lrn("regr.svm")),
      po("nop"))) %>>%
  po("featureunion") %>>%
  lrn("regr.ranger")
```



BRANCHING

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

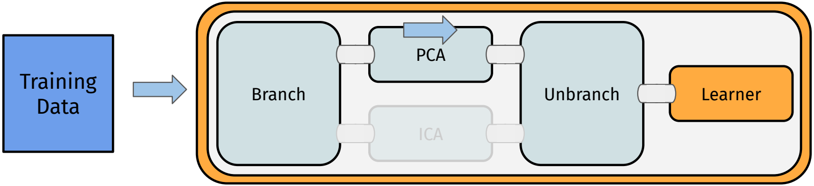
Execute only one of several alternative paths



BRANCHING

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

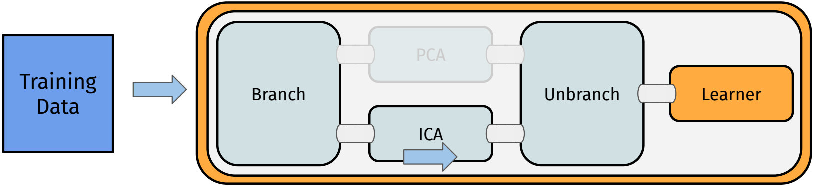
```
> graph_branch$pipeops$branch$
    param_set$values$selection = "pca"
```



BRANCHING

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

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

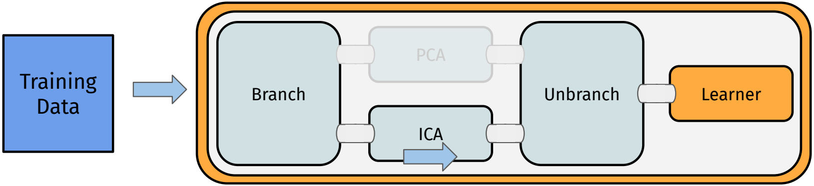


BRANCHING

Alternative:

```
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

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
```

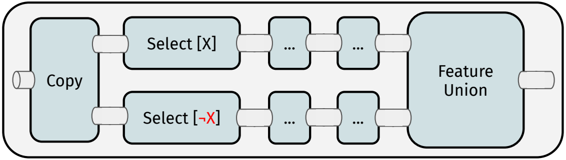
RESTRICT PIPEOPS TO COLS WITH SELECTORS

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 -0.7781478  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
```

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]]
```

Having trouble remembering these?

"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`

`ppl()` accesses the `mlr_graphs` “Dictionary” of pre-constructed partial `Graphs`.

```
head(as.data.table(mlr_graphs), 5)

#>             key
#> 1:      bagging
#> 2:       branch
#> 3:   greplicate
#> 4:    robustify
#> 5: targettrafo
```

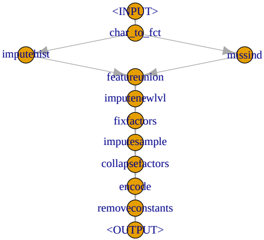
“PIPELINES” DICTIONARY & SHORT FORM

Many frequently used *patterns* for pipelines

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

Collection of these is in `mlr3pipelines`

```r
gr = ppl("robustify")
plot(gr)
```



“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`

```
gr = ppl("branch", list(po("pca"), po("nop")))
plot(gr)
```

AutoML with ‘mlr3pipelines‘

AUTOML <3 PIPELINES

- AutoML: Automatic Machine Learning

AUTOML <3 PIPELINES

- AutoML: Automatic Machine Learning
- Let the algorithm make decisions about
  1. *what learner to use*,
  2. *what preprocessing to use*, and
  3. *what hyperparameters* to use.

AUTOML <3 PIPELINES

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

AUTOML <3 PIPELINES

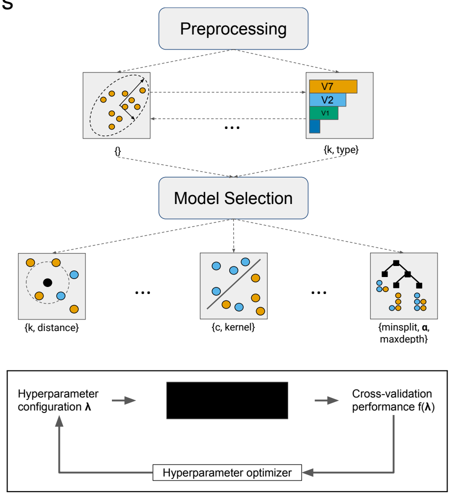
- AutoML: Automatic Machine Learning
- Let the algorithm make decisions about
  1. *what learner to use*,
  2. *what preprocessing to use*, and
  3. *what hyperparameters* to use.
- (1) and (2) are decisions about *graph structure* in `mlr3pipelines`

$\Rightarrow$ The problem reduces to **pipelines + parameter tuning**

AUTOML WITH MLR3PIPELINES

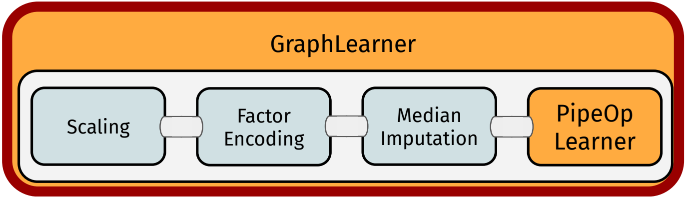
AutoML in a Nutshell

- Preprocessing steps
- ML Algorithms
- Tuner



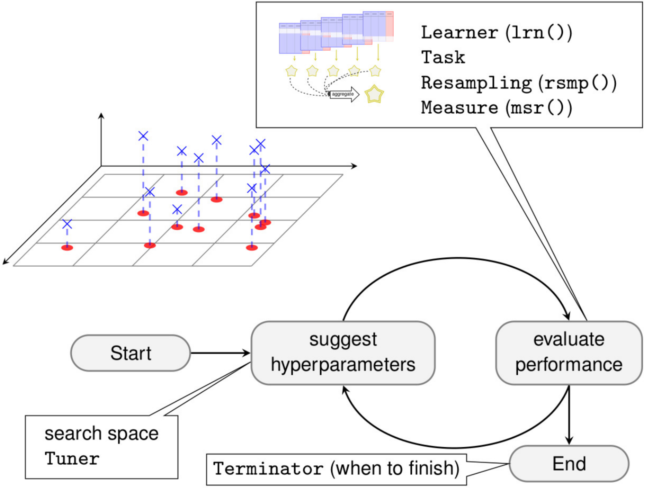
GRAPHLEARNER

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




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

TUNING



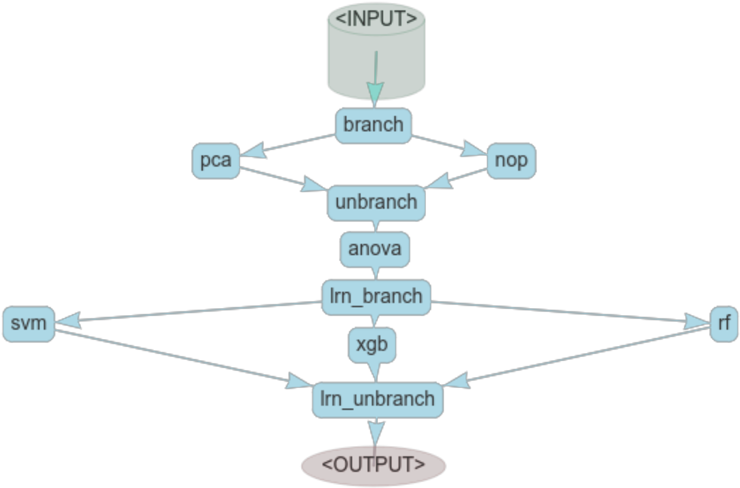
aggregate
Learner (lrn())
Task
Resampling (rsmp())
Measure (msr())
Start
suggest hyperparameters
evaluate performance
search space
Tuner
Terminator (when to finish)
End

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"),
  "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



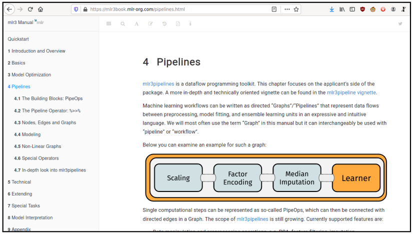
PIPELINES TUNING

```r
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 = TuningInstance$new(tsk("sonar"), glrn, rsmp("cv", iters=3),
  msr("classif.ce"), ps, term("evals", n_evals = 10))
tnr("random_search")$tune(inst)
```

mlr3(pipelines) Resources

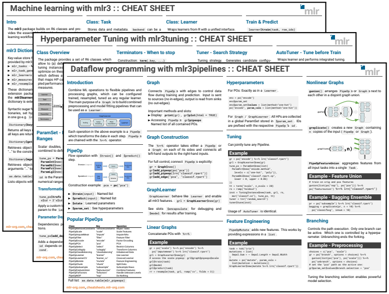
MLR3(PIPELINES) RESOURCES

mlr3 book



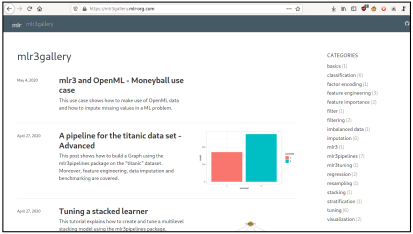
`https://mlr3book.mlr-org.com/`

“cheat sheets”



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

mlr3 Use Case “Gallery”



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

OUTLOOK

What is to come?

- `mlr3pipelines`: caching, parallelization
- Better **tuners**: Bayesian Optimization, Hyperband
- Survival and Forecasting (via `mlr3proba`, `mlr3forecast`)
- Deep Learning (via `mlr3keras`)

Thanks! Please ask questions!