MLR3PIELINES:
Machine Learning Pipelines as Graphs

Martin Binder, Florian Pfisterer, Michel Lang, Bernd Bischl
Computational Statistics, LMU
**MLR3PIPCLES**

**Machine Learning Workflows:**

- **Preprocessing:** Feature extraction, feature selection, missing data imputation,…
- **Ensemble methods:** Model averaging, model stacking
- **m1r3:** modular model fitting

⇒ **m1r3pipelines:** modular ML workflows

(replaces m1r2’s m1rCPO and most “wrappers”)
Machine Learning Workflows

– what do they look like?
Machine Learning Workflows

– what do they look like?

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

- `$\text{train}()$: process data and create `$\text{state}$`

```latex
\text{State} \\
(\text{learned parameters of operation})
```

### Diagram:

- **Training Data** → **Scaling** → **Transformed Data**
- **State** (learned parameters of operation)
**The Building Blocks**

**PipeOp: Single Unit of Data Operation**

- `$train()`: process data and create `$state`
- `$predict()`: process data depending on the `$state`

![Diagram showing `$train()` process: Training Data $\rightarrow$ Scaling $\rightarrow$ Transformed Data]

![Diagram showing `$predict()` process: New Data $\rightarrow$ Scaling $\rightarrow$ Transformed Data]
**The Building Blocks**

**PipeOp: Single Unit of Data Operation**

- `$\text{train()}$`: process data and create `$\text{state}$`
- `$\text{predict()}$`: process data depending on the `$\text{state}$`
- Multiple inputs or multiple outputs

![Diagram of PipeOp building blocks: Scaling, PCA, Copy, Path Branching, Feature Union, Model Averaging, Path Un-Branching](image)
PipeOps so far and planned

- Simple preprocessing operations: scale, pca, apply, mutate
- Missing value imputation: impute
- Feature selection and filtering: select, filter
- Categorical data encoding: encode
- Undersampling / subsampling: balancesample, subsample, chunk
- Learners: learner, learner_cv
- Ensemble methods on Predictions: majorityvote, modelavg
- Simultaneous and alternative branching: copy, branch, unbranch
- Combination of data: featureunion
- Backup prediction: backuplearner
- Text processing (planned)
- Time series and spatio-temporal data (planned)
- Multi-output and ordinal targets (planned)
- Outlier detection (planned)
- Hurdle models (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

\[
\text{greplicate}(\text{graph}, N)
\]
**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**

```r
graph_pp = mlr_pipeops$get("scale") %>>%
  mlr_pipeops$get("encode") %>>%
  mlr_pipeops$get("impute") %>>%
  mlr_pipeops$get("learner",
    learner = mlr_learners$get("classif.rpart"))
```

![Pipeline Diagram]

- Scaling
- Factor Encoding
- Median Imputation
- Learner
**mlr3pipelines in Action**

**Linear Preprocessing Pipeline**

```r
graph_pp = "scale" %>>% "encode" %>>% "impute" %>>% mlr_pipeops$get("learner",
  learner = mlr_learners$get("classif.rpart"))
```

![Pipeline Diagram]

[Diagram showing the pipeline with stages for scaling, encoding, imputation, and learner]
**mlr3pipelines in Action**

**Linear Preprocessing Pipeline**
- **train()ing**: Data propagates and creates $\textit{states}$

```r
glrn = GraphLearner$new(graph_pp)
glrn$train(task)
```
mlr3pipelines in Action

Linear Preprocessing Pipeline

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

```r
grln = GraphLearner$new(graph_pp)
grln$train(task)
```
**mlr3pipelines in Action**

**Linear Preprocessing Pipeline**

- *train()*ing: Data propagates and creates $\text{states}$

```r
glrn = GraphLearner$new(graph_pp)
glrn$train(task)
```
**mlr3pipelines in Action**

**Linear Preprocessing Pipeline**
- `train()`ing: Data propagates and creates `states`

```r
glrn = GraphLearner$new(graph_pp)
glrn$train(task)
```
**mlr3pipelines** in **Action**

**Linear Preprocessing Pipeline**
- `train()`ing: Data propagates and creates `$states$

```r
glrn = GraphLearner$new(graph_pp)
glrn$train(task)
```
**mlr3pipelines in Action**

**Linear Preprocessing Pipeline**

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

\`glrn$\text{predict}(\text{task})\`
**mlr3pipelines in Action**

**Linear Preprocessing Pipeline**

- Setting / retrieving parameters: `$param_set`

```r
graph_pp$pipeops$impute$param_set$values$method_num = "mean"
```

Retrieving state:
```
$state
```
of individual Pipelops (after `$train()`)
```
graph_pp$pipeops$scale$state %>%
head(1)
```

Retrieving intermediate results:
```
$.result (set debug option before)
```
```
graph_pp$pipeops$scale$.result[[1]]$data () %>%
head(3)
```

## $center

<table>
<thead>
<tr>
<th>Petal.Length</th>
<th>Petal.Width</th>
<th>Sepal.Length</th>
<th>Sepal.Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.758000</td>
<td>1.199333</td>
<td>5.843333</td>
<td>3.057333</td>
</tr>
</tbody>
</table>

## Species

<table>
<thead>
<tr>
<th>Petal.Length</th>
<th>Petal.Width</th>
<th>Sepal.Length</th>
<th>Sepal.Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.335752</td>
<td>-1.311052</td>
<td>-0.8976739</td>
<td>1.0156020</td>
</tr>
<tr>
<td>-1.335752</td>
<td>-1.311052</td>
<td>-1.1392005</td>
<td>-0.1315388</td>
</tr>
<tr>
<td>-1.392399</td>
<td>-1.311052</td>
<td>-1.3807271</td>
<td>0.3273175</td>
</tr>
</tbody>
</table>
**mlr3pipelines in Action**

**Linear Preprocessing PipeLine**

- Setting / retrieving parameters: `$param_set`
  
  ```r
  graph_pp$pipeops$impute$param_set$values$method_num = "mean"
  ```

- Retrieving state: `$state` of individual PipeOps (*after* `$train()`)
  
  ```r
  graph_pp$pipeops$scale$state %>% head(1)
  ```
  ```r
  ## $center
  ## Petal.Length  Petal.Width  Sepal.Length  Sepal.Width
  ##    3.758000    1.199333   5.843333   3.057333
  ```
**mlr3pipelines in Action**

**Linear Preprocessing Pipeline**

- Setting / retrieving parameters: `$param_set`
  
  ```r
  graph_pp$pipeops$impute$param_set$values$method_num = "mean"
  ```

- Retrieving state: `$state` of individual PipeOps *(after $train()*)
  
  ```r
  graph_pp$pipeops$scale$state %>% head(1)
  ```

  ```r
  #> $center
  #>    Petal.Length Petal.Width Sepal.Length Sepal.Width
  #> 1  3.758000   1.199333    5.843333    3.057333
  ```

- Retrieving intermediate results: `.result` *(set debug option before)*
  
  ```r
  graph_pp$pipeops$scale$.result[[1]][[1]]$data() %>% head(3)
  ```

  ```r
  #>             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
  ```
Nonlinear Pipelines
**mlr3pipelines in Action**

**Ensemble Method: Bagging**

```r
single_path = "subsample" %>>% 
mlr_pipeops$get("learner", 
  learner = mlr_learners$get("classif.rpart"))
```

```
single_path { Subsample Decision Tree Subsample Decision Tree Subsample Decision Tree Subsample Decision Tree }
```
**mlr3pipelines in Action**

**Ensemble Method: Bagging**

```r
single_path = "subsample" %>>%
mlr_pipeops$\texttt{get("learner",}
\hspace{1em} learner = mlr_learners$\texttt{get("classif.rpart")})

graph_bag = \texttt{greplicate\texttt{(single_path, }n = 3\texttt{) \%>>%}}
mlr_pipeops$\texttt{get("majorityvote", }\texttt{innum} = 3\texttt{)}
```
mlr3pipelines in Action

Ensemble Method: Bagging

```
single_path = "subsample" %>>%
mlr_pipeops$\text{get}("learner",
    learner = mlr_learners$\text{get}("classif.rpart"))

graph_bag = \text{greplicate}(\text{single_path}, \text{n} = 3) %>>%
mlr_pipeops$\text{get}("majorityvote", innum = 3)
```
mlr3pipelines in Action

Ensemble Method: Bagging

define single_path = "subsample" %>%
mlr_pipeops$%>% get("learner",
  learner = mlr_learners$get("classif.rpart"))

define graph_bag = greplicate(single_path, n = 3) %>%
mlr_pipeops$%>% get("majorityvote", innum = 3)
mlr3pipelines in Action

Ensemble Method: Bagging

```r
single_path = "subsample" %>>% 
  mlr_pipeops$get("learner",
    learner = mlr_learners$get("classif.rpart"))

graph_bag = greplicate(single_path, n = 3) %>>% 
  mlr_pipeops$get("majorityvote", innum = 3)
```
**mlr3pipelines in Action**

**Ensemble Method: Bagging**

```r
single_path = "subsample" %>>%
mlr_pipeops$get("learner",
    learner = mlr_learners$get("classif.rpart"))

graph_bag = greplicate(single_path, n = 3) %>>%
mlr_pipeops$get("majorityvote", innum = 3)
```
mlr3pipelines in Action

Ensemble Method: Bagging

```
single_path = "subsample" %>>%  
mlr_pipeops$get("learner",  
    learner = mlr_learners$get("classif.rpart"))

graph_bag = greplicate(single_path, n = 3) %>>%  
mlr_pipeops$get("majorityvote", innum = 3)
```
mlr3pipelines in Action

Ensemble Method: Bagging

```r
single_path = "subsample" %>%
  mlr_pipeops$get("learner",
    learner = mlr_learners$get("classif.rpart"))

graph_bag = greplicate(single_path, n = 3) %>%
  mlr_pipeops$get("majorityvote", innum = 3)
```
mlr3pipelines in Action

Ensemble Method: Bagging

```r
single_path = "subsample" %>>%
mlr_pipeops$get("learner",
    learner = mlr_learners$get("classif.rpart"))

df <- df %>%
  mlr_pipeops$get("majorityvote", innum = 3)
```

Decision Tree
---
Subsample
---
Decision Tree
---
Subsample
---
Majority Vote
---
Model
---
Model
---
Model

Training Data
---
**mlr3pipelines in Action**

**Ensemble Method: Stacking**

```r
graph_stack = gunion(
  list(
    mlr_pipeops$<get>("learner_cv",
      learner = mlr_learners$<get>("regr.lm")),
    mlr_pipeops$<get>("learner_cv",
      learner = mlr_learners$<get>("regr.svm")),
    "null")) %>>%
mlr_pipeops$<get>("featureunion", innum = 3) %>>%
mlr_pipeops$<get>("learner",
  learner = mlr_learners$<get>("regr.ranger"))
```

[Diagram showing the ensemble method: Training Data, Linear Model, SVM, NULL, Feature Union, Random Forest]
mlr3pipelines in Action

Branching

```r
graph_branch = mlr_pipeops$get("branch", c("pca", "ica")) %>>%
gunion(list("pca", "ica")) %>>%
mlr_pipeops$get("unbranch", c("pca", "ica")) %>>%
mlr_pipeops$get("learner",
  learner = mlr_learners$get("classif.kknn"))
```

Execute only one of several alternative paths
mlr3pipelines in Action

Branching

```r
graph_branch = mlr_pipeops$\text{get}("branch", c("pca", "ica")) \%>>\%
gunion(list("pca", "ica")) \%>>\%
mlr_pipeops$\text{get}("unbranch", c("pca", "ica")) \%>>\%
mlr_pipeops$\text{get}("learner",
  learner = mlr_learners$\text{get}("classif.kknn"))

> graph_branch$pipeops$branch$
  param_set$values$selection = "pca"
```
**mldr3pipelines in Action**

**Branching**

```r
graph_branch = mlr_pipeops$get("branch", c("pca", "ica")) %>>% gunion(list("pca", "ica")) %>>% mlr_pipeops$get("unbranch", c("pca", "ica")) %>>% mlr_pipeops$get("learner", 
  learner = mlr_learners$get("classif.kknn"))
```

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

```r
library("paradox") ; library("mlr3tuning")

glrn = "scale" %>% mlr_pipeops$get("learner",
  mlr_learners$get("classif.rpart"))

ps = ParamSet$new(list(
  ParamLgl$new("scale.scale"),
  ParamInt$new("classif.rpart.minsplit", 1, 20))
)

ff = PerformanceEvaluator$new(task, glrn, "cv", "classif.ce", ps)
tuner = TunerRandomSearch$new(ff, TerminatorEvaluations$new(10))

tuner$tune()

tuner$tune_result()
```
NOT SHOWN HERE:
- Many more PipeOps: select, apply, encode, ...
- Automatic type-checking when constructing Graphs
- Interactive (html + javascript) plots
- Extensible by R6 inheritance of PipeOp base class

UPCOMING FEATURES
- More PipeOps
- Caching of expensive results
- Automatically parallel execution of concurrent operations

Thanks! Questions? Comments? Comment on Github?

mlr3: https://github.com/mlr-org/mlr3
mlr3pipelines: https://github.com/mlr-org/mlr3pipelines