Introduction

Combine ML operations to flexible pipelines and processing graphs, which can be configured trained, resampled, tuned as any regular learner. The main purpose of a graph is to build combined preprocessing and model fitting pipelines that can be used as a Learner.

PipeOp

Flow operation with $\text{train}()$ and $\text{predict}()$ step.

- $\text{train(input)}$: Training Data
- $\text{predict(input)}$: New Data

Construction example: pca = po("pca")

<table>
<thead>
<tr>
<th>Populer PipeOps</th>
<th>Key</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>PipeOpRemoveConstants</td>
<td>&quot;removeconstants&quot;</td>
<td>Removes constants</td>
</tr>
<tr>
<td>PipeOpScale</td>
<td>&quot;scale&quot;</td>
<td>Scale Features</td>
</tr>
<tr>
<td>PipeOpImputeMean</td>
<td>&quot;impute&quot;</td>
<td>Impute NAs</td>
</tr>
<tr>
<td>PipeOpFilter</td>
<td>&quot;filter&quot;</td>
<td>Feature Filter</td>
</tr>
<tr>
<td>PipeOpEncode</td>
<td>&quot;encode&quot;</td>
<td>Factor Encoding</td>
</tr>
<tr>
<td>PipeOpPCA</td>
<td>&quot;pca&quot;</td>
<td>PCA</td>
</tr>
<tr>
<td>PipeOpSelect</td>
<td>&quot;select&quot;</td>
<td>Select Columns</td>
</tr>
<tr>
<td>PipeOpCApply</td>
<td>&quot;capply&quot;</td>
<td>Transform Columns</td>
</tr>
<tr>
<td>PipeOpClassBalancing</td>
<td>&quot;classbalancing&quot;</td>
<td>Imbalanced Data</td>
</tr>
<tr>
<td>PipeOpLearner</td>
<td>&quot;learner&quot;</td>
<td>Use Learner</td>
</tr>
<tr>
<td>PipeOpLearnerCV</td>
<td>&quot;learner_cv&quot;</td>
<td>Crossval Learner</td>
</tr>
<tr>
<td>PipeOpMutate</td>
<td>&quot;mutate&quot;</td>
<td>Feature Engineering</td>
</tr>
<tr>
<td>PipeOpChunk</td>
<td>&quot;chunk&quot;</td>
<td>Split Data</td>
</tr>
<tr>
<td>PipeOpSubsample</td>
<td>&quot;subsample&quot;</td>
<td>Subsample Rows</td>
</tr>
<tr>
<td>PipeOpFeatureUnion</td>
<td>&quot;featureunion&quot;</td>
<td>Combine Features</td>
</tr>
<tr>
<td>PipeOpFixFactors</td>
<td>&quot;fixfactors&quot;</td>
<td>Handle Unknown Levels</td>
</tr>
<tr>
<td>PipeOpNOP</td>
<td>&quot;nop&quot;</td>
<td>Do Nothing</td>
</tr>
</tbody>
</table>

Full list: as.data.table(mlr_pipeops)

Graph

Connects PipeOps with edges to control data flow during training and prediction. Input is sent to sources (no in-edges), output is read from sinks (no out-edges).

Important methods and slots:

- $\text{Display print}(g)$
- $\text{grplot}(html = \text{TRUE})$
- $\text{Accessing PipeOps: } g@pipeops$
- $\text{Named list of all contained POs.}$

Graph Construction

The $\text{<>}$ operator takes either a PipeOp or a Graph on each of its sides and connects all left-hand outputs to the right-hand inputs.

For full control, connect PipeOps explicitly:

```
gr = Graph$new()
g$add_pipeop(po("pca"))
g$add_edge("pca", "classif.rpart")
```

Linear Graphs

Concatenate POs with $\text{<>}$:

```
gr = Graph$new()
gr$add_pipeop(po("encode"))
gr$add_pipeop(po("classif.rpart"))
gr$add_edge("pca", "classif.rpart")
```

Hyperparameters

For POs: Exactly as in a Learner.

```
enc = po("encode")
enc$param_set = list(method = "one-hot")
```

Graph / GraphLearner: All HPs are collected in a global ParamSet stored in $\text{graph_set}$. IDs are prefixed with the respective PipeOp id.

Tuning

Can jointly tune any Pipeline.

```
enc = po("encode")
enc$param_set = list(method = "one-hot")
```

Feature Engineering

PipeOpMutate adds new features. This works by providing expressions in a List.

```
mutate = po("mutate")
```

Branching

Controls the path execution. Only one branch can be active. Which one is controlled by a hyperparameter. Unbranching ends the forking.

```
branch = po("branch")
```

PipeOpFeatureUnion aggregates features from all input tasks into a single Task.