Applied Machine Learning

MLR3 Pipelines:
Part 2

Learning goals

- Targeting Columns
- Complex ML Pipelines
- AutoML Concepts

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
#>      <fctr>        <num>       <num>        <num>       <num>
#> 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
#>     <fctr> <num>  <num>        <num>       <num>
#> 1:  setosa -0.78  0.378          1.4         0.2
#> 2:  setosa -0.94 -0.137          1.4         0.2
#> 3:  setosa -1.15  0.045          1.3         0.2
```

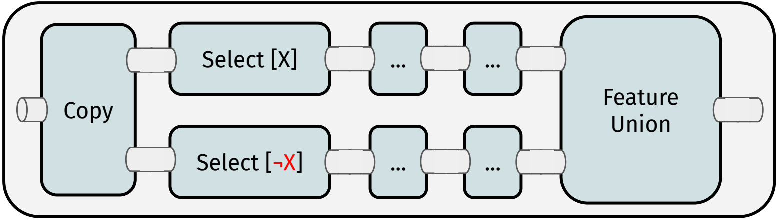
TARGETING COLUMNS

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
#>      <fctr> <num>  <num>        <num>       <num>
#> 1:   setosa -0.78  0.378          1.4         0.2
#> 2:   setosa -0.94 -0.137          1.4         0.2
#> 3:   setosa -1.15  0.045          1.3         0.2
```

Complex ML Pipelines and AutoML

AUTOML <3 PIPELINES

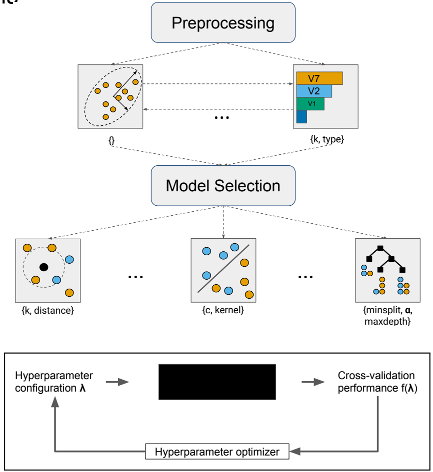
- AutoML: Automatic Machine Learning
- Let the algorithm make decisions about
  1. *what learner* to use,
  2. *what preprocessing* to use, and
  3. *what hyperparameters* to use.
- (1) and (2) are decisions about *graph structure* in `mlr3pipelines`

⇒ The problem reduces to **pipelines + parameter tuning**

AUTOML WITH MLR3PIPELINES

AutoML in a Nutshell

- Preprocessing steps
- ML Algorithms
- Tuner

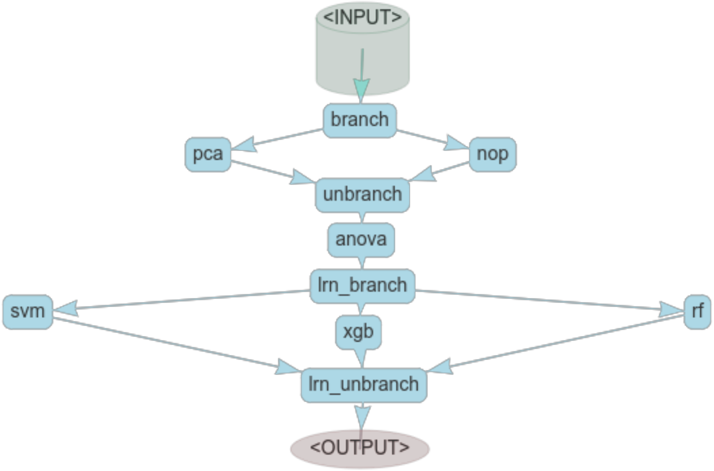


PIPELINES TUNING

- Works **exactly** as in basic `mlr3` / `mlr3tuning`
- `PipeOps` have *hyperparameters* (using `paradox` pkg)
- `Graphs` have hyperparameters of all components *combined*
- $\Rightarrow$ 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 = as_learner(gr)
```

PIPELINES TUNING



PIPELINES TUNING

```r
ps = ps(
  branch.selection = p_fct(levels = c("pca", "nothing")),
  anova.filter.frac = p_dbl(lower = 0.1, upper = 1),
  lrn_branch.selection = p_fct(levels = c("svm", "xgb", "rf")),
  rf.mtry.ratio = p_int(lower = 1L, upper = 20L, trafo = function(x) 1/x,
    depends = lrn_branch.selection == "rf" ),
  xgb.nrounds = p_int(lower = 1, upper = 500,
    depends = lrn_branch.selection == "xgb"),
  svm.cost = p_dbl(lower = -12, upper = 4, trafo = function(x) 2^x,
    depends = lrn_branch.selection == "svm"),
  svm.gamma = p_dbl(lower = -12, upper = -1, trafo = function(x) 2^x,
    depends = lrn_branch.selection == "svm"))

inst = ti(task = tsk("sonar"), learner = glrn,
  resampling = rsmp("cv", folds = 3), measures =  msr("classif.ce"),
  terminator = trm("evals", n_evals = 10), search_space = ps)

gsearch = tnr("random_search")
gsearch$optimize(inst)
```

Summary

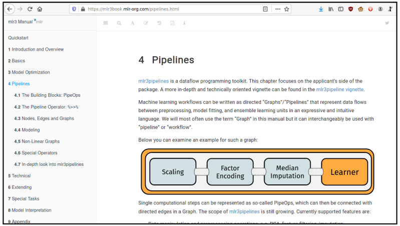
MLR3PIPELINES

`mlr3pipelines` overview:

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

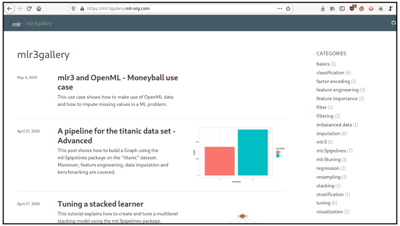
MLR3(PIPELINES) RESOURCES

mlr3 book



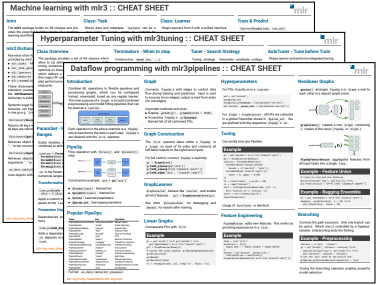
https://mlr3book.mlr-org.com/

mlr3 Use Case "Gallery"



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

"cheat sheets"



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