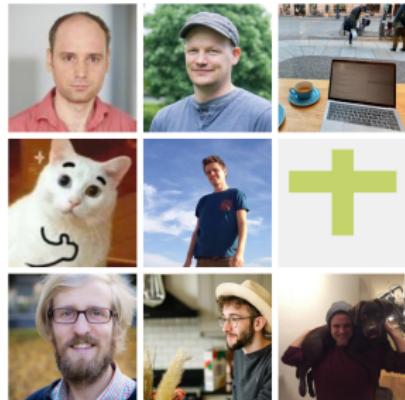


Modern Machine Learning in R



<https://mlr-org.com/>

<https://github.com/mlr-org>



**Bernd Bischl, Michel Lang, Martin Binder, Florian Pfisterer, Jakob Richter,
Patrick Schratz, Lennart Schneider, Raphael Sonabend, Marc Becker**

December 11, 2020

Intro

SO YOU WANT TO DO ML IN R

- R gives you access to many machine learning methods

SO YOU WANT TO DO ML IN R

- R gives you access to many machine learning methods
- ... but without a unified interface

SO YOU WANT TO DO ML IN R

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

Example:

```
# Specify what we want to model in a formula: target ~ features
svm_model = e1071::svm(Species ~ ., data = iris)
```

SO YOU WANT TO DO ML IN R

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

Example:

```
# Specify what we want to model in a formula: target ~ features
svm_model = e1071::svm(Species ~ ., data = iris)
```

vs.

```
# Pass the features as a matrix and the target as a vector  
xgb_model = xgboost::xgboost(data = as.matrix(iris[1:4]),  
    label = iris$Species, nrounds = 10)
```

SO YOU WANT TO DO ML IN R

```
library("mlr3")
```

Ingredients:

- Data / Task
- Learning Algorithms
- Performance Evaluation
- Performance Comparison

MLR3 PHILOSOPHY

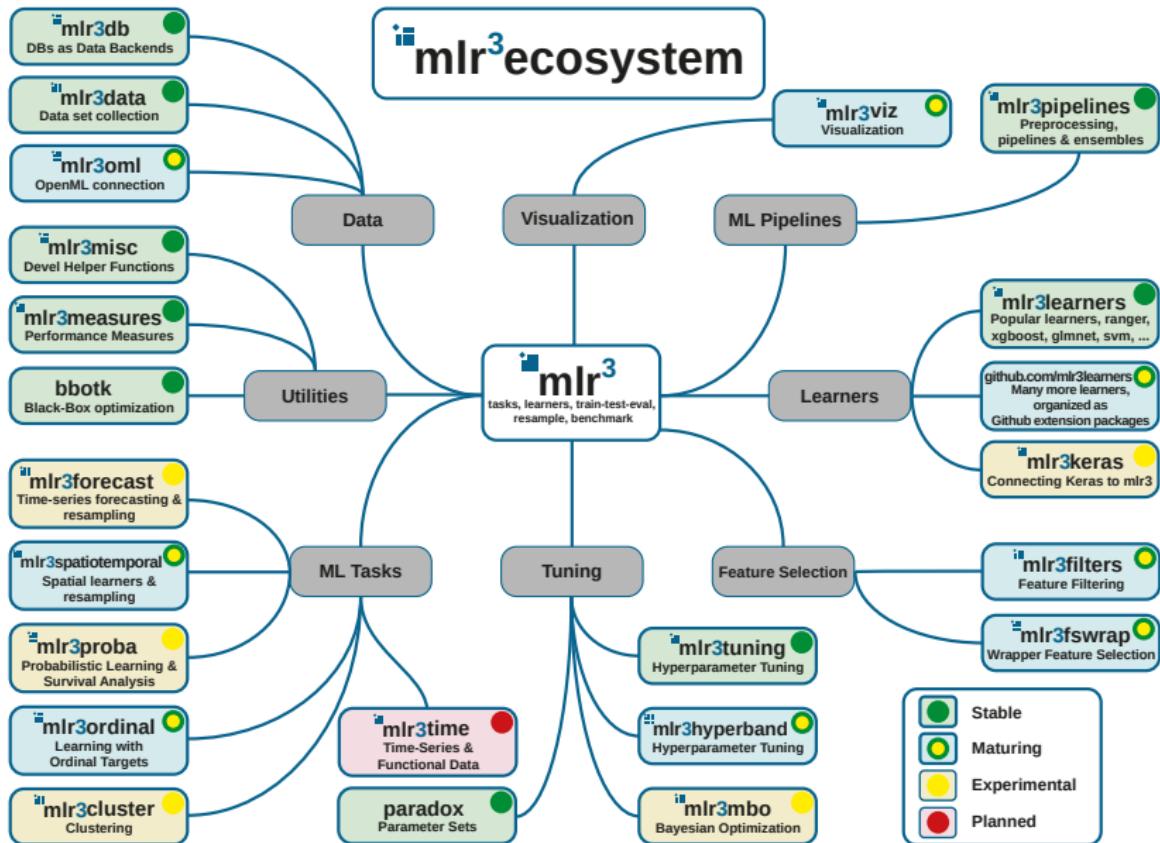
- Overcome limitations of S3 with the help of **R6**
 - Truly object-oriented: data and methods live in the same object
 - Make use of inheritance
 - Reference semantics

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



Control of Execution

CONTROL OF EXECUTION

Parallelization

```
future::plan("multicore")
```

- runs each resampling iteration as a job
- also allows nested resampling (although not needed here)

Encapsulation

```
learner$encapsulate = c(train = "callr", predict = "callr")
```

- Spawns a separate R process to train the learner
- Learner may segfault without tearing down the session
- Logs are captured
- Possibility to have a fallback to create predictions

Dictionaries

DICTIONARIES

- Ordinary constructors: `TaskClassif$new()` /
`LearnerClassifRpart$new()`

DICTIONARIES

- Ordinary constructors: `TaskClassif$new()` /
`LearnerClassifRpart$new()`
- ⇒ `mlr3` offers *Short Form Constructors* that are less verbose

DICTIONARIES

- Ordinary constructors: `TaskClassif$new()` /
`LearnerClassifRpart$new()`
- ⇒ `mlr3` offers *Short Form Constructors* that are less verbose
- They access Dictionary of objects:

DICTIONARIES

- Ordinary constructors: `TaskClassif$new()` /
`LearnerClassifRpart$new()`
- ⇒ `mlr3` offers *Short Form Constructors* that are less verbose
- They access Dictionary of objects:

Object	Dictionary	Short Form
Task	<code>mlr_tasks</code>	<code>tsk()</code>
Learner	<code>mlr_learners</code>	<code>lrn()</code>
Measure	<code>mlr_measures</code>	<code>msr()</code>
Resampling	<code>mlr_resamplings</code>	<code>rsmp()</code>

Dictionaries can get populated by add-on packages (e.g. `mlr3learners`)

DICTIONARIES

```
# list items
tsk()

#> <DictionaryTask> with 10 stored values
#> Keys: boston_housing, breast_cancer, german_credit, iris,
#> mtcars, pima, sonar, spam, wine, zoo

# retrieve object
tsk("iris")

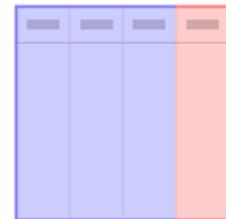
#> <TaskClassif:iris> (150 x 5)
#> * Target: Species
#> * Properties: multiclass
#> * Features (4):
#>   - dbl (4): Petal.Length, Petal.Width, Sepal.Length,
#>     Sepal.Width
```

Data

DATA

- Tabular data
- Features
- Target / outcome to predict
 - discrete for classification
 - continuous for regression

⇒ target determines the machine learning “Task”



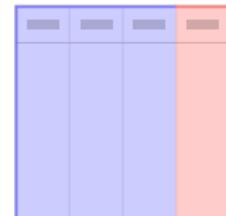
```
print(iris) # included in R

#>   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#> 1         5.1        3.5       1.4        0.2  setosa
#> 2         4.9        3.0       1.4        0.2  setosa
#> ...
```

DATA

- Tabular data
- Features
- Target / outcome to predict
 - discrete for classification
 - continuous for regression

⇒ target determines the machine learning “Task”



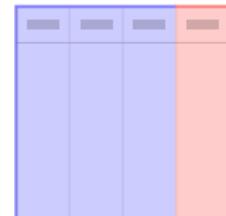
```
print(iris) # included in R
```

```
#> Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#> 1          5.1        3.5       1.4        0.2 setosa
#> 2          4.9        3.0       1.4        0.2 setosa
#> ...
```

DATA

- Tabular data
- Features
- Target / outcome to predict
 - discrete for classification
 - continuous for regression

⇒ target determines the machine learning “Task”



```
print(iris) # included in R
```

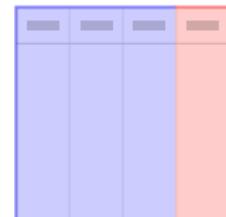
```
#> Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#> 1          5.1        3.5       1.4        0.2 setosa
#> 2          4.9        3.0       1.4        0.2 setosa
#> ...
```

```
task = TaskClassif$new("iris", iris, "Species")
```

DATA

- Tabular data
- Features
- Target / outcome to predict
 - discrete for classification
 - continuous for regression

⇒ target determines the machine learning “Task”



```
print(iris) # included in R
```

```
#> Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#> 1          5.1        3.5       1.4        0.2 setosa
#> 2          4.9        3.0       1.4        0.2 setosa
#> ...
```

Task ID data target name
↓ ↓ ↓

```
task = TaskClassif$new("iris", iris, "Species")
```

DATA

```
task = TaskClassif$new("iris", iris, "Species")
```

```
print(task)
```

```
# <TaskClassif:iris> (150 x 5)
# * Target: Species
# * Properties: multiclass
# * Features (4):
#   - dbl (4): Petal.Length, Petal.Width, Sepal.Length,
#     Sepal.Width
```

```
task$ncol
task$nrow
task$feature_names
task$target_names
```

```
task$head(n = )
task$truth(row_ids = )
task$data(rows = ,
          cols = )
```

```
task$select(cols = )
task$filter(rows = )
task$cbind(data = )
task$rbind(data = )
```

Learning Algorithms

LEARNING ALGORITHMS

- Get a Learner provided by `mlr`

```
learner = lrn("classif.rpart")
```

LEARNING ALGORITHMS

- Get a Learner provided by `mlr`

```
learner = lrn("classif.rpart")
```

- Train the Learner

```
learner$train(task)
```

LEARNING ALGORITHMS

- Get a Learner provided by `mlr`

```
learner = lrn("classif.rpart")
```

- Train the Learner

```
learner$train(task)
```

- The `$model` is the `rpart` model: a decision tree

```
print(learner$model)

#> n= 150
#>
#> node), split, n, loss, yval, (yprob)
#>       * denotes terminal node
#>
#> 1) root 150 100 setosa (0.333 0.333 0.333)
#>    2) Petal.Length< 2.5 50  0 setosa (1.000 0.000 0.000) *
#>    3) Petal.Length>=2.5 100  50 versicolor (0.000 0.500 0.500)
#>      6) Petal.Width< 1.8 54   5 versicolor (0.000 0.907 0.093) *
#>      7) Petal.Width>=1.8 46   1 virginica (0.000 0.022 0.978) *
```

PREDICTION

- Let's make a prediction for some new data, e.g.:

```
new_data
```

```
#   Sepal.Length Sepal.Width Petal.Length Petal.Width
# 1           4          3          2          1
# 2           2          2          3          2
```

PREDICTION

- Let's make a prediction for some new data, e.g.:

```
new_data
#   Sepal.Length Sepal.Width Petal.Length Petal.Width
# 1           4         3          2          1
# 2           2         2          3          2
```

- To do so, we call the `$predict_newdata()` method using the new data:

```
prediction = learner$predict_newdata(new_data)
```

PREDICTION

- Let's make a prediction for some new data, e.g.:

```
new_data
#   Sepal.Length Sepal.Width Petal.Length Petal.Width
# 1           4         3          2          1
# 2           2         2          3          2
```

- To do so, we call the `$predict_newdata()` method using the new data:

```
prediction = learner$predict_newdata(new_data)
```

- We get a `Prediction` object:

```
prediction
#> <PredictionClassif> for 2 observations:
#>   row_id truth    response
#>     1  <NA>    setosa
#>     2  <NA> versicolor
```

HYPERPARAMETERS

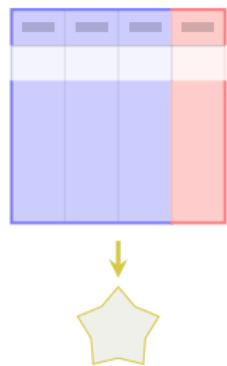
- Learners have *hyperparameters*

```
as.data.table(learner$param_set)[, 1:6]
```

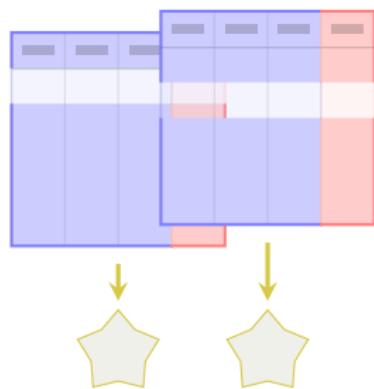
#>		id	class	lower	upper	levels	nlevels
#> 1:		minsplit	ParamInt	1	Inf		Inf
#> 2:		minbucket	ParamInt	1	Inf		Inf
#> 3:		cp	ParamDbl	0	1		Inf
#> 4:		maxcompete	ParamInt	0	Inf		Inf
#> 5:		maxsurrogate	ParamInt	0	Inf		Inf
#> 6:		maxdepth	ParamInt	1	30		30
#> 7:		usesurrogate	ParamInt	0	2		3
#> 8:	surrogatestyle	ParamInt		0	1		2
#> 9:		xval	ParamInt	0	Inf		Inf
#> 10:		keep_model	ParamLgl	NA	NA	TRUE, FALSE	2

Resampling

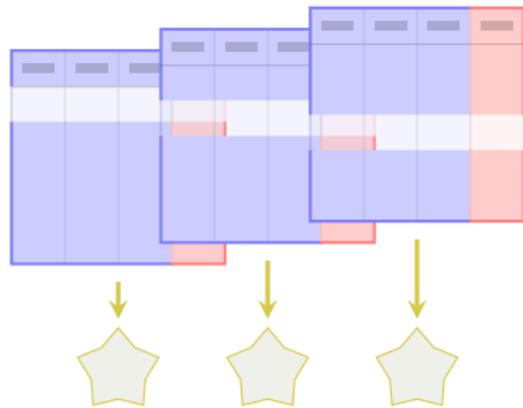
RESAMPLING



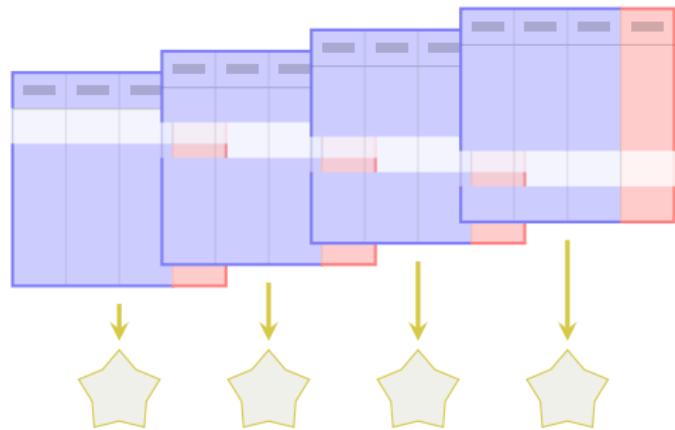
RESAMPLING



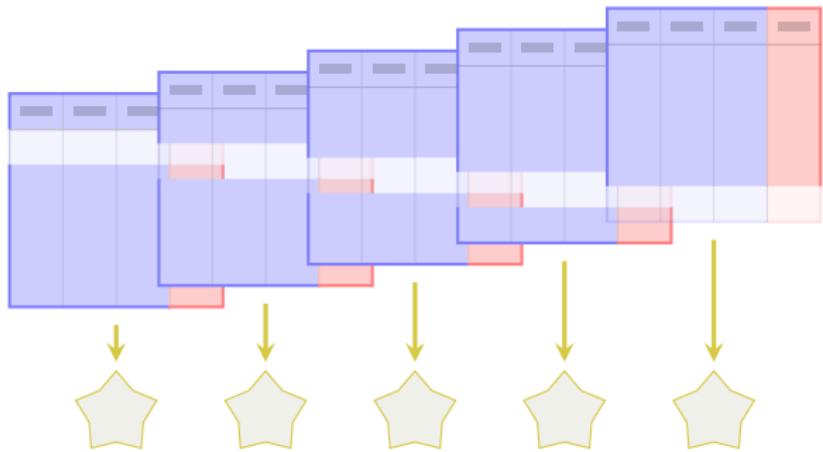
RESAMPLING



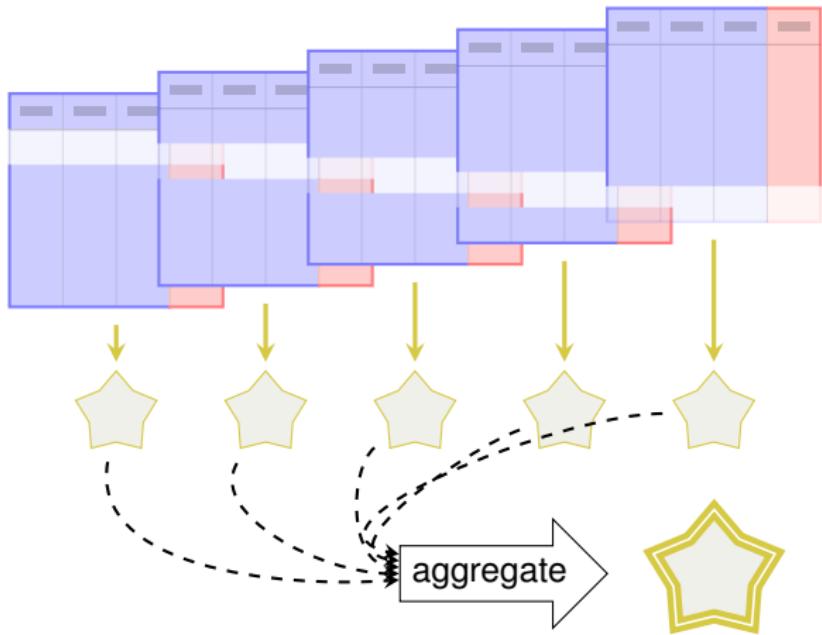
RESAMPLING



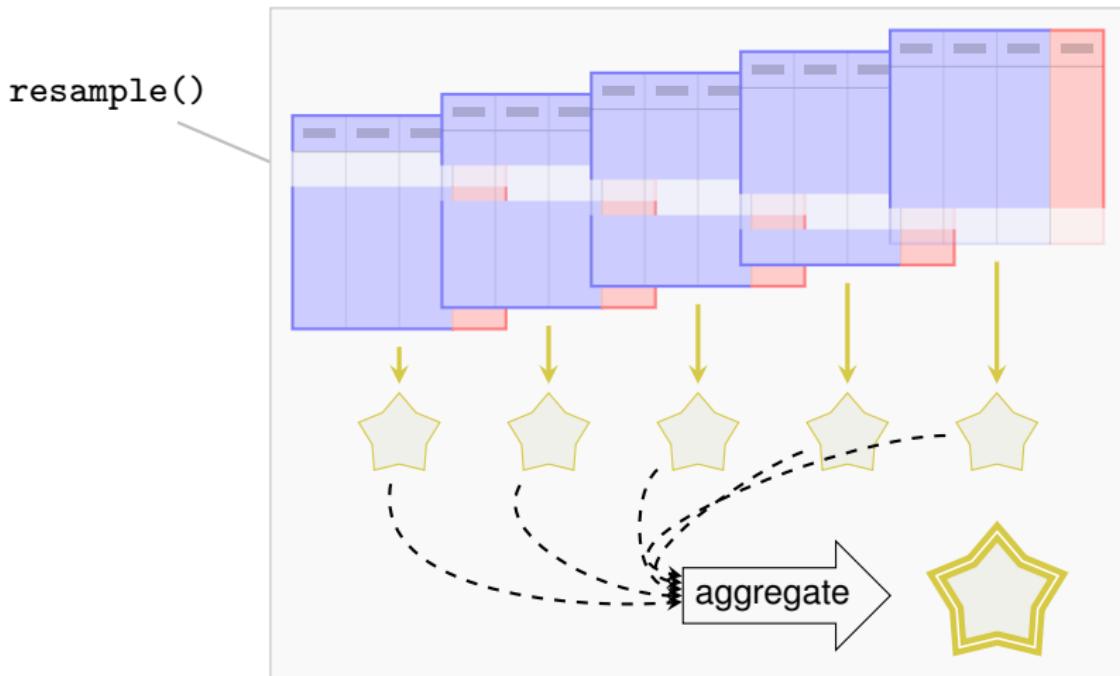
RESAMPLING



RESAMPLING



RESAMPLING



RESAMPLING

- Resample description: How to split the data

```
cv5 = rsmp("cv", folds = 5)
```

RESAMPLING

- Resample description: How to split the data

```
cv5 = rsmp("cv", folds = 5)
```

- Use the `resample()` function for resampling:

```
rr = resample(task, learner, cv5)
```

RESAMPLING

- Resample description: How to split the data

```
cv5 = rsmp("cv", folds = 5)
```

- Use the `resample()` function for resampling:

```
rr = resample(task, learner, cv5)
```

- We get a `ResamplingResult` object:

```
print(rr)

#> <ResampleResult> of 5 iterations
#> * Task: iris
#> * Learner: classif.rpart
#> * Warnings: 0 in 0 iterations
#> * Errors: 0 in 0 iterations
```

RESAMPLING RESULTS

- Calculate performance:

```
rr$aggregate(msr("classif.ce"))
#> classif.ce
#>      0.067
```

RESAMPLING RESULTS

- Calculate performance:

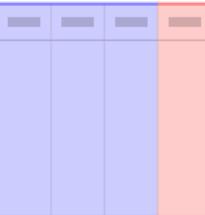
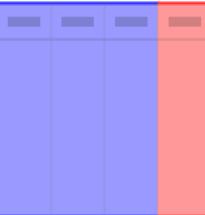
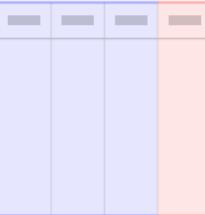
```
rr$aggregate(msr("classif.ce"))
#> classif.ce
#>     0.067
```

- Get predictions

```
rr$prediction()
#> <PredictionClassif> for 150 observations:
#>   row_id    truth  response
#>   3      setosa    setosa
#>   13     setosa    setosa
#>   14     setosa    setosa
#>   ---
#>   136  virginica virginica
#>   139  virginica versicolor
#>   141  virginica virginica
```

Benchmark

PERFORMANCE COMPARISON

	Learner 1	Learner 2	Learner 3
			
			
			

PERFORMANCE COMPARISON

- Multiple Learners, multiple Tasks:

```
library("mlr3learners")
learners = list(lrn("classif.rpart"), lrn("classif.kknn"))
tasks = list(tsk("iris"), tsk("sonar"), tsk("wine"))
```

PERFORMANCE COMPARISON

- Multiple Learners, multiple Tasks:

```
library("mlr3learners")
learners = list(lrn("classif.rpart"), lrn("classif.kknn"))
tasks = list(tsk("iris"), tsk("sonar"), tsk("wine"))
```

- Set up the *design* and execute benchmark:

```
design = benchmark_grid(tasks, learners, cv5)
bmr = benchmark(design)
```

PERFORMANCE COMPARISON

- Multiple Learners, multiple Tasks:

```
library("mlr3learners")
learners = list(lrn("classif.rpart"), lrn("classif.kknn"))
tasks = list(tsk("iris"), tsk("sonar"), tsk("wine"))
```

- Set up the *design* and execute benchmark:

```
design = benchmark_grid(tasks, learners, cv5)
bmr = benchmark(design)
```

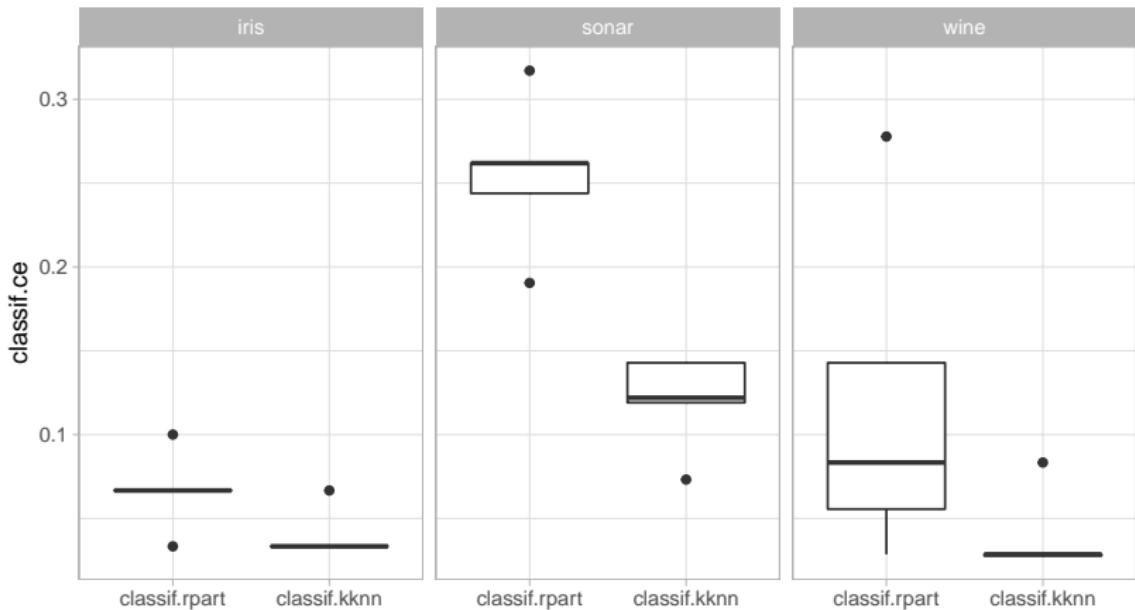
- We get a `BenchmarkResult` object which shows that `kknn` outperforms `rpart`:

```
bmr_ag = bmr$aggregate()
bmr_ag[, c("task_id", "learner_id", "classif.ce")]
#>   task_id   learner_id classif.ce
#> 1:   iris classif.rpart    0.067
#> 2:   iris classif.kknn    0.040
#> 3:  sonar classif.rpart    0.255
#> 4:  sonar classif.kknn    0.120
#> 5:  wine classif.rpart    0.118
#> 6:  wine classif.kknn    0.039
```

BENCHMARK RESULT

The `mlr3viz` package contains `autoplot()` functions for many `mlr3` objects

```
library(mlr3viz)
autoplot(bmr)
```



Tuning

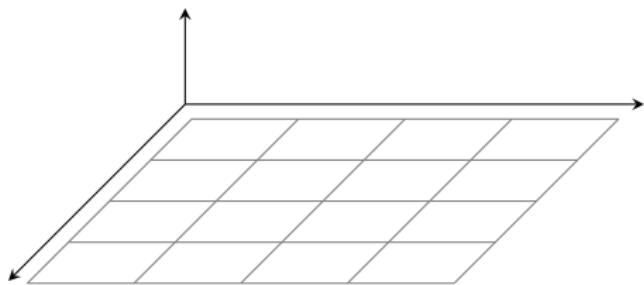
TUNING

- Behavior of most methods depends on *hyperparameters*
 - We want to choose them so our algorithm performs well
 - Good hyperparameters are data-dependent
- ⇒ We do *black box optimization* (“Try stuff and see what works”)

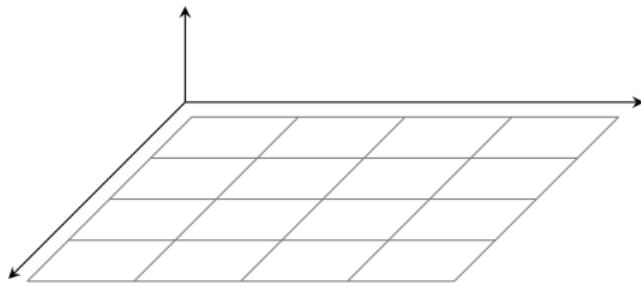
Tuning toolbox for `mlr3`:

```
library("bbottk")
library("mlr3tuning")
```

TUNING

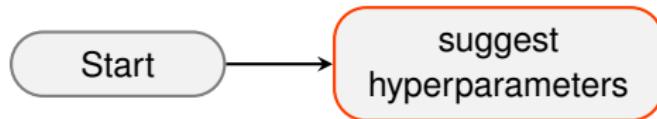
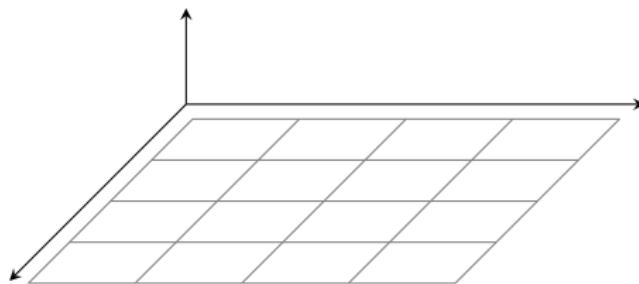


TUNING

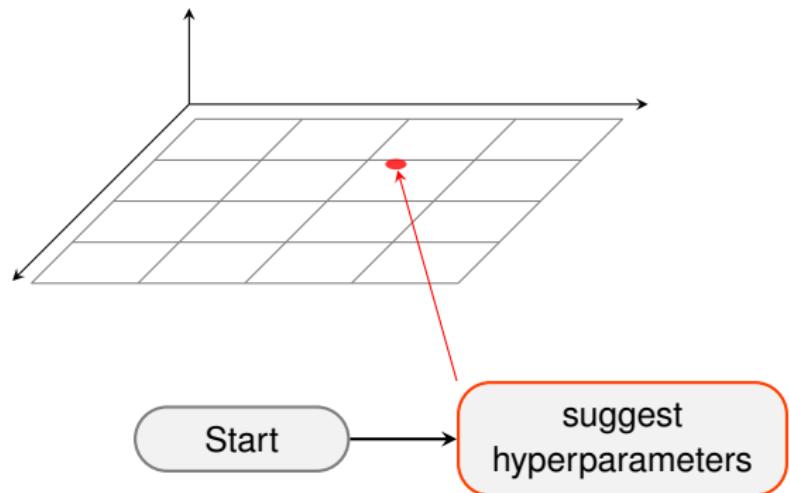


Start

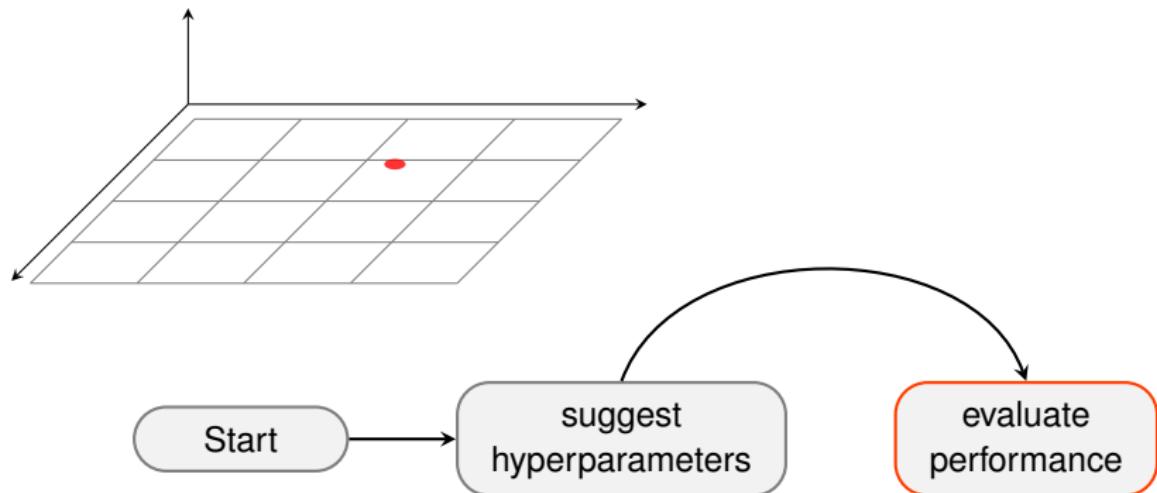
TUNING



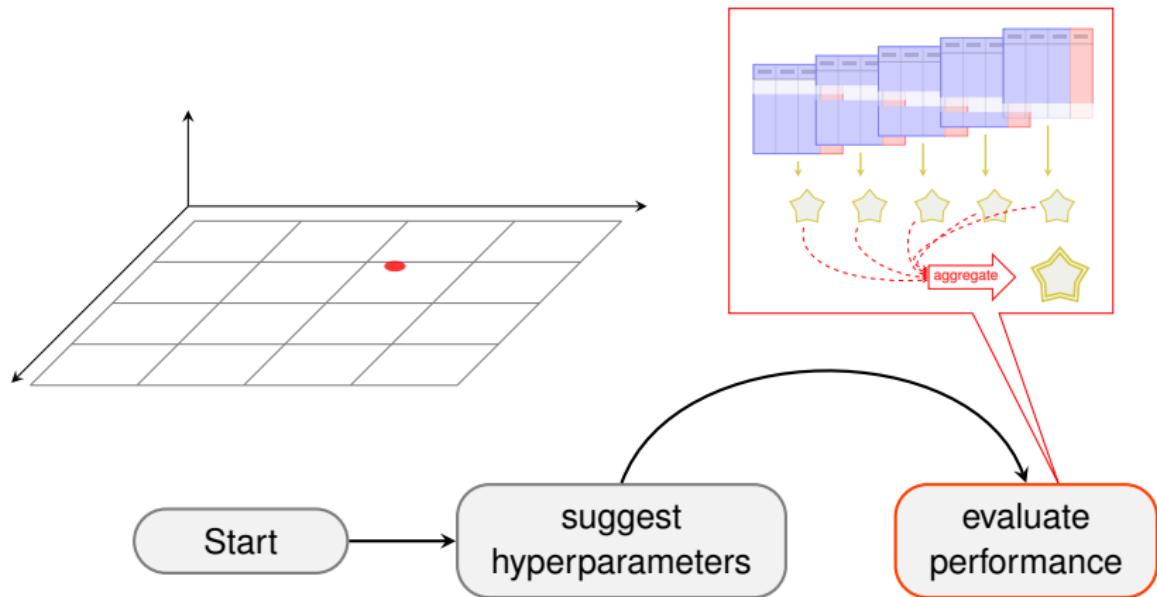
TUNING



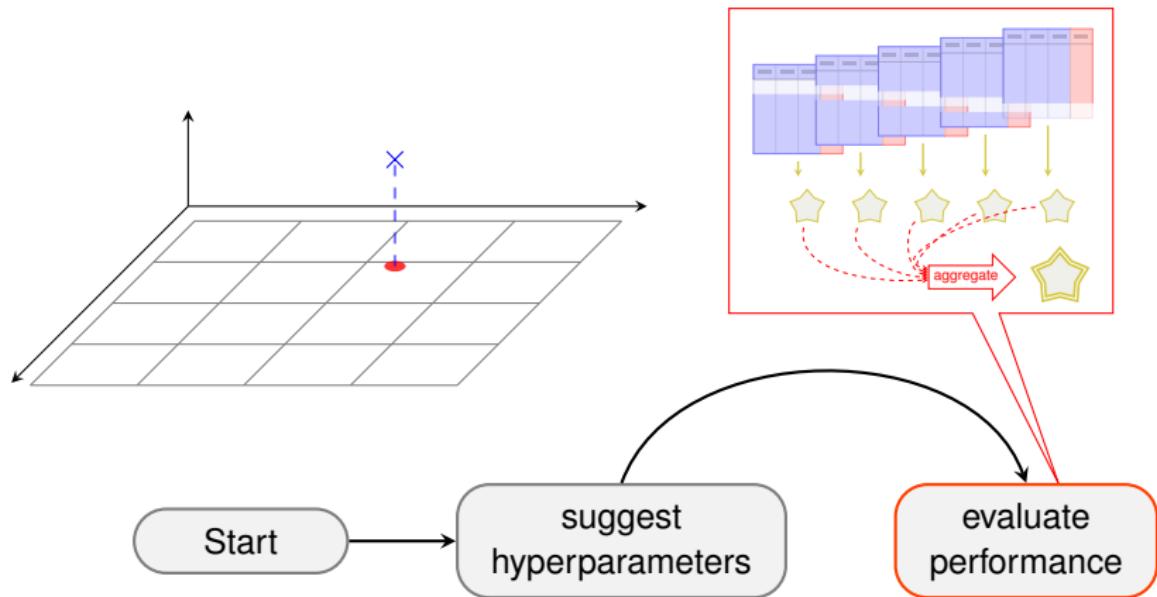
TUNING



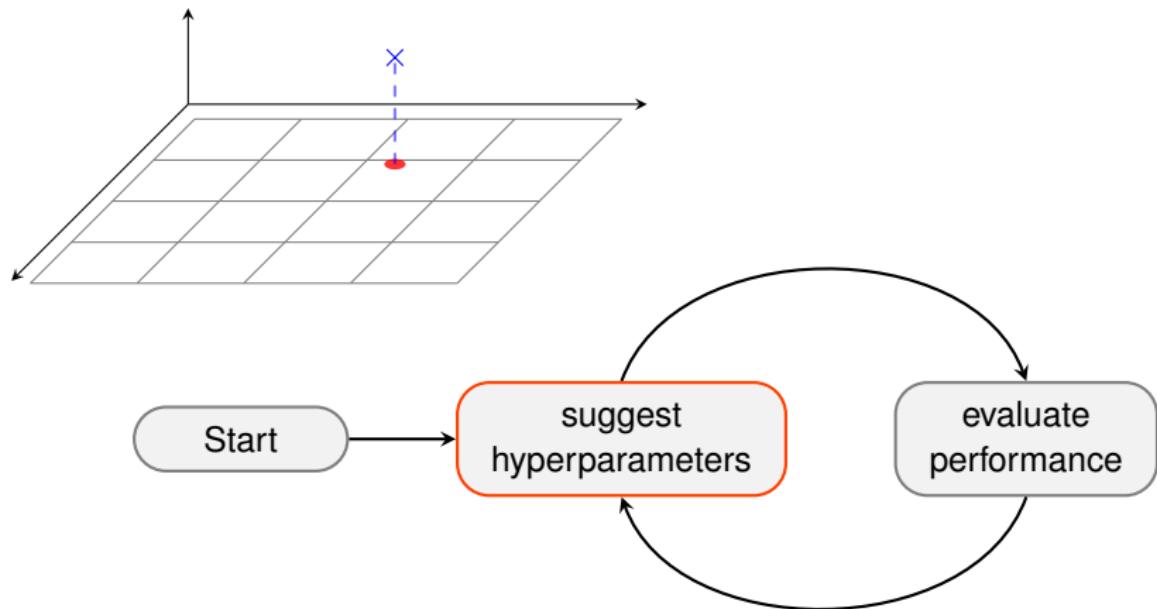
TUNING



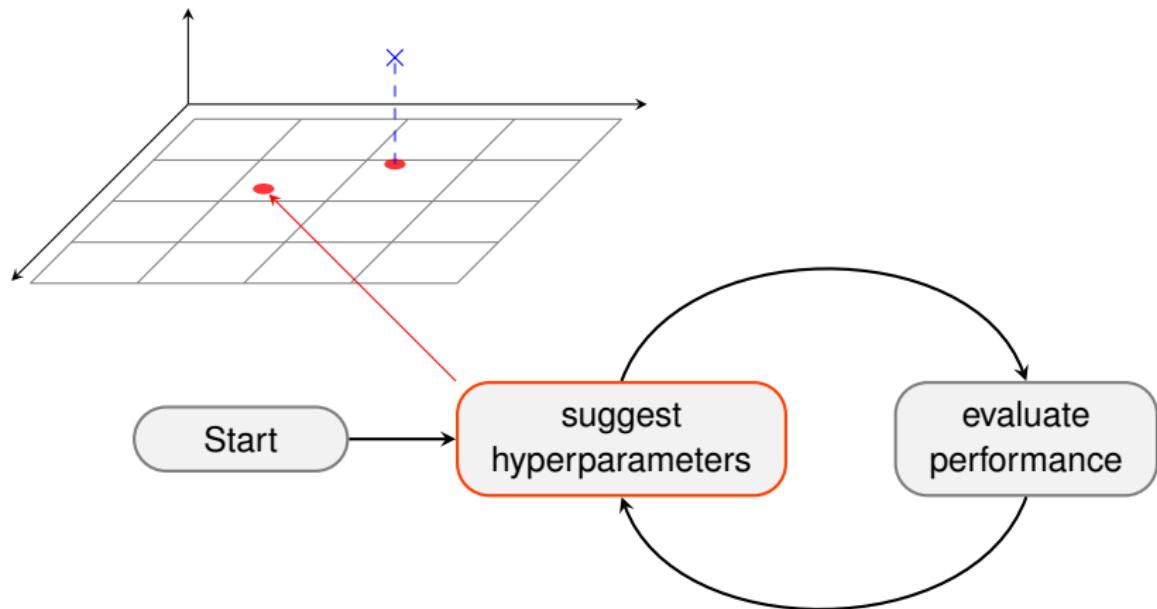
TUNING



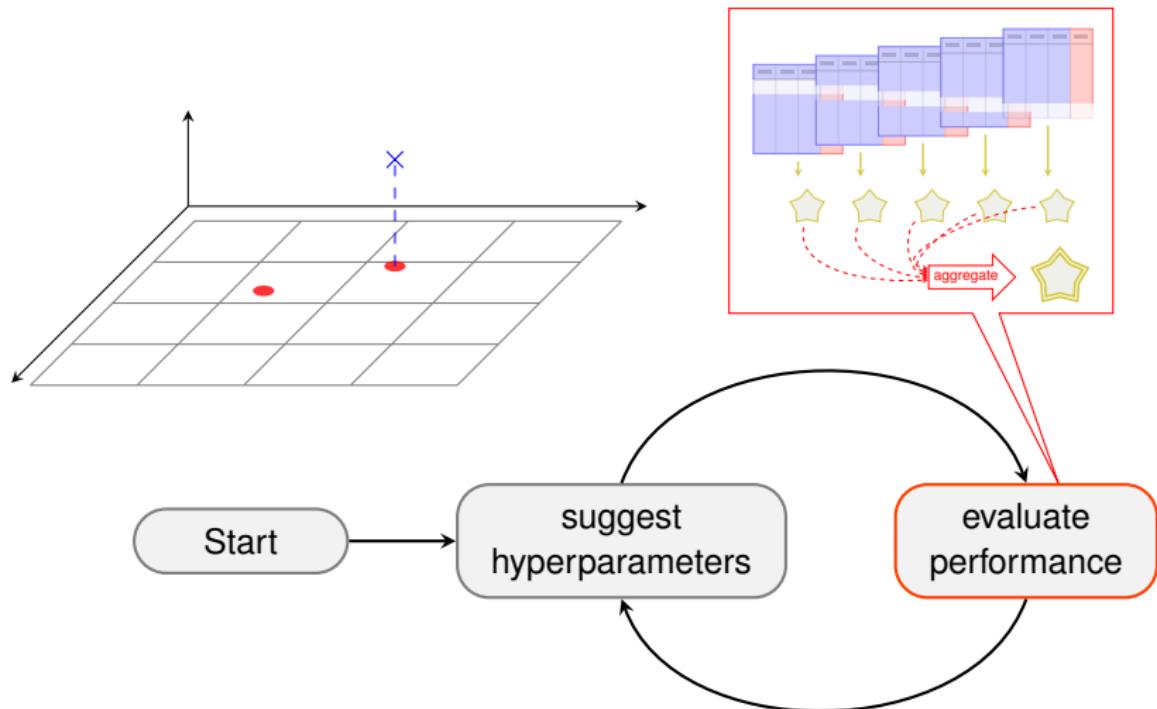
TUNING



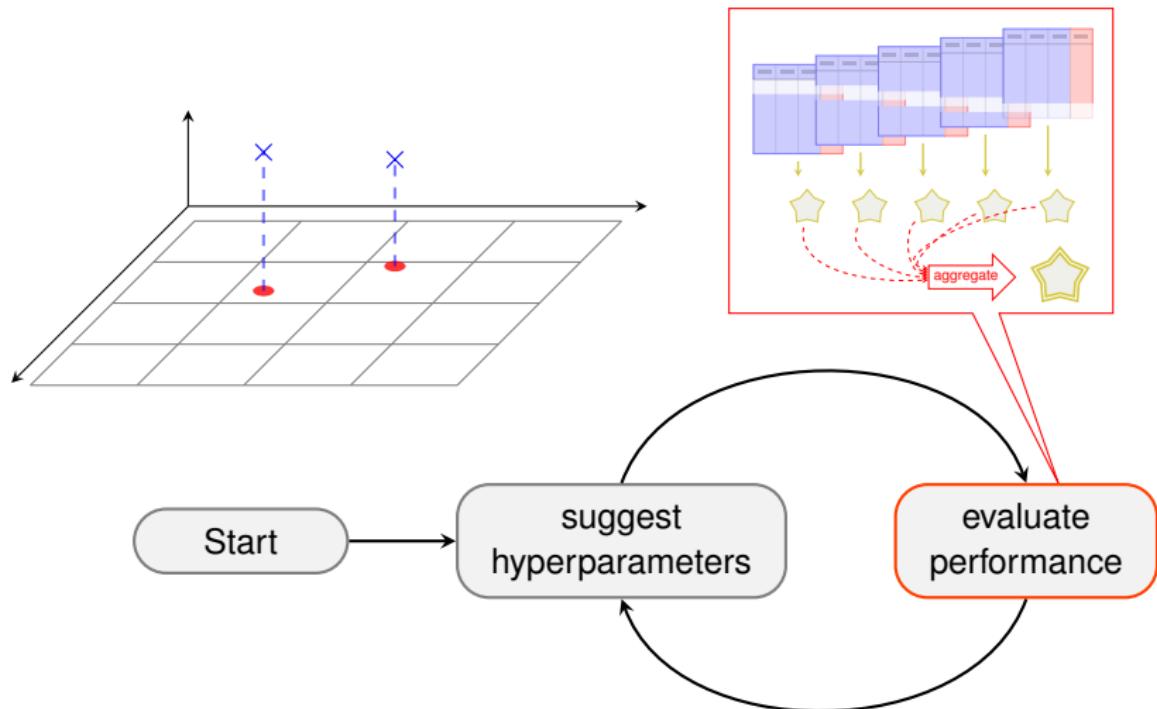
TUNING



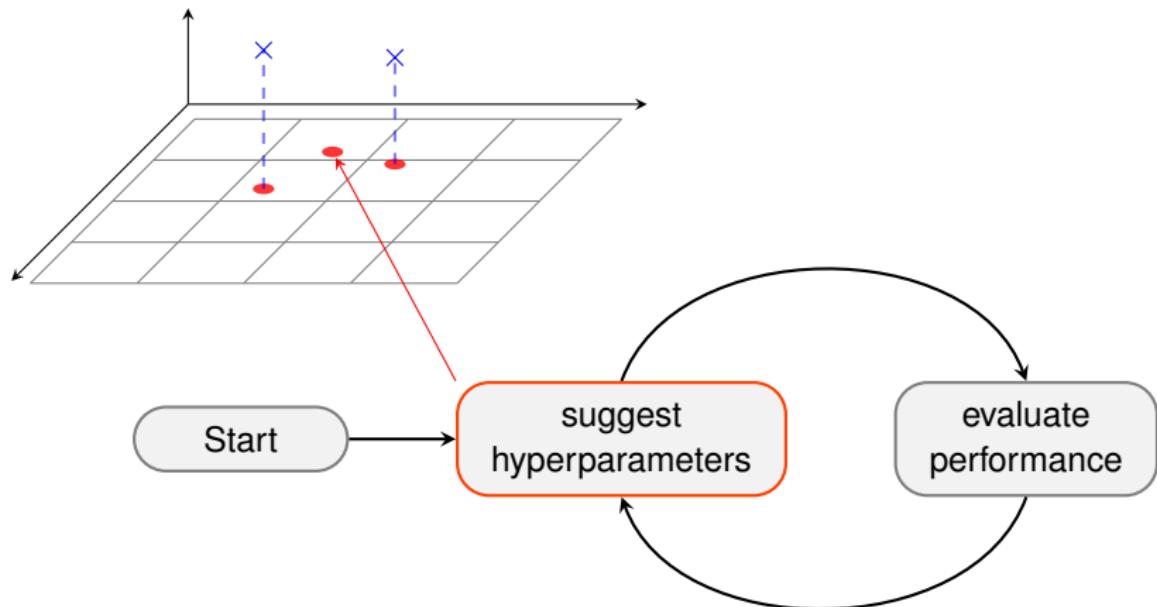
TUNING



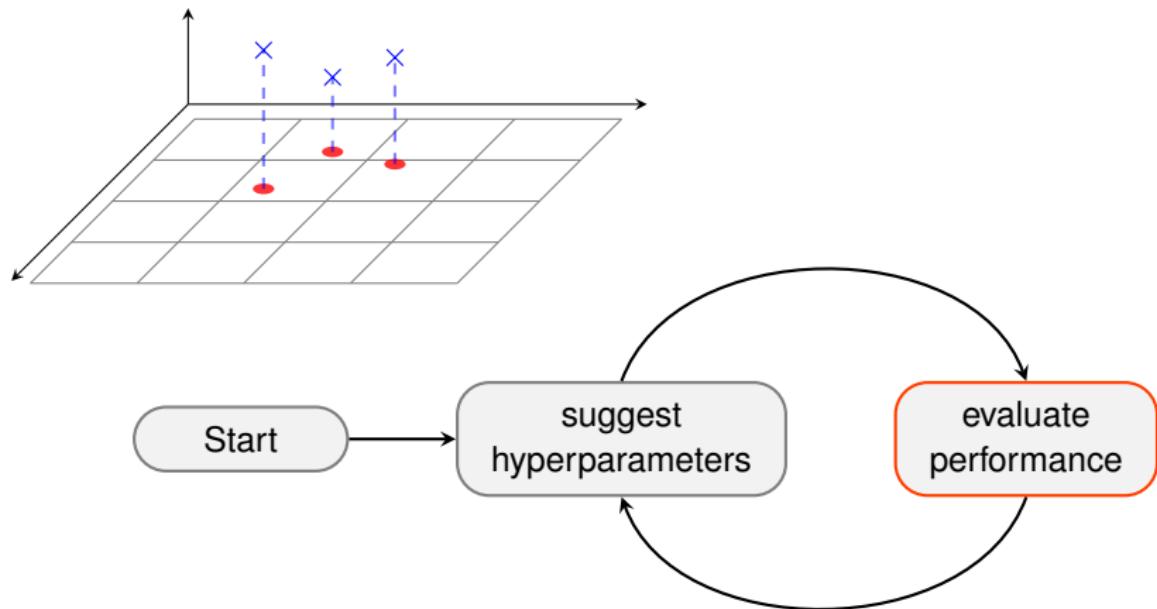
TUNING



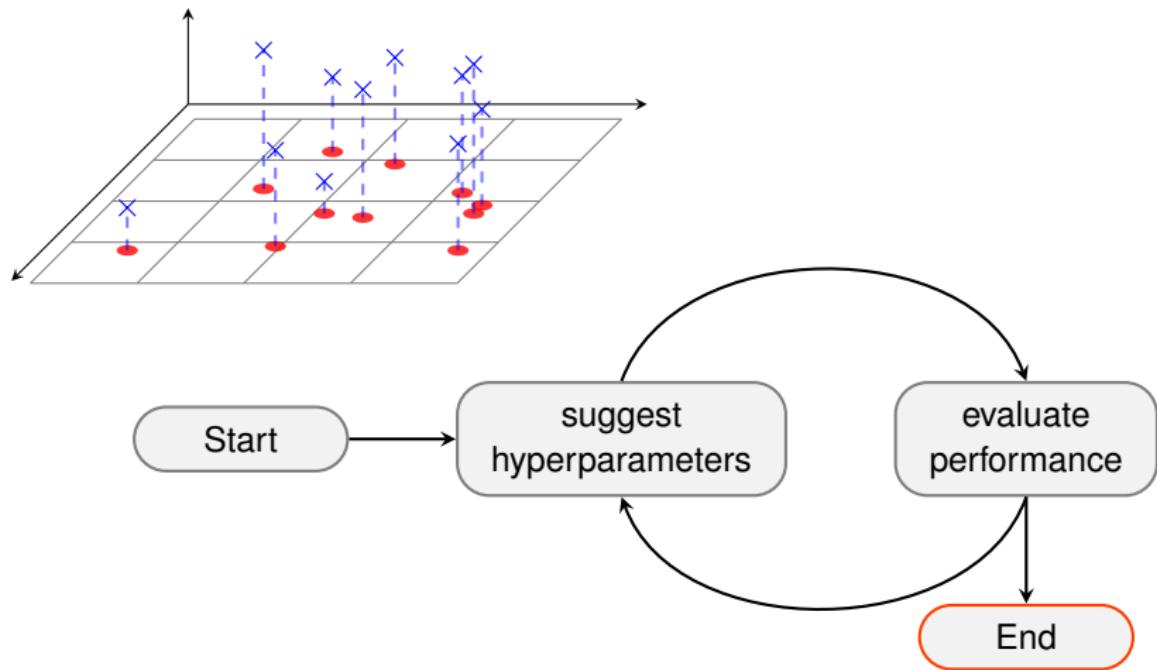
TUNING



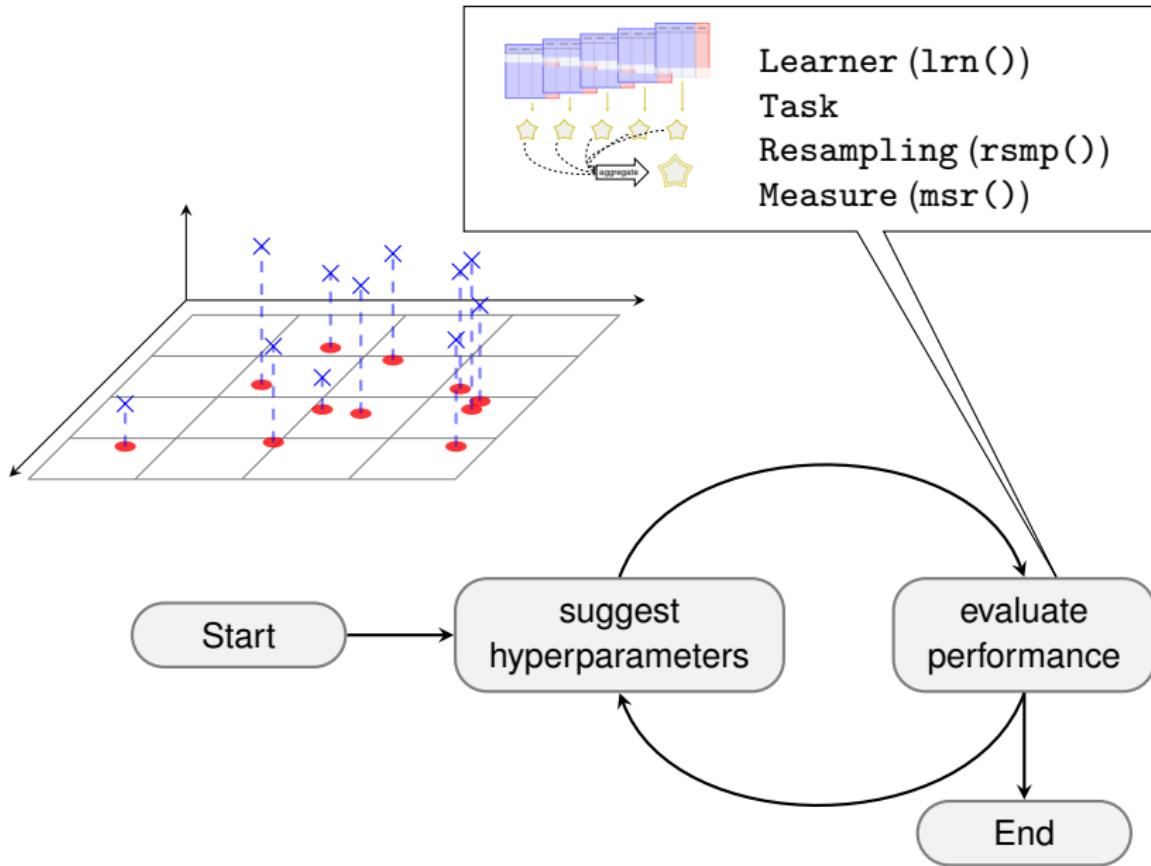
TUNING



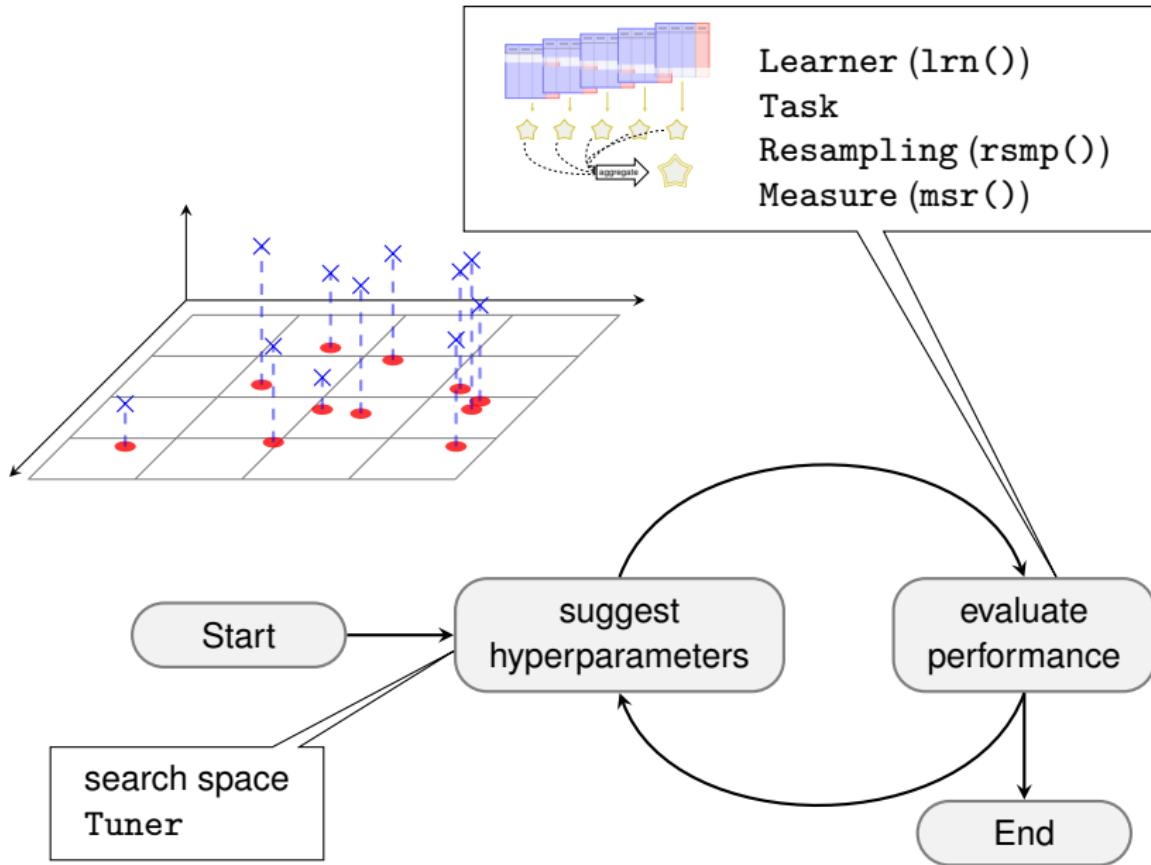
TUNING



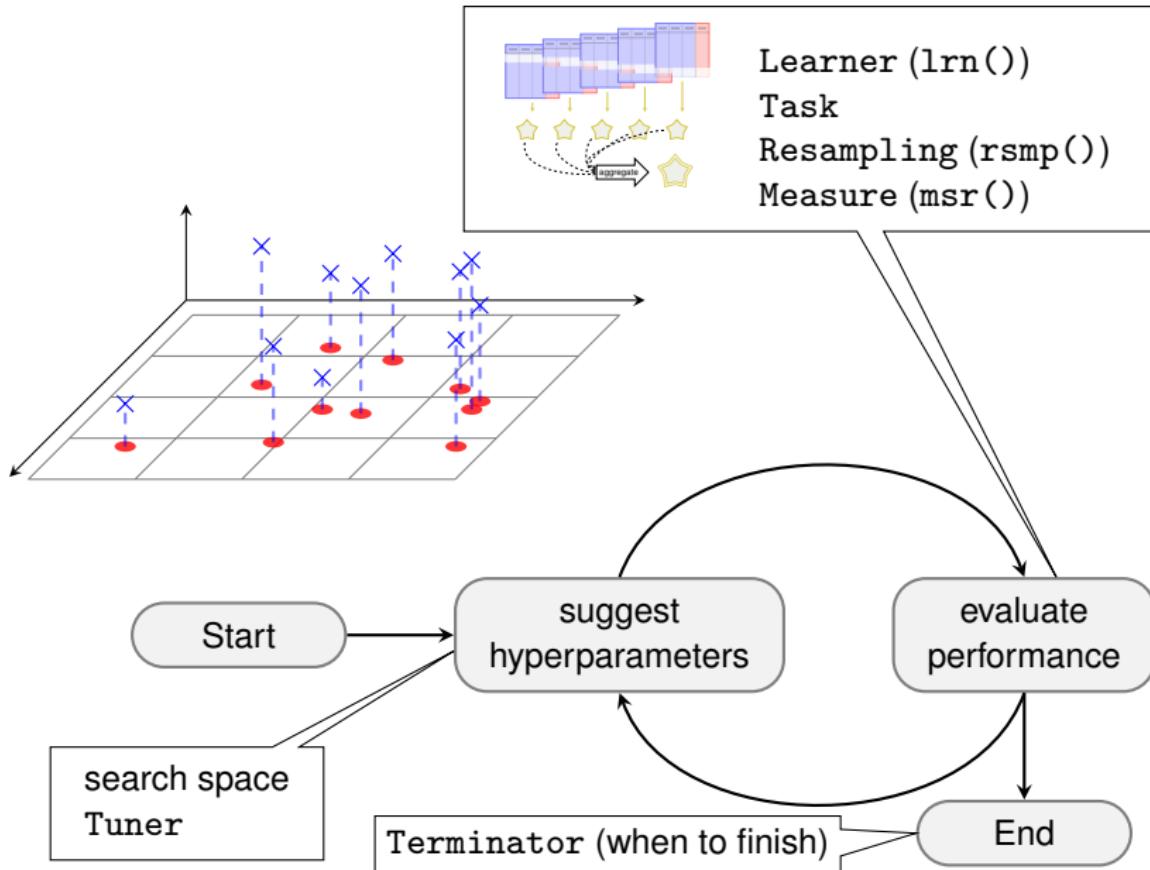
TUNING



TUNING

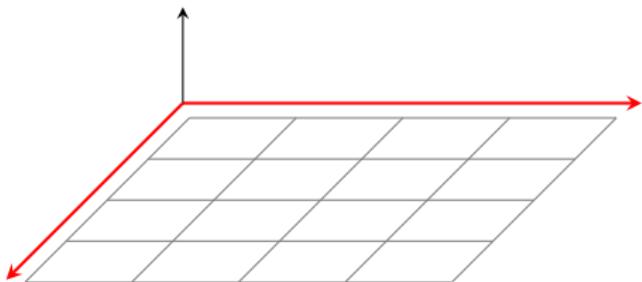


TUNING

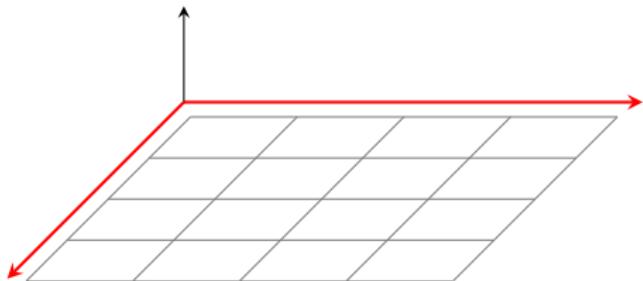


Tuning in mlr3

SEARCH SPACE

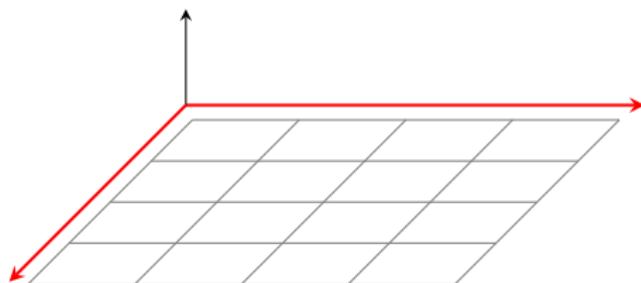


SEARCH SPACE



```
ps(param1 = ..., param2 = ..., ...)
```

SEARCH SPACE



```
ps(param1 = ..., param2 = ..., ...)
```

Numerical parameter `p_dbl(lower, upper)`

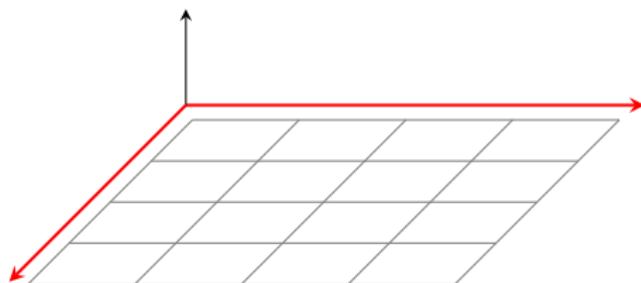
Integer parameter `p_int(lower, upper)`

Discrete parameter `p_fct(levels)`

Logical parameter `p_lgl()`

Untyped parameter `p_uty()`

SEARCH SPACE



```
ps(param1 = ..., param2 = ..., ...)
```

Numerical parameter `p_dbl(lower, upper)`

Integer parameter `p_int(lower, upper)`

Discrete parameter `p_fct(levels)`

Logical parameter `p_lgl()`

Untyped parameter `p_uty()`

```
library("paradox")
searchspace_knn = ps(k = p_int(1, 20))
```

TERMINATION

- Tuning needs a *termination condition*: when to finish
- Terminator class
- `mlr_terminators` dictionary, `trm()` short form

- `as.data.table(mlr_terminators)`

```
#>          key
#> 1:      clock_time
#> 2:      combo
#> 3:      evals
#> 4:      none
#> 5:      perf_reached
#> 6:      run_time
#> 7:      stagnation
#> 8:      stagnation_batch
```

- `trm("evals", n_evals = 20)`

```
#> <TerminatorEvals>
#> * Parameters: n_evals=20
```

TUNING METHOD

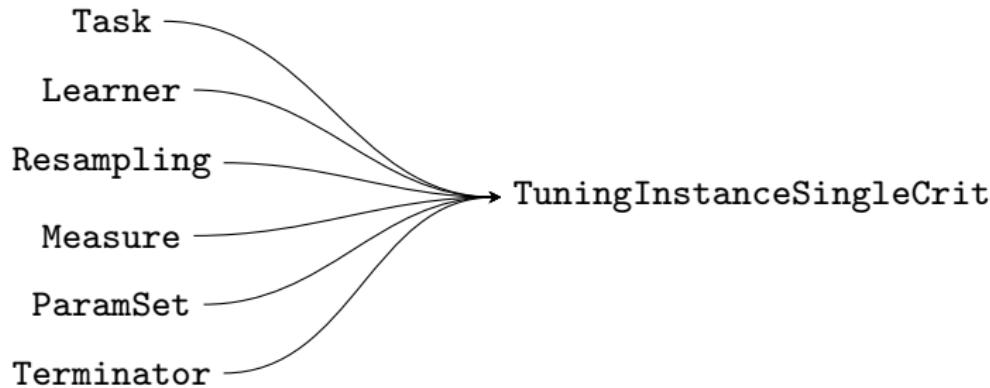
- load Tuner with `tunr()`, set parameters

- ```
gsearch = tunr("grid_search", resolution = 3)

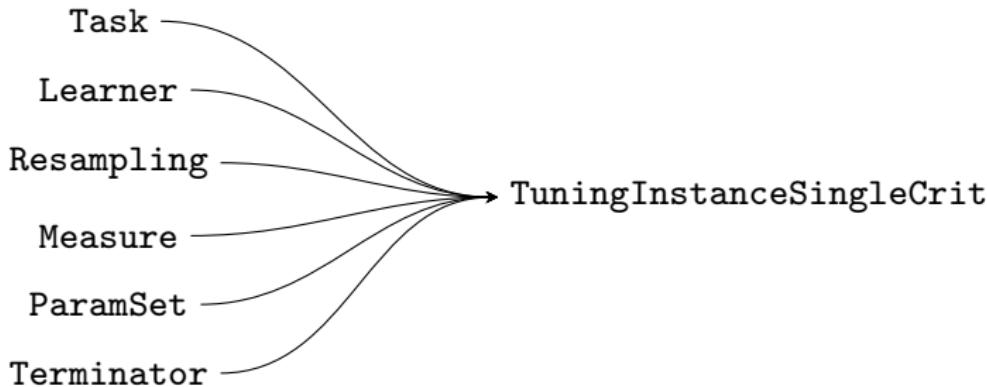
print(gsearch)
#> <TunerGridSearch>
#> * Parameters: resolution=3, batch_size=1
#> * Parameter classes: ParamLgl, ParamInt, ParamDbl, ParamFct
#> * Properties: dependencies, single-crit, multi-crit
#> * Packages: -
```

- common parameter `batch_size` for parallelization

# CALLING THE TUNER



# CALLING THE TUNER



```
inst = TuningInstanceSingleCrit$new(
 tsk("iris"), lrn("classif.knn", kernel="rectangular"),
 rsmp("holdout"), msr("classif.ce"), trm("none"),
 searchspace_knn
)
```

# CALLING THE TUNER

```
gsearch$optimize(inst)

#> INFO [15:07:32.603] Starting to optimize 1 parameter(s) with '<OptimizerGridSearch>'

#> INFO [15:07:32.642] Evaluating 1 configuration(s)

#> INFO [15:07:33.800] Result of batch 1:

#> INFO [15:07:33.801] k classif.ce uhash
#> INFO [15:07:33.801] 10 0.04 cc2a0cc9-b233-4aa7-abce-6d2f39d8ddba

#> INFO [15:07:33.803] Evaluating 1 configuration(s)

#> INFO [15:07:34.016] Result of batch 2:

#> INFO [15:07:34.018] k classif.ce uhash
#> INFO [15:07:34.018] 1 0.06 4c6e7c20-8b21-4999-bd71-129126c7a918

#> INFO [15:07:34.020] Evaluating 1 configuration(s)

#> INFO [15:07:34.116] Result of batch 3:

#> INFO [15:07:34.118] k classif.ce uhash
#> INFO [15:07:34.118] 20 0.08 b3f6a62c-d6fd-4eb9-94ca-db66118f5fed

#> INFO [15:07:34.124] Finished optimizing after 3 evaluation(s)

#> INFO [15:07:34.125] Result:

#> INFO [15:07:34.127] k learner_param_vals x_domain classif.ce
#> INFO [15:07:34.127] 10 <list[2]> <list[1]> 0.04

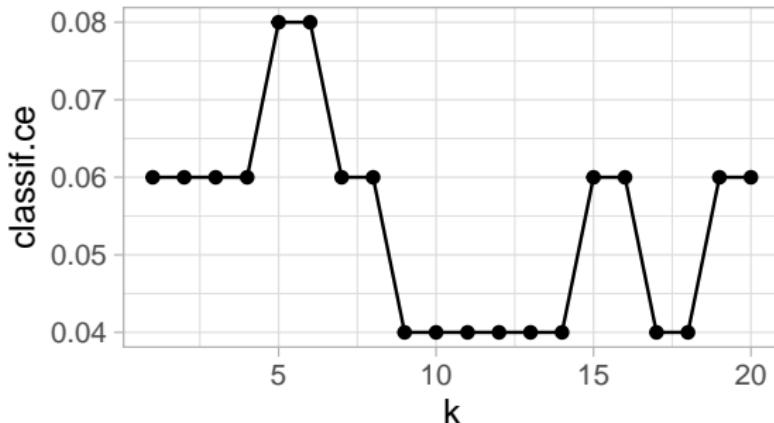
#> k learner_param_vals x_domain classif.ce
#> 1: 10 <list[2]> <list[1]> 0.04
```

# TUNING RESULTS

```
gsearch = tnr("grid_search", resolution = 20)
inst = TuningInstanceSingleCrit$new(
 tsk("iris"), lrn("classif.kknn", kernel="rectangular"),
 rsmp("holdout"), msr("classif.ce"), trm("none"),
 searchspace_knn)
gsearch$optimize(inst)

#> k learner_param_vals x_domain classif.ce
#> 1: 11 <list[2]> <list[1]> 0.04

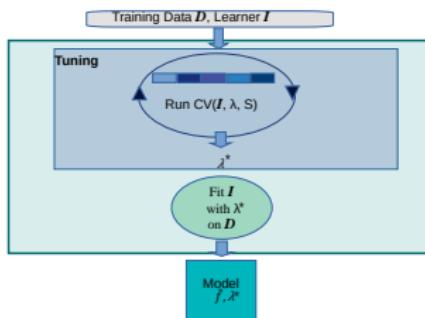
ggplot(inst$archive$data(), aes(x = k, y = classif.ce)) +
 geom_line() + geom_point()
```



## **Nested Resampling**

# NESTED RESAMPLING

- Need to perform nested resampling to estimate tuned learner performance
- ⇒ Treat tuning as if it were a Learner!
  - Training:
    - ➊ Tune model using (inner) resampling
    - ➋ Train final model with best parameters on all (i.e. outer resampling) data
  - Predicting: Just use final model



# NESTED RESAMPLING

```
optlrn = AutoTuner$new(lrn("classif.kknn", kernel="rectangular"),
 rsmp("holdout"), msr("classif.ce"), trm("none"),
 tnr("grid_search", resolution = 10), searchspace_knn)
```

```
optlrn$train(tsk("iris"))
```

```
optlrn$model$learner

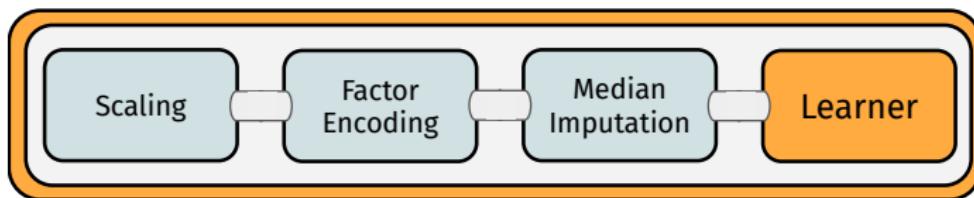
#> <LearnerClassifKKNN:classif.kknn>
#> * Model: list
#> * Parameters: kernel=rectangular, k=18
#> * Packages: kknn
#> * Predict Type: response
#> * Feature types: logical, integer, numeric, factor, ordered
#> * Properties: multiclass, twoclass
```

# MLR3PIPELINES

## Machine Learning Workflows:

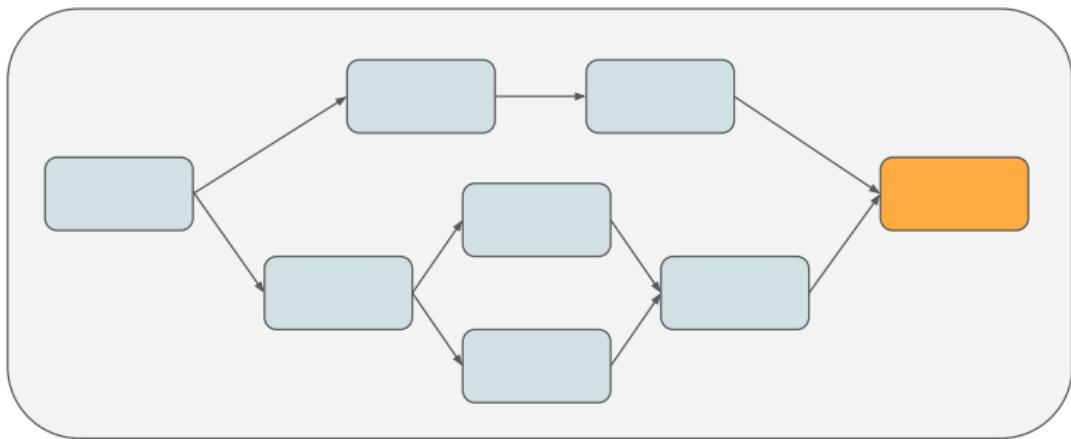
- **Preprocessing:** Feature extraction, feature selection, missing data imputation,...
- **Ensemble methods:** Model averaging, model stacking
- **mlr3:** modular model fitting

⇒ **mlr3pipelines:** modular ML workflows



# MACHINE LEARNING WORKFLOWS

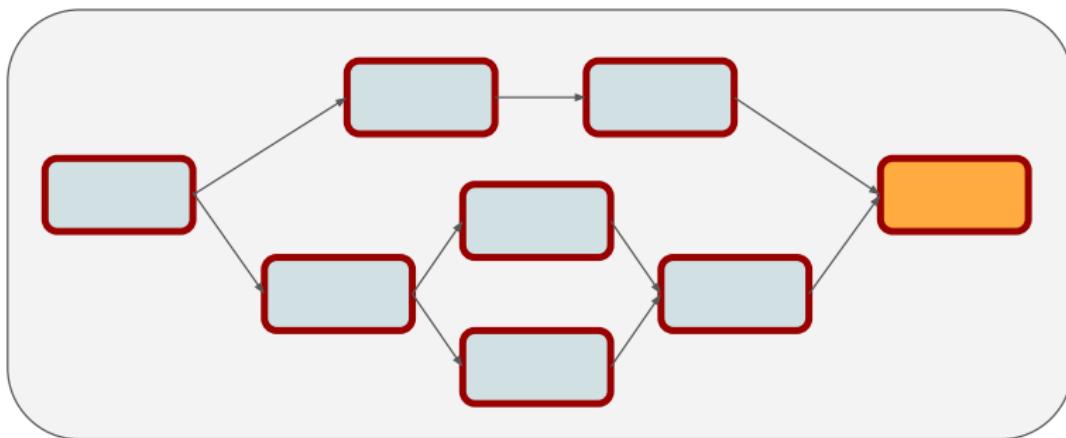
– what do they look like?



# MACHINE LEARNING WORKFLOWS

– what do they look like?

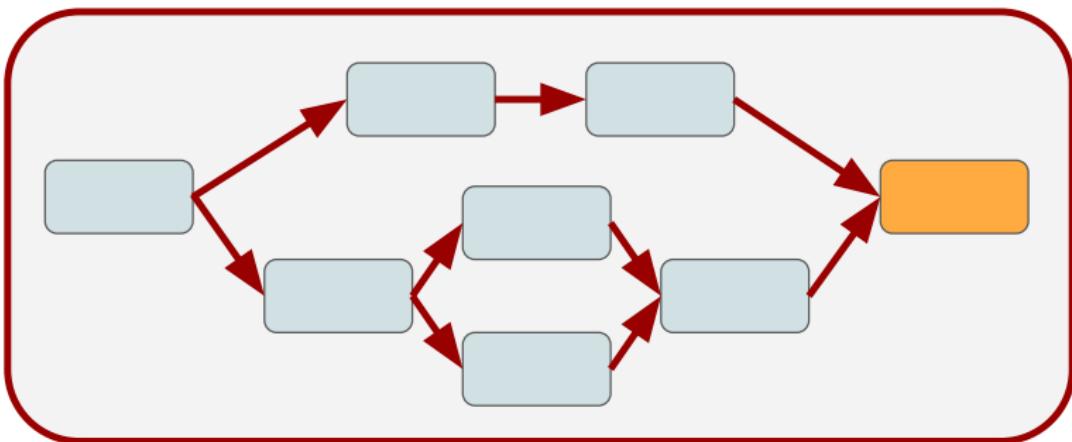
- **Building blocks:** *what is happening? → PipeOp*



# MACHINE LEARNING WORKFLOWS

– what do they look like?

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



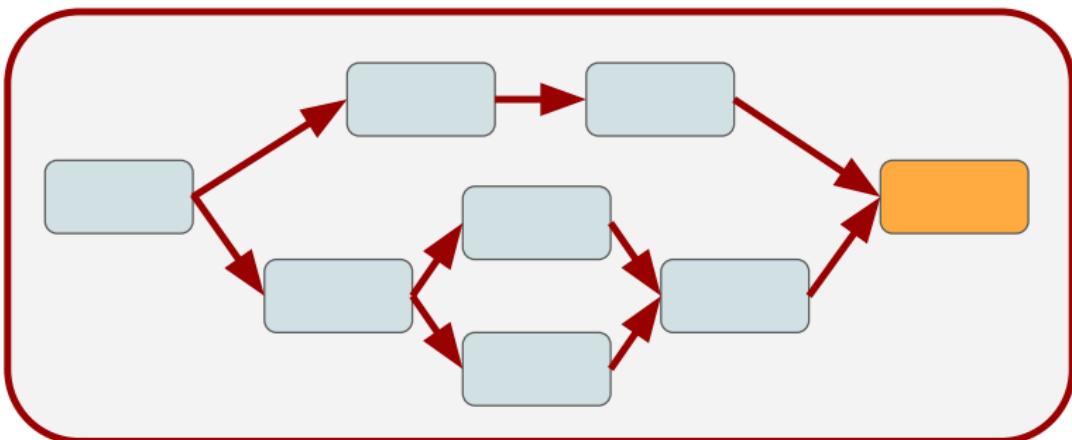
# MACHINE LEARNING WORKFLOWS

– 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

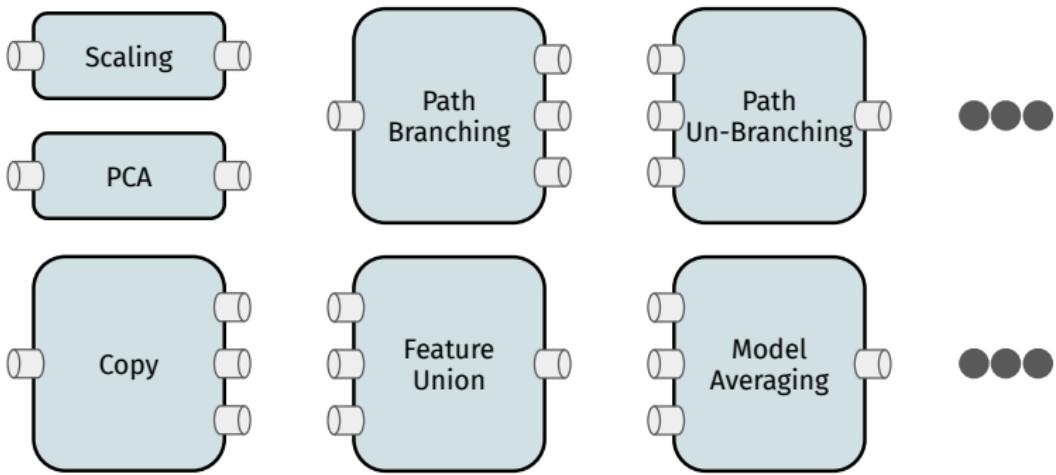


# PipeOps

# THE BUILDING BLOCKS

## PipeOp: Single Unit of Data Operation

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



# PIPEOPS SO FAR

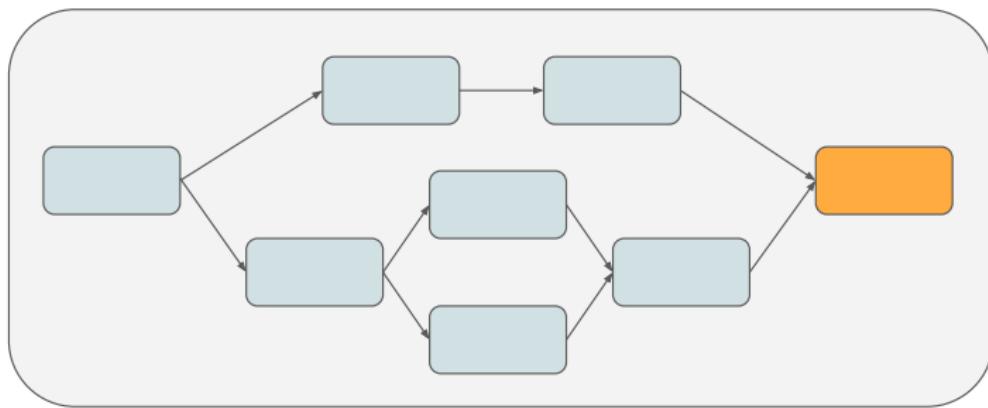
```
as.data.table(mlr_pipeops)

#> key packages tags ...
#> 1 boxcox bestNormalize data transform
#> 2 branch meta
#> 3 chunk meta
#> 4 classbalancing imbalanced data, data transform
#> 5 classifavg stats ensemble
#> 6 classweights imbalanced data, data transform
#> 7 colapply data transform
#> 8 collapsefactors data transform
#> 9 colroles data transform
#> 10 copy meta
#> 11 datefeatures data transform
#> 12 encode stats encode, data transform
#> 13 encodeimpact encode, data transform
#> 14 encodelmer lme4, nloptr encode, data transform
#> 15 featureunion ensemble
#> 16 filter feature selection, data transform
#> 17 fixfactors robustify, data transform
#> [...]
```

# **Graph Operations**

# THE STRUCTURE

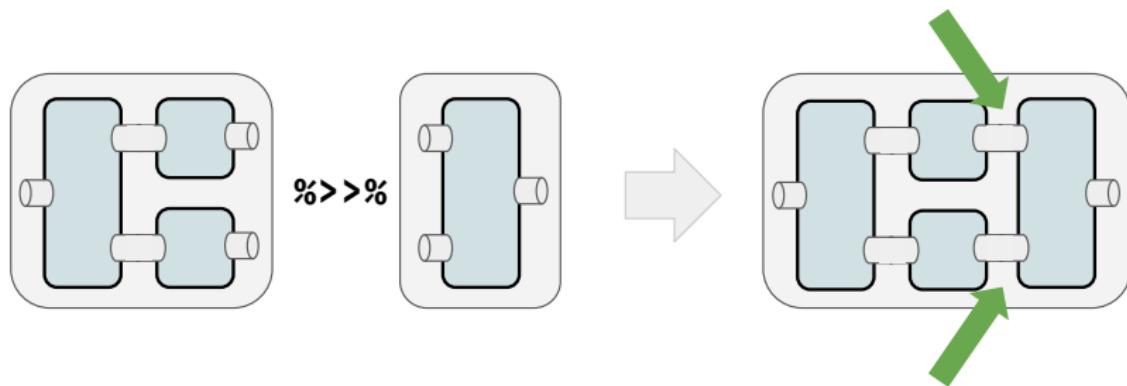
## Graph Operations



# THE STRUCTURE

## Graph Operations

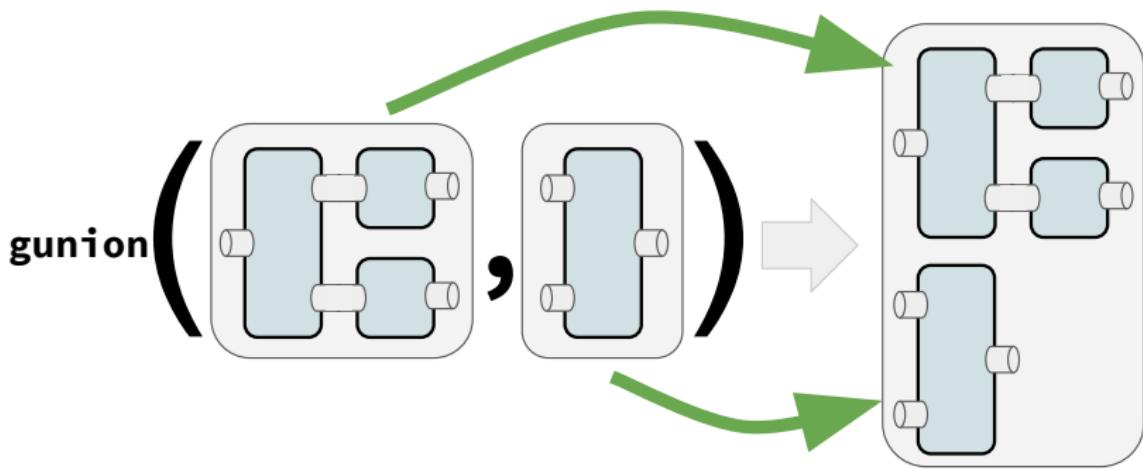
- The `%>>%`-operator concatenates Graphs and PipeOps



# THE STRUCTURE

## Graph Operations

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

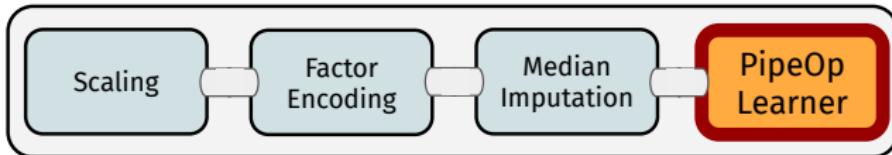


# **Linear Pipelines**

# LEARNERS AND GRAPHS

## PipeOpLearner

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



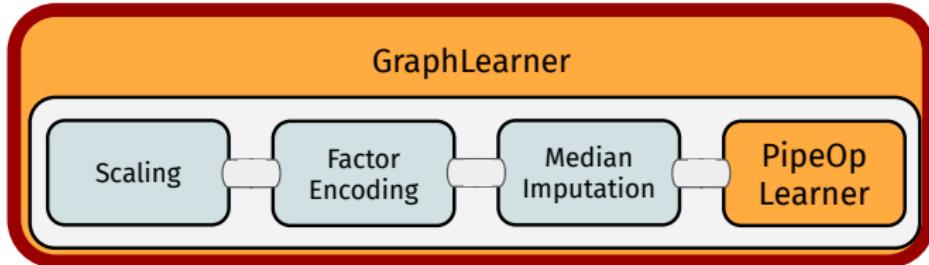
# LEARNERS AND GRAPHS

## PipeOpLearner

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

## GraphLearner

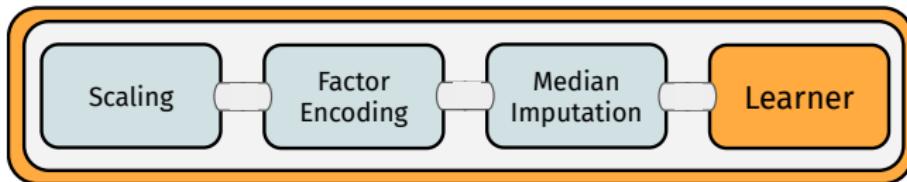
- Graph as a Learner
- All benefits of `mlr3`: **resampling, tuning, nested resampling, ...**



# MLR3PIPELINES IN ACTION

## Linear Preprocessing Pipeline

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

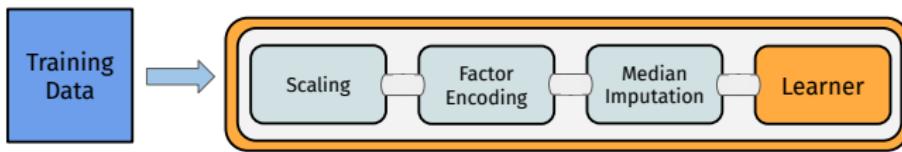


# MLR3PIPELINES IN ACTION

## Linear Preprocessing Pipeline

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

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

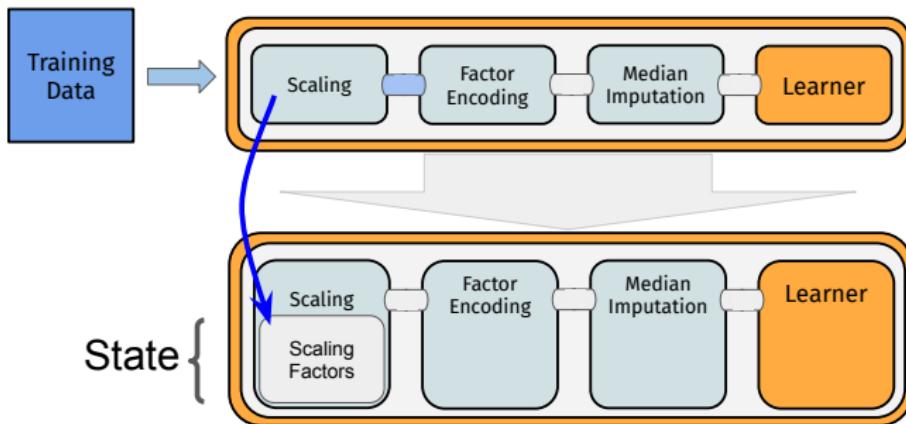


# MLR3PIPELINES IN ACTION

## Linear Preprocessing Pipeline

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

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

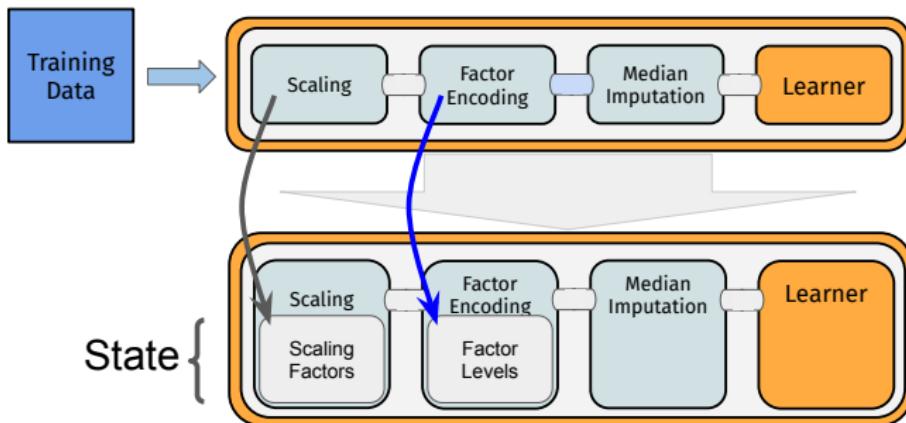


# MLR3PIPELINES IN ACTION

## Linear Preprocessing Pipeline

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

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

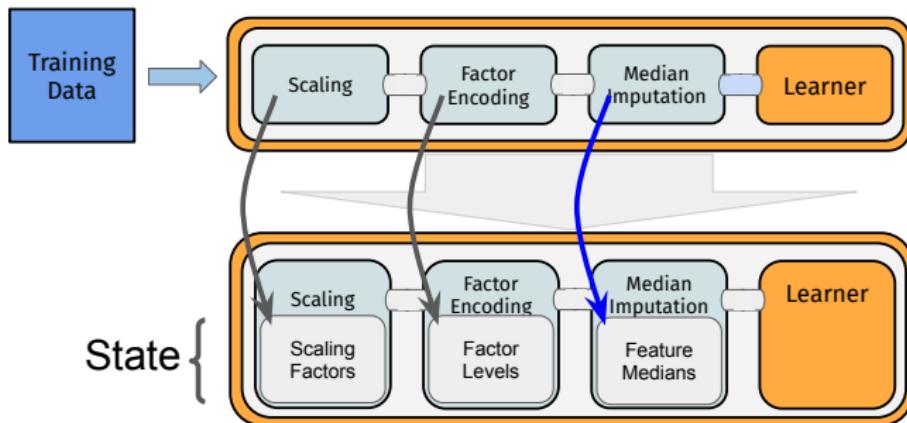


# MLR3PIPELINES IN ACTION

## Linear Preprocessing Pipeline

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

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

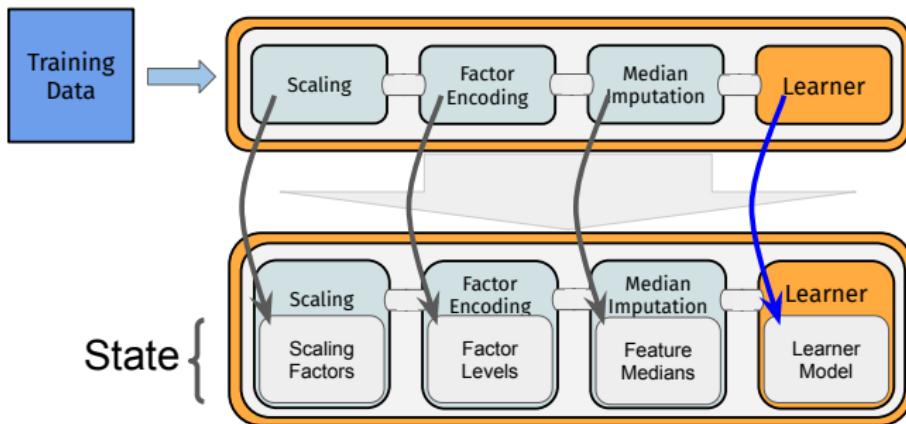


# MLR3PIPELINES IN ACTION

## Linear Preprocessing Pipeline

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

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

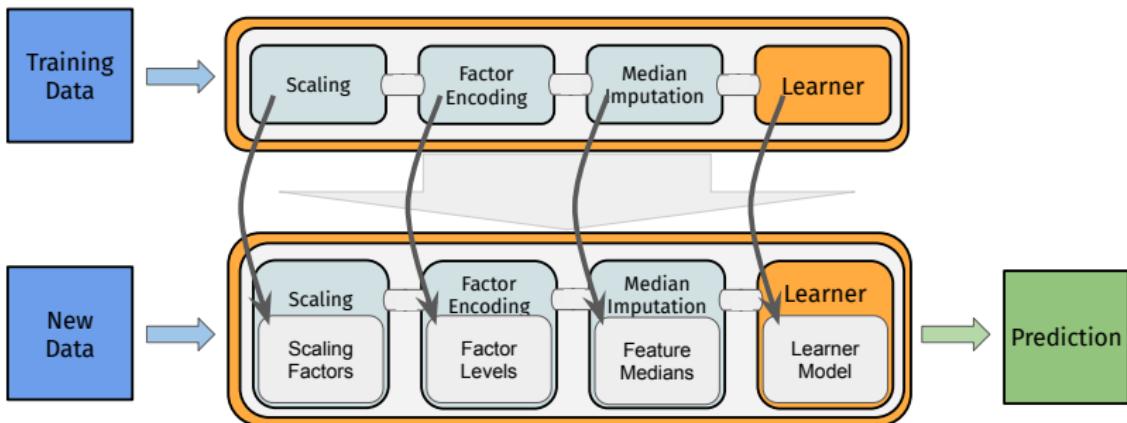


# MLR3PIPELINES IN ACTION

## Linear Preprocessing Pipeline

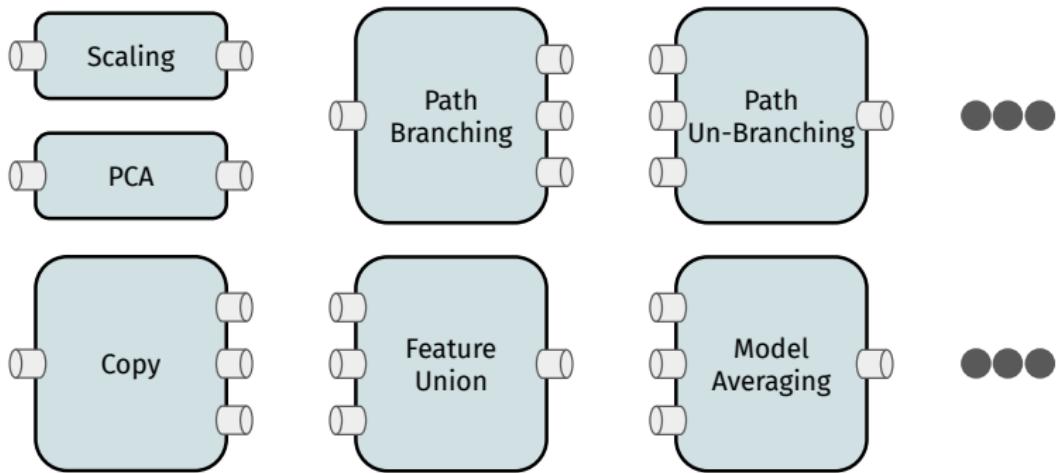
- `train()`ing: Data propagates and creates `$states`
- `predict()`ition: Data propagates, uses `$states`

```
glrn$predict(task)
```



# **Nonlinear Pipelines**

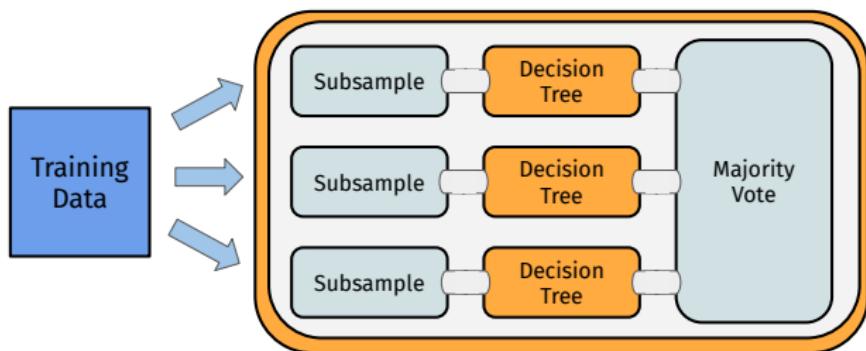
# PIPEOPS WITH MULTIPLE INPUTS / OUTPUTS



# MLR3PIPELINES IN ACTION

## Ensemble Method: Bagging

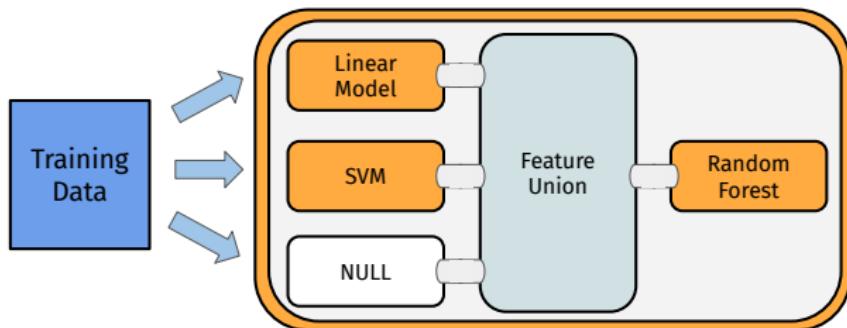
```
single_path = po("subsample") %>>% lrn("classif.rpart")
graph_bag = pipeline_greplicate(single_path, n = 3) %>>%
 po("classifavg")
```



# 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")
```

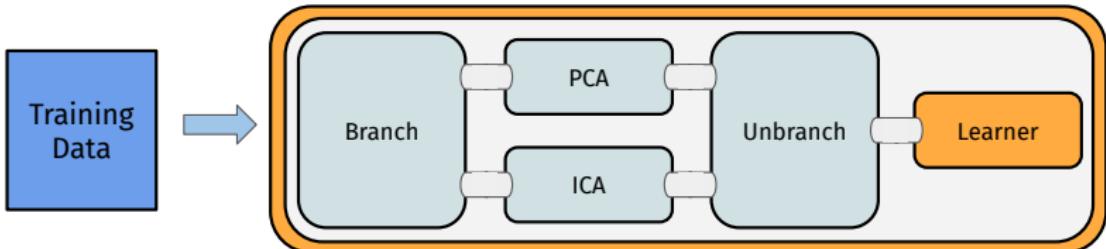


# MLR3PIPELINES IN ACTION

## Branching

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

Execute only one of several alternative paths

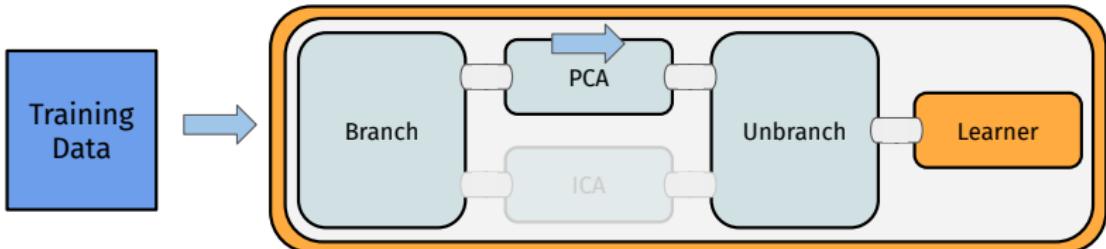


# MLR3PIPELINES IN ACTION

## Branching

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

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

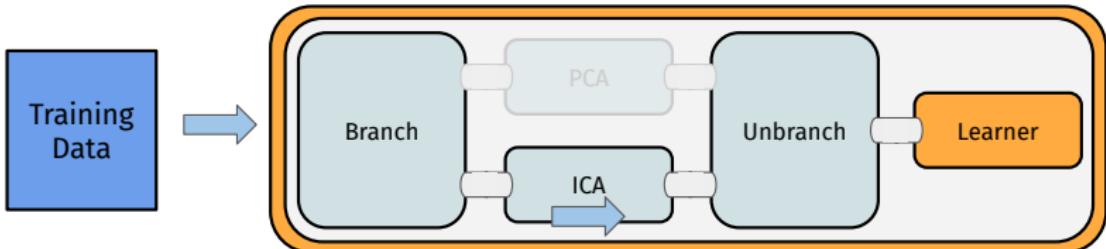


# MLR3PIPELINES IN ACTION

## Branching

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

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



# “PIPELINES” DICTIONARY & SHORT FORM

Many frequently used *patterns* for pipelines

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

# **AutoML with mlr3pipelines**

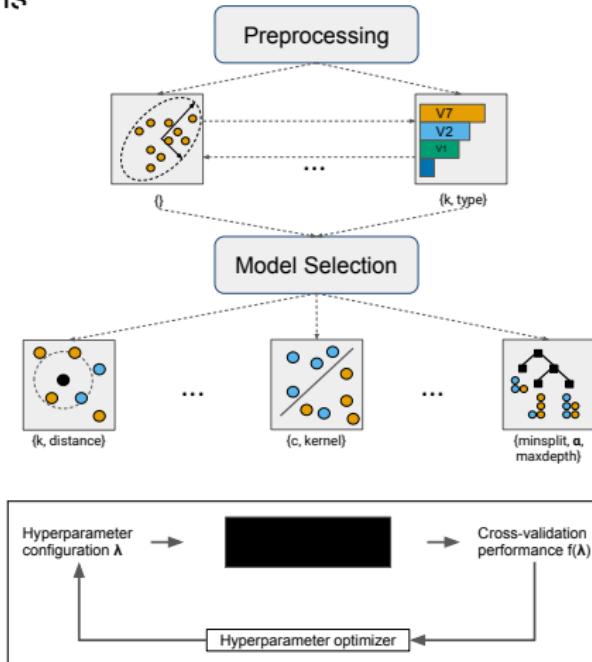
# AUTOML <3 PIPELINES

- AutoML: Automatic Machine Learning
- Let the algorithm make decisions about
  - ➊ *what learner* to use,
  - ➋ *what preprocessing* to use, and
  - ➌ *what hyperparameters* to use.

# AUTOML WITH MLR3PIPELINES

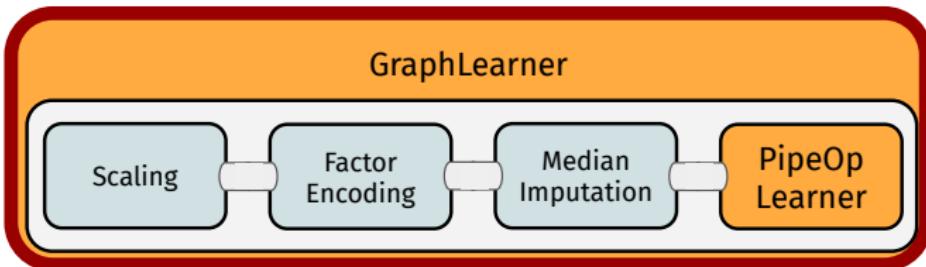
## AutoML in a Nutshell

- Preprocessing steps
- ML Algorithms
- Tuner



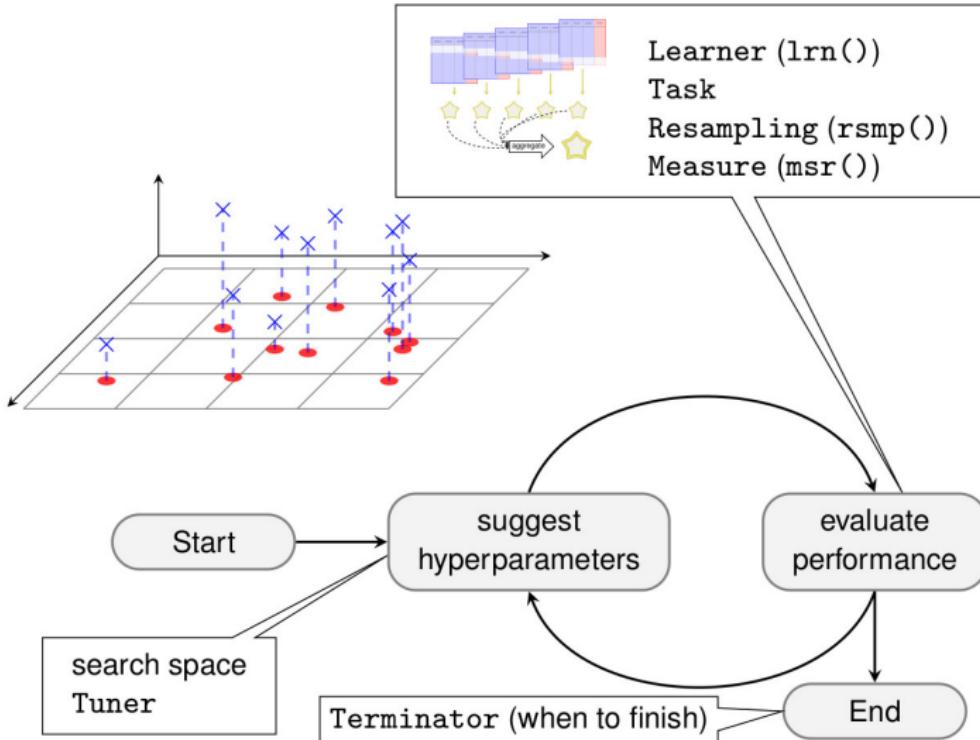
# GRAPHLEARNER

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



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

# TUNING

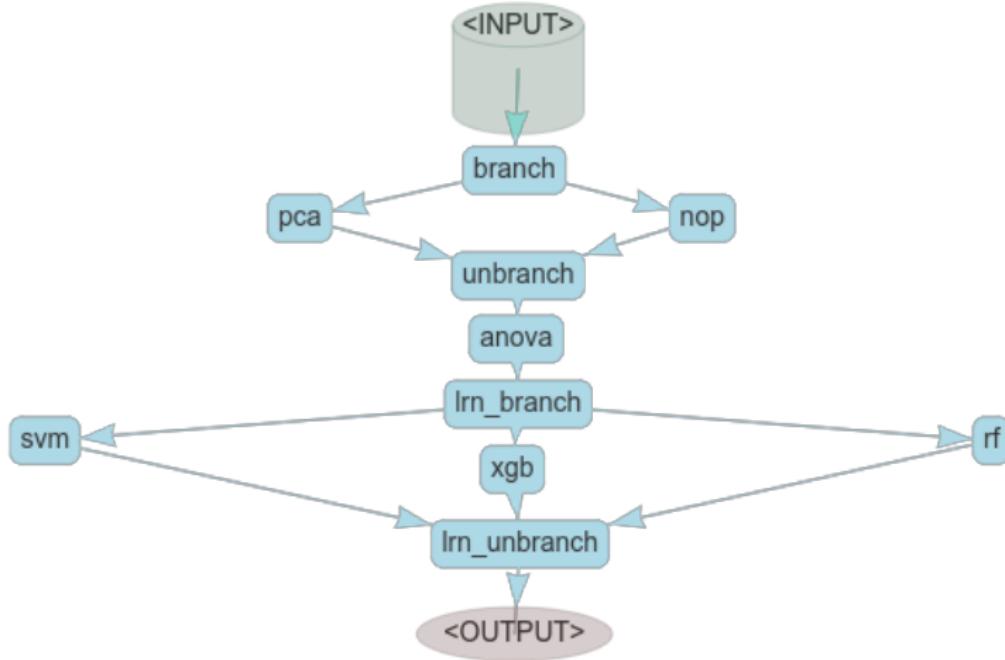


# PIPELINES TUNING

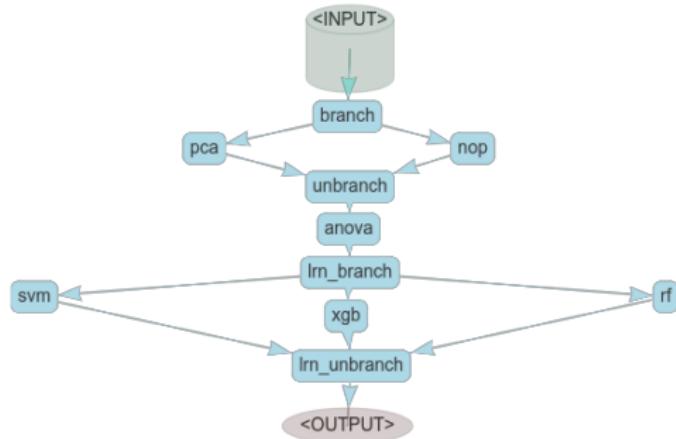
- Works **exactly** as in basic `mlr3` / `mlr3tuning`
- PipeOps have *hyperparameters* (using `paradox` pkg)
- Graphs have hyperparameters of all components *combined*
- ⇒ Joint **tuning** and nested CV of complete graph

```
p1 = ppl("branch", list(
 "pca" = po("pca"),
 "nothing" = po("nop")))
p2 = flt("anova")
p3 = ppl("branch", list(
 "svm" = lrn("classif.svm", id = "svm"),
 "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



```
glrn$param_set$values = list(
 branch.selection = to_tune(),
 anova.filter.frac = to_tune(0.1, 1),
 lrn_branch.selection = to_tune(),
 rf.mtry = to_tune(1, 20),
 xgb.nrounds = to_tune(1, 500),
 svm.cost = to_tune(p_dbl(-12, 4, trafo = function(x) 2^x)),
 svm.gamma = to_tune(p_dbl(-12, -1, trafo = function(x) 2^x)),
 xgb.verbose = 0, svm.type = "C-classification", svm.kernel = "radial"
)
```

# PIPELINES TUNING

```
glnr$param_set$values = list(
 branch.selection = to_tune(),
 anova.filter.frac = to_tune(0.1, 1),
 lrn_branch.selection = to_tune(),
 rf.mtry = to_tune(1, 20),
 xgb.nrounds = to_tune(1, 500),
 svm.cost = to_tune(p_dbl(-12, 4, trafo = function(x) 2^x)),
 svm.gamma = to_tune(p_dbl(-12, -1, trafo = function(x) 2^x)),
 xgb.verbose = 0, svm.type = "C-classification", svm.kernel = "radial"
)

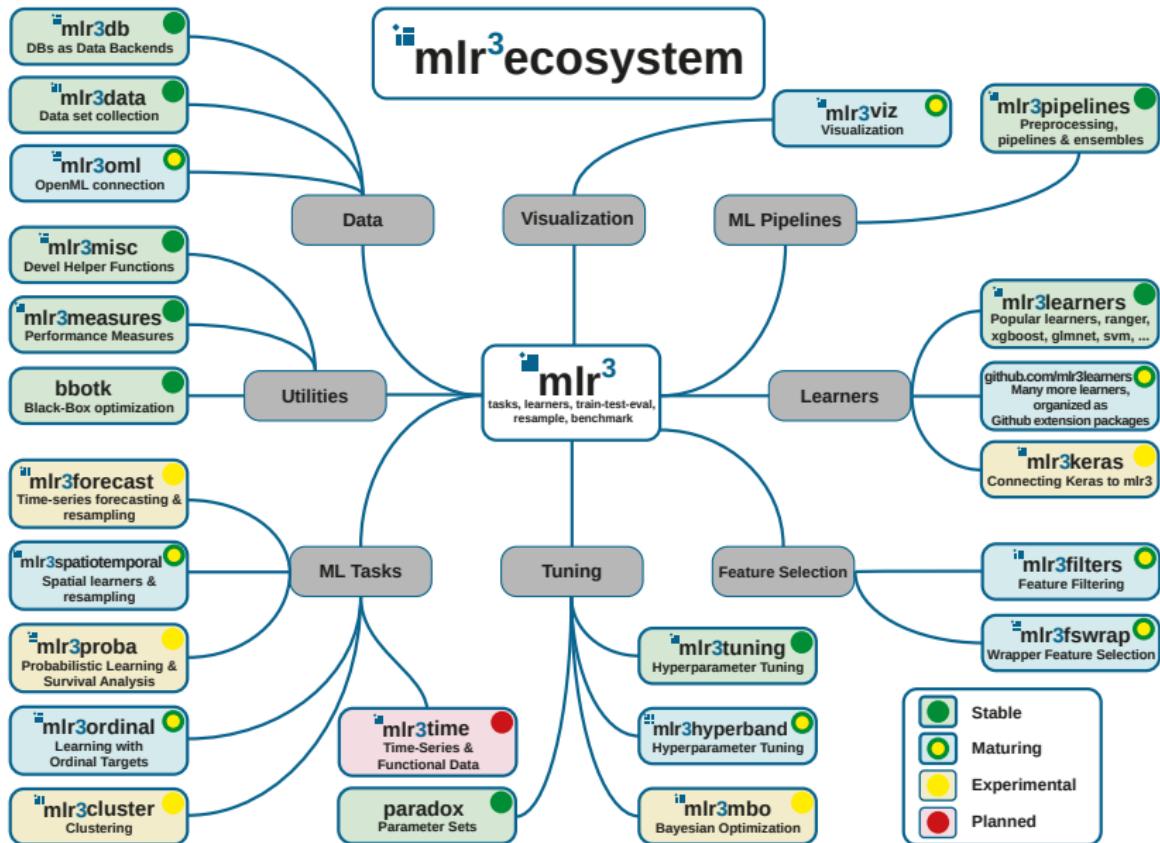
inst = TuningInstanceSingleCrit$new(tsk("sonar"), glnr,
 rsmp("cv", folds = 3), msr("classif.ce"), trm("evals", n_evals = 10))
tnr("random_search")$optimize(inst)
```

# 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!



# MLR3(PIPELINES) RESOURCES

## mlr3 book

The screenshot shows a browser window with the URL <https://book.mlr-org.com/pipelines.html>. The page title is "4 Pipelines". The content discusses mlr3pipelines as a pipeline programming toolkit, mentioning PipeOps and PipeLearner. It includes a diagram of a pipeline with four steps: Scaling, Factor Encoding, Median Imputation, and Learner. Below the diagram, it says "Single computational steps can be represented as so-called PipeOps, which can then be connected with directed edges in a graph. The scope of mlr3pipelines is still growing. Currently supported features are: \* Pipelines".

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

## mlr3 Use Case “Gallery”

The screenshot shows a browser window with the URL <https://gallery.mlr-org.com>. It lists several use cases:

- mlr3 and OpenML - Moneyball use case**: This use case shows how to make use of OpenML data and how to impute missing values in a ML problem.
- A pipeline for the titanic data set - Advanced**: This post shows how to build a Graph using the mlr3pipelines package on the "titanic" dataset. Moreover, feature engineering, data imputation and benchmarking are covered.
- Tuning a stacked learner**: This tutorial explains how to create and tune a multilevel classifier model using the mlr3pipelines package.

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

## “cheat sheets”

The image displays three "CHEAT SHEET" documents for machine learning with mlr3:

- Machine learning with mlr3 :: CHEAT SHEET**: This sheet covers basic tasks like Train & Predict, Class Task, Class Learner, and Model Interpretation. It includes sections on Hyperparameter Tuning with mlr3tuning, Dataflow programming with mlr3pipelines, and Feature Engineering.
- Hyperparameter Tuning with mlr3tuning :: CHEAT SHEET**: This sheet provides a quick reference for hyperparameter tuning, including terminologies like "Tuner", "Search Strategy", and "AutoTuner", and how-to guides for "Tuning", "Tuning before", and "Tuning after".
- Dataflow programming with mlr3pipelines :: CHEAT SHEET**: This sheet covers Dataflow basics, Graph construction, and Training. It includes examples for "Cross-validation", "Parallel", "Pipeline", "Flowchart", and "Graph".
- Feature Engineering :: CHEAT SHEET**: This sheet provides a quick reference for feature engineering, including Linear Models, Feature Selection, and Feature Extraction. It includes examples for "Example - Feature Selection" and "Example - Preprocessing".

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

**Outro**

# MLR3PIPELINES

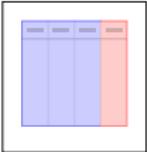
mlr3pipelines overview:

- 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

# OVERVIEW

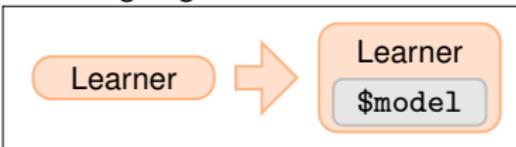
Ingredients:

Data



TaskClassif,  
TaskRegr,  
tsk()

Learning Algorithms



lrn() ⇒ Learner,  
\$train(),  
\$predict() ⇒ Prediction

Performance Evaluation



rsmp() ⇒ Resampling,  
msr() ⇒ Measure,  
resample() ⇒ ResamplingResult,  
\$aggregate()

Performance Comparison



benchmark\_grid(),  
benchmark() ⇒ BenchmarkResult

# RECAP

```
inst = TuningInstanceSingleCrit$new(
 tsk("iris"), lrn("classif.kknn", kernel="rectangular"),
 rsmp("holdout"), msr("classif.ce"), trm("evals", n_evals = 2),
 ps(k = p_int(1, 20))
)

gsearch = tnr("grid_search", resolution = 3)

gsearch$optimize(inst)

#> k learner_param_vals x_domain classif.ce
#> 1: 1 <list[2]> <list[1]> 0.04
```