Modern Machine Learning in R

mlr3

Michel Lang, Bernd Bischl, Jakob Richter, Patrick Schratz, Martin Binder

September 27, 2019

Intro

So you want to do ML in R

CRAN serves hundreds of packages for machine learning.
Instead of having to dig into everyone just use:

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

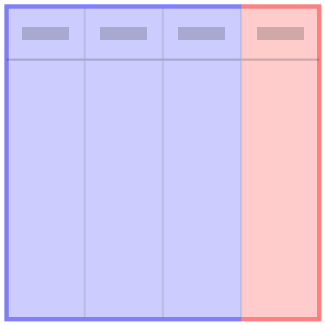
Ingredients:

- Data
- Learning Algorithms
- Performance Evaluation
- Performance Comparison

Data

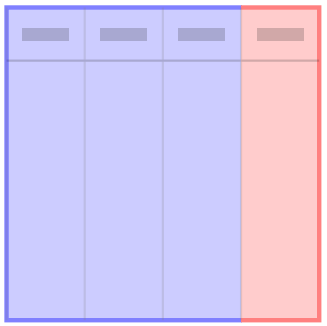
Data

- Tabular data



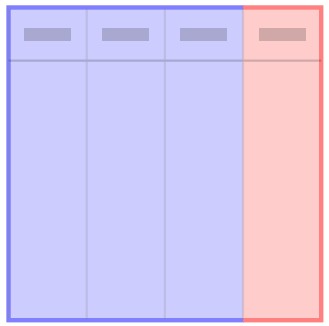
Data

- Tabular data
- Features



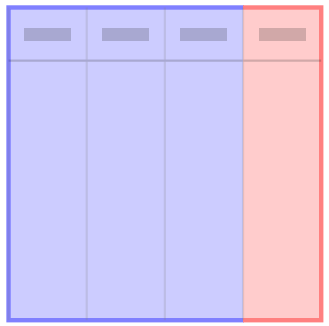
Data

- Tabular data
- Features
- Target / outcome to predict



Data

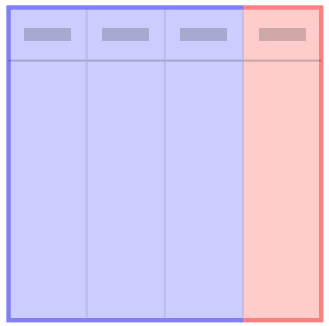
- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression



Data

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

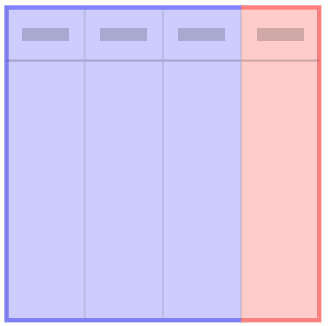
  $\Rightarrow$ data determines the machine learning "Task"



Data

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

  $\Rightarrow$ data determines the machine learning "Task"

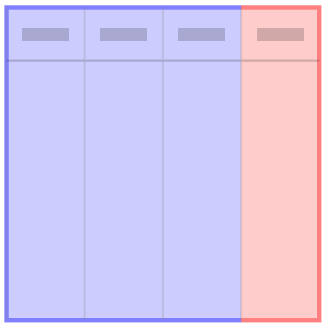


```
print(iris)  # included in R

#>   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

  $\Rightarrow$ data determines the machine learning "Task"

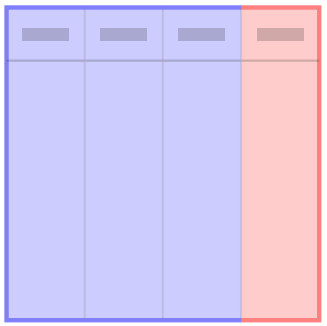


```
print(iris)  # included in R

#>   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

  $\Rightarrow$ data determines the machine learning "Task"



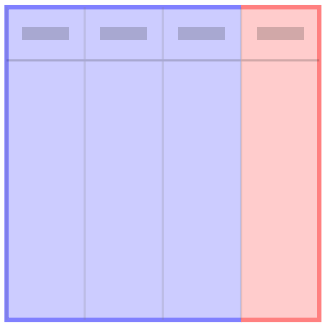
```
print(iris)  # included in R

#>   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

  $\Rightarrow$ data 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
#> ...
```

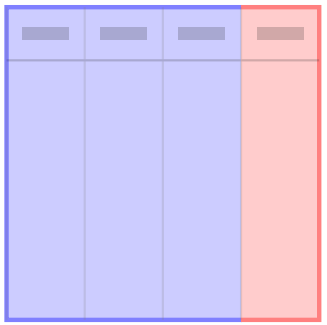
Task ID

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

Data

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

  $\Rightarrow$ data 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
#> ...
```

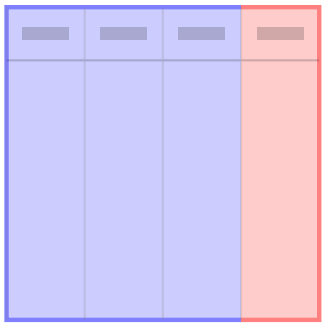
Task ID ↘ data ↓

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

Data

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

  $\Rightarrow$ data 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
```

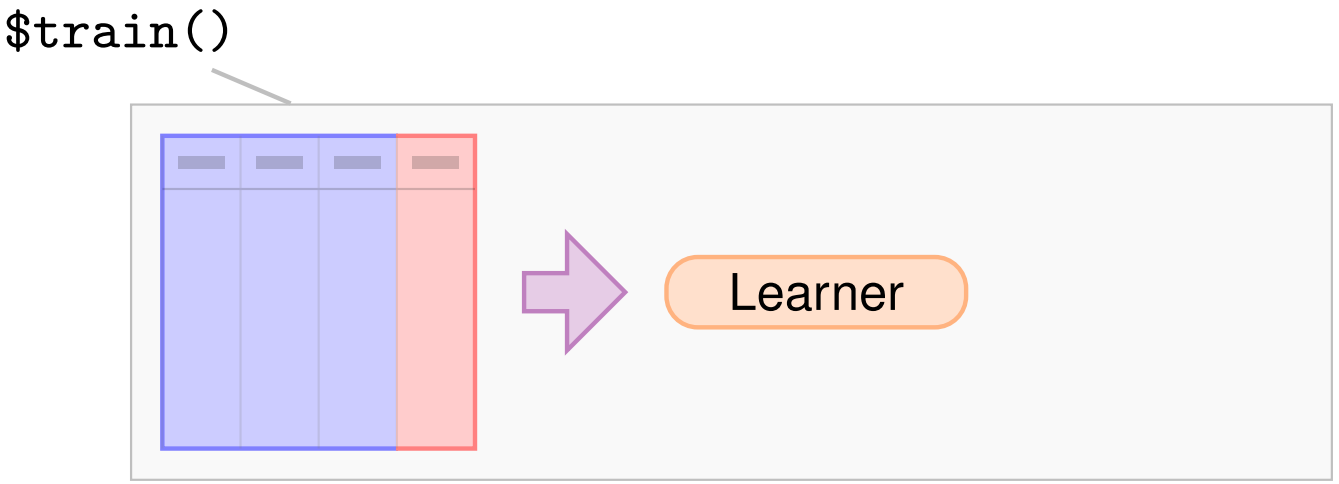
```
task$ncol
task$nrow
task$feature_names
task$target_names
```

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

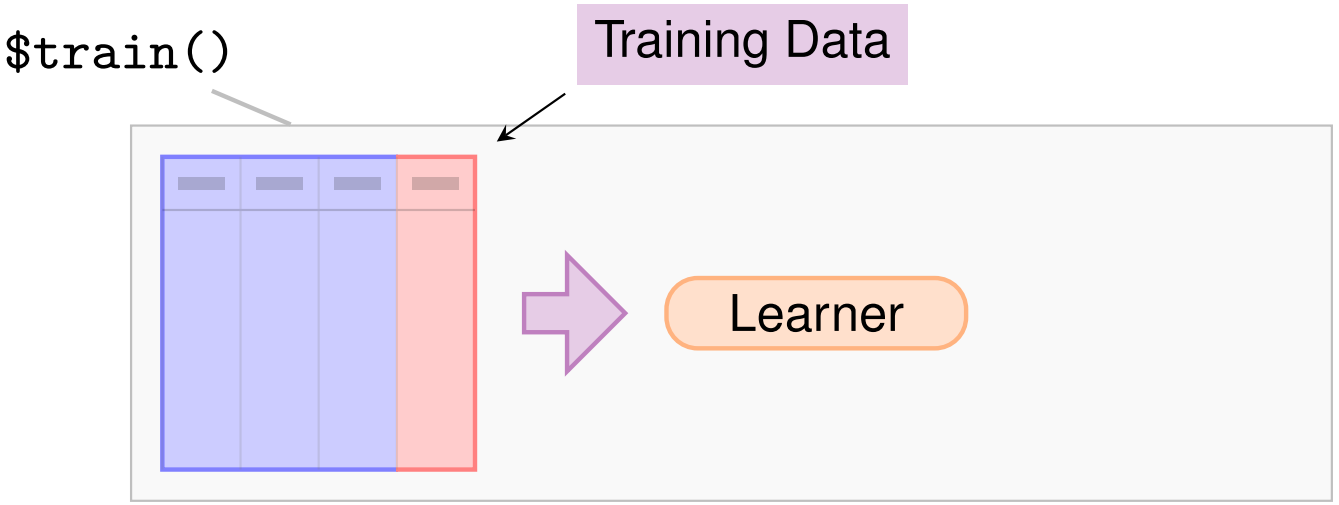
```
task$select(cols = )
task$filter(rows = )
task$cbind(data = )
task$rbind(data = )
```

Learning Algorithms

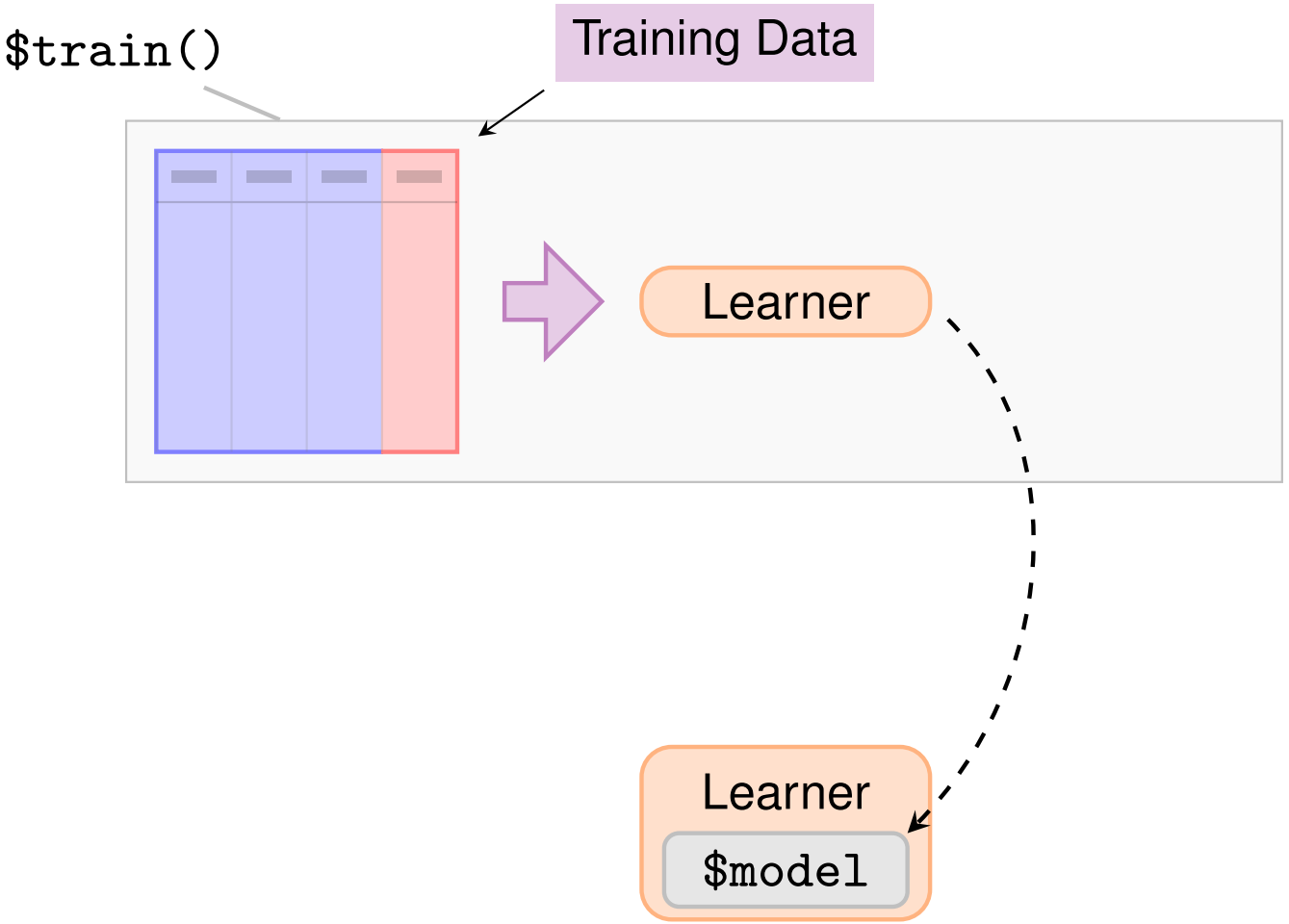
Learning Algorithms



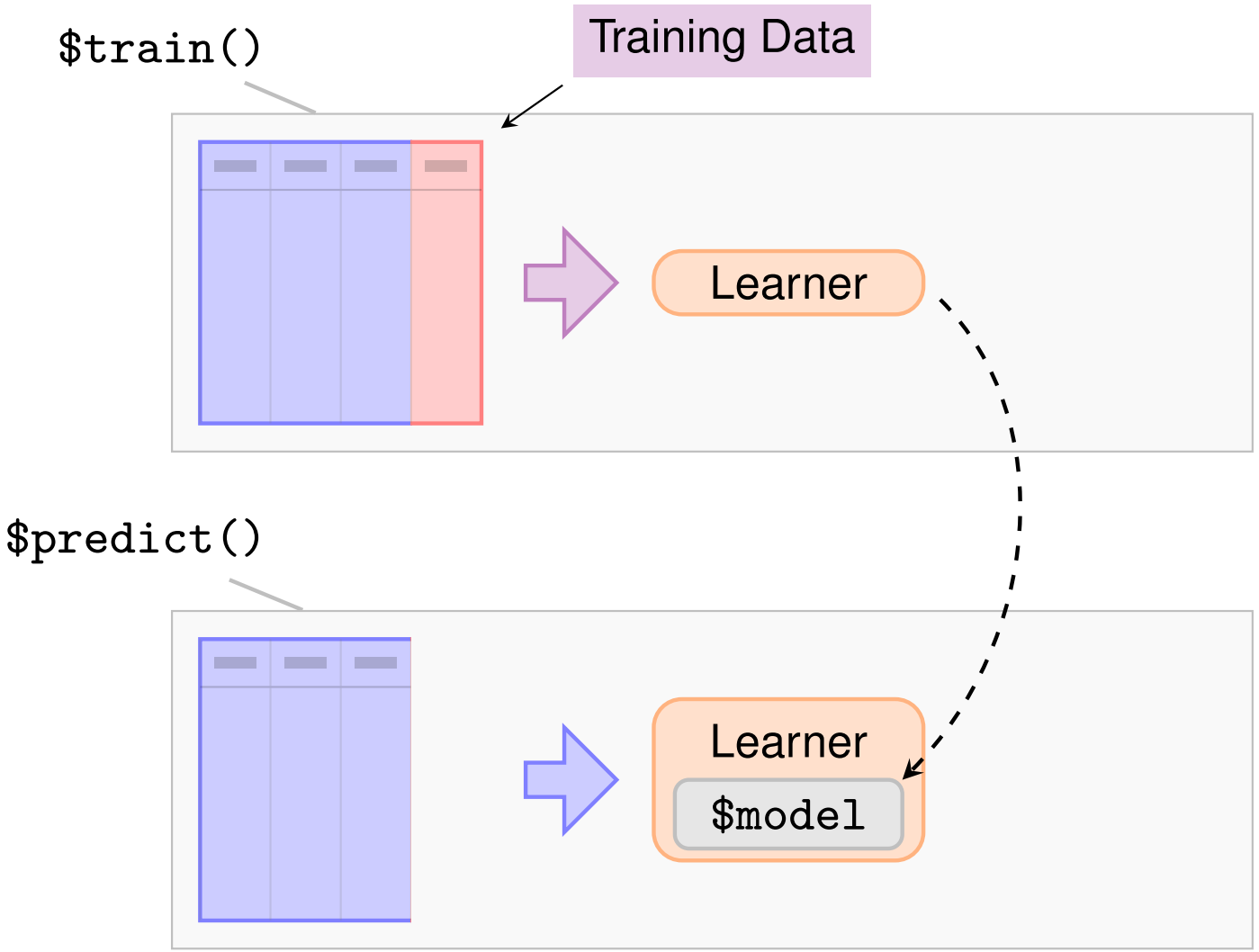
Learning Algorithms



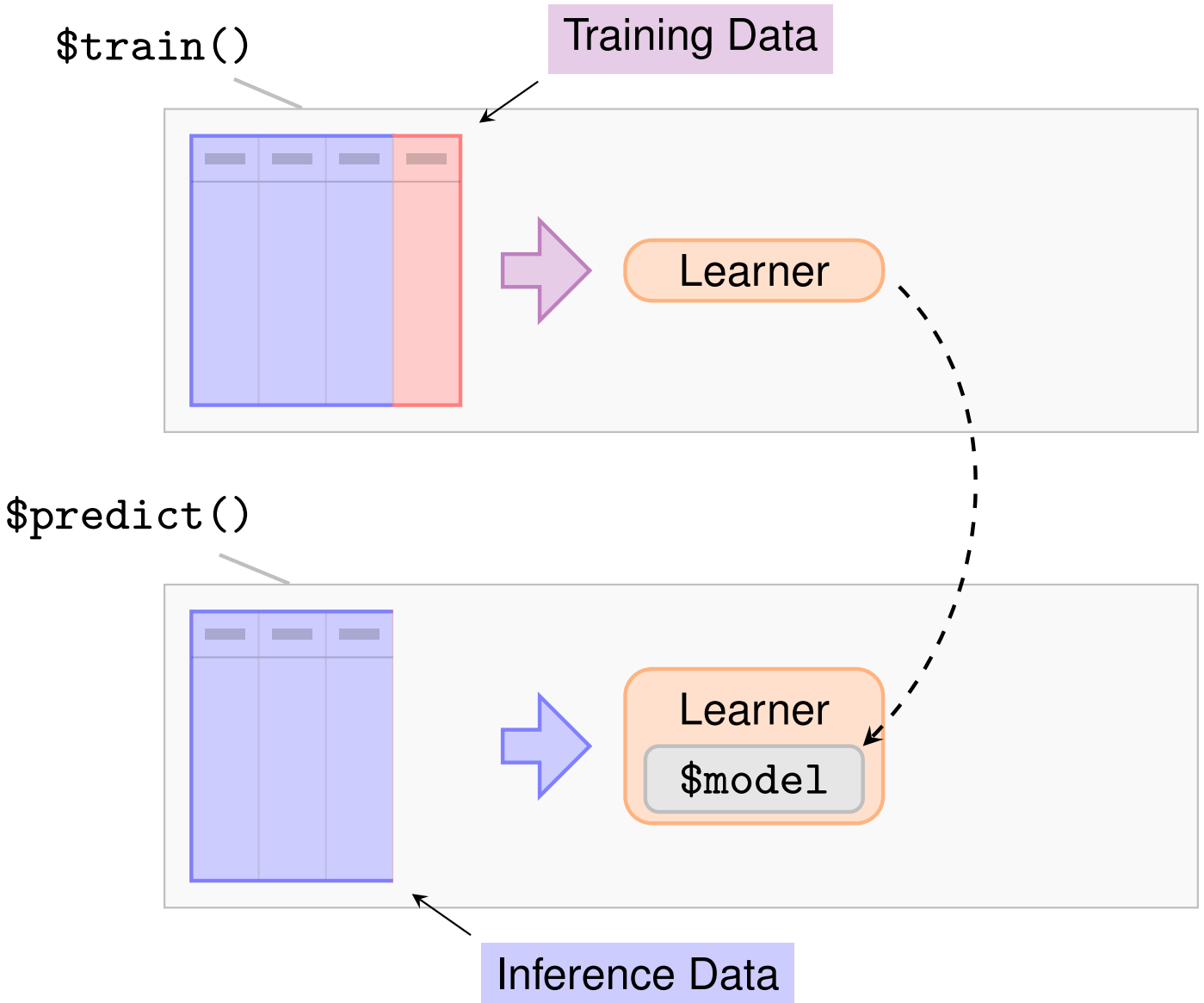
Learning Algorithms



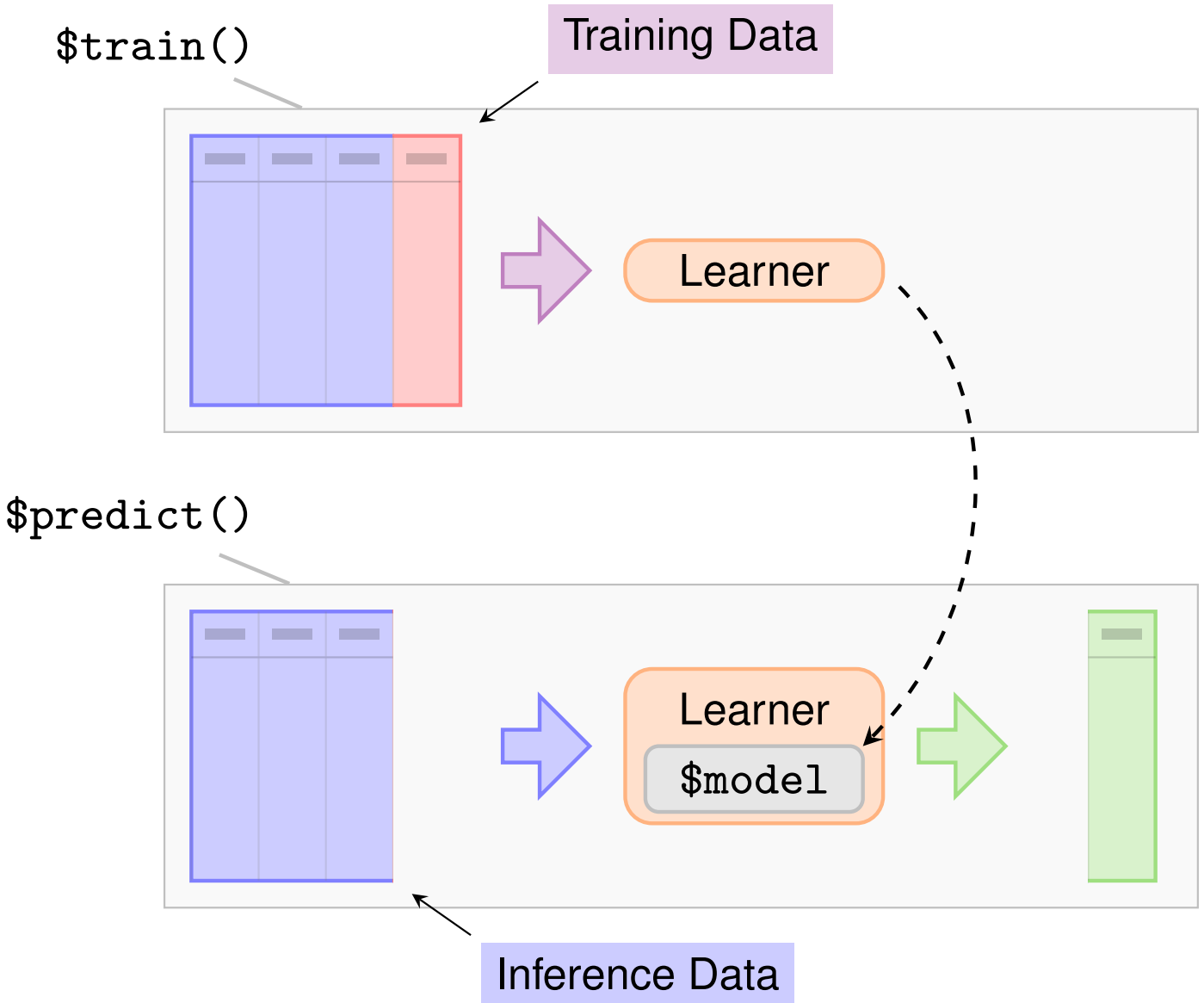
Learning Algorithms



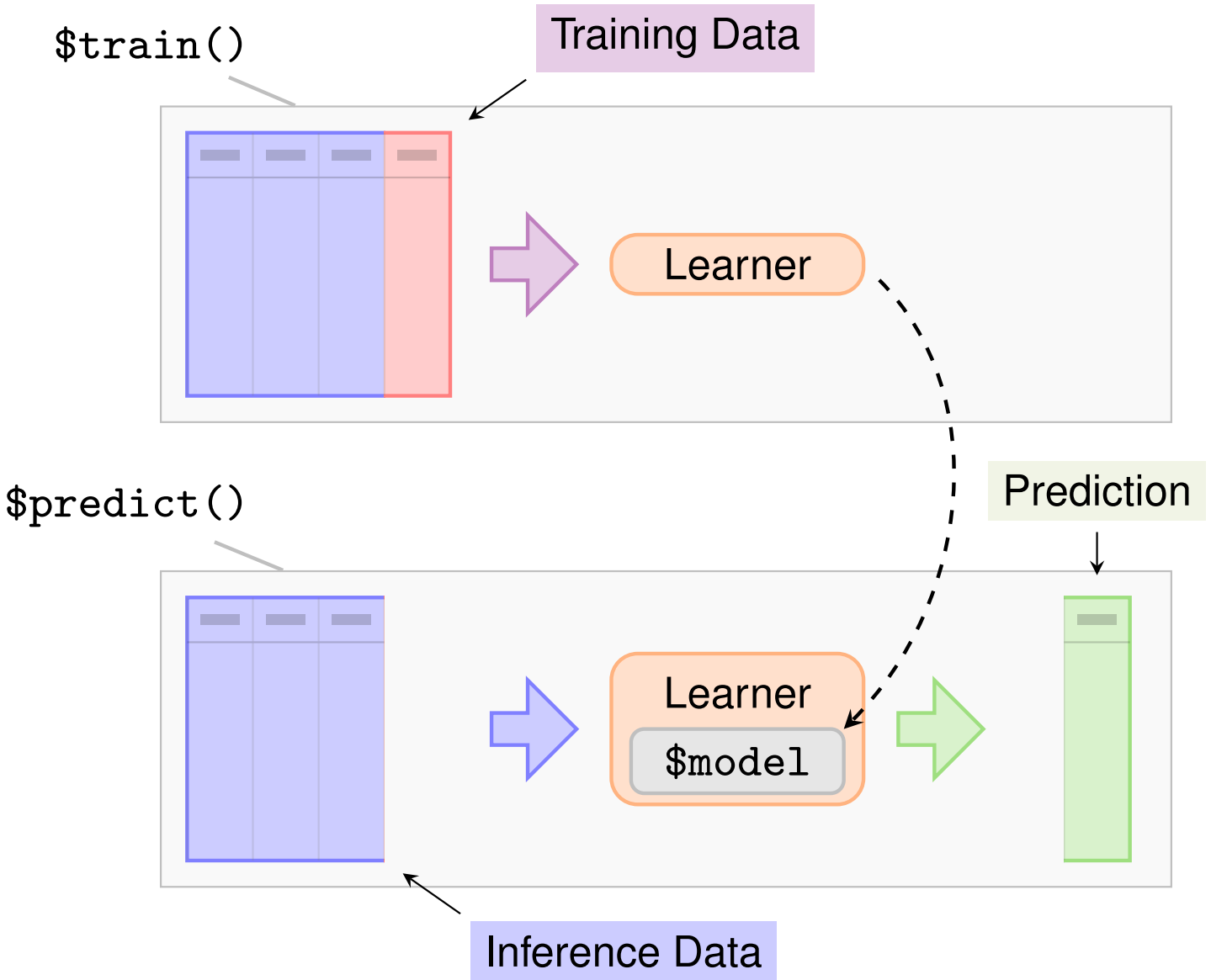
Learning Algorithms



Learning Algorithms



Learning Algorithms



Learning Algorithms

- Get a `Learner` provided by `mlr3`

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

Learning Algorithms

- Get a `Learner` provided by `mlr3`

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

- Train the `Learner`

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

Learning Algorithms

- Get a `Learner` provided by `mlr3`

```
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) *
```

Hyperparameters

- Learners have *hyperparameters*

```
learner$param_set

#> ParamSet:
#>              id    class lower upper levels default  value
#>          <char>   <char> <num> <num> <list>  <list> <list>
#> 1:     minsplit ParamInt     1   Inf             20
#> 2:           cp ParamDbl     0     1           0.01
#> 3:   maxcompete ParamInt     0   Inf              4
#> 4: maxsurrogate ParamInt     0   Inf              5
#> 5:     maxdepth ParamInt     1    30             30
#> 6:         xval ParamInt     0   Inf             10      0
```

Hyperparameters

- Learners have *hyperparameters*

```
learner$param_set

#> ParamSet:
#>              id    class lower upper levels default  value
#>          <char>   <char> <num> <num> <list>  <list> <list>
#> 1:     minsplit ParamInt     1   Inf             20
#> 2:           cp ParamDbl     0     1           0.01
#> 3:   maxcompete ParamInt     0   Inf              4
#> 4: maxsurrogate ParamInt     0   Inf              5
#> 5:     maxdepth ParamInt     1    30             30
#> 6:         xval ParamInt     0   Inf             10      0
```

- Changing them changes the `Learner` behavior

```
learner$param_set$values = list(maxdepth = 1)

learner$train(task)
```

Hyperparameters

- This gives a smaller decision tree

```
print(learner$model)

#> n= 150
#>
#> node), split, n, loss, yval, (yprob)
#>       * denotes terminal node
#>
#> 1) root 150 100 setosa (0.33 0.33 0.33)
#>   2) Petal.Length< 2.5 50   0 setosa (1.00 0.00 0.00) *
#>   3) Petal.Length>=2.5 100  50 versicolor (0.00 0.50 0.50) *
```

Hyperparameters

- This gives a smaller decision tree

```
print(learner$model)

#> n= 150
#>
#> node), split, n, loss, yval, (yprob)
#>       * denotes terminal node
#>
#> 1) root 150 100 setosa (0.33 0.33 0.33)
#>   2) Petal.Length< 2.5 50   0 setosa (1.00 0.00 0.00) *
#>   3) Petal.Length>=2.5 100  50 versicolor (0.00 0.50 0.50) *
```

- Instead of assigning `$values` a `list()`, we can change individual parameters

```
learner$param_set$values$maxdepth = 10
```

Prediction

- Lets make a prediction

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

Prediction

- Lets make a prediction

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

- Call `$predict_newdata()` with the data and the old **Task**

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

Prediction

- Lets make a prediction

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

- Call `$predict_newdata()` with the data and the old **Task**

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

- We get a `Prediction` object:

```
prediction
#> <PredictionClassif> for 2 observations:
#>  row_id truth   response
#>     151  <NA>     setosa
#>     152  <NA> versicolor
```

Prediction

- Lets make a prediction

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

- Call `$predict_newdata()` with the data and the old **Task**

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

- We get a `Prediction` object:

```
prediction
#> <PredictionClassif> for 2 observations:
#>  row_id truth  response
#>     151  <NA>     setosa
#>     152  <NA> versicolor
```

Prediction

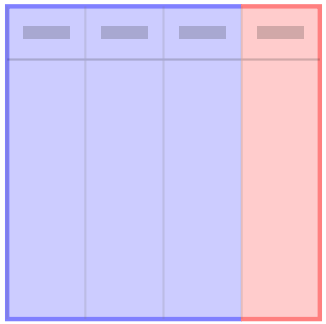
- We can make the `Learner` predict *probabilities* when we set `predict_type`:

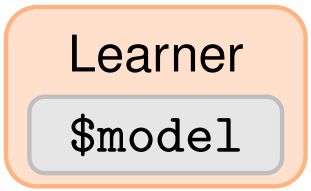
```
learner$predict_type = "prob"
learner$predict_newdata(task, new_data)
```

```
# <PredictionClassif> for 2 observations:
#  row_id truth  response prob.setosa prob.versicolor
#     151  <NA>    setosa           1             0.0
#     152  <NA> virginica           0             0.5
#  prob.virginica
#             0.0
#             0.5
```

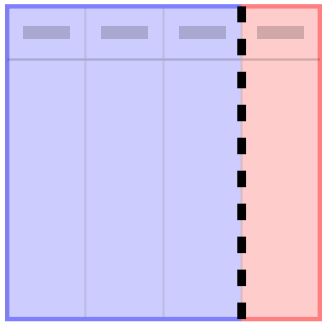
Performance

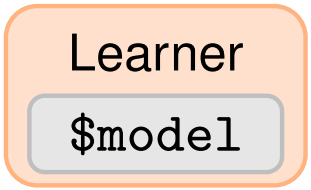
Performance Evaluation



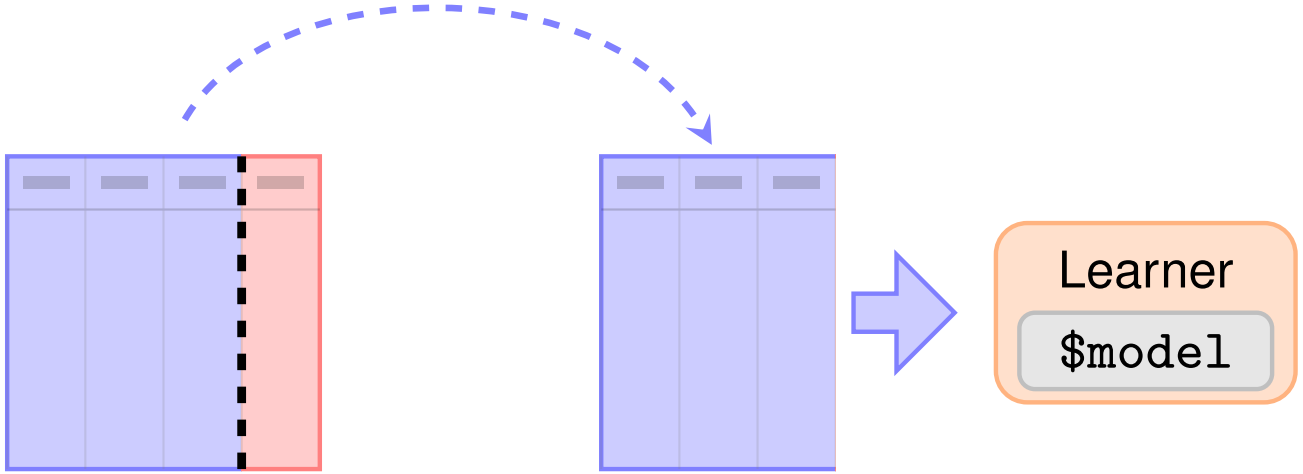


Performance Evaluation

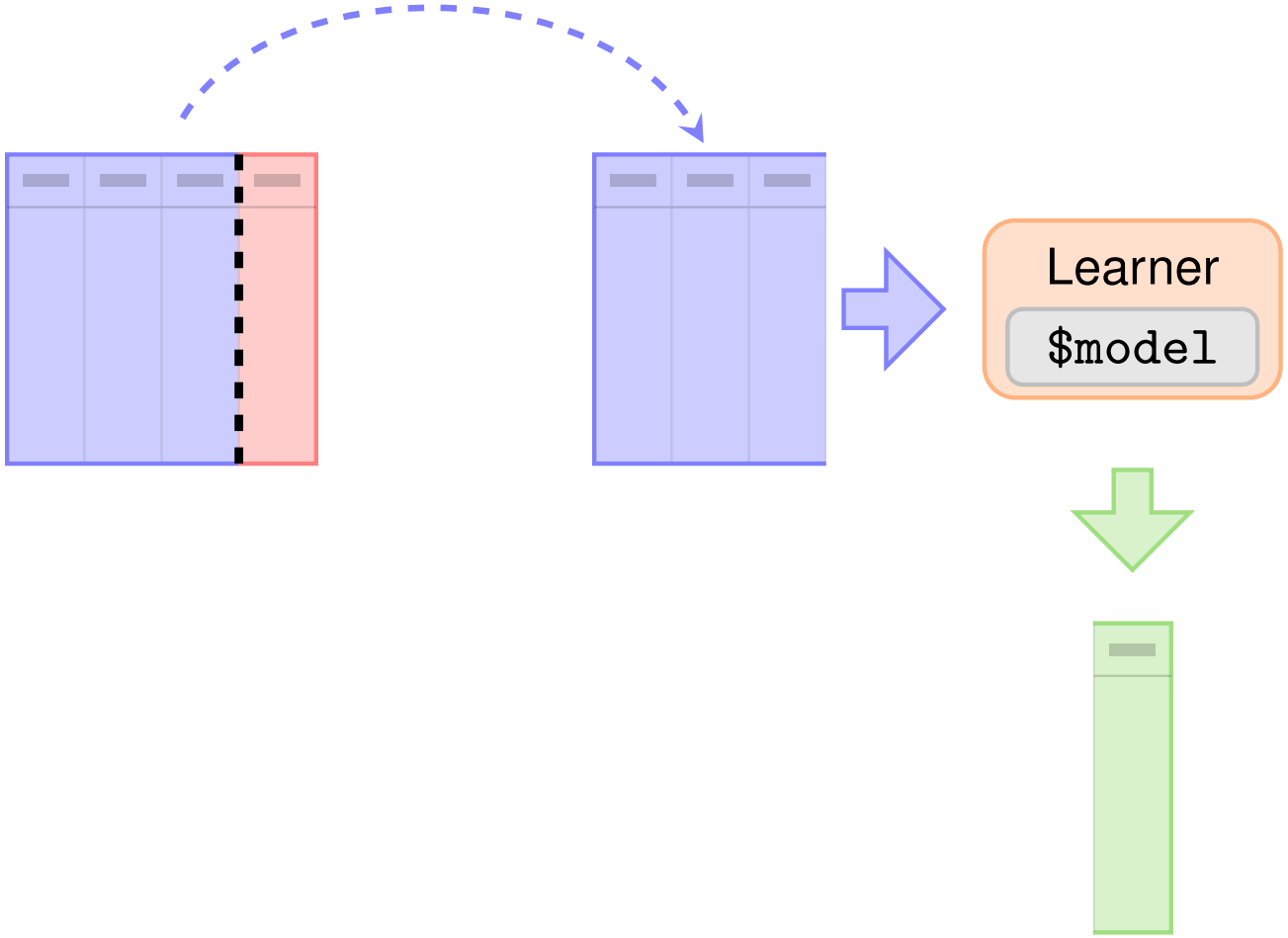




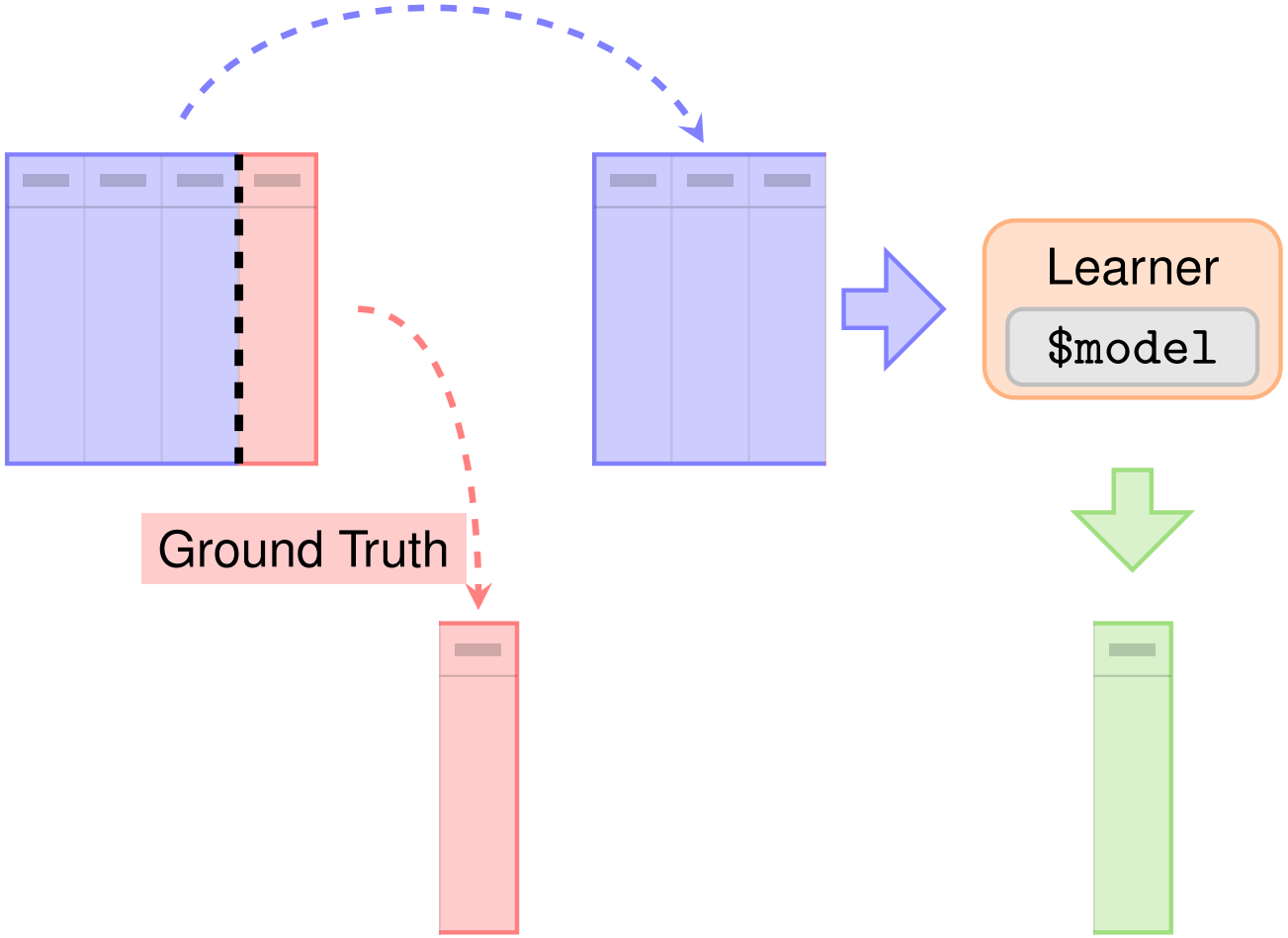
Performance Evaluation



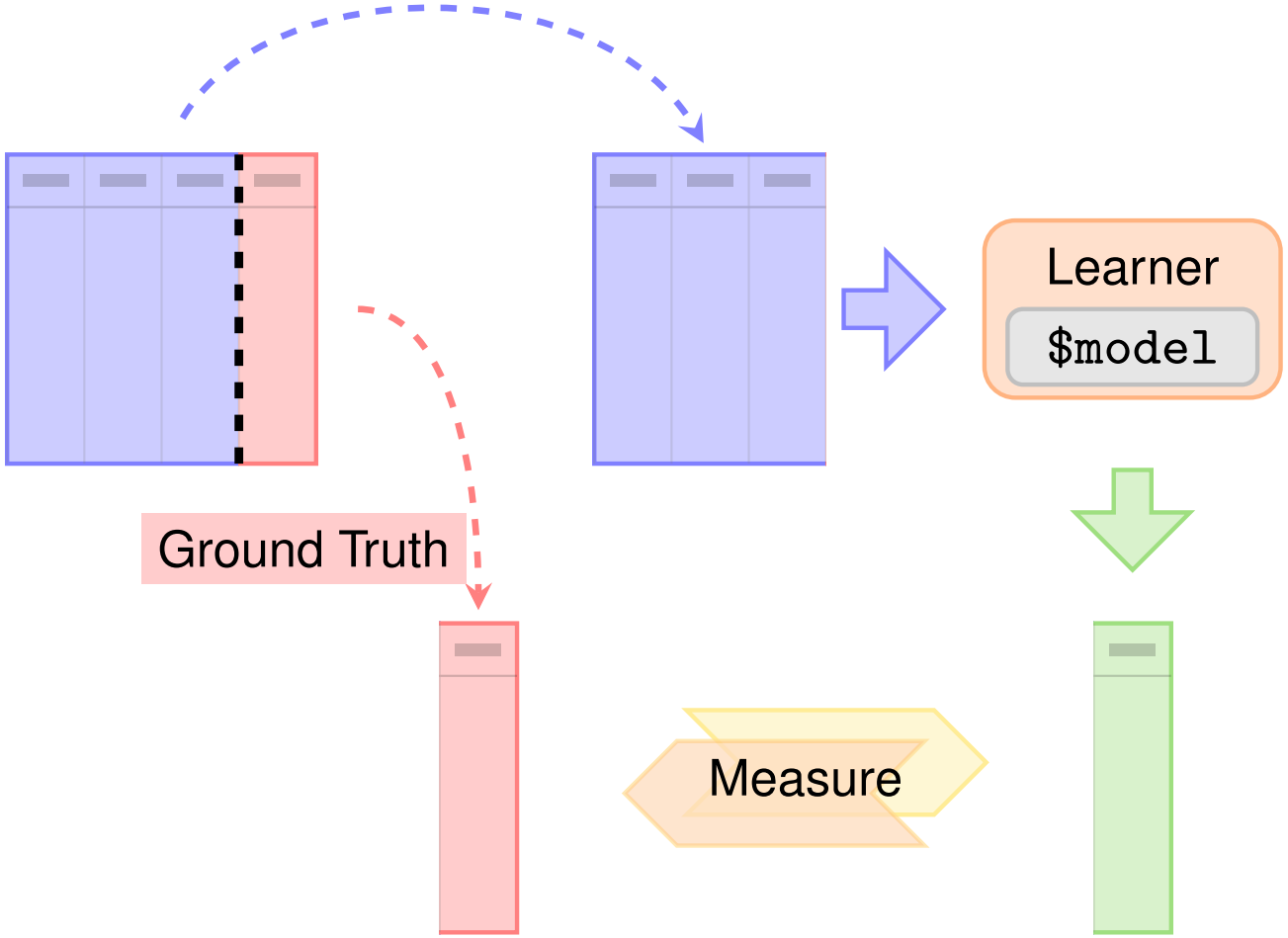
Performance Evaluation



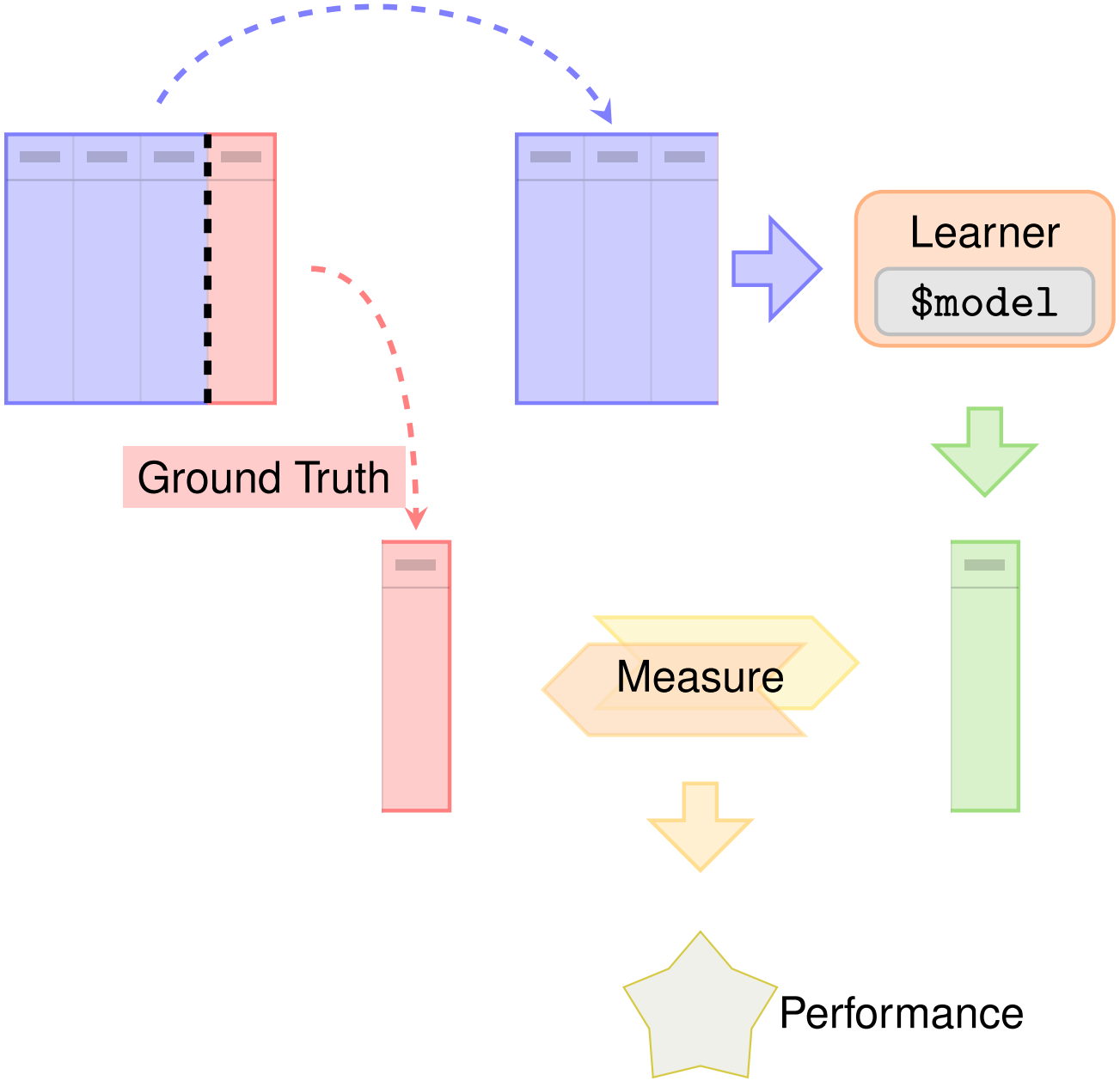
Performance Evaluation



Performance Evaluation



Performance Evaluation



Performance Evaluation

- Prediction 'Task' with known data

```
known_truth_task$data()
#    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#     <fctr>        <num>       <num>        <num>       <num>
# 1:  setosa            2           1            4           3
# 2:  setosa            3           2            2           2
```

Performance Evaluation

- Prediction 'Task' with known data

```
known_truth_task$data()
#    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#     <fctr>        <num>       <num>        <num>       <num>
# 1:  setosa            2           1            4           3
# 2:  setosa            3           2            2           2
```

- Predict again

```
pred = learner$predict(known_truth_task)
pred
#> <PredictionClassif> for 2 observations:
#>  row_id  truth  response
#>       1 setosa    setosa
#>       2 setosa virginica
```

Performance Evaluation

- Prediction 'Task' with known data

```
known_truth_task$data()
#    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#     <fctr>        <num>       <num>        <num>       <num>
# 1:  setosa            2           1            4           3
# 2:  setosa            3           2            2           2
```

- Predict again

```
pred = learner$predict(known_truth_task)
pred
#> <PredictionClassif> for 2 observations:
#>  row_id  truth  response
#>       1 setosa    setosa
#>       2 setosa virginica
```

- Score the prediction

```
pred$score(msr("classif.ce"))
#> classif.ce
#>        0.5
```

Performance Evaluation

- Prediction 'Task' with known data

```
known_truth_task$data()
#    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#     <fctr>        <num>       <num>        <num>       <num>
# 1:  setosa            2           1            4           3
# 2:  setosa            3           2            2           2
```

- Predict again

```
pred = learner$predict(known_truth_task)
pred
#> <PredictionClassif> for 2 observations:
#>  row_id  truth  response
#>       1 setosa    setosa
#>       2 setosa virginica
```

- Score the prediction

```
pred$score(msr("classif.ce"))
#> classif.ce
#>        0.5
```

Performance Evaluation

- Confusion Matrix

```
pred
```

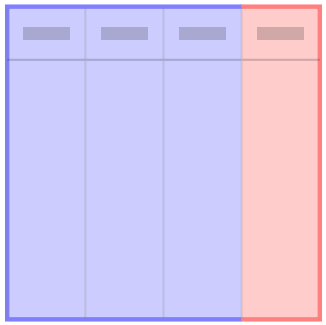
```
#> <PredictionClassif> for 2 observations:
#>  row_id  truth  response
#>       1 setosa    setosa
#>       2 setosa virginica
```

```
pred$confusion
```

```
#>             truth
#> response     setosa versicolor virginica
#>   setosa          1          0         0
#>   versicolor      0          0         0
#>   virginica       1          0         0
```

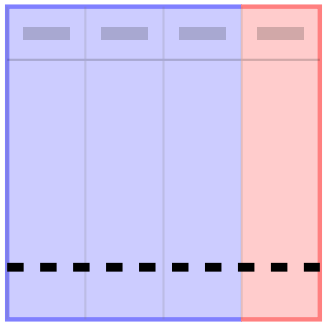
Resampling

Resampling



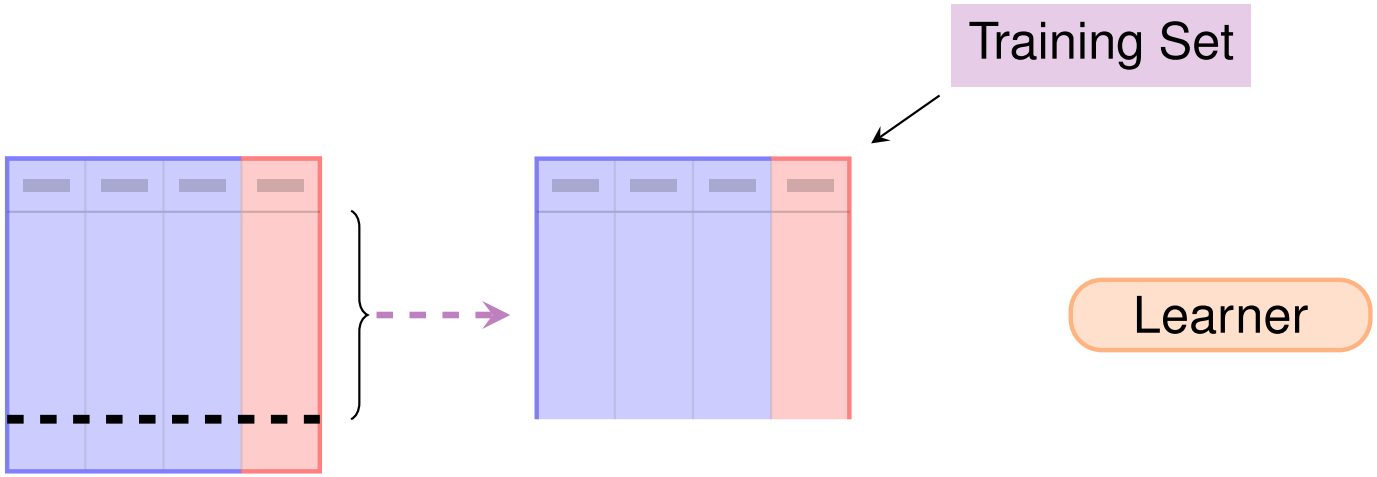
Learner

Resampling

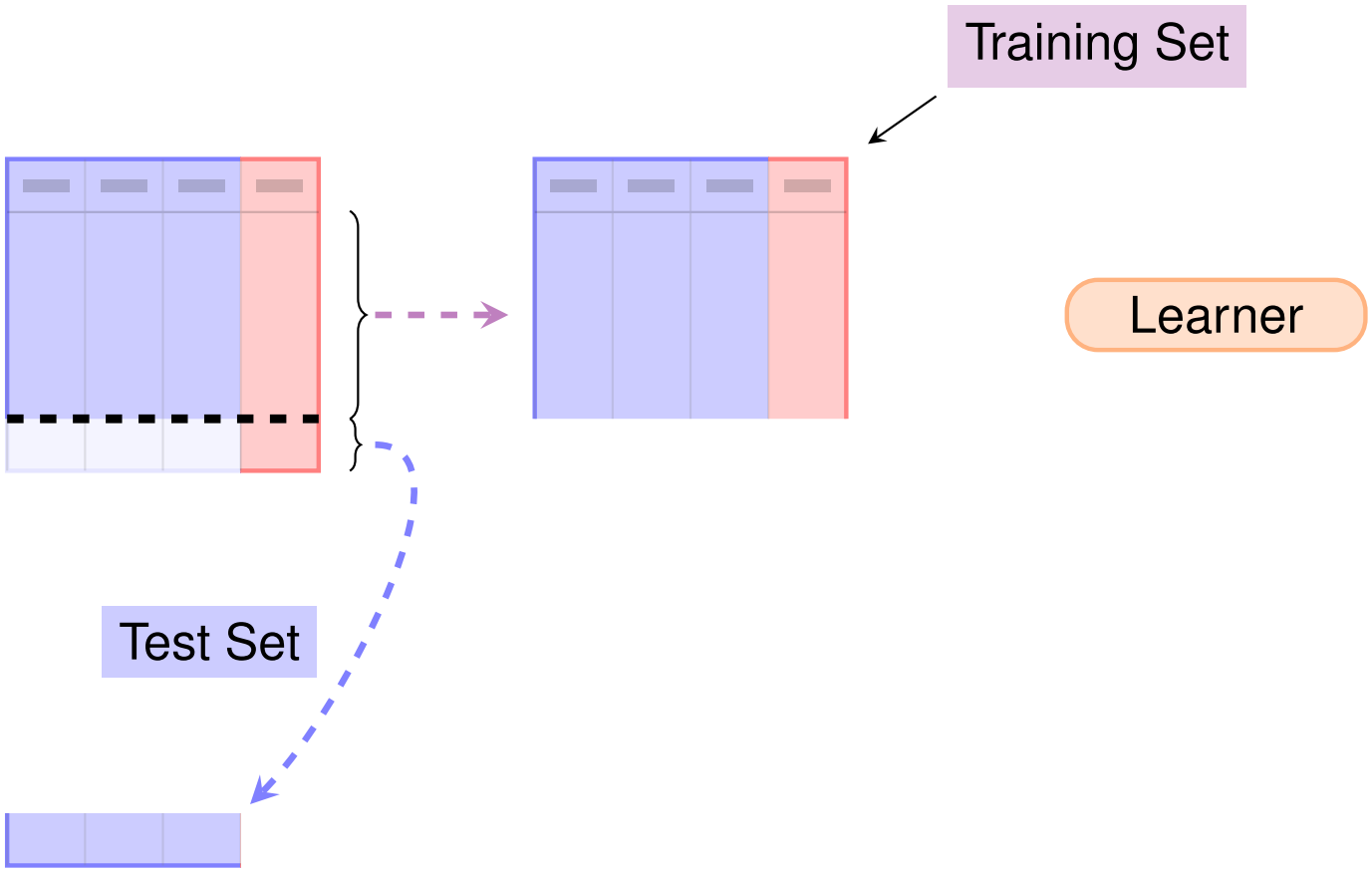


Learner

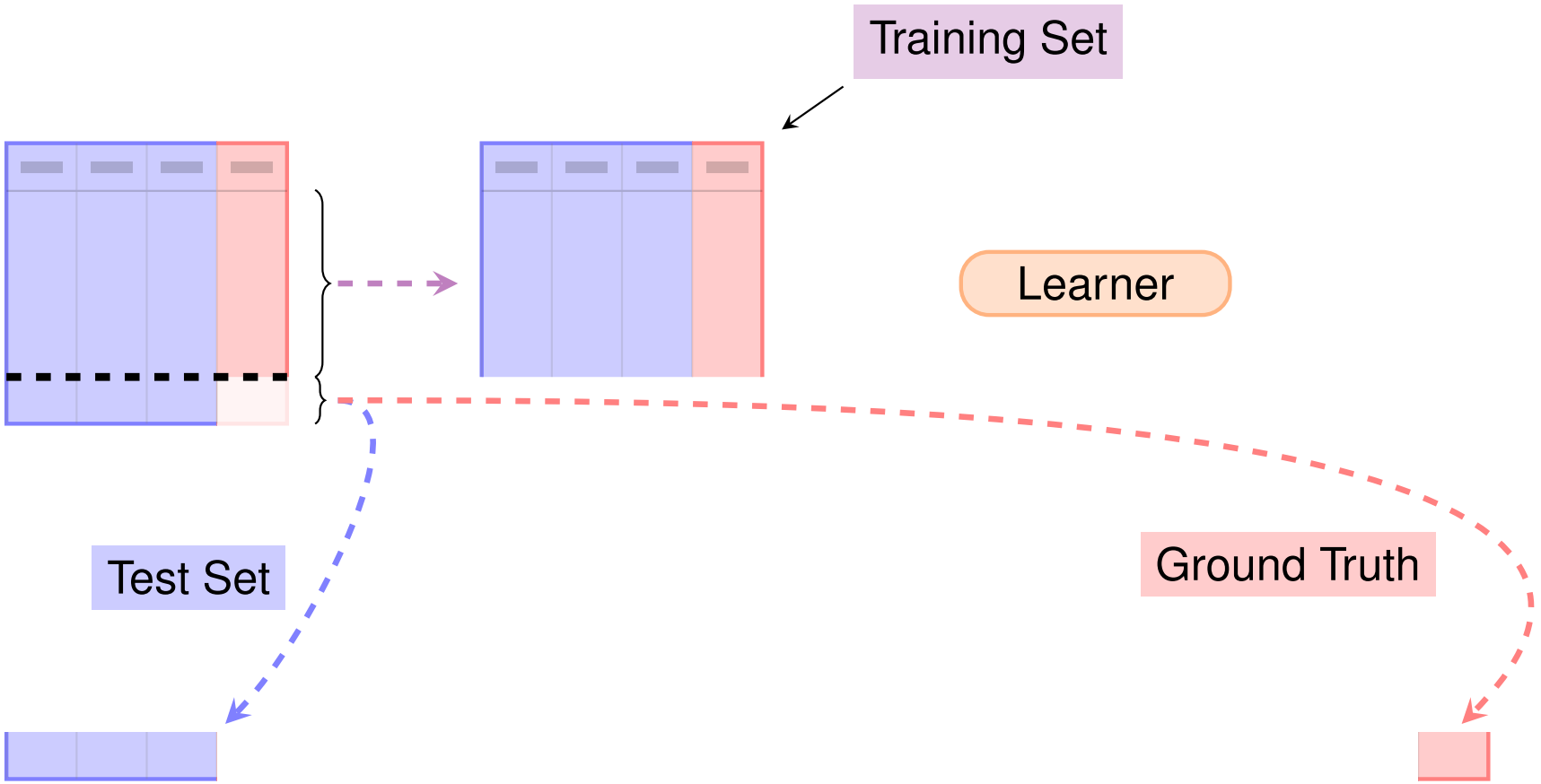
Resampling



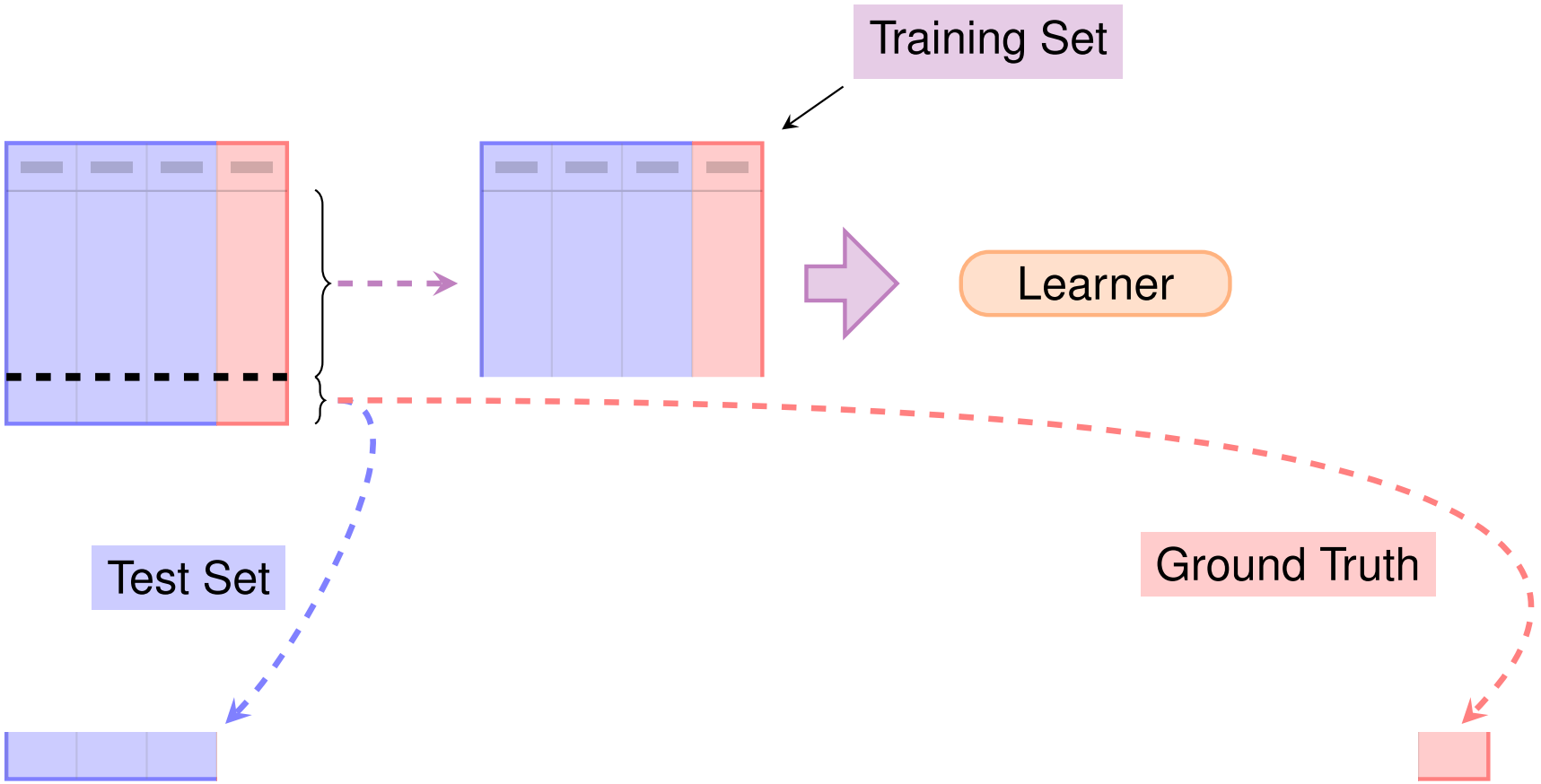
Resampling



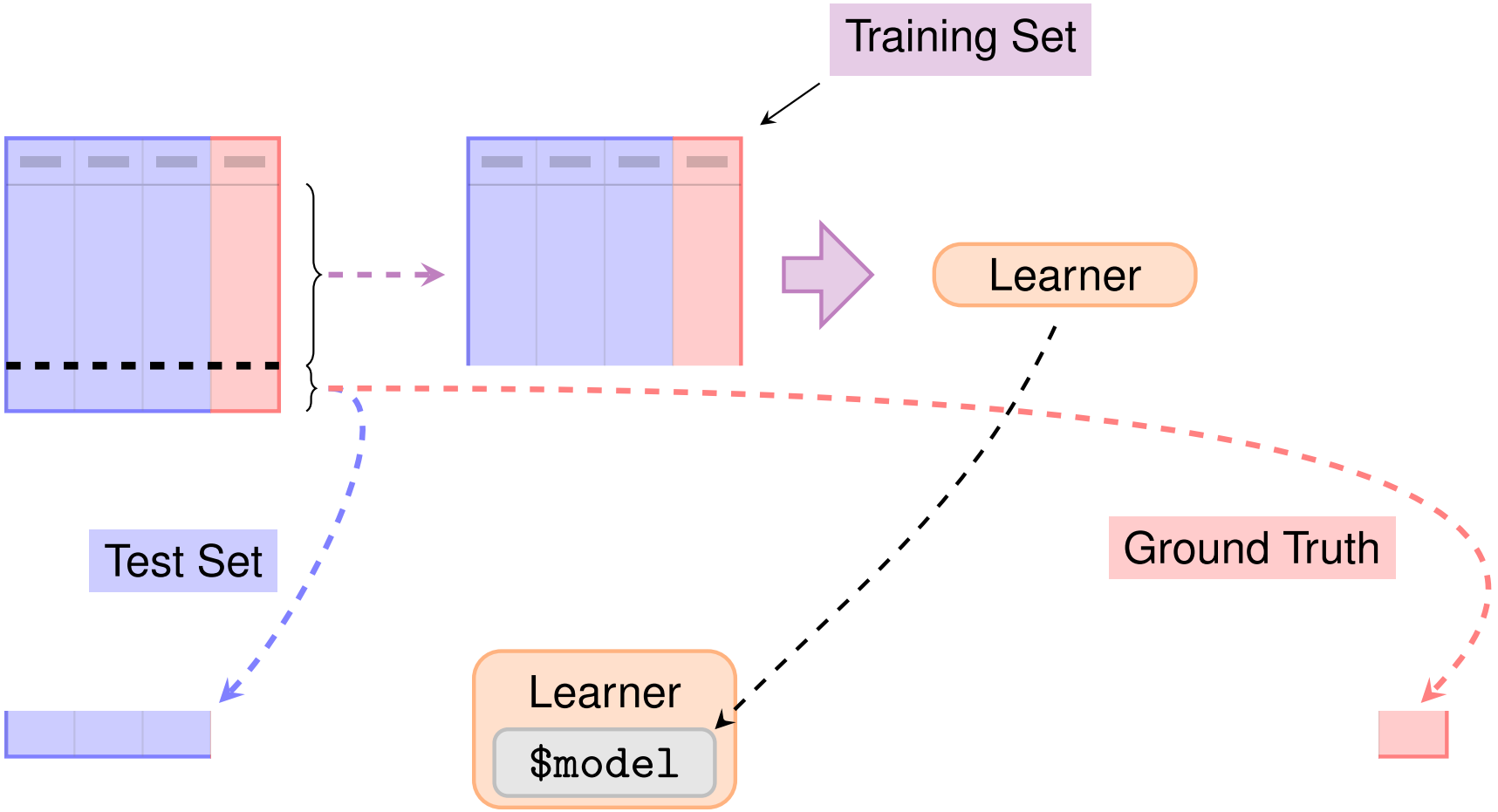
Resampling



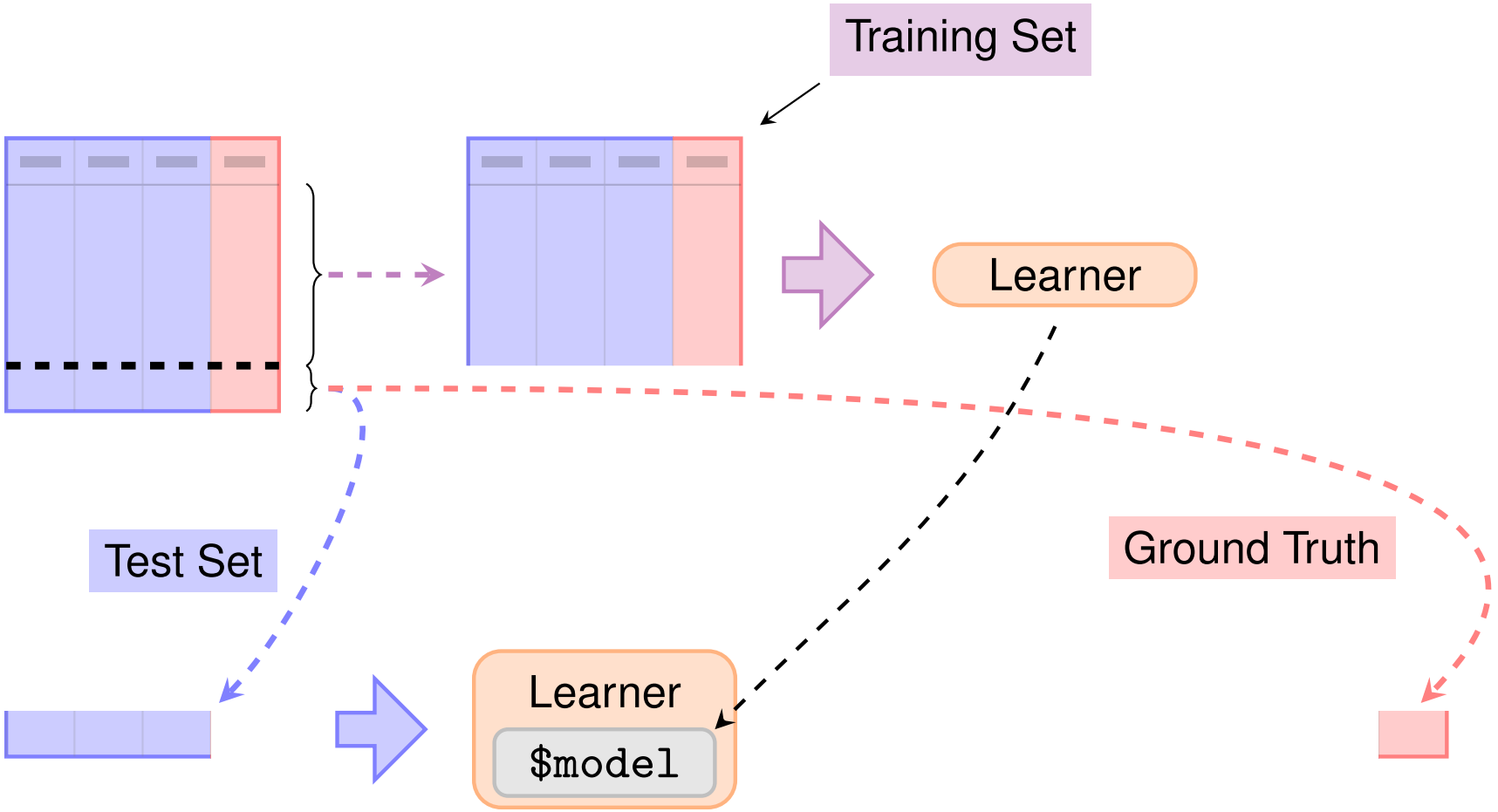
Resampling



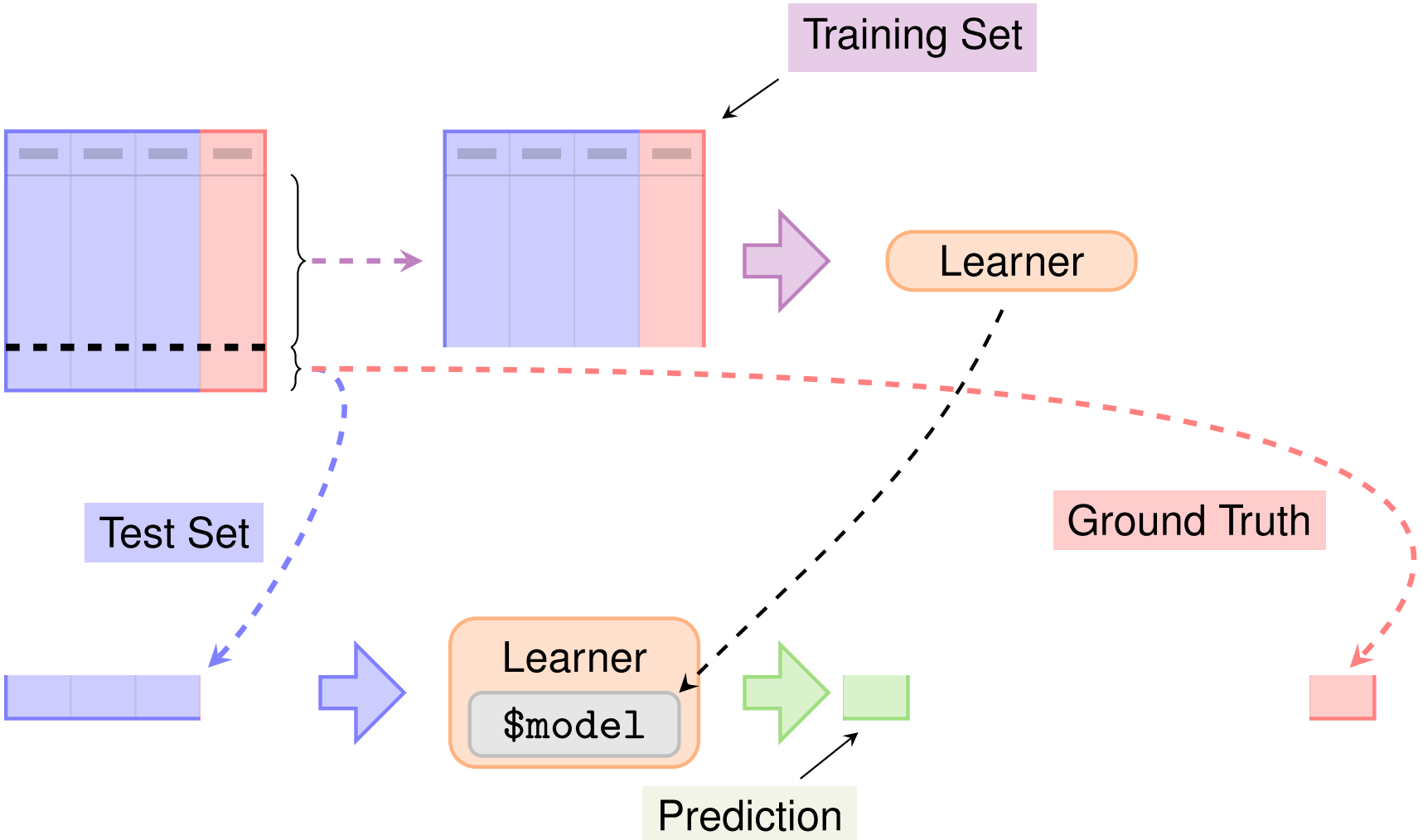
Resampling



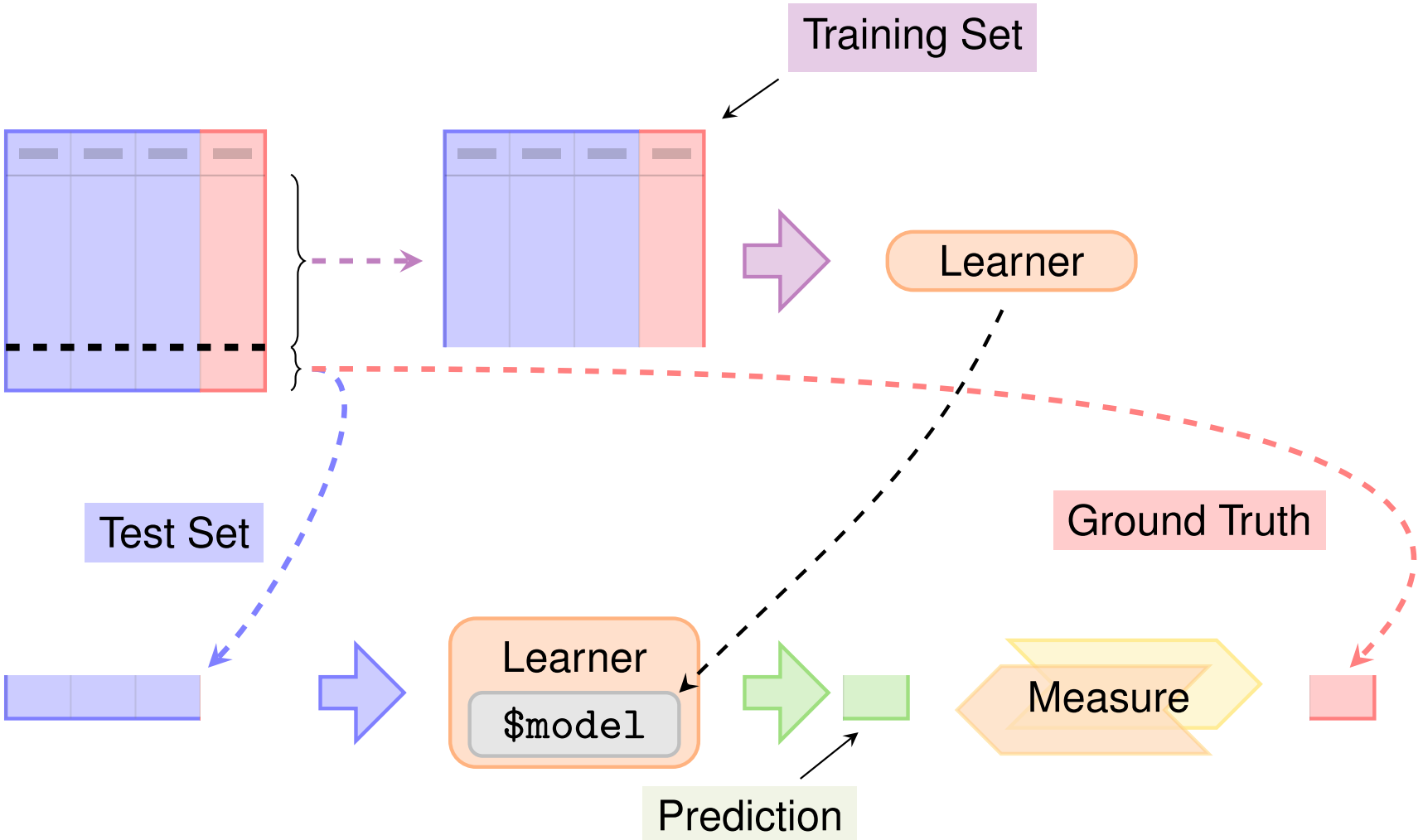
Resampling



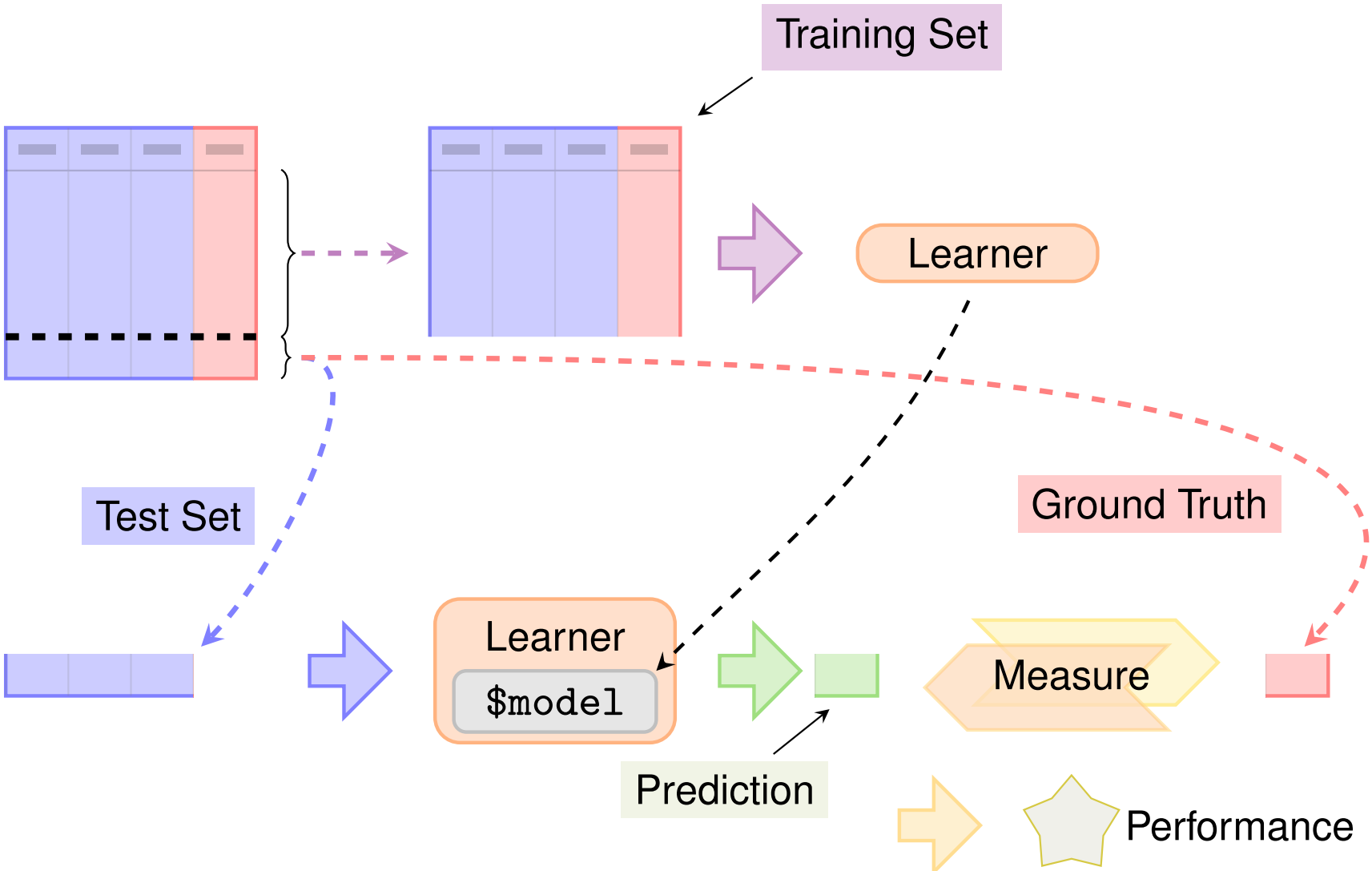
Resampling


Training Set
Learner
Test Set
Ground Truth
Learner
$model
Prediction

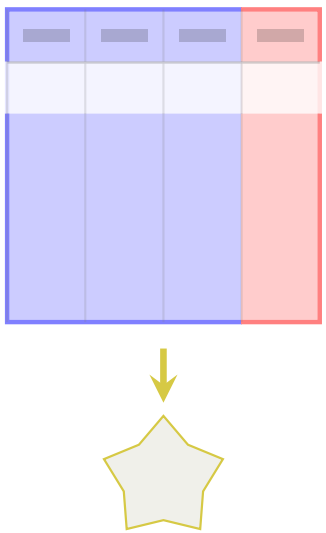
Resampling


Training Set
Learner
Test Set
Ground Truth
Learner
$model
Measure
Prediction

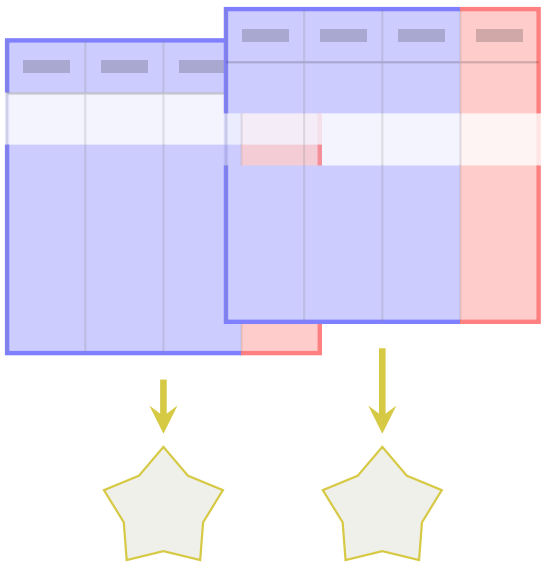
Resampling


Training Set
Learner
Test Set
Ground Truth
Learner
$model
Measure
Prediction
Performance

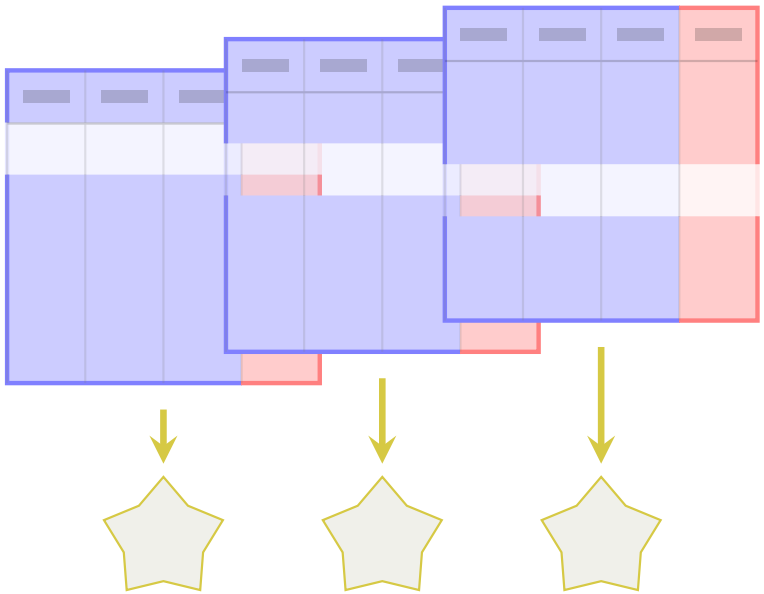
Resampling



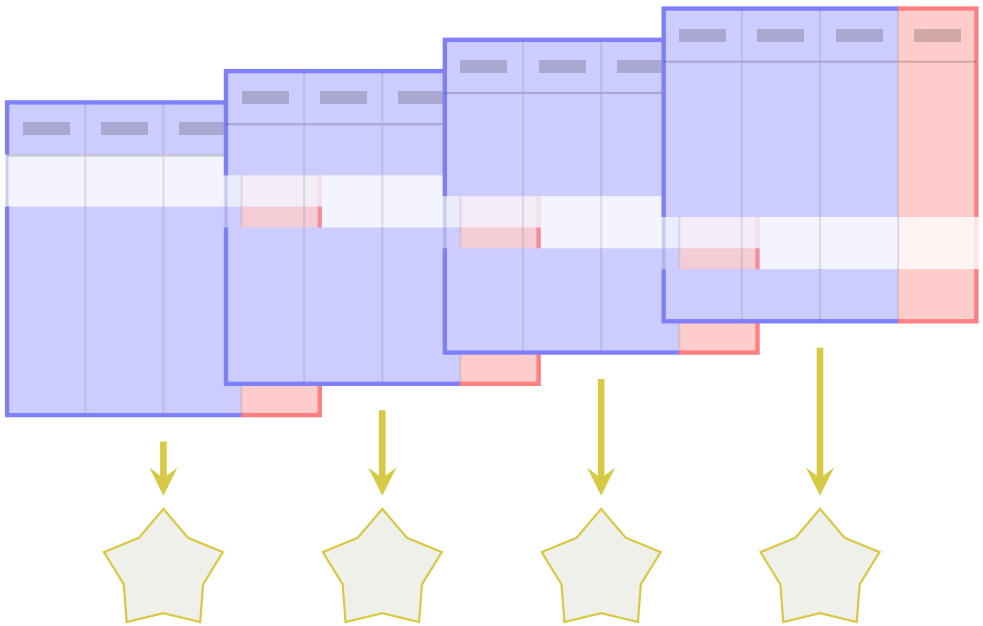
Resampling



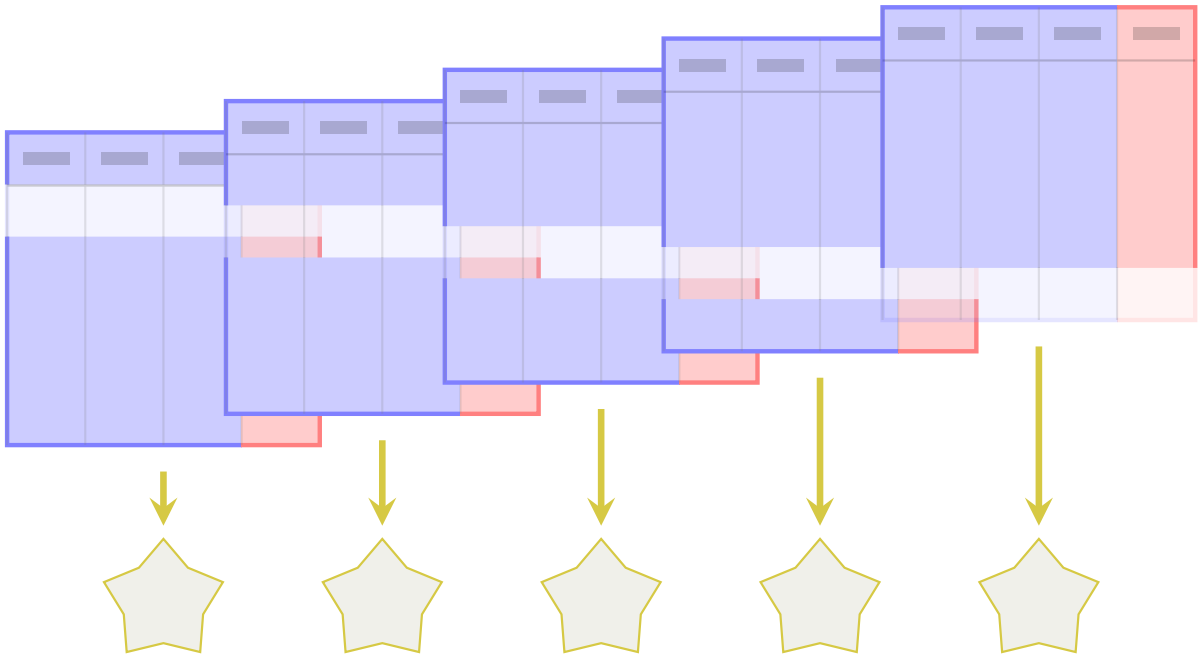
Resampling



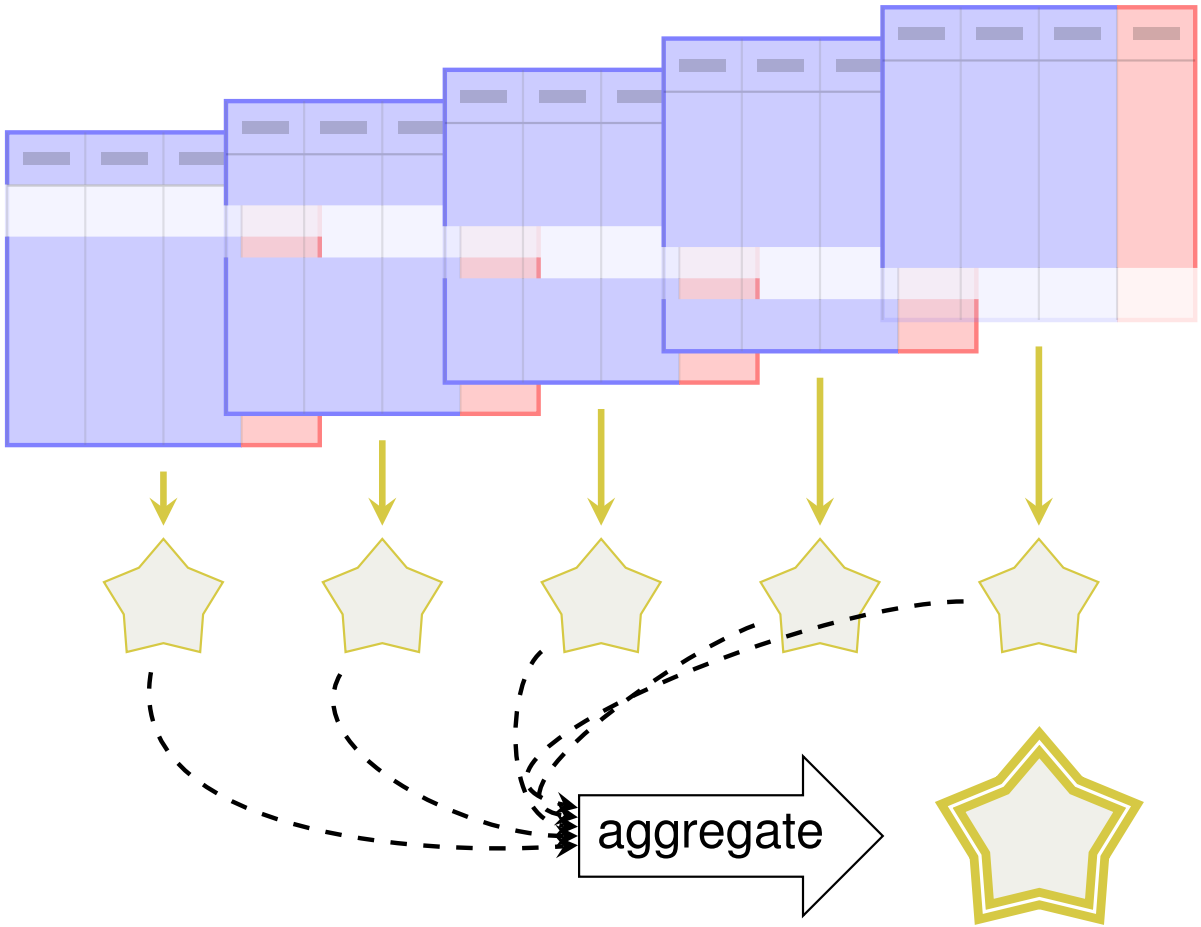
Resampling



Resampling

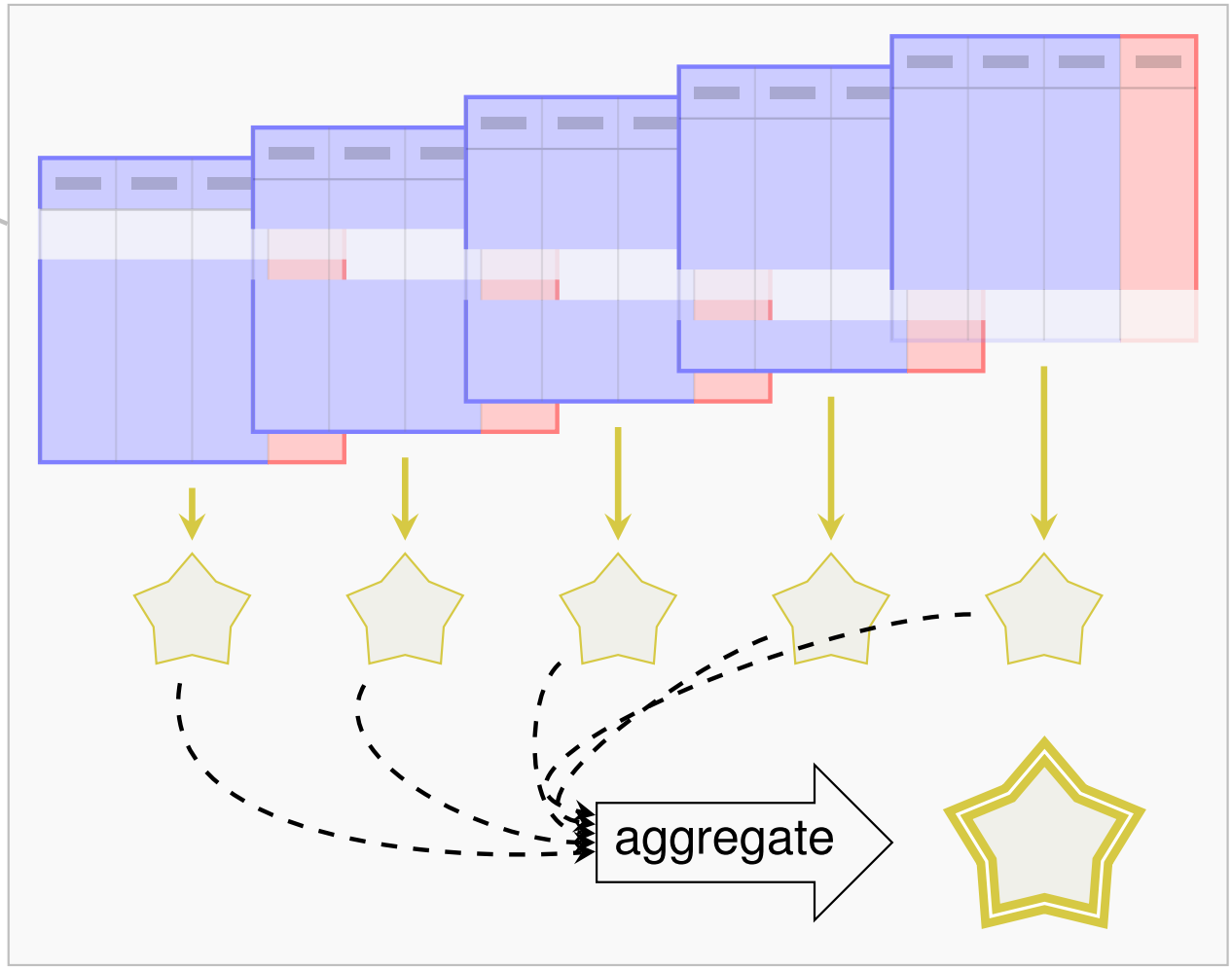


Resampling



Resampling

`resample()`



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:

```
result = resample(task, learner, cv5, store_models = TRUE)
```

Resampling

- Resample description: How to split the data

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

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

```
result = resample(task, learner, cv5, store_models = TRUE)
```

- The resampling result:

```
print(result)
#> <ResampleResult> of 5 iterations
#> * Task: iris
#> * Learner: classif.rpart
#> * Performance: 0.060 [classif.ce]
#> * Warnings: 0 in 0 iterations
#> * Errors: 0 in 0 iterations
```

Resampling

- Resample description: How to split the data

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

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

```
result = resample(task, learner, cv5, store_models = TRUE)
```

- The resampling result:

```
print(result)
#> <ResampleResult> of 5 iterations
#> * Task: iris
#> * Learner: classif.rpart
#> * Performance: 0.060 [classif.ce]
#> * Warnings: 0 in 0 iterations
#> * Errors: 0 in 0 iterations
```

Resampling

- Get performance:

```
result$aggregate(msr("classif.ce"))
#> classif.ce
#>       0.06
```

Resampling

- Get performance:

```
result$aggregate(msr("classif.ce"))
#> classif.ce
#>       0.06
```

- Get predictions

```
result$prediction()
#> <PredictionClassif> for 150 observations:
#>     row_id     truth  response
#>          2    setosa    setosa
#>          4    setosa    setosa
#>          7    setosa    setosa
#> ---
#>        131 virginica virginica
#>        145 virginica virginica
#>        146 virginica virginica
```

Resampling

- Predictions of individual folds

```
result$predictions()[[1]]
#> <PredictionClassif> for 30 observations:
#>     row_id     truth  response
#>          2    setosa    setosa
#>          4    setosa    setosa
#>          7    setosa    setosa
#> ---
#>        142 virginica virginica
#>        143 virginica virginica
#>        147 virginica virginica
```

Resampling

- Predictions of individual folds

```
result$predictions()[[1]]
#> <PredictionClassif> for 30 observations:
#>     row_id     truth  response
#>          2    setosa    setosa
#>          4    setosa    setosa
#>          7    setosa    setosa
#> ---
#>        142 virginica virginica
#>        143 virginica virginica
#>        147 virginica virginica
```

- Score of individual folds

```
result$score()[, .(iteration, classif.ce)]
#>    iteration classif.ce
#>        <int>      <num>
#> 1:         1      0.067
#> 2:         2      0.100
#> 3:         3      0.033
#> 4:         4      0.033
#> 5:         5      0.067
```

Resampling

- Acess to models of individual folds (only if `store_models = TRUE`)

```
result$data$learner[[1]]$importance()
#>  Petal.Width Petal.Length Sepal.Length  Sepal.Width
#>           72           66           45           29
```

Resampling

- Acess to models of individual folds (only if `store_models = TRUE`)

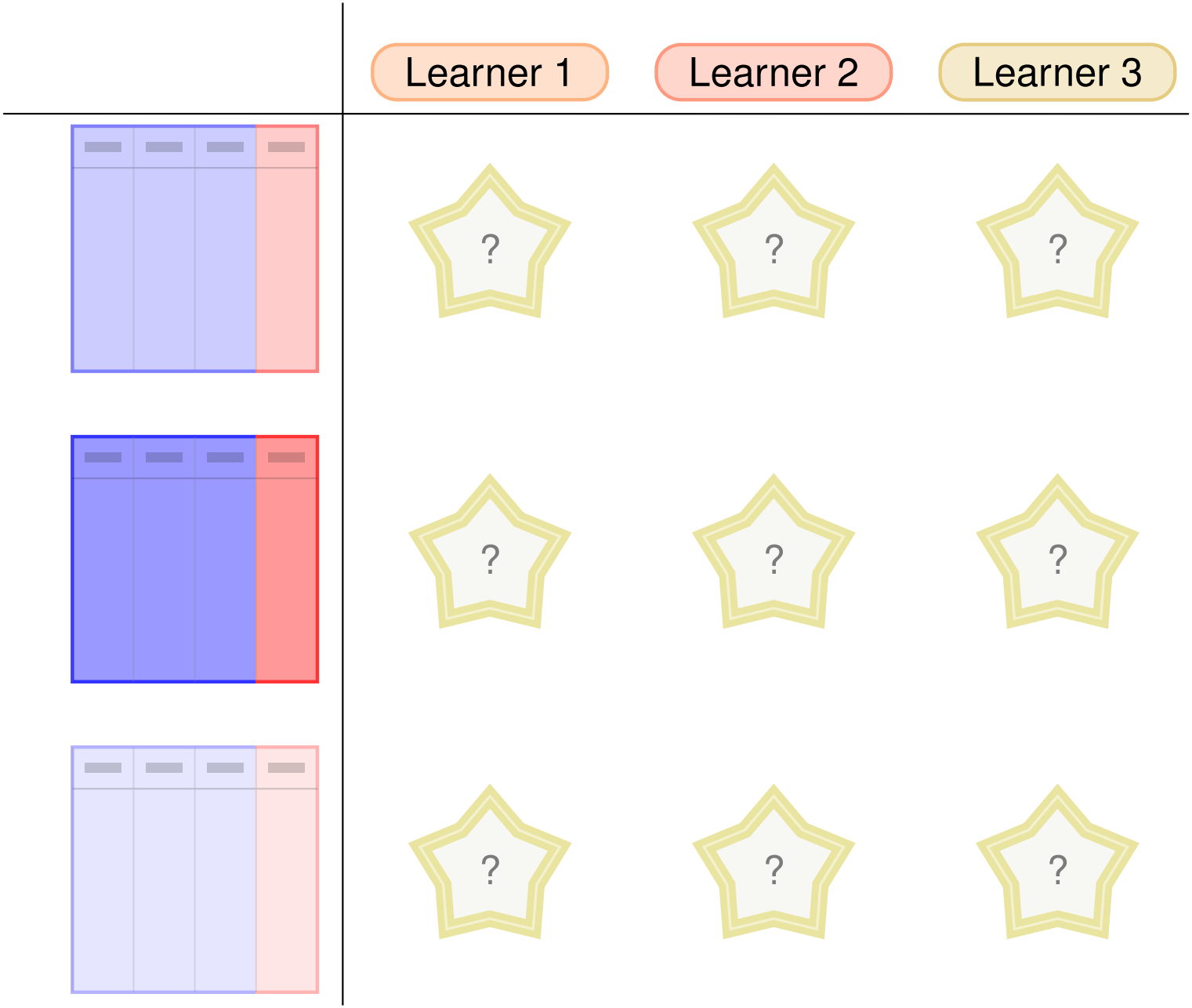
```
result$data$learner[[1]]$importance()
#>  Petal.Width Petal.Length Sepal.Length  Sepal.Width
#>           72           66           45           29
```

- Aggregate over multiple folds:

```
sapply(result$data$learner, function(x) x$importance()) %>%
  apply(1, mean)
#>  Petal.Width Petal.Length Sepal.Length  Sepal.Width
#>           72           65           45           28
```

Benchmark

Performance Comparison



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)
```

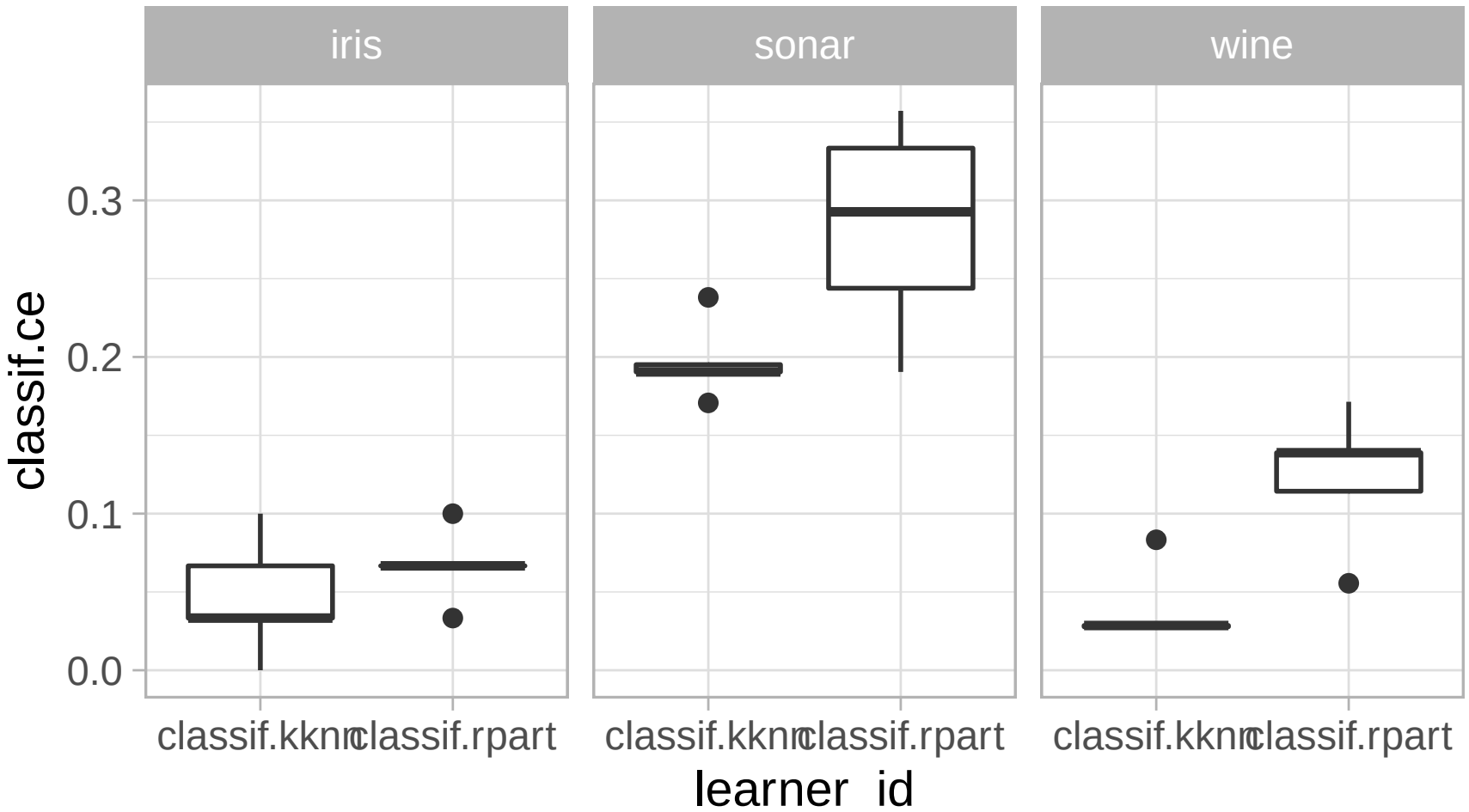
- The benchmark result shows that `kknn` outperforms `rpart`:

```
bmr$aggregate()[, .(task_id, learner_id, classif.ce)]
#>    task_id    learner_id classif.ce
#>     <char>        <char>      <num>
#> 1:    iris classif.rpart      0.067
#> 2:    iris  classif.kknn      0.047
#> 3:   sonar classif.rpart      0.284
#> 4:   sonar  classif.kknn      0.197
#> 5:    wine classif.rpart      0.124
#> 6:    wine  classif.kknn      0.039
```

Performance Comparison

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

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



Dictionaries

Dictionaries

- Ordinary constructors: `LearnerClassifRpart$new()`

Dictionaries

- Ordinary constructors: `LearnerClassifRpart$new()`
- `mlr3` comes with many predefined objects (Learners, Tasks, Measures, ...) stored in `Dictionary` objects.
- Use dictionaries to list available objects (e.g. Learners)

$\Rightarrow$ Short form getters for each `Dictionary`

Dictionaries

- Ordinary constructors: `LearnerClassifRpart$new()`
- `mlr3` comes with many predefined objects (Learners, Tasks, Measures, ...) stored in `Dictionary` objects.
- Use dictionaries to list available objects (e.g. Learners)

$\Rightarrow$ Short form getters for each `Dictionary`

Object	Dictionary	Short Form
`Task`	`mlr_tasks`	`tsk()`
`Learner`	`mlr_learners`	`lrn()`
`Measure`	`mlr_measures`	`msr()`
`Resampling`	`mlr_resamplings`	`rsmp()`

Dictionaries

- Ordinary constructors: `LearnerClassifRpart$new()`
- `mlr3` comes with many predefined objects (Learners, Tasks, Measures, ...) stored in `Dictionary` objects.
- Use dictionaries to list available objects (e.g. Learners)

$\Rightarrow$ Short form getters for each `Dictionary`

Object	Dictionary	Short Form
`Task`	`mlr_tasks`	`tsk()`
`Learner`	`mlr_learners`	`lrn()`
`Measure`	`mlr_measures`	`msr()`
`Resampling`	`mlr_resamplings`	`rsmp()`

```
mlr_learners

#> <DictionaryLearner> with 21 stored values
#> Keys: classif.debug, classif.featureless, classif.glmnet,
#>   classif.kknn, classif.lda, classif.log_reg,
#>   classif.naive_bayes, classif.qda, classif.ranger,
#>   classif.rpart, classif.svm, classif.xgboost,
#>   regr.featureless, regr.glmnet, regr.kknn, regr.km,
#>   regr.lm, regr.ranger, regr.rpart, regr.svm, regr.xgboost
```

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

How to find help

How to find help for mlr3

- How do I start?
  - Check these slides
  - **Check the mlr3book mlr3book.mlr-org.com**
- What can I do with these R6 objects?
  1. Find out what kind of R6 object you have:

```
class(bmr)
#> [1] "BenchmarkResult" "R6"
```

  2. Go to the corresponding help page:

```
?BenchmarkResult
```

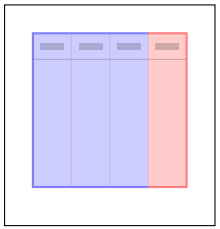
- Why does this not work?
  - Ask at stackoverflow stackoverflow.com/questions/tagged/mlr3
  - Write a GitHub issue (in the according project)

mlr3 recap

So you want to do ML in R

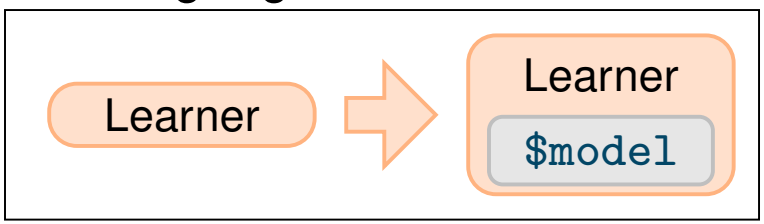
Ingredients:

Data



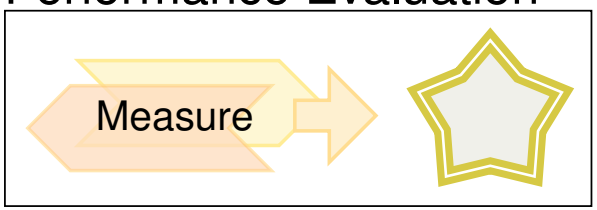
`TaskClassif`,
`TaskRegr`,
`tsk()`

Learning Algorithms



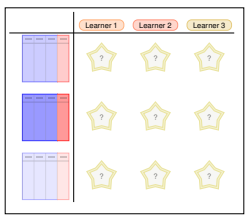
`lrn()`,
`$train()`,
`$predict()`

Performance Evaluation



`msr()`,
`resample()`
`$aggregate()`

Performance Comparison



`benchmark_grid()`,
`benchmark()`

mlr3tuning Intro

Tuning

- Behaviour of most methods depends on *hyperparameters*

Tuning

- Behaviour of most methods depends on *hyperparameters*
- We want to choose them so our algorithm performs well

Tuning

- Behaviour of most methods depends on *hyperparameters*
- We want to choose them so our algorithm performs well
- Good hyperparameters are data-dependent

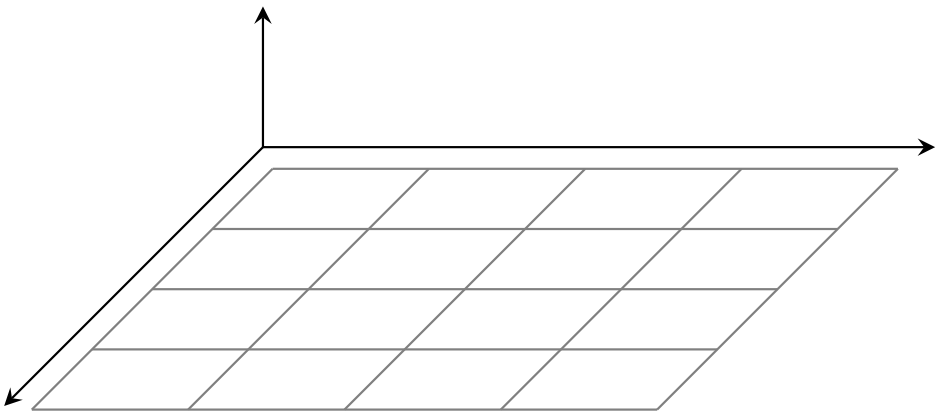
Tuning

- Behaviour of most methods depends on *hyperparameters*
- We want to choose them so our algorithm performs well
- Good hyperparameters are data-dependent

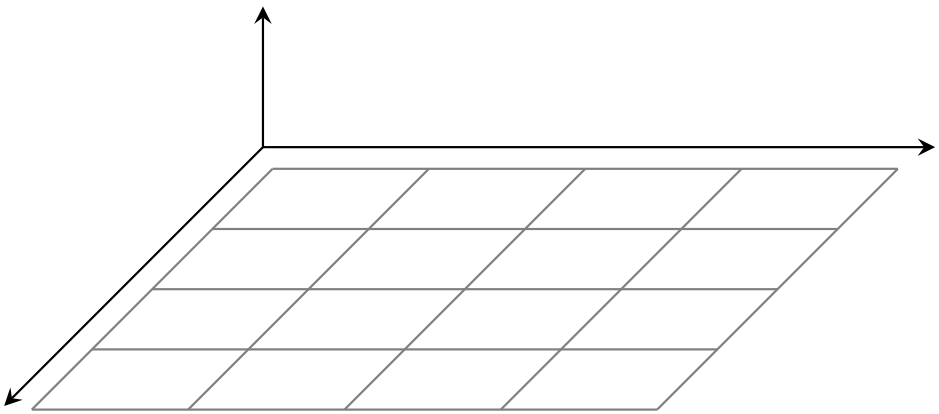
$\Rightarrow$ We do *black box optimization* (“Try stuff and see what works”)

Tuning

Tuning

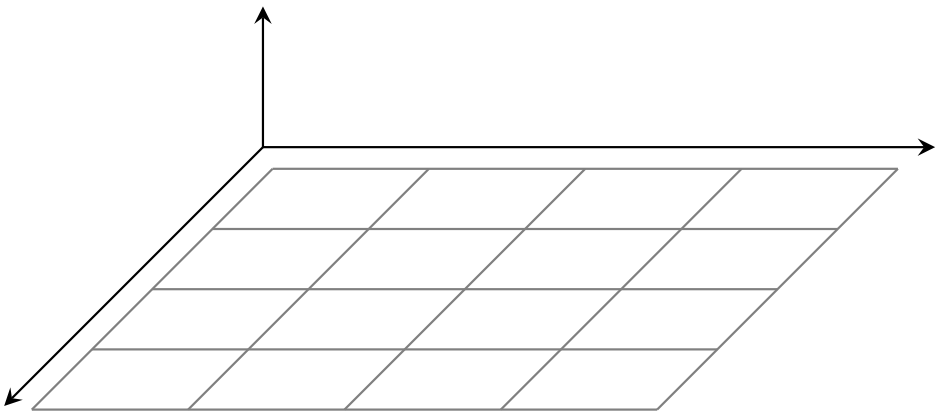


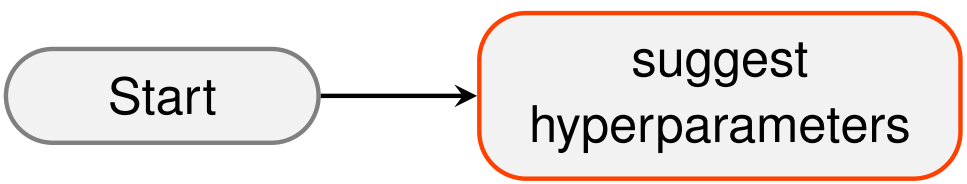
Tuning



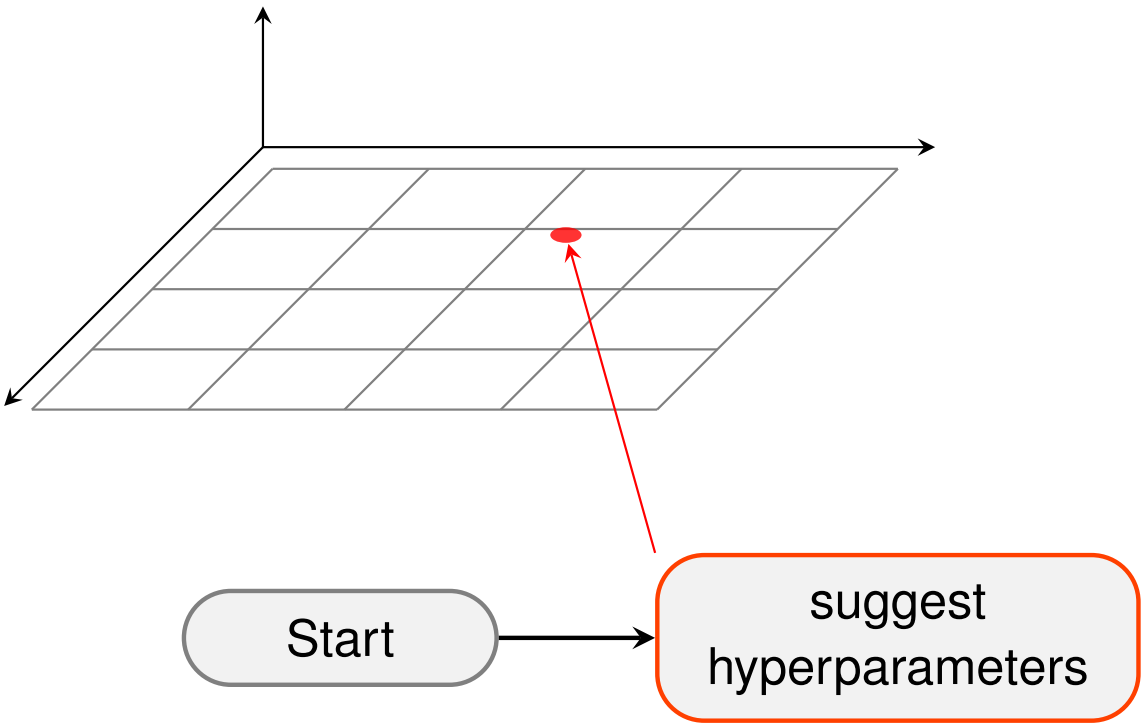
Start

Tuning

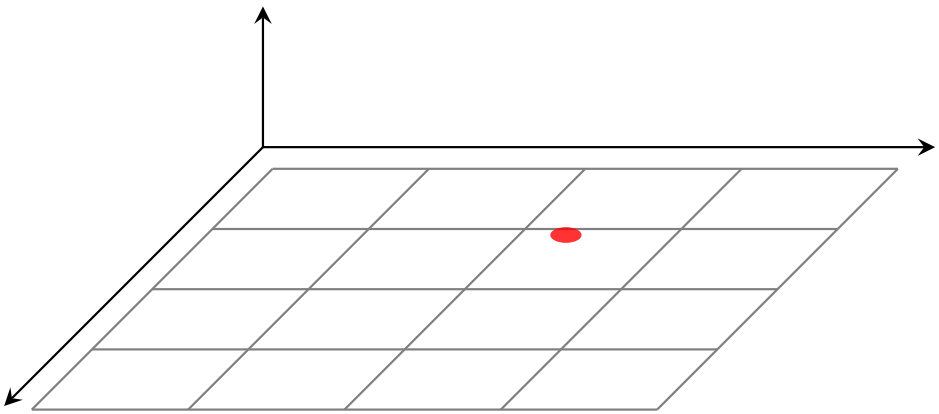


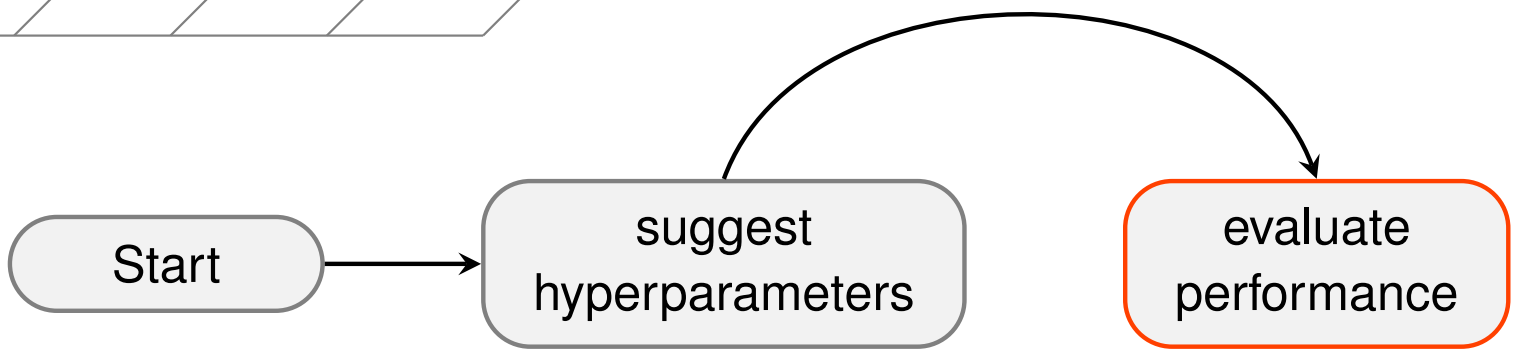


Tuning

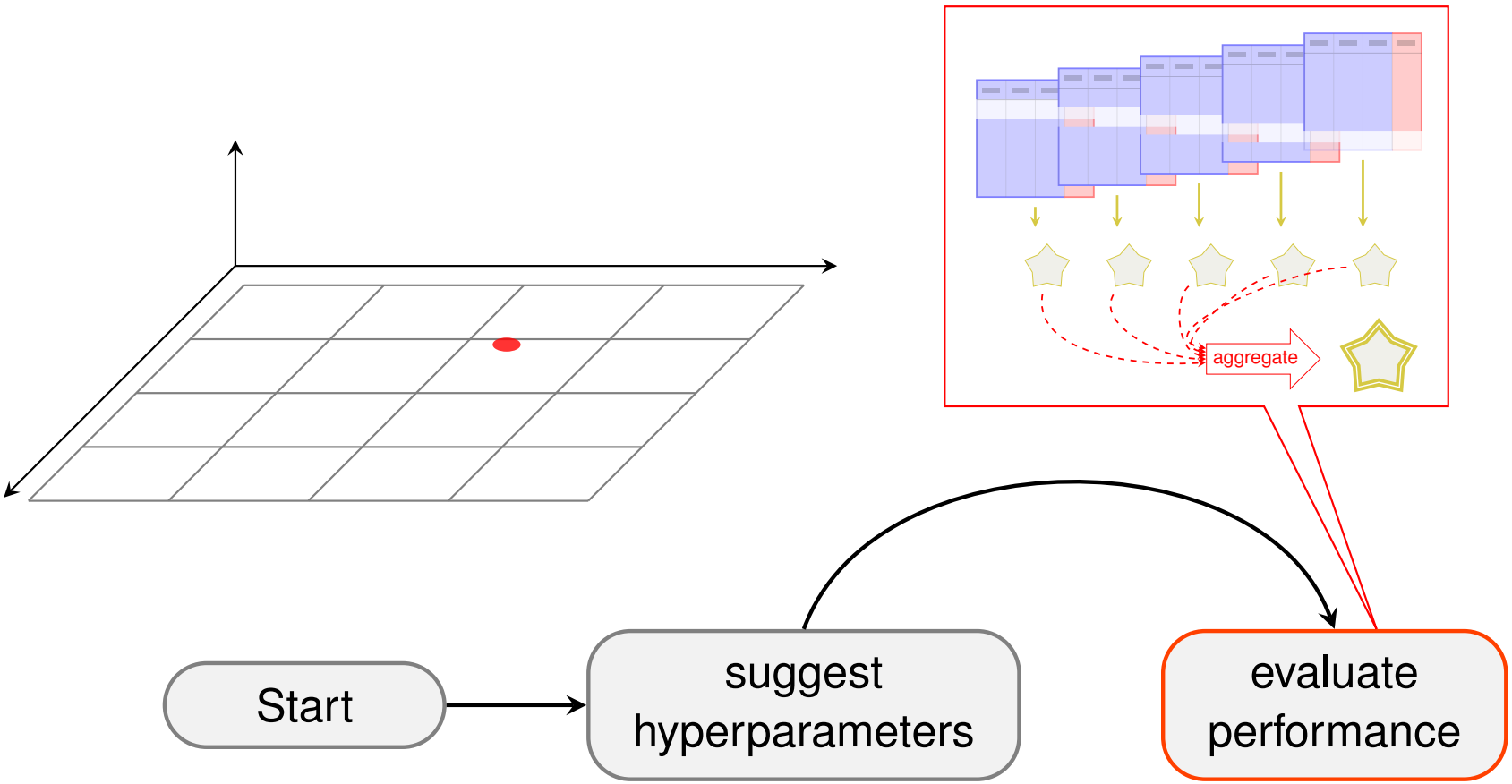


Tuning

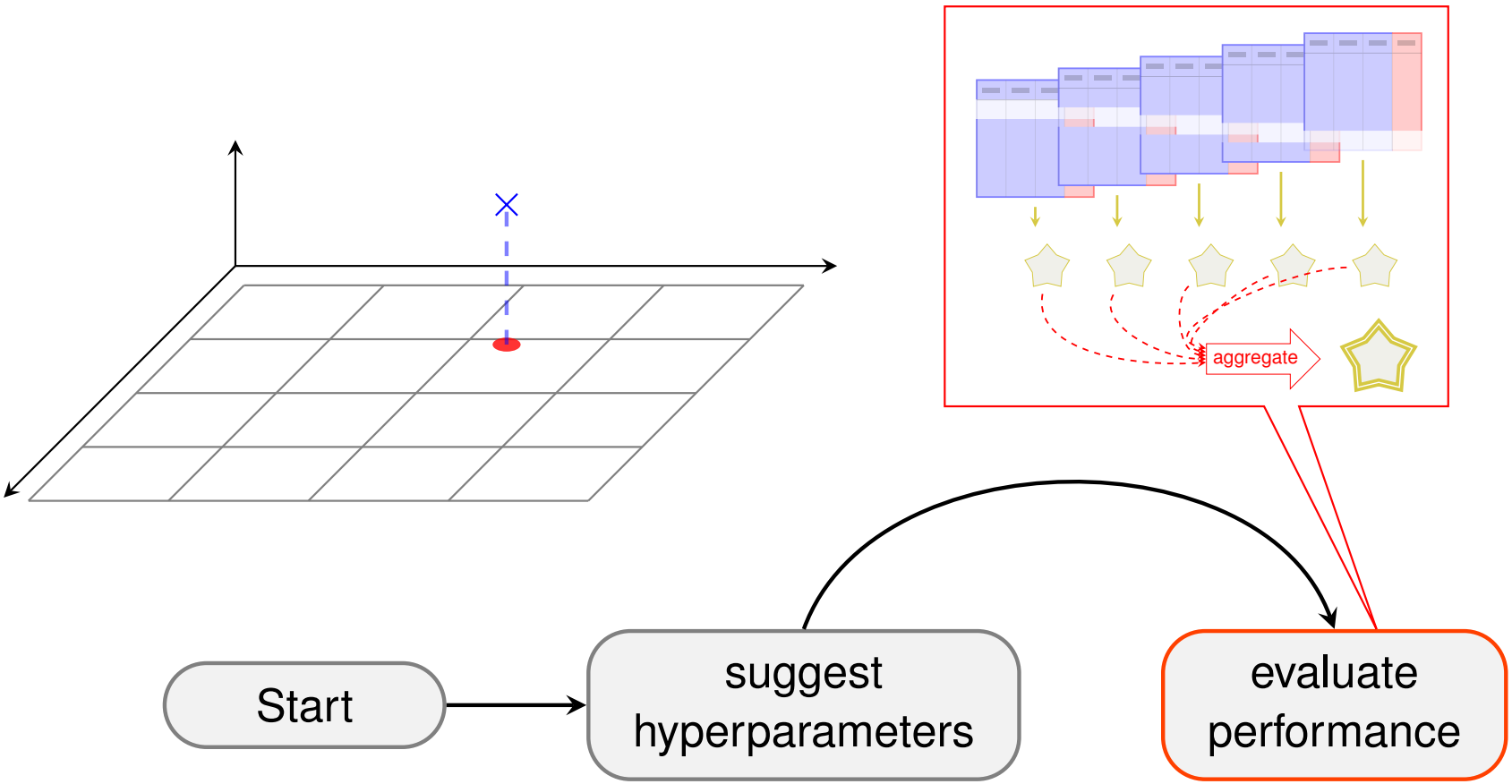




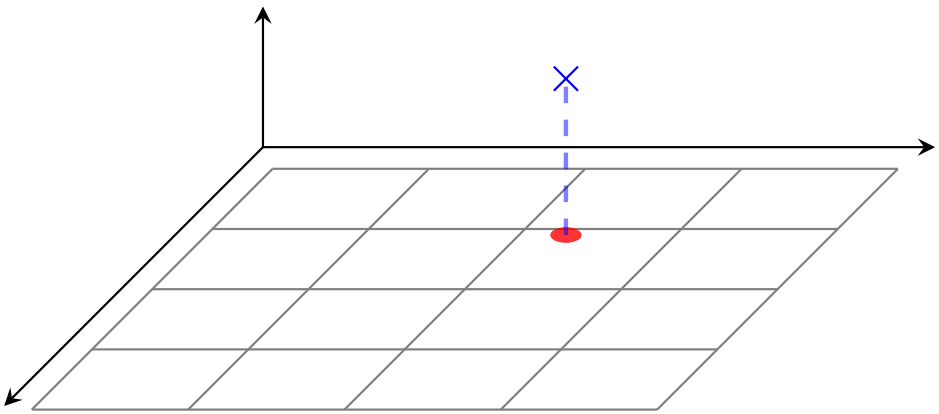
Tuning

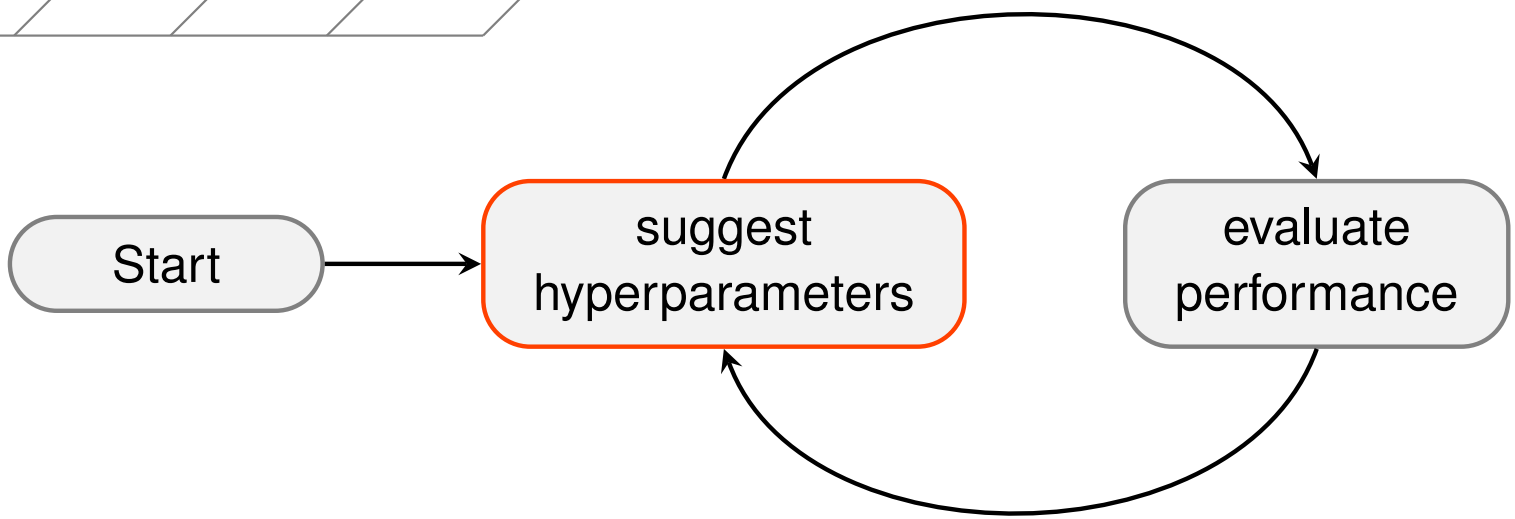

aggregate
Start
suggest hyperparameters
evaluate performance

Tuning

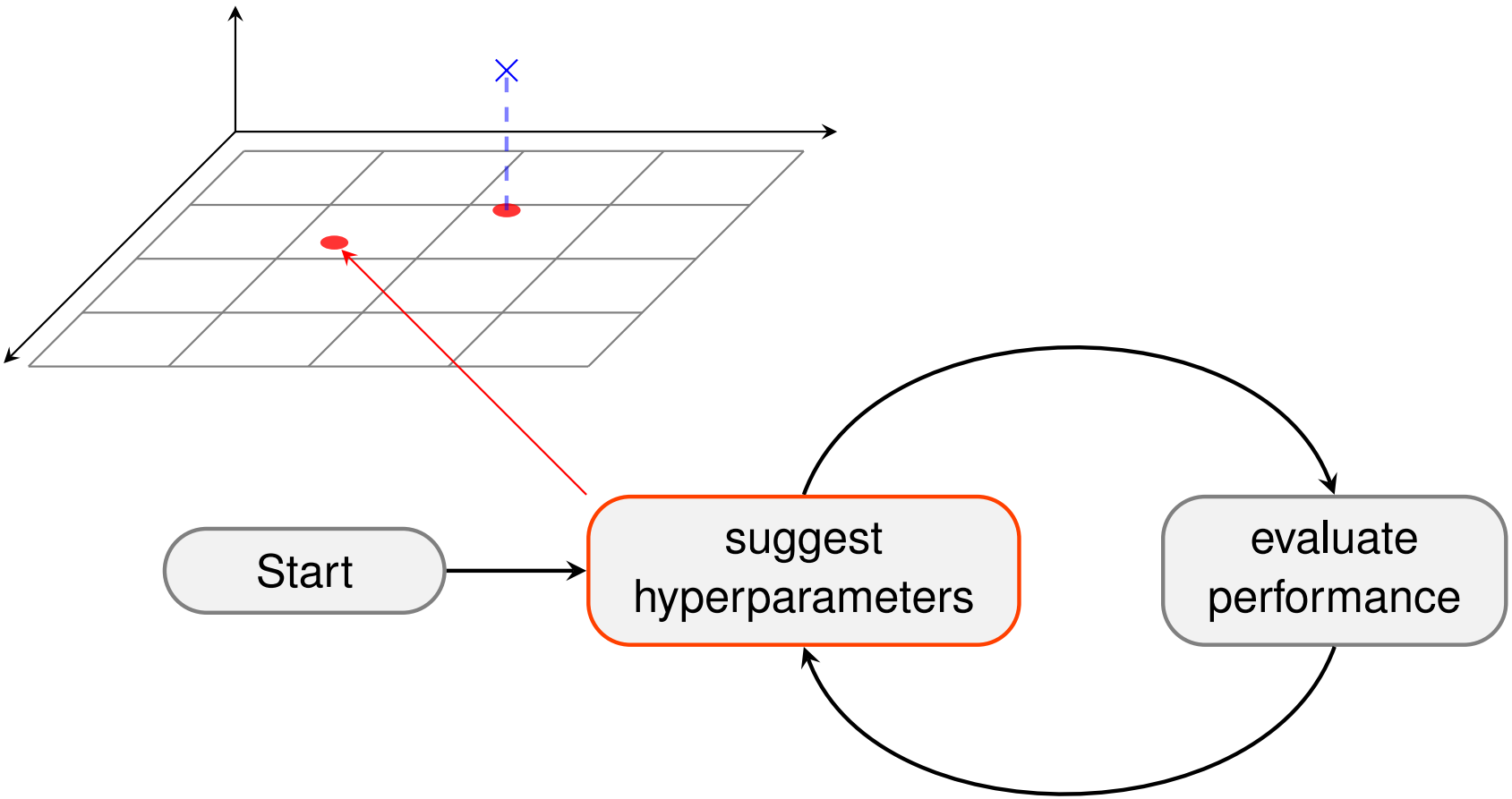

aggregate
Start
suggest hyperparameters
evaluate performance

Tuning

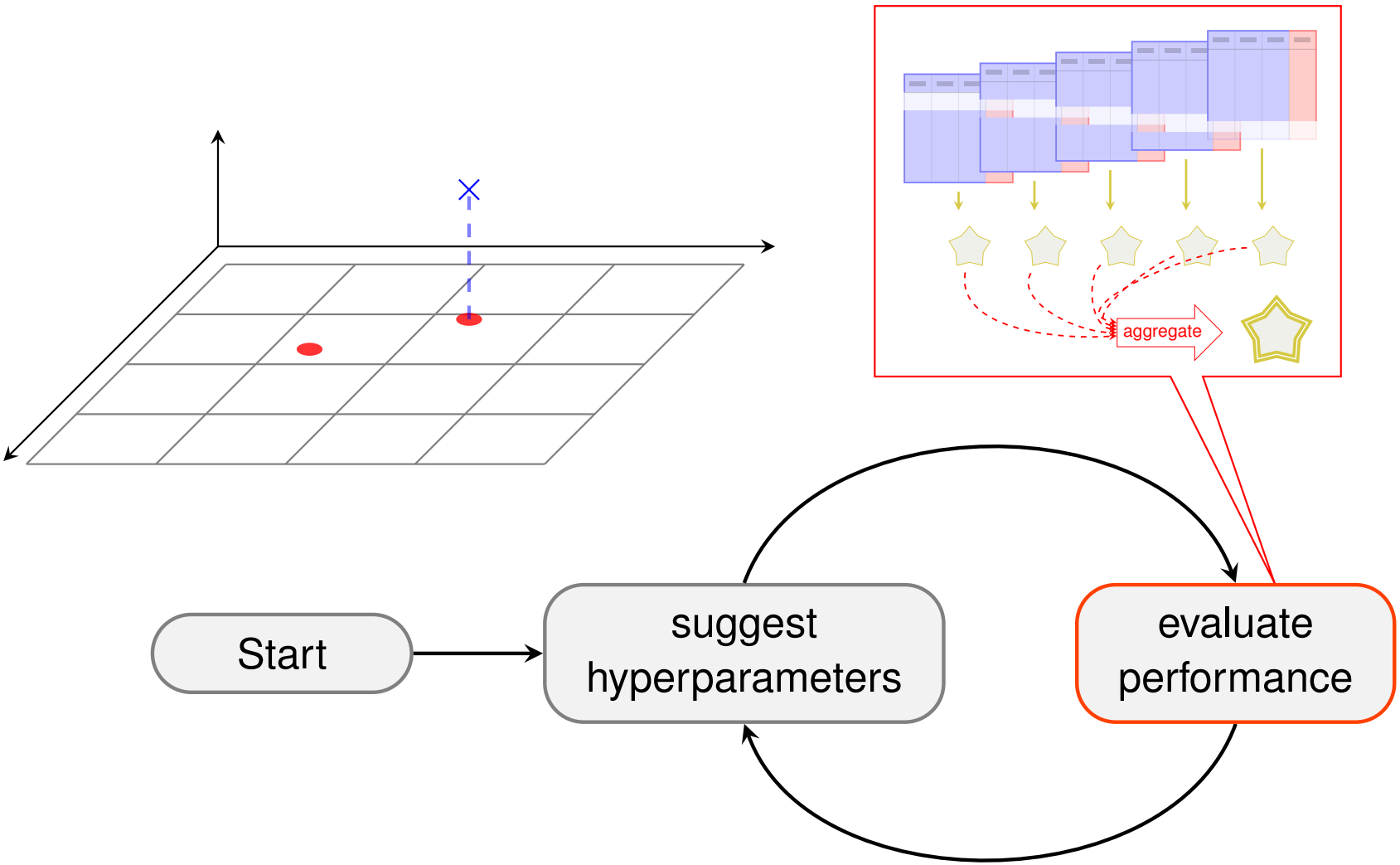




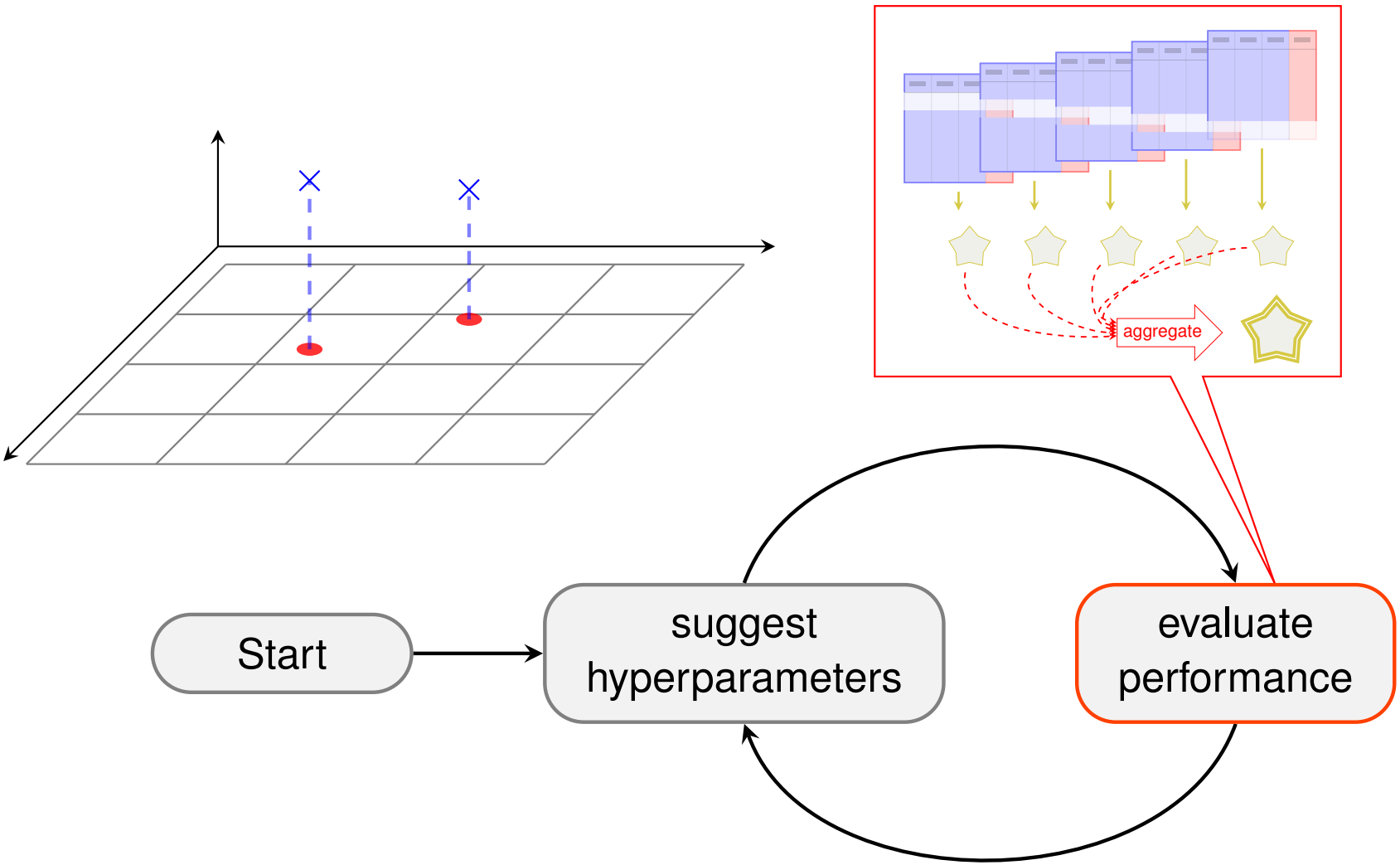
Tuning



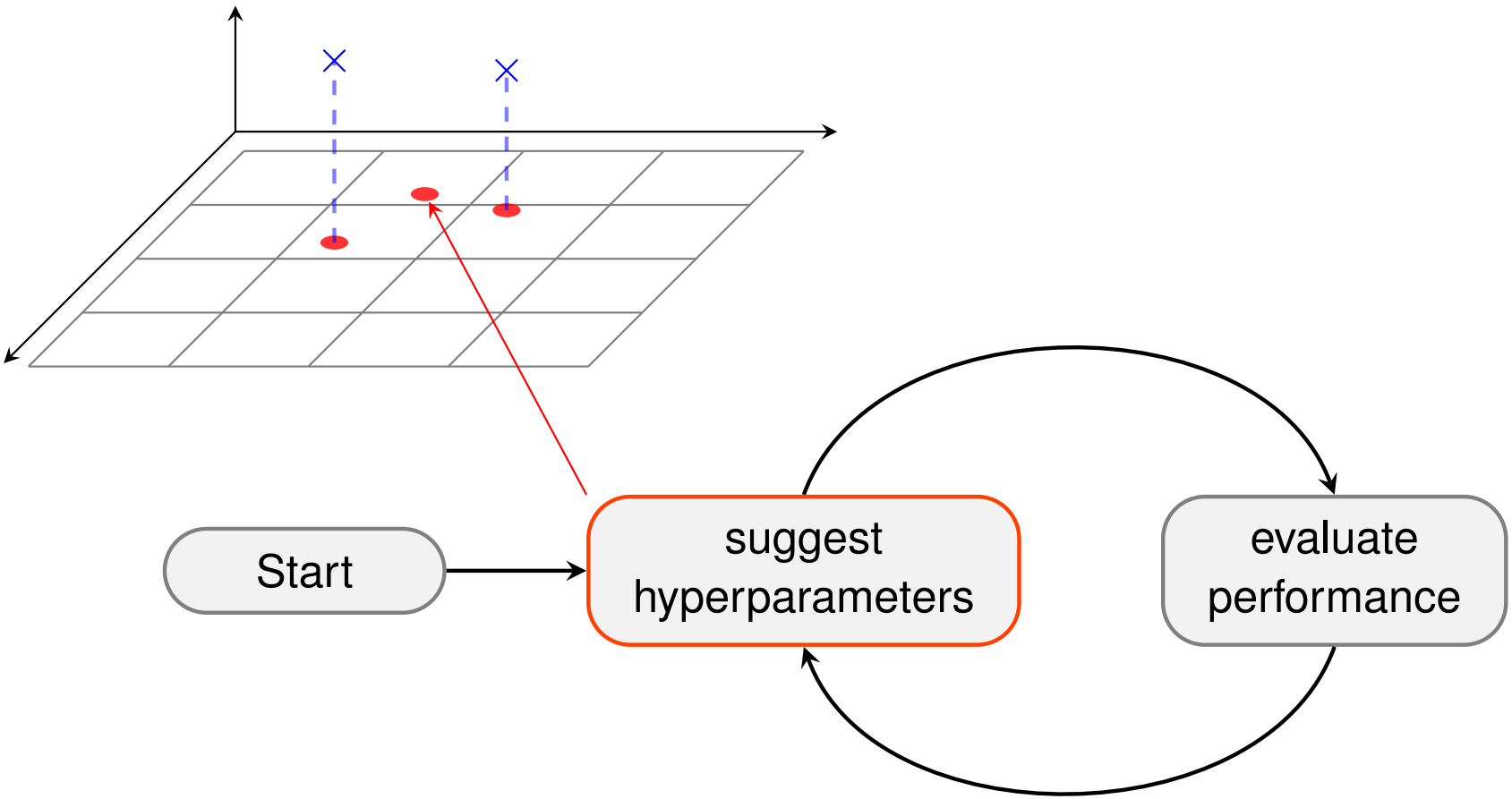
Tuning


aggregate
Start
suggest hyperparameters
evaluate performance

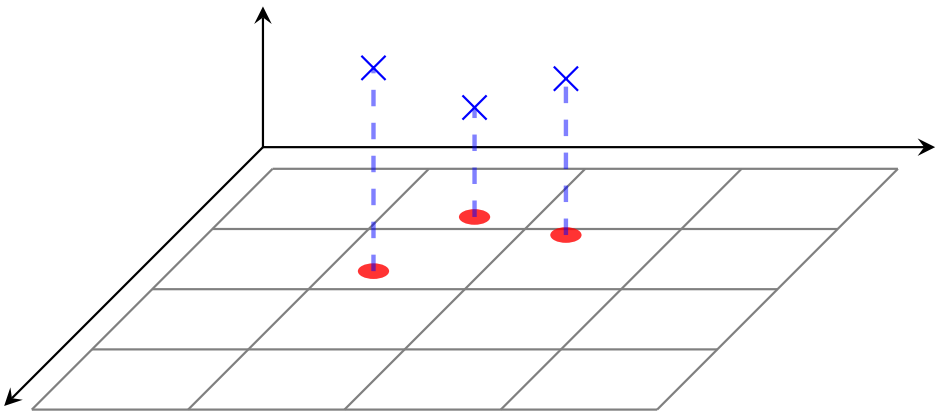
Tuning

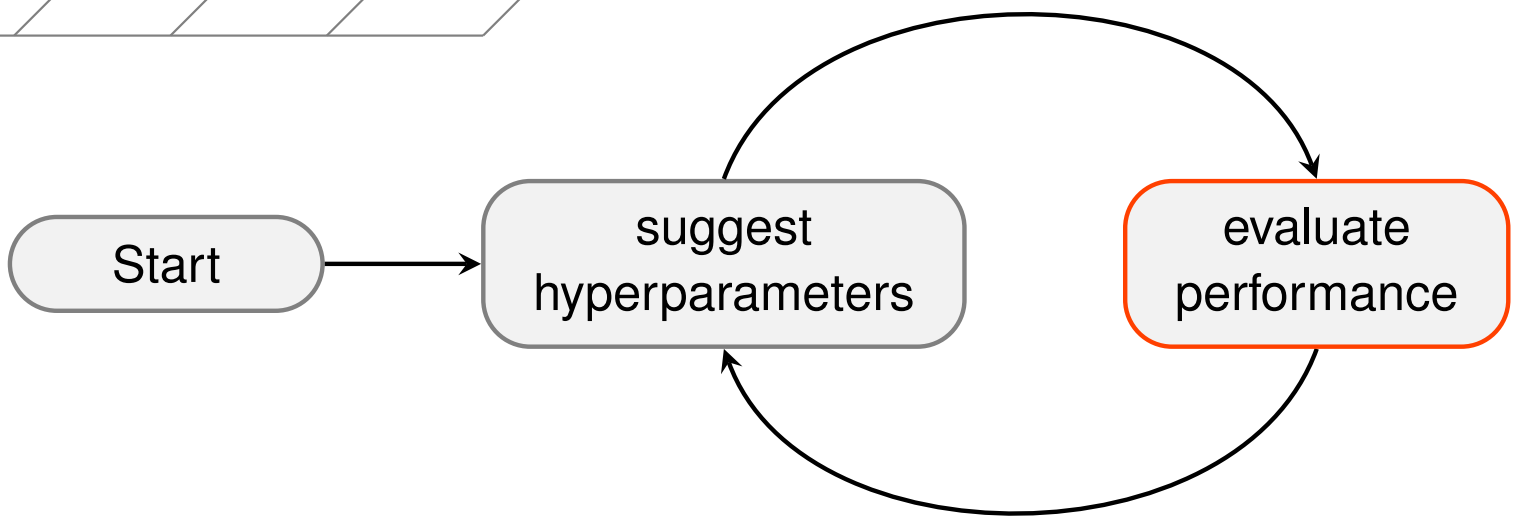

aggregate
Start
suggest hyperparameters
evaluate performance

Tuning

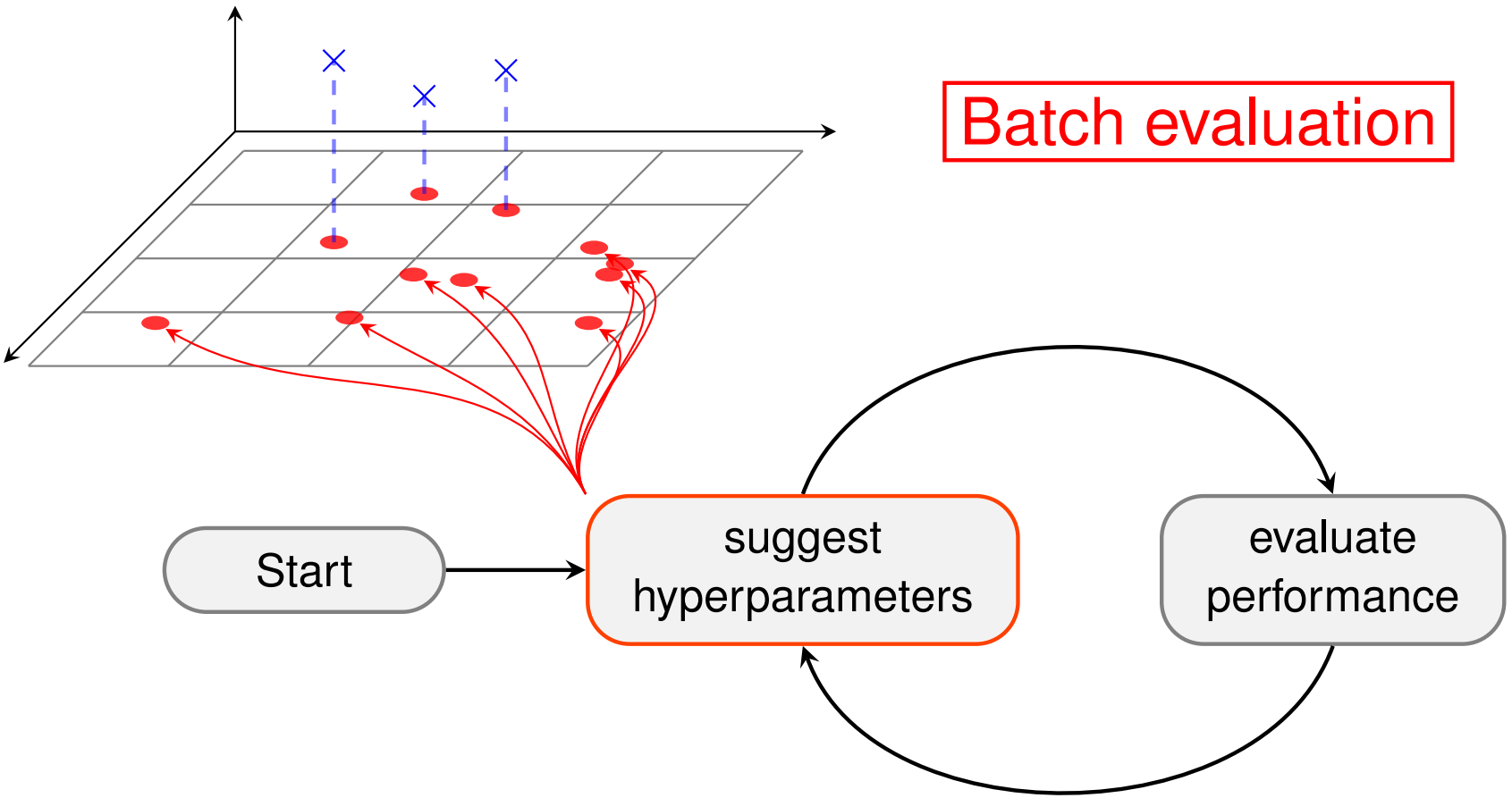


Tuning

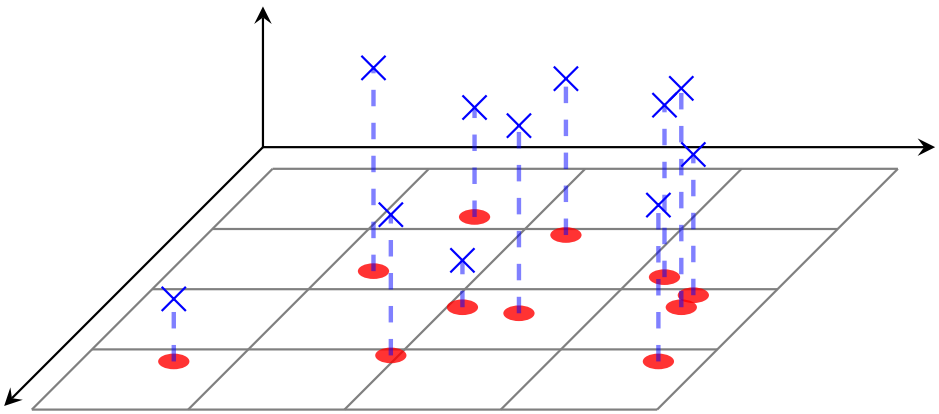




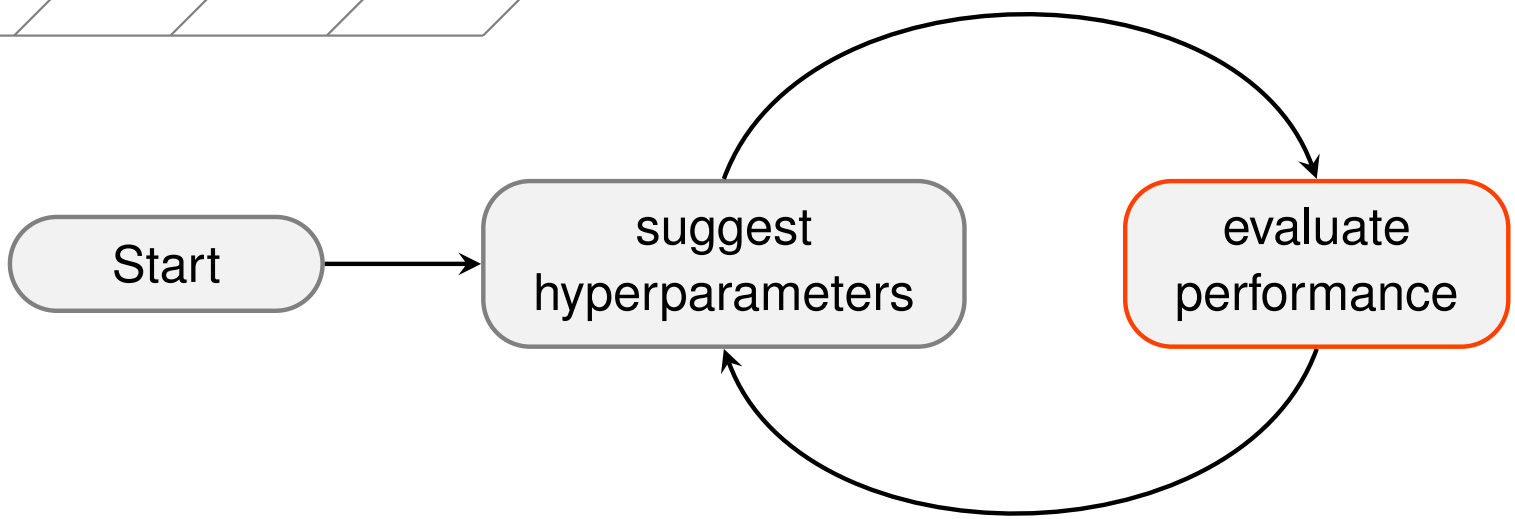
Tuning


Batch evaluation
Start
suggest hyperparameters
evaluate performance

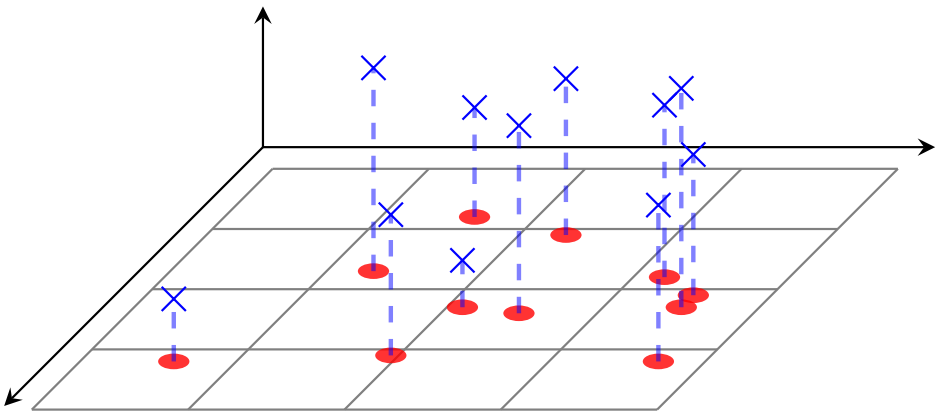
Tuning

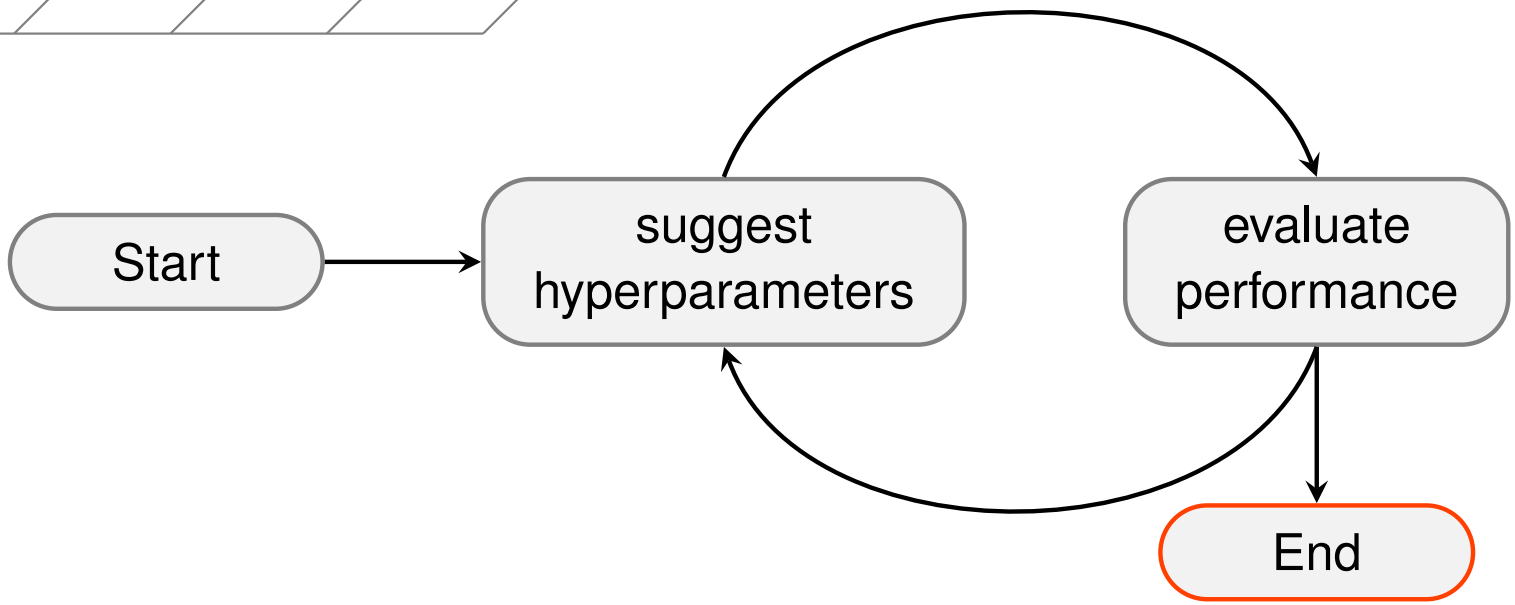


Batch evaluation

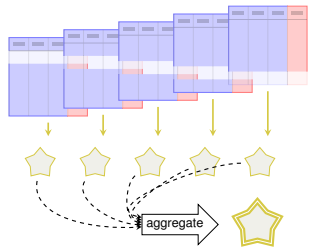


Tuning

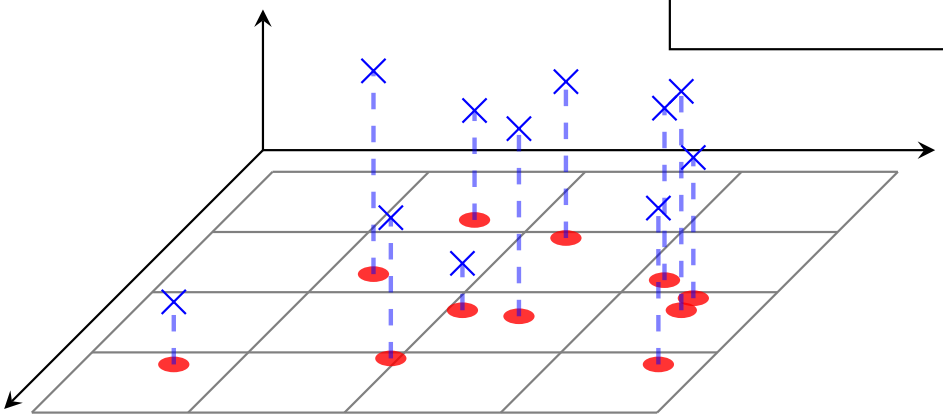


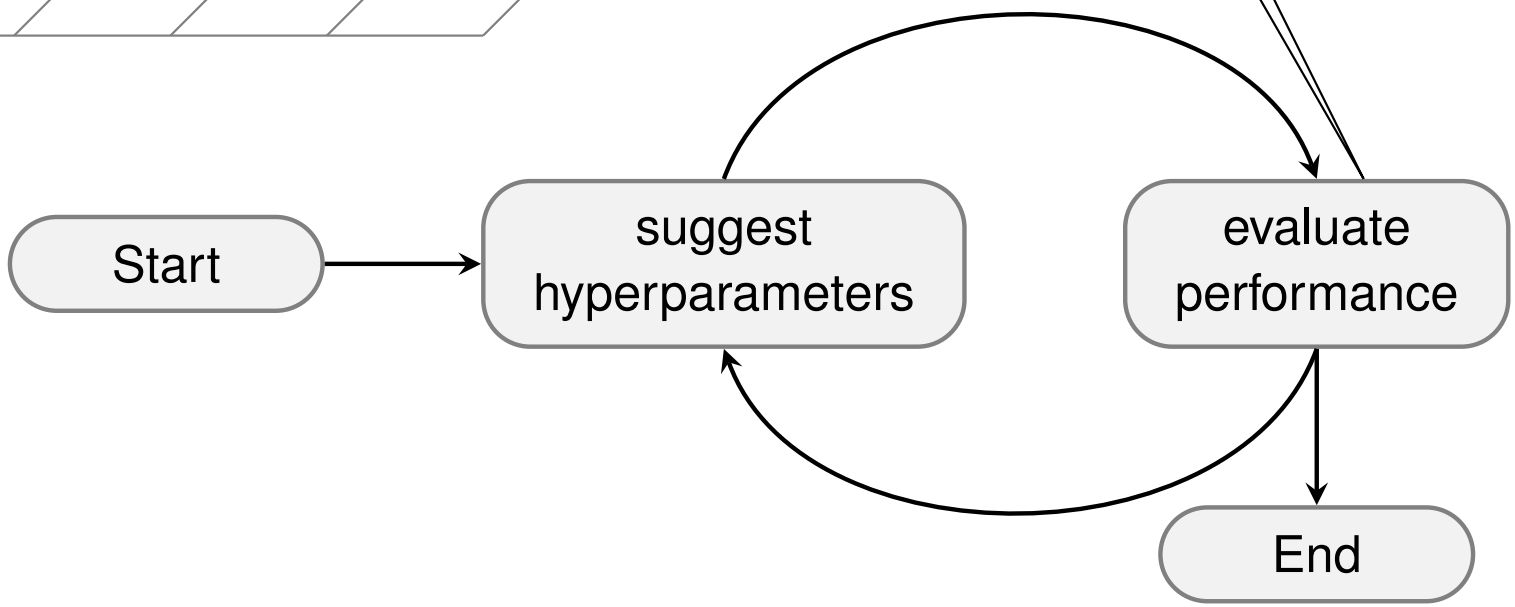


Tuning

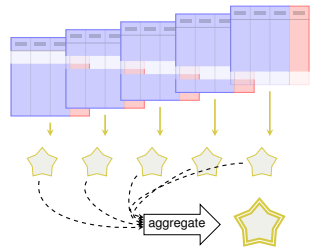


`Learner` (`lrn()`)
`Task`
`Resampling` (`rsmp()`)
`Measure` (`msr()`)

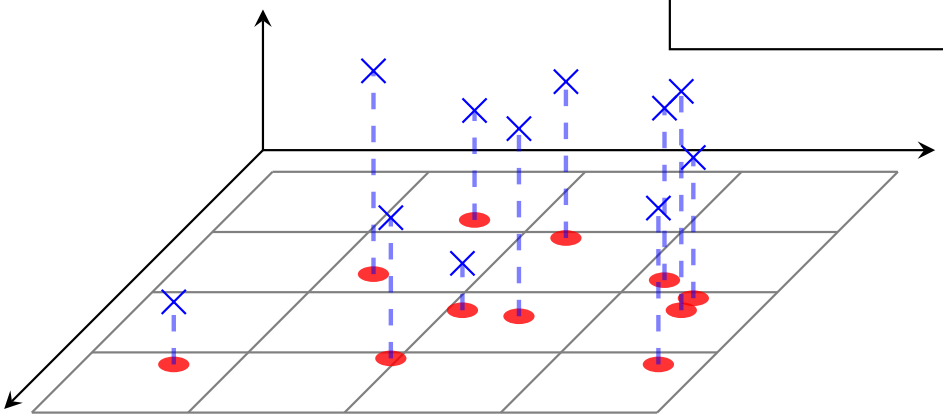




Tuning



`Learner` (`lrn()`)
`Task`
`Resampling` (`rsmp()`)
`Measure` (`msr()`)



Start

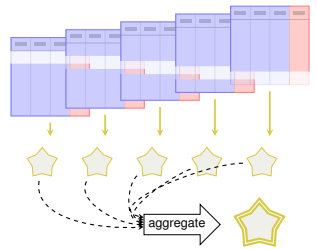
suggest hyperparameters

evaluate performance

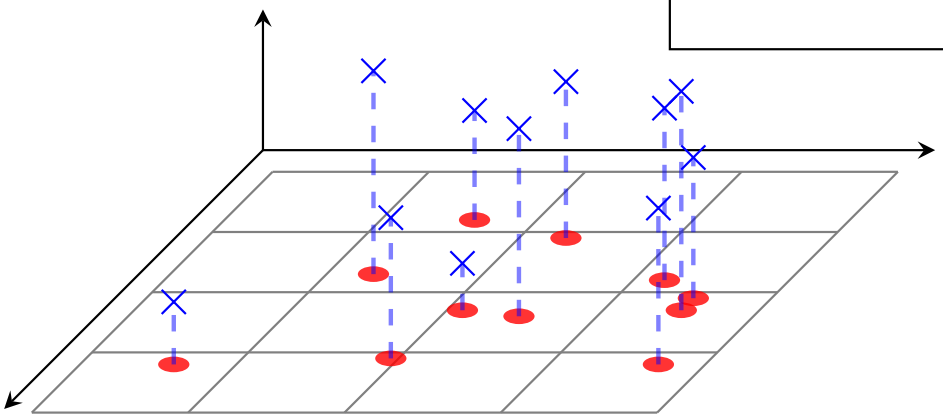
End

search space
`Tuner`

Tuning



`Learner` (`lrn()`)
`Task`
`Resampling` (`rsmp()`)
`Measure` (`msr()`)



Start

suggest hyperparameters

evaluate performance

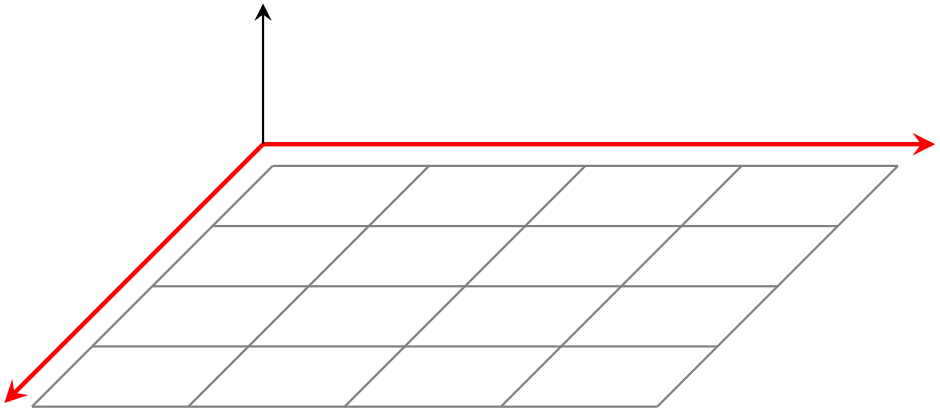
search space
`Tuner`

`Terminator (term())`

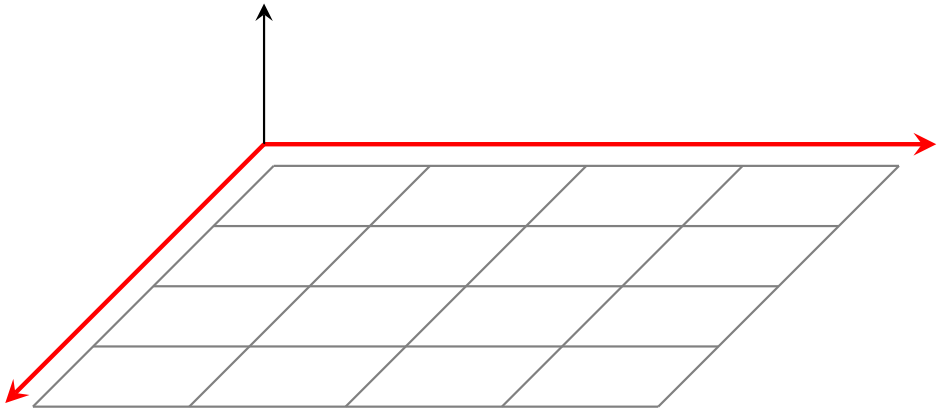
End

Search Space

Search Space

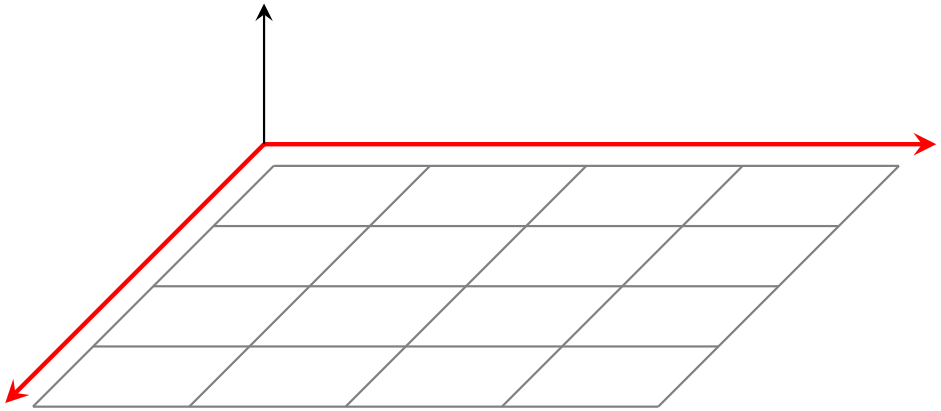


Search Space



```
ParamSet$new(list(param1, param2, ...))
```

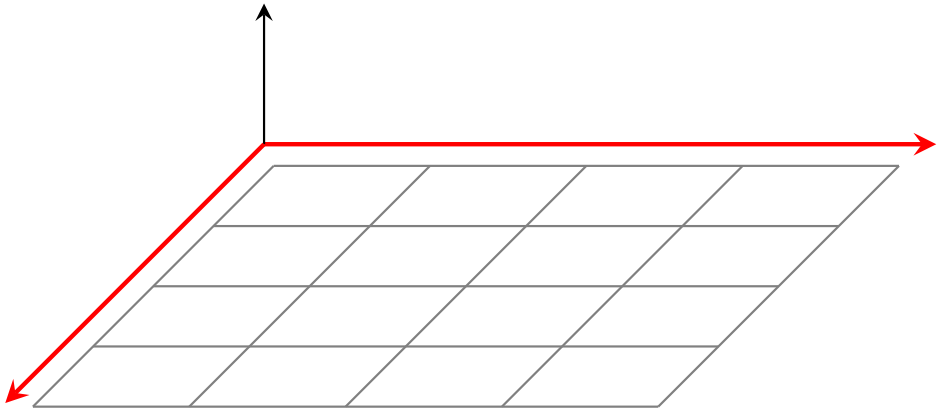
Search Space



```
ParamSet$new(list(param1, param2, ...))
```

Numerical parameter	`ParamDbl$new(id, lower, upper)`
Integer parameter	`ParamInt$new(id, lower, upper)`
Discrete parameter	`ParamFct$new(id, levels)`
Logical parameter	`ParamLgl$new(id)`
Untyped parameter	`ParamUty$new(id)`

Search Space



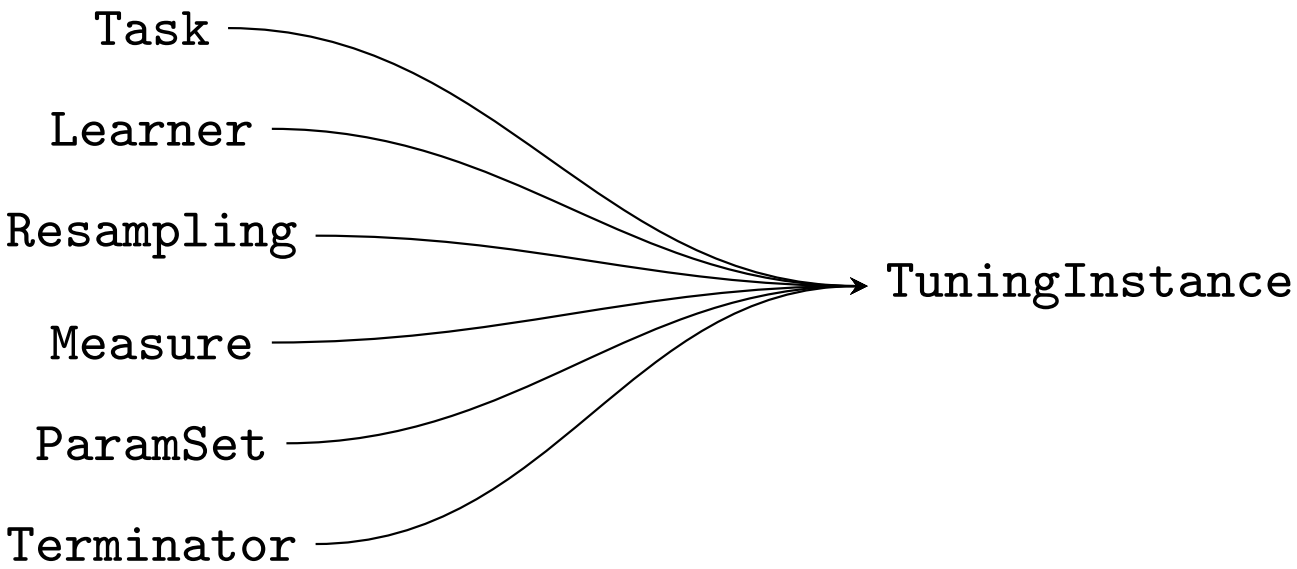
```
ParamSet$new(list(param1, param2, ...))
```

Numerical parameter	`ParamDbl$new(id, lower, upper)`
Integer parameter	`ParamInt$new(id, lower, upper)`
Discrete parameter	`ParamFct$new(id, levels)`
Logical parameter	`ParamLgl$new(id)`
Untyped parameter	`ParamUty$new(id)`

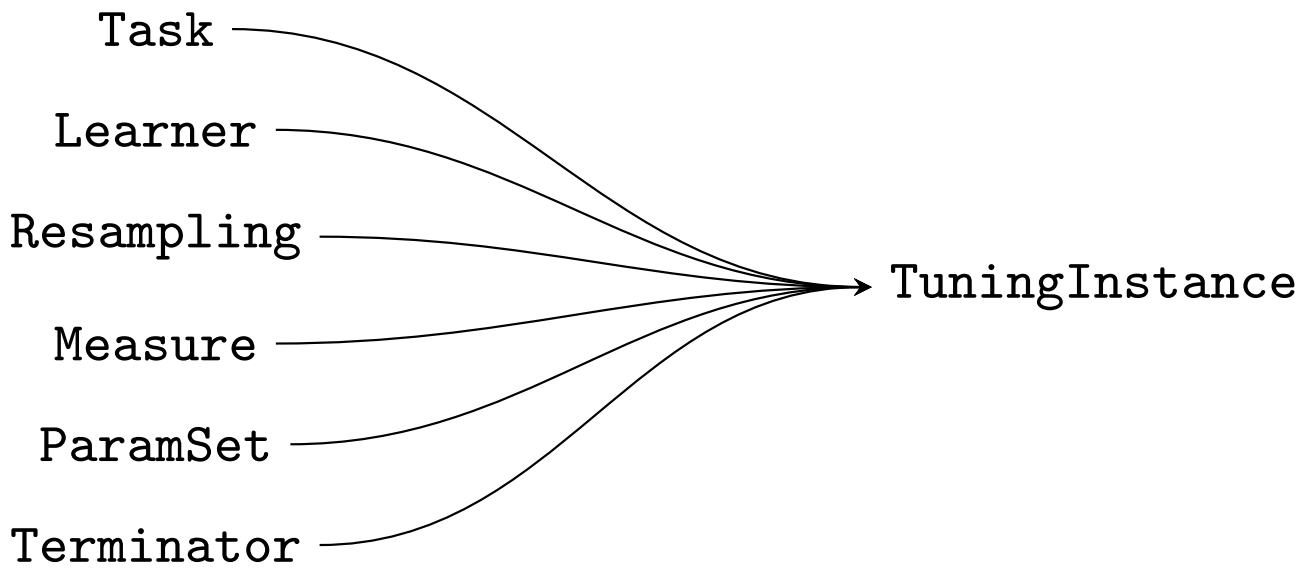
```
library("paradox")
searchspace_knn = ParamSet$new(list(
  ParamInt$new("k", 1, 20)
))
```

Tuning with mlr3tuning

Objects in Tuning



Objects in Tuning




```
library("mlr3tuning")

inst = TuningInstance$new(
  tsk("iris"), lrn("classif.kknn", kernel="rectangular"),
  rsmp("cv"), msr("classif.ce"),
  searchspace_knn, term("evals", n_evals = 20)
)
```

Tuning Method

- need to choose a *tuning method*

Tuning Method

- need to choose a *tuning method*
  - Available: Grid Search, Random Search, Simulated Annealing

Tuning Method

- need to choose a *tuning method*
  - Available: Grid Search, Random Search, Simulated Annealing
  - In process: Bayesian Optimization, Iterated F-racing, EAs, Hyperband

Tuning Method

- need to choose a *tuning method*
  - Available: Grid Search, Random Search, Simulated Annealing
  - In process: Bayesian Optimization, Iterated F-racing, EAs, Hyperband
  - Check `mlr_tuners`.

Tuning Method

- need to choose a *tuning method*
  - Available: Grid Search, Random Search, Simulated Annealing
  - In process: Bayesian Optimization, Iterated F-racing, EAs, Hyperband
  - Check `mlr_tuners`.
- `Tuner` class, load with `tnr()`, set parameters

Tuning Method

- need to choose a *tuning method*
  - Available: Grid Search, Random Search, Simulated Annealing
  - In process: Bayesian Optimization, Iterated F-racing, EAs, Hyperband
  - Check `mlr_tuners`.
- `Tuner` class, load with `tnr()`, set parameters
- ```
  gsearch = tnr("grid_search", resolution = 20)

  print(gsearch)
  #> <TunerGridSearch>
  #> * Parameters: resolution=20, batch_size=1
  #> * Packages: -
  #> * Properties: dependencies
  ```
  ```

Tuning Method

- need to choose a *tuning method*
  - Available: Grid Search, Random Search, Simulated Annealing
  - In process: Bayesian Optimization, Iterated F-racing, EAs, Hyperband
  - Check `mlr_tuners`.
- `Tuner` class, load with `tnr()`, set parameters
- ```
  gsearch = tnr("grid_search", resolution = 20)

  print(gsearch)
  #> <TunerGridSearch>
  #> * Parameters: resolution=20, batch_size=1
  #> * Packages: -
  #> * Properties: dependencies
  ```
- common parameter `batch_size` for parallelization
  ```

Calling the Tuner

```
gsearch$tune(inst)
```

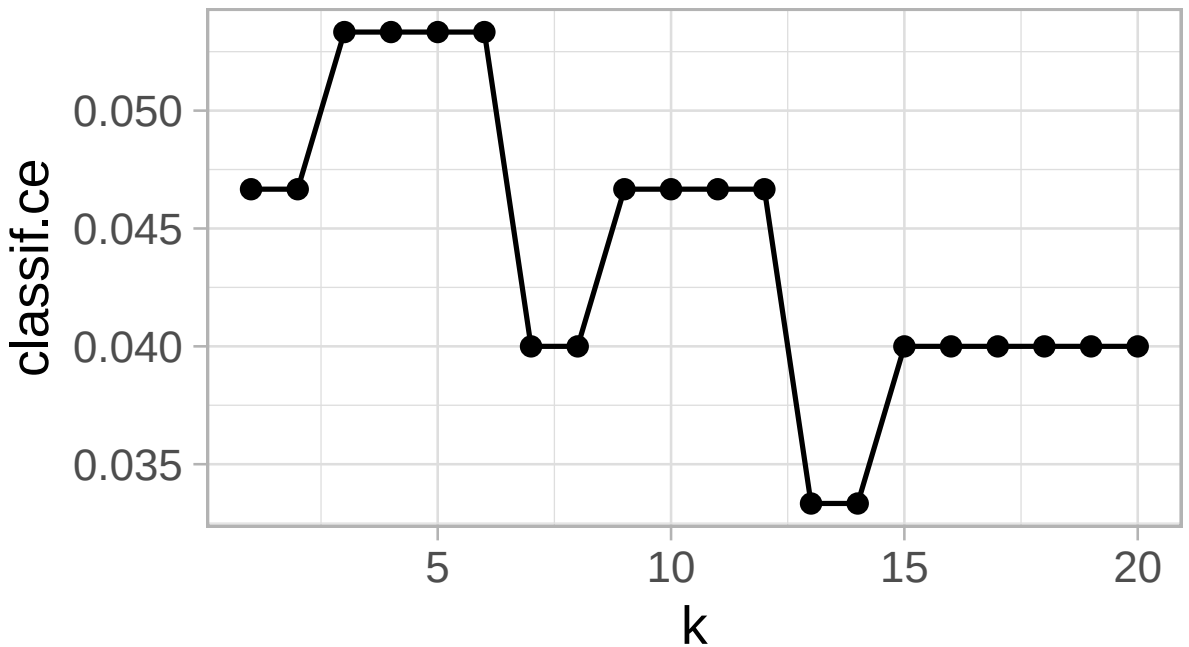
Calling the Tuner

```
gsearch$tune(inst)
```

```
inst$result

#> $tune_x
#> $tune_x$k
#> [1] 13
#>
#>
#> $params
#> $params$kernel
#> [1] "rectangular"
#>
#> $params$k
#> [1] 13
#>
#>
#> $perf
#> classif.ce
#>      0.033
```

Tuning Results

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



Recap

```
inst = TuningInstance$new(
  tsk("iris"), lrn("classif.kknn", kernel="rectangular"),
  rsmp("cv"), msr("classif.ce"),
  searchspace_knn, term("evals", n_evals = 20)
)

gsearch = tnr("grid_search", resolution = 20)

gsearch$tune(inst)
```

Note: Other tuning methods need a terminator, which can be:

- iterations
- performance threshold
- model runtime and wall-clock time

Check `mlr_terminators`.

Parameter Transformation

Parameter Transformation

- Sometimes we do not want to sample evenly from a range

Parameter Transformation

- Sometimes we do not want to sample evenly from a range
- $k = 1$ vs. $k = 2$ probably more interesting than $k = 101$ vs. $k = 102$

Parameter Transformation

- Sometimes we do not want to sample evenly from a range
- $k = 1$ vs. $k = 2$ probably more interesting than $k = 101$ vs. $k = 102$

$\Rightarrow$ Transformations

Parameter Transformation

- Sometimes we do not want to sample evenly from a range
- $k = 1$ vs. $k = 2$ probably more interesting than $k = 101$ vs. $k = 102$

$\Rightarrow$ Transformations

- Part of `ParamSet`

Parameter Transformation

- Sometimes we do not want to sample evenly from a range
- $k = 1$ vs. $k = 2$ probably more interesting than $k = 101$ vs. $k = 102$

$\Rightarrow$ Transformations

- Part of `ParamSet`

Example:

Parameter Transformation

- Sometimes we do not want to sample evenly from a range
- $k = 1$ vs. $k = 2$ probably more interesting than $k = 101$ vs. $k = 102$

$\Rightarrow$ Transformations

- Part of `ParamSet`

Example:

1. sample from $\log(1) \ldots \log(100)$ (`k_before_trafo`)

Parameter Transformation

- Sometimes we do not want to sample evenly from a range
- $k = 1$ vs. $k = 2$ probably more interesting than $k = 101$ vs. $k = 102$

$\Rightarrow$ Transformations

- Part of `ParamSet`

Example:

1. sample from $\log(1) \ldots \log(100)$ (`k_before_trafo`)
2. transform by $\exp()$ in `trafo` function

Parameter Transformation

- Sometimes we do not want to sample evenly from a range
- $k = 1$ vs. $k = 2$ probably more interesting than $k = 101$ vs. $k = 102$

$\Rightarrow$ Transformations

- Part of `ParamSet`

Example:

1. sample from $\log(1) \ldots \log(100)$ (`k_before_trafo`)
2. transform by $\exp()$ in `trafo` function
3. don't forget to `round` (k must be integer)

Parameter Transformation

- Sometimes we do not want to sample evenly from a range
- $k = 1$ vs. $k = 2$ probably more interesting than $k = 101$ vs. $k = 102$

$\Rightarrow$ Transformations

- Part of `ParamSet`

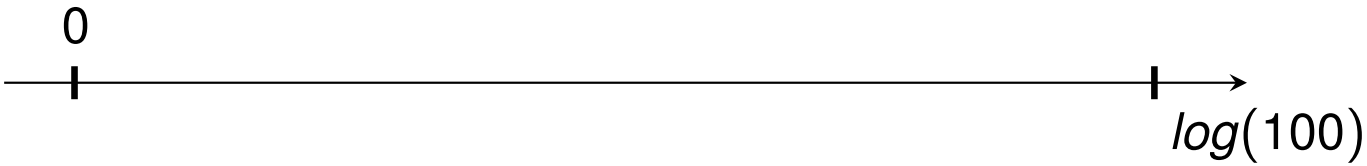
Example:

1. sample from $\log(1) \ldots \log(100)$ (`k_before_trafo`)
2. transform by $\exp()$ in `trafo` function
3. don't forget to `round` (k must be integer)

```
searchspace_knn_trafo = ParamSet$new(list(
  ParamDbl$new("k_before_trafo", log(1), log(100))
))
searchspace_knn_trafo$trafo = function(x, param_set) {
  return(list(k = round(exp(x$k_before_trafo))))
}
```

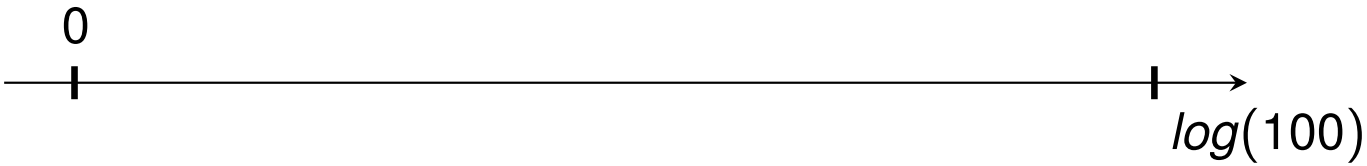
Parameter Transformation

What is our transformation doing?



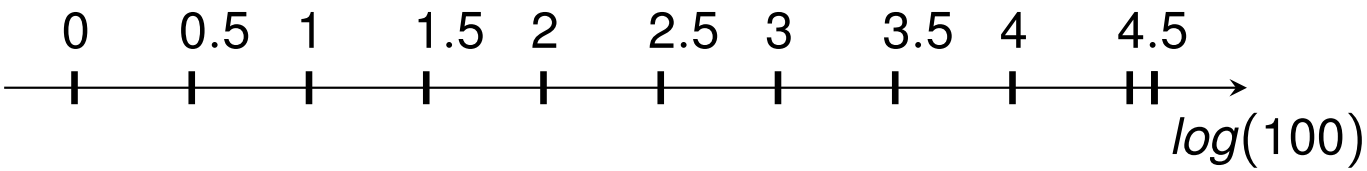
Parameter Transformation

What is our transformation doing?



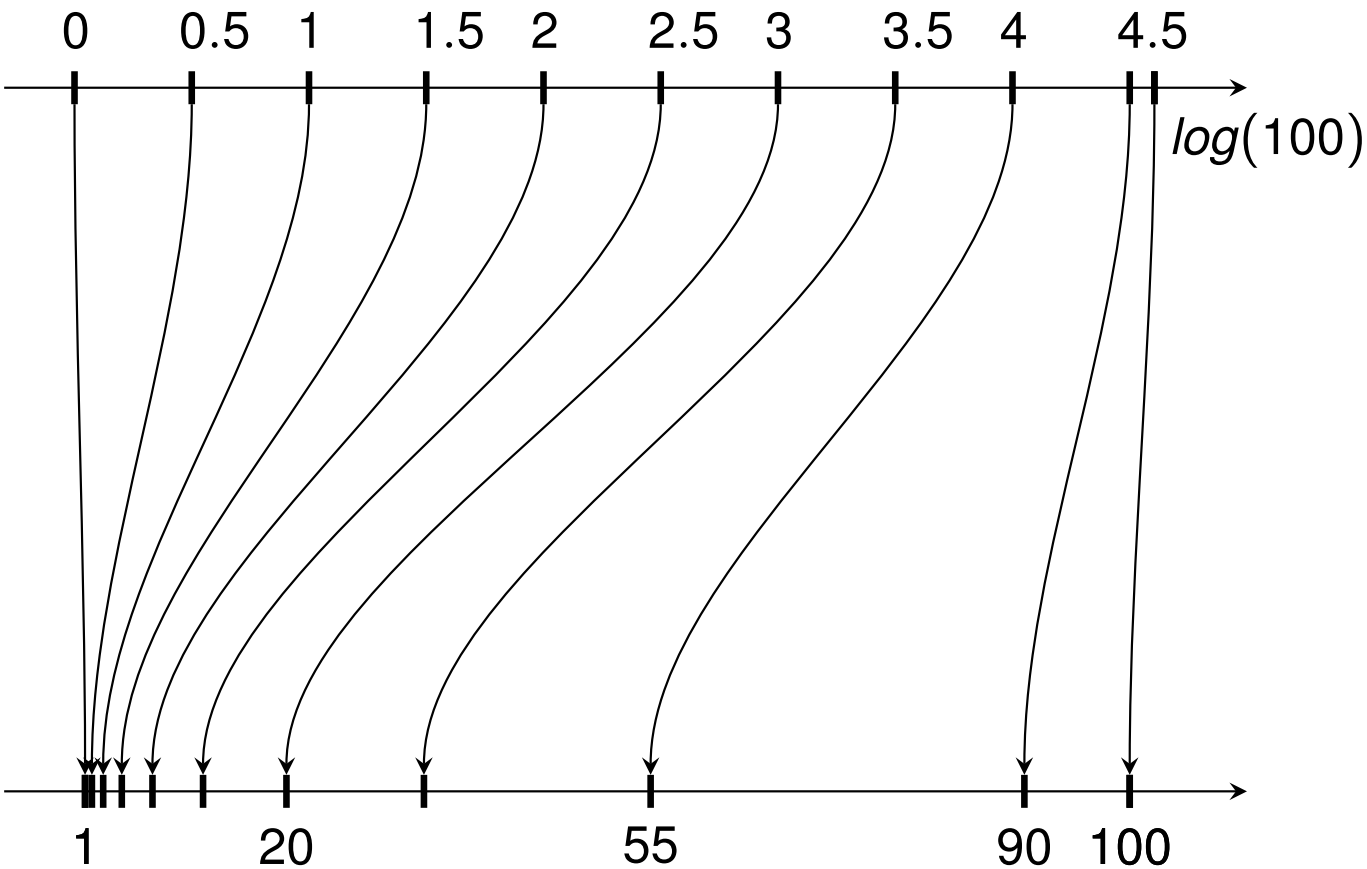
Parameter Transformation

What is our transformation doing?



Parameter Transformation

What is our transformation doing?



Parameter Transformation

Tuning again. . .

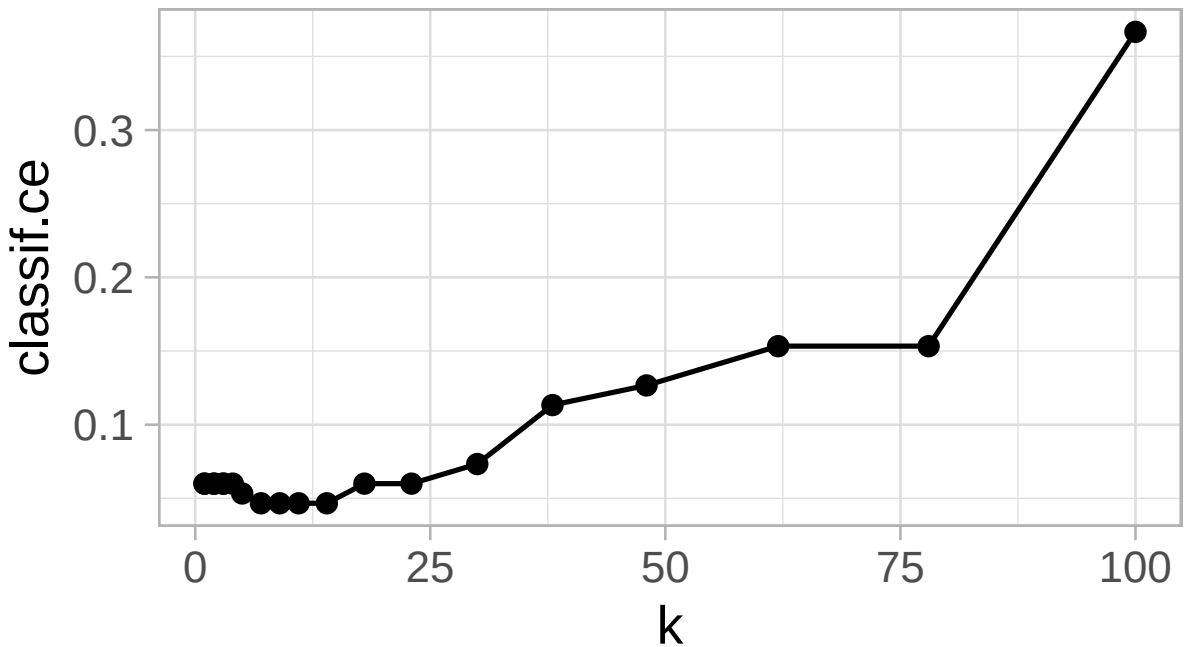
Parameter Transformation

Tuning again...

```
inst$result

#> $tune_x
#> $tune_x$k_before_trafo
#> [1] 2.7
#>
#>
#> $params
#> $params$kernel
#> [1] "rectangular"
#>
#> $params$k
#> [1] 14
#>
#>
#> $perf
#> classif.ce
#>      0.047
```

Parameter Transformation

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



Nested Resampling

Nested Resampling

- Need to perform nested resampling to estimate tuned learner performance

Nested Resampling

- Need to perform nested resampling to estimate tuned learner performance

$\Rightarrow$ Treat tuning as if it were a `Learner`!

- Training:
  1. Tune model using (inner) resampling
  2. Train final model with best parameters on all (i.e. outer resampling) data
- Predicing: Just use final model

Nested Resampling

- Need to perform nested resampling to estimate tuned learner performance

$\Rightarrow$ Treat tuning as if it were a `Learner`!

  - Training:
    1. Tune model using (inner) resampling
    2. Train final model with best parameters on all (i.e. outer resampling) data
  - Predicing: Just use final model

- **AutoTuner**

Nested Resampling

- Need to perform nested resampling to estimate tuned learner performance

$\Rightarrow$ Treat tuning as if it were a `Learner`!

  - Training:
    1. Tune model using (inner) resampling
    2. Train final model with best parameters on all (i.e. outer resampling) data
  - Predicing: Just use final model

- **AutoTuner**

```
optlrn = AutoTuner$new(lrn("classif.kknn", kernel="rectangular"),
  rsmp("cv"), msr("classif.ce"), searchspace_knn,
  term("none"), tnr("grid_search", resolution = 20))
```

Nested Resampling

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

Nested Resampling

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

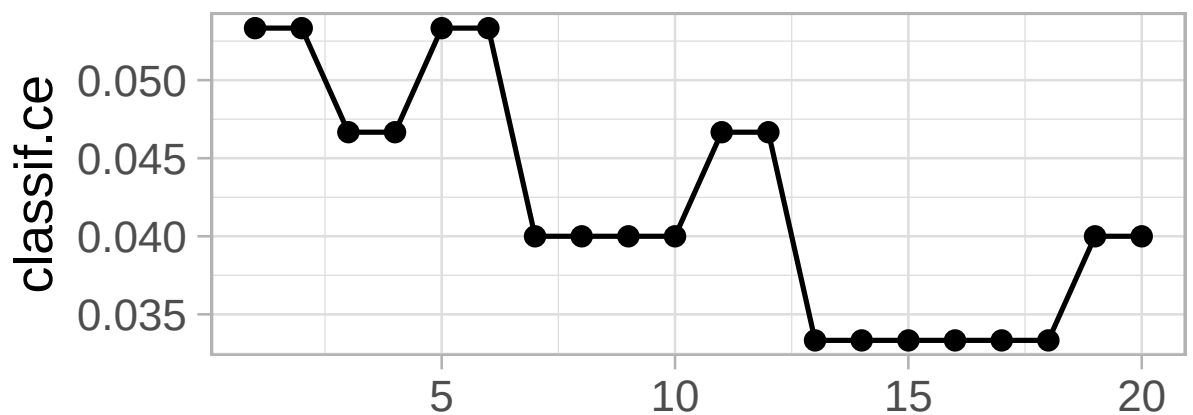
```
optlrn$model$learner

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

Nested Resampling

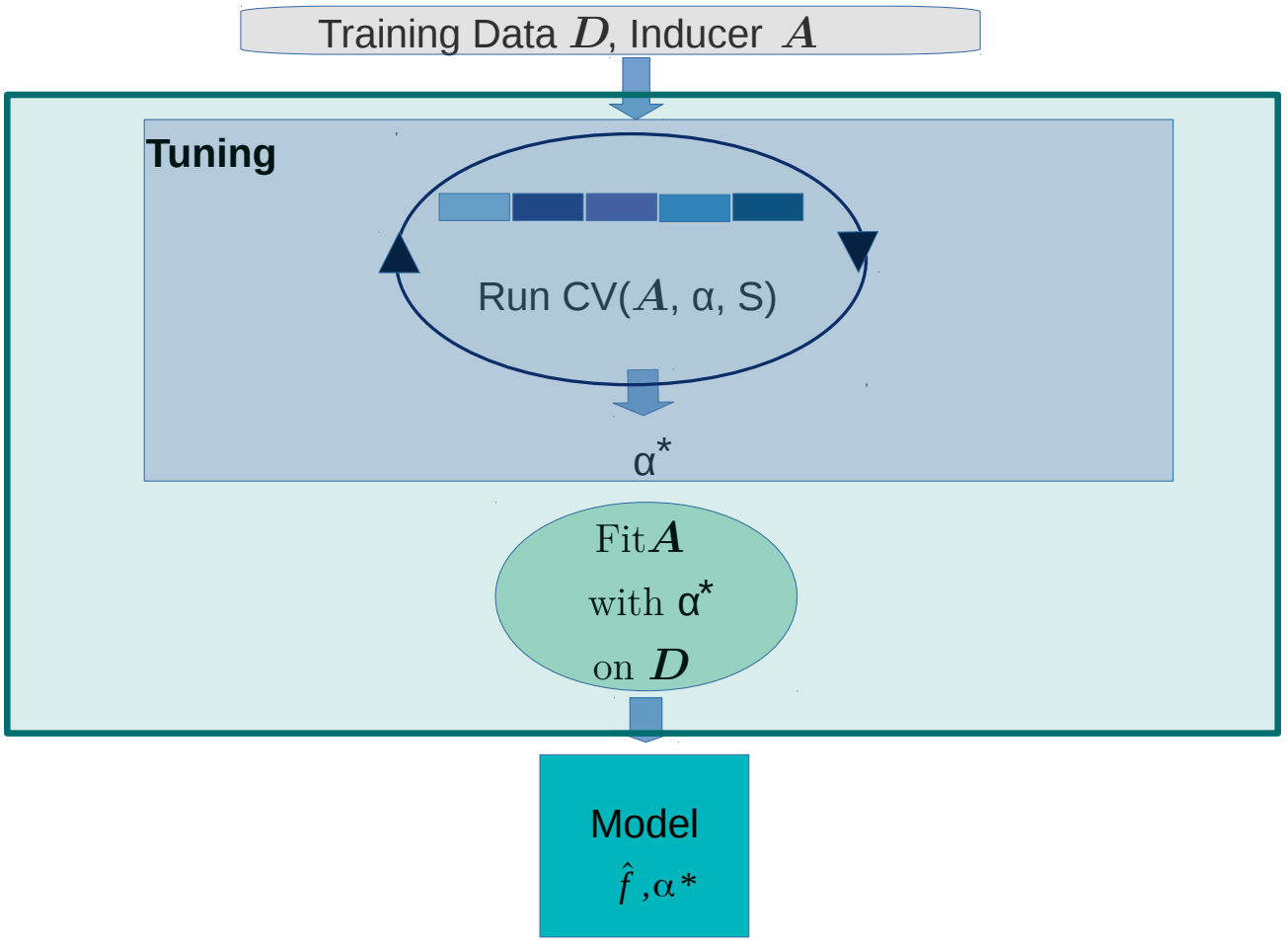
```
optlrn$train(tsk("iris"))
```

```
optlrn$model$learner

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

```
ggplot(optlrn$model$tuning_instance$archive(unnest = "params"),
  aes(x = k, y = classif.ce)) + geom_line() + geom_point()
```



Nested Resampling



Nested Resampling

```
resample(tsk("iris"), optlrn, rsmp("cv"))

#> <ResampleResult> of 10 iterations
#> * Task: iris
#> * Learner: classif.kknn.tuned
#> * Performance: 0.053 [classif.ce]
#> * Warnings: 0 in 0 iterations
#> * Errors: 0 in 0 iterations
```

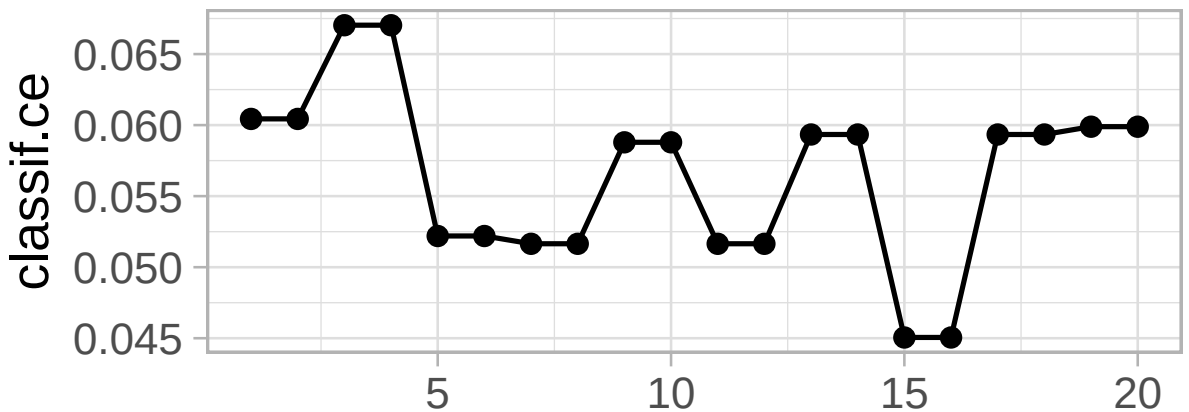
Nested Resampling

```
res = resample(tsk("iris"), optlrn, rsmp("cv"),
  store_model = TRUE)
```

Nested Resampling

```
res = resample(tsk("iris"), optlrn, rsmp("cv"),
  store_model = TRUE)
```

```
ggplot(res$learners[[1]]$
       model$tuning_instance$archive(unnest = "params"),
  aes(x = k, y = classif.ce)) + geom_line() + geom_point()
```



mlr3tuning recap

Tuning with mlr3tuning

Tuning a `Learner`

1. Construct a `TuningInstance`
   - `Task`—the Data to tune over
   - `Learner`—the algorithm to tune
   - `Resampling`—the resampling method to use
   - `Measure`—how to evaluate performance
   - `ParamSet`—the search space, possibly with `trafo`
   - `Terminator`—when to quit
2. Create a `Tuner`
   - Usually using `tnr()`
   - May have some parameters, e.g. `batch_size`
3. Call `tuner$tune()`

Nested Resampling

1. Construct an `AutoTuner`
   - Constructor takes all arguments of a `TuningInstance` *except* *`Task`*
   - Also takes the `Tuner` as an argument
2. Use like a normal `Learner` in `resample()` and `benchmark()`

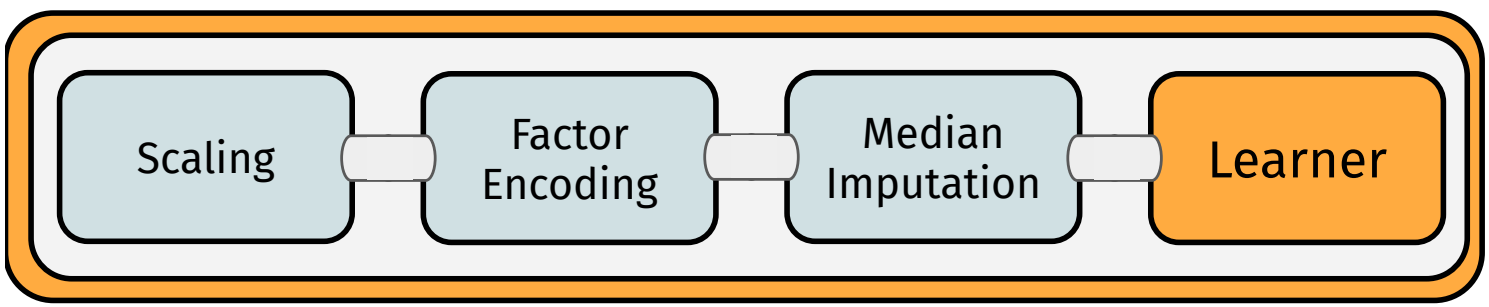
mlr3pipelines into

mlr3pipelines

Machine Learning Workflows:

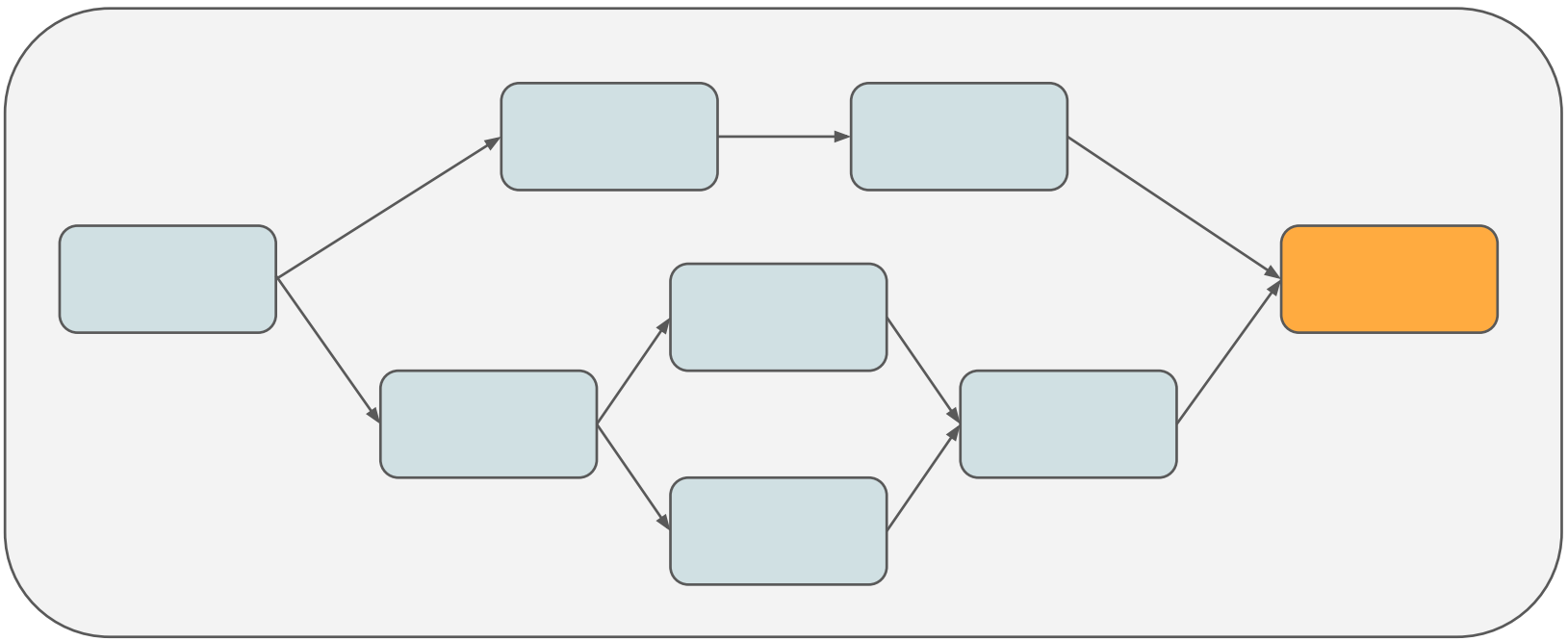
- **Preprocessing**: Feature extraction, feature selection, missing data imputation,...
- **Ensemble methods**: Model averaging, model stacking
- `mlr3`: modular model fitting

$\Rightarrow$ `mlr3pipelines`: modular ML workflows



Machine Learning Workflows

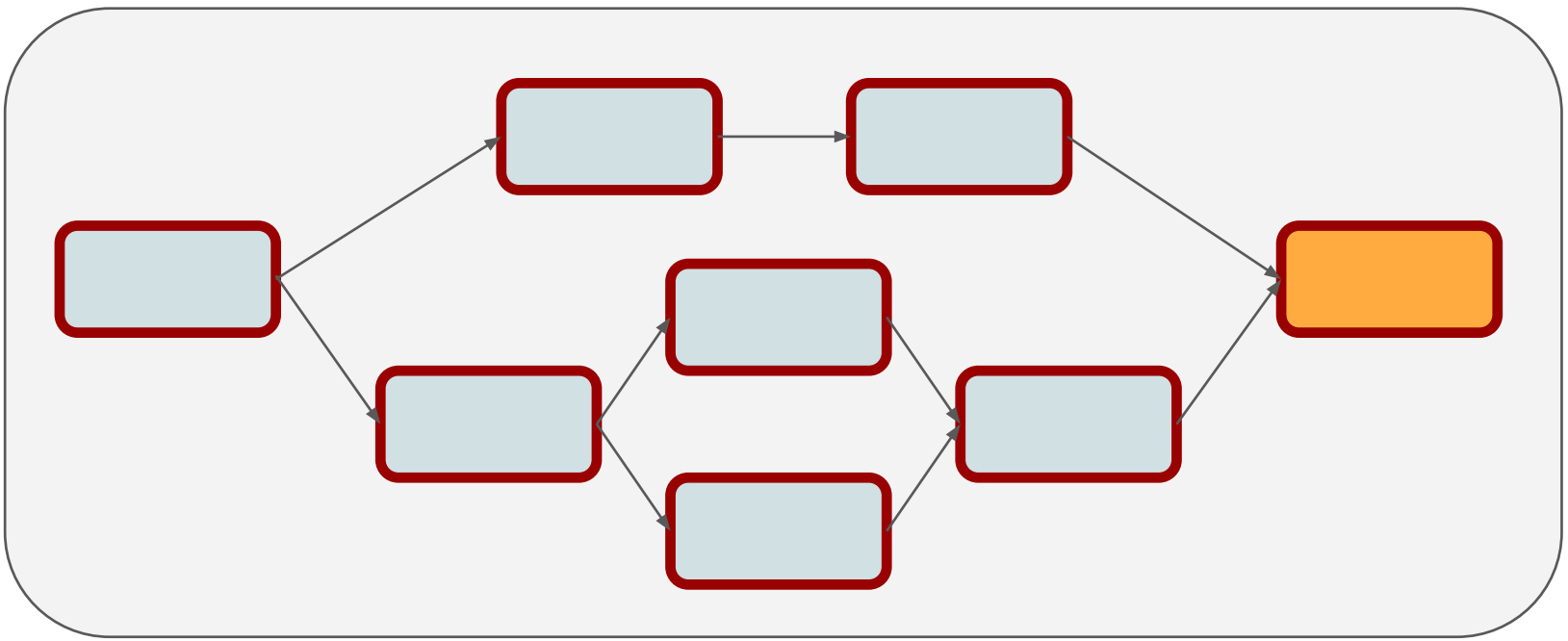
– what do they look like?



Machine Learning Workflows

– what do they look like?

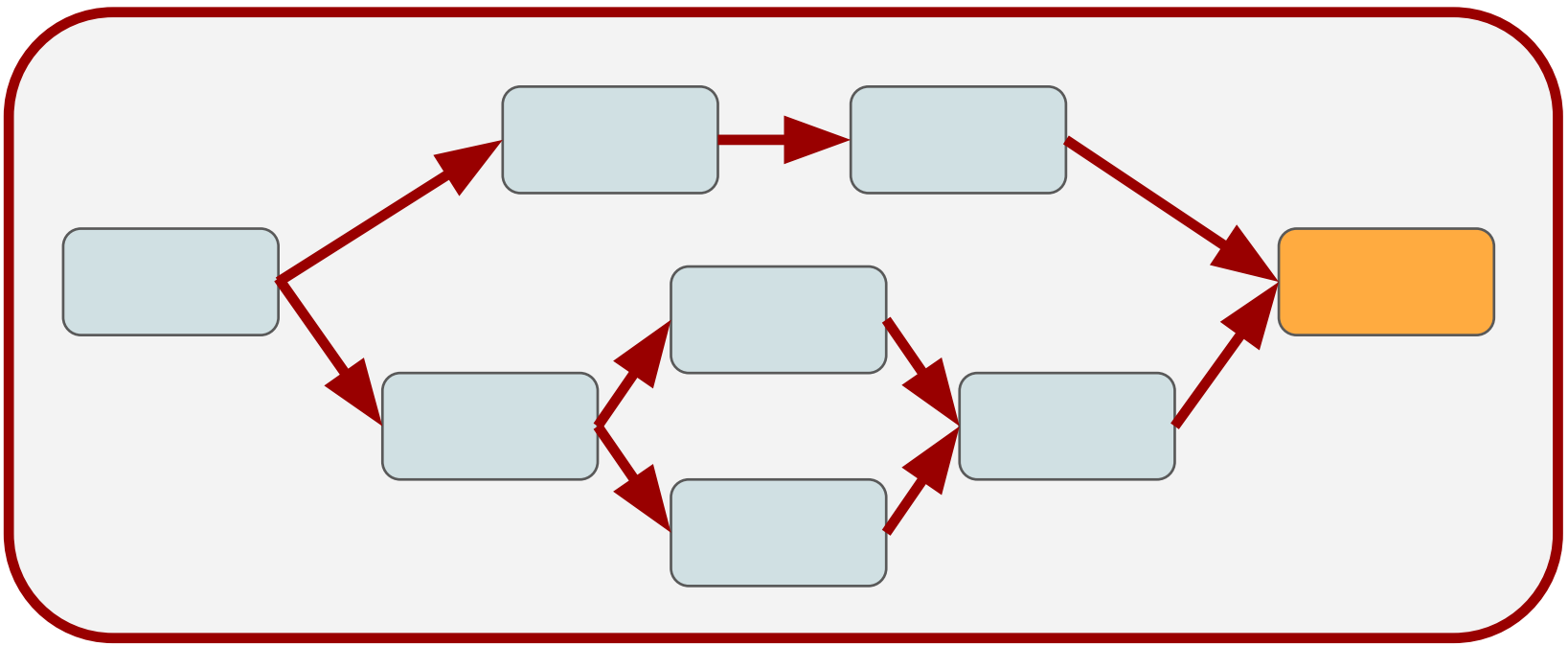
- **Building blocks**: *what* is happening? $\rightarrow$ `PipeOp`



Machine Learning Workflows

– what do they look like?

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

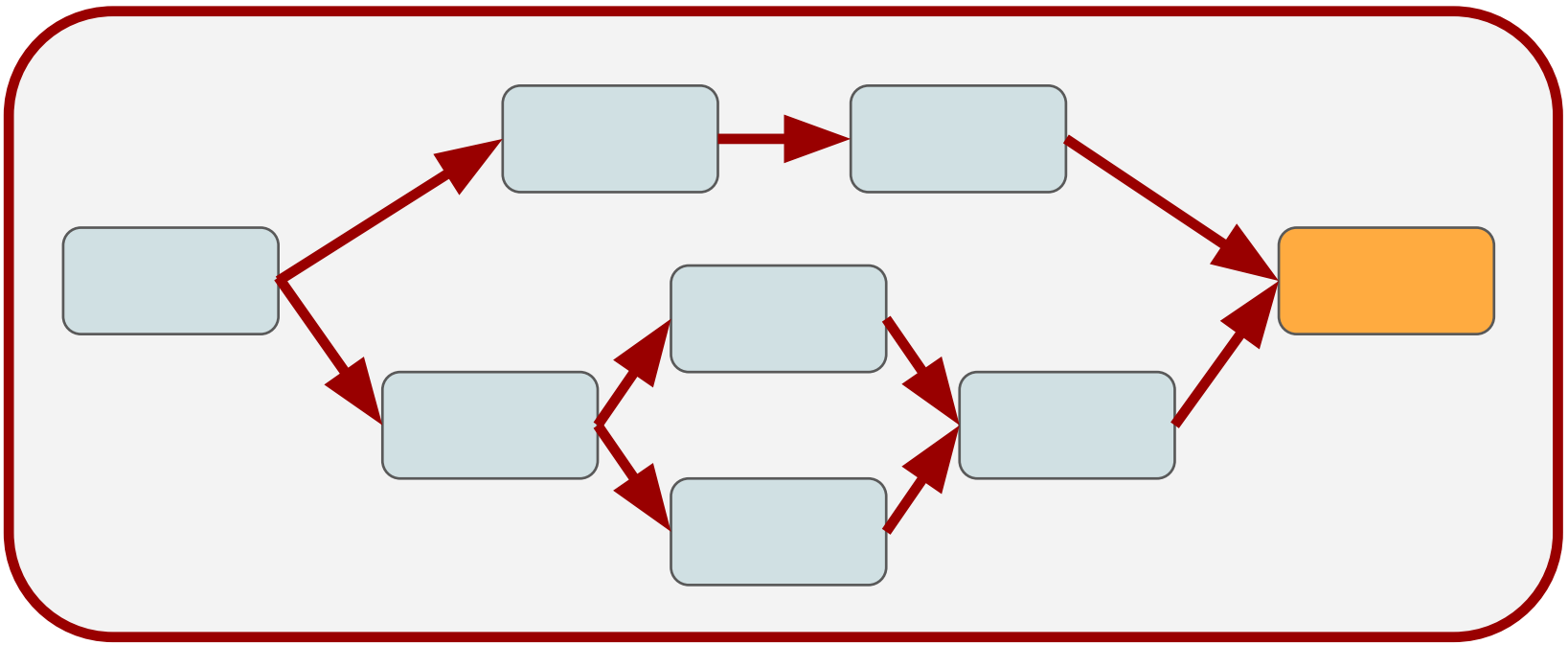


Machine Learning Workflows

– 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

The Building Blocks

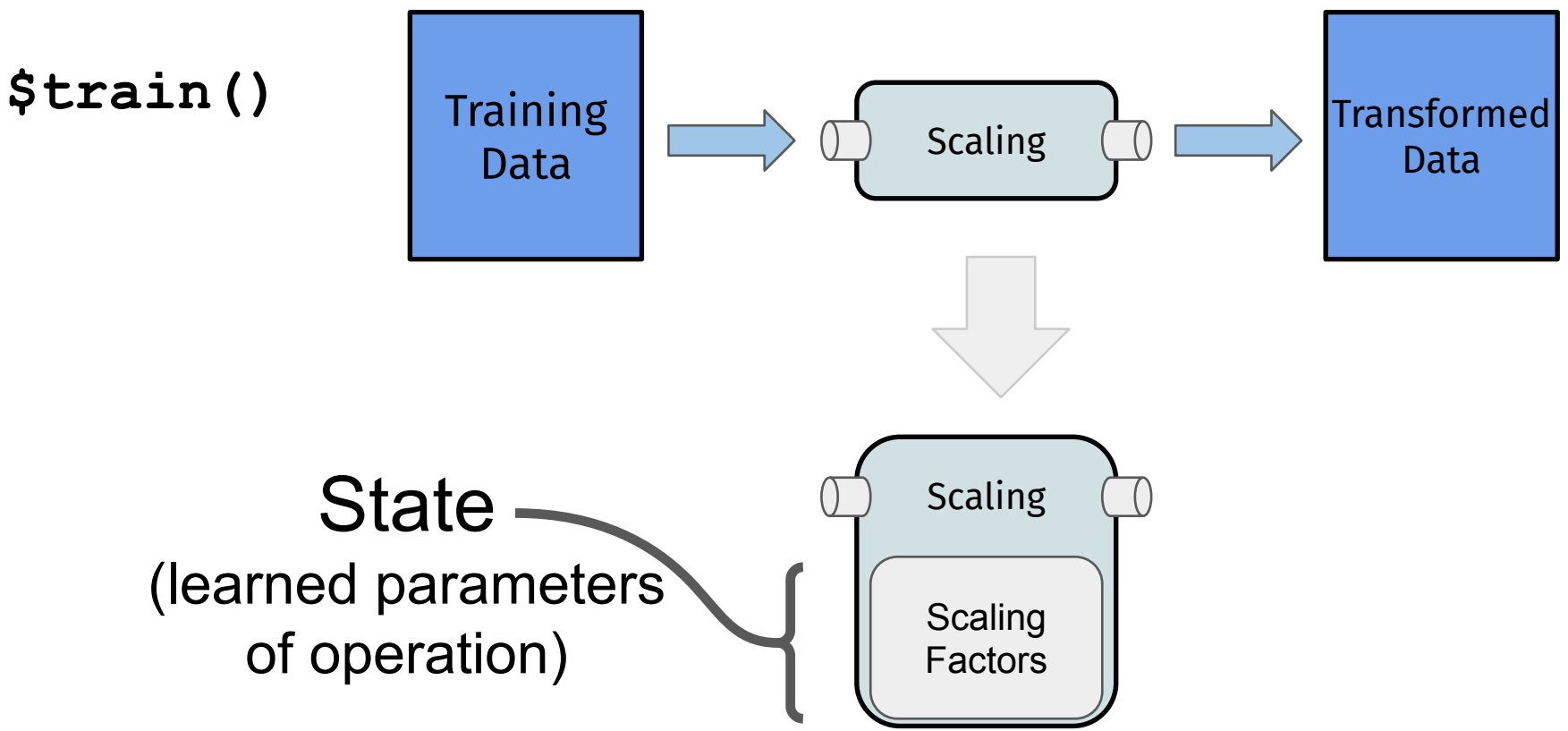
PipeOp: Single Unit of Data Operation

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

The Building Blocks

PipeOp: Single Unit of Data Operation

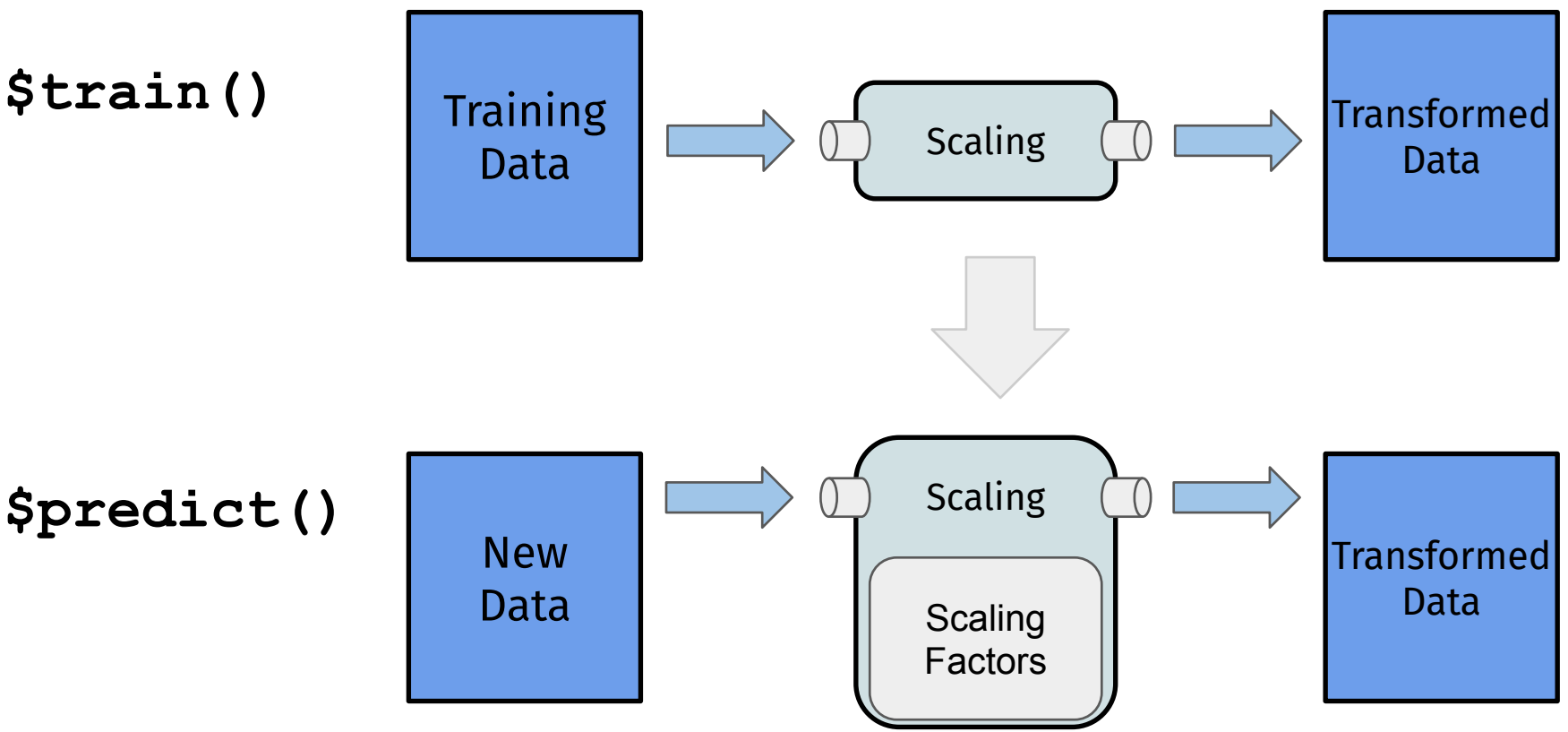
- `pip = po("scale")` to construct
- `pip$train()`: process data and create `pip$state`



The Building Blocks

PipeOp: Single Unit of Data Operation

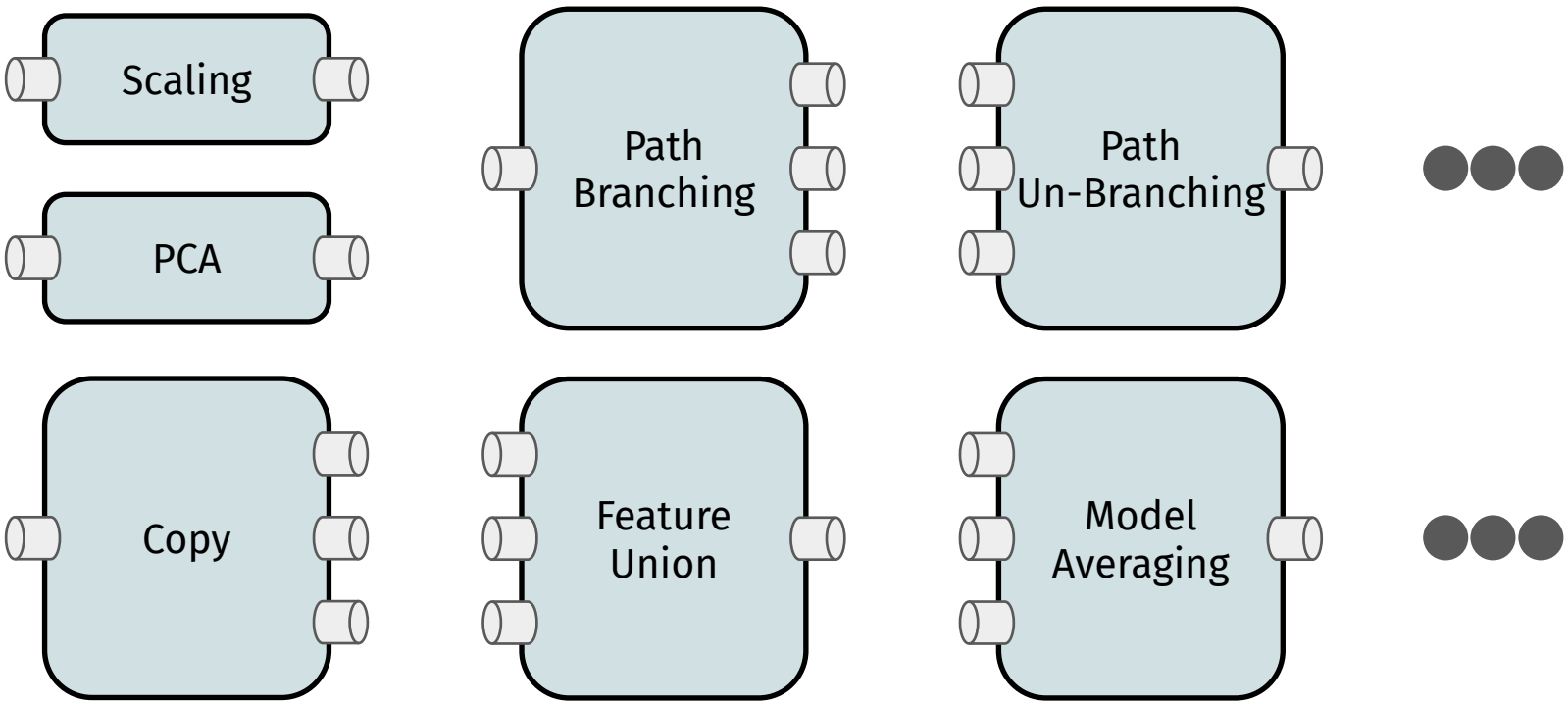
- `pip = po("scale")` to construct
- `pip$train()`: process data and create `pip$state`
- `pip$predict()`: process data depending on the `pip$state`



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



The Building Blocks

```
po = po("scale")

trained = po$train(list(task))

trained[[1]]$head(3)

#>    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#>     <fctr>        <num>       <num>        <num>       <num>
#> 1:  setosa         -1.3        -1.3         -0.9        1.02
#> 2:  setosa         -1.3        -1.3         -1.1       -0.13
#> 3:  setosa         -1.4        -1.3         -1.4        0.33
```

The Building Blocks

```
po = po("scale")

trained = po$train(list(task))

trained[[1]]$head(3)

#>    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#>     <fctr>        <num>       <num>        <num>       <num>
#> 1:  setosa         -1.3        -1.3         -0.9        1.02
#> 2:  setosa         -1.3        -1.3         -1.1       -0.13
#> 3:  setosa         -1.4        -1.3         -1.4        0.33
```

```
head(po$state, 2)

#> $center
#> Petal.Length  Petal.Width Sepal.Length  Sepal.Width
#>          3.8          1.2          5.8          3.1
#>
#> $scale
#> Petal.Length  Petal.Width Sepal.Length  Sepal.Width
#>         1.77         0.76         0.83         0.44
```

The Building Blocks

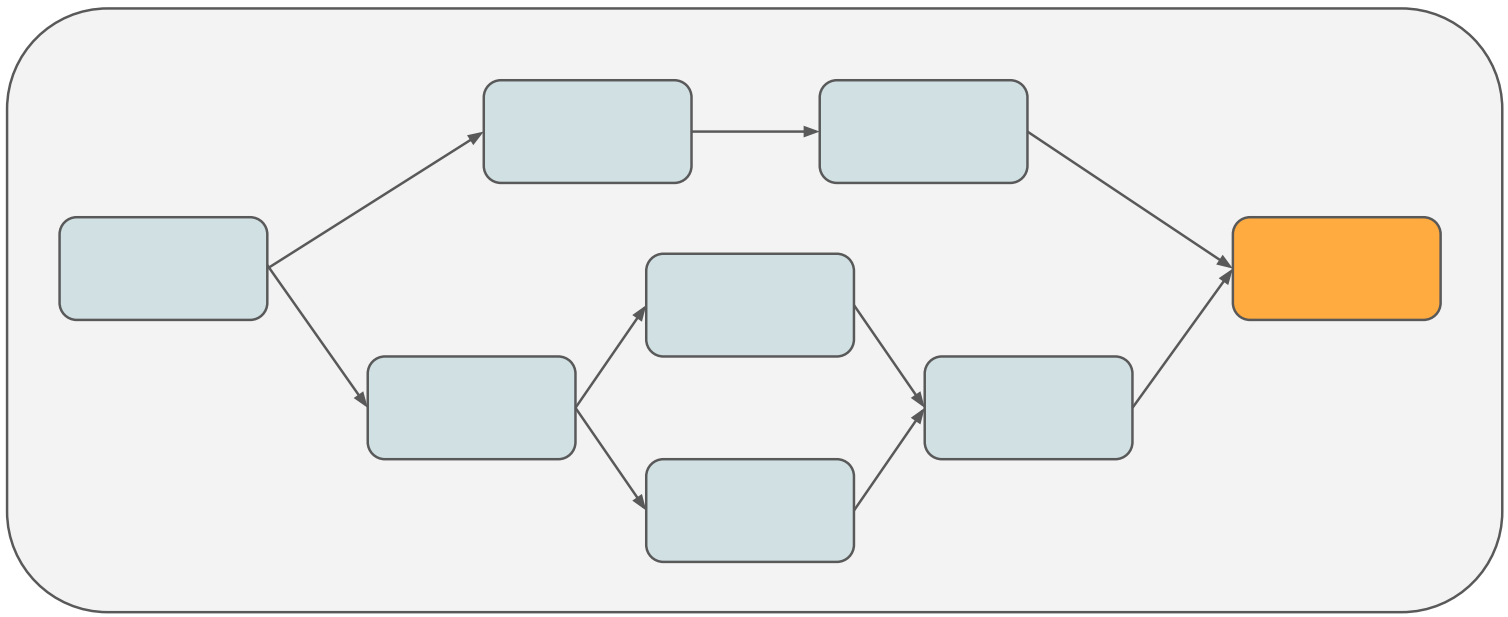
```
po = po("scale")

trained = po$train(list(task))

trained[[1]]$head(3)

#>    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#>     <fctr>        <num>       <num>        <num>       <num>
#> 1:  setosa         -1.3        -1.3         -0.9        1.02
#> 2:  setosa         -1.3        -1.3         -1.1       -0.13
#> 3:  setosa         -1.4        -1.3         -1.4        0.33
```

```
smalltask = task$clone()$filter(1:3)
po$predict(list(smalltask))[[1]]$data()

#>    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#>     <fctr>        <num>       <num>        <num>       <num>
#> 1:  setosa         -1.3        -1.3         -0.9        1.02
#> 2:  setosa         -1.3        -1.3         -1.1       -0.13
#> 3:  setosa         -1.4        -1.3         -1.4        0.33
```

Graph Operations

The Structure

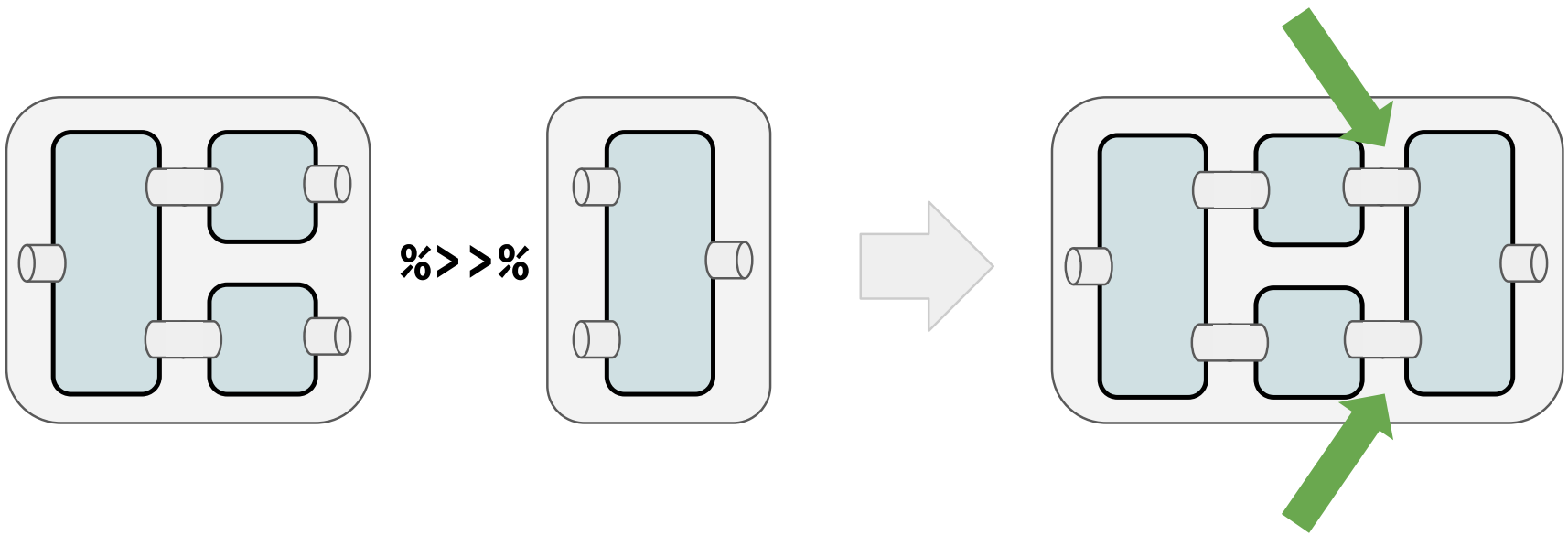
Graph Operations



The Structure

Graph Operations

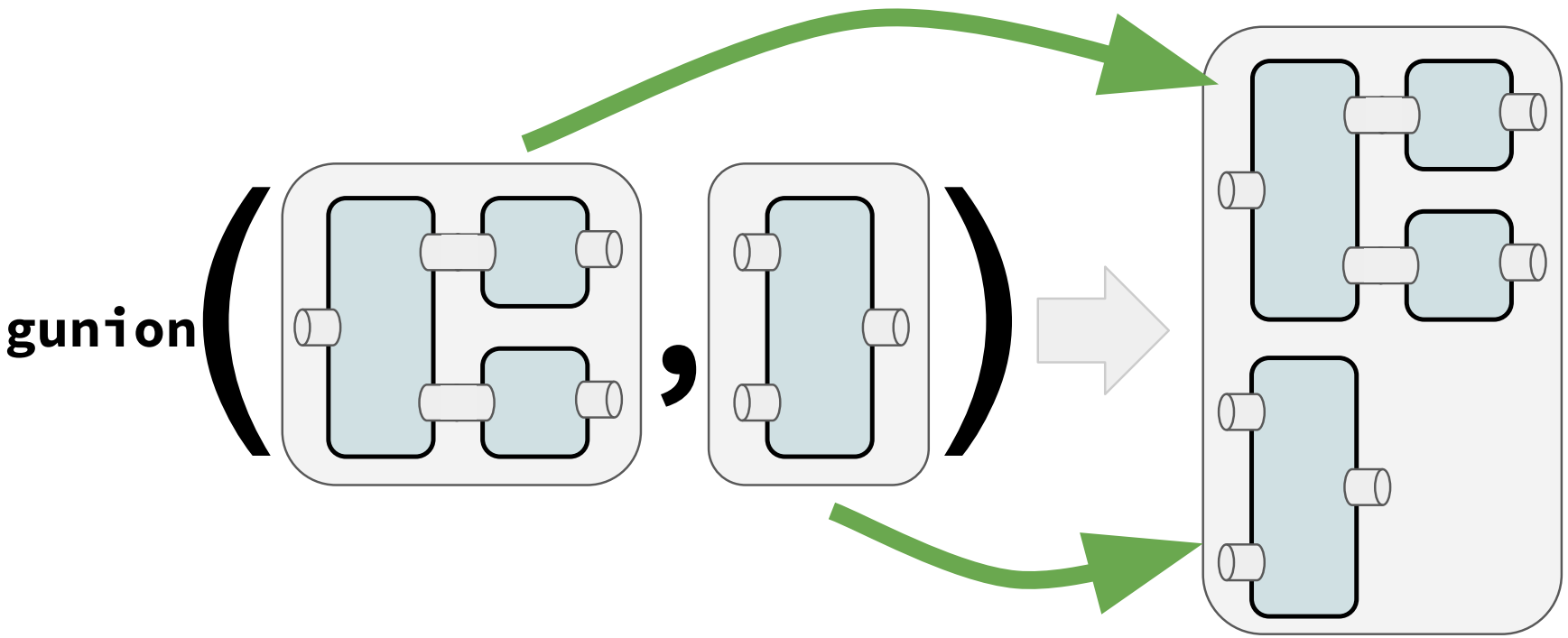
- The %>>%-operator concatenates `Graph`s and `PipeOp`s



The Structure

Graph Operations

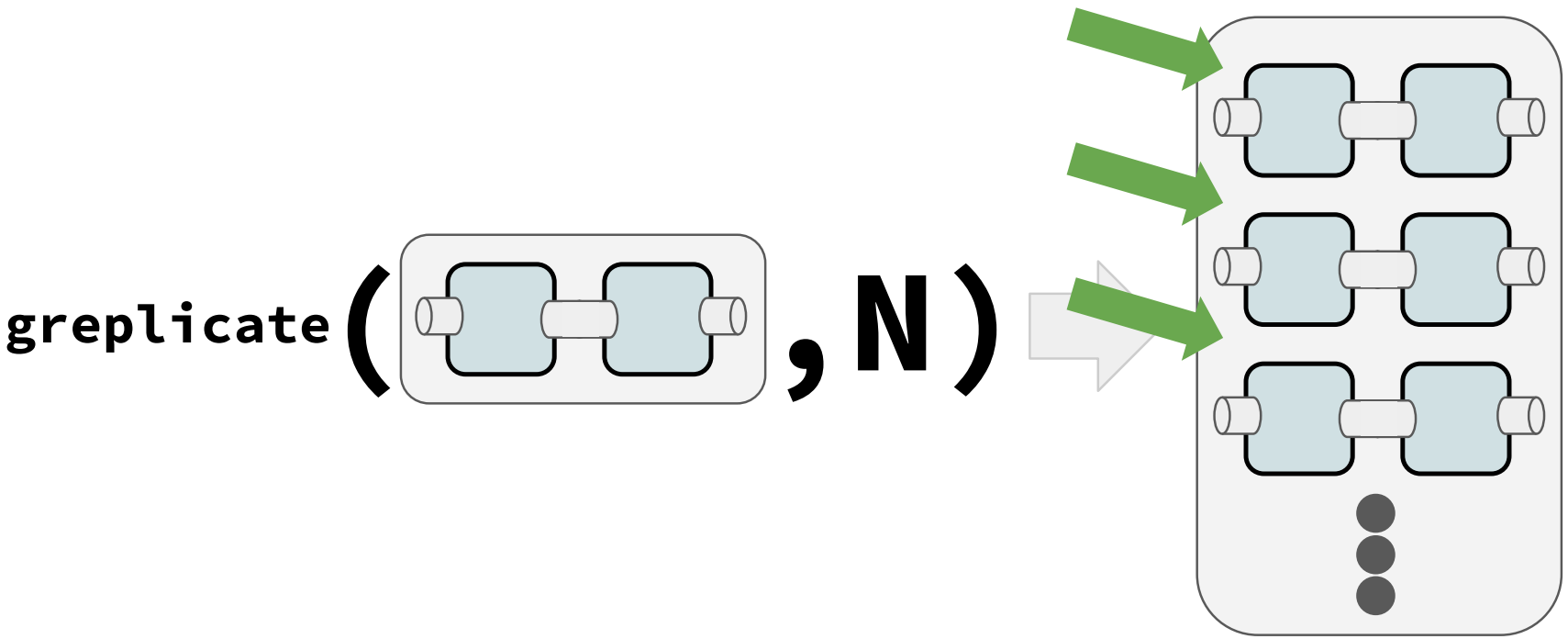
- The %>>%-operator concatenates `Graph`s and `PipeOp`s
- The `gunion()`-function unites `Graph`s and `PipeOp`s



The Structure

Graph Operations

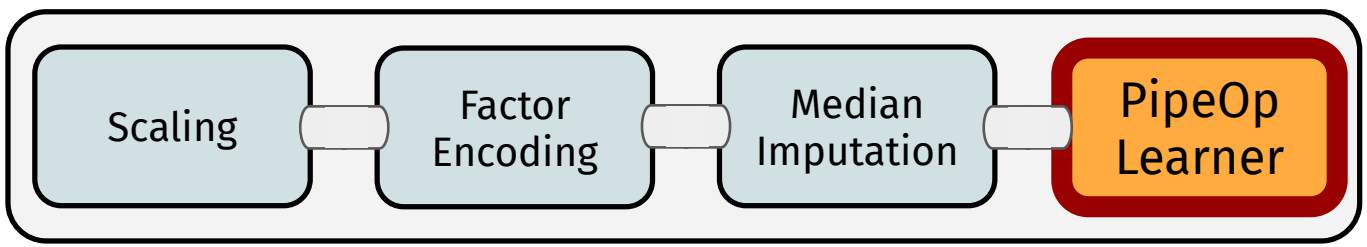
- The %>>%-operator concatenates `Graph`s and `PipeOp`s
- The `gunion()`-function unites `Graph`s and `PipeOp`s
- The `greplicate()`-function unites copies of `Graph`s and `PipeOp`s



Learners and Graphs

PipeOpLearner

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



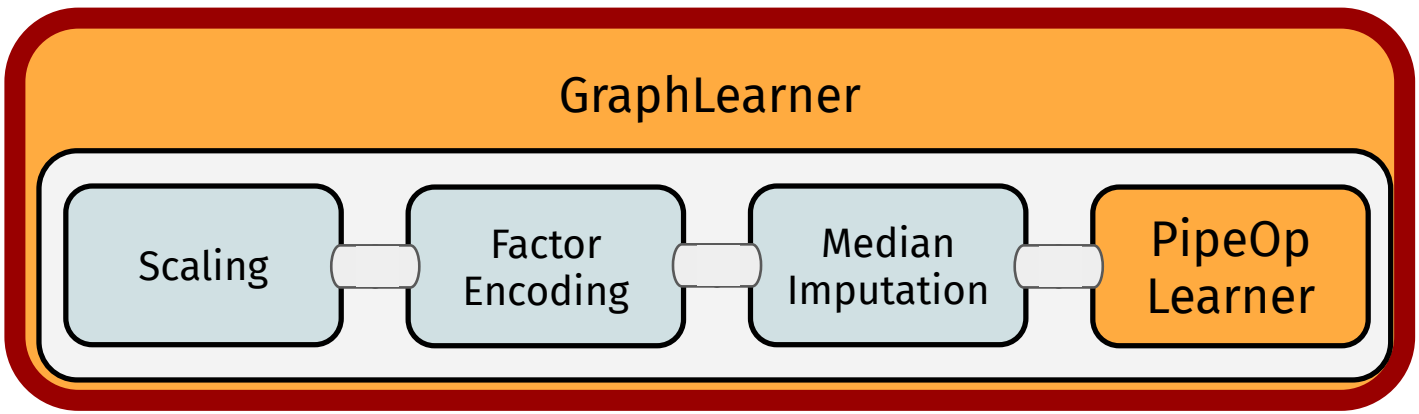
Learners and Graphs

PipeOpLearner

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

GraphLearner

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

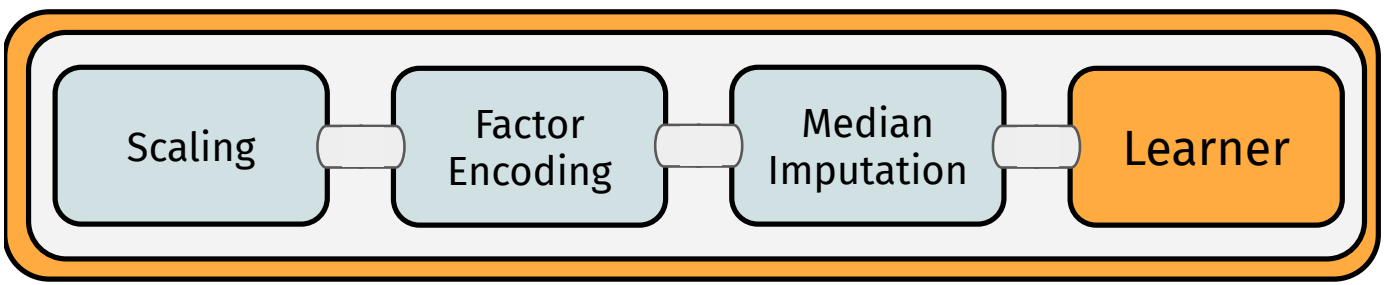


Linear Pipelines

mlr3pipelines in Action

Linear Preprocessing Pipeline

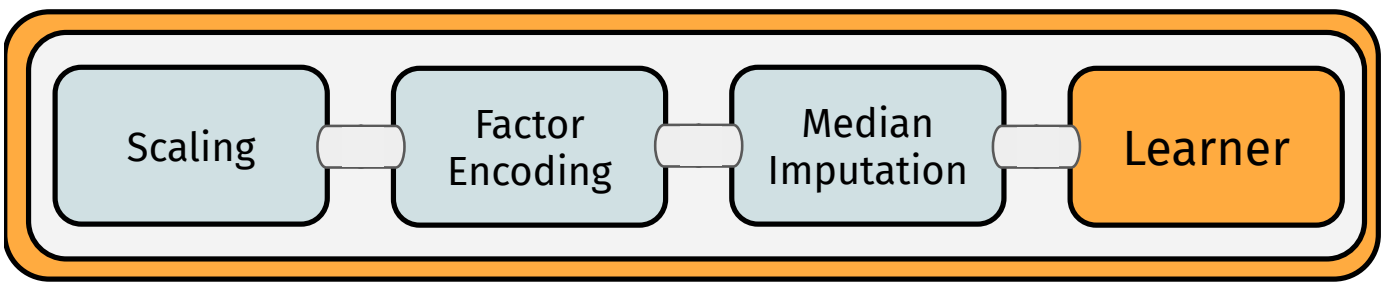
```
graph_pp = PipeOpScale$new() %>>%
  PipeOpEncode$new() %>>%
  PipeOpImputeMedian$new() %>>%
  PipeOpLearner$new(
    learner = lrn("classif.rpart"))
```



mlr3pipelines in Action

Linear Preprocessing Pipeline

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

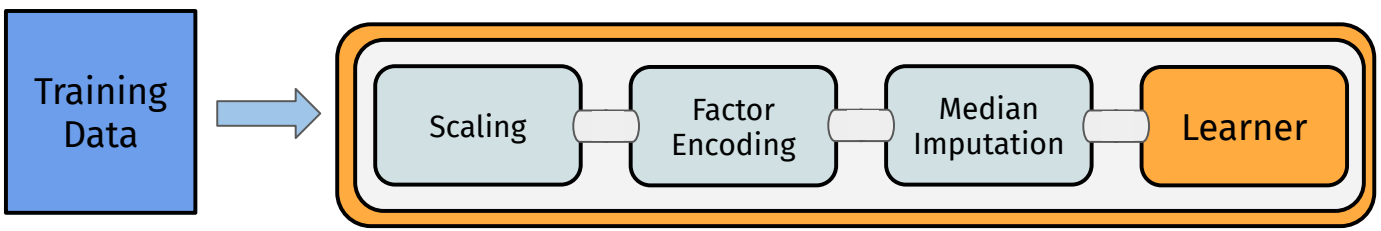


mlr3pipelines in Action

Linear Preprocessing Pipeline

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

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

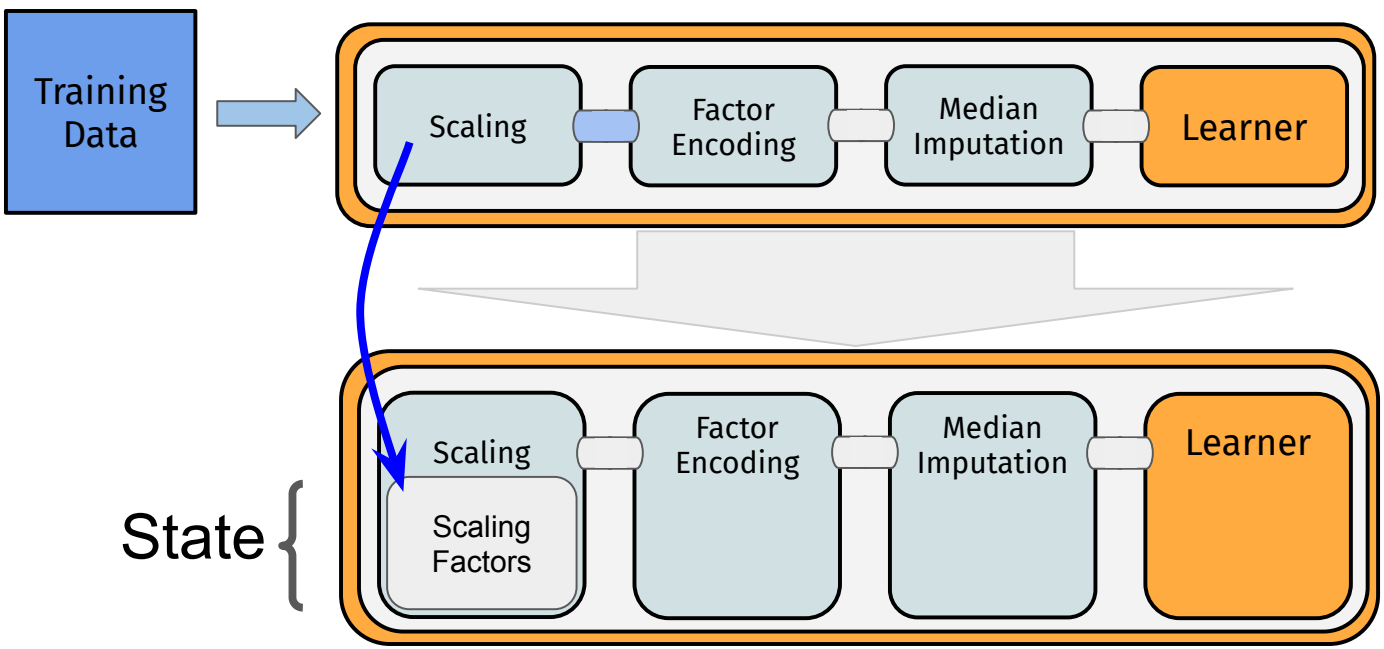


mlr3pipelines in Action

Linear Preprocessing Pipeline

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

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

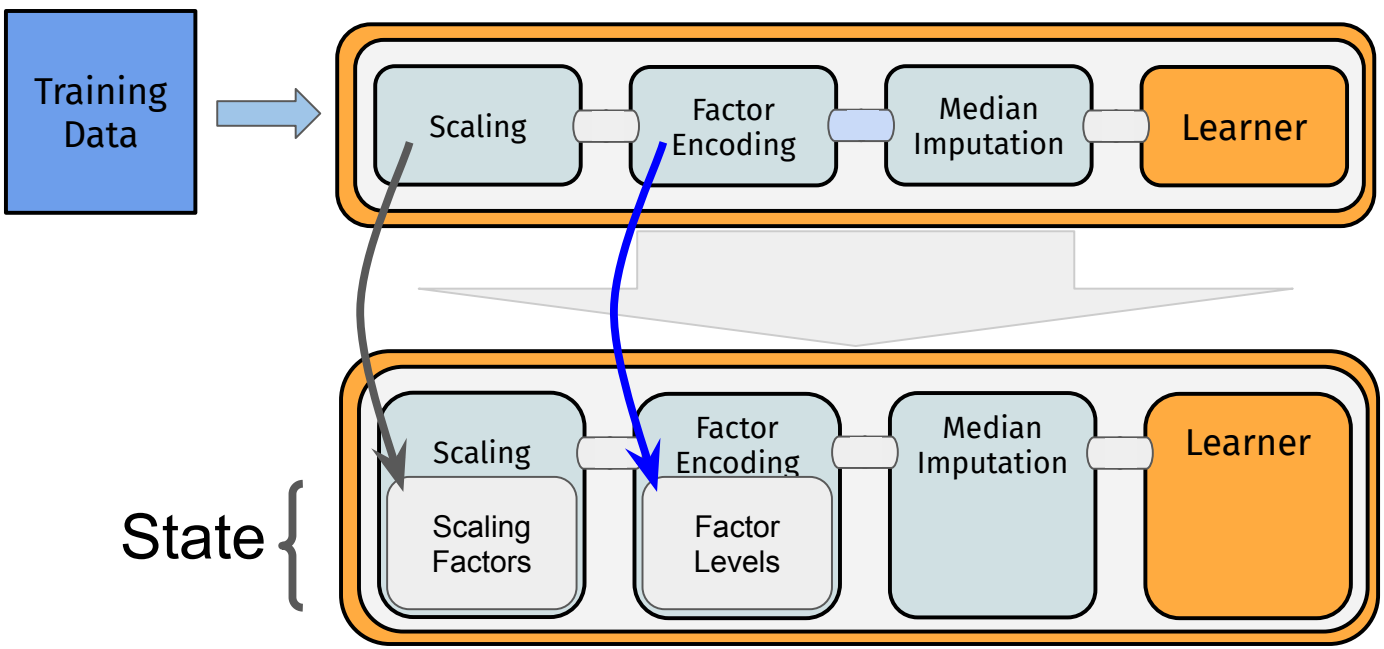


mlr3pipelines in Action

Linear Preprocessing Pipeline

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

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

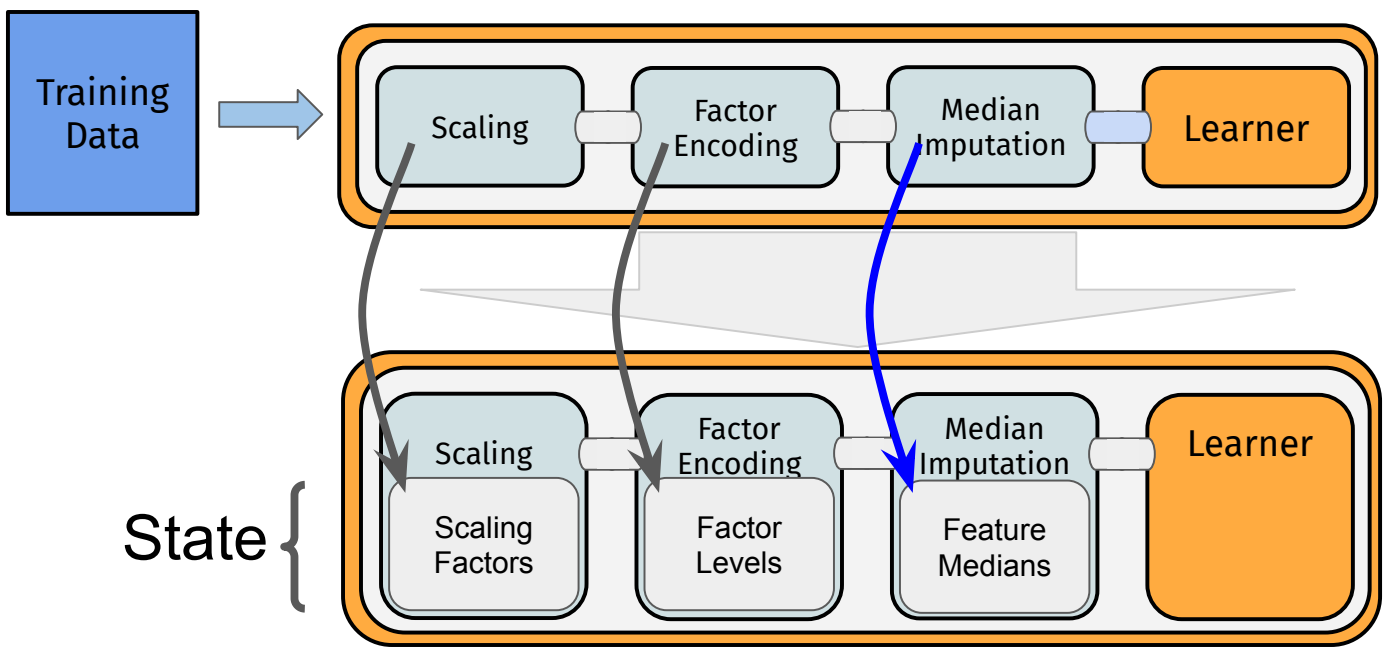


mlr3pipelines in Action

Linear Preprocessing Pipeline

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

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

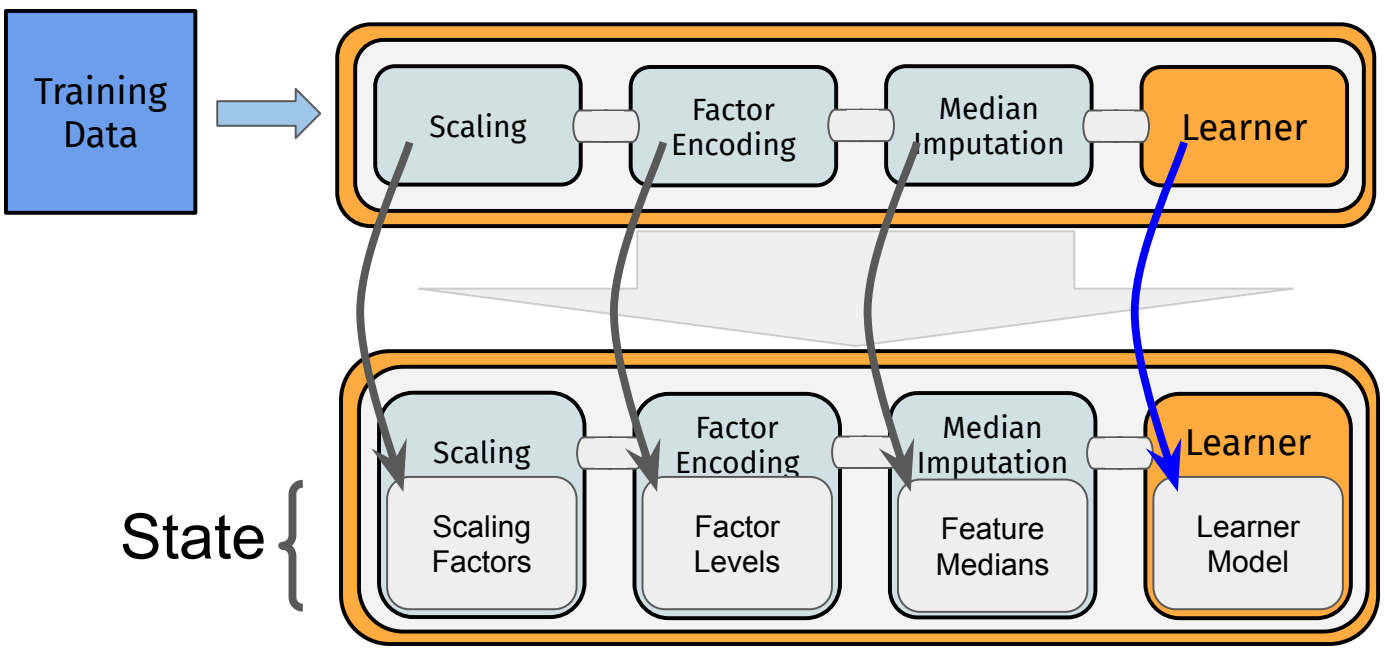


mlr3pipelines in Action

Linear Preprocessing Pipeline

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

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

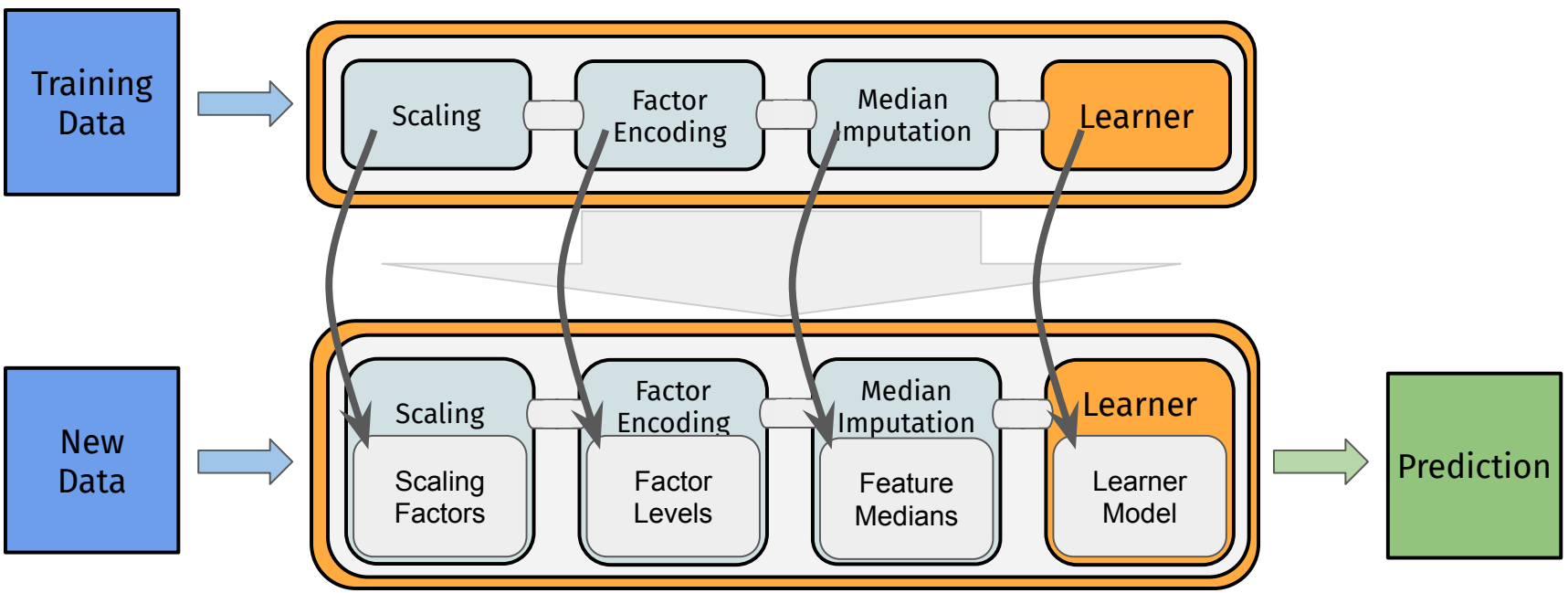


mlr3pipelines in Action

Linear Preprocessing Pipeline

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

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



mlr3pipelines in Action

Linear Preprocessing Pipeline `scale %>>% encode %>>% impute %>>% rpart`

- Setting / retrieving parameters: `$param_set`

```
graph_pp$pipeops$scale$param_set$values$center = FALSE
```

mlr3pipelines in Action

Linear Preprocessing Pipeline `scale %>>% encode %>>% impute %>>% rpart`

- Setting / retrieving parameters: `$param_set`

```
graph_pp$pipeops$scale$param_set$values$center = FALSE
```

- Retrieving state: `$state` of individual `PipeOp`s (*after* `$train()`)

```
graph_pp$pipeops$scale$state$scale
#> Petal.Length  Petal.Width Sepal.Length  Sepal.Width
#>          4.2          1.4          5.9          3.1
```

mlr3pipelines in Action

Linear Preprocessing Pipeline `scale %>>% encode %>>% impute %>>% rpart`

- Setting / retrieving parameters: `$param_set`

```
graph_pp$pipeops$scale$param_set$values$center = FALSE
```

- Retrieving state: `$state` of individual `PipeOp`s (*after* `$train()`)

```
graph_pp$pipeops$scale$state$scale
#> Petal.Length  Petal.Width Sepal.Length  Sepal.Width
#>          4.2          1.4          5.9          3.1
```

- Retrieving intermediate results: `$.result` (set debug option before)

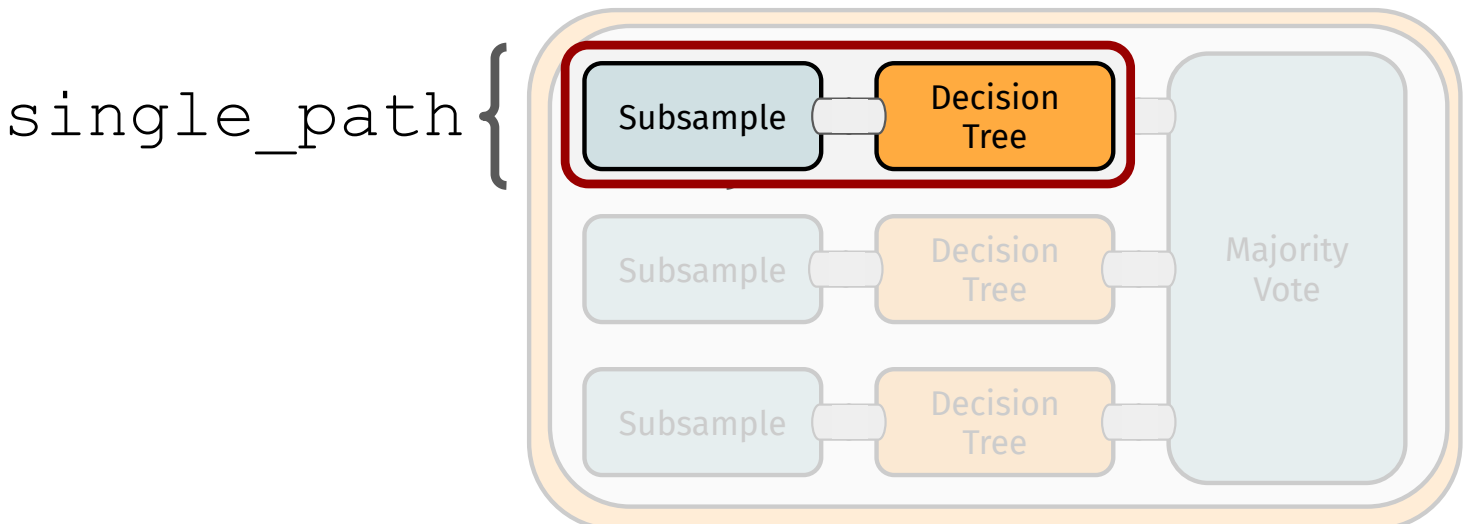
```
graph_pp$pipeops$scale$.result[[1]]$head(3)
#>    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#>     <fctr>        <num>       <num>        <num>       <num>
#> 1:  setosa         0.34        0.14         0.86        1.13
#> 2:  setosa         0.34        0.14         0.83        0.97
#> 3:  setosa         0.31        0.14         0.79        1.03
```

Nonlinear Pipelines

mlr3pipelines in Action

Ensemble Method: Bagging

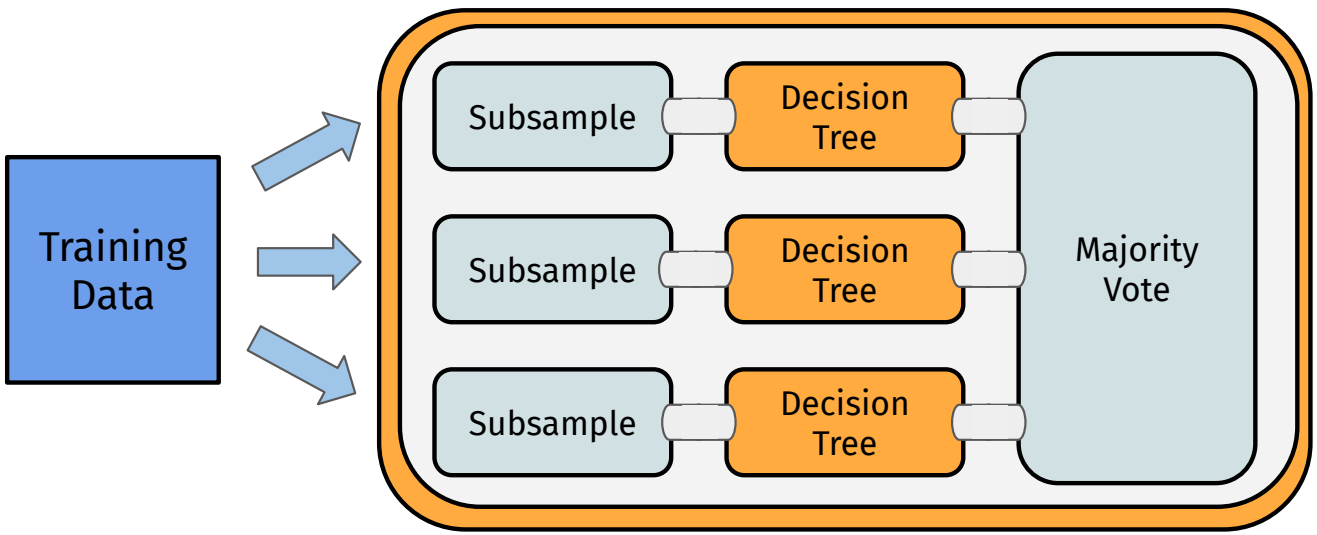
```
single_path = po("subsample") %>>% lrn("classif.rpart")
```



mlr3pipelines in Action

Ensemble Method: Bagging

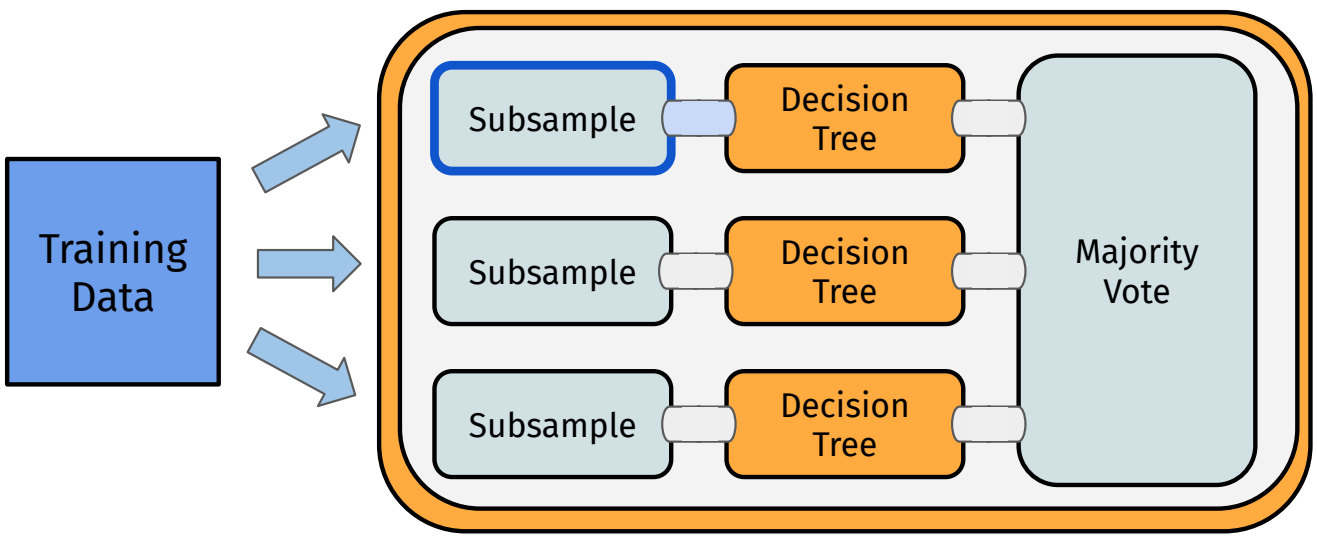
```
single_path = po("subsample") %>>% lrn("classif.rpart")

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



mlr3pipelines in Action

Ensemble Method: Bagging

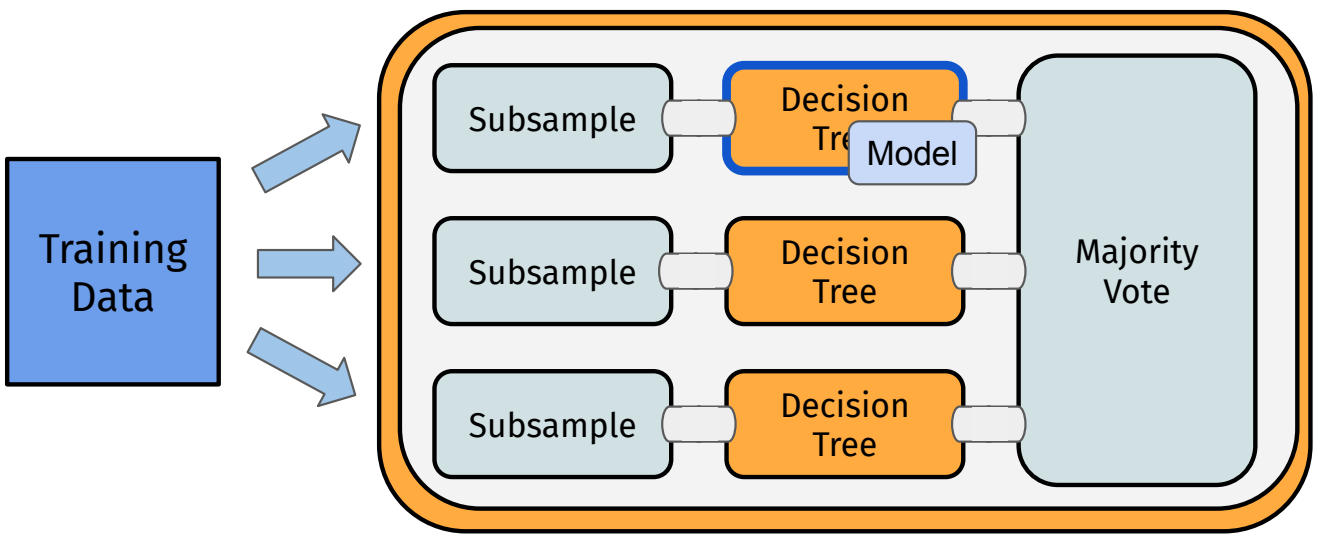
```
single_path = po("subsample") %>>% lrn("classif.rpart")

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



mlr3pipelines in Action

Ensemble Method: Bagging

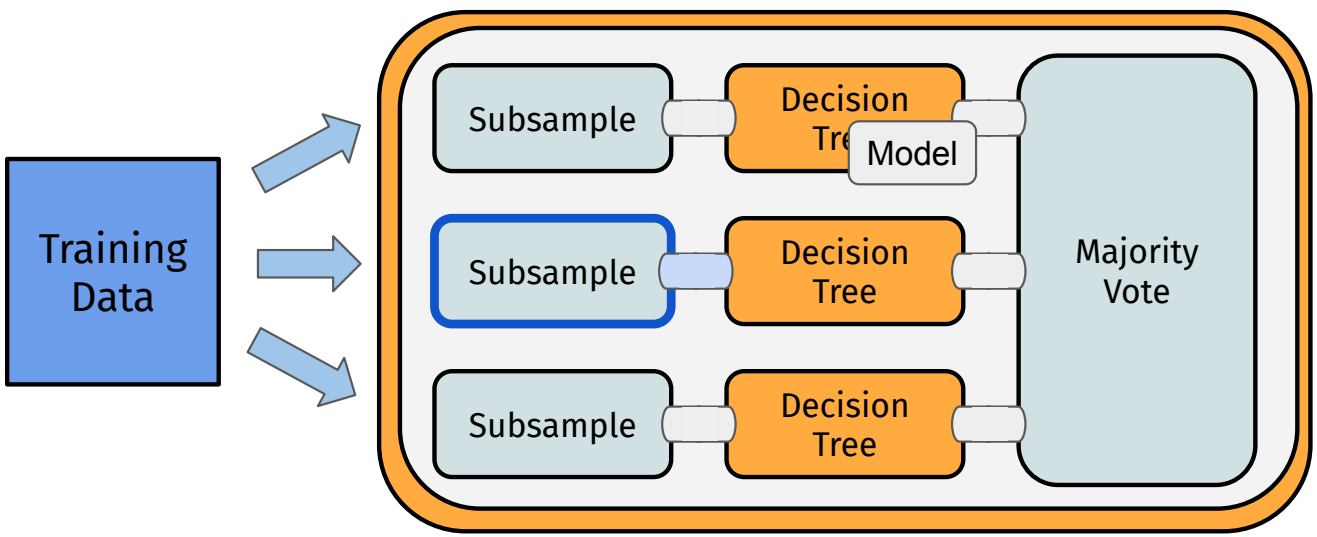
```
single_path = po("subsample") %>>% lrn("classif.rpart")

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



mlr3pipelines in Action

Ensemble Method: Bagging

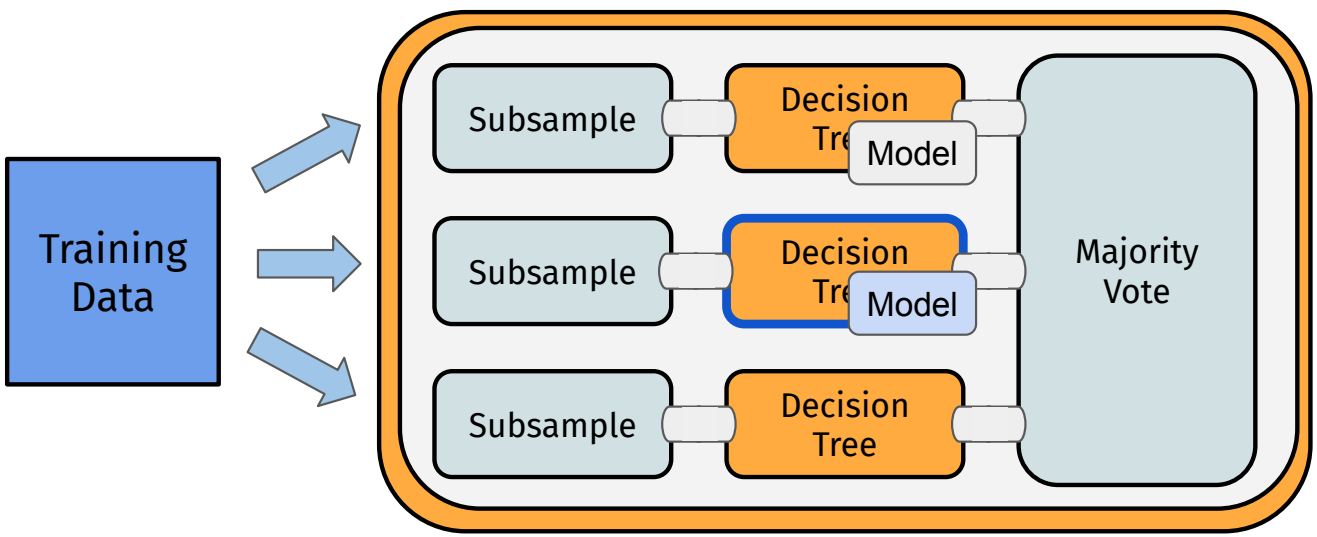
```
single_path = po("subsample") %>>% lrn("classif.rpart")

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



mlr3pipelines in Action

Ensemble Method: Bagging

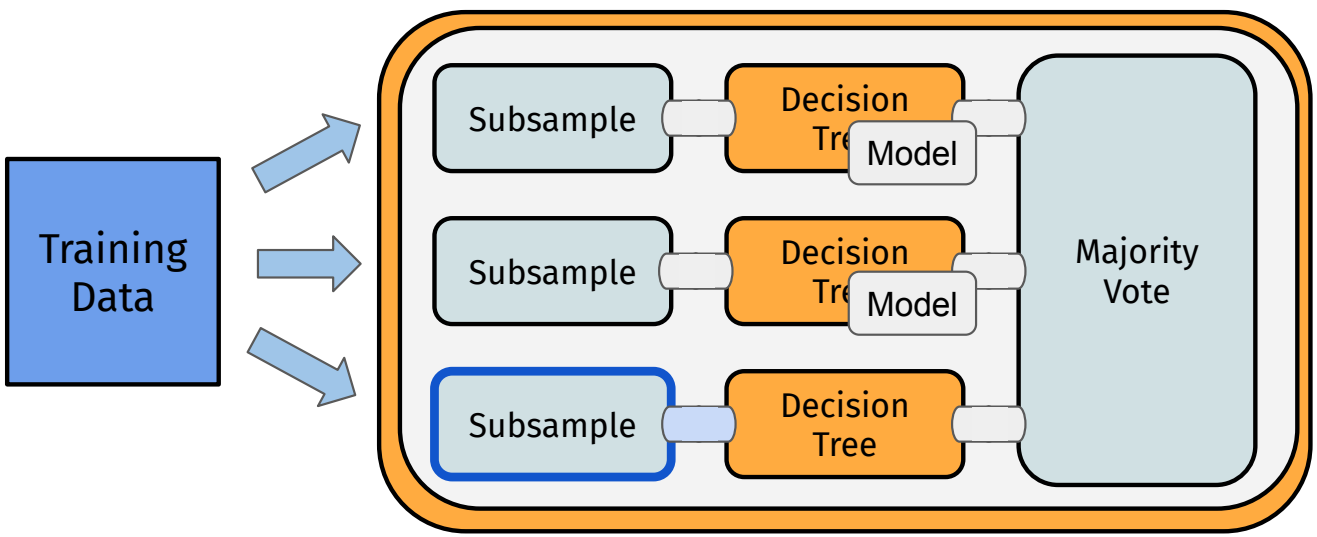
```
single_path = po("subsample") %>>% lrn("classif.rpart")

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



mlr3pipelines in Action

Ensemble Method: Bagging

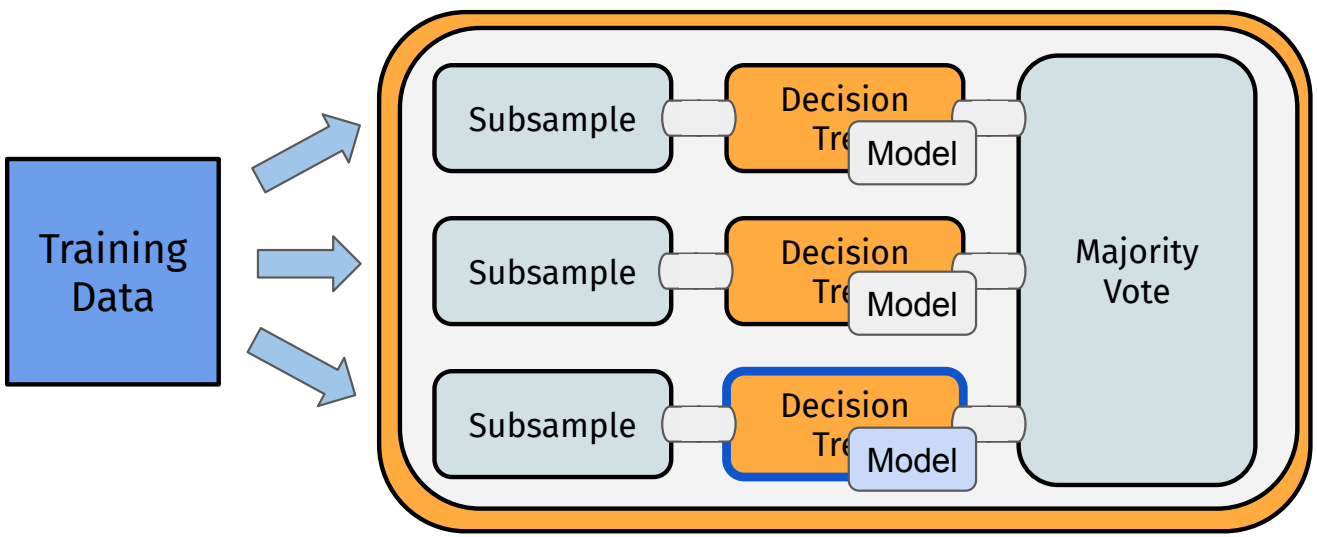
```
single_path = po("subsample") %>>% lrn("classif.rpart")

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



mlr3pipelines in Action

Ensemble Method: Bagging

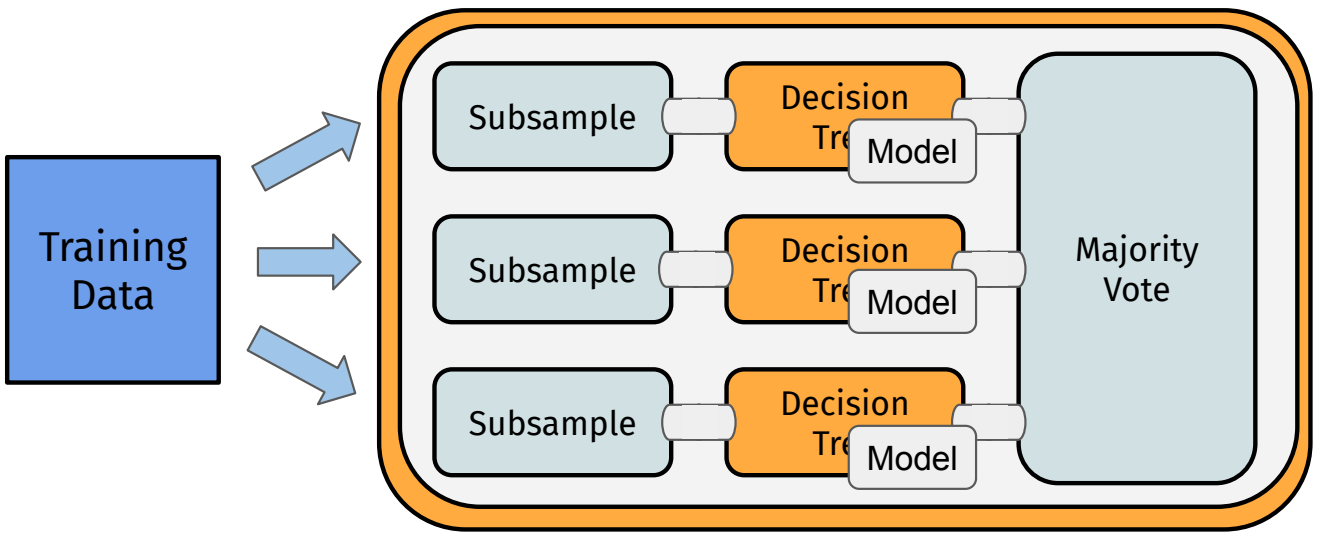
```
single_path = po("subsample") %>>% lrn("classif.rpart")

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



mlr3pipelines in Action

Ensemble Method: Bagging

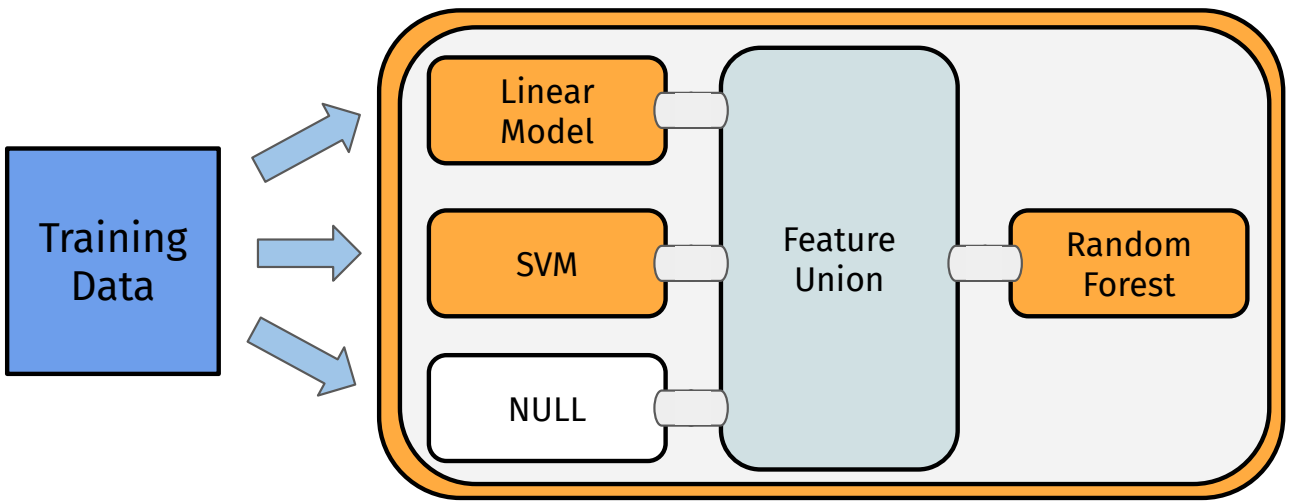
```
single_path = po("subsample") %>>% lrn("classif.rpart")

graph_bag = 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")
```

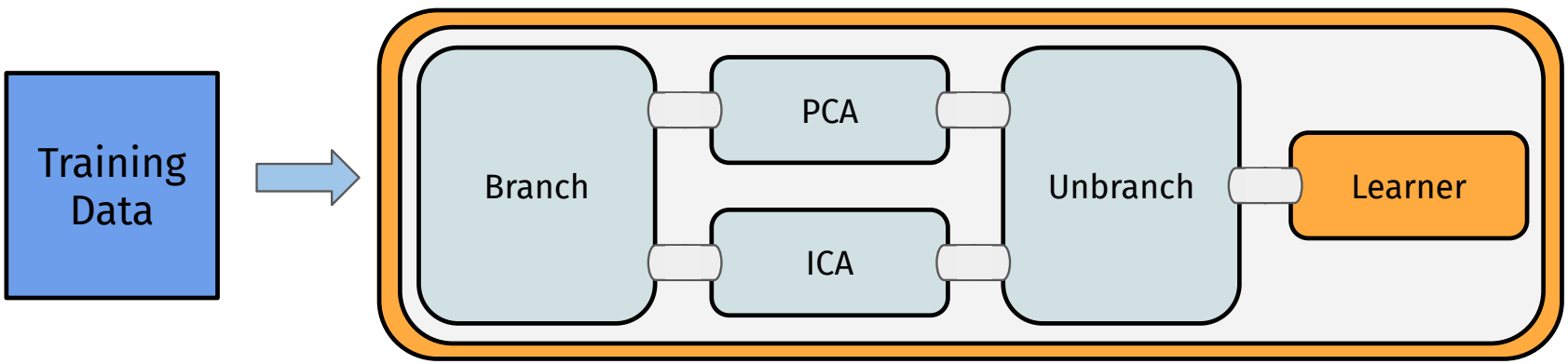


mlr3pipelines in Action

Branching

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

Execute only one of several alternative paths

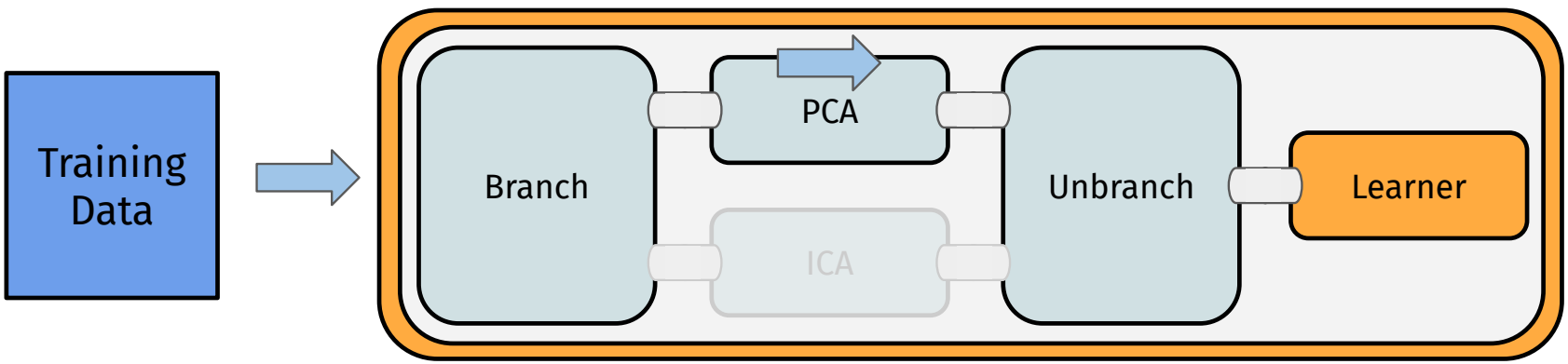


mlr3pipelines in Action

Branching

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

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

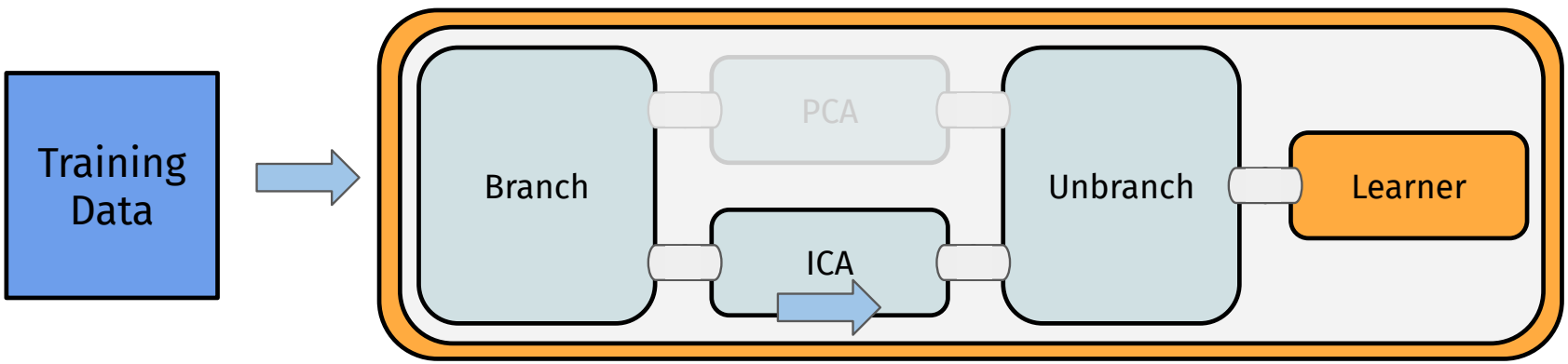


mlr3pipelines in Action

Branching

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

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



Hyperparameters and Tuning

Hyperparameters and Tuning

- `PipeOps` have *hyperparameters* (using `paradox` pkg)
- `Graph`s have hyperparameters of all components *combined*
- $\Rightarrow$ simultaneous **Tuning** of `Learner` and preprocessing (`mlr3tuning` package)

```
library("paradox") ; library("mlr3tuning")
glrn = po("scale") %>>% lrn("classif.rpart")
ps = ParamSet$new(list(
  ParamLgl$new("scale.scale"),
  ParamInt$new("classif.rpart.minsplit", 1, 20)
))
inst = TuningInstance$new(task, glrn,
  rsmp("cv"), msr("classif.ce"), ps,
  term("evals", n_evals = 10))

tnr("random_search")$tune(inst)

inst$result
```

mlr3pipelines recap

mlr3pipelines

`mlr3pipelines` overview:

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

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`, `...`)

Internal Data Structure

Results objects (`resample()`, `benchmark()`, ...) share the same structure

```
as.data.table(result)

#>              task                learner     resampling iteration
#>            <list>                 <list>         <list>     <int>
#> 1: <TaskClassif> <LearnerClassifRpart> <ResamplingCV>         1
#> 2: <TaskClassif> <LearnerClassifRpart> <ResamplingCV>         2
#> 3: <TaskClassif> <LearnerClassifRpart> <ResamplingCV>         3
#> 4: <TaskClassif> <LearnerClassifRpart> <ResamplingCV>         4
#> 5: <TaskClassif> <LearnerClassifRpart> <ResamplingCV>         5
#>    prediction
#>        <list>
#> 1:     <list>
#> 2:     <list>
#> 3:     <list>
#> 4:     <list>
#> 5:     <list>
```

Combining R6 and data.table

- Not the objects are stored, but pointers to them
- Inexpensive to work on:
  - `rbind()`: copying R6 objects $\leftrightarrow$ copying pointers
  - `cbind()`, `data.table()` over-allocates columns, no copies

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
- Possibilty to have a fallback to create predictions

Out-of-memory Data

- Task stores data in a `DataBackend`:
  - `DataBackendDataTable`: Default backend for dense data (in-memory)
  - `DataBackendMatrix`: Backend for sparse numerical data (in-memory)
  - `DataBackendDplyr`: Backend for many DBMS (out-of-memory)
  - `DataBackendCbind`: Combine backends in a `cbind()` fashion (virtual)
  - `DataBackendRbind`: Combine backends in a `rbind()` fashion (virtual)
- Backends are immutable
  - Filtering rows or selecting columns just modifies the "view" on the data
  - Multiple tasks can share the same backend
- Example: Interface a read-only MariaDB with `DataBackendDplyr`, add generated features via `DataBackendDataTable`

Status of mlr3

- Under active development
  - Minor API changes are possible
  - GitHub versions mostly stable
  - UX improvements (feedback welcome!)
- On CRAN
  - mlr3, mlr3db, mlr3filters, mlr3learners
- On GitHub
  - mlr3pipelines, mlr3survival, mlr3tuning, mlr3viz, mlr3spatiotemporal
- More coming mlr-org/mlr3/wiki/Extension-Packages!
- mlr3book covers actual state.

We are always open for contributions!