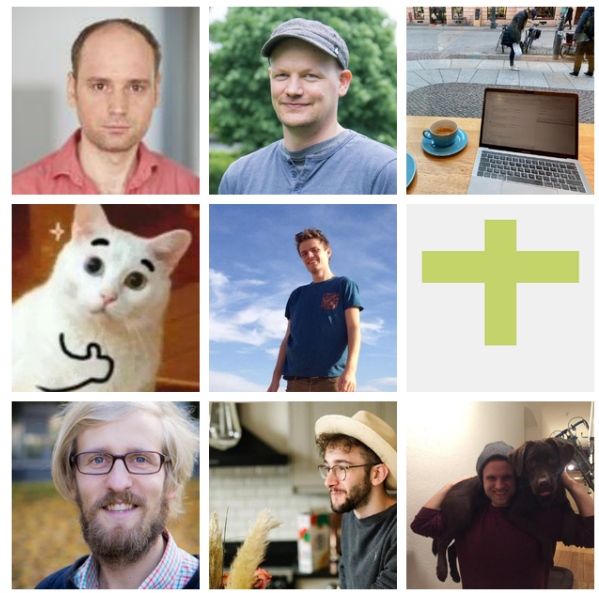
Tuning Machine Learning Algorithms with mlr3

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, Giuseppe Casalicchio

October 22, 2021

Intro

TUNING

- Behavior of most methods depends on *hyperparameters*

TUNING

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

TUNING

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

TUNING

- Behavior 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

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

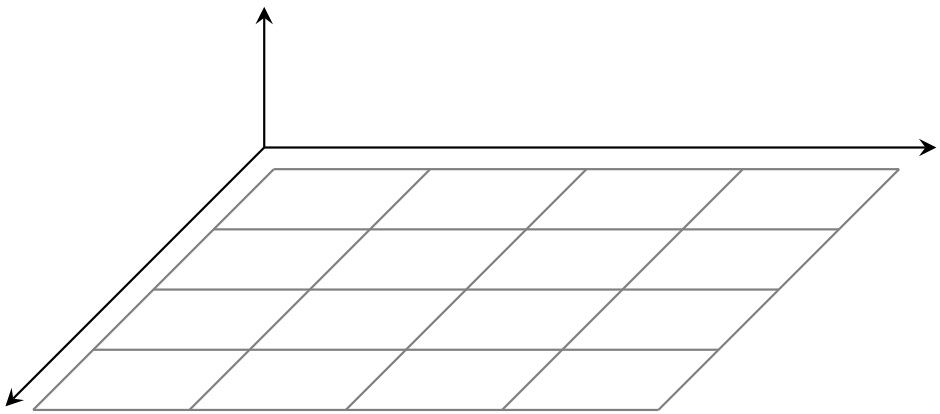
$\Rightarrow$ We do *black box optimization* (“Try stuff and see what works”)

Tuning toolbox for `mlr3`:

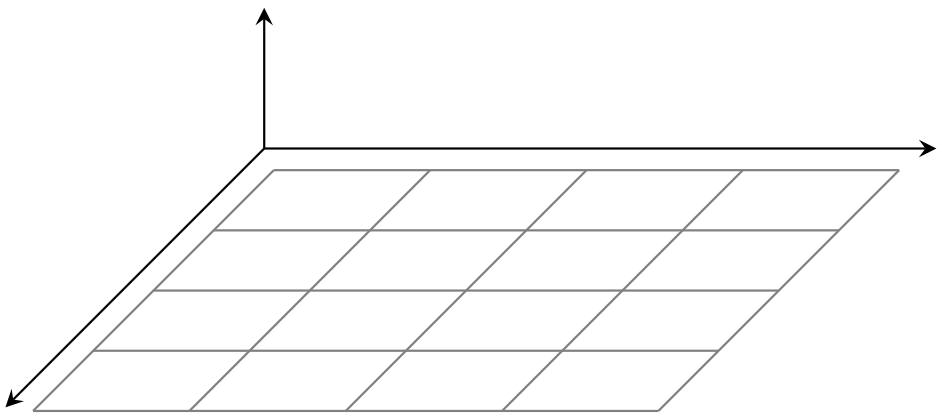
```
library("bbotk")
library("mlr3tuning")
```

Tuning

TUNING

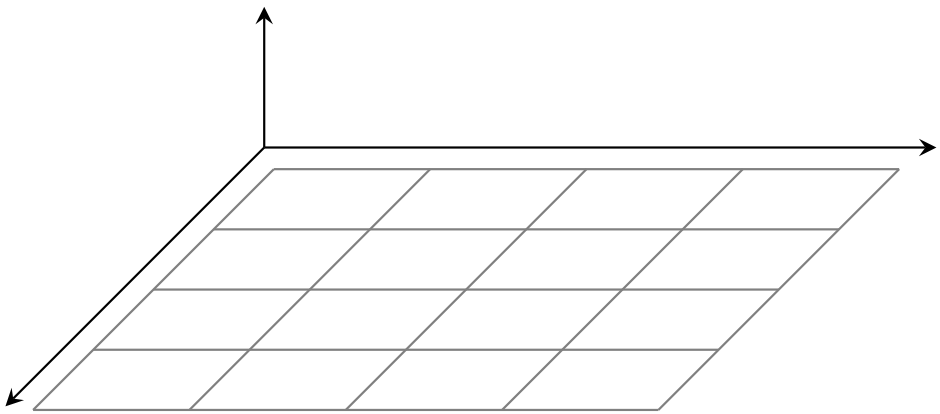


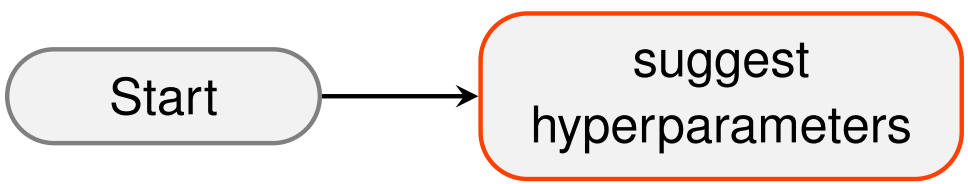
TUNING



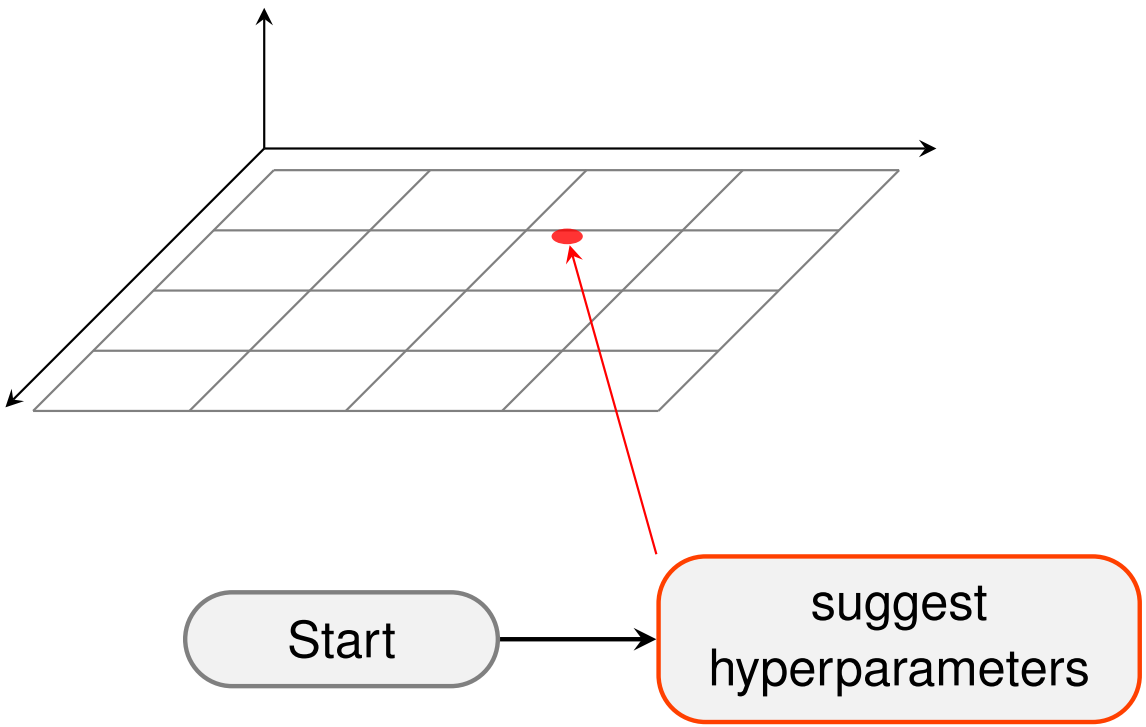
Start

TUNING

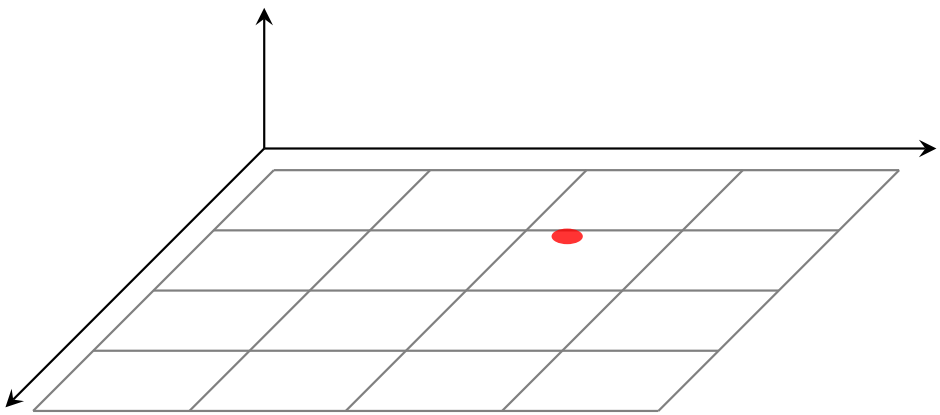


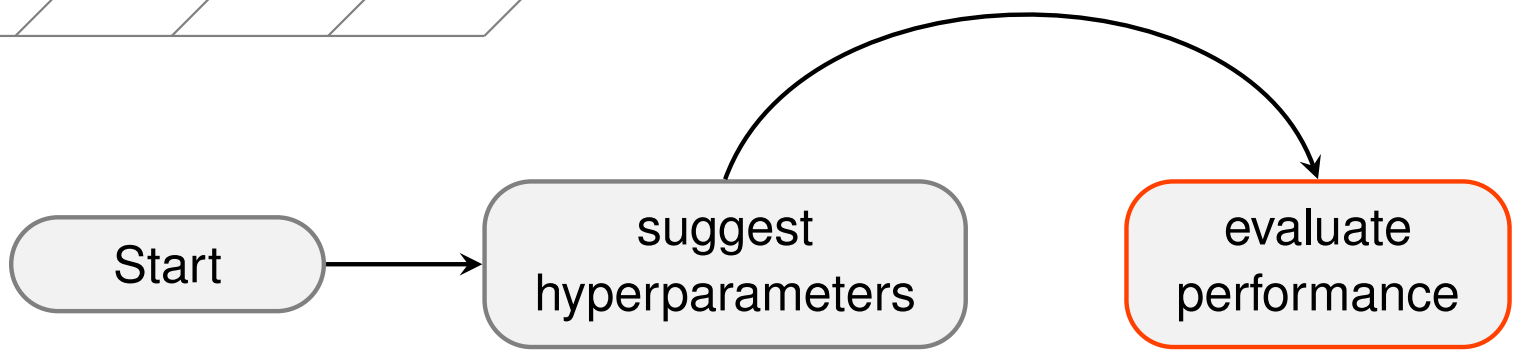


TUNING

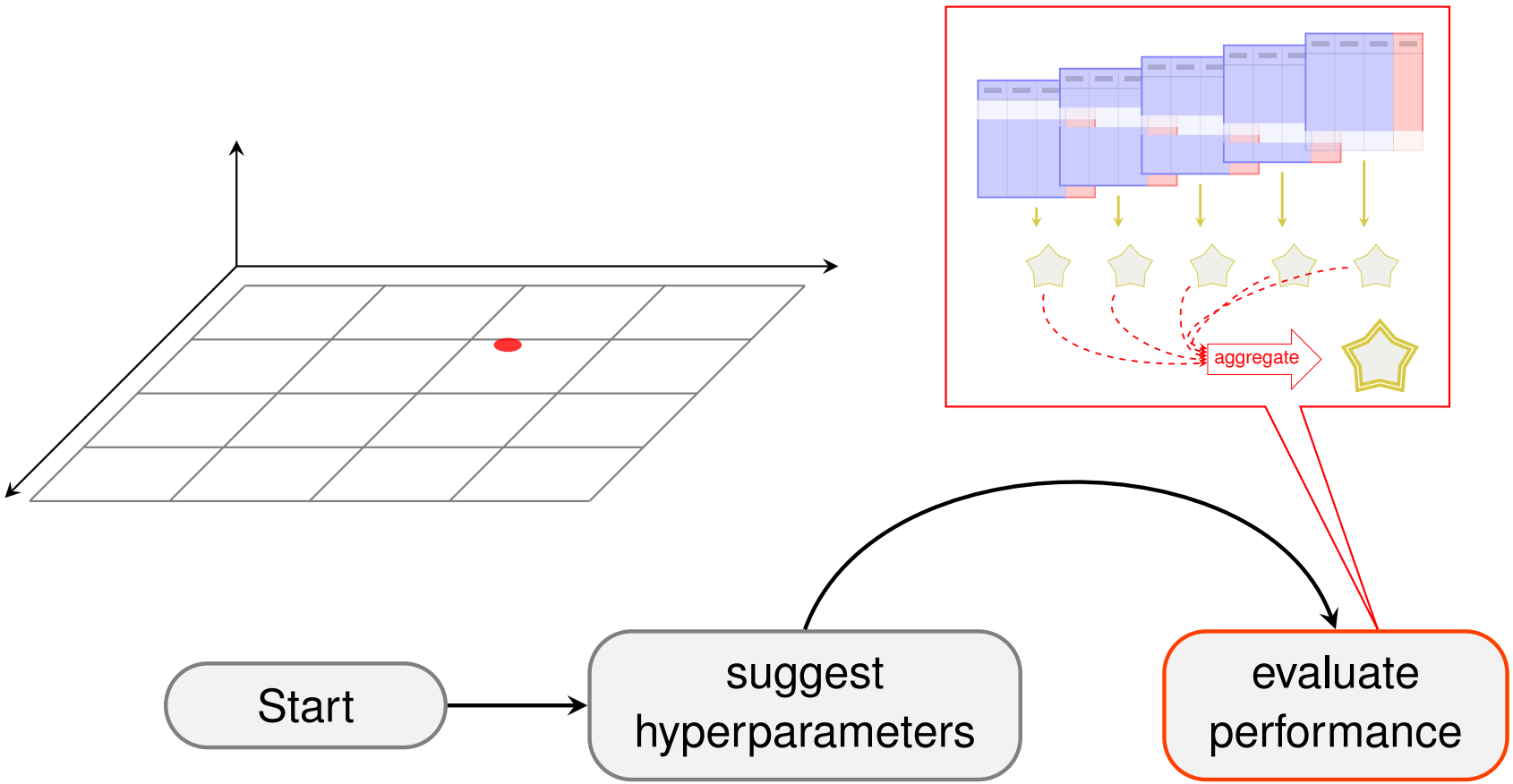


TUNING

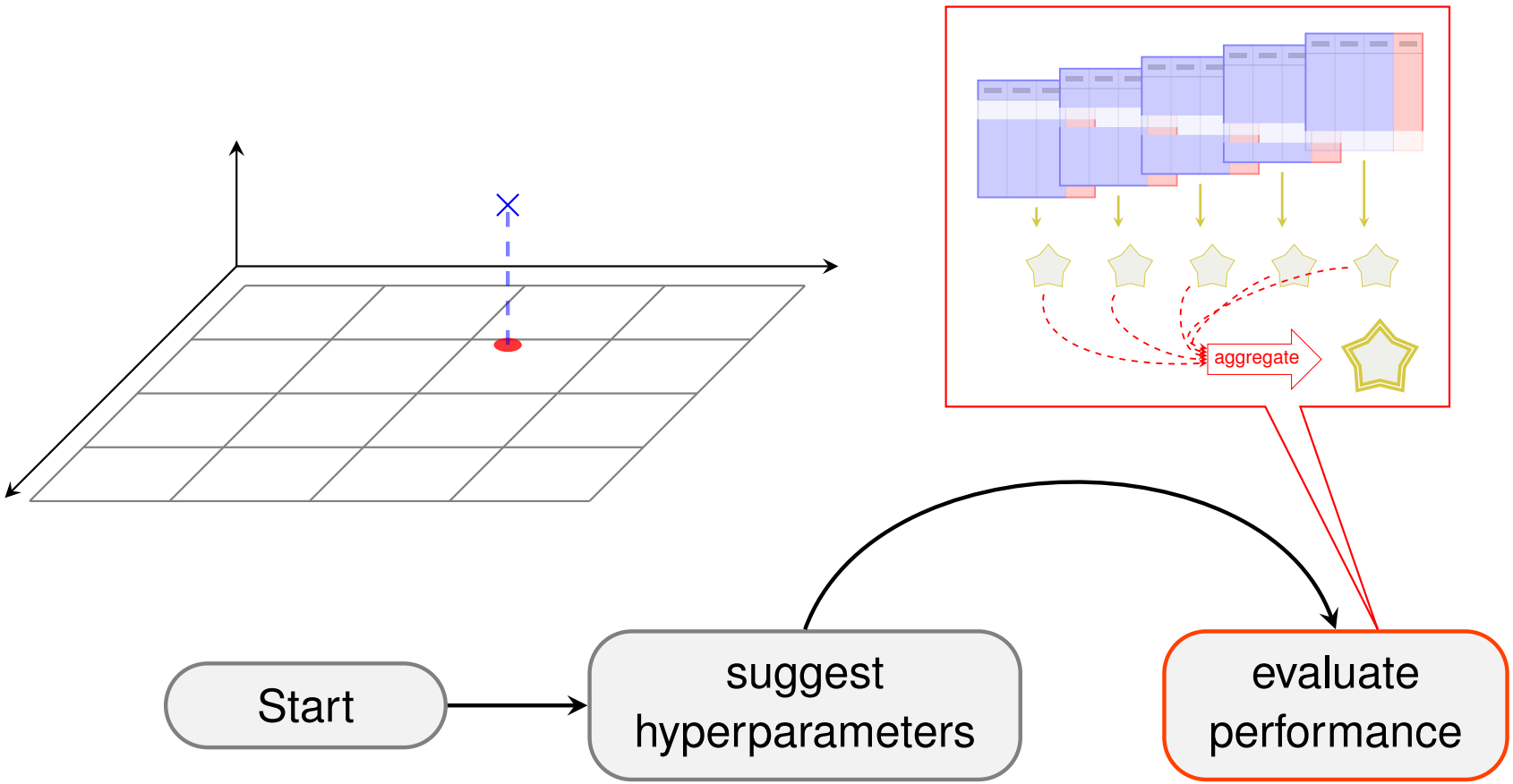




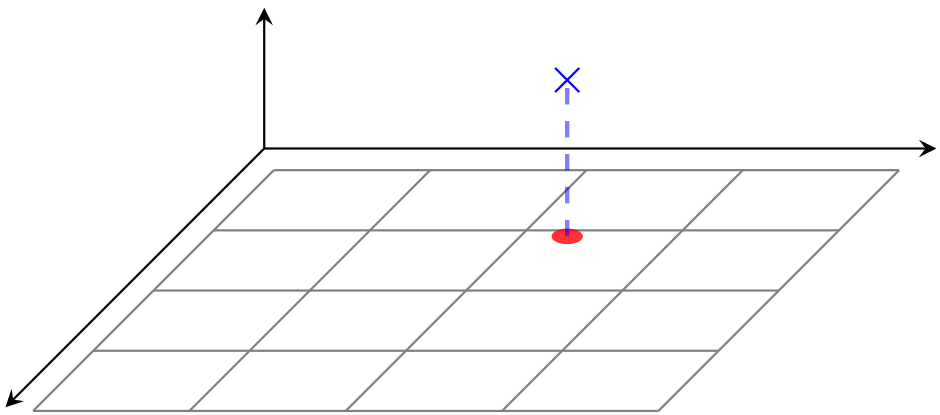
TUNING

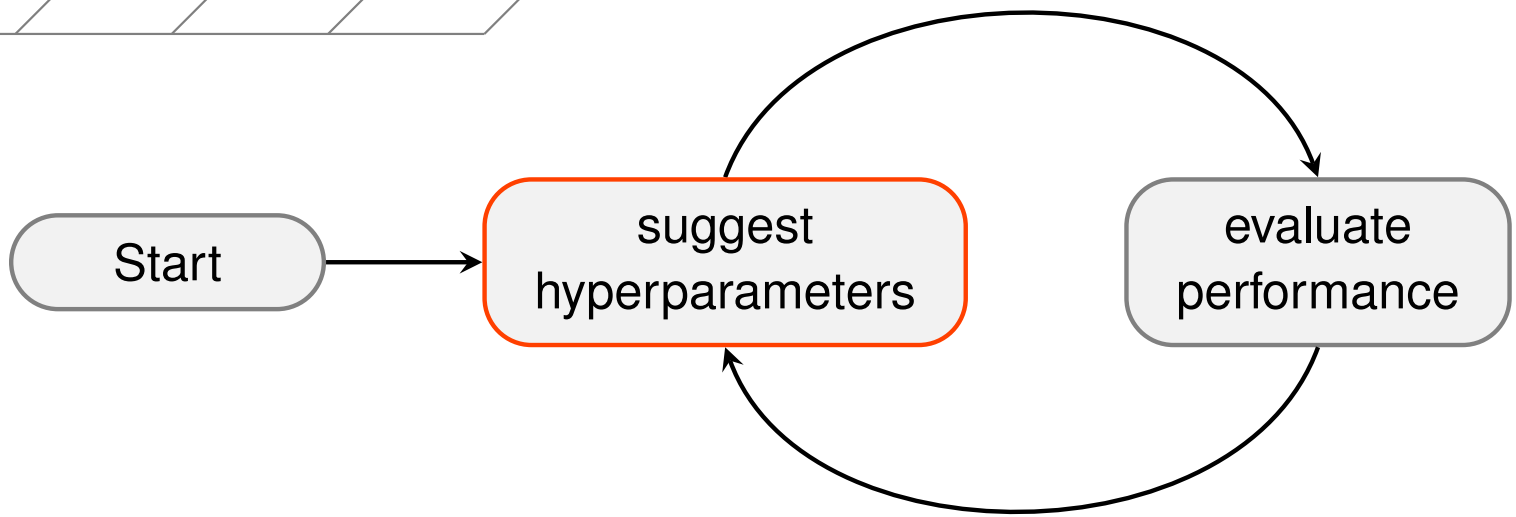

aggregate
Start
suggest hyperparameters
evaluate performance

TUNING

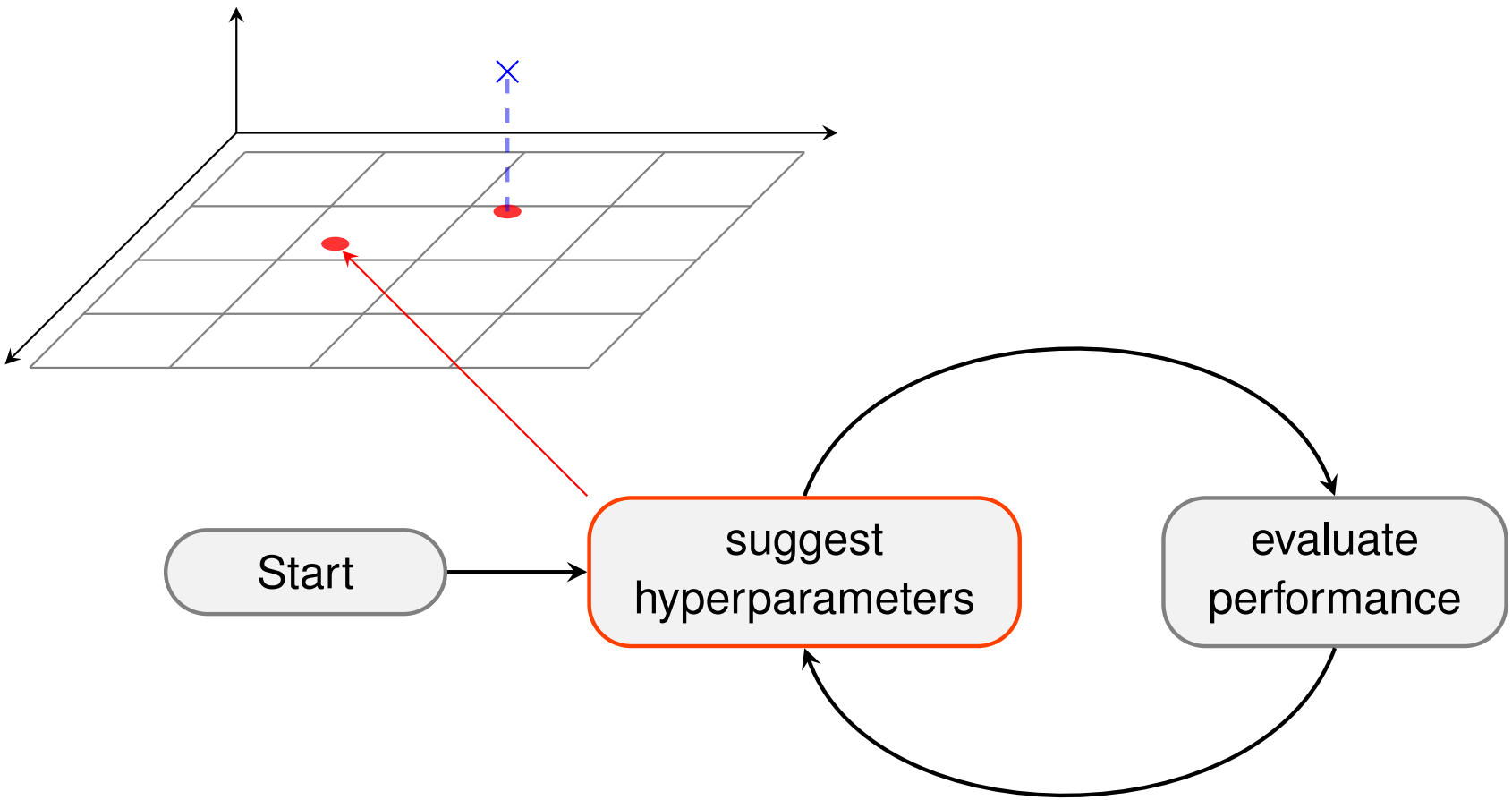

aggregate
Start
suggest hyperparameters
evaluate performance

TUNING

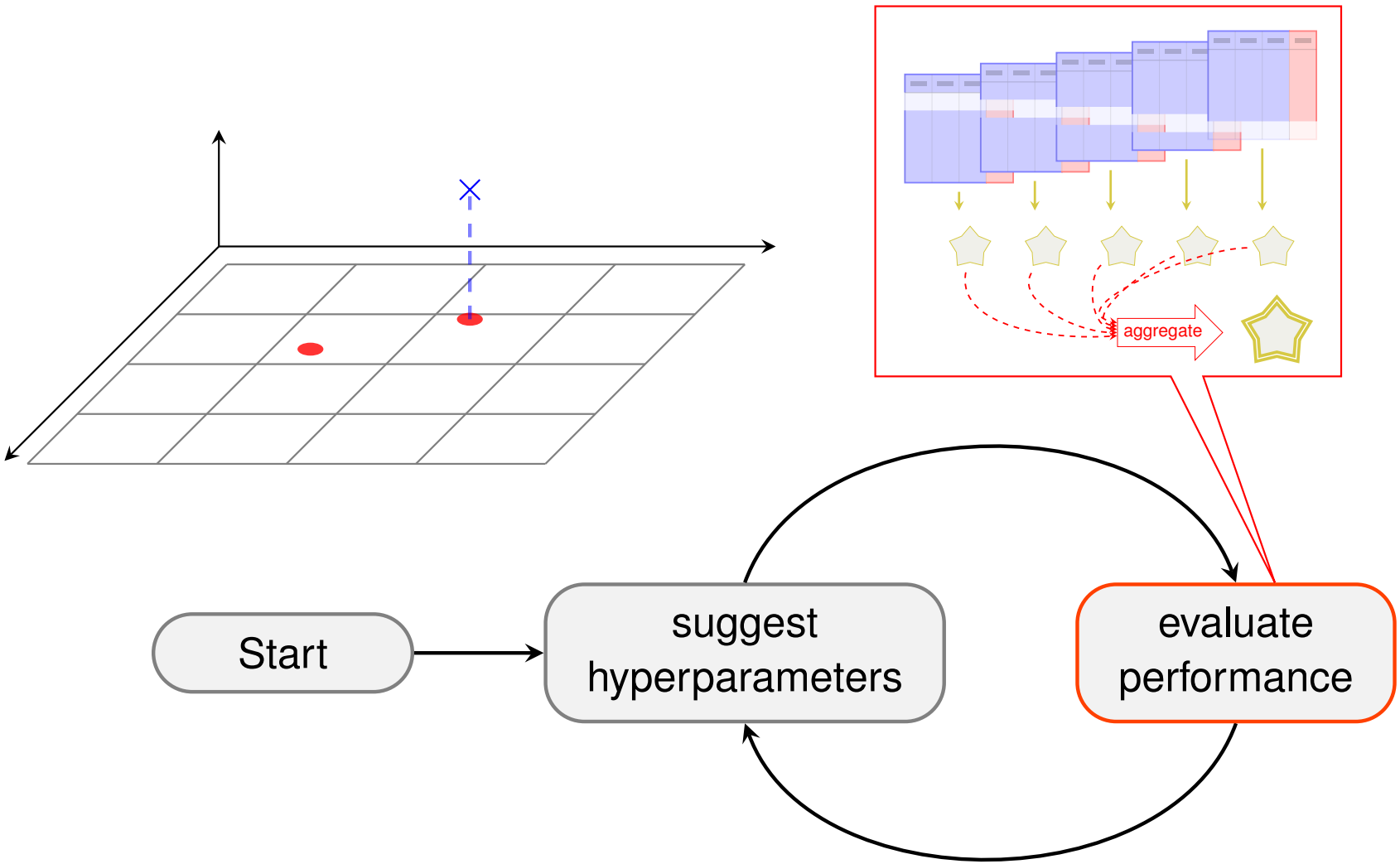




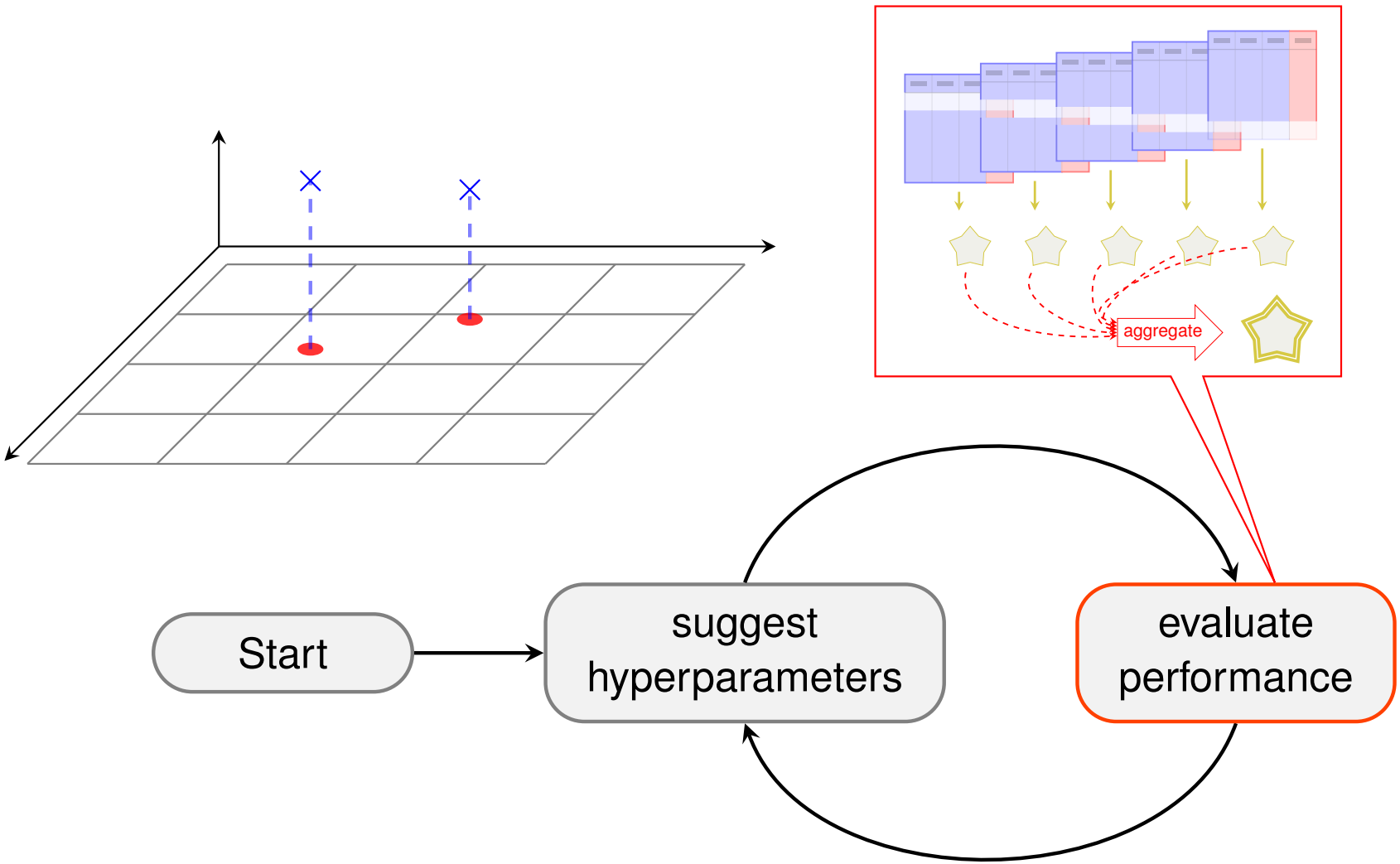
TUNING



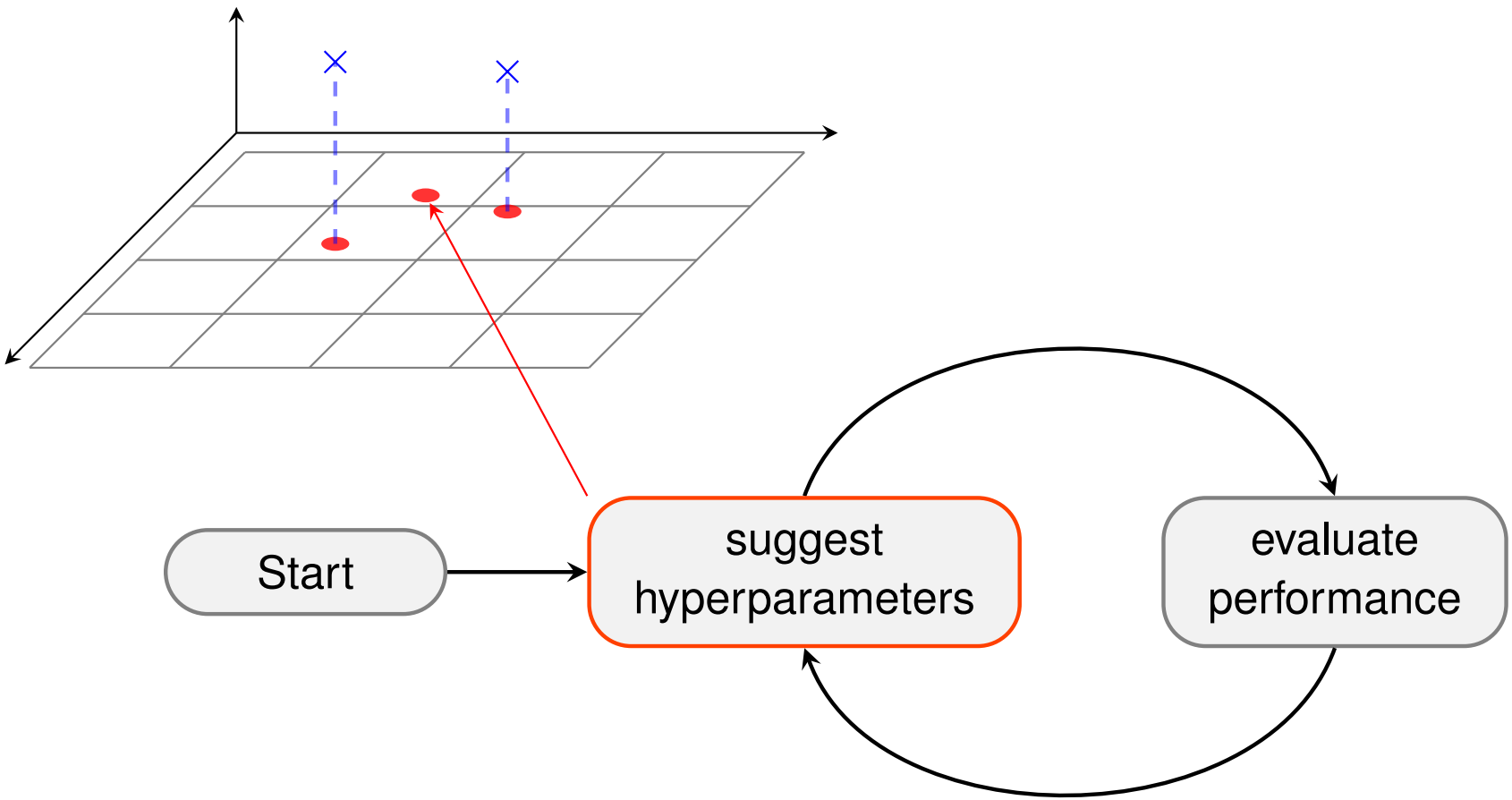
TUNING


aggregate
Start
suggest hyperparameters
evaluate performance

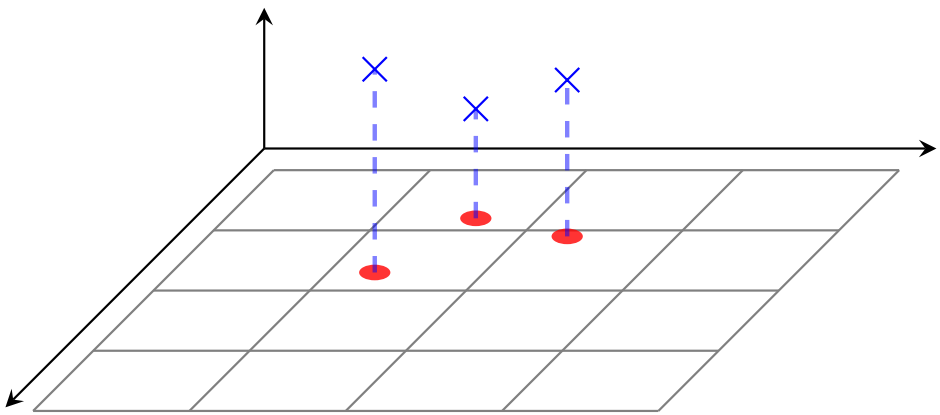
TUNING

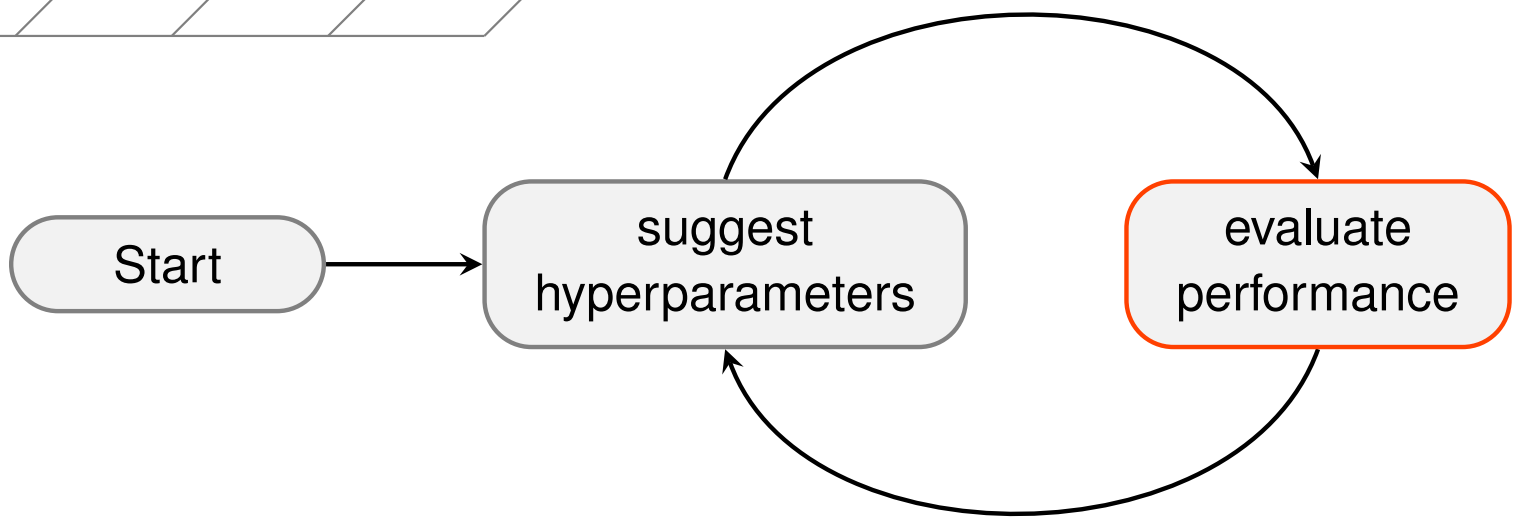

aggregate
Start
suggest
hyperparameters
evaluate
performance

TUNING

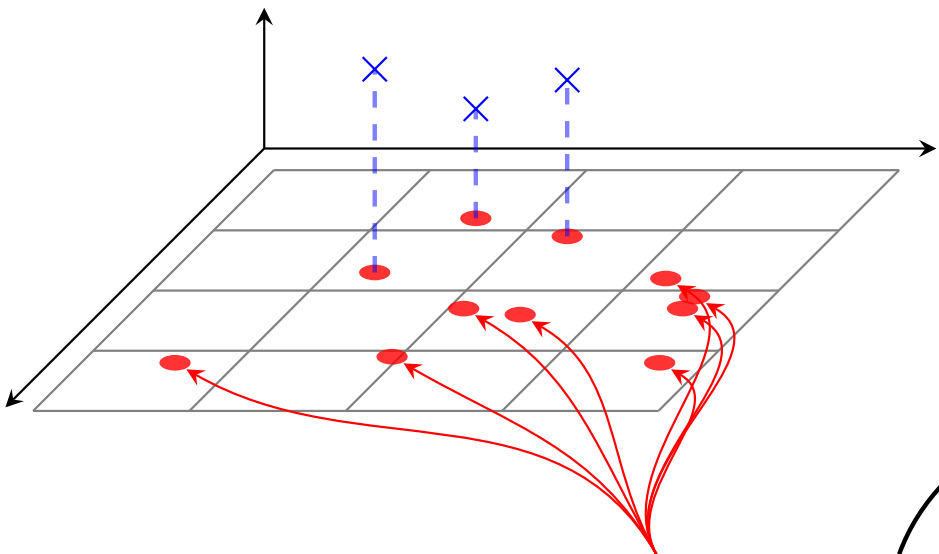


TUNING

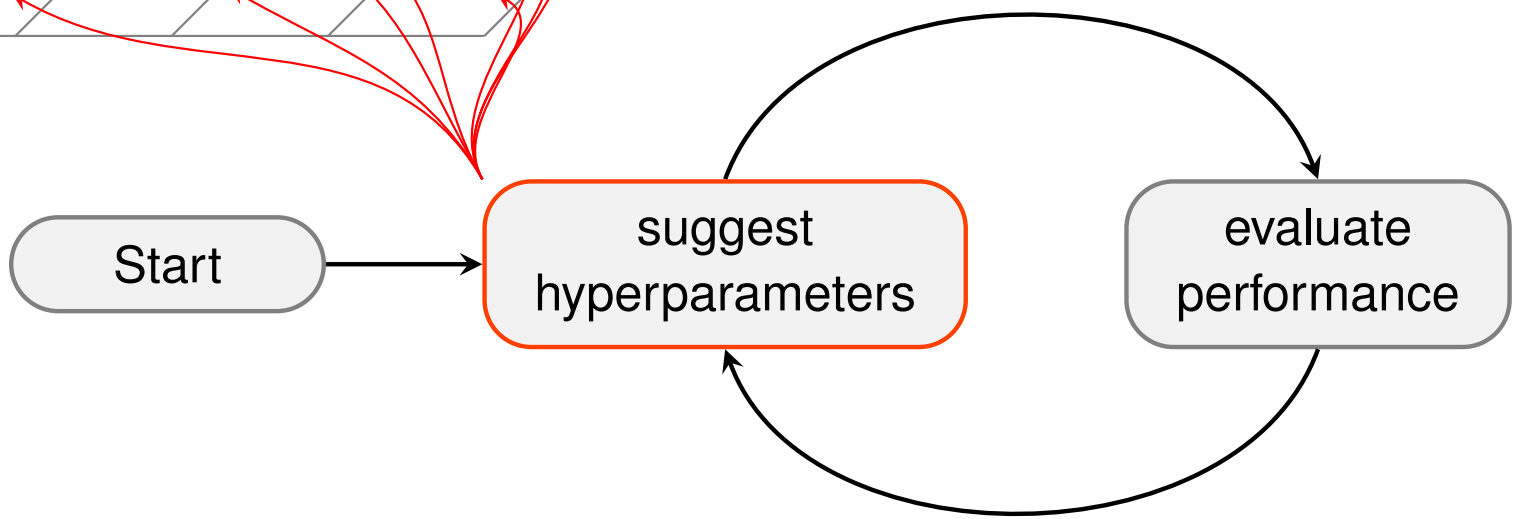




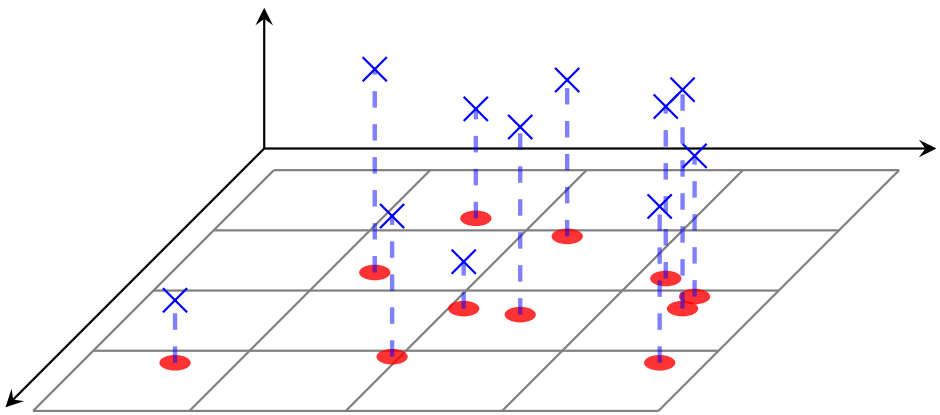
TUNING



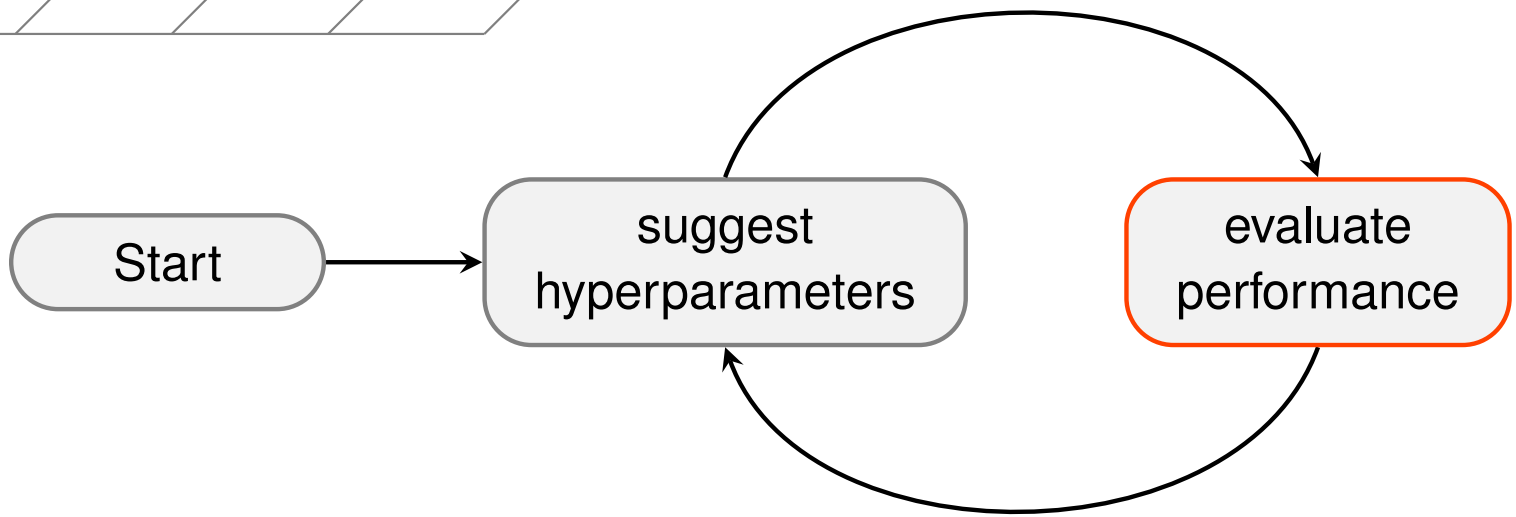
Batch evaluation



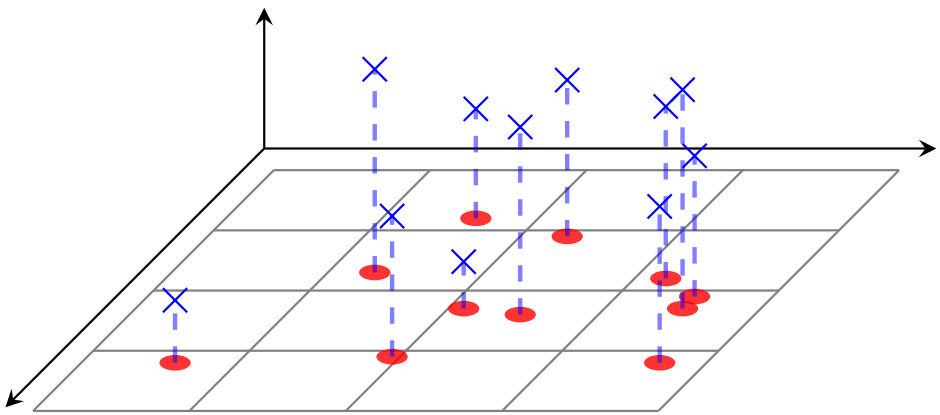
TUNING

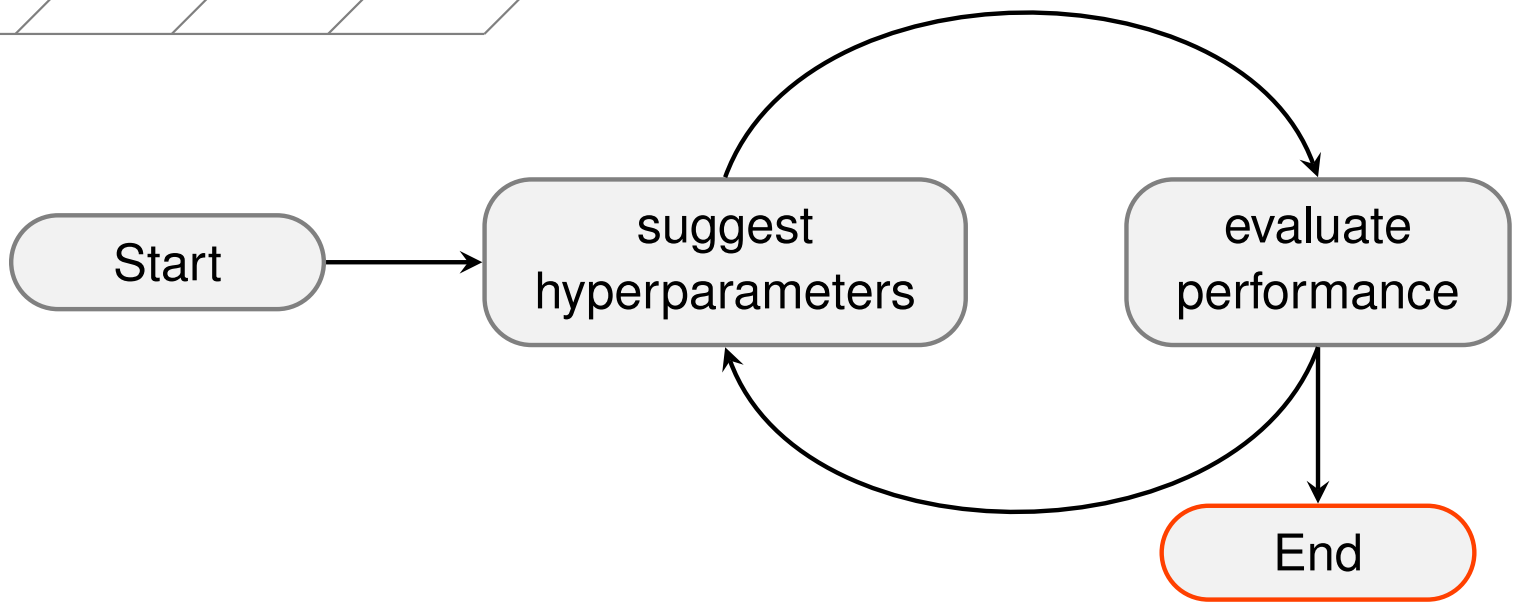


Batch evaluation

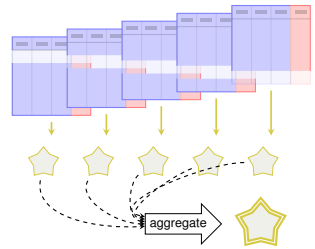


TUNING

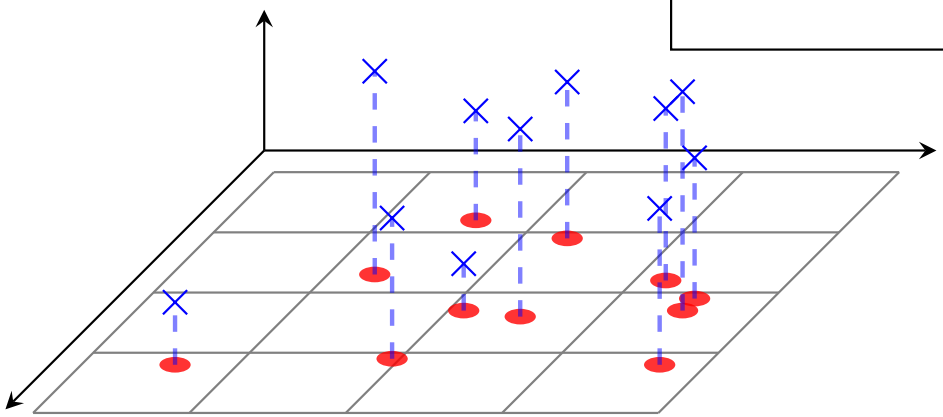


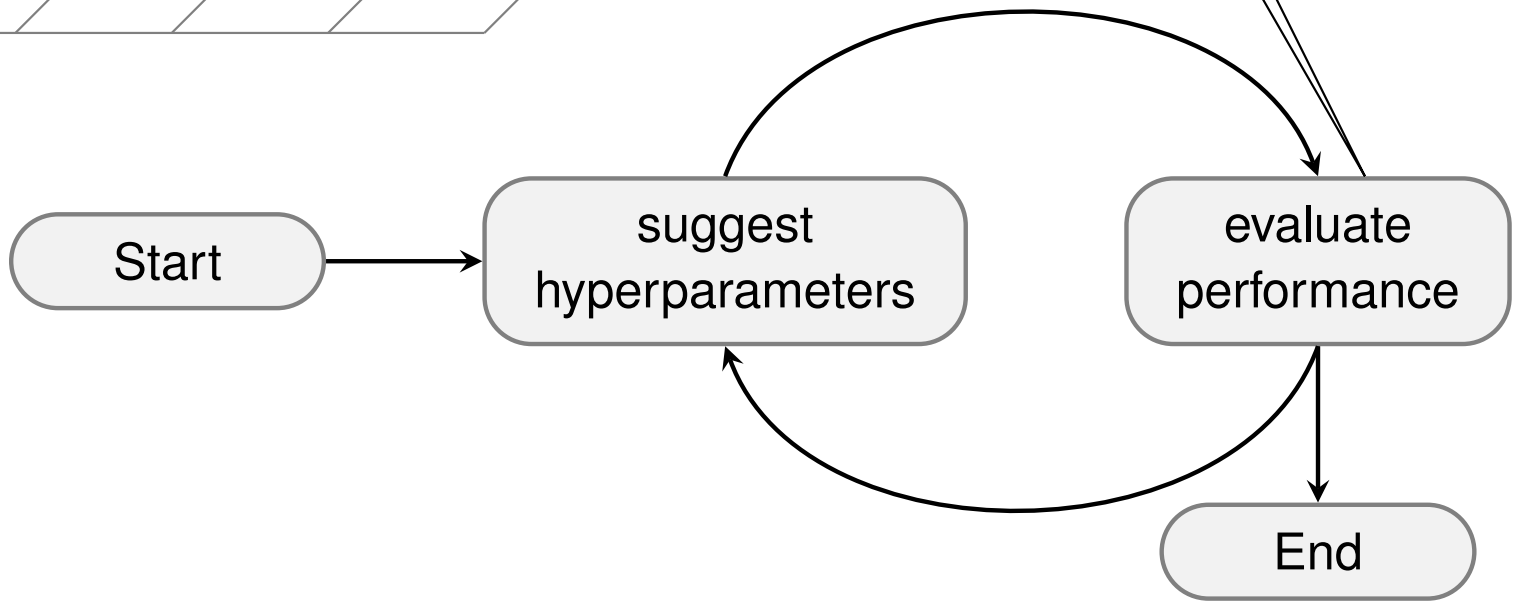


TUNING

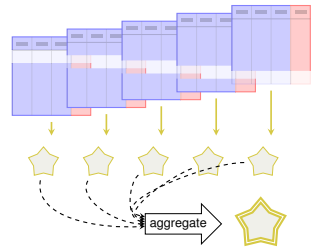


`Learner (lrn())`
`Task`
`Resampling (rsmp())`
`Measure (msr())`

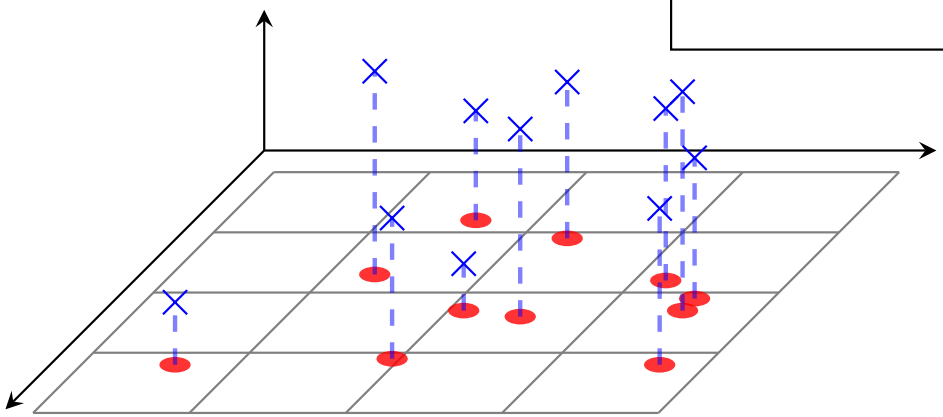




TUNING



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



Start

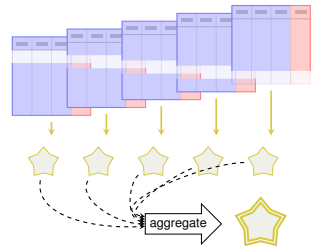
suggest hyperparameters

evaluate performance

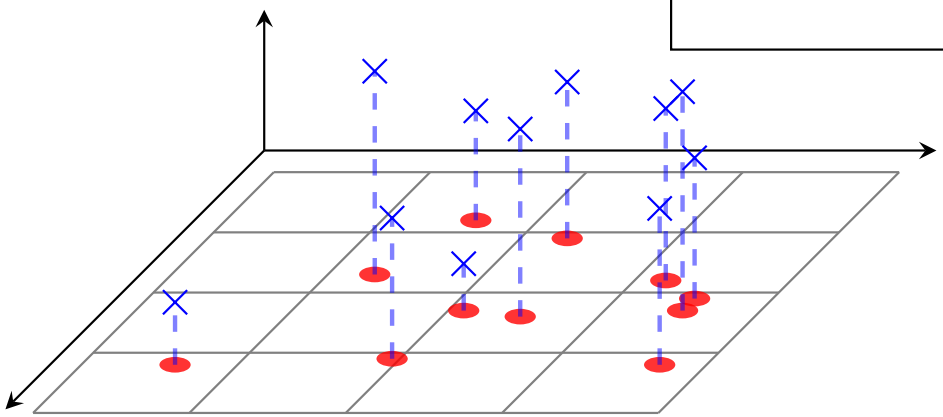
End

search space
`Tuner`

TUNING



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



Start

suggest hyperparameters

evaluate performance

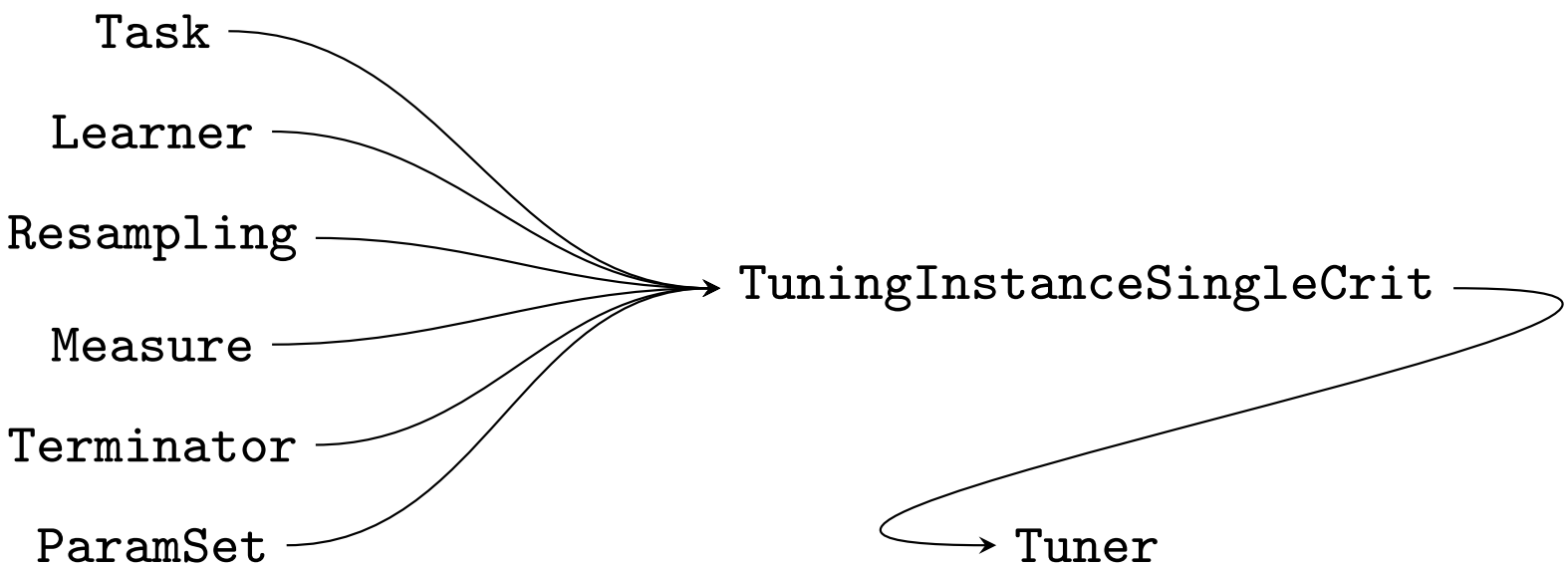
search space
`Tuner`

`Terminator` (when to finish)

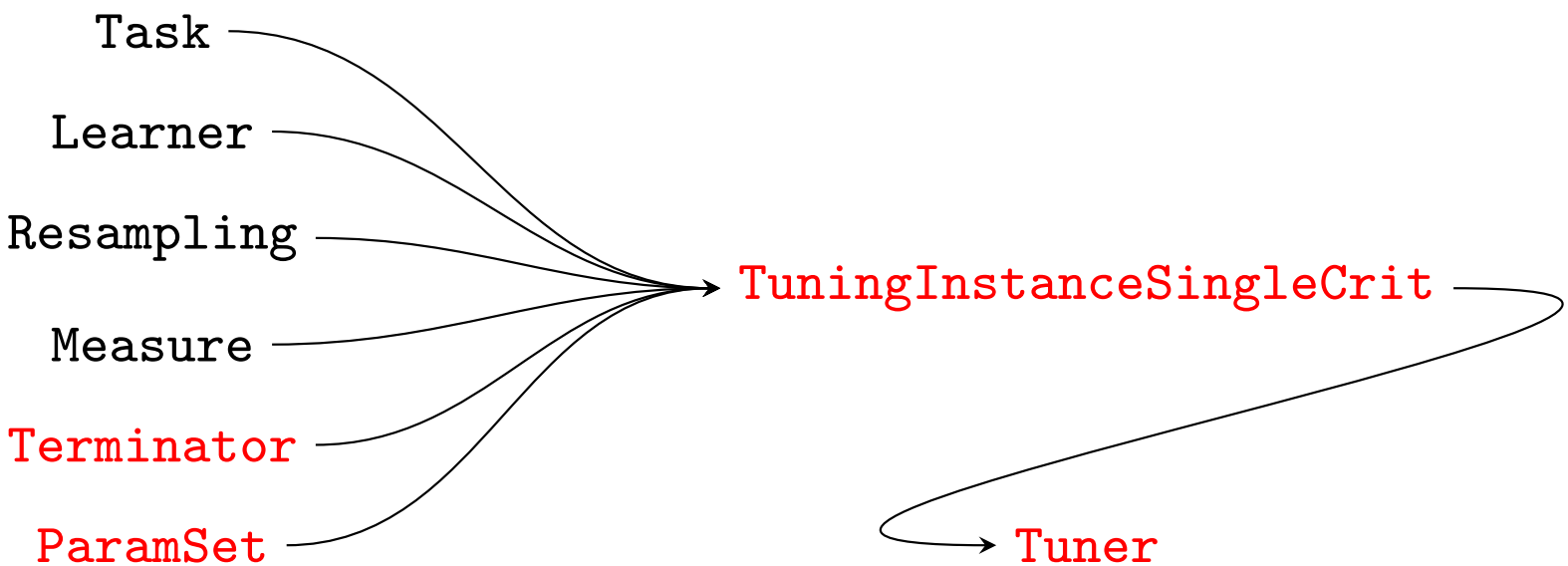
End

Tuning in mlr3

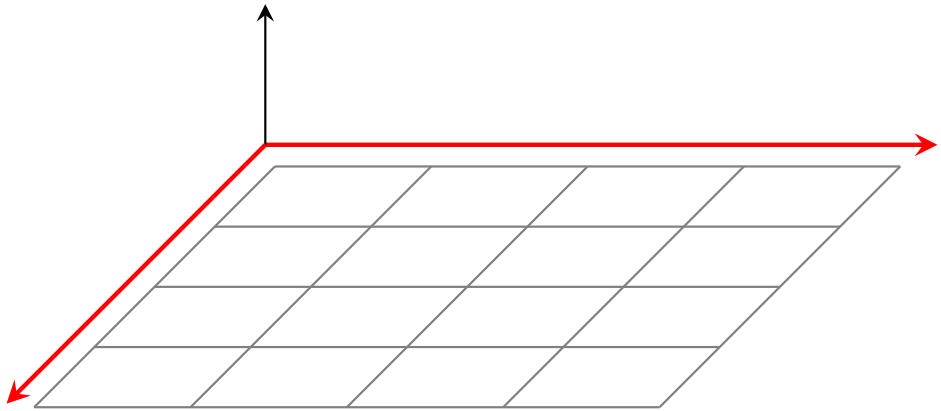
OBJECTS IN TUNING



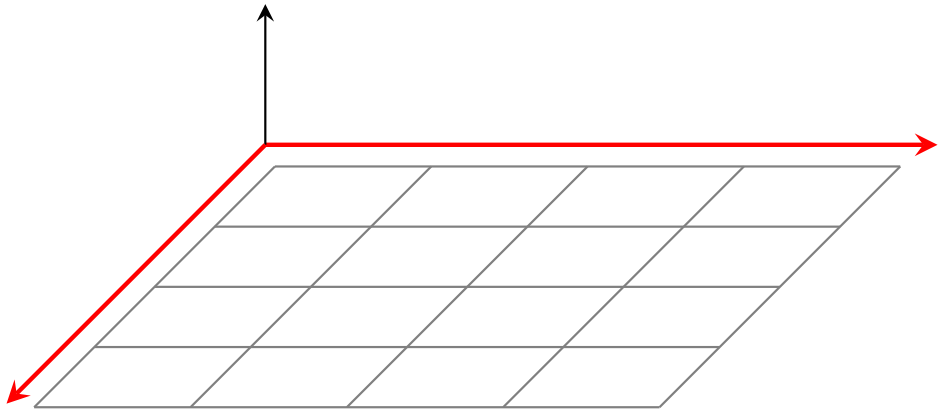
OBJECTS IN TUNING



SEARCH SPACE

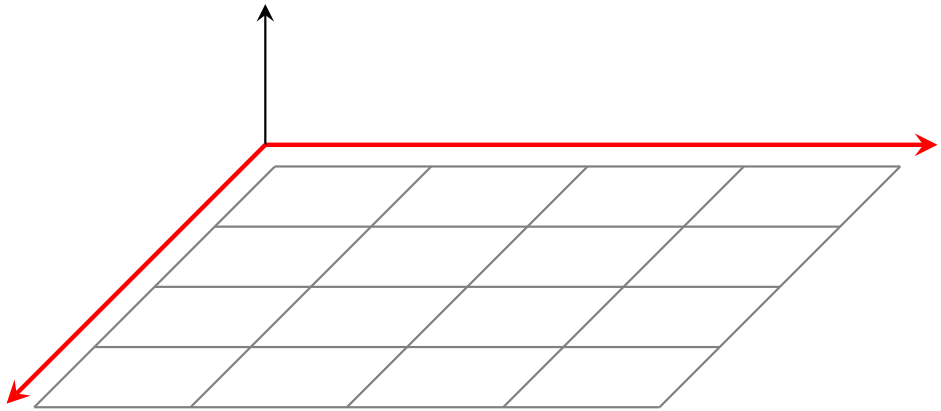


SEARCH SPACE



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

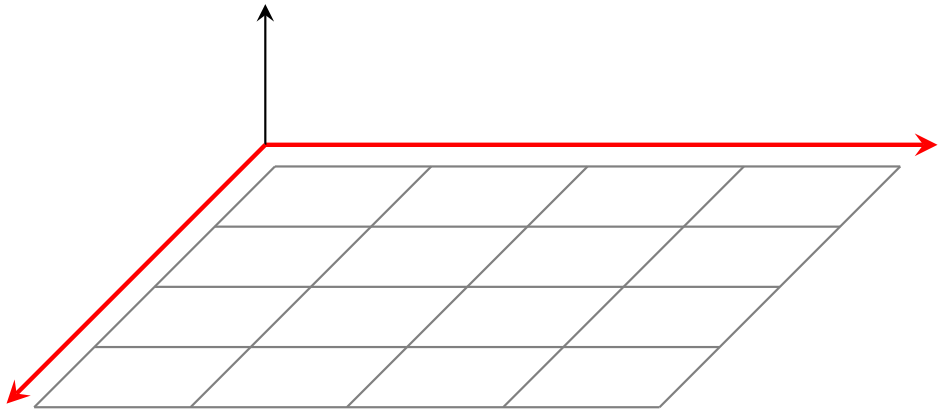
SEARCH SPACE



```
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

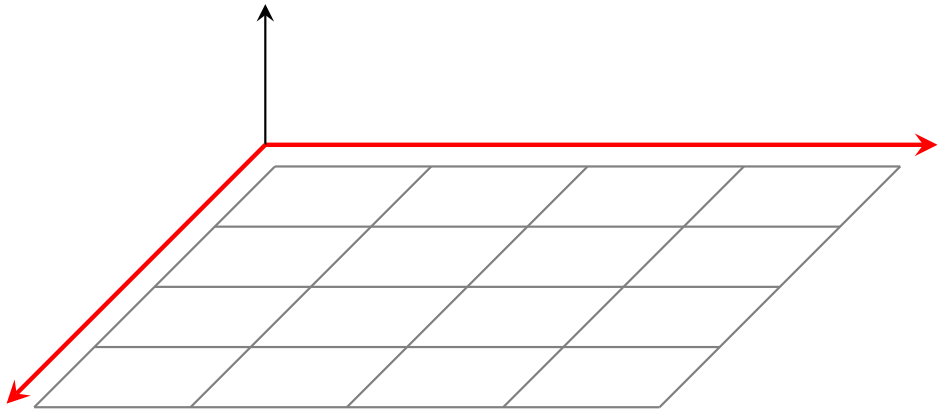


```
ParamSet$new(list(param1, param2, ...))
```

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", lower = 1, upper = 20)
))
```

SEARCH SPACE SHORT FORM



```
ps(id1 = domain1, id2 = domain2, ...)
```

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(lower = 1, upper = 20)
)
```

TERMINATION

- Tuning needs a *termination condition*: when to finish

TERMINATION

- Tuning needs a *termination condition*: when to finish
- `Terminator` class

TERMINATION

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

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
  ```
  ```

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, k=0
```

TUNING METHOD

- need to choose a *tuning method*

TUNING METHOD

- need to choose a *tuning method*
- `Tuner` class

TUNING METHOD

- need to choose a *tuning method*
- `Tuner` class
- `mlr_tuners` dictionary, `tnr()` short form

TUNING METHOD

- need to choose a *tuning method*
- `Tuner` class
- `mlr_tuners` dictionary, `tnr()` short form
- ```
  as.data.table(mlr_tuners)
  #>                key
  #> 1:          cmaes
  #> 2: design_points
  #> 3:          gensa
  #> 4:   grid_search
  #> 5:          irace
  #> 6:         nloptr
  #> 7: random_search
  ```
  ```

TUNING METHOD

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

TUNING METHOD

- load `Tuner` with `tnr()`, set parameters
-
```
gsearch = tnr("grid_search", resolution = 3)

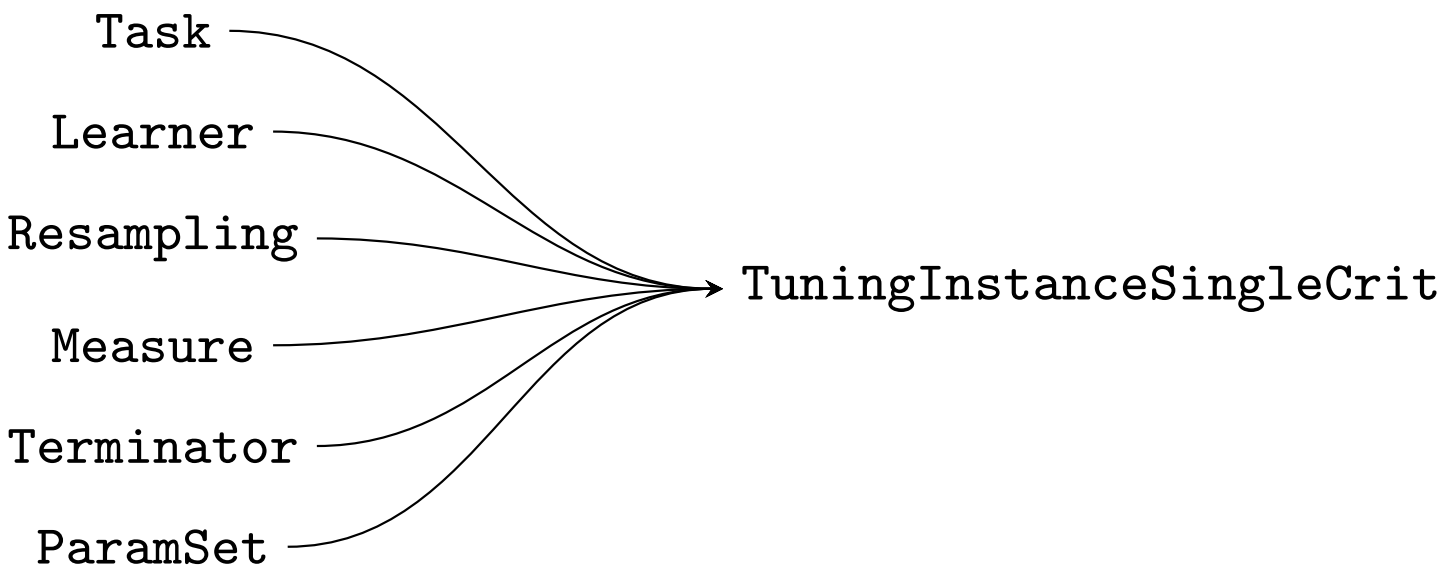
print(gsearch)
#> <TunerGridSearch>
#> * Parameters: resolution=3, batch_size=1
#> * Parameter classes: ParamLgl, ParamInt, ParamDbl, ParamFct
#> * Properties: dependencies, single-crit, multi-crit
#> * Packages: -
```

TUNING METHOD

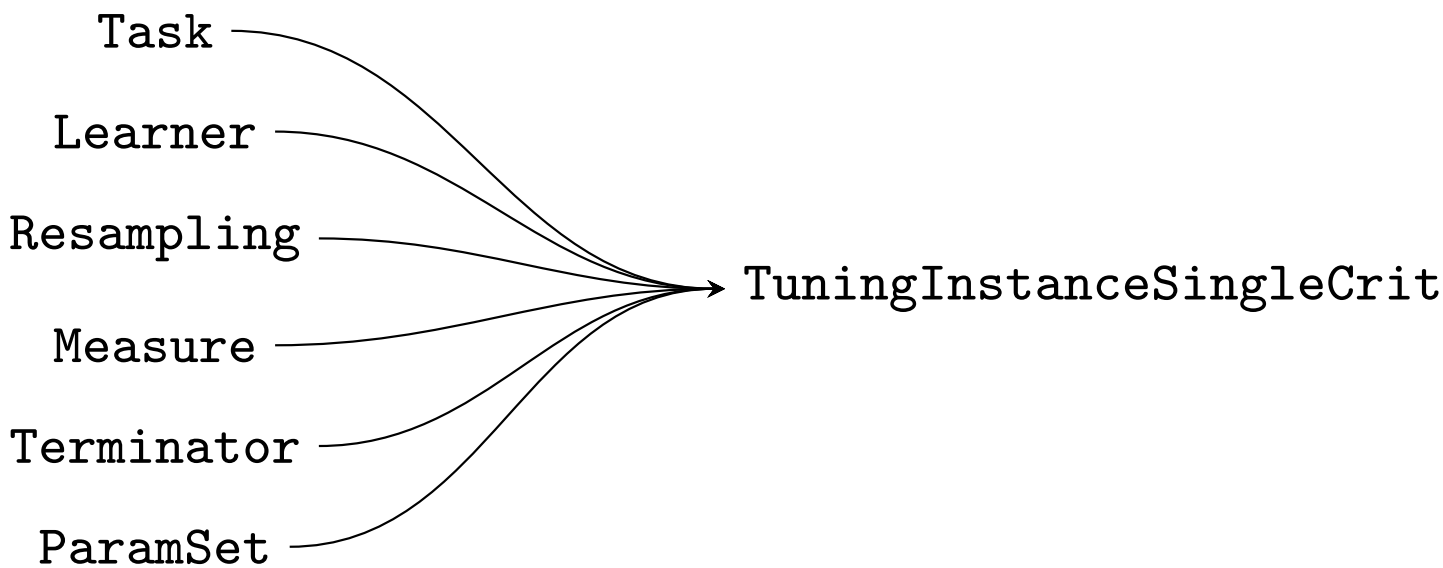
- load `Tuner` with `tnr()`, set parameters
- ```
  gsearch = tnr("grid_search", resolution = 3)

  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(task = tsk("iris"),
  learner = lrn("classif.kknn", kernel = "rectangular"),
  resampling = rsmp("holdout"), measure = msr("classif.ce"),
  terminator = trm("none"), search_space = searchspace_knn
)
```

CALLING THE TUNER

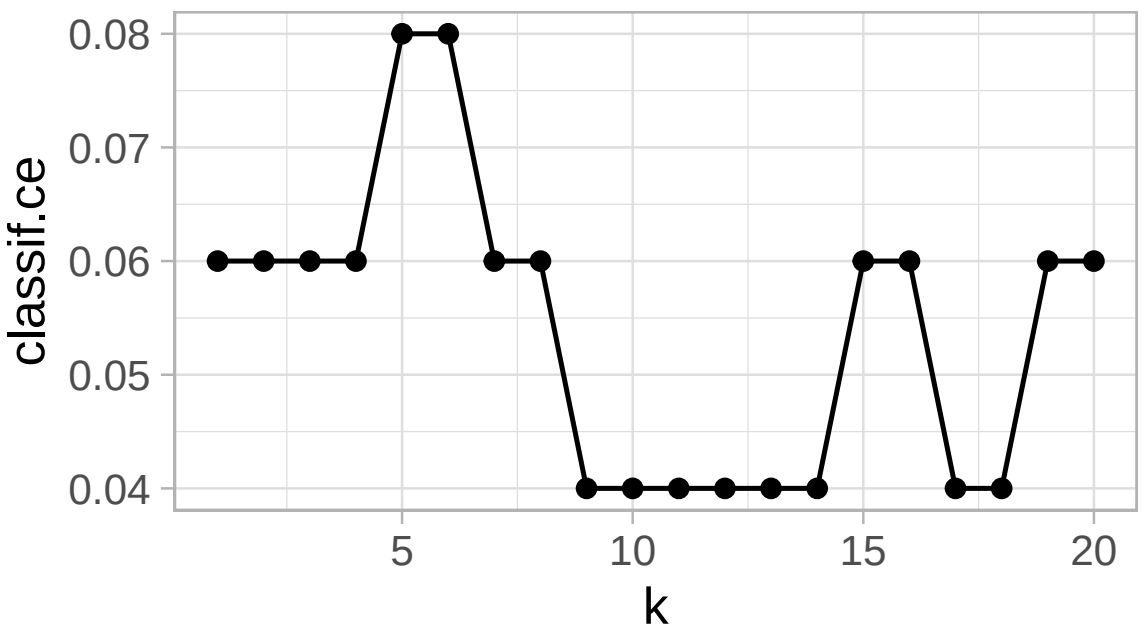
```
gsearch$optimize(inst)
#> INFO  [17:08:25.057] [bbotk] Starting to optimize 1 parameter(s) with '<OptimizerGridSea
#> INFO  [17:08:25.276] [bbotk] Evaluating 1 configuration(s)
#> INFO  [17:08:27.073] [bbotk] Result of batch 1:
#> INFO  [17:08:27.075] [bbotk]   k classif.ce runtime_learners
#> INFO  [17:08:27.075] [bbotk]  10       0.04             0.01 8ce6dc04-17e7-4687-906f-932
#> INFO  [17:08:27.077] [bbotk] Evaluating 1 configuration(s)
#> INFO  [17:08:27.234] [bbotk] Result of batch 2:
#> INFO  [17:08:27.235] [bbotk]  k classif.ce runtime_learners
#> INFO  [17:08:27.235] [bbotk]  1       0.06             0.03 efa6c724-c1d3-485b-8547-38af
#> INFO  [17:08:27.238] [bbotk] Evaluating 1 configuration(s)
#> INFO  [17:08:27.418] [bbotk] Result of batch 3:
#> INFO  [17:08:27.421] [bbotk]   k classif.ce runtime_learners
#> INFO  [17:08:27.421] [bbotk]  20       0.08             0.02 15b34537-ce27-4169-9e88-332
#> INFO  [17:08:27.432] [bbotk] Finished optimizing after 3 evaluation(s)
#> INFO  [17:08:27.434] [bbotk] Result:
#> INFO  [17:08:27.437] [bbotk]   k learner_param_vals  x_domain classif.ce
#> INFO  [17:08:27.437] [bbotk]  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

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

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

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



RECAP

1. Create a `Task`, `Learner`, `Resampling`, `Measure`, `Terminator` (defines when to stop), and a `ParamSet` (defines the search space):

```
task = tsk("iris")
learner = lrn("classif.kknn", kernel = "rectangular")
resampling = rsmp("holdout")
measure = msr("classif.ce")
terminator = trm("evals", n_evals = 2)
searchspace_knn = ParamSet$new(list(
  ParamInt$new("k", lower = 1, upper = 20)
))
```

2. Create the `TuningInstanceSingleCrit` object:

```
inst = TuningInstanceSingleCrit$new(task, learner,
  resampling, measure, terminator, searchspace_knn)
```

3. Create the `Tuner` (tuning method) and optimize the learner by passing over the previously created instance to the `$optimize` method:

```
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
```

Parameter Transformation

PARAMETER TRANSFORMATION

- Sometimes we do not want to optimize over an evenly spaced range

PARAMETER TRANSFORMATION

- Sometimes we do not want to optimize over an evenly spaced range
- $k = 1$ vs. $k = 2$ probably more interesting than $k = 101$ vs. $k = 102$

PARAMETER TRANSFORMATION

- Sometimes we do not want to optimize over an evenly spaced 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 optimize over an evenly spaced 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 optimize over an evenly spaced 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 optimize over an evenly spaced range
- $k = 1$ vs. $k = 2$ probably more interesting than $k = 101$ vs. $k = 102$

$\Rightarrow$ Transformations

- Part of `ParamSet`

Example:

1. optimize from $\log(1) \ldots \log(100)$

PARAMETER TRANSFORMATION

- Sometimes we do not want to optimize over an evenly spaced range
- $k = 1$ vs. $k = 2$ probably more interesting than $k = 101$ vs. $k = 102$

$\Rightarrow$ Transformations

- Part of `ParamSet`

Example:

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

PARAMETER TRANSFORMATION

- Sometimes we do not want to optimize over an evenly spaced range
- $k = 1$ vs. $k = 2$ probably more interesting than $k = 101$ vs. $k = 102$

$\Rightarrow$ Transformations

- Part of `ParamSet`

Example:

1. optimize from $\log(1) \ldots \log(100)$
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 optimize over an evenly spaced range
- $k = 1$ vs. $k = 2$ probably more interesting than $k = 101$ vs. $k = 102$

$\Rightarrow$ Transformations

- Part of `ParamSet`

Example:

1. optimize from $\log(1) \ldots \log(100)$
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", log(1), log(50))
))
searchspace_knn_trafo$trafo = function(x, param_set) {
  x$k = round(exp(x$k))
  return(x)
}
```

PARAMETER TRANSFORMATION

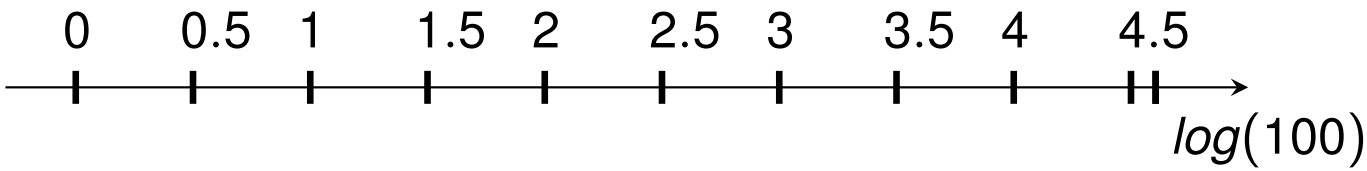
What is our transformation doing?

PARAMETER TRANSFORMATION

What is our transformation doing?

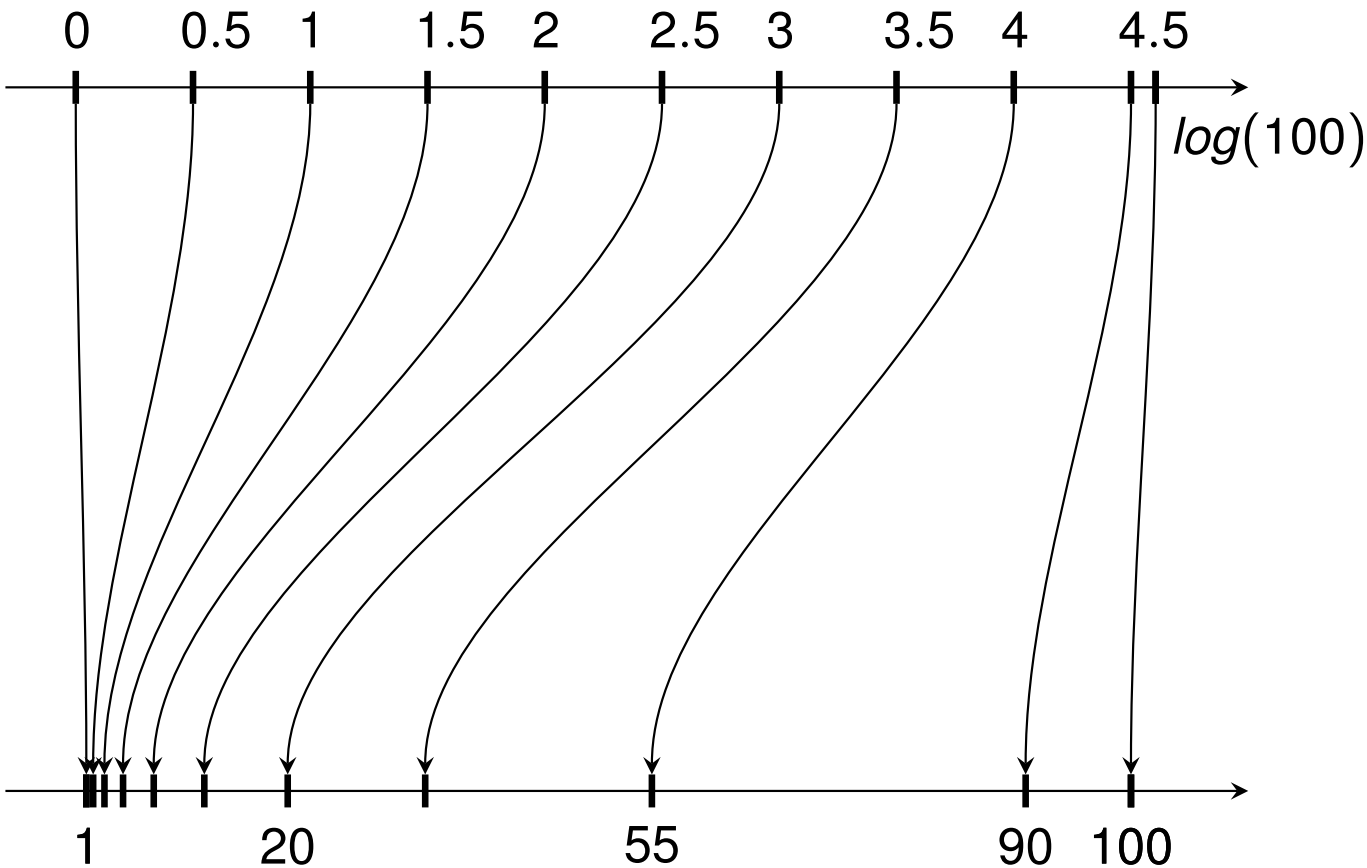
PARAMETER TRANSFORMATION

What is our transformation doing?



PARAMETER TRANSFORMATION

What is our transformation doing?



PARAMETER TRANSFORMATION

Tuning again. . .

PARAMETER TRANSFORMATION

Tuning again. . .

```
inst$result

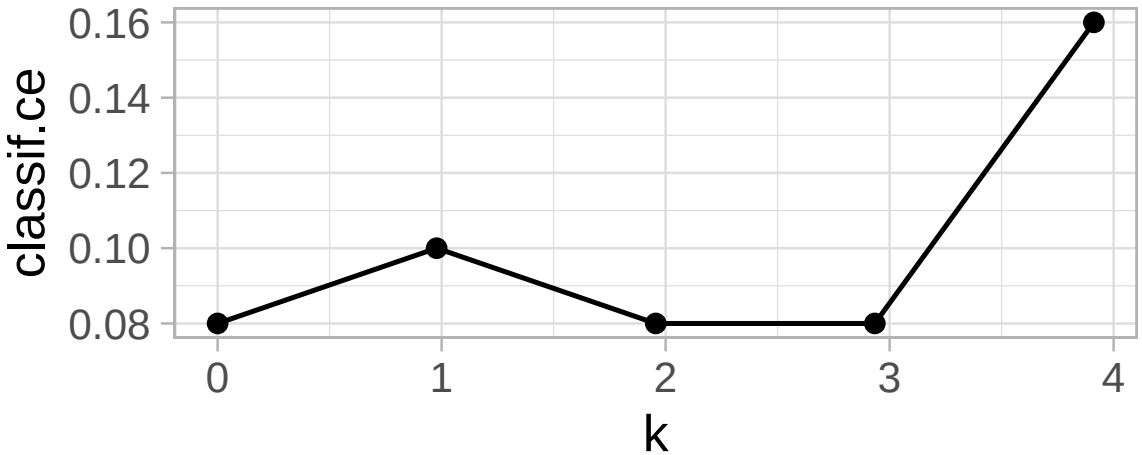
#>      k learner_param_vals  x_domain classif.ce
#> 1: 2.9          <list[2]> <list[1]>       0.08

inst$result$x_domain

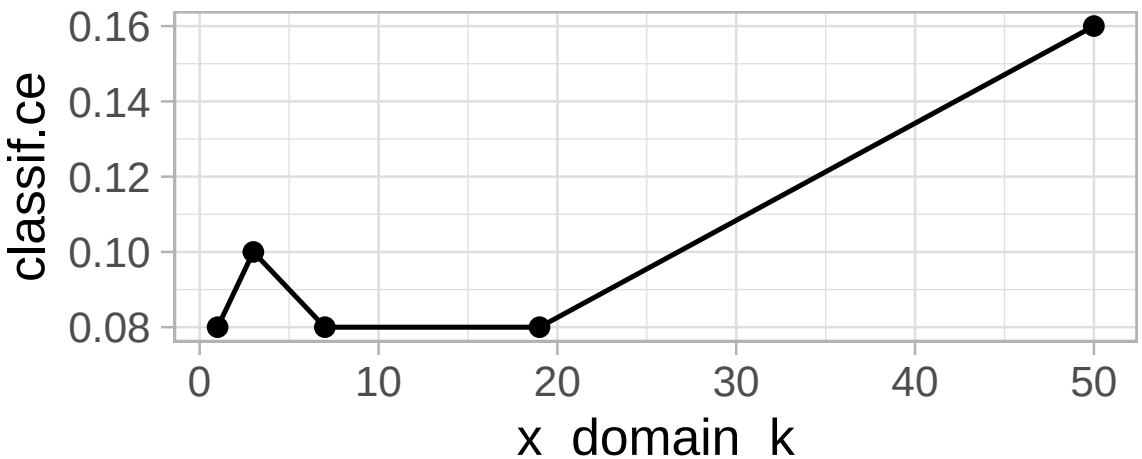
#> [[1]]
#> [[1]]$k
#> [1] 19
```

PARAMETER TRANSFORMATION

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




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



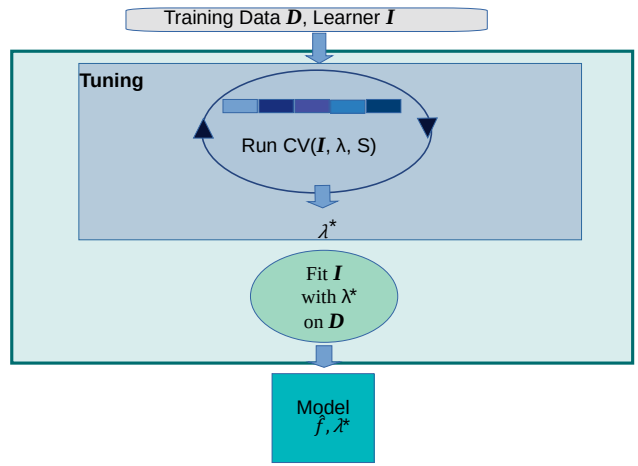
Nested Resampling

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
  - Predicting: Just use final model



NESTED RESAMPLING

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

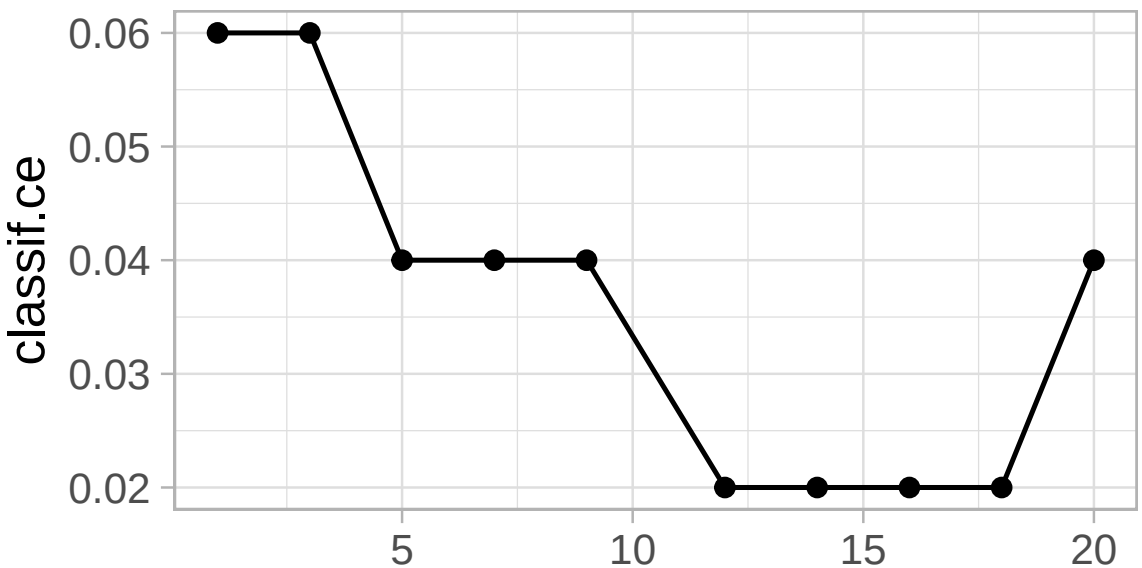
```
optlrn$train(tsk("iris"))
```

```
optlrn$model$learner

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

NESTED RESAMPLING

```
archive = as.data.table(optlrn$tuning_instance$archive)
ggplot(archive, aes(x = k, y = classif.ce)) +
  geom_line() + geom_point() + xlab("")
```



NESTED RESAMPLING

```
rr = resample(task = tsk("iris"), learner = optlrn,
  resampling = rsmp("holdout"), store_models = TRUE)
archive = as.data.table(rr$learners[[1]]$tuning_instance$archive)
ggplot(archive, aes(x = k, y = classif.ce)) +
  geom_line() + geom_point() + xlab("")
```