

# Tuning Machine Learning Algorithms with mlr3

## mlr3tuning

Department of Statistics – LMU Munich



# 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
- ⇒ 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
- ⇒ We do *black box optimization* (“Try stuff and see what works”)

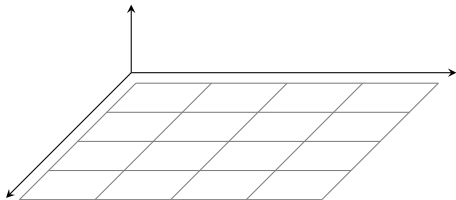
Tuning toolbox for mlr3:

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

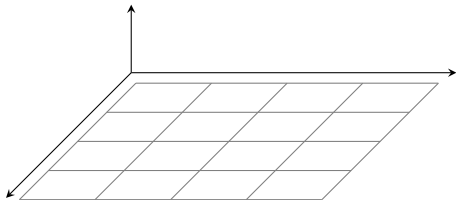
# Tuning



# TUNING

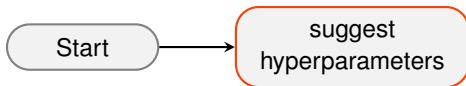
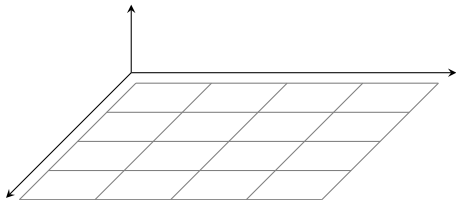


# TUNING

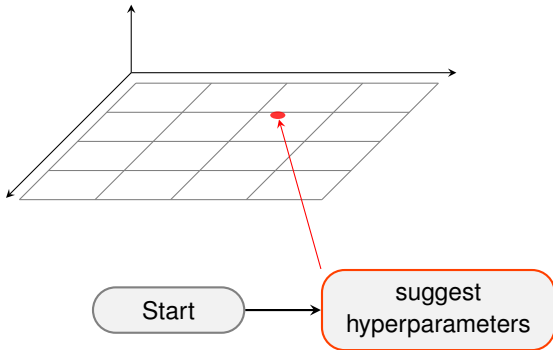


Start

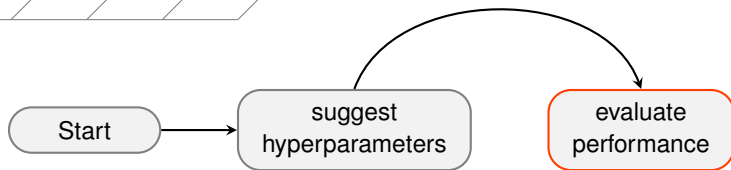
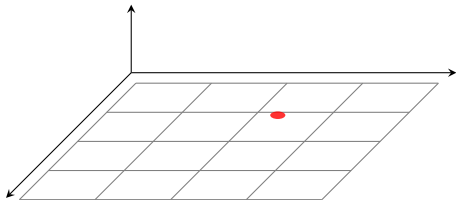
# TUNING



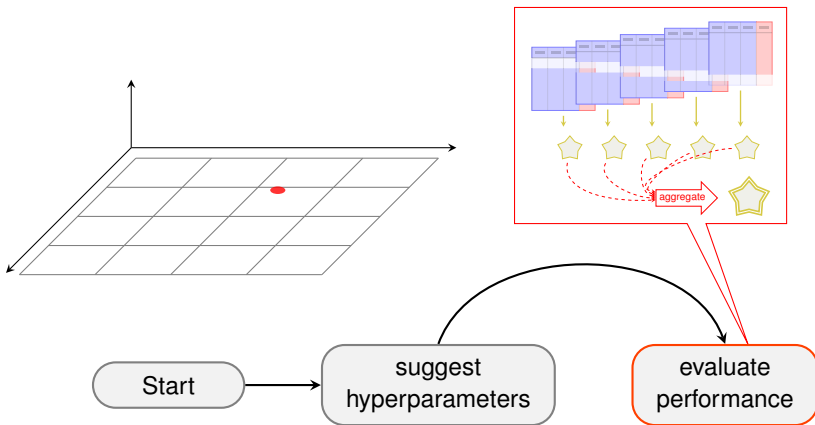
# TUNING



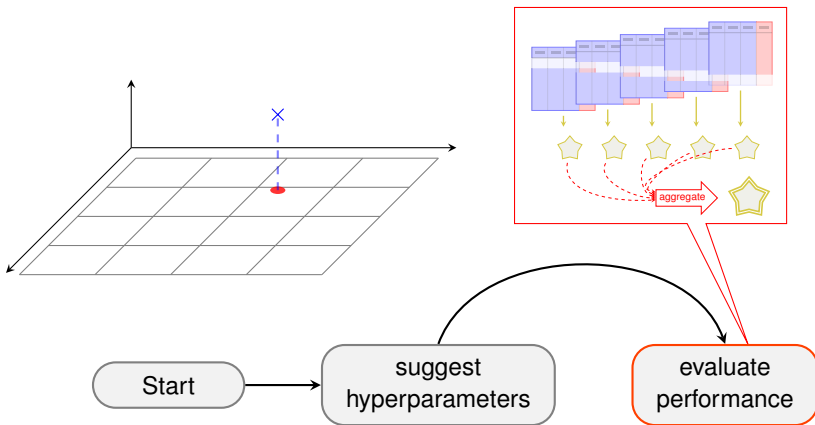
# TUNING



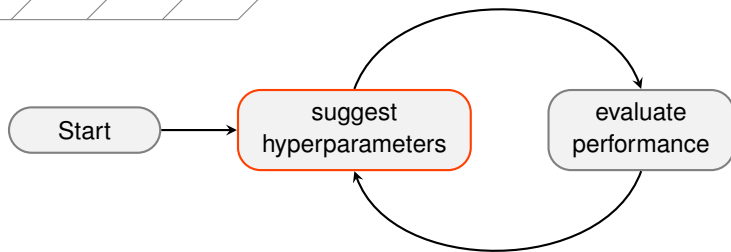
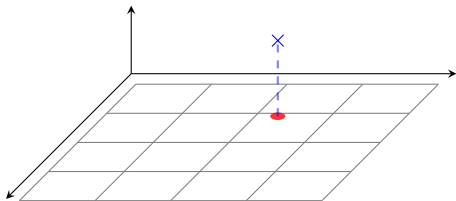
# TUNING



# TUNING

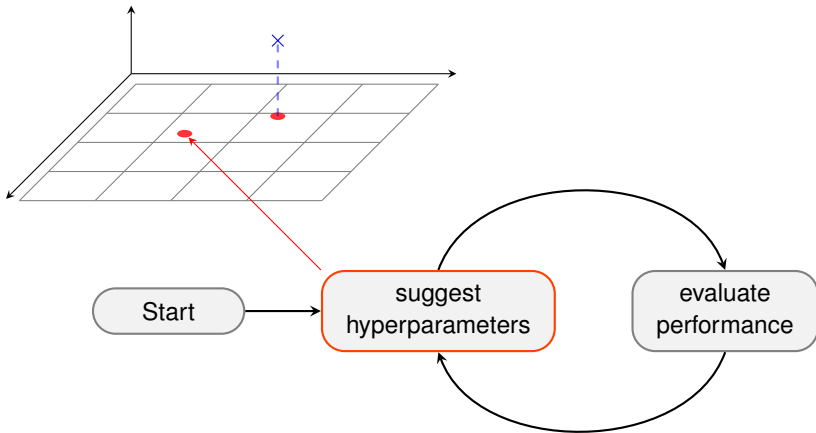


# TUNING

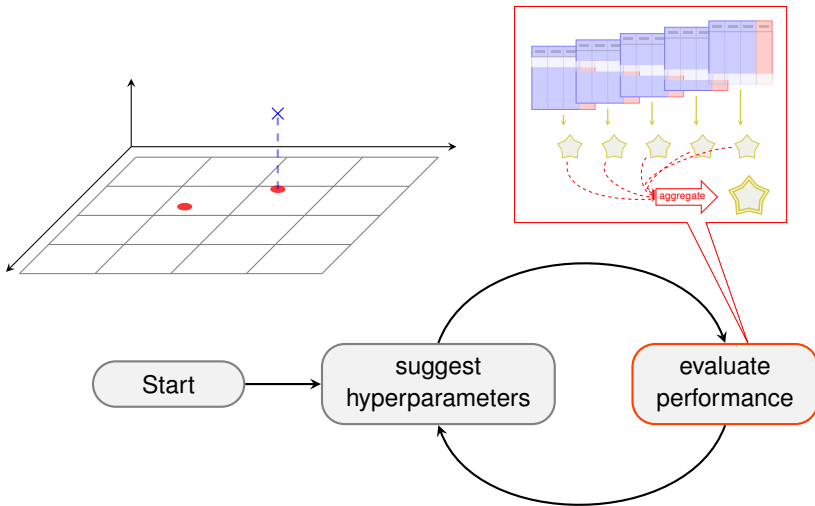




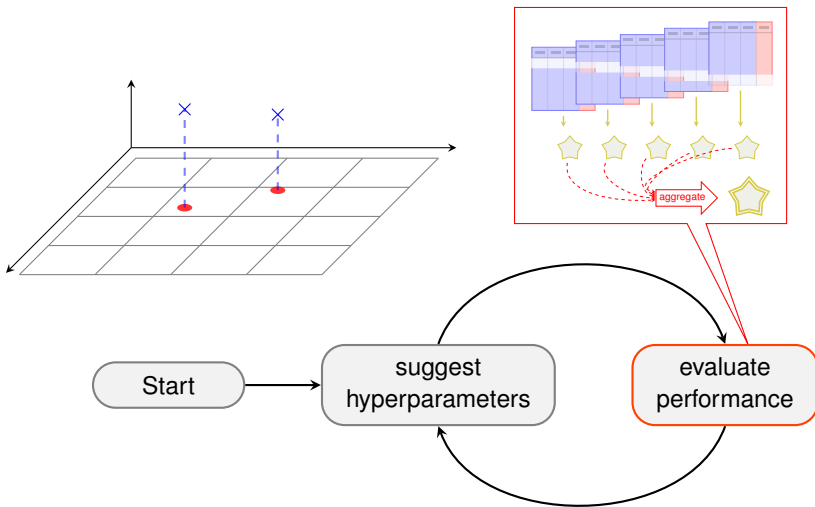
# TUNING



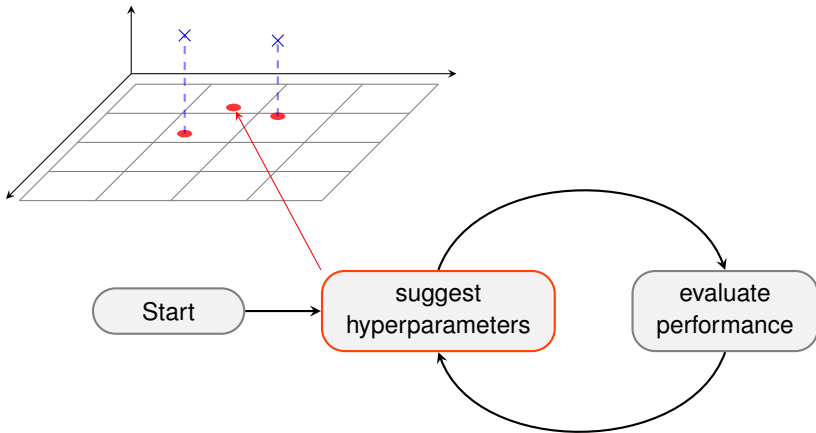
# TUNING



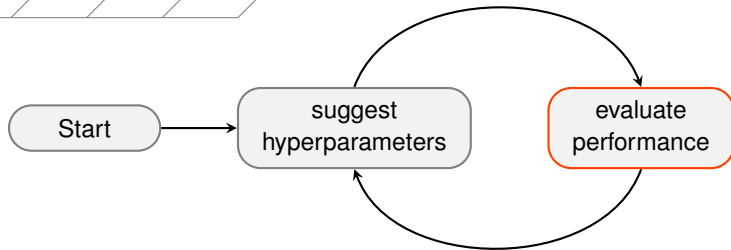
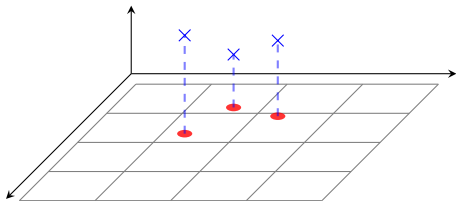
# TUNING



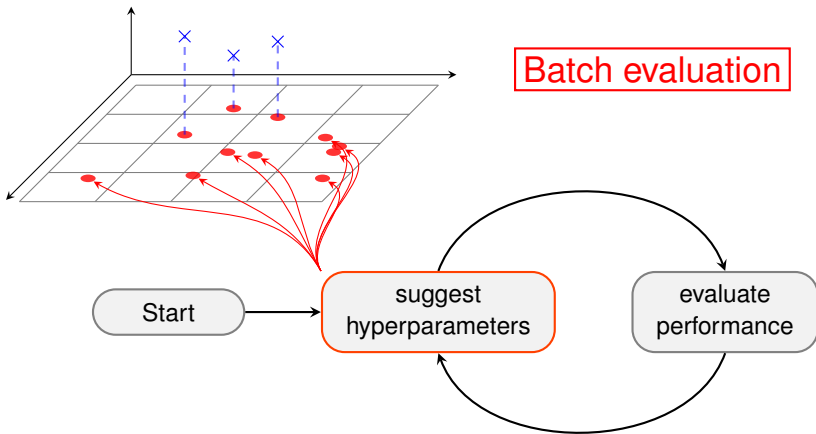
# TUNING



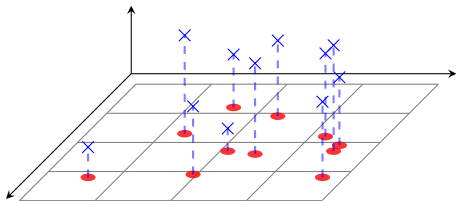
# TUNING



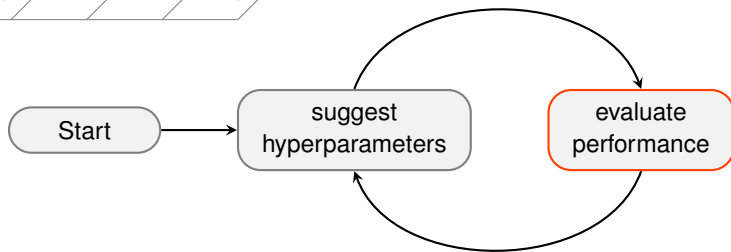
# TUNING



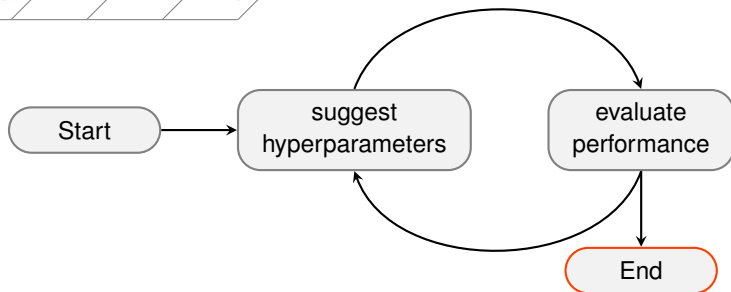
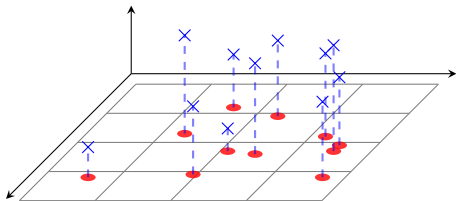
# TUNING



Batch evaluation

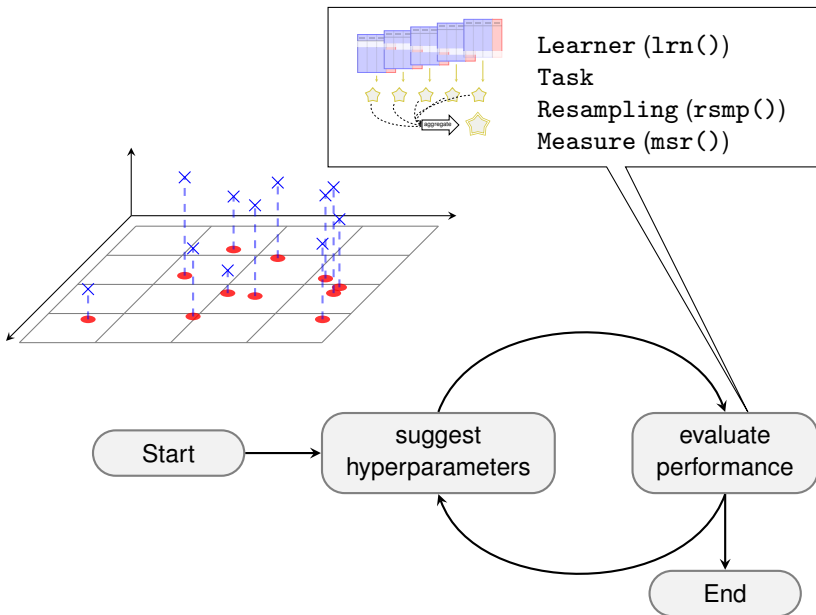


# TUNING

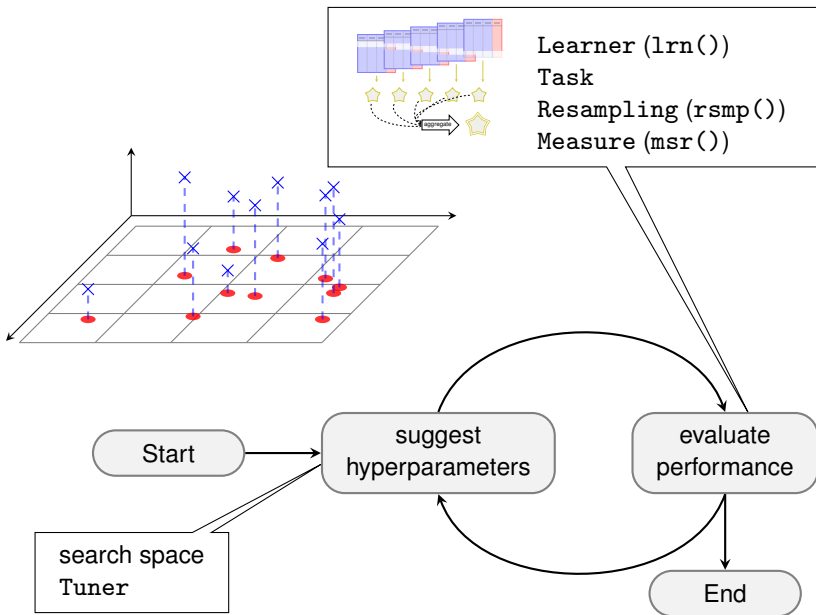




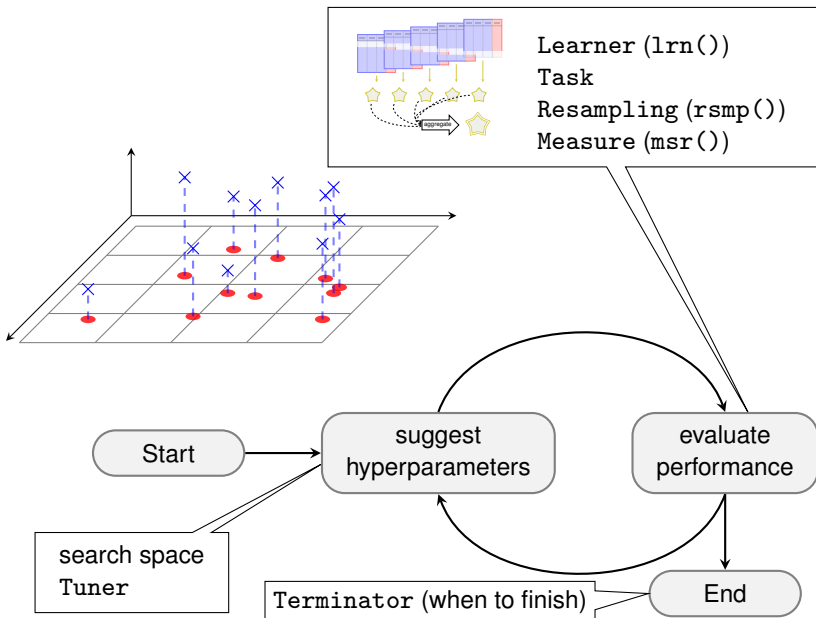
# TUNING



# TUNING

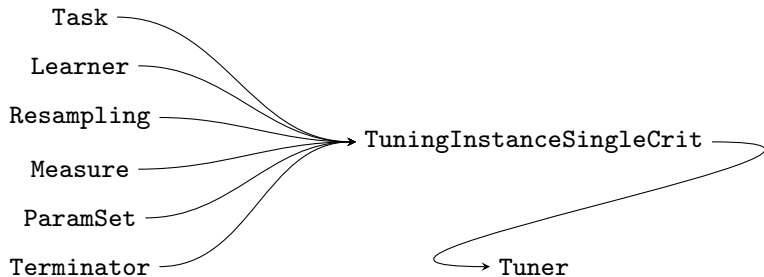


# TUNING

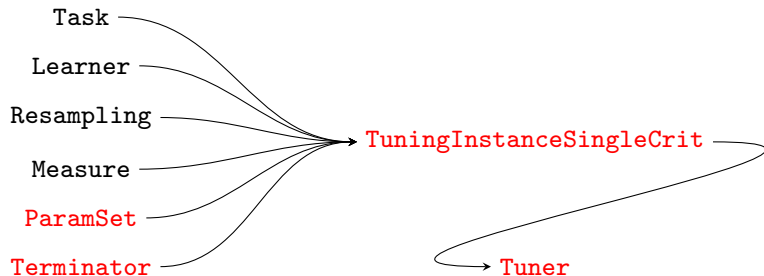


# Tuning in mlr3

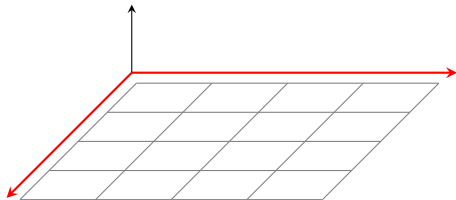
# OBJECTS IN TUNING



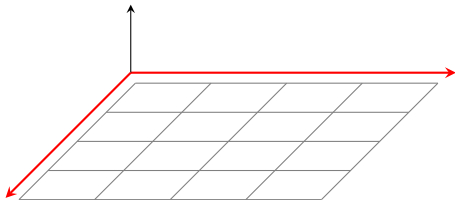
# OBJECTS IN TUNING



# SEARCH SPACE



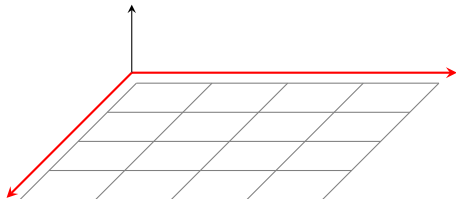
# SEARCH SPACE



```
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)
```

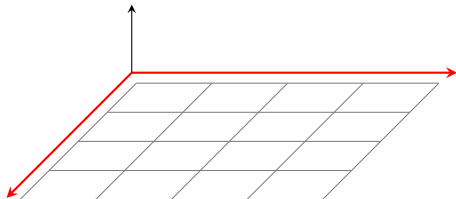
*Logical* parameter

```
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", 1, 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
```

# 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: design_points
#> 2:          gensa
#> 3:   grid_search
#> 4:         nloptr
#> 5: 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

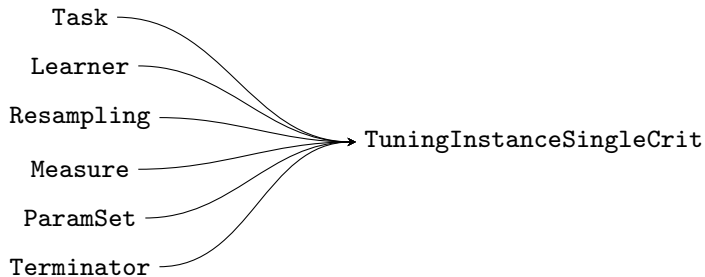
- load Tuner with `tnr()`, set parameters

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

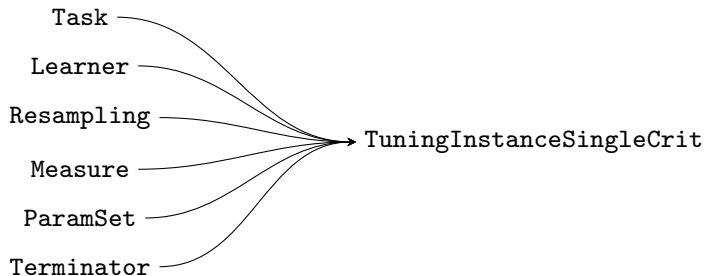
```
print(gsearch)
#> <TunerGridSearch>
#> * Parameters: resolution=3, batch_size=1
#> * Parameter classes: ParamLgl, ParamInt, ParamDbl, ParamFct
#> * Properties: dependencies, single-crit, multi-crit
#> * Packages: -
```

- common parameter `batch_size` for parallelization

# CALLING THE TUNER



# CALLING THE TUNER



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



# CALLING THE TUNER

```
gsearch$optimize(inst)
```

```
#> INFO [16:10:21.277] Starting to optimize 1 parameter(s) with '<Optim
#> INFO [16:10:21.327] Evaluating 1 configuration(s)
#> INFO [16:10:22.323] Result of batch 1:
#> INFO [16:10:22.325] k classif.ce      resample_result
#> INFO [16:10:22.325] 10      0.04 <ResampleResult[18]>
#> INFO [16:10:22.327] Evaluating 1 configuration(s)
#> INFO [16:10:22.568] Result of batch 2:
#> INFO [16:10:22.570] k classif.ce      resample_result
#> INFO [16:10:22.570] 1      0.06 <ResampleResult[18]>
#> INFO [16:10:22.572] Evaluating 1 configuration(s)
#> INFO [16:10:22.628] Result of batch 3:
#> INFO [16:10:22.630] k classif.ce      resample_result
#> INFO [16:10:22.630] 20      0.08 <ResampleResult[18]>
#> INFO [16:10:22.636] Finished optimizing after 3 evaluation(s)
#> INFO [16:10:22.637] Result:
#> INFO [16:10:22.639] k learner_param_vals x_domain classif.ce
#> INFO [16:10:22.639] 10      <list[2]> <list[1]>      0.04
#> k learner_param_vals x_domain classif.ce
#> 1: 10      <list[2]> <list[1]>      0.04
```

# TUNING RESULTS

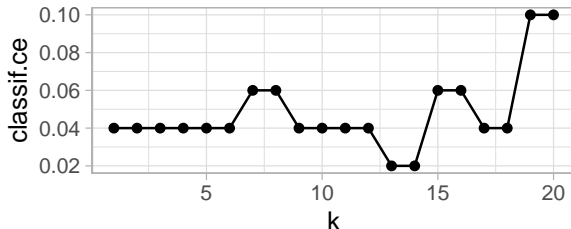
```
gsearch = tnr("grid_search", resolution = 20)

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

gsearch$optimize(inst)

#>      k learner_param_vals  x_domain classif.ce
#> 1: 14          <list[2]> <list[1]>         0.02

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



# RECAP

```
inst = TuningInstanceSingleCrit$new(  
  tsk("iris"), lrn("classif.kknn", kernel="rectangular"),  
  rsmp("holdout"), msr("classif.ce"),  
  searchspace_knn, trm("evals", n_evals = 2)  
)  
  
gsearch = tnr("grid_search", resolution = 3)  
  
gsearch$optimize(inst)  
  
#>    k learner_param_vals  x_domain classif.ce  
#> 1: 1                <list[2]> <list[1]>    0.06
```

# 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.  $k = 102$

⇒ 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.  $k = 102$

⇒ 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.  $k = 102$

⇒ 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.  $k = 102$

⇒ Transformations

- Part of `ParamSet`

Example:

- 1 optimize from  $\log(1) \dots \log(100)$  (`k_before_trafo`)

# 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.  $k = 102$

⇒ Transformations

- Part of `ParamSet`

Example:

- 1 optimize from  $\log(1) \dots \log(100)$  (`k_before_trafo`)
- 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.  $k = 102$

⇒ Transformations

- Part of `ParamSet`

Example:

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

⇒ Transformations

- Part of ParamSet

Example:

- 1 optimize from  $\log(1) \dots \log(100)$  (`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(50))  
))  
searchspace_knn_trafo$trafo = function(x, param_set) {  
  return(list(k = round(exp(x$k_before_trafo))))  
}
```

# PARAMETER TRANSFORMATION

What is our transformation doing?



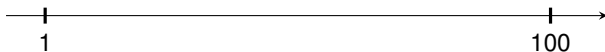
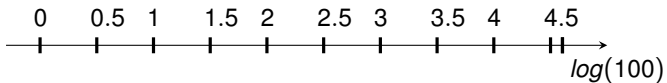
# PARAMETER TRANSFORMATION

What is our transformation doing?



# PARAMETER TRANSFORMATION

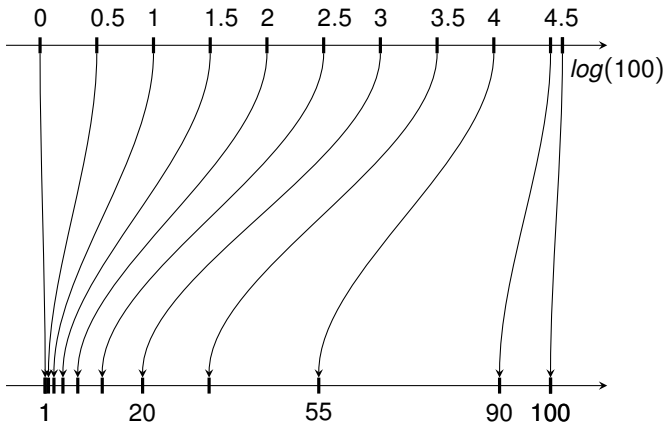
What is our transformation doing?





# PARAMETER TRANSFORMATION

What is our transformation doing?



# PARAMETER TRANSFORMATION

Tuning again...

# PARAMETER TRANSFORMATION

Tuning again...

```
inst$result

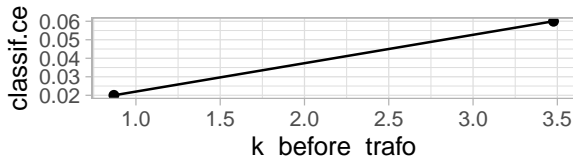
#>      k_before_trafo learner_param_vals  x_domain classif.ce
#> 1:                0.87                <list[2]> <list[1]>      0.02

inst$result$x_domain

#> [[1]]
#> [[1]]$k
#> [1] 2
```

# PARAMETER TRANSFORMATION

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



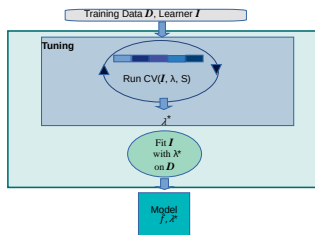
```
ggplot(inst$archive$data(unnest = "x_domain"),  
  aes(x = x_domain_k, y = classif.ce)) + geom_line() + geom_point()
```



# Nested Resampling

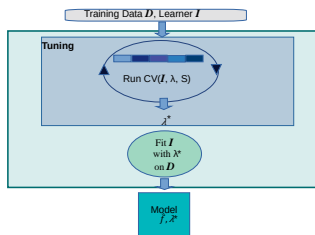
# NESTED RESAMPLING

- Need to perform nested resampling to estimate tuned learner performance
- ⇒ 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

- Need to perform nested resampling to estimate tuned learner performance
- ⇒ 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(lrn("classif.kknn", kernel="rectangular"),  
  rsmp("holdout"), msr("classif.ce"), searchspace_knn,  
  trm("none"), tnr("grid_search", resolution = 10))
```



# NESTED RESAMPLING

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

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

# NESTED RESAMPLING

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

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

```
optltn$model$learner
```

```
#> <LearnerClassifKKNN:classif.kknn>  
#> * Model: list  
#> * Parameters: kernel=rectangular, k=16  
#> * Packages: kknn  
#> * Predict Type: response  
#> * Feature types: logical, integer, numeric, factor, ordered  
#> * Properties: multiclass, twoclass
```

# NESTED RESAMPLING

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

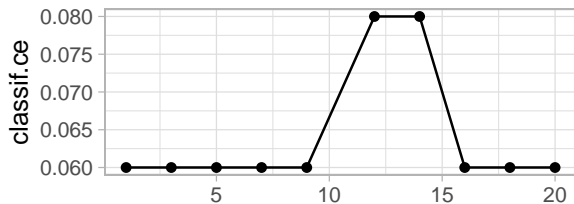
```
optltn$train(tsk("iris"))
```

```
optltn$model$learner
```

```
#> <LearnerClassifKKNN:classif.kknn>  
#> * Model: list  
#> * Parameters: kernel=rectangular, k=16  
#> * Packages: kknn  
#> * Predict Type: response  
#> * Feature types: logical, integer, numeric, factor, ordered  
#> * Properties: multiclass, twoclass
```

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

# NESTED RESAMPLING

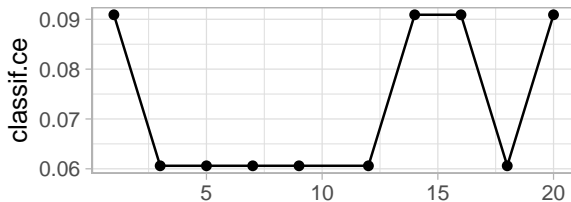


# NESTED RESAMPLING

```
result = resample(tsk("iris"), optlrn, rsmp("holdout"),  
                 store_models = TRUE)
```

# NESTED RESAMPLING

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



# NESTED RESAMPLING

Aggregate performances of outer folds

```
result$aggregate()
```

```
#> classif.ce
```

```
#>      0.04
```

# NESTED RESAMPLING

Aggregate performances of outer folds

```
result$aggregate()  
  
#> classif.ce  
#>      0.04
```

Retrieve inner tuning results

```
result$data$learner[[1]]$tuning_result  
  
#>      k learner_param_vals  x_domain classif.ce  
#> 1: 3      <list[2]> <list[1]>      0.061
```



# Outro

# TUNING WITH MLR3TUNING

## Tuning a Learner

- 1 Construct a `TuningInstanceSingleCrit`
  - 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$optimize()`

## Nested Resampling

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