Tuning Machine Learning Algorithms with mlr3

mlr3tuning

Department of Statistics – LMU Munich

November 07, 2019

Intro

TUNING

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

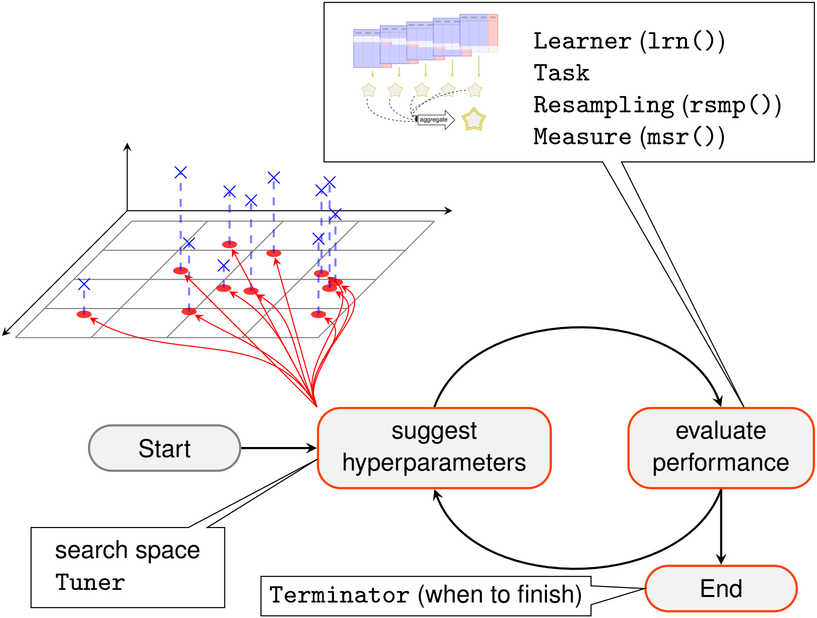
$\Rightarrow$ We do *black box optimization* ("Try stuff and see what works")

Tuning toolbox for `mlr3`:

```r
library("mlr3tuning")
```

Tuning

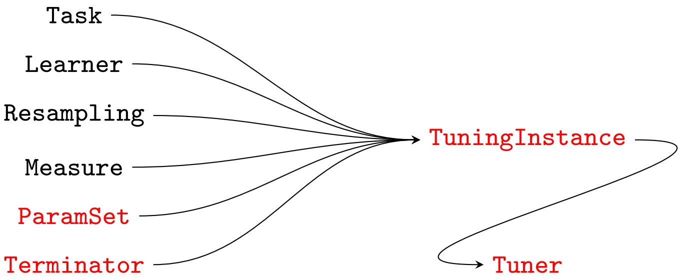
TUNING



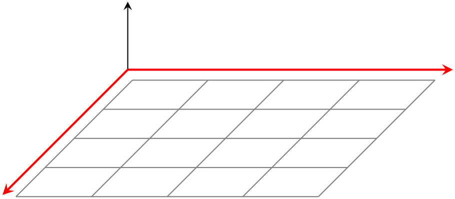
aggregate
Learner (lrn())
Task
Resampling (rsmp())
Measure (msr())
Start
suggest hyperparameters
evaluate performance
search space
Tuner
Terminator (when to finish)
End

Tuning in mlr3

OBJECTS IN TUNING



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
- `Terminator` class
- `mlr_terminators` dictionary, `term()` short form
- ```
  as.data.table(mlr_terminators)

  #>              key
  #> 1:    clock_time
  #> 2:         combo
  #> 3:         evals
  #> 4:    model_time
  #> 5:          none
  #> 6:  perf_reached
  #> 7:    stagnation
  ```
- ```
  term("evals", n_evals = 20)

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

TUNING METHOD

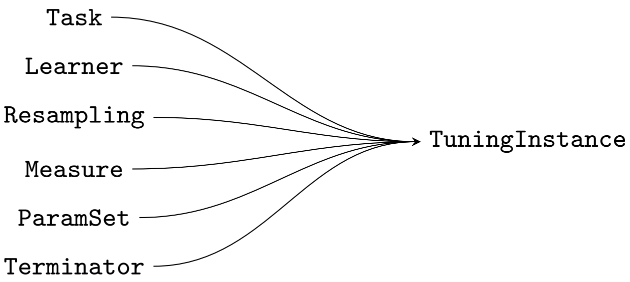
- need to choose a *tuning method*
- `Tuner` class
- `mlr_tuners` dictionary, `tnr()` short form
- ```
  as.data.table(mlr_tuners)
  ```
  ```
  #>               key
  #> 1: design_points
  #> 2:         gensa
  #> 3:   grid_search
  #> 4: random_search
  ```
  ```

TUNING METHOD

- load `Tuner` 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




```r
inst = TuningInstance$new(
  tsk("iris"), lrn("classif.kknn", kernel="rectangular"),
  rsmp("cv"), msr("classif.ce"),
  searchspace_knn, term("none")
)
```

CALLING THE TUNER

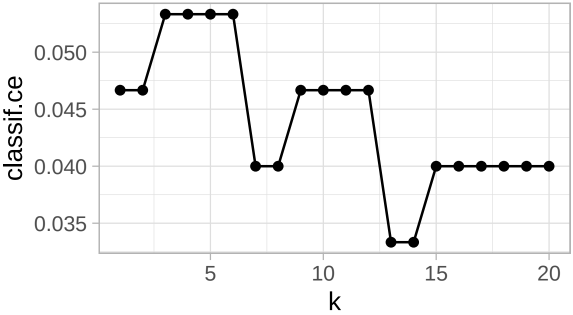
```r
gsearch$tune(inst)
```

```r
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

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



RECAP

```r
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)
```

Parameter Transformation

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`

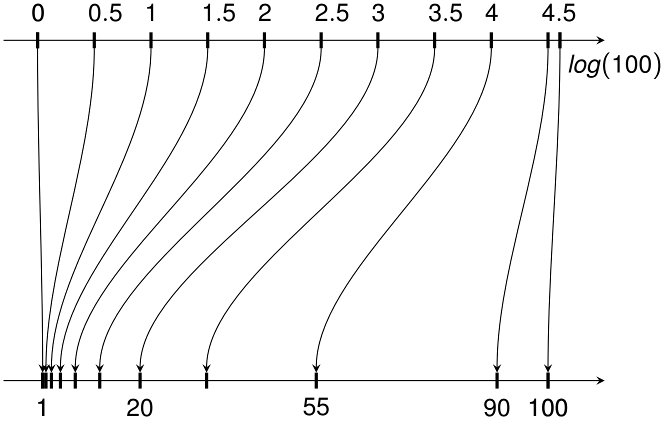
Example:

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

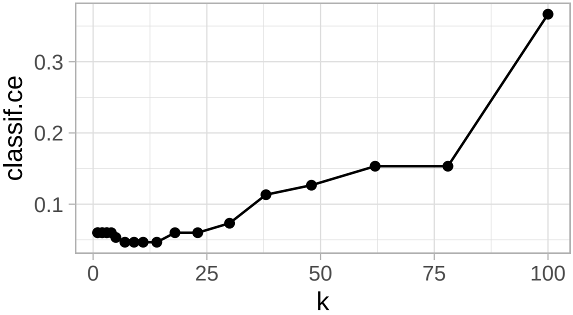
Tuning again...

```
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

```r
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

⇒ 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))
```

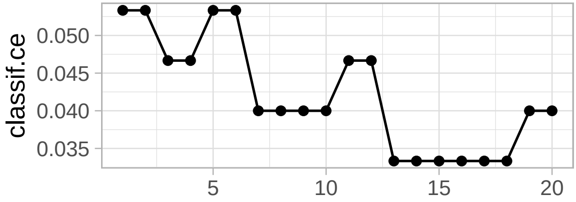
NESTED RESAMPLING

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

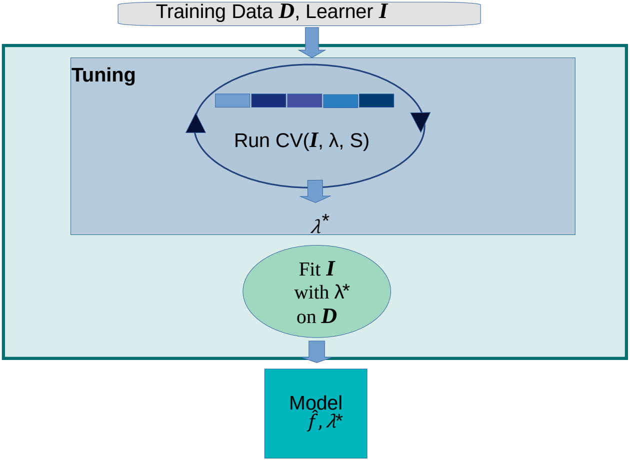
```
optlrn$model$learner

#> <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

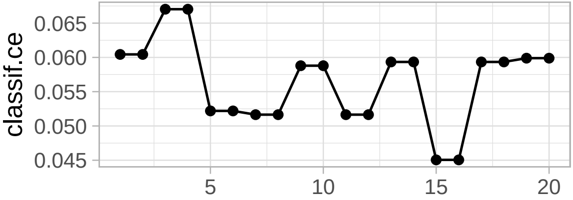
```r
resample(tsk("iris"), optlrn, rsmp("cv"))

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

NESTED RESAMPLING

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

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



Outro

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()`