Introduction to Machine Learning and Efficient Hyper-Parameter Tuning with mlr in R

Bernd Bischl

Joint work:

- Michel Lang (Dortmund, mlr, OpenML)
- Jakob Richter (Dortmund, mlr)
- Lars Kotthoff (Cork, mlr)
- Dominik Kirchhoff (Dortmund, OpenML)
- Julia Schiffner (Duesseldorf, mlr)
- Eric Studerus (Basel, mlr)
- Luis Torgo (Porto, OpenML)

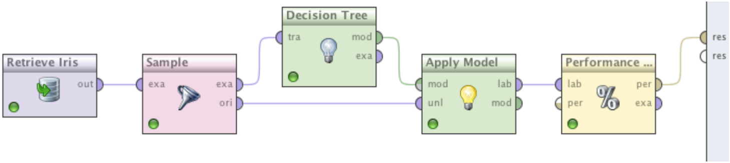
R?

- The lingua franca of statistical computing (and data science?)
- Free and open source
- KDNuggets: Still Nr.1 in
  Top Languages for analytics, data mining, data science
- Packages: ca. 6K on CRAN, ca. 1K on BioConductor
- Rapid prototyping + interfacing
- You can be reasonably fast and work on large data if you know what you are doing
- I have authored or co-developed about 15 (?) packages now over the last 5 years

mlr?

`mlr 2.3`: `https://github.com/berndbischl/mlr`
(also on CRAN)

- Machine learning experiments are well structured
- Definition by plugging operators together (e.g., Weka or RapidMiner):



- No unified interface for machine learning in R!
- Experiments require lengthy, tedious and error-prone code

mlr: abstractions, glue code and some own implementations
Goal: **Get a DSL for ML!**

Task Abstractions

- Regression, (cos-sens.) classification, clustering, survival tasks
- Internally: data frame with annotations: target column(s), weights, misclassification costs, ...)

```
task = makeClassifTask(data = iris, target = "Species")
print(task)

## Supervised task: iris
## Type: classif
## Target: Species
## Observations: 150
## Features:
## numerics  factors  ordered
##        4        0        0
## Missings: FALSE
## Has weights: FALSE
## Has blocking: FALSE
## Classes: 3
##     setosa versicolor  virginica
##         50         50         50
## Positive class: NA
```

Learner Abstractions

- 54 classification, 6 clustering, 45 regression, 10 survival
- Reduction algorithms for cost-sensitive
- Internally: functions to train and predict, parameter set and annotations

```
lrn = makeLearner("classif.rpart")
print(lrn)

## Learner classif.rpart from package rpart
## Type: classif
## Name: Decision Tree; Short name: rpart
## Class: classif.rpart
## Properties: twoclass,multiclass,missings,numerics,factors,ordered,prob,weigh
## Predict-Type: response
## Hyperparameters: xval=0
```

Learner Abstractions

```
getParamSet(lrn)

##                    Type len  Def   Constr Req Tunable
## minsplit         integer   -   20 1 to Inf   -    TRUE
## minbucket        integer   -    - 1 to Inf   -    TRUE
## cp               numeric   - 0.01   0 to 1   -    TRUE
## maxcompete       integer   -    4 0 to Inf   -    TRUE
## maxsurrogate     integer   -    5 0 to Inf   -    TRUE
## usesurrogate    discrete   -    2    0,1,2   -    TRUE
## surrogatestyle  discrete   -    0      0,1   -    TRUE
## maxdepth         integer   -   30  1 to 30   -    TRUE
## xval             integer   -   10 0 to Inf   -    TRUE
## parms            untyped   -    -        -   -   FALSE
##                 Trafo
## minsplit            -
## minbucket           -
## cp                  -
## maxcompete          -
## maxsurrogate        -
## usesurrogate        -
## surrogatestyle      -
## maxdepth            -
## xval                -
## parms               -
```

Performance Measures

- 22 classification, 7 regression, 1 survival, 5 clustering, 4 general
- Internally: performance function, aggregation function and annotations

```
print(mmce)

## Name: Mean misclassification error
## Performance measure: mmce
## Properties: classif,classif.multi,req.pred,req.truth
## Minimize: TRUE
## Best: 0; Worst: 1
## Aggregated by: test.mean
## Note:

print(timetrain)

## Name: Time of fitting the model
## Performance measure: timetrain
## Properties: classif,classif.multi,regr,surv,costsens,cluster,req.model
## Minimize: TRUE
## Best: 0; Worst: Inf
## Aggregated by: test.mean
## Note:
```

Resampling

- Resampling techniques: CV, Bootstrap, Subsampling, ...

```
cv3f = makeResampleDesc("CV", iters = 3, stratify = TRUE)
```

- 10-fold CV of rpart on iris

```
lrn = makeLearner("classif.rpart")
cv10f = makeResampleDesc("CV", iters = 10)
measures = list(mmce, acc)

resample(lrn, task, cv10f, measures)$aggr

## mmce.test.mean  acc.test.mean
##        0.07333        0.92667
```

Benchmarking

- Compare multiple learners on multiple tasks
- Fair comparisons: same training and test sets for each learner

```
task = list(iris.task, sonar.task)

learners = list(
  makeLearner("classif.rpart"),
  makeLearner("classif.randomForest"),
  makeLearner("classif.ksvm")
)

benchmark(learners, tasks, cv10f, mmce)

## Error in isScalarValue(x):  object 'tasks' not found
```

Parallelization

- Activate with `parallelMap::parallelStart`
- Backends: `local`, `multicore`, `socket`, `mpi` and `BatchJobs`

```
parallelStart("BatchJobs")
benchmark([...])
parallelStop()
```

- Parallelization levels

```
parallelShowRegisteredLevels()

## Error in eval(expr, envir, enclos):  could not find function
"parallelShowRegisteredLevels"
```

  Defaults to first possible / most outer loop

- Few iterations in benchmark (loop over `learners` × `tasks`), many in resampling

```
parallelStart("multicore", level = "mlr.resample")
```

Visualizations: Predictions

```
plotLearnerPrediction(makeLearner("classif.randomForest"), task,
  features = c("Sepal.Length", "Sepal.Width"))

## Error in knit2pdf("talk.Rnw"):  Assertion on 'task' failed.  One of
the following must apply:
##  * checkClass:  Must have class 'ClassifTask', but
##  * has class 'list'
##  * checkClass:  Must have class 'RegrTask', but
##  * has class 'list'
##  * checkClass:  Must have class 'ClusterTask', but
##  * has class 'list'
```

Visualizations II : ROC

```r
learners = list(
  makeLearner("classif.rpart", predict.type = "prob"),
  makeLearner("classif.qda", predict.type = "prob")
)
br = benchmark(learners, sonar.task)
plotROCRCurves(br)

## Error in knit2pdf("talk.Rnw"):  Assertion on 'obj' failed:  Must have
class 'ROCRCurvesData', but has classes 'BenchmarkResult','list'
```

Wrapper

Create new learners by wrapping existing ones

- Preprocessing: PCA, normalization (z-transformation)
- Filter: correlation- and entropy-based, $\mathcal{X}^2$-test, mRMR, ...
- Feature Selection: (floating) sequential forward/backward, exhaustive search, genetic algorithms, ...
- Impute: dummy variables, imputations with mean, median, min, max, empirical distribution or other learners
- Bagging to fuse learners on bootstraped samples
- Stacking to combine models in heterogenous ensembles
- Over- and Undersampling for unbalanced classification
- Parameter Tuning: grid, optim, random search, genetic algorithms, CMAES, iRace, MBO

Model Selection Example (0)

- Random search for RBF SVM on a log scale

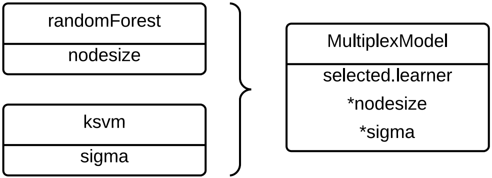
```
lrn = makeLearner("classif.ksvm", kernel = "rbfdot")

rdesc = makeResampleDesc("Holdout")
ctrl = makeTuneControlRandom(maxit = 2L)
tune.ps = makeParamSet(
  makeNumericParam("C", lower = -10, upper = 10,
    trafo = function(x) 2^x),
  makeNumericParam("sigma", lower = -10, upper = 10,
    trafo = function(x) 2^x)
)
tuneParams(lrn, iris.task, rdesc, mmce, tune.ps, ctrl)

## Tune result:
## Op. pars: C=44.3; sigma=0.00937
## mmce.test.mean=0.02
```

Model Selection Example (1)

- Goal: Find "best" model for given task
- Model performance strongly depends on choice of parameters
- Detect inferior models early, don't waste too much time tuning
- Define a multiplex model



- Let a tuner exploit interesting configurations (model + parameters)

Model Selection Example (2)

```r
# create multiplexed learner
lrn = makeModelMultiplexer(list(
  makeLearner("classif.randomForest", ntree = 100),
  makeLearner("classif.ksvm", kernel = "rbfdot")
))

# wrap in tuning
inner = makeResampleDesc("CV", iters = 3L)
ctrl = makeTuneControlIrace(maxExperiments = 200L)
tune.ps = makeModelMultiplexerParamSet(lrn,
  makeIntegerParam("nodesize", lower = 1L, upper = 20L),
  makeNumericParam("sigma", lower = -10, upper = 10,
    trafo = function(x) 2^x)
  )
lrn = makeTuneWrapper(lrn, inner, mmce, tune.ps, ctrl)
```

Model Selection Example (3)

```
task = makeClassifTask(data = Sonar, target = "Class")

## Error in assertDataFrame(data):  object 'Sonar' not found

outer = makeResampleDesc("Subsample", iters = 1)
res = resample(lrn, task, outer, models = TRUE)

## Error in knit2pdf("talk.Rnw"):  Assertion on 'task' failed:  Must
have class 'Task', but has class 'list'

res$models[[1]]

## Error in eval(expr, envir, enclos):  object 'res' not found
```

Tuned multiplexed and prefiltered survival models applied on high-dimensional gene expression data:
M. Lang, H. Kotthaus, P. Marwedel, J. Rahnenführer, B. Bischl. *Automatic model selection for high-dimensional survival analysis.* Journal of Statistical Computation and Simulation (2014)

OpenML-R-Package

Current API:

- Explore data and tasks
- Download data and tasks
- Register learners
- Upload runs

Already nicely connected to mlr!

Explore and Select Data

```
options(width = 80)
authenticateUser() # uses my OML config file

## Authenticating user at server:  bernd_bischl@gmx.net
## Retrieved session hash.  Valid until:  2015-06-15 15:47:08

listOMLDataSets()[1:3, 1:5]

## Downloading 'http://www.openml.org/api/?f=openml.data' to '<mem>'

##   did status        name NumberOfClasses NumberOfFeatures
## 1   1 active      anneal               6               39
## 2   2 active anneal.ORIG               6               39
## 3   3 active    kr-vs-kp               2               37

listOMLTasks()[1:3, 1:5]

## Downloading 'http://www.openml.org/api/?f=openml.tasks&task_type_id=1' to
'<mem>'

##   task_id                 task_type did status        name
## 1       1 Supervised Classification   1 active      anneal
## 2       2 Supervised Classification   2 active anneal.ORIG
## 3       3 Supervised Classification   3 active    kr-vs-kp
```

Download a Data Set

```r
# uses built in caching from disk
getOMLDataSet(5L)

## Getting data set '5' from OpenML repository.
## Downloading
'http://www.openml.org/api/?f=openml.data.description&data.id=5' to
'/tmp/Rtmp7nrBCK/cache/datasets/5/description.xml'
## Downloading
'http://openml.liacs.nl/data/download/5/dataset_5_arrhythmia.arff' to
'/tmp/Rtmp7nrBCK/cache/datasets/5/dataset.arff'

##
## Data Set "arrhythmia" :: (Version = 1, OpenML ID = 5)
##   Default Target Attribute: class
```

Download a Task

```
# uses built in caching from disk
oml.task = getOMLTask(1L)

## Downloading task '1' from OpenML repository.
## Downloading 'http://www.openml.org/api/?f=openml.task.get&task_id=1'
to '/tmp/Rtmp7nrBCK/cache/tasks/1/task.xml'
## Getting data set '1' from OpenML repository.
## Downloading
'http://www.openml.org/api/?f=openml.data.description&data.id=1' to
'/tmp/Rtmp7nrBCK/cache/datasets/1/description.xml'
## Downloading
'http://openml.liacs.nl/data/download/1/dataset_1_anneal.arff' to
'/tmp/Rtmp7nrBCK/cache/datasets/1/dataset.arff'
## Downloading
'http://www.openml.org/api_splits/get/1/Task_1_splits.arff' to
'/tmp/Rtmp7nrBCK/cache/tasks/1/datasplits.arff'

print(oml.task)

##
## OpenML Task 1 :: (Data ID = 1)
##   Task Type            : Supervised Classification
##   Data Set             : anneal :: (Version = 2, OpenML ID = 1)
##   Target Feature(s)    : class
##   Estimation Procedure : Stratified crossvalidation (1 x 10 folds)
```

Running a Task

```
lrn = makeLearner("classif.rpart")
res =  runTaskMlr(oml.task, lrn)

## Warning in fixupData.Task(task, target, fixup.data):  Empty factor
levels were dropped for columns:
family,product.type,steel,condition,formability,surface.finish,enamelability,m,
## Removing 7 columns:  product.type,m,marvi,corr,jurofm,s,p
## [Resample] cross-validation iter:  1
## [Resample] cross-validation iter:  2
## [Resample] cross-validation iter:  3
## [Resample] cross-validation iter:  4
## [Resample] cross-validation iter:  5
## [Resample] cross-validation iter:  6
## [Resample] cross-validation iter:  7
## [Resample] cross-validation iter:  8
## [Resample] cross-validation iter:  9
## [Resample] cross-validation iter:  10
## [Resample] Result:  acc.test.mean=0.977
## Downloading
'http://www.openml.org/api/?f=openml.implementation.exists&name=classif.rpart&e
to '<mem>'
## Downloading
'http://www.openml.org/api/?f=openml.implementation.exists&name=classif.rpart&e
to '<mem>'
```

Uploading Learner and Predictions

```
hash = authenticateUser("your@email.com", "your_password")
impl = createOpenMLImplementationForMlrLearner(lrn)
uploadOpenMLImplementation(impl, session.hash = hash)
uploadOpenMLRun(oml.task, lrn, impl, pred, hash)
```

Automatic Model Selection

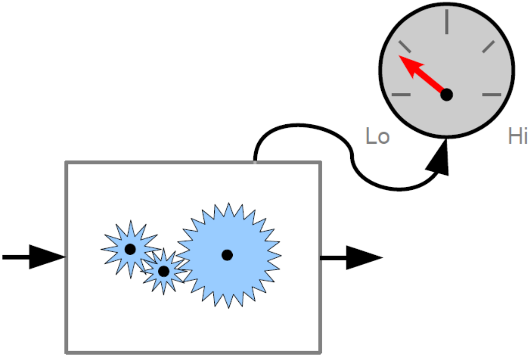
Prior approaches:

- Looking for the silver bullet model
  $\rightsquigarrow$ Failure
- Exhaustive benchmarking / search
  $\rightsquigarrow$ Per data set: too expensive
  $\rightsquigarrow$ Over many: contradicting results
- Meta-Learning:
  $\rightsquigarrow$ Failure
  $\rightsquigarrow$ Usually not for preprocessing / hyperparamters

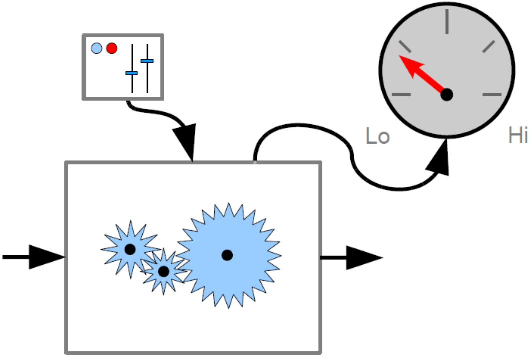
Goal:

- Data dependent
- Automatic
- Efficient

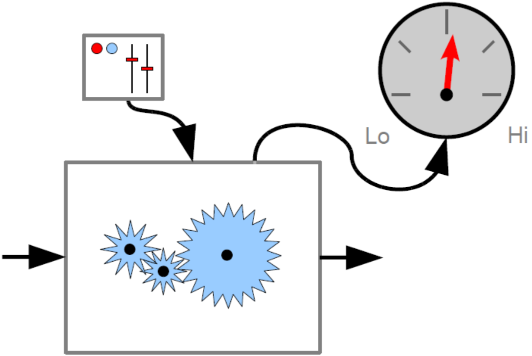
Black-Box-Perspective in Configuration



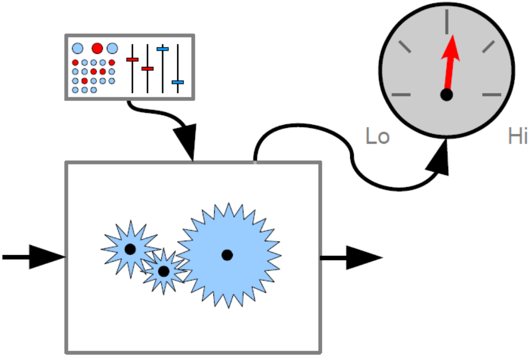
Black-Box-Perspective in Configuration



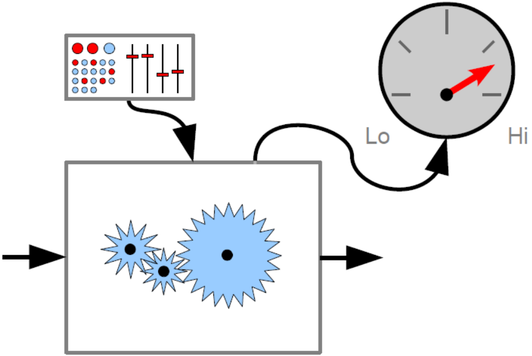
Black-Box-Perspective in Configuration



Black-Box-Perspective in Configuration



Black-Box-Perspective in Configuration

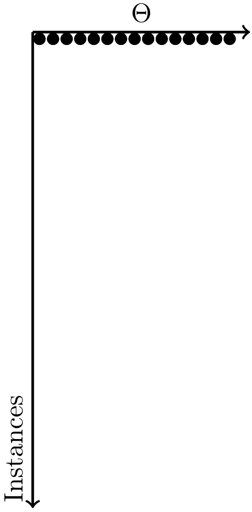


General Algorithm Configuration

- Assume a (parametrized) algorithm a
- Parameter space $\theta \in \Theta$
  might be discrete and dependent / hierarchical
- Stochastic generating process for instances $i \sim P$, where we draw i.i.d. from. (Usually predefined set of instances, and i.i.d.-ness somewhat violated)
- Run algorithm a on i and measure performance
  $f(i, \theta) = run(i, a(\theta))$
- Objective: $\min_{\theta \in \Theta} E_P[f(i, \theta)]$
- No derivative for $f(\cdot, \theta)$, black-box
- f is stochastic / noisy
- f is likely expensive to evaluate
- Consequence: very hard problem

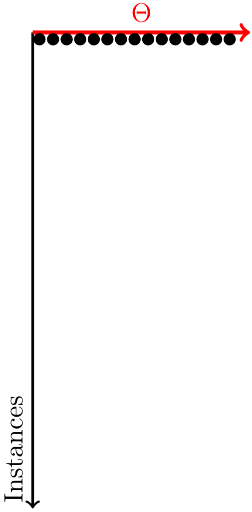
$\rightsquigarrow$ Usual approaches: racing or model-based / bayesian optimization

Idea of (F-)Racing



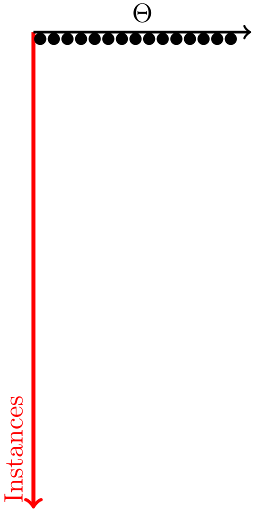
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed candidates
- Output: Remaining candidates
- Yes, the testing completely ignores "sequentiality" and is somewhat heuristic.

Idea of (F-)Racing



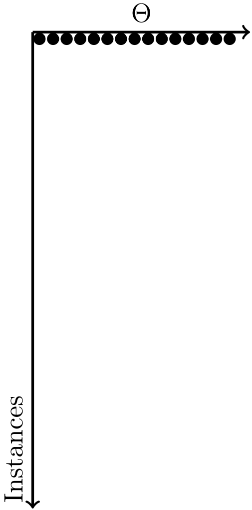
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed candidates
- Output: Remaining candidates
- Yes, the testing completely ignores "sequentiality" and is somewhat heuristic.

Idea of (F-)Racing



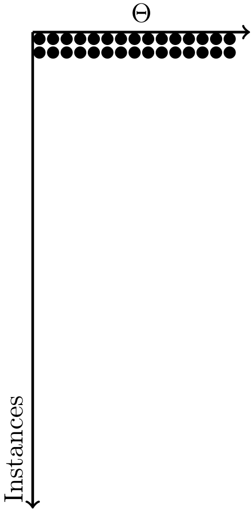
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One “generation”
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed candidates
- Output: Remaining candidates
- Yes, the testing completely ignores “sequentiality” and is somewhat heuristic.

Idea of (F-)Racing



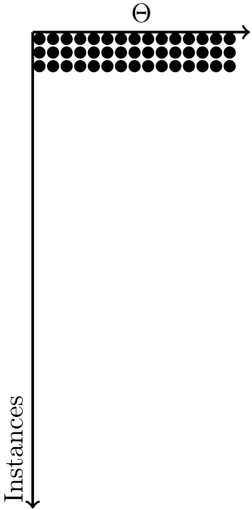
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed candidates
- Output: Remaining candidates
- Yes, the testing completely ignores "sequentiality" and is somewhat heuristic.

Idea of (F-)Racing



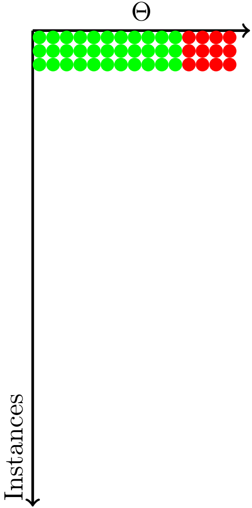
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed candidates
- Output: Remaining candidates
- Yes, the testing completely ignores "sequentiality" and is somewhat heuristic.

Idea of (F-)Racing



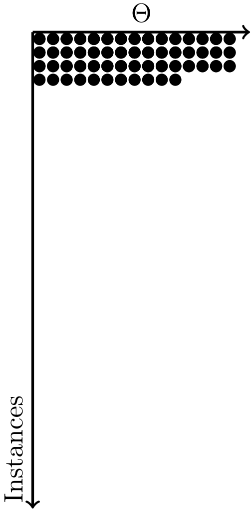
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed candidates
- Output: Remaining candidates
- Yes, the testing completely ignores "sequentiality" and is somewhat heuristic.

Idea of (F-)Racing



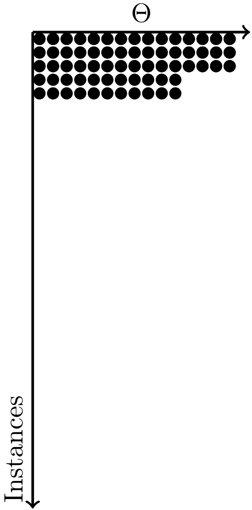
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed candidates
- Output: Remaining candidates
- Yes, the testing completely ignores "sequentiality" and is somewhat heuristic.

Idea of (F-)Racing



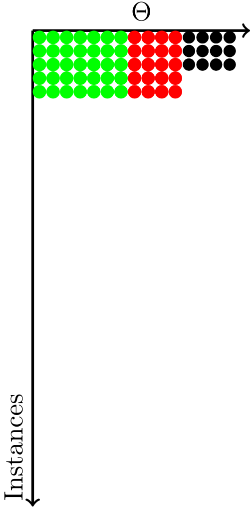
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed candidates
- Output: Remaining candidates
- Yes, the testing completely ignores "sequentiality" and is somewhat heuristic.

Idea of (F-)Racing



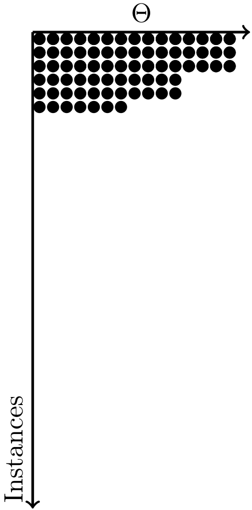
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed candidates
- Output: Remaining candidates
- Yes, the testing completely ignores "sequentiality" and is somewhat heuristic.

Idea of (F-)Racing



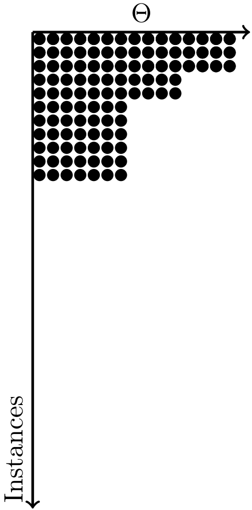
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed candidates
- Output: Remaining candidates
- Yes, the testing completely ignores "sequentiality" and is somewhat heuristic.

Idea of (F-)Racing



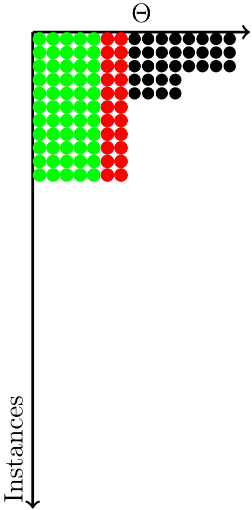
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed candidates
- Output: Remaining candidates
- Yes, the testing completely ignores "sequentiality" and is somewhat heuristic.

Idea of (F-)Racing



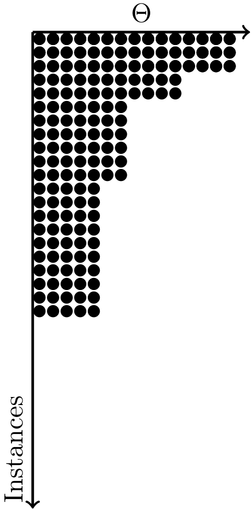
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, . . .
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed candidates
- Output: Remaining candidates
- Yes, the testing completely ignores "sequentiality" and is somewhat heuristic.

Idea of (F-)Racing



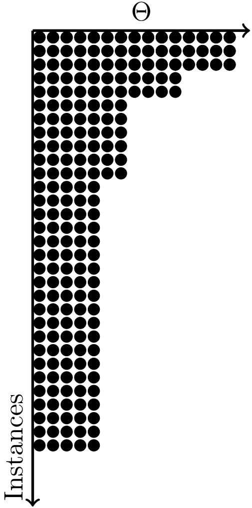
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, . . .
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed candidates
- Output: Remaining candidates
- Yes, the testing completely ignores "sequentiality" and is somewhat heuristic.

Idea of (F-)Racing



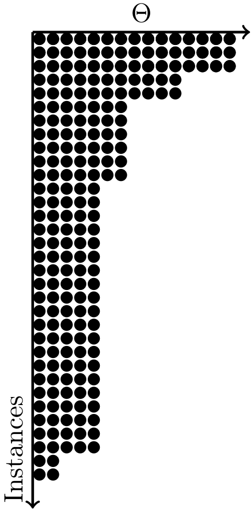
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed candidates
- Output: Remaining candidates
- Yes, the testing completely ignores "sequentiality" and is somewhat heuristic.

Idea of (F-)Racing



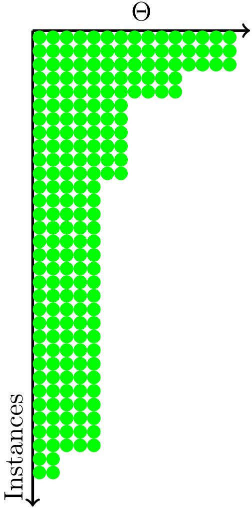
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed candidates
- Output: Remaining candidates
- Yes, the testing completely ignores "sequentiality" and is somewhat heuristic.

Idea of (F-)Racing



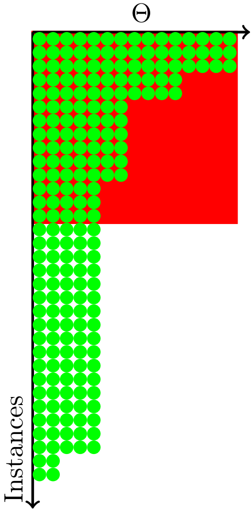
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed candidates
- Output: Remaining candidates
- Yes, the testing completely ignores "sequentiality" and is somewhat heuristic.

Idea of (F-)Racing



- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed candidates
- Output: Remaining candidates
- Yes, the testing completely ignores "sequentiality" and is somewhat heuristic.

Idea of (F-)Racing



- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed candidates
- Output: Remaining candidates
- Yes, the testing completely ignores "sequentiality" and is somewhat heuristic.

Idea of Iterated F-Racing

Why doesn't normal Racing work very often?
Because we might have many of even an infinite number of candidates

- Have a stochastic model to draw candidates from in every generation
- For each parameter: Univariate, independent distribution (factorized joint distribution)
- Sample distributions centered at "elite" candidates from previous generation(s)
- Reduce distributions' width / variance in later generations for convergence

Idea of Iterated F-Racing

Whats good about this

- Very simple and generic algorithm
- Can easily be parallelized
- A nice R package exists: irace[1]

What might be not so good

- Quite strong (wrong?) assumptions in the probability model
- Sequential model-based optimization is probably more efficient
  (But be careful: Somewhat my personal experience and bias, as not so many large scale comparisons exist)

[1] Lopez-Ibanez et al, "The irace package, Iterated Race for Automatic Algorithm Configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université libre de Bruxelles, Belgium, 2011."

Sequential model-based optimization

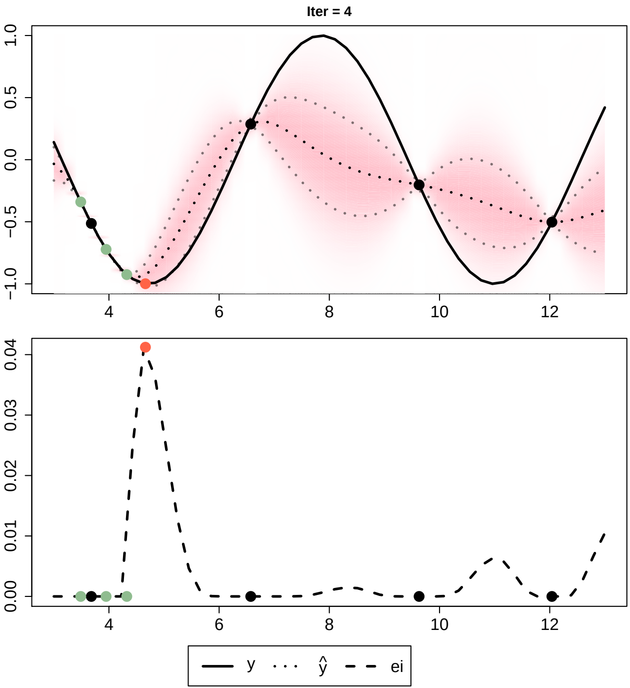
- Let's focus on a simpler problem for now
- Setting: Expensive black-box poblem $f : x \to \mathbb{R} = min!$
- Classical problem: Computer simulation with a bunch of control parameters and performance output
- Idea: Let's approximate f via regression!

Generic MBO Pseudo Code

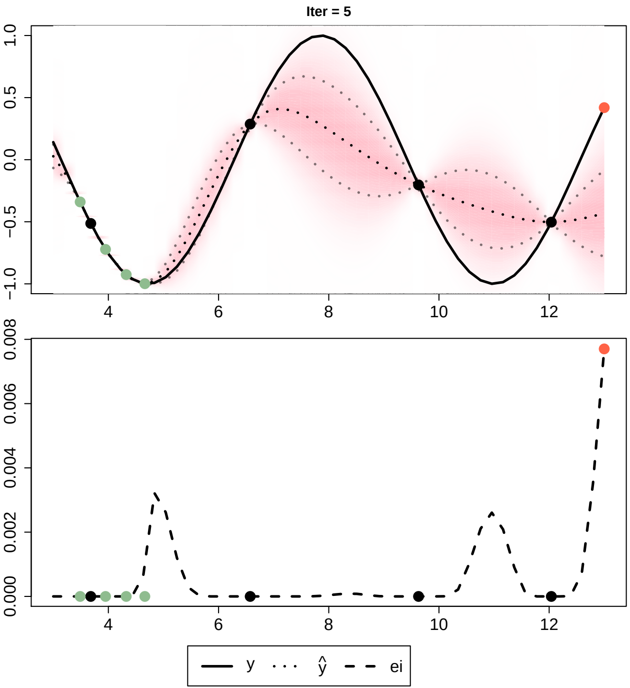
- Create initial space filling design and evaluate with f
- In each iteration:
  - Fit regression model
  - Propose point via infill criterion, e.g., expected improvement

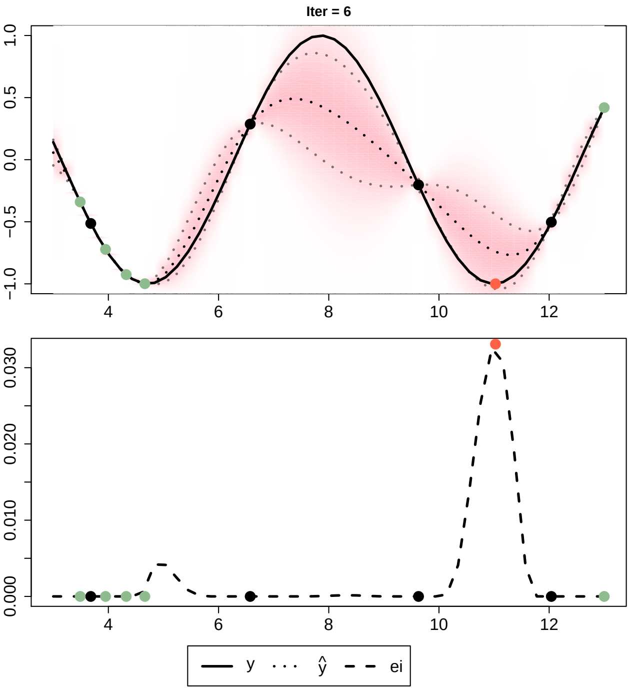
$$\mathrm{EI}(x) \uparrow \iff \hat{y} \downarrow \wedge \widehat{\mathrm{se}}\,(\hat{y}) \uparrow$$

  - Evaluate proposed point
  - Add to design



Iter = 4
1.0
0.5
0.0
−0.5
−1.0
4
6
8
10
12
0.04
0.03
0.02
0.01
0.00
4
6
8
10
12
y
ŷ
ei

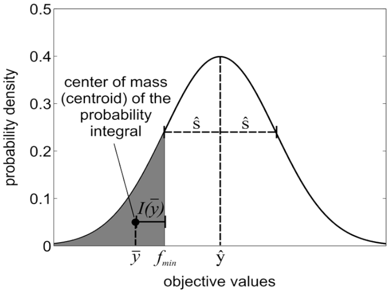

Iter = 5
1.0
0.5
0.0
−0.5
−1.0
4
6
8
10
12
0.008
0.006
0.004
0.002
0.000
4
6
8
10
12
y
ŷ
ei

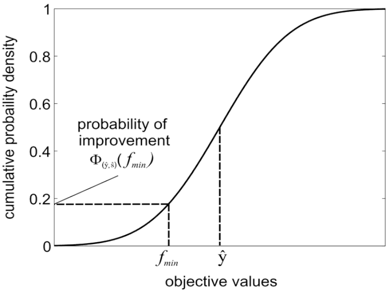

Iter = 6
1.0
0.5
0.0
−0.5
−1.0
4
6
8
10
12
0.030
0.020
0.010
0.000
4
6
8
10
12
y
ŷ
ei

Infill Criterion - Expected improvement

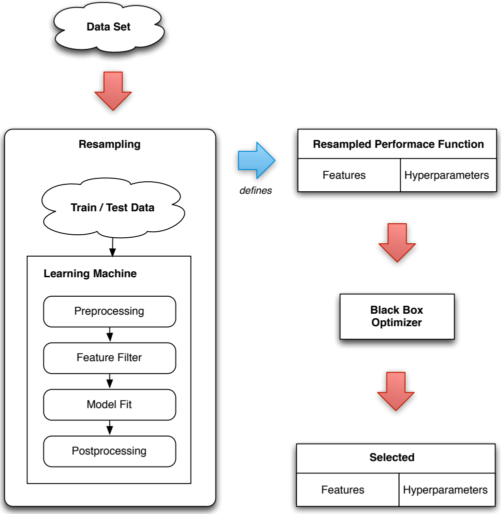
- Define improvement at x over best visited point with $y = y_{min}$ as random variable $I(\boldsymbol{x}) = |y_{min} - y(\boldsymbol{x})|^+$
- For kriging $y(\boldsymbol{x}) \sim N(\hat{y}(\boldsymbol{x}), \hat{s}^2(\boldsymbol{x}))$ (given x and observed data)
- Now define $EI(\boldsymbol{x})$ simply as conditional expectation
- Expectation is integral over normal density starting at y_{min}

Result: $EI(x) = (y_{min} - \hat{y}(\boldsymbol{x}))\,\Phi\left(\frac{y_{min} - \hat{y}(\boldsymbol{x})}{\hat{s}(\boldsymbol{x})}\right) + \hat{s}(\boldsymbol{x})\phi\left(\frac{y_{min} - \hat{y}(\boldsymbol{x})}{\hat{s}(\boldsymbol{x})}\right)$





Model selection in Machine Learning



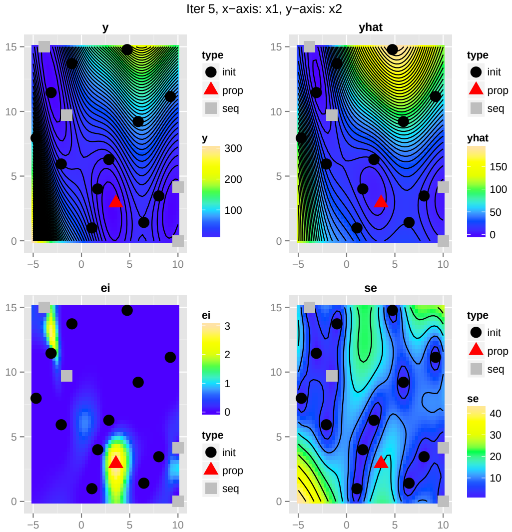
$\leadsto$ Minimal risk principle (2nd level inference)

From Normal MBO to Hyperarameter Tuning

- Instances are resampling training / test splits
- Discrete choices like *which method to apply* become categorical parameters
- Chain mlr operations (e.g. feature filter + ML model) so we can jointly optimize complex systems
- For discrete parameters we can either use special GP kernels or random forests
- Dependent parameters can be handled via special kernels or imputation
- In the future: Estimate and respect resource requirements to improve efficiency

mlrMBO: Model-Based / Bayesian Optimization Toolbox

- Any Regression
- Arbitrary Infill
- Single - or multi-crit
- Parallel
- Algorithm-Configuration
- Active Research



`https://github.com/berndbischl/mlrMBO`

Summary: Why is This Useful?

- Expensive optimization problem, e.g. parameter optimization of an expensive simulator
- For efficient model selection in ML, especially on Big Data
- General algorithm configuration, e.g., solvers for discrete optimization problems
- Multicrit is possible too, we did this e.g. for SVMs on large data

Selected Publications

- M. Lang, H. Kotthaus, P. Marwedel, C. Weihs, J. Rahnenführer, and B. Bischl. **Automatic model selection for high-dimensional survival analysis**. Journal of Statistical Computation and Simulation, 85(1):62–76, 2015.
- P. Koch, B. Bischl, O. Flasch, T. Bartz-Beielstein, C. Weihs, and W. Konen: **Tuning and evolution of support vector kernels**. Evolutionary Intelligence, 5(3):153–170, 2012.
- D. Horn, T. Wagner, D. Biermann, C. Weihs, and B. Bischl: **Model-based multi-objective optimization: Taxonomy, multi-point proposal, toolbox and benchmark**. In Evolutionary Multi-Criterion Optimization (EMO), Lecture Notes in Computer Science, 2015.
- B. Bischl, S. Wessing, N. Bauer, K. Friedrichs, and C. Weihs: **MOI-MBO: Multiobjective infill for parallel model-based optimization**. In Learning and Intelligent Optimization Conference (LION), 2014.
- B. Bischl, J. Schiffner, and C. Weihs: **Benchmarking classification algorithms on high-performance computing clusters**. Studies in Data Analysis, Machine Learning and Knowledge Discovery.

The End...

- Probably: I am overtime already, as always. Sorry....
- Still: I left out so many interesting details w.r.t. to details
  Talk to me if you are interested!

Kriging and local uncertainty prediction

Model: Zero-mean GP with const. trend and cov. kernel $k_\theta(x_1, x_2)$.

- $\boldsymbol{y} = (y_1, \ldots, y_n)^T$, $\boldsymbol{K} = (k(\boldsymbol{x}_i, \boldsymbol{x}_j))_{i,j=1,\ldots,n}$
- $\boldsymbol{k}_*(\boldsymbol{x}) = (k(\boldsymbol{x}_1, \boldsymbol{x}), \ldots, k(\boldsymbol{x}_n, \boldsymbol{x}))^T$
- $\hat{\mu} = \mathbf{1}^T \boldsymbol{K}^{-1} \boldsymbol{y} / \mathbf{1}^T \boldsymbol{K}^{-1} \mathbf{1}$ (BLUE)
- Prediction: $\hat{y}(x) = E[Y(x)|Y(x_i) = y_i, i = 1, \ldots, n] = \hat{\mu} + \mathbf{k}_n(x)^T K^{-1}(\mathbf{y} - \hat{\mu}\mathbf{1})$
- Prediction: $\hat{y}(\boldsymbol{x}) = \hat{\mu} + \boldsymbol{k}_*(\boldsymbol{x})^T \boldsymbol{K}^{-1}(\boldsymbol{y} - \hat{\mu}\mathbf{1})$
- Uncertainty: $s^2(x) = Var[Y(x)|Y(x_i) = y_i, i = 1, \ldots, n] = \sigma^2 - \mathbf{k}_n^T(x) K^{-1} \mathbf{k}_n(x) + \frac{(1 - \mathbf{1}^T K^{-1} \mathbf{k}_n^T(x))^2}{\mathbf{1}^T K^{-1} \mathbf{1}}$