Machine Learning in R: Package `mlr`

Bernd Bischl
Computational Statistics, LMU

Welcome!

- Project home page

  `https://github.com/mlr-org/mlr`

  - **Tutorial** for online viewing / download, including many examples
  - R documentation rendered in HTML
  - If you are interested you can ask questions in the github issue tracker
- 8-10 main developers, quite a few contributors, 3 GSOC projects in 2015 and one coming in 2016
- About 20K lines of code, 8K lines of unit tests

Section 1

Part: mlr basics

What is (supervised) machine learning?

- Learning structure in data:
  Classification, regression, survival analysis, clustering, ...
- The art of predicting stuff
- Model optimization
- Understanding of grey-box models

Disclaimer

- The list is subjective and naively tailored to this talk
- ML is based on math and statistics, we will (mainly) talk about structure, software, and practical issues here

Motivation

The good news

- CRAN serves hundreds of packages for machine learning
- Often compliant to the unwritten interface definition:

```
> model = fit(target ~ ., data = train.data, ...)
> predictions = predict(model, newdata = test.data, ...)
```

The bad news

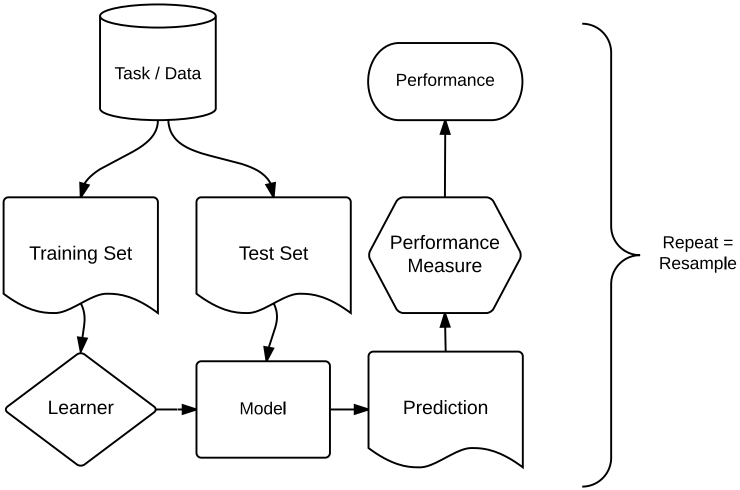
- Some packages API is "just different"
- Functionality is always package or model-dependent, even though the procedure might be general
- No meta-information available or buried in docs

Our goal: A domain-specific language for many machine learning concepts!

Motivation: `mlr`

- Unified interface for the basic building blocks: tasks, learners, resampling, hyperparameters, ...
- Reflections: nearly all objects are queryable (i.e. you can ask them for their properties and program on them)
- The OO-structure allows many generic algorithms:
  - Bagging
  - Stacking
  - Feature Selection
  - ...
- Easily extensible via S3
  - Extension is not covered here, but explained in detail in the online tutorial
  - You do not need to understand S3 to use `mlr`
  - Wondering why we don't use S4? We care about code bloat and speed.

Building Blocks



- `mlr` objects: tasks, learners, measures, resampling instances.

Task Abstraction

- Tasks encapsulate data and meta-information about it
- Regression, classification, clustering, survival tasks
- Data is stored inside an environment to save memory

```
> 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 Abstraction

- Internal structure of learners:
  - wrappers around `fit()` and `predict()` of the package
  - description of the parameter set
  - annotations
- Naming convention: `<tasktype>.<functionname>`
  e.g.: classif.svm, regr.lm
- Adding custom learners is covered in the tutorial

```
> lrn = makeLearner("classif.svm", predict.type = "prob", kernel = "linear", cost = 1)
> print(lrn)

## Learner classif.svm from package e1071
## Type: classif
## Name: Support Vector Machines (libsvm); Short name: svm
## Class: classif.svm
## Properties: twoclass,multiclass,numerics,factors,prob,class.weights
## Predict-Type: prob
## Hyperparameters: kernel=linear,cost=1
```

What Learners are available? I

Classification (72)

- LDA, QDA, RDA, MDA
- Trees and forests
- Boosting (different variants)
- SVMs (different variants)
- ...

Regression (52)

- Linear, lasso and ridge
- Boosting
- Trees and forests
- Gaussian processes
- ...

Clustering (8)

- K-Means
- EM
- DBscan
- X-Means
- ...

Survival (11)

- Cox-PH
- Cox-Boost
- Random survival forest
- Penalized regression
- ...

We can explore them on the webpage – or ask `mlr`

What Learners are available? II

```
> # list all classification learners which can predict probabilities
> # and allow multiclass classification
> listLearners("classif",
+   properties = c("prob", "multiclass"))[1:5, c(-2, -5, -16)]

##              class short.name package    type installed numerics
## 1  classif.avNNet     avNNet    nnet classif      TRUE     TRUE
## 2 classif.cforest    cforest   party classif      TRUE     TRUE
## 3   classif.ctree      ctree   party classif      TRUE     TRUE
## 4     classif.gbm        gbm     gbm classif      TRUE     TRUE
## 5     classif.IBk        ibk   RWeka classif      TRUE     TRUE
##   factors ordered missings weights prob oneclass twoclass
## 1    TRUE   FALSE    FALSE    TRUE TRUE    FALSE     TRUE
## 2    TRUE    TRUE     TRUE    TRUE TRUE    FALSE     TRUE
## 3    TRUE    TRUE     TRUE    TRUE TRUE    FALSE     TRUE
## 4    TRUE   FALSE     TRUE    TRUE TRUE    FALSE     TRUE
## 5    TRUE   FALSE    FALSE   FALSE TRUE    FALSE     TRUE
##   class.weights    se lcens rcens icens
## 1         FALSE FALSE FALSE FALSE FALSE
## 2         FALSE FALSE FALSE FALSE FALSE
## 3         FALSE FALSE FALSE FALSE FALSE
## 4         FALSE FALSE FALSE FALSE FALSE
## 5         FALSE FALSE FALSE FALSE FALSE
```

Parameter Abstraction

- Extensive meta-information for hyperparameters available: storage type, constraints, defaults, dependencies
- Automatically checked for feasibility
- You can program on parameters!

```
> getParamSet(lrn)

##                        Type  len              Def                               Constr Req Tunable Trafo
## type                discrete    - C-classification C-classification,nu-classification   -    TRUE     -
## cost                 numeric    -                1                            0 to Inf   Y    TRUE     -
## nu                   numeric    -              0.5                         -Inf to Inf   Y    TRUE     -
## class.weights numericvector <NA>                -                            0 to Inf   -    TRUE     -
## kernel              discrete    -           radial   linear,polynomial,radial,sigmoid   -    TRUE     -
## degree               integer    -                3                            1 to Inf   Y    TRUE     -
## coef0                numeric    -                0                         -Inf to Inf   Y    TRUE     -
## gamma                numeric    -                -                            0 to Inf   Y    TRUE     -
## cachesize            numeric    -               40                         -Inf to Inf   -    TRUE     -
## tolerance            numeric    -            0.001                            0 to Inf   -    TRUE     -
## shrinking            logical    -             TRUE                                   -   -    TRUE     -
## cross                integer    -                0                            0 to Inf   -   FALSE     -
## fitted               logical    -             TRUE                                   -   -   FALSE     -
## scale          logicalvector <NA>             TRUE                                   -   -    TRUE     -
```

Performance Measures

- Performance measures evaluate the predictions a test set and aggregate them over multiple in resampling iterations
- 22 classification, 10 regression, 5 cluster, 1 survival
- Internally: performance and aggregation function, annotations
- Adding custom measures is covered in the tutorial

```
> 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:

> listMeasures("classif")[1:12]

##  [1] "timepredict" "gmean"       "acc"         "auc"
##  [5] "ber"         "fn"          "fp"          "fnr"
##  [9] "gpr"         "featperc"    "ppv"         "fpr"
```

Resampling Abstraction I

- Procedure: Train, Predict, Eval, Repeat.
- Aim: Estimate expected model performance.
  - Hold-Out
  - Cross-validation (normal, repeated)
  - Bootstrap (OOB, B632, B632+)
  - Subsampling
  - Stratification
  - Blocking
- Instantiate it or not (= create data split indices)

```
> rdesc = makeResampleDesc("CV", iters = 3)
> rin = makeResampleInstance(rdesc, task = task)
> str(rin$train.inds)

## List of 3
##  $ : int [1:100] 43 73 67 14 11 84 120 80 19 89 ...
##  $ : int [1:100] 43 27 81 95 11 131 84 58 80 89 ...
##  $ : int [1:100] 73 67 27 14 81 95 131 58 120 19 ...
```

Resampling Abstraction II

Resampling a learner

- Measures on test (or train) sets
- Returns aggregated values, predictions and some useful extra information

```
> lrn = makeLearner("classif.rpart")
> rdesc = makeResampleDesc("CV", iters = 3)
> measures = list(mmce, timetrain)
> r = resample(lrn, task, rdesc, measures = measures)
```

- For the lazy

```
> r = crossval(lrn, task, iters = 3, measures = measures)
```

Resampling Abstraction III

```
> print(r)

## Resample Result
## Task: iris
## Learner: classif.rpart
## mmce.aggr: 0.07
## mmce.mean: 0.07
## mmce.sd: 0.04
## timetrain.aggr: 0.01
## timetrain.mean: 0.01
## timetrain.sd: 0.00
## Runtime: 0.152449
```

Container object: Measures (aggregated and for each test set), predictions, models, ...

Section 2

Benchmarking and Model Comparison

Benchmarking and Model Comparison I

Benchmarking

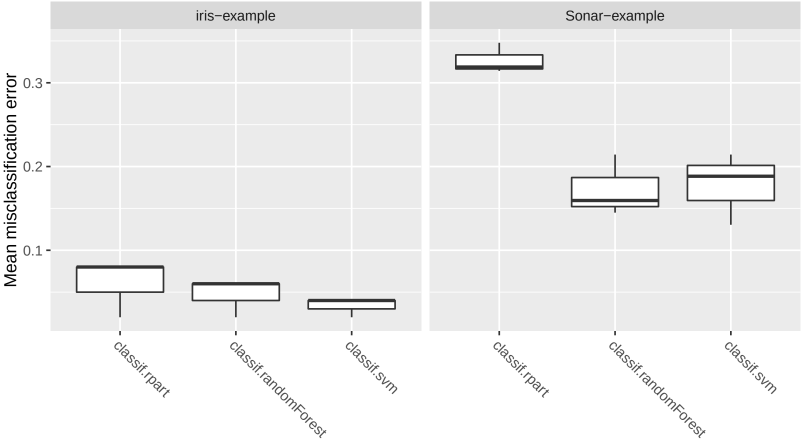
- Comparison of multiple models on multiple data sets
- Aim: Find best learners for a data set or domain, learn about learner characteristics, ...

```
> # these are predefined in mlr for toying around:
> tasks = list(iris.task, sonar.task)
> learners = list(
+   makeLearner("classif.rpart"),
+   makeLearner("classif.randomForest", ntree = 500),
+   makeLearner("classif.svm")
+ )
>
> rdesc = makeResampleDesc("CV", iters = 3)
> br = benchmark(learners, tasks, rdesc)
```

Container object: Results, individual predictions, ...

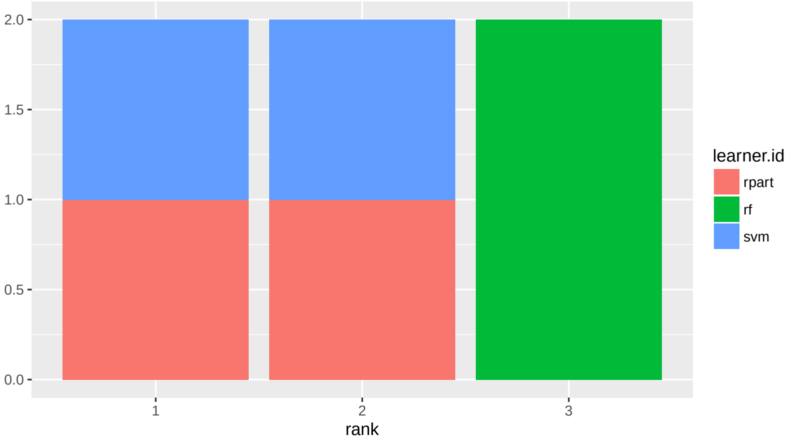
Benchmarking and Model Comparison II

```
> plotBMRBoxplots(br)
```



Benchmarking and Model Comparison III

```
> plotBMRRanksAsBarChart(br)
```



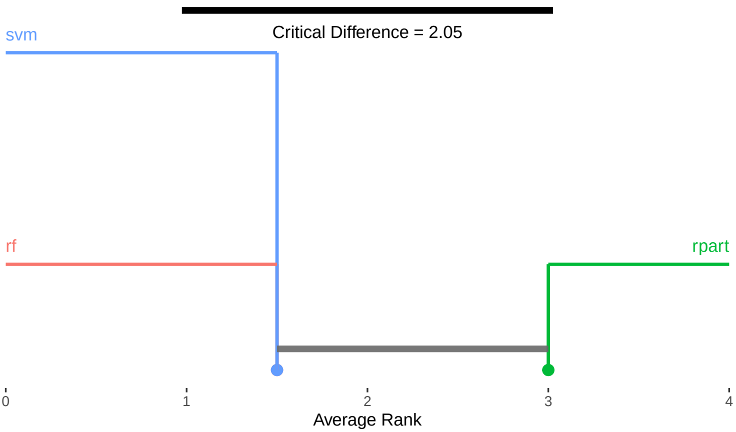
Benchmarking and Model Comparison IV

```
> g = generateCritDifferencesData(br, p.value = 0.1, test = "nemenyi")

## Loading required package:  PMCMR

> plotCritDifferences(g)
```

Benchmarking and Model Comparison V



Section 3

Hyperparameter Tuning and Model Selection

Hyperparameter Tuning

Tuning

- Used to find "best" hyperparameters for a method in a data-dependent way
- General procedure: Tuner proposes param point, eval by resampling, feedback value to tuner

Grid search

- Basic method: Exhaustively try all combinations of finite grid
  $\rightsquigarrow$ Inefficient, combinatorial explosion, searches irrelevant areas

Random search

- Randomly draw parameters
  $\rightsquigarrow$ Scales better then grid search, easily extensible

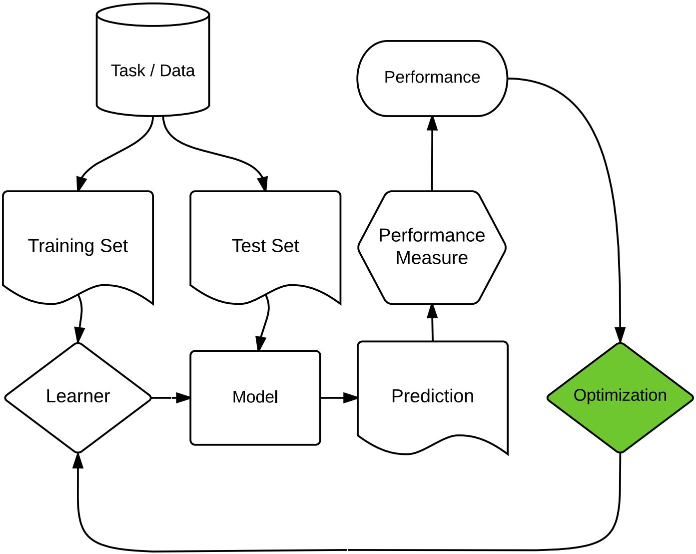
Automatic Model Selection

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

Adaptive tuning

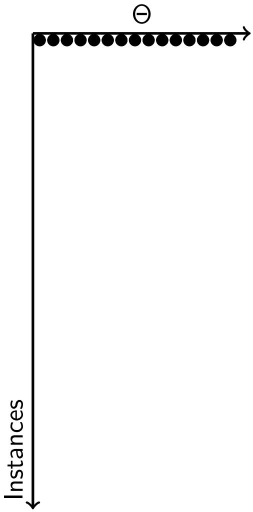


General Algorithm Configuration

- 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.
- 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**

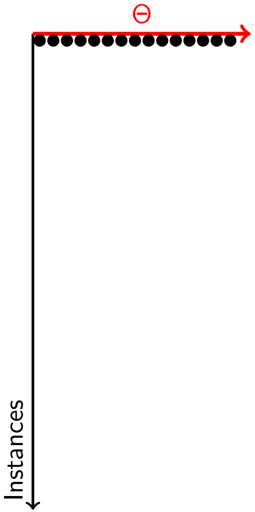
$\leadsto$ Racing or model-based / bayesian optimization

Idea of (F-)Racing



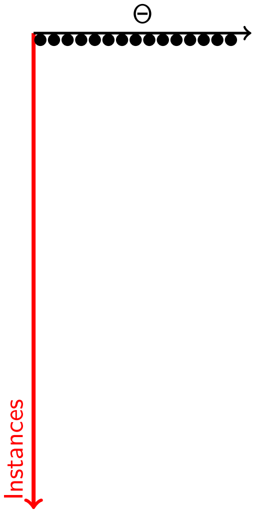
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed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

Idea of (F-)Racing



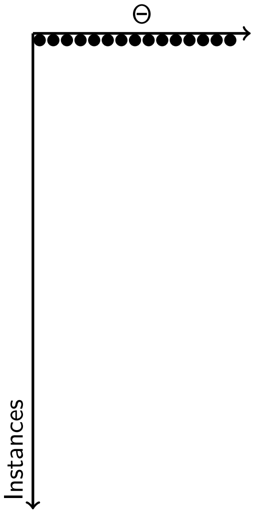
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed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

Idea of (F-)Racing



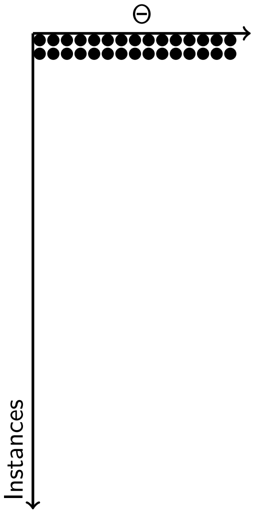
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, . . .
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed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

Idea of (F-)Racing



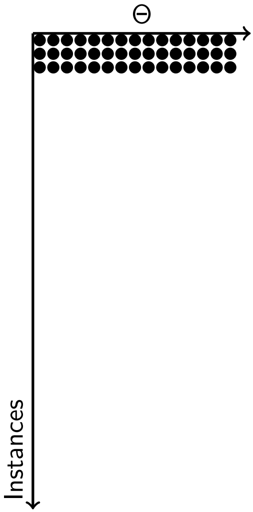
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed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

Idea of (F-)Racing



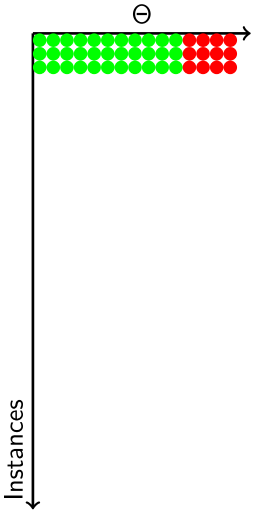
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed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

Idea of (F-)Racing



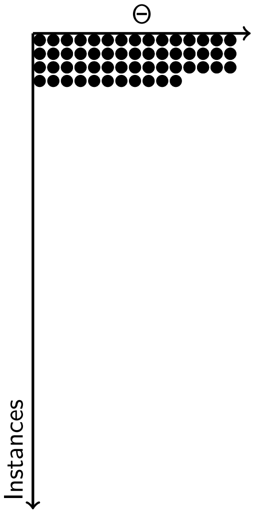
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed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

Idea of (F-)Racing



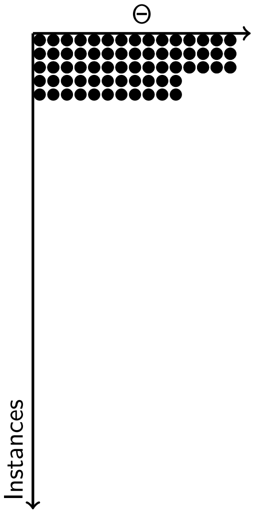
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, . . .
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed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

Idea of (F-)Racing



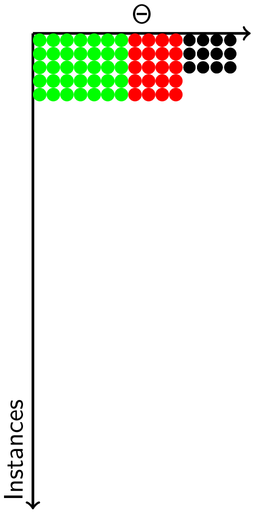
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed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

Idea of (F-)Racing



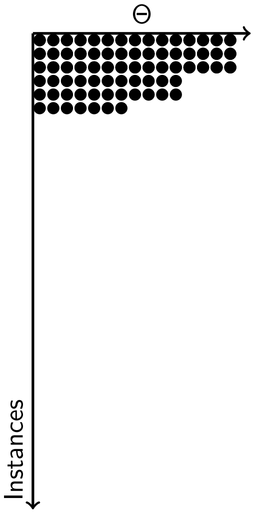
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, . . .
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed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

Idea of (F-)Racing



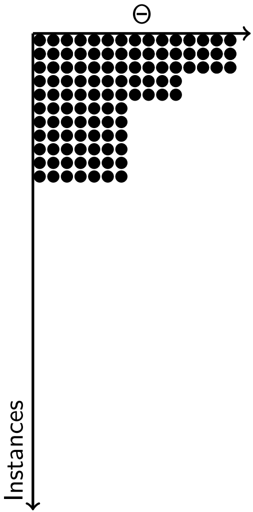
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed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

Idea of (F-)Racing



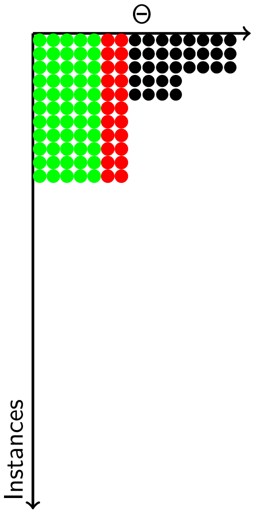
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed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

Idea of (F-)Racing



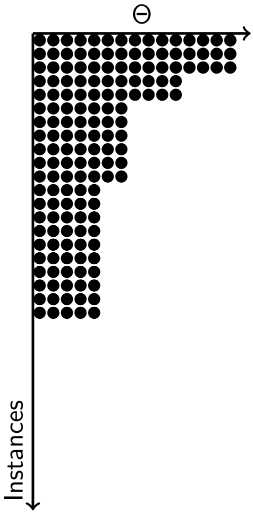
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One “generation”
  - Evaluate all candidates on an instance, and another, . . .
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed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

Idea of (F-)Racing



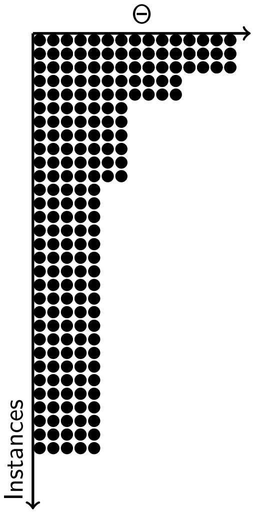
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed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

Idea of (F-)Racing



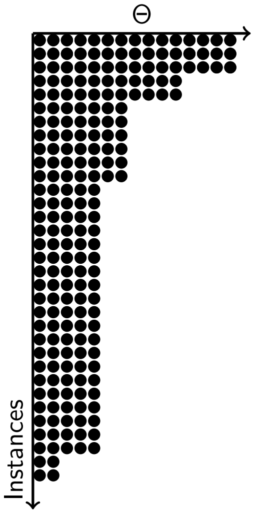
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed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

Idea of (F-)Racing



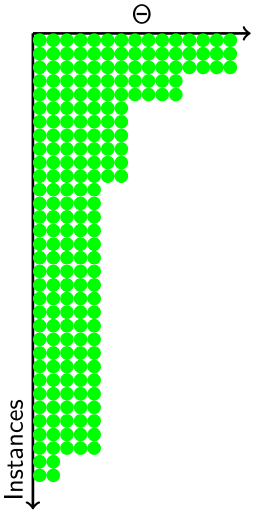
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
    - Evaluate all candidates on an instance, and another, ...
    - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
    - Remove outperformed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

Idea of (F-)Racing



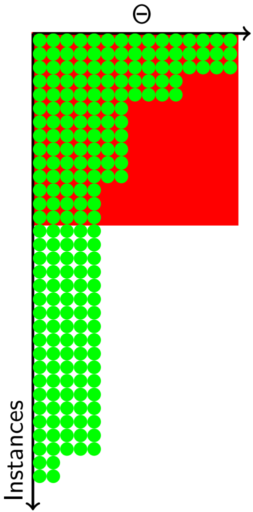
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
    - Evaluate all candidates on an instance, and another, ...
    - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
    - Remove outperformed 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

What might be problematic?

- We might have many or an infinite number of candidates

Iterated racing

- 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)

Whats good about this

- Very simple and generic algorithm
- Can easily be parallelized

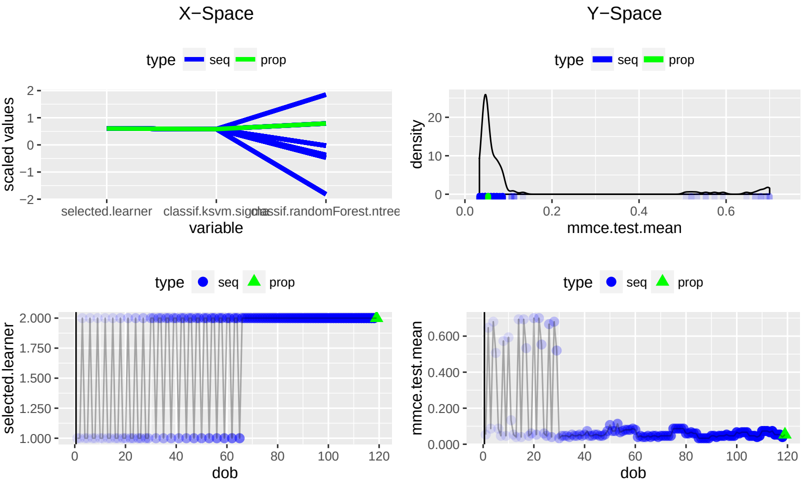
IRACE I

```
> bls = list(
+   makeLearner("classif.ksvm"),
+   makeLearner("classif.randomForest")
+ )
> lrn = makeModelMultiplexer(bls)
> ps = makeModelMultiplexerParamSet(lrn,
+   makeNumericParam("sigma", lower = -10, upper = 10, trafo = function(x) 2^x),
+   makeIntegerParam("ntree", lower = 1L, upper = 500L)
+ )
> rdesc = makeResampleDesc("CV", iters = 2L)
>
> ctrl = makeTuneControlIrace(maxExperiments = 120L)
> res = tuneParams(lrn, iris.task, rdesc, par.set = ps, control = ctrl)
> #Container object: Best params, performance, complete tuning trace
> print(res)

## Tune result:
## Op. pars: selected.learner=classif.randomForest; classif.randomForest.ntree=5
## mmce.test.mean=0.0541

> plotOptPath(res$opt.path, iters = 119, pause = FALSE, x.over.time = list("selected.learne
```

IRACE II



X−Space
type seq prop
scaled values
2
1
0
−1
−2
selected.learner
classif.ksvm.sigma
classif.randomForest.ntree
variable
Y−Space
type seq prop
density
20
10
0
0.0
0.2
0.4
0.6
mmce.test.mean
type seq prop
selected.learner
2.000
1.750
1.500
1.250
1.000
0
20
40
60
80
100
120
dob
type seq prop
mmce.test.mean
0.600
0.400
0.200
0.000
0
20
40
60
80
100
120
dob

Section 4

More nice features

Parallelization I

- We use our own package: `parallelMap`
- Setup:

```
> parallelStart("multicore")
> benchmark(...)
> parallelStop()
```

- Backends: `local`, `multicore`, `socket`, `mpi` and `BatchJobs`
- The latter means support for: makeshift SSH-clusters and HPC schedulers like SLURM, Torque/PBS, SGE or LSF
- Levels allow fine grained control over the parallelization
  - `mlr.resample`: Job = "train / test step"
  - `mlr.tuneParams`: Job = "resample with these parameter settings"
  - `mlr.selectFeatures`: Job = "resample with this feature subset"
  - `mlr.benchmark`: Job = "evaluate this learner on this data set"

Parallelization II

```
> lrns = list(makeLearner("classif.rpart"), makeLearner("classif.svm"))
> rdesc = makeResampleDesc("Bootstrap", iters = 100)
> parallelStart("multicore", 8)
> b = benchmark(lrns, iris.task, rdesc)
> parallelStop()
```

Parallelize the bootstrap instead:

```
> parallelStart("multicore", 8, level = "mlr.resample")
> b = benchmark(lrns, iris.task, rdesc)
> parallelStop()
```

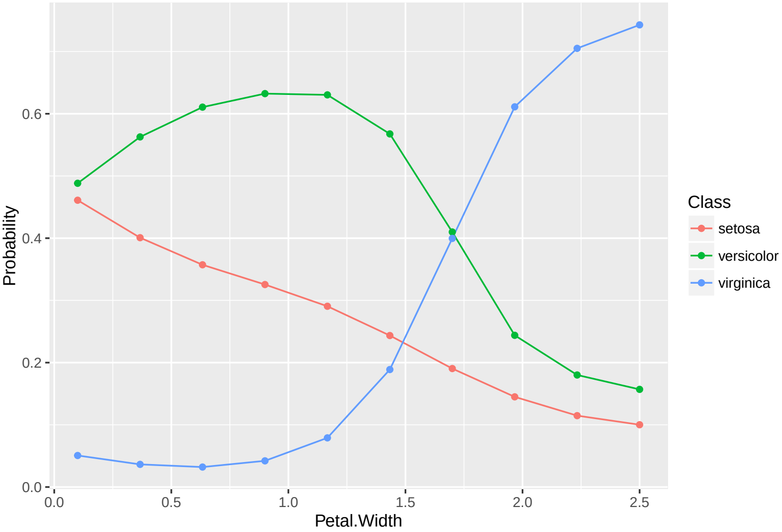
Partial Predictions Plots I

Partial Predictions

- Estimate how the learned prediction function is affected by one or more features.
- Displays marginalized version of the predictions of one or multiple effects.
- Reduce high dimensional function estimated by the learner.

```
> library(kernlab)
> lrn.classif = makeLearner("classif.svm", predict.type = "prob")
> fit.classif = train(lrn.classif, iris.task)
> pd = generatePartialPredictionData(fit.classif, iris.task, "Petal.Width")
>
> plotPartialPrediction(pd)
```

Partial Predictions Plots II



`mlr` Learner Wrappers I

What?

- Extend the functionality of learners by adding an `mlr` wrapper to them
- The wrapper hooks into the train and predict of the base learner and extends it
- This way, you can create a new `mlr` learner with extended functionality
- Hyperparameter definition spaces get joined!

mlr Learner Wrappers II

Available Wrappers

- Preprocessing: PCA, normalization (z-transformation)
- Parameter Tuning: grid, optim, random search, genetic algorithms, CMAES, iRace, MBO
- 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

R Example with FilterWrapper

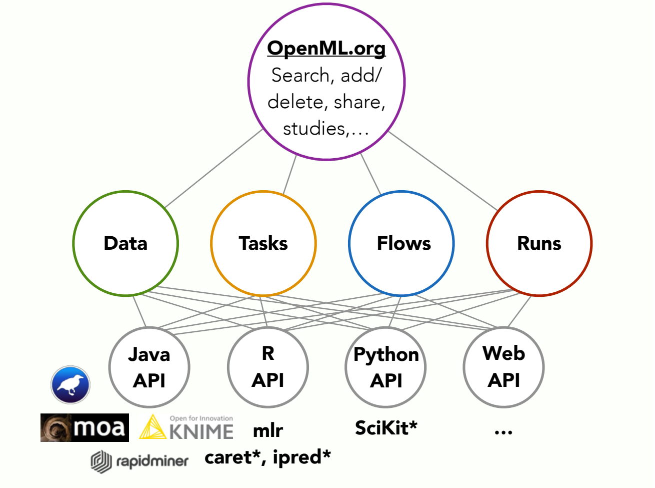
- A Learner can be fused with any wrapper, e.g. with a feature filter.
- `makeFilterWrapper` introduces the feature selection threshold `fw.perc` (selects `fw.perc*100%` of the top scoring features) as new hyperparameter.
- The optimal value for `fw.perc` can be determined by grid-search.

```
> lrn = makeFilterWrapper(learner = "classif.lda", fw.method = "information.gain")
> getParamSet(lrn)

##                        Type len     Def                                   Constr Req Tuna
## fw.method          discrete   -       - anova.test,carscore,cforest.importanc...   -    T
## fw.perc             numeric   -       -                                   0 to 1   -    T
## fw.abs              integer   -       -                                 0 to Inf   -    T
## fw.threshold        numeric   -       -                              -Inf to Inf   -    T
## fw.mandatory.feat   untyped   -       -                                        -   -    T
## method             discrete   -  moment                        moment,mle,mve,t   -    T
## nu                  numeric   -       -                                 2 to Inf   Y    T
## tol                 numeric   -  0.0001                                 0 to Inf   -    T
## predict.method     discrete   - plug-in              plug-in,predictive,debiased   -    T
## CV                  logical   -   FALSE                                        -   -   FA
```

OpenML

Main idea: Make ML experiments reproducible, computer-readable and allow collaboration with others.



OpenML R-Package I

`https://github.com/openml/r`

Tutorial

- `http://openml.github.io/openml-r`
- Caution: Work in progress

Current API in R

- Explore and Download data and tasks
- Register learners and upload runs
- Explore your own and other people's results

OpenML R-Package II

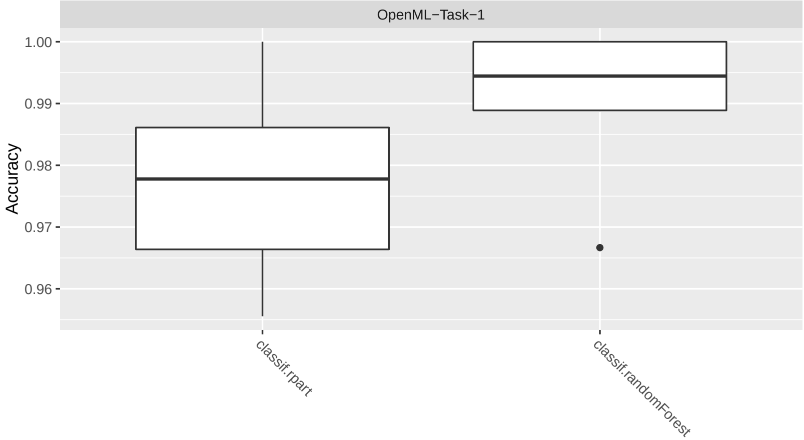
```
> library(OpenML)
> # set apikey after install (here public read-only key)
> setOMLConfig(apikey = "c1994bdb7ecb3c6f3c8f3b35f4b47f1f", arff.reader = "RWeka")

## OpenML configuration:
##   server           : http://www.openml.org/api/v1
##   cachedir         : /tmp/RtmpaUy0SS/cache
##   verbosity        : 1
##   arff.reader      : RWeka
##   confirm.upload   : TRUE
##   apikey           : ***************************47f1f

> oml.task = getOMLTask(1)
> res1 = runTaskMlr(oml.task, makeLearner("classif.rpart"))
> res2 = runTaskMlr(oml.task, makeLearner("classif.randomForest"))
> bmr = mergeBenchmarkResultLearner(res1$bmr, res2$bmr)
```

OpenML R-Package III

```
> plotBMRBoxplots(bmr)
```



Section 5

The End

There is more ...

- Clustering and Survival analysis
- Regular cost-sensitive learning (class-specific costs)
- Cost-sensitive learning (example-dependent costs)
- ROC and learning curves
- Imbalancy correction
- Multi-Label learning
- Bayesian optimization
- Multi-criteria optimization
- Ensembles, generic bagging and stacking
- Some interactive plots with `ggvis`
- ...

Outlook

We are working on

- Even better tuning system
- More interactive and 3D plots
- Large-Scale learning on databases
- Keeping the data on hard disk & distributed storage
- Time-Series tasks
- Large-Scale usage of OpenML
- `auto-mlr`
- ...

Thanks!