Applied Machine Learning and Efficient Model Selection with `mlr`

Bernd Bischl and Michel Lang

UseR 2015, Aalborg

Welcome!

- Project home page
  `https://github.com/berndbischl/mlr`
  - R documentation rendered in HTML
  - Tutorial for online viewing / download, including many examples
  - Don't hesitate to interrupt us
  - There will be a coffee break
- If you do not have `mlr` installed yet, please do so (see wiki page)

Overview

Section 1

Introduction

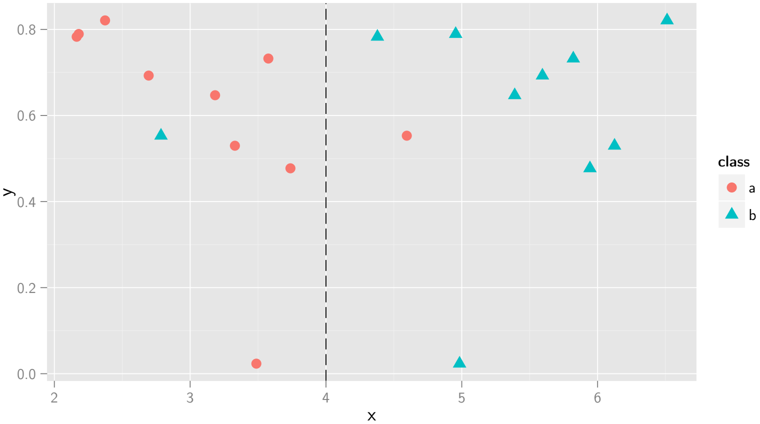
What is (supervised) machine learning?

- Learning structure in data
- The art of predicting stuff
- Model optimization
- Understanding of grey-box models

Disclaimer

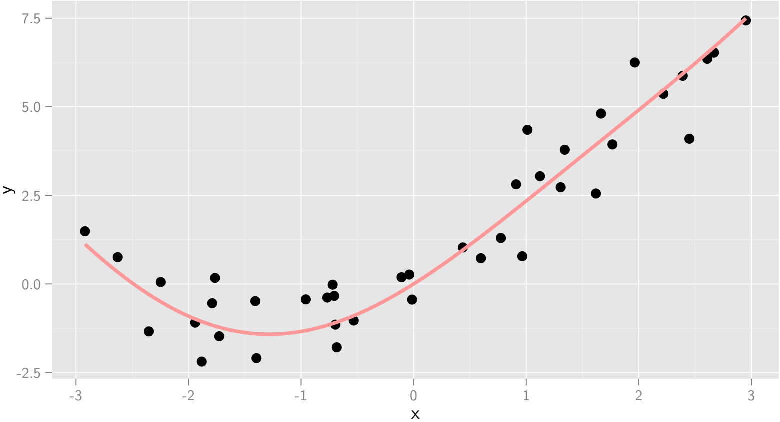
- The list is subjective and naively tailored to this talk
- ML is based on math and statistics, we will (mainly) talk about structure, software, and practical issues here

Supervised Classification tasks



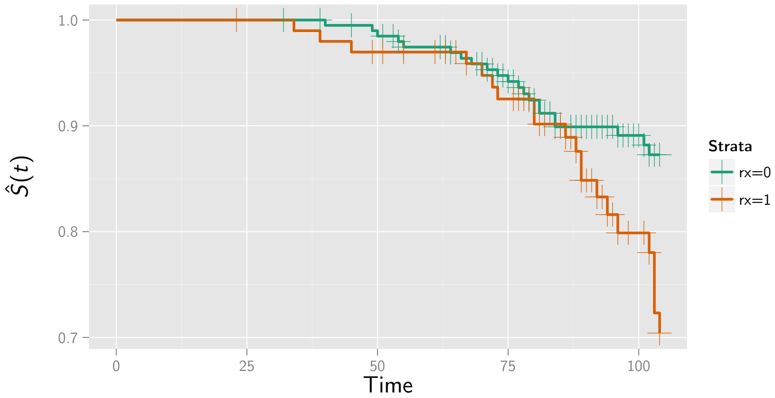
Goal: Predict a class (or membership probabilities)

Supervised Regression tasks



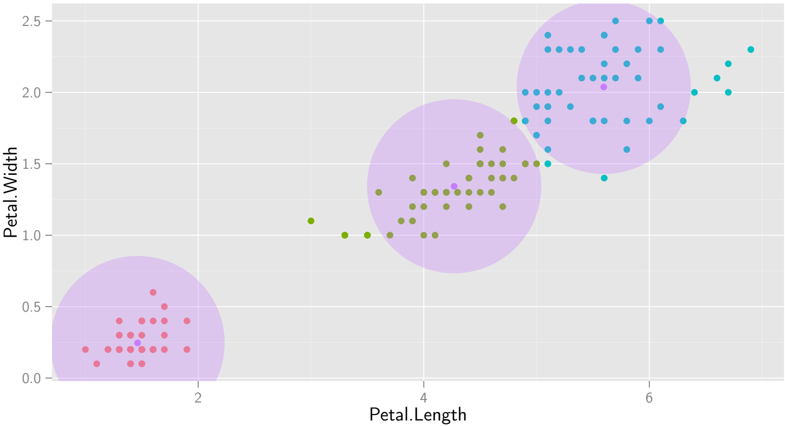
Goal: Predict a continuous output

Supervised Survival tasks



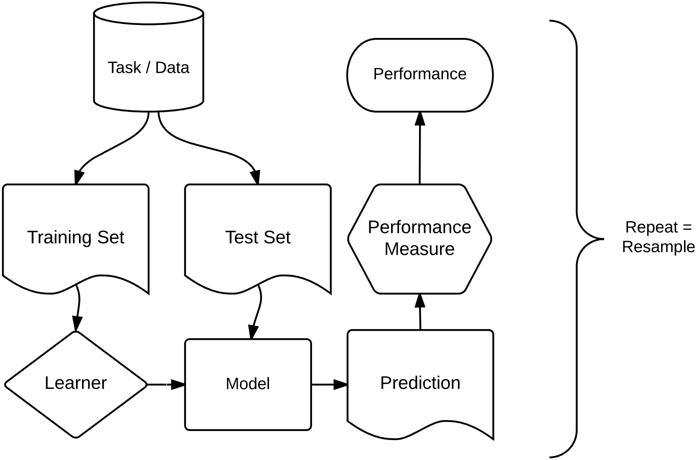
Goal: Predict a survival function $\hat{S}(t)$, i.e. the probability to survive to time point t

Unsupervised Cluster tasks



Goal: Group data into similar clusters (or estimate fuzzy membership probabilities)

Abstractions for Machine Learning



- All learning tasks fit well into this scheme
- Challenge as statistician: find a suitable model to maximize the outcome (or minimize the loss)

Section 2

Why mlr?

Motivation

The good news

- CRAN serves hundreds of packages for machine learning (cf. CRAN task view machine learning)
- Many packages are compliant to the unwritten interface definition:

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

Motivation

The bad news

- Some packages do not support the formula interface or their 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 (sometimes not documented at all)
- Many packages require the user to "guess" good hyperparameters
- Larger experiments lead to lengthy, tedious and error-prone code

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

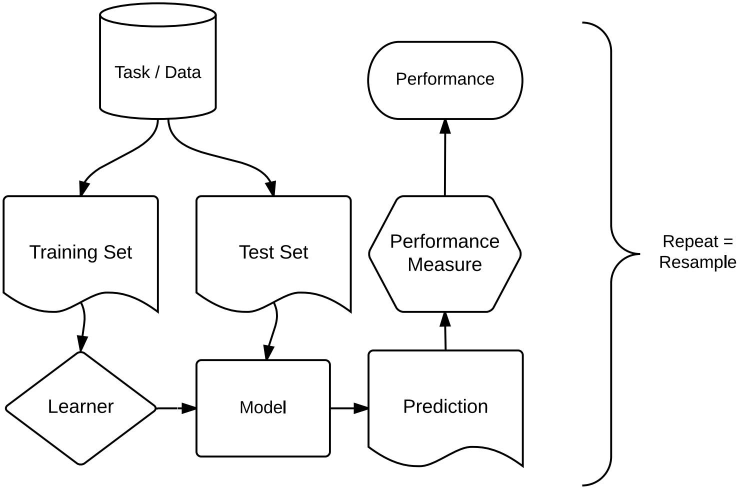
Motivation: `mlr`

- 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.

Section 3

Building Blocks

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
```

Task Abstraction: API I

```
> getTaskId(task)

## [1] "iris"

> str(getTaskData(task))

## 'data.frame': 150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1
```

Task Abstraction: API II

```
> str(getTaskDescription(task))

## List of 11
##  $ id          : chr "iris"
##  $ type        : chr "classif"
##  $ target      : chr "Species"
##  $ size        : int 150
##  $ n.feat      : Named int [1:3] 4 0 0
##   ..- attr(*, "names")= chr [1:3] "numerics" "factors" "ordered"
##  $ has.missings: logi FALSE
##  $ has.weights : logi FALSE
##  $ has.blocking: logi FALSE
##  $ class.levels: chr [1:3] "setosa" "versicolor" "virginica"
##  $ positive    : chr NA
##  $ negative    : chr NA
##  - attr(*, "class")= chr [1:2] "TaskDescClassif" "TaskDesc"
```

Task Abstraction: API III

```
> getTaskSize(task)

## [1] 150

> getTaskFeatureNames(task)

## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"

> getTaskTargetNames(task)

## [1] "Species"

> getTaskFormula(task)

## Species ~ .
## <environment: 0xd911ed8>

> summary(getTaskTargets(task))

##     setosa versicolor  virginica
##         50         50         50
```

Learner Abstraction I

- Internal structure of learners:
  - wrappers around `fit()` and `predict()` of the package
  - description of the parameter set
  - annotations
- Naming convention: `<tasktype>.<functionname>`

```
> makeLearner("classif.rpart")
> makeLearner("regr.rpart")
```

- Adding custom learners is covered in the tutorial

Learner Abstraction II

```
> 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
```

What Learners are available? I

Classification (54)

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

Clustering (6)

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

Regression (45)

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

Survival (10)

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

What Learners are available? II

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

```
> # list all classification learners which can predict probabilities
> # and allow multiclass classification
> head(unname(
+   listLearners("classif", properties = c("prob", "multiclass"))
+ ))

## [1] "classif.bdk"        "classif.boosting"   "classif.cforest"
## [4] "classif.ctree"      "classif.extraTrees" "classif.gbm"
```

What Learners are available? III

Get all applicable learners for a task

```
> head(unname(listLearners(task)))

## [1] "classif.bdk"         "classif.boosting"   "classif.cforest"
## [4] "classif.ctree"       "classif.extraTrees" "classif.fnn"
```

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 Trafo
## minsplit        integer   -   20 1 to Inf   -     -
## minbucket       integer   -    - 1 to Inf   -     -
## cp              numeric   - 0.01   0 to 1   -     -
## maxcompete      integer   -    4 0 to Inf   -     -
## maxsurrogate    integer   -    5 0 to Inf   -     -
## usesurrogate   discrete   -    2    0,1,2   -     -
## surrogatestyle discrete   -    0      0,1   -     -
## maxdepth        integer   -   30  1 to 30   -     -
## xval            integer   -   10 0 to Inf   -     -
## parms           untyped   -    -        -   -     -
```

Learner Abstraction: API

```
> lrn$properties

## [1] "twoclass"   "multiclass" "missings"   "numerics"   "factors"
## [6] "ordered"    "prob"       "weights"

> getHyperPars(lrn)

## $xval
## [1] 0

> lrn = setHyperPars(lrn, cp = 0.3)
> lrn = setPredictType(lrn, "prob")
> lrn = setPredictThreshold(lrn, 0.7);
```

Performance Measures

- Performance measures evaluate the predictions a test set and aggregate them over multiple in resampling iterations
- 22 classification, 7 regression, 5 cluster, 1 survival
- Internally: performance function, default aggregation function and 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:
```

What measures are available?

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

```
> listMeasures("classif")

##  [1] "f1"              "featperc"        "mmce"
##  [4] "tn"              "tp"              "mcc"
##  [7] "fn"              "fp"              "npv"
## [10] "bac"             "timeboth"        "acc"
## [13] "ppv"             "multiclass.auc" "brier"
## [16] "fnr"             "auc"             "tnr"
## [19] "ber"             "timepredict"     "fpr"
## [22] "gmean"           "tpr"             "gpr"
## [25] "fdr"             "timetrain"

> listMeasures(task)

## [1] "featperc"       "mmce"           "timeboth"
## [4] "acc"            "multiclass.auc" "ber"
## [7] "timepredict"    "timetrain"
```

Training and prediction

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] 68 75 78 11 31 145 63 110 123 9 ...
##  $ : int [1:100] 68 75 27 11 41 128 63 110 58 98 ...
##  $ : int [1:100] 27 78 41 31 145 128 123 58 9 46 ...
```

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: 0.05
## mmce.mean: 0.05
## mmce.sd: 0.03
## timetrain.aggr: 0.00
## timetrain.mean: 0.00
## timetrain.sd: 0.00
## Runtime: 0.0250013
```

Resampling Abstraction IV

```
> names(r)

##  [1] "learner.id"     "task.id"         "measures.train"
##  [4] "measures.test"  "aggr"            "pred"
##  [7] "models"         "err.msgs"        "extract"
## [10] "runtime"

> r$measures.test

##   iter    mmce timetrain
## 1    1 0.02667     0.005
## 2    2 0.06667     0.004

> r$aggr

##      mmce.test.mean timetrain.test.mean
##             0.04667             0.00450
```

Resampling Abstraction V

```
> head(as.data.frame(r$pred))

##    id  truth response iter  set
## 3   3 setosa   setosa    1 test
## 5   5 setosa   setosa    1 test
## 8   8 setosa   setosa    1 test
## 11 11 setosa   setosa    1 test
## 13 13 setosa   setosa    1 test
## 15 15 setosa   setosa    1 test
```

Configuring the Package

- What to do when training fails? error, warn, or be quiet?
  ➔ You don't want to stop in complex loops like `benchmark`
  ➔ `FailureModel` is created that predicts NAs
- Show verbose info messages?
- What if parameters are not described in learner?
- `?configureMlr` sets global flags and can be overwritten for individual learners

Section 4

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, ...

Benchmarking and Model Comparison II

Benchmarking in mlr

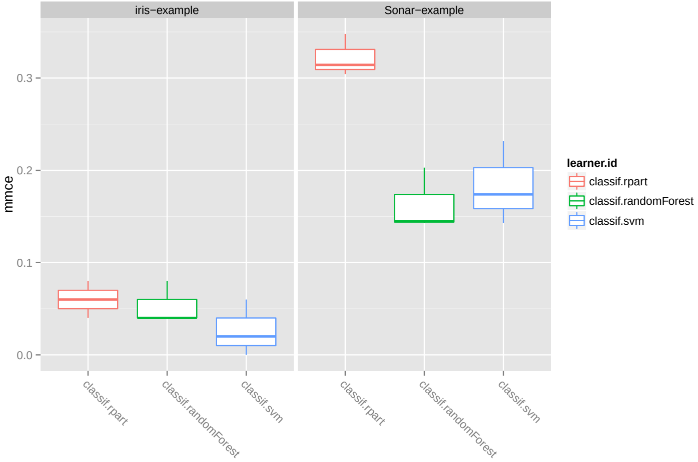
- Train and test sets are synchronized, i.e. all learners see the same data splits
- Can be done in parallel (see later)
- Can be combined with feature selection / tuning / nested resampling (see later)
- Results stored in well-defined container object, with getters and converters
- We are working on standard analysis tools

Benchmarking and Model Comparison III

```
> library(mlr)
> # lets try a couple of methods on some (mlr example) tasks
>
> # 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)
> set.seed(1)
> br = benchmark(learners, tasks, rdesc)
```

Benchmarking and Model Comparison IV

```
> plotBenchmarkResult(br)
```



Benchmarking and Model Comparison V

```
> getBMRAggrPerformances(br, as.df = TRUE)

##          task.id            learner.id mmce.test.mean
## 1  iris-example         classif.rpart        0.06000
## 2  iris-example classif.randomForest        0.05333
## 3  iris-example           classif.svm        0.02667
## 4 Sonar-example         classif.rpart        0.32215
## 5 Sonar-example classif.randomForest        0.16356
## 6 Sonar-example           classif.svm        0.18288
```

Benchmarking and Model Comparison VI

```
> getBMRPerformances(br, as.df = TRUE)

##          task.id          learner.id iter   mmce
## 1   iris-example        classif.rpart    1 0.0600
## 2   iris-example        classif.rpart    2 0.0400
## 3   iris-example        classif.rpart    3 0.0800
## 4   iris-example classif.randomForest    1 0.0400
## 5   iris-example classif.randomForest    2 0.0400
## 6   iris-example classif.randomForest    3 0.0800
## 7   iris-example          classif.svm    1 0.0000
## 8   iris-example          classif.svm    2 0.0200
## 9   iris-example          classif.svm    3 0.0600
## 10 Sonar-example        classif.rpart    1 0.3478
## 11 Sonar-example        classif.rpart    2 0.3043
## 12 Sonar-example        classif.rpart    3 0.3143
## 13 Sonar-example classif.randomForest    1 0.2029
## 14 Sonar-example classif.randomForest    2 0.1449
## 15 Sonar-example classif.randomForest    3 0.1429
## 16 Sonar-example          classif.svm    1 0.2319
## 17 Sonar-example          classif.svm    2 0.1739
## 18 Sonar-example          classif.svm    3 0.1429
```

Benchmarking and Model Comparison VII

```
> head(getBMRPredictions(br, as.df = TRUE), 10)

##          task.id    learner.id id  truth response iter  set
## 1  iris-example classif.rpart  1 setosa   setosa    1 test
## 2  iris-example classif.rpart  3 setosa   setosa    1 test
## 3  iris-example classif.rpart 12 setosa   setosa    1 test
## 4  iris-example classif.rpart 17 setosa   setosa    1 test
## 5  iris-example classif.rpart 22 setosa   setosa    1 test
## 6  iris-example classif.rpart 24 setosa   setosa    1 test
## 7  iris-example classif.rpart 25 setosa   setosa    1 test
## 8  iris-example classif.rpart 26 setosa   setosa    1 test
## 9  iris-example classif.rpart 31 setosa   setosa    1 test
## 10 iris-example classif.rpart 34 setosa   setosa    1 test
```

Section 5

Hyperparameter Tuning

Hyperparameter Tuning I

Tuning

- Used to find "best" hyperparameters for a method in a data-dependent way
- Essential for some methods, e.g. SVMs

Tuning in `mlr`

- General procedure: Tuner proposes param point, eval by resampling, feedback value to tuner
- Multiple tuners through exactly the same interface
- All evals and more info is logged into `OptPath` object

Hyperparameter Tuning II

Grid search

- Basic method: Exhaustively try all combinations of finite grid
- Inefficient, combinatorial explosion
- Searches large, irrelevant areas
- Reasonable for continuous parameters?
- Still often default method

Random search

- Randomly draw parameters
- `mlr` supports all types and dependencies
- Scales better then grid search, easily extensible

Tuning

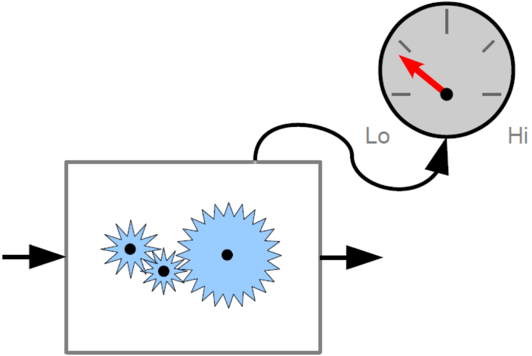
Automatic Model Selection

Prior approaches:

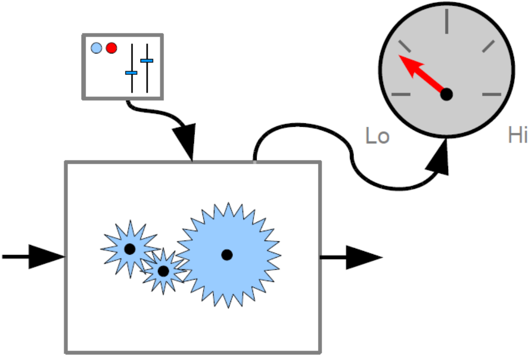
- Looking for the silver bullet model
  $\rightsquigarrow$ Failure
- Exhaustive benchmarking / search
  $\rightsquigarrow$ Per data set: too expensive
  $\rightsquigarrow$ Over many: contradicting results
- Meta-Learning:
  $\rightsquigarrow$ Failure
  $\rightsquigarrow$ Usually not for preprocessing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

Goal: Data dependent + Automatic + Efficient

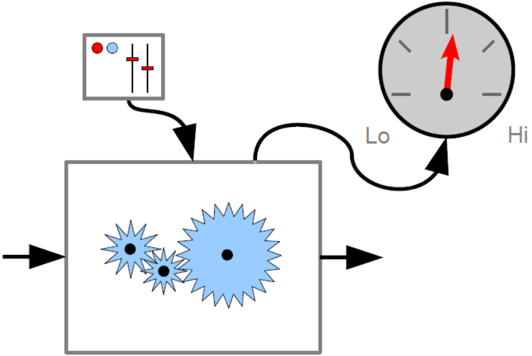
Black-Box-Perspective in Configuration



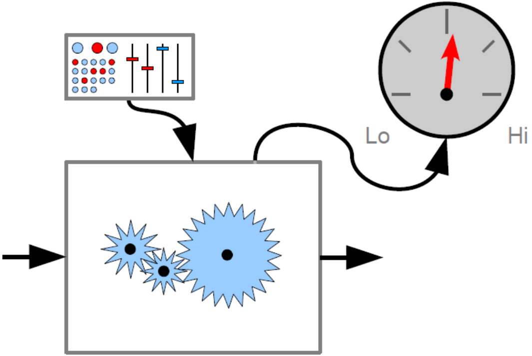
Black-Box-Perspective in Configuration



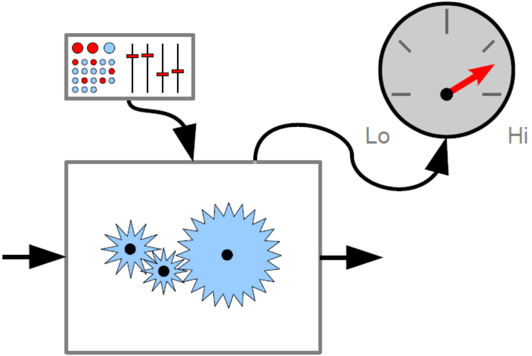
Black-Box-Perspective in Configuration



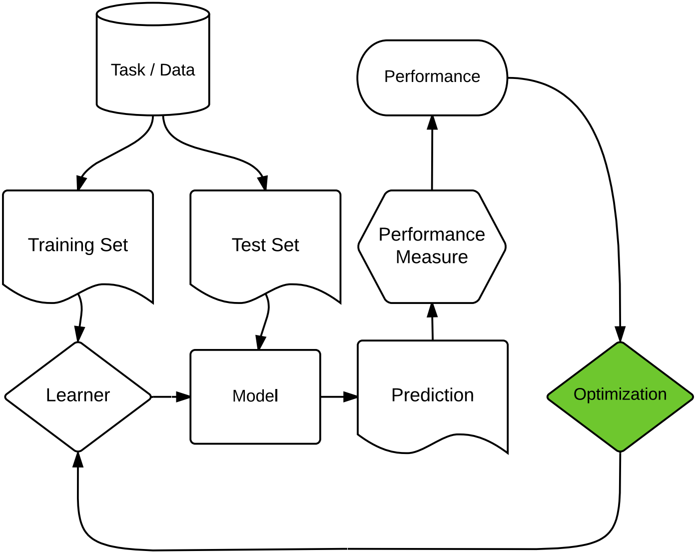
Black-Box-Perspective in Configuration



Black-Box-Perspective in Configuration



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

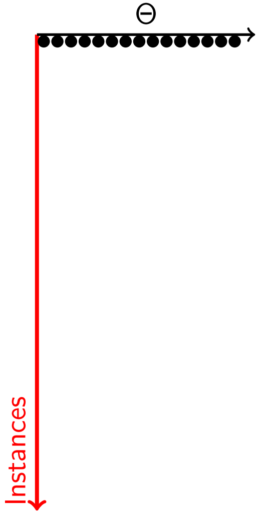
Idea of (F-)Racing

- 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.

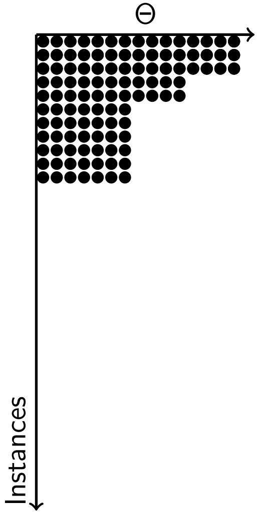
Idea of (F-)Racing

- 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

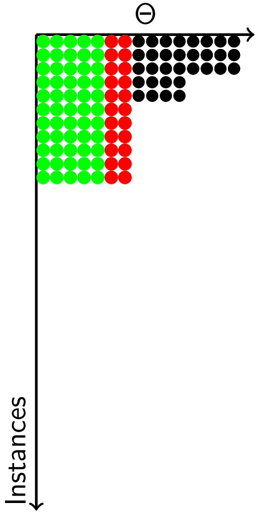
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed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

Idea of (F-)Racing



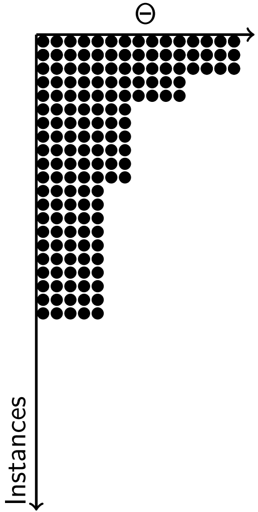
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, . . .
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed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

Idea of (F-)Racing



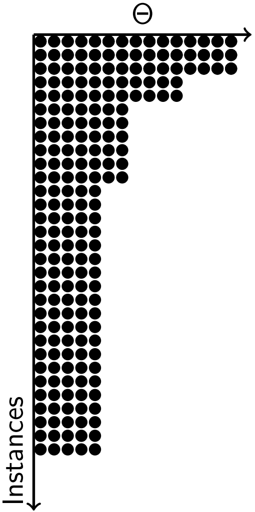
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed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

Idea of (F-)Racing



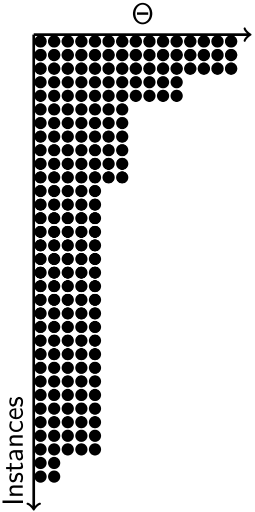
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, . . .
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed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

Idea of (F-)Racing



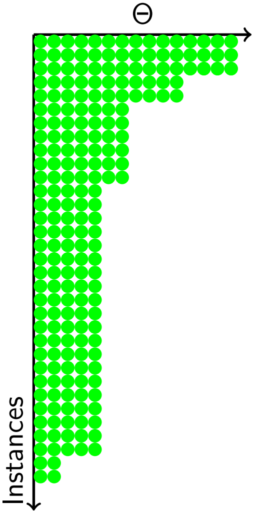
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed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

Idea of (F-)Racing



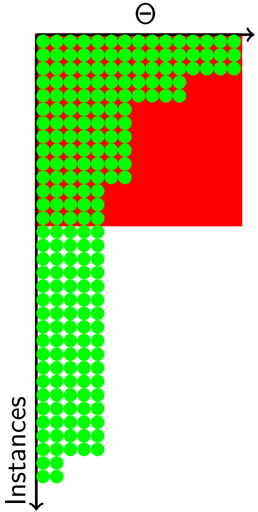
- Write down all candidate solutions
- Iterate the following till budget exhausted
- One "generation"
  - Evaluate all candidates on an instance, and another, ...
  - After some time, compare candidates via statistical test, e.g., Friedman test with post-hoc analysis for pairs
  - Remove outperformed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

Section 6

Feature Selection

Feature Selection I

- Reduce dimensionality, increase interpretability and predictive performance
- Concepts:

  Filter: Preliminary step, independent from model

  Wrapper: Wrapped around model fit which is iteratively scored

  Embedded: Model has feature selection embedded, e.g. lasso regression

Feature Selection II

Feature Filters

- Usually: Quickly compute a numerical score per feature
- Encodes influence of feature on output
- Often independent of ML model
- Often fast to compute
- Can be used to visualize data structure
- Can be used to rank or threshold the feature set, and to reduce feature set size
- Terrible if complex correlations exist

Feature Selection III

Filter examples

- Correlation between x_i and y in regression
- Mutual information in classification
- The random forest importance value
- χ^2-statistic for independence between x_i and y

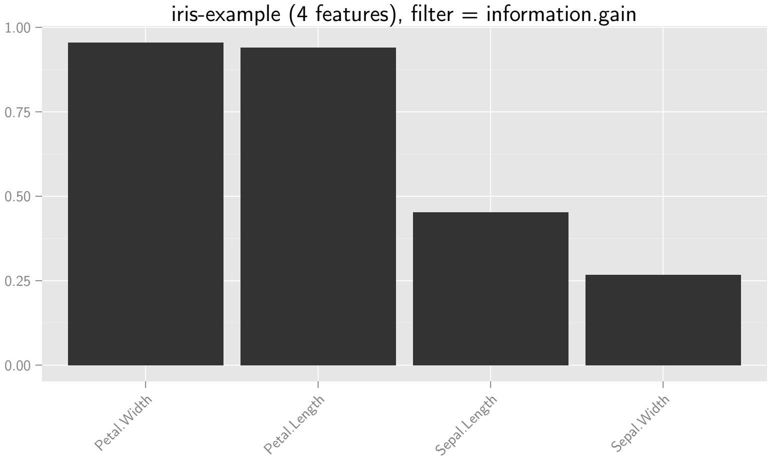
Feature Selection IV

```
> fv = generateFilterValuesData(iris.task, method = "information.gain")
> print(fv)

## FilterValues:
## Task: iris-example
##           name    type information.gain
## 1 Sepal.Length numeric           0.4521
## 2  Sepal.Width numeric           0.2673
## 3 Petal.Length numeric           0.9403
## 4  Petal.Width numeric           0.9554

> task2 = filterFeatures(iris.task, fval = fv, perc = 0.5)
> print(getTaskFeatureNames(task2))

## [1] "Petal.Length" "Petal.Width"
```

You can optimize this selection threshold jointly with the model!

Feature Selection V

```
> plotFilterValues(fv)
```

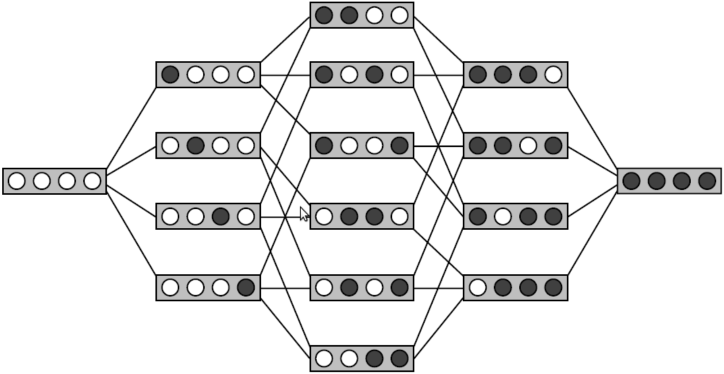


Feature Selection VI

Wrapper approach

- Evaluate feature sets with learner, e.g. by cross-validation
- Measures probably what you are interested in
- Will be slow in very high-dimensional spaces
- Sequential Forward Search (SFS) (or backward)
- Sequential Floating Forward Search (SFFS)
- Genetic Algorithm (GA)

Feature Selection VII



Feature Selection VIII

```
> ## Specify the search strategy
> ctrl = makeFeatSelControlSequential(method = "sfs", alpha = 0.05)
>
> ## Select features
> rdesc = makeResampleDesc("CV", iters = 10)
> sfeats = selectFeatures(learner = "regr.lm", task = bh.task,
+   resampling = rdesc, control = ctrl, show.info = FALSE)
> sfeats

## FeatSel result:
## Features (11): crim, zn, chas, nox, rm, dis, rad, tax, ptratio, b, lstat
## mse.test.mean=23.7
```

Feature Selection IX

```
> analyzeFeatSelResult(sfeats)


## Features         : 11
## Performance      : mse.test.mean=23.7
## crim, zn, chas, nox, rm, dis, rad, tax, ptratio, b, lstat
##
## Path to optimum:
## - Features:    0  Init   :                       Perf = 84.896  Diff: NA  *
## - Features:    1  Add    : lstat                 Perf = 39.013  Diff: 45.883  *
## - Features:    2  Add    : rm                    Perf = 31.553  Diff: 7.4592  *
## - Features:    3  Add    : ptratio               Perf = 27.992  Diff: 3.5617  *
## - Features:    4  Add    : dis                   Perf = 27.189  Diff: 0.80245  *
## - Features:    5  Add    : nox                   Perf = 25.734  Diff: 1.4555  *
## - Features:    6  Add    : b                     Perf = 25.207  Diff: 0.52638  *
## - Features:    7  Add    : zn                    Perf = 24.935  Diff: 0.27213  *
## - Features:    8  Add    : chas                  Perf = 24.73  Diff: 0.20546  *
## - Features:    9  Add    : rad                   Perf = 24.595  Diff: 0.1346  *
## - Features:   10  Add    : tax                   Perf = 24.11  Diff: 0.48483  *
## - Features:   11  Add    : crim                  Perf = 23.695  Diff: 0.41533  *
##
## Stopped, because no improving feature was found.
```

Section 7

mlr Learner Wrappers

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!

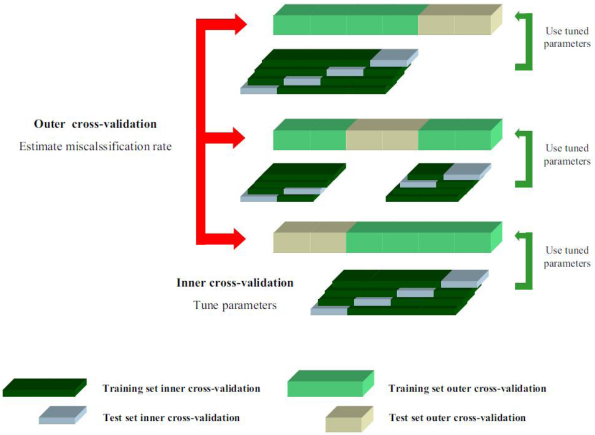
mlr Learner Wrappers II

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

Nested Resampling

- Using the TuningWrapper or FeatureSelectionWrapper allows to enable nested resampling
- Ensures **unbiased** results for model optimization
- Everything else is statistically unsound



R Example

Complex tuning example

Section 8

Parallelization

Parallelization I

- We use our own package: `parallelMap`
- Initialize a backend with `parallelStart`
- Stop with `parallelStop`

```
> 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
- The first loop which is marked as parallel executable will be automatically parallelized

Parallelization II

Parallelization levels

- Which loop to parallelize depends on number of iterations
- Levels allow fine grained control over the parallelization
  - `mlr.resample`: Each resampling iteration (a train / test step) is a parallel job.
  - `mlr.benchmark`: Each experiment "run this learner on this data set" is a parallel job.
  - `mlr.tuneParams`: Each evaluation in hyperparameter space "resample with these parameter settings" is a parallel job. How many of these can be run independently in parallel depends on the tuning algorithm.
  - `mlr.selectFeatures`: Each evaluation in feature space "resample with this feature subset" is a parallel job.

Parallelization III

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

## Starting parallelization in mode=multicore with cpus=8.

> benchmark(learners = lrns, tasks = iris.task, resamplings = rdesc)

## Mapping in parallel:  mode = multicore; cpus = 8; elements = 2.

##         task.id    learner.id mmce.test.mean
## 1 iris-example classif.rpart        0.05659
## 2 iris-example   classif.svm        0.04218

> parallelStop()

## Stopped parallelization.  All cleaned up.
```

Parallelization IV

Parallelize the bootstrap instead:

```
> parallelStart("multicore", 8, level = "mlr.resample")

## Starting parallelization in mode=multicore with cpus=8.

> benchmark(learners = lrns, tasks = iris.task, resamplings = rdesc)

## Mapping in parallel:  mode = multicore; cpus = 8; elements = 100.
## Mapping in parallel:  mode = multicore; cpus = 8; elements = 100.

##         task.id    learner.id mmce.test.mean
## 1 iris-example classif.rpart        0.05810
## 2 iris-example   classif.svm        0.04363

> parallelStop()

## Stopped parallelization.  All cleaned up.
```

Section 9

Visualizations

Visualizations

- We use `ggplot2` and interactive `ggvis` as a standard, if possible
- Some plots use Viper Charts as backend (cost curves, lift charts, ...)
- Current GSOC project with Zach Jones
  - Demo plots for models in teaching
  - ROC curves
  - Threshold vs. Performance
  - Partial dependency plot
  - Learning curves

Visualizations

Section 10

CARET VS. `MLR`

caret vs. mlr I

Why we like `mlr` more

- `mlr` has (in our opinion) a better OO design, class structure and infrastructure for future development and for users
- This makes things combinable, extensible, predictable
- More flexible parallelization with `parallelMap`, even on HPCs via `BatchJobs`
- Tuning with advanced methods like `irace`
- Fusing learners with other operations like pre-processing and feature selection
- Nested resampling (required for unbiased results)
- Survival and cluster analysis

Section 11

OpenML

OpenML-R-Package I

Caution: Work in progress

OpenML?

- Main idea: Make ML experiments reproducible and most parts computer-readable
- Share everything
- Enrich with meta-information
- Later: Mine the results, meta-learn on it

1 minute intro

Data from various sources **analysed and organised online** for easy access

Scientists can **broadcast data,** explaining the challenge that needs to be addressed. OpenML will (for known data formats) **automatically analyze the data**, compute data characteristics, **annotate and index it for easy search**

1 minute intro

Scientific tasks that can be **interpreted by tools**, and **solved collaboratively**

Tasks are **realtime (collaborative) data mining challenges**, allowing anyone to build on previous results. OpenML creates **machine-readable descriptions** so that tools can **automatically download data**, use the correct procedures, and **upload all results online**.

1 minute intro

Tool plugins
for automated
data download,
workflow upload and
experiment logging
and sharing

Flows are implementations of algorithms, workflows, or scripts **solving OpenML tasks**. OpenML keeps track of **flow details and versioning**, **organizes all their results** for easy comparison, even across tools.

1 minute intro

Experiments auto-uploaded, linked to **data, flows** and **authors**, and organised for easy reuse

Runs contain the results that **flows** obtained on specific tasks. Runs are **fully reproducible**, linked to the underlying data, tasks, flows and authors. OpenML **organizes all results online** for **discovery, comparison and reuse**

OpenML-R-Package I

Let's visit website and project page

OpenML-R-Package II

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

Current API in R

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

Already nicely connected to `mlr`!

OpenML: Explore and Select Data I

```
> library(OpenML)
> authenticateUser() # uses my OML config file

## Authenticating user at server:  bernd_bischl@gmx.net
## Retrieved session hash.  Valid until:  2015-06-30 11:37:13

> 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
```

OpenML: Explore and Select Data II

```
> 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
```

OpenML: Download a Data Set

```
> # uses built in caching from disk
> getOMLDataSet(6)

## Getting data set '6' from OpenML repository.
## Downloading
'http://www.openml.org/api/?f=openml.data.description&data.id=6' to
'/tmp/RtmpHTqqZA/cache/datasets/6/description.xml'
## Downloading
'http://openml.liacs.nl/data/download/6/dataset_6_letter.arff' to
'/tmp/RtmpHTqqZA/cache/datasets/6/dataset.arff'

##
## Data Set "letter" :: (Version = 1, OpenML ID = 6)
##   Default Target Attribute: class
```

OpenML: Download a Task I

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

## Downloading task '1' from OpenML repository.
## Downloading 'http://www.openml.org/api/?f=openml.task.get&task_id=1'
to '/tmp/Rtmp0QG2mN/cache/tasks/1/task.xml'
## Authenticating user at server:  bernd_bischl@gmx.net
## Retrieved session hash.  Valid until:  2015-06-30 12:19:28
## Getting data set '1' from OpenML repository.
## Downloading
'http://www.openml.org/api/?f=openml.data.description&data.id=1' to
'/tmp/Rtmp0QG2mN/cache/datasets/1/description.xml'
## Downloading
'http://openml.liacs.nl/data/download/1/dataset_1_anneal.arff' to
'/tmp/Rtmp0QG2mN/cache/datasets/1/dataset.arff'
## Downloading
'http://www.openml.org/api_splits/get/1/Task_1_splits.arff' to
'/tmp/Rtmp0QG2mN/cache/tasks/1/datasplits.arff'
```

OpenML: Download a Task II

```
> 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)
```

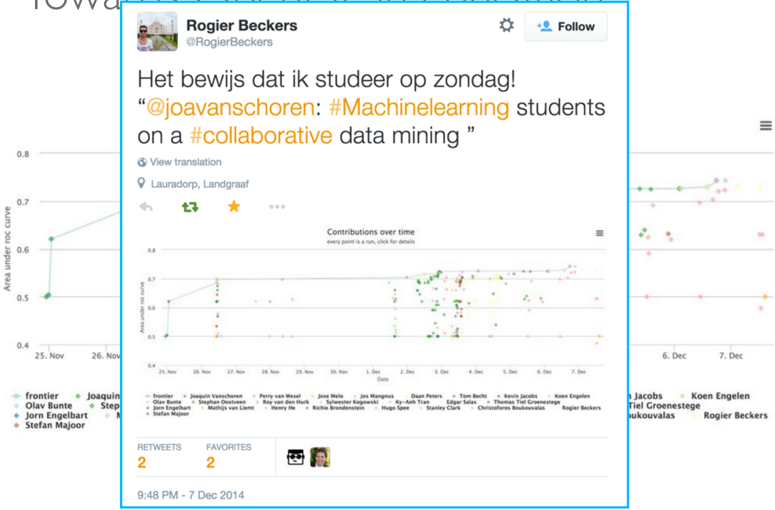
OpenML: Run a Task

```
> 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,marvi,corr,blue.2Fbright.2F
## 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&external_version=R_7f351643
to '<mem>'
```

OpenML: Upload 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)
```

Towards OpenML in education



Section 12

The End

There is more . . .

- Regular cost-sensitive learning (class-specific costs)
- Cost-sensitive learning (example-dependent costs)
- Model-based optimization
- Multi-criteria optimization
- OpenML
- . . .

Outlook

We are working on

- Even better tuning system
- More interactive plots
- Large-Scale SVM ensembles
- Time-Series tasks
- Better benchmark analysis
- Multi-Label classification
- ...

Thanks!