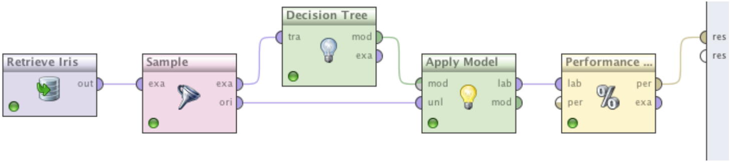
mlr?

- No unifying interface for machine learning in R
- Experiments require lengthy, tedious and error-prone code
- Machine learning is (also) experimental science:
  We need powerful and flexible tools!
- mlr now exists for 3-4 years (or longer?), grown quite large
- Still heavily in development, but official releases are stable
- Was used for nearly all of my papers
- We cannot cover everything today, short intro + overview
  Focus: Resampling / model selection / benchmarking!

mlr?

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



mlr: abstractions, glue code and some own implementations

The mlr Team

- Bernd Bischl (Muenchen)
- Michel Lang (Dortmund)
- Jakob Richter (Dortmund)
- Lars Kotthoff (Cork)
- Julia Schiffner (Duesseldorf)
- Eric Studerus (Basel)

Package + Documentation

Main project page on Github

- URL: `https://github.com/berndbischl/mlr`
- Contains further links, tutorial, issue tracker.
- Official versions are released to CRAN.

How to install

- `install.packages("mlr")`
- `install_github("mlr", username = "berndbischl")`

Documentation

- Tutorial on project page (still growing)
- R docs and examples in HTML on project page (and in package)

- Disclaimer:
  Slides are here to remind me of what I want to show you.
- And the covered examples have to be short.
- Refer to tutorial, examples and technical docs later!

Hence: Let's explore the web material a bit!

Features I

- Clear S3 / object oriented interface
- Easy extension mechanism through S3 inheritance

- Abstract description of learners and data by properties
- Description of data and task
- Many convenience methods, generic building blocks for own experiments
- Resampling like bootstrapping, cross-validation and subsampling
- Easy tuning of hyperparameters
- Variable selection
- Benchmark experiments with 2 levels of resampling

Features II

- Growing tutorial / examples
- Extensive unit-testing (testthat)
- Extensive argument checks to help user with errors
- Parallelization through parallelMap
  - Local, socket, MPI and BatchJobs modes
  - Parallelize experiments without touching code
  - Job granularity can be changed, e.g., so jobs don't complete too early

Remarks on S3

- Not much needed (for usage)
- Makes extension process easy
- Extension is explained in tutorial
- If you simply use the package, you don't really need to care!

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              -
```

Learner Abstractions

```
head(listLearners("classif", properties = c("prob", "multiclass")))

##           classif.bdk     classif.boosting
##         "classif.bdk"   "classif.boosting"
##       classif.cforest        classif.ctree
##     "classif.cforest"      "classif.ctree"
##    classif.extraTrees          classif.gbm
## "classif.extraTrees"        "classif.gbm"
```

Learner Abstractions

```
head(listLearners(iris.task))

##           classif.bdk     classif.boosting
##         "classif.bdk"   "classif.boosting"
##       classif.cforest        classif.ctree
##     "classif.cforest"      "classif.ctree"
##    classif.extraTrees          classif.fnn
##  "classif.extraTrees"        "classif.fnn"
```

Performance Measures

- 22 classification, 7 regression, 1 survival
- 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:
```

Performance Measures

```
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"
```

Overview of Implemented Learners

Classification

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

Clustering

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

Much better documented on web page, let's go there!

Regression

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

Survival Analysis

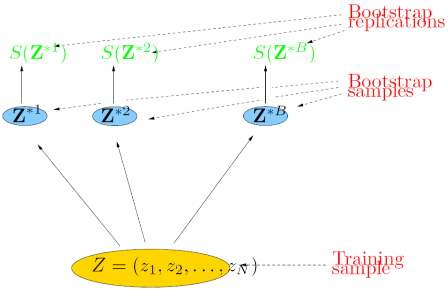
- Cox-PH
- Cox-Boost
- Random survival forest
- Penalized regression
- ...

Example

ex1.R: Training and prediction

Resampling

- Hold-Out
- Cross-validation
- Bootstrap
- Subsampling
- Stratification
- Blocking
- and quite a few extensions



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
##        0.06667        0.93333

crossval(lrn, task)$aggr

## mmce.test.mean
##        0.06667
```

Example

ex2.R: Resampling

Benchmarking

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

```
data("Sonar", package = "mlbench")
tasks = list(
  makeClassifTask(data = iris, target = "Species"),
  makeClassifTask(data = Sonar, target = "Class")
)
learners = list(
  makeLearner("classif.rpart"),
  makeLearner("classif.randomForest"),
  makeLearner("classif.ksvm")
)

benchmark(learners, tasks, cv10f, mmce)
```

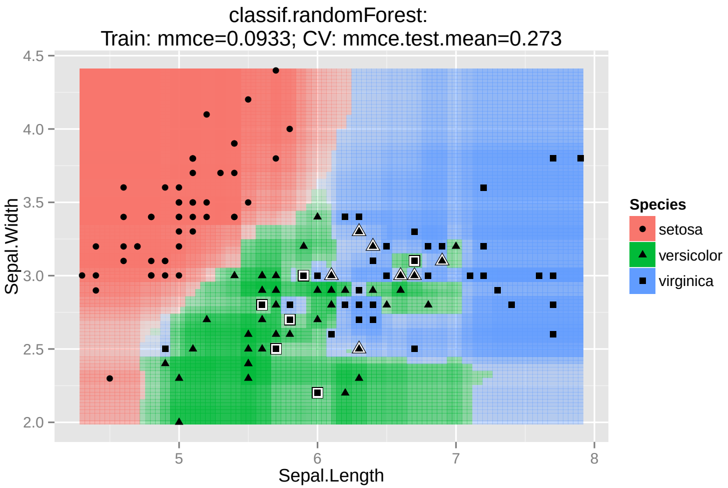
```
##   task.id          learner.id mmce.test.mean
## 1    iris       classif.rpart        0.06000
## 2    iris classif.randomForest       0.04667
## 3    iris        classif.ksvm        0.04000
## 4   Sonar       classif.rpart        0.30762
## 5   Sonar classif.randomForest       0.20619
## 6   Sonar        classif.ksvm        0.19214
```

Example

ex3.R: Benchmarking

Visualizations

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



Example

ex4.R: ROC analysis

Remarks on model selection I

Basic machine learning

- Fit parameters of model to predict new data
- Generalisation error commonly estimated by resampling, e.g. 10-fold cross-validation

2nd, 3rd, ... level of inference

- Comparing inducers or hyperparameters is harder
- Feature selection either in 2nd level or adds a 3rd one ...
- Statistical comparisons on the 2nd stage are non-trivial

- Still active research
- Very likely that high performance computing is needed

Tuning / Grid Search / Random search

Tuning

- Used to find “best” hyperparameters for a method in a data-dependend way
- Must be done for some methods, like SVMs

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

Tuning / Grid Search / Random search

Random search

- Randomly draw parameters
- mlr supports all types and depencies here
- Scales better then grid search

Example

ex5.R: Basic tuning: grid search

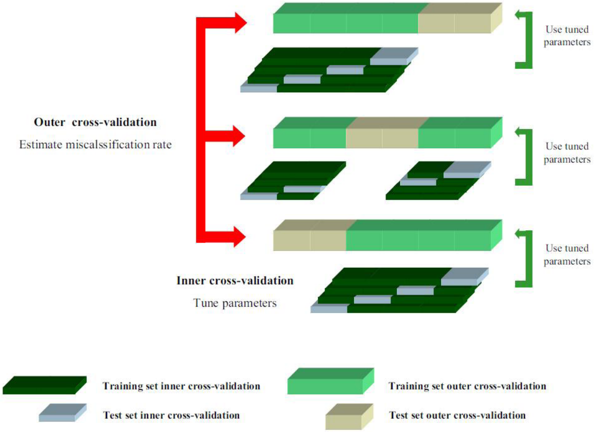
Remarks on Model Selection II

Salzberg (1997):
On comparing classifiers: Pitfalls to avoid and a recommended approach

- Many articles do not contain sound statistical methodology
- Compare against enough reasonable algorithms
- Do not just report mean performance values
- Do not cheat through repeated tuning
- Use double cross-validation for tuning and evaluation
- Apply the correct test (assumptions?)
- Adapt for multiple testing
- Think about independence and distributions
- Do not rely solely on UCI $\rightarrow$ We might overfit on it

Nested Resampling

- Ensures unbiased results for meta model optimization
- Use this for tuning and feature selection



Example

ex6.R: Tuning + nested resampling via wrappers

Parallelization

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

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

- Parallelization levels

```
parallelGetRegisteredLevels()

## mlr: mlr.benchmark, mlr.resample, mlr.selectFeatures, mlr.tuneParams
```

Defaults to first possible / most outer loop

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

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

Parallelization

- parallelMap is documented here:
  `https://github.com/berndbischl/parallelMap`
- BatchJobs is documented here:
  `https://github.com/tudo-r/BatchJobs`
  Make sure to read the Wiki page for Dortmund!

Example

ex7.R: Parallelization

Outlook / Not Shown Today I

- Regression
- Survival analysis
- Clustering
- Regular cost-sensitive learning (class-specific costs)
- Cost-sensitive learning (example-dependent costs)
- Smarter tuning algorithms
  CMA-ES, iterated F-facing, model-based optimization ...
- Multi-critera optimization
- ...

Outlook / Not Shown Today II

- Variable selection
  Filters: Simply imported available R methods
  Wrappers: Forward, backward, stochastic search, GAs
- Bagging for arbitary base learners
- Generic imputation for missing values
- Wrapping / tuning of preprocessing
- Over / Undersampling for unbalanced class sizes
- mlr connection to OpenML
- ...