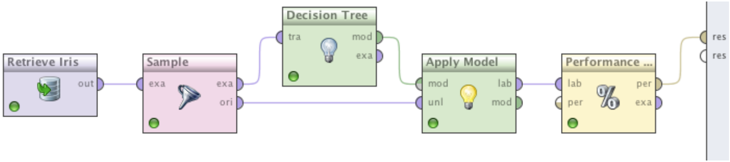
mlr: Machine Learning in R

Michel Lang, Bernd Bischl, Jakob Richter

`mlr 2.0`: `https://github.com/berndbischl/mlr`
(uploaded to CRAN)

mlr?

- 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

Task Abstractions

- Regression, classification, survival and cost-sensitive 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
##        4        0
## Missings: FALSE
## Has weights: FALSE
## Has blocking: FALSE
## Classes: 3
##     setosa versicolor  virginica
##         50         50         50
## Positive class: NA
```

Learner Abstractions

- 37 classification, 20 regression, 6 survival
- Internally: functions to train and predict, parameter set and annotations

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

## Learner classif.rpart from package rpart
## Type: classif
## Class: classif.rpart
## Properties: twoclass,multiclass,missings,numerics,factors,prob,weights
## Predict-Type: response
## Hyperparameters: xval=0
```

Learner Abstractions

```r
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   -    -        -   -     -
```

Performance Measures

- 20 classification, 7 regression, 1 survival
- Internally: performance function, aggregation function and annotations

```
print(mmce)

## Performance measure: mmce
## Properties: classif,classif.multi
## Minimize: TRUE
## Best: 0; Worst: 1
## Aggregated by: test.mean

print(timetrain)

## Performance measure: timetrain
## Properties: classif,classif.multi,regr,surv,costsens
## Minimize: TRUE
## Best: 0; Worst: Inf
## Aggregated by: test.mean
```

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

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

benchmark(learners, tasks, cv10f, mmce)

##    task        learner mmce.test.mean
## 1  iris  classif.rpart        0.04667
## 2  iris classif.logreg        0.33333
## 3  iris   classif.ksvm        0.06000
## 4 Sonar  classif.rpart        0.16810
## 5 Sonar classif.logreg        0.30667
## 6 Sonar   classif.ksvm        0.28881
```

Parallelization

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

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

- Parallelization levels

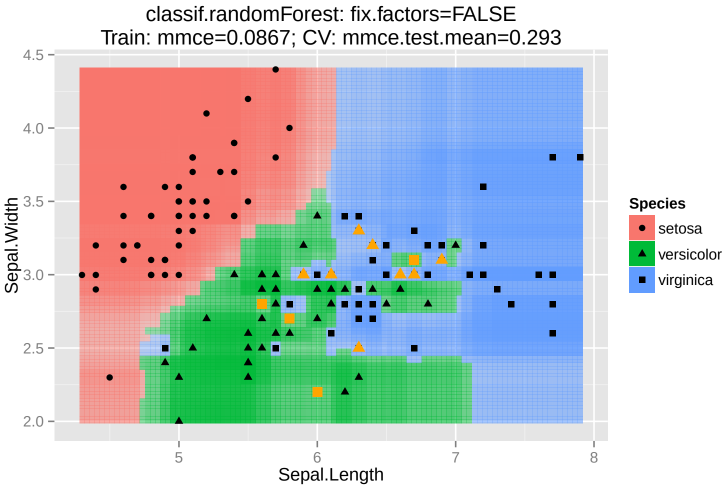
```
parallelShowRegisteredLevels()

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

  Defaults to first possible / most outer loop
- Few iterations in benchmark (loop over `learners` × `tasks`), many in resampling

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

Visualizations

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



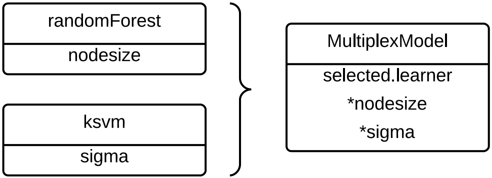
Wrapper

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
- Over- and Undersampling for unbalanced classification
- Parameter Tuning: grid, optim, random search, genetic algorithms, CMAES, iRace, MBO

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)

```r
task = makeClassifTask(data = Sonar, target = "Class")
outer = makeResampleDesc("Subsample", iters = 1)
res = resample(lrn, task, outer, models = TRUE)
res$models[[1]]

## Model for id = ModelMultiplexer.tuned class = TuneWrapper
## Trained on obs: 138
## Used features: 60
## Hyperparameters: selected.learner=classif.randomForest
##
## Optimization result:
## Tune result:
## Op. pars: selected.learner=classif.ksvm; classif.ksvm.sigma=0.0284
## mmce.test.mean=0.183
```

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)

Future Work

- Improve survival analysis and cost sensitive classification
- Connect with experiment database OpenML (`www.openml.org`)
- Support unsupervised tasks, i. e. clustering
- Support multicriteria optimization

Examples and tutorial: `https://github.com/berndbischl/mlr`