Machine Learning in R: Package `mlr`

Bernd Bischl
Computational Statistics, LMU

About

- Project home page

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

- **Tutorial** for online viewing / download, including many examples
- 8-10 main developers, quite a few contributors, 4 GSOC projects in 2015/16 and one in 2017
- About 30K lines of code, 8K lines of unit tests
- Need help? Ask on stackoverflow with tag *mlr* or open an issue

Motivation

The good news

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

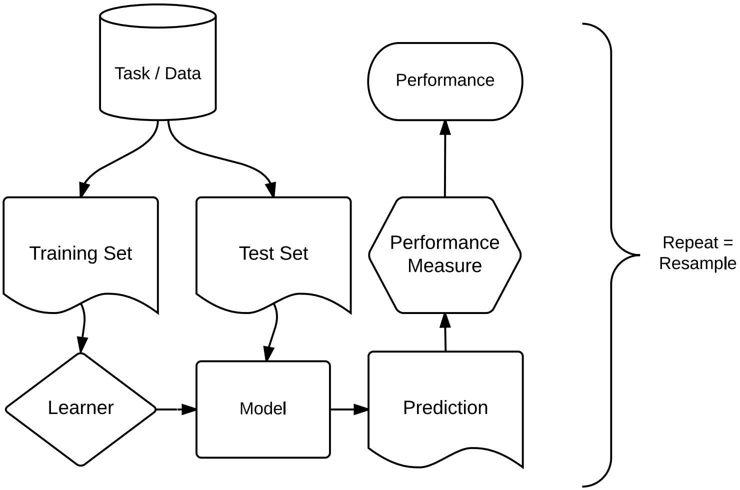
```
> model = fit(target ~ ., data = train.data, ...)
> predictions = predict(model, newdata = test.data, ...)
```

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!

Building Blocks



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

Motivation: mlr

- Clean and extensible via S3
- 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
  - Resampling
  - Tuning
  - Feature selection
  - Wrapping / Pipelining
  - Nested Resampling
  - ...

Task Abstraction

- Classification
- Regression
- Survival analysis
- Clustering
- Multi-Label
- Cost-Sensitive learning
- Functional Data

Learner Abstraction

Classification (84)

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

Regression (61)

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

Clustering (9)

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

Survival (12)

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

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

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     -
```

Resampling Abstraction

- Procedure: Train, Predict, Eval, Repeat.
- Aim: Estimate expected model performance.
  - Hold-Out
  - Cross-validation (normal, repeated)
  - Bootstrap (OOB, B632, B632+)
  - Subsampling
  - Stratification
  - Blocking
- Benchmarking / Model comparison with one command

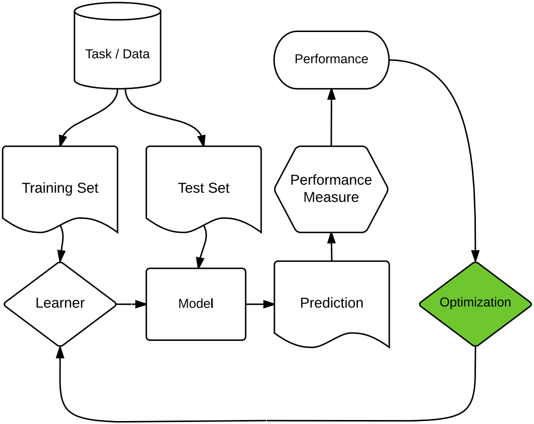
Configuring the Package

- 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

Hyperparameter Tuning

Tuning

- Find "best" hyperparameters data-dependently
- Tuner proposes config, eval by resampling, feedback to tuner

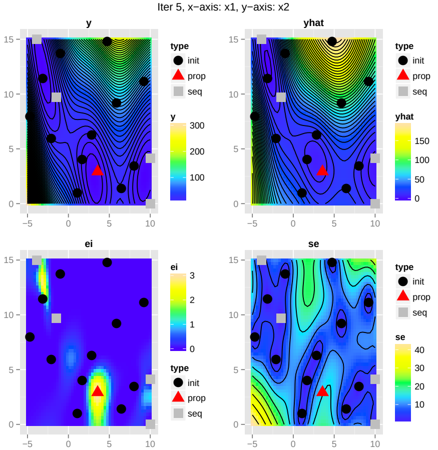


Implemented Tuning Techniques

- Grid Search
- Random Search
- Simulated Annealing
- Evolutionary Algorithms / CMAES
- Iterated F-Racing
- Model-based Optimization / Bayesian Optimization

mlrMBO: Model-Based Optimization Toolbox

- Any regression from mlr
- Arbtritrary infill
- Single - or multi-crit
- Multi-point proposal
- Via parallelMap and batchtools runs on many parallel backends and clusters
- Algorithm configuration
- Active research



- mlrMBO: `https://github.com/mlr-org/mlrMBO`
- mlrMBO Paper on arXiv (under review) `https://arxiv.org/abs/1703.03373`

Parallelization

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

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

- Backends: `local`, `multicore`, `socket`, `mpi` and `batchtools`
- The latter means support for: makeshift SSH-clusters, Docker swarm 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"

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!

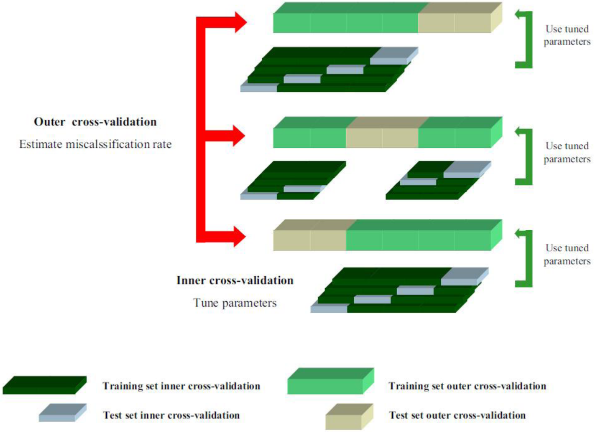
mlr Learner Wrappers II

Available Wrappers

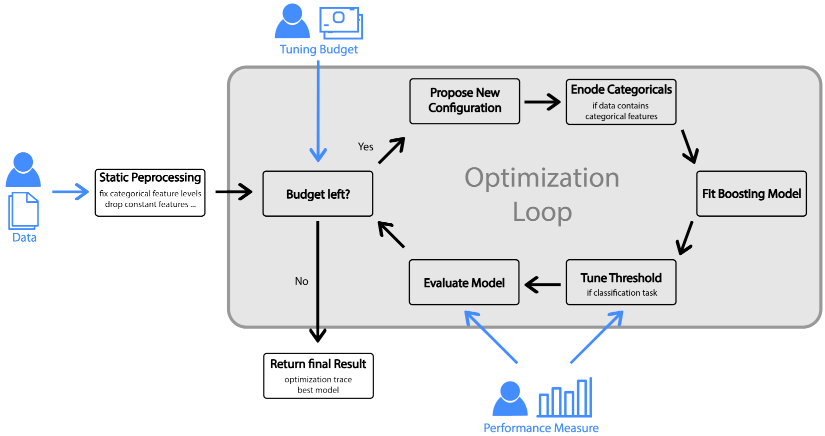
- Preprocessing: PCA, normalization (z-transformation)
- Parameter Tuning: grid, optim, random search, genetic algorithms, CMAES, iRace, MBO
- Filter: correlation- and entropy-based, $\mathcal{X}^2$-test, mRMR, ...
- Feature Selection: (floating) sequential forward/backward, exhaustive search, genetic 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



AutoXGBoost I



available @ Github: `https://github.com/ja-thomas/autoxgboost`

autoxgboost II

- Only a task is require
- Model-based hyperparameter tuning
- Threshold optimization
- Encoding of categorical features
- Handles missing values

AUTOXGBOOST III

```
> library(autoxgboost)
> titanic = read.csv("titanic_train.csv")
> task = makeClassifTask("titanic", titanic, target = "Survived")
> autoxgboost(task, iterations = 10L)

## Autoxgboost tuning result
##
## Recommended parameters:
##               eta: 0.157
##             gamma: 1.428
##         max_depth: 10
##  colsample_bytree: 0.888
## colsample_bylevel: 0.515
##            lambda: 0.003
##             alpha: 0.023
##         subsample: 0.933
##  scale_pos_weight: 0.005
##           nrounds: 5
##
##
## Preprocessing pipeline:
## (fixfactors >> dummyencode >> impact.encode.classif >> dropconst)(fixfactors.drop.unuse
##
## With tuning result: mmce = 0.123
## Classification Threshold: 0.641
```

There is more ...

- ROC and learning curves
- Imbalancy correction
- Multi-Label learning
- Multi-criteria optimization
- Ensembles, generic bagging and stacking
- ...

Outlook

We are working on

- mlr2 - nextgen
- Composable preprocessing operators: mlrCPO
- Learning defaults automatically
- ...