Machine Learning in R: Package `mlr`

Jakob Richter, TU Dortmund

About

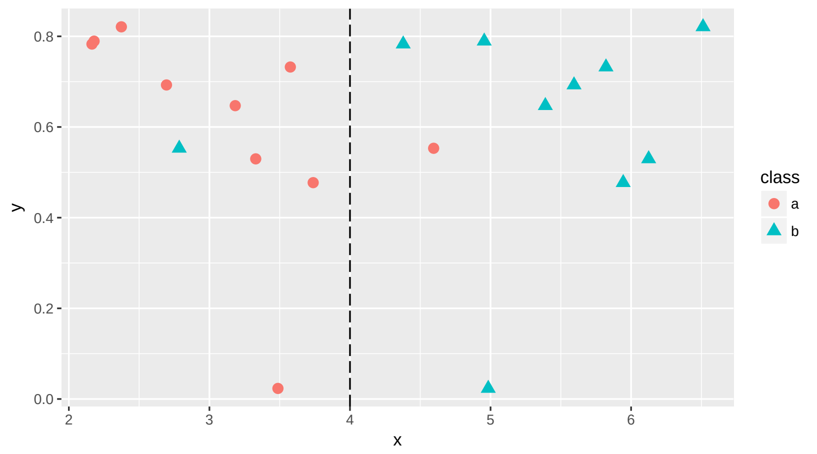
- Project home page

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

  - **Tutorial** including many examples
  - R documentation
  - Ask questions in the github issue tracker or stackoverflow
- 8-10 main developers, quite a few contributors, 5 GSOC projects sinse 2015
- About 20K lines of code, 8K lines of unit tests

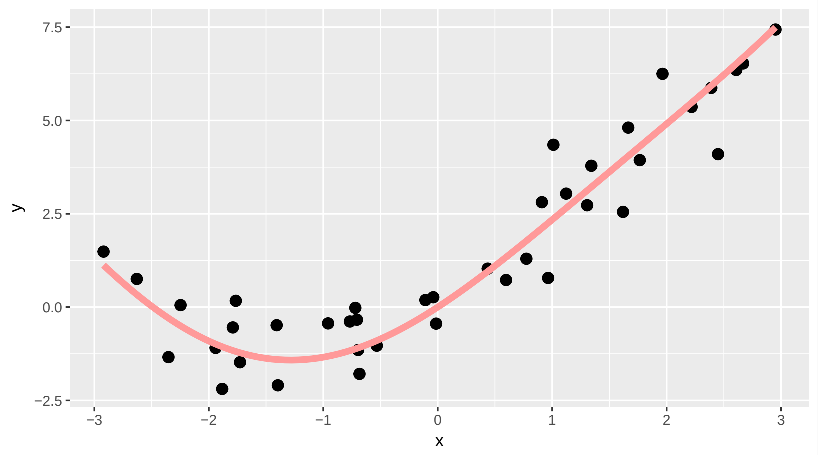
mlr

Supervised Classification tasks



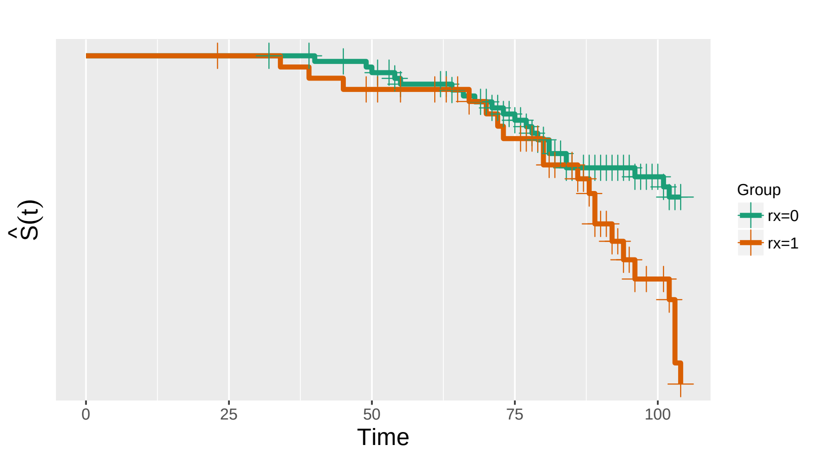
GOAL: Predict a class (or membership probabilities)

Supervised Regression tasks



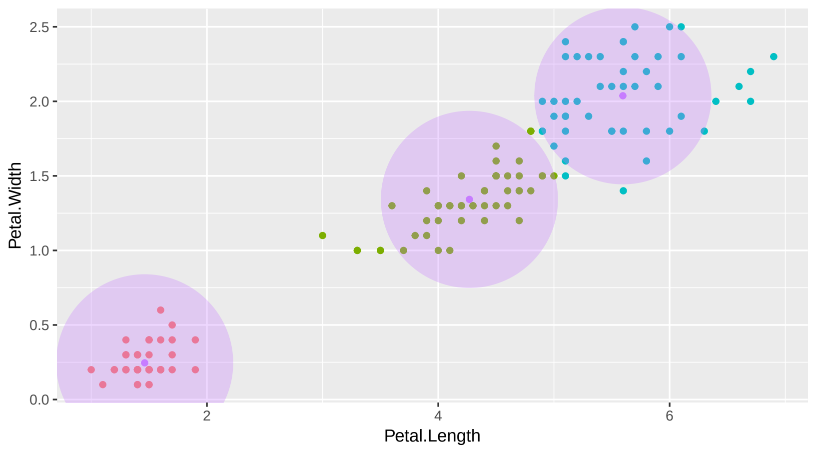
GOAL: Predict a continuous output

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)

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
- Result: 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
  - Explained in detail in the online tutorial
  - You do not need to understand S3 to use `mlr`

What Learners are available? i

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
- ...

What Learners are available? ii

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

mlr

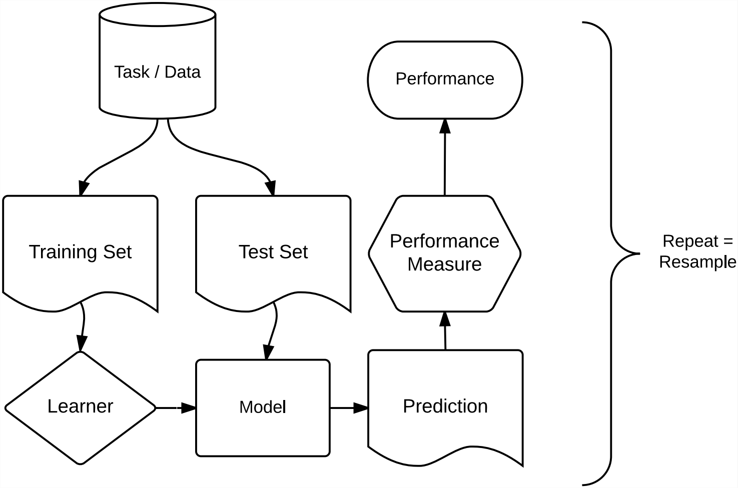
What Learners are available? iii

List all classification learners which can predict probabilities and allow multiclass classification:

```
> listLearners("classif", properties = c("prob", "multiclass"))[, c("class", "name", "factors", "missings")]

##                class                                                        name factors missings
## 1 classif.adaboostm1                                                 ada Boosting M1    TRUE    FALSE
## 2   classif.boosting                                                 Adabag Boosting    TRUE     TRUE
## 3        classif.C50                                                             C50    TRUE     TRUE
## 4     classif.cforest                       Random forest based on conditional inference trees    TRUE     TRUE
## 5       classif.ctree                                      Conditional Inference Trees    TRUE     TRUE
## 6   classif.cvglmnet GLM with Lasso or Elasticnet Regularization (Cross Validated Lambda)    TRUE    FALSE
## ... (#rows: 49, #cols: 4)
```

Building Blocks



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

Task Abstraction

- Tasks encapsulate data and meta-information about it
- Regression, classification, clustering, survival tasks

```
> task = makeClassifTask(data = iris, target = "Species")
> print(task)
```

```
## Supervised task: iris
## Type: classif
## Target: Species
## Observations: 150
## Features:
##    numerics     factors     ordered functionals
##           4           0           0           0
## Missings: FALSE
## Has weights: FALSE
## Has blocking: FALSE
## Has coordinates: 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 specific package
  - description of the parameter set, annotations, ...
- Naming convention: `<tasktype>.<functionname>`

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

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-classifica... 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     -
```

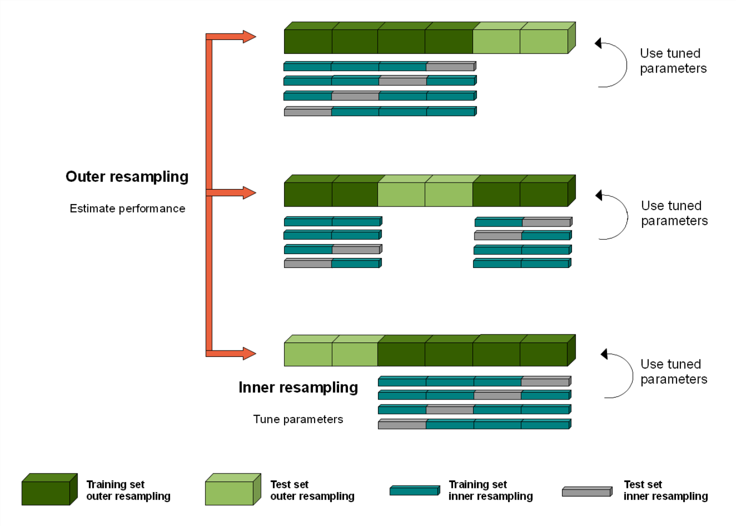
mlr

Basic Usage: Train/Predict/Evaluate

```
> #Split data in train and test data
> iris.train = iris[seq(1, 150, by = 2), ] # 1, 3, 5, 7, ... obs.
> iris.test = iris[seq(2, 150, by = 2), ] # 2, 4, 6, 8, ... obs.
>
> # create a task
> task = makeClassifTask(data = iris.train, target = "Species")
>
> # create a learner
> lrn = makeLearner("classif.rpart")
>
> # train the model
> mod = train(lrn, task)
>
> # predict the test data
> pred = predict(mod, newdata = iris.test)
>
> # evaluate performance of the model on the test data
> performance(pred, mmce)

##       mmce
## 0.05333333
```

Resampling Abstraction i



Resampling Abstraction ii

- 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 [1:50] 59 32 52 29 23 53 48 9 16 65 ...
##  $ : int [1:50] 72 23 53 14 9 10 4 40 46 70 ...
##  $ : int [1:50] 59 72 32 52 29 14 48 16 10 65 ...
```

Resampling Abstraction iii

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)
```

mlr

Resampling Abstraction iv

```
> print(r)
## Resample Result
## Task: iris.train
## Learner: classif.rpart
## Aggr perf: mmce.test.mean=0.0800000,timetrain.test.mean=0.0030000
## Runtime: 0.0253763
```

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

Performance Measures

- Performance measures evaluate the predictions a test set and aggregate them over multiple in resampling iterations
- 33 classification, 17 regression, 5 cluster, 4 survival
- 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
## Arguments: list()
## Note: Defined as: mean(response != truth)

> head(listMeasures("classif"))

## [1] "tnr"      "tpr"      "featperc" "f1"       "mmce"     "mcc"

> head(listMeasures(task))

## [1] "featperc" "mmce"     "lsr"      "bac"      "qsr"      "timeboth"
```

Benchmarking and Model Comparison i

Benchmarking

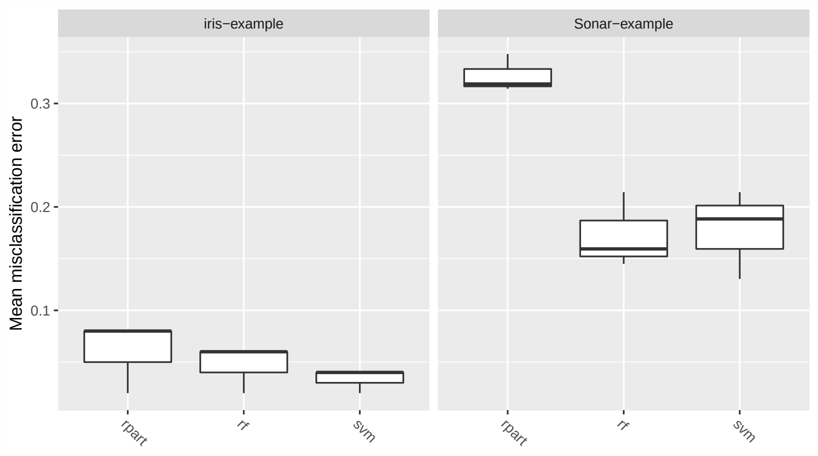
- Comparison of multiple models on multiple data sets
- Aim: Find best learners for a data set or domain, learn about learner characteristics, ...

```
> # these are predefined in mlr for toying around:
> tasks = list(iris.task, spam.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)
```



mlr

Automatic Model Selection

Prior approaches:

- Finding the unviversally best method
  $\rightsquigarrow$ Not found yet
- Exhaustive benchmarking / search
  $\rightsquigarrow$ Per data set: too expensive
  $\rightsquigarrow$ Over many: contradicting results
- Meta-Learning:
  $\rightsquigarrow$ No promising results yet
  $\rightsquigarrow$ Usually not for preprocessing / hyperparamters

Goal: Data dependent + Automatic + Efficient

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

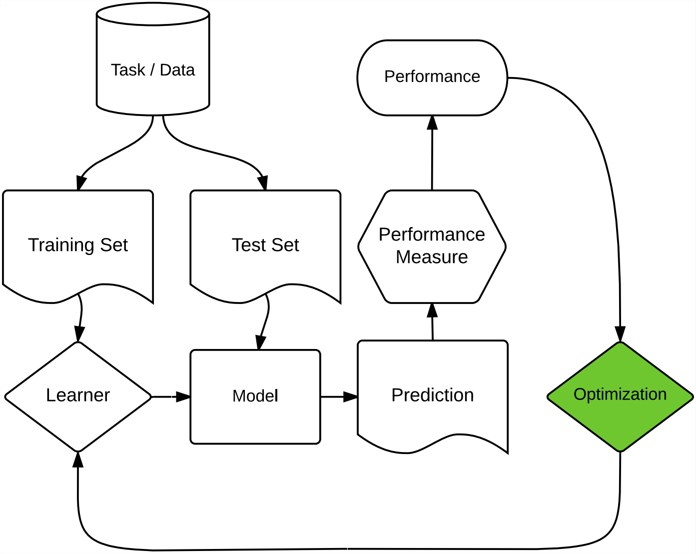
Grid search

- 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

Adaptive tuning



Tuning Example: Grid Search i

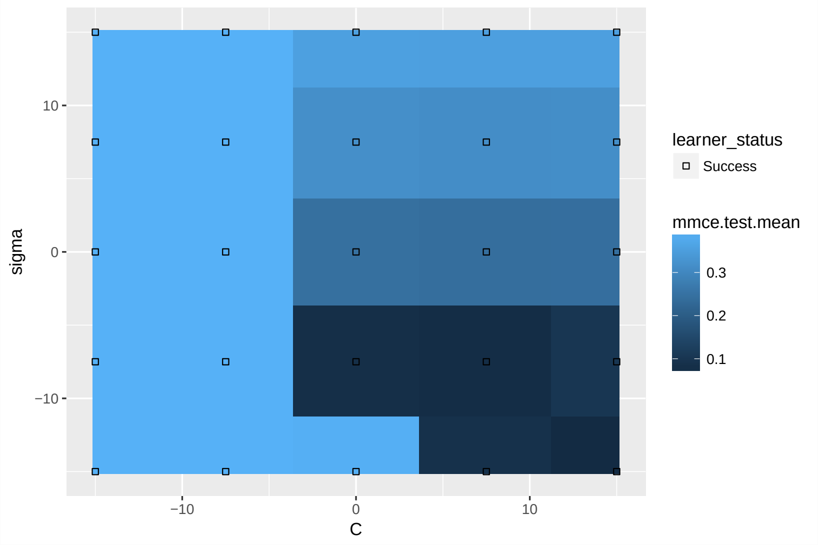
```
> ps = makeParamSet(
+   makeNumericParam("C", lower = -15, upper = 15, trafo = function(x) 2^x),
+   makeNumericParam("sigma", lower = -15, upper = 15, trafo = function(x) 2^x)
+ )
> ctrl = makeTuneControlGrid(resolution = 5)
> rdesc = makeResampleDesc("CV", iters = 2L)
> res = tuneParams("classif.ksvm", task = spam.task, control = ctrl,
+   resampling = rdesc, par.set = ps, show.info = FALSE)
> res

## Tune result:
## Op. pars: C=3.28e+04; sigma=3.05e-05
## mmce.test.mean=0.0693331


> pe = mlr::generateHyperParsEffectData(res)
> plotHyperParsEffect(pe, "C", "sigma", z = "mmce.test.mean", plot.type = "heatmap",
+   interpolate = makeLearner("regr.kknn", k = 1), show.experiments = TRUE)
```

mlr

Tuning Example: Grid Search ii



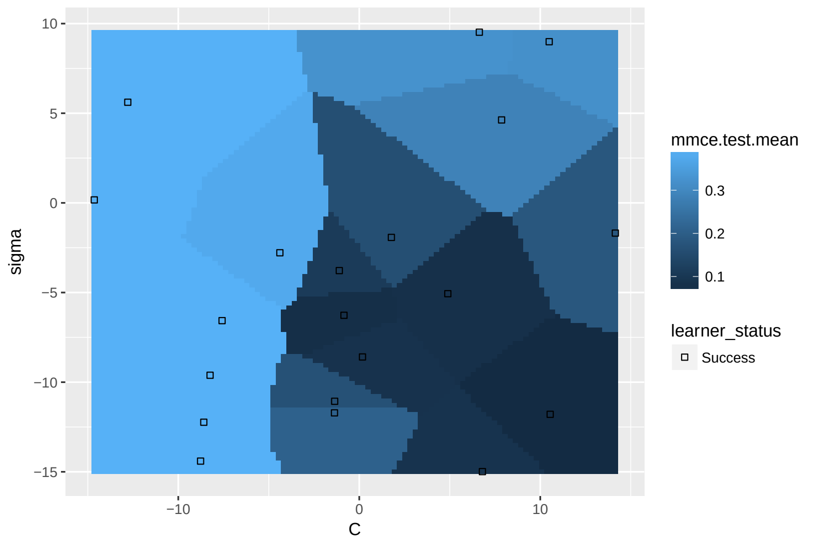
Tuning Example: Random Search i

```
> ps = makeParamSet(
+   makeNumericParam("C", lower = -15, upper = 15, trafo = function(x) 2^x),
+   makeNumericParam("sigma", lower = -15, upper = 15, trafo = function(x) 2^x)
+ )
> ctrl = makeTuneControlRandom(maxit = 20L)
> rdesc = makeResampleDesc("CV", iters = 2L)
> res = tuneParams("classif.ksvm", task = spam.task, control = ctrl,
+   resampling = rdesc, par.set = ps, show.info = FALSE)
> res

## Tune result:
## Op. pars: C=1.5e+03; sigma=0.000282
## mmce.test.mean=0.0649839


> pe = mlr::generateHyperParsEffectData(res)
> plotHyperParsEffect(pe, "C", "sigma", z = "mmce.test.mean", plot.type = "heatmap",
+   interpolate = makeLearner("regr.kknn", k = 1), show.experiments = TRUE)
```

Tuning Example: Random Search ii



Baysisan Optimization of Hyperparameters with `mlrMBO` i

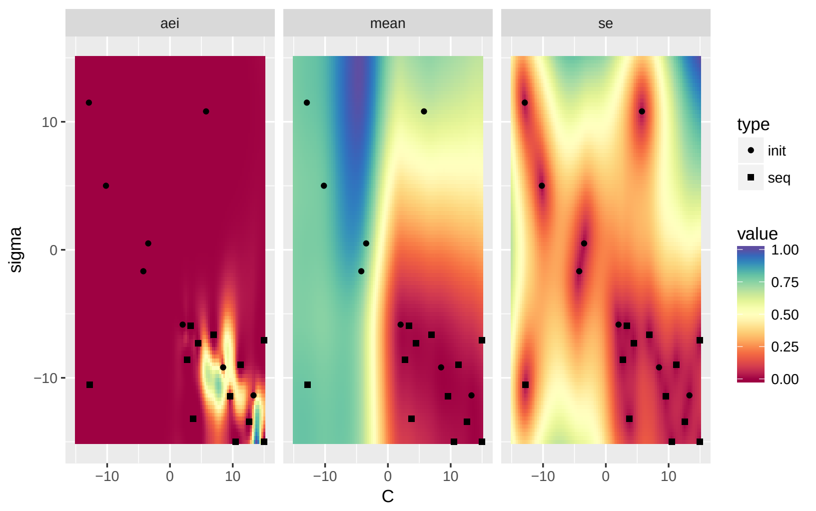
- State-of-the-art tuning for expensive Problems
- Multi-criterial (e.g. FPR vs. TNR)
- Parallelization by Multi-point proposals
- Tutorial and Examples: `https://mlr-org.github.io/mlrMBO/`

```
> library(mlrMBO)
> ps = makeParamSet(
+   makeNumericParam("C", lower = -15, upper = 15, trafo = function(x) 2^x),
+   makeNumericParam("sigma", lower = -15, upper = 15, trafo = function(x) 2^x)
+ )
> mbo.ctrl = setMBOControlInfill(makeMBOControl(), crit = crit.aei)
> ctrl = makeTuneControlMBO(budget = 20L, mbo.control = mbo.ctrl)
> rdesc = makeResampleDesc("CV", iters = 2L)
> (res = tuneParams("classif.ksvm", task = spam.task, control = ctrl,
+   resampling = rdesc, par.set = ps, show.info = FALSE))

## Tune result:
## Op. pars: C=355; sigma=0.00172
## mmce.test.mean=0.0649849


> plot(res$mbo.result$final.opt.state, scale.panels = TRUE)
```

Baysisan Optimization of Hyperparameters with `mlrMBO` ii



Tuning Example: mlrHyperopt

- R-Package mlrHyperopt for effortless tuning
- Documentation: `http://jakob-r.github.io/mlrHyperopt`
- No knowledge of parameters needed.
- Decides automatically for suitable tuning method.

```
> library(mlrHyperopt)
> res = hyperopt(task = spam.task, learner = "classif.ksvm")
> res

## Tune result:
## Op. pars: C=281; sigma=0.00177
## mmce.test.mean=0.0599887
```

Model Multiplexer

The model multiplexer allows for tuning over multiple learners!

```
> bls = list(
+   makeLearner("classif.ksvm"),
+   makeLearner("classif.randomForest")
+ )
> lrn = makeModelMultiplexer(bls)
> ps = makeModelMultiplexerParamSet(lrn,
+   makeNumericParam("sigma", lower = -15, upper = 15, trafo = function(x) 2^x),
+   makeNumericParam("C", lower = -15, upper = 15, trafo = function(x) 2^x),
+   makeIntegerParam("mtry", lower = 1L, upper = 8L)
+ )
> rdesc = makeResampleDesc("CV", iters = 2L)
> ctrl = makeTuneControlIrace(maxExperiments = 120L)
> res = tuneParams(lrn, spam.task, rdesc, par.set = ps, control = ctrl)
> res

## Tune result:
## Op. pars: selected.learner=classif.rand...; classif.randomForest.mtry=6
## mmce.test.mean=0.0522484
```

mlr

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!

Available Wrappers

- Preprocessing: PCA, normalization, dummy encoding, ...
- 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

mlr Learner Wrappers iii

```
> set.seed(1)
> library(ggplot2); library(RColorBrewer)
> lrn = makeLearner("classif.randomForest", ntree = 200)
> lrn = makeRemoveConstantFeaturesWrapper(learner = lrn)
> lrn = makeDownsampleWrapper(learner = lrn)
> lrn = makeFilterWrapper(lrn, fw.method = "gain.ratio")
> filterParams(getParamSet(lrn), tunable = TRUE, type = c("numeric", "integer"))
##                   Type len Def      Constr Req Tunable Trafo
## fw.perc        numeric   -   -      0 to 1   -    TRUE     -
## fw.abs         integer   -   -    0 to Inf   -    TRUE     -
## fw.threshold numeric   -   - -Inf to Inf   -    TRUE     -
## dw.perc        numeric   -   1      0 to 1   -    TRUE     -
## ntree          integer   - 500    1 to Inf   -    TRUE     -
## mtry           integer   -   -    1 to Inf   -    TRUE     -
## nodesize       integer   -   1    1 to Inf   -    TRUE     -
## maxnodes       integer   -   -    1 to Inf   -    TRUE     -
```

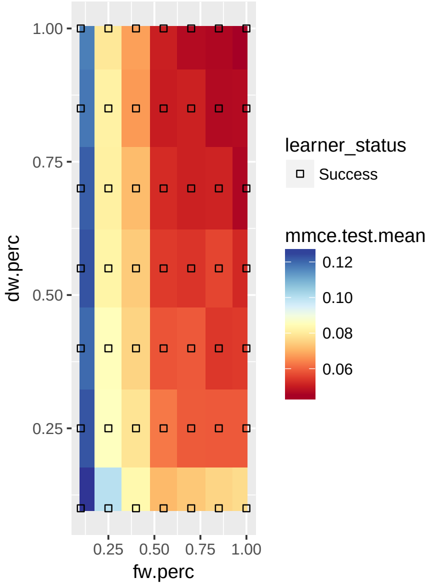
mlr Learner Wrappers iv

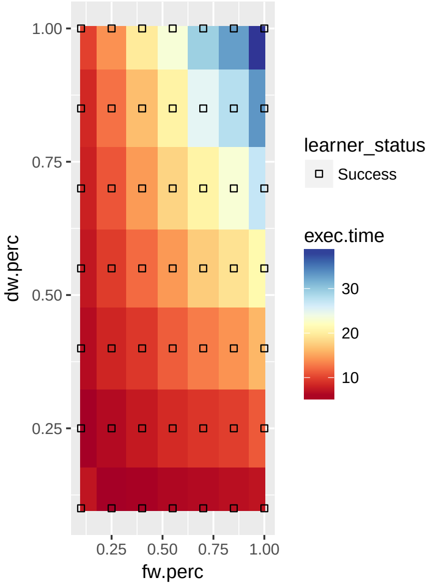
```
> ps = makeParamSet(
+   makeNumericParam("fw.perc", lower = 0.1, upper = 1),
+   makeNumericParam("dw.perc", lower = 0.1, upper = 1))
> res = tuneParams(lrn, spam.task, resampling = cv10, par.set = ps,
+   control = makeTuneControlGrid(resolution = 7), show.info = FALSE)
> res

## Tune result:
## Op. pars: fw.perc=1; dw.perc=1
## mmce.test.mean=0.0447736
```

```
> pe = generateHyperParsEffectData(res)
> brewer.div = colorRampPalette(brewer.pal(11, "RdYlBu"), interpolate = "spline")
> plotHyperParsEffect(pe, "fw.perc", "dw.perc", z = "mmce.test.mean", plot.type = "heatmap"
+   interpolate = makeLearner("regr.kknn", k = 1), show.experiments = TRUE) +
+   scale_fill_gradientn(colours = brewer.div(200))
> plotHyperParsEffect(pe, "fw.perc", "dw.perc", z = "exec.time", plot.type = "heatmap",
+   interpolate = makeLearner("regr.kknn", k = 1), show.experiments = TRUE) +
+   scale_fill_gradientn(colours = brewer.div(200))
```

`mlr` Learner Wrappers v





Moving on with mlr

- Learn all details in the tutorial:
  `https://mlr-org.github.io/mlr/`
- Book a Machine Learning in R Course:
  `http://dortmunder-r-kurse.de/kurse/machine-learning-in-r/`
- Ask general questions in stackoverflow: `https://stackoverflow.com/questions/tagged/mlr`
- Found bugs? Report them:
  `https://github.com/mlr-org/mlr/issues`
- Want to contribute? Join our slack:
  `https://mlr-org.slack.com/`.