Machine Learning in R: Package mlr

Janek Thomas, Bernd Bischl
Computational Statistics, LMU

About

- Project home page

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

  - **Tutorial** for online viewing / download, including many examples
  - R documentation rendered in HTML
  - If you are interested you can ask questions in the github issue tracker
- 8-10 main developers, quite a few contributors, 4 GSOC projects in 2015/16 and one coming in 2017
- About 20K lines of code, 8K lines of unit tests

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

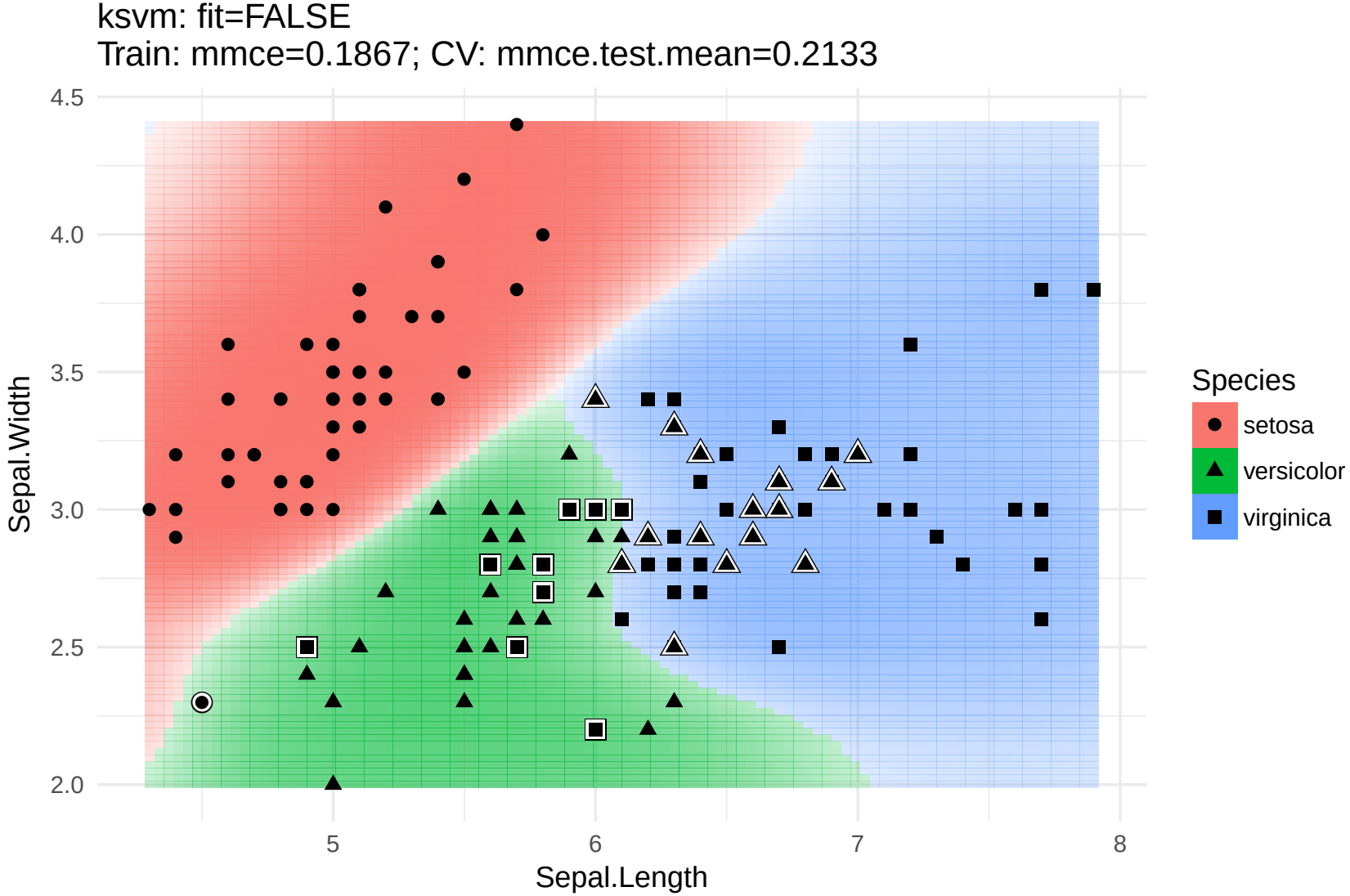
- Some packages API is "just different"
- Functionality is always package or model-dependent, even though the procedure might be general
- No meta-information available or buried in docs

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

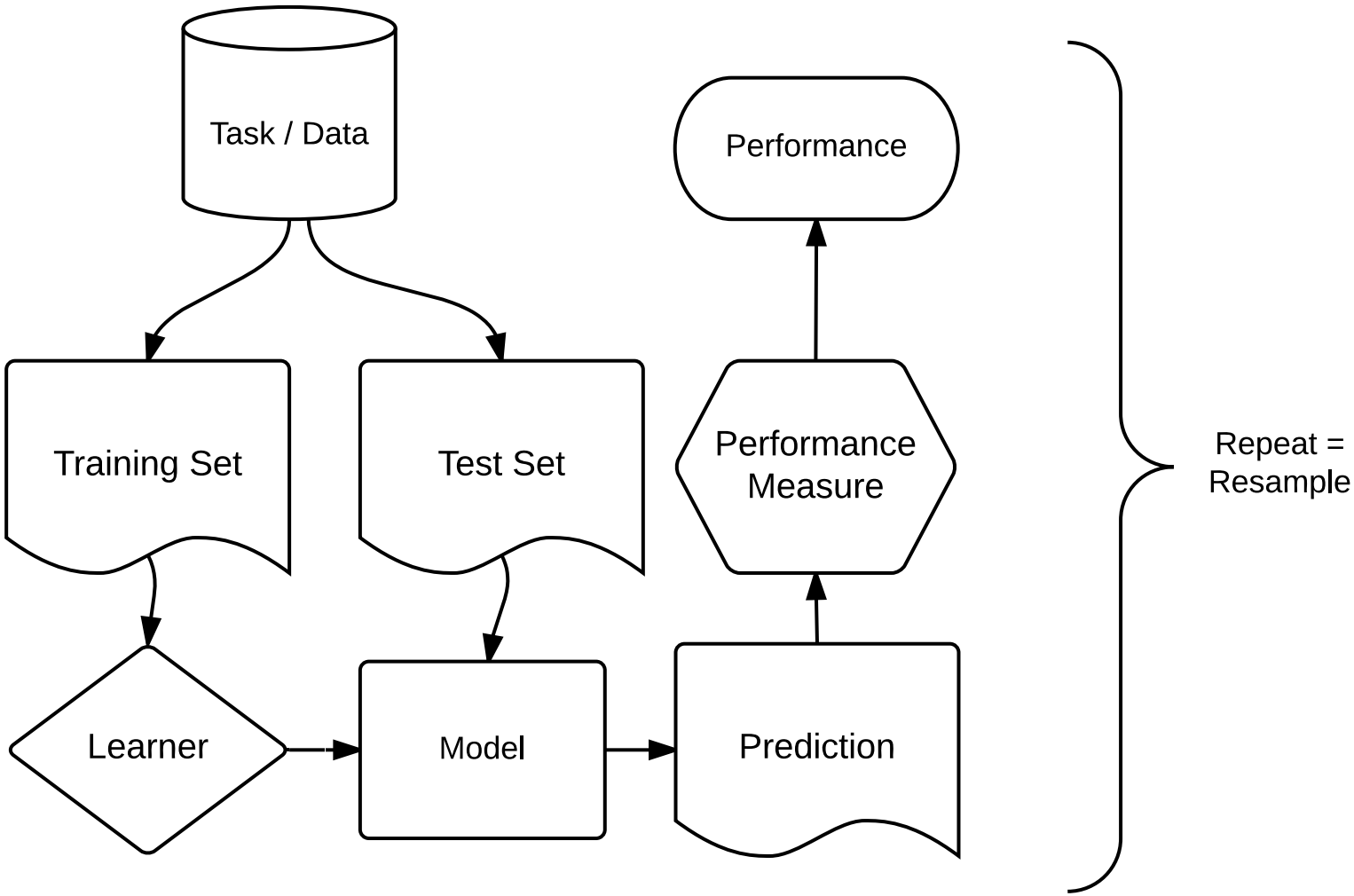
mlr

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.


ksvm: fit=FALSE
Train: mmce=0.1867; CV: mmce.test.mean=0.2133
Sepal.Width
Sepal.Length
4.5
4.0
3.5
3.0
2.5
2.0
5
6
7
8
Species
setosa
versicolor
virginica

Building Blocks



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

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
- ...

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

What Learners are available? II

```
> # list all classification learners which can predict probabilities
> # and allow multiclass classification
> listLearners("classif",
+   properties = c("prob", "multiclass"))[1:5, c(-2, -5, -16)]

##                class short.name      package    type installed
## 1 classif.adaboostm1 adaboostm1        RWeka classif      TRUE
## 2   classif.boosting     adabag adabag,rpart classif      TRUE
## 3        classif.C50        C50          C50 classif      TRUE
## 4    classif.cforest    cforest        party classif      TRUE
## 5      classif.ctree      ctree        party classif      TRUE
##   numerics factors ordered missings weights prob oneclass twoclass
## 1     TRUE    TRUE   FALSE    FALSE   FALSE TRUE    FALSE     TRUE
## 2     TRUE    TRUE   FALSE     TRUE   FALSE TRUE    FALSE     TRUE
## 3     TRUE    TRUE   FALSE     TRUE    TRUE TRUE    FALSE     TRUE
## 4     TRUE    TRUE    TRUE     TRUE    TRUE TRUE    FALSE     TRUE
## 5     TRUE    TRUE    TRUE     TRUE    TRUE TRUE    FALSE     TRUE
##   class.weights featimp oobpreds functionals single.functional    se
## 1         FALSE   FALSE    FALSE       FALSE             FALSE FALSE
## 2         FALSE    TRUE    FALSE       FALSE             FALSE FALSE
## 3         FALSE   FALSE    FALSE       FALSE             FALSE FALSE
## 4         FALSE    TRUE    FALSE       FALSE             FALSE FALSE
## 5         FALSE   FALSE    FALSE       FALSE             FALSE FALSE
##   lcens rcens icens
## 1 FALSE FALSE FALSE
## 2 FALSE FALSE FALSE
## 3 FALSE FALSE FALSE
## 4 FALSE FALSE FALSE
## 5 FALSE FALSE FALSE
## ... (#elements: 22)
```

Parameter Abstraction

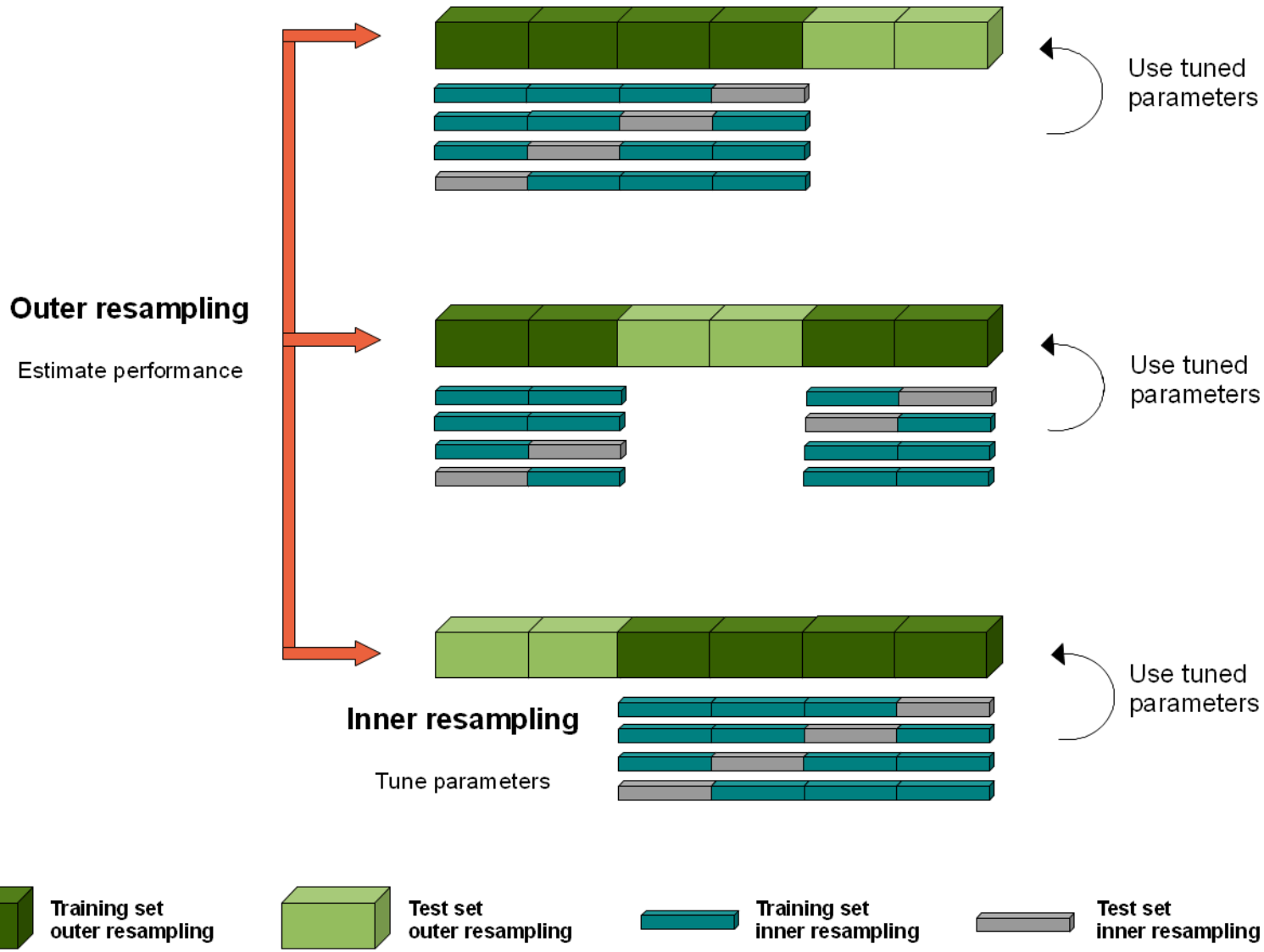
- Extensive meta-information for hyperparameters available: storage type, constraints, defaults, dependencies
- Automatically checked for feasibility
- You can program on parameters!

```
> 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     -
```

Basic Usage: Train/Predict/Evaluate I

```
> #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
## 0.05333
```

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 [1:100] 70 116 84 17 149 52 57 72 106 139 ...
##  $ : int [1:100] 98 70 116 149 52 57 106 95 131 81 ...
##  $ : int [1:100] 98 84 17 72 139 94 131 135 129 81 ...
```

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)
```

Resampling Abstraction IV

```
> print(r)

## Resample Result
## Task: iris
## Learner: classif.rpart
## Aggr perf: mmce.test.mean=0.0667,timetrain.test.mean=0.0060
## Runtime: 0.0424366
```

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

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, ...

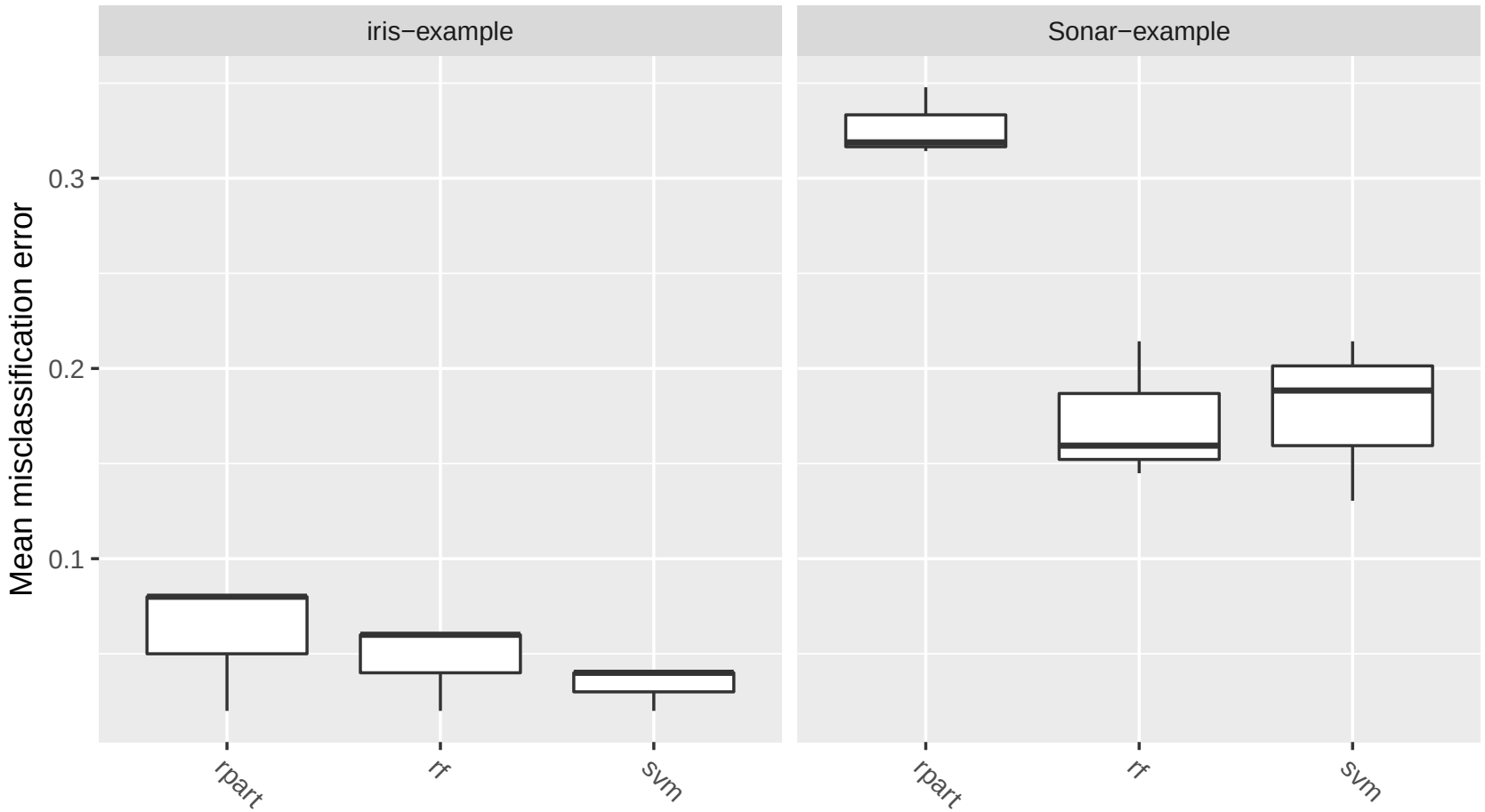
```
> # these are predefined in mlr for toying around:
> tasks = list(iris.task, sonar.task)
> learners = list(
+   makeLearner("classif.rpart"),
+   makeLearner("classif.randomForest", ntree = 500),
+   makeLearner("classif.svm")
+ )
>
> rdesc = makeResampleDesc("CV", iters = 3)
> br = benchmark(learners, tasks, rdesc)
```

Container object: Results, individual predictions, ...

mlr

Benchmarking and Model Comparison II

```
> plotBMRBoxplots(br)
```



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
  $\leadsto$ Inefficient, combinatorial explosion, searches irrelevant areas

Random search

- Randomly draw parameters
  $\leadsto$ Scales better then grid search, easily extensible

mlr

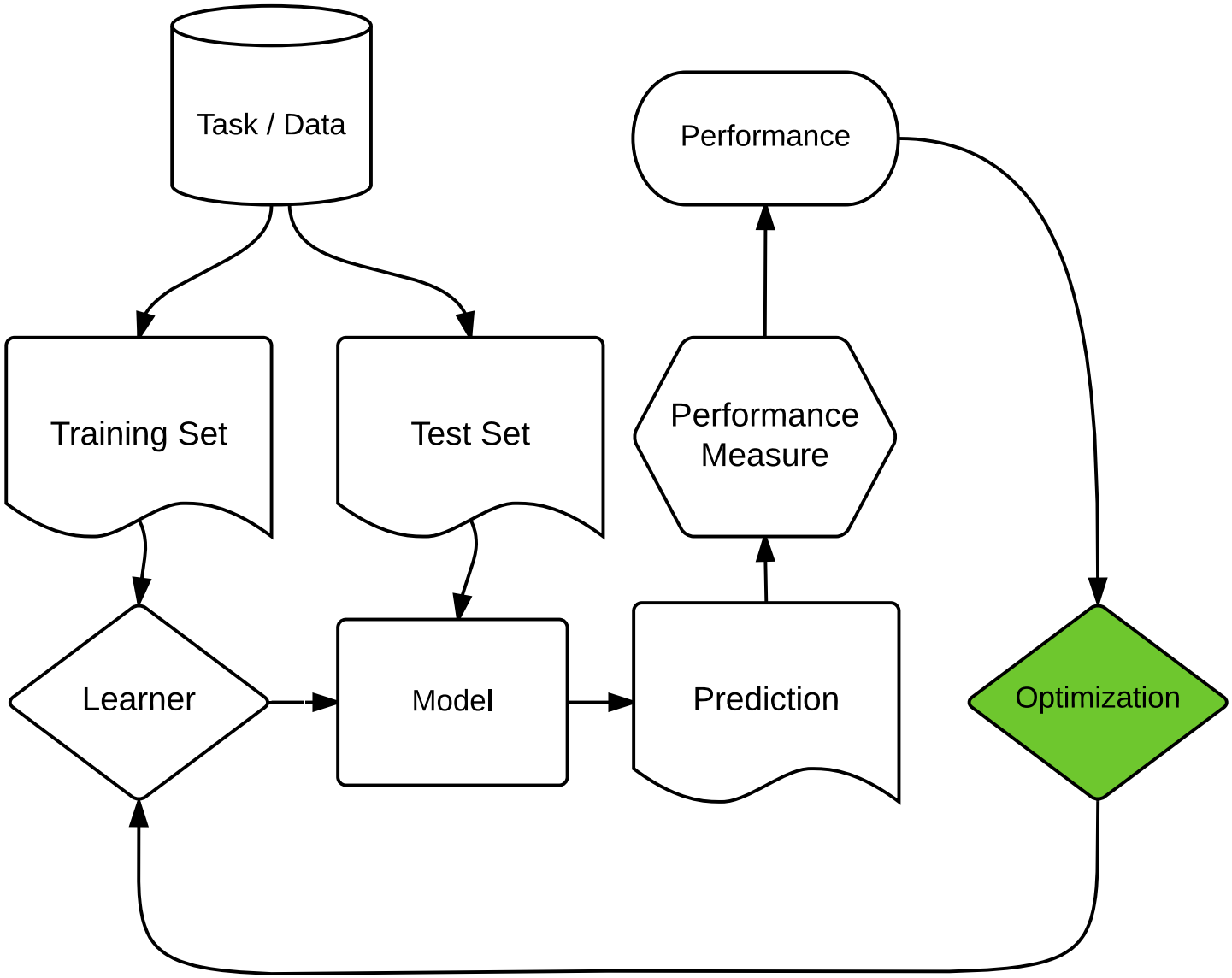
Automatic Model Selection

Prior approaches:

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

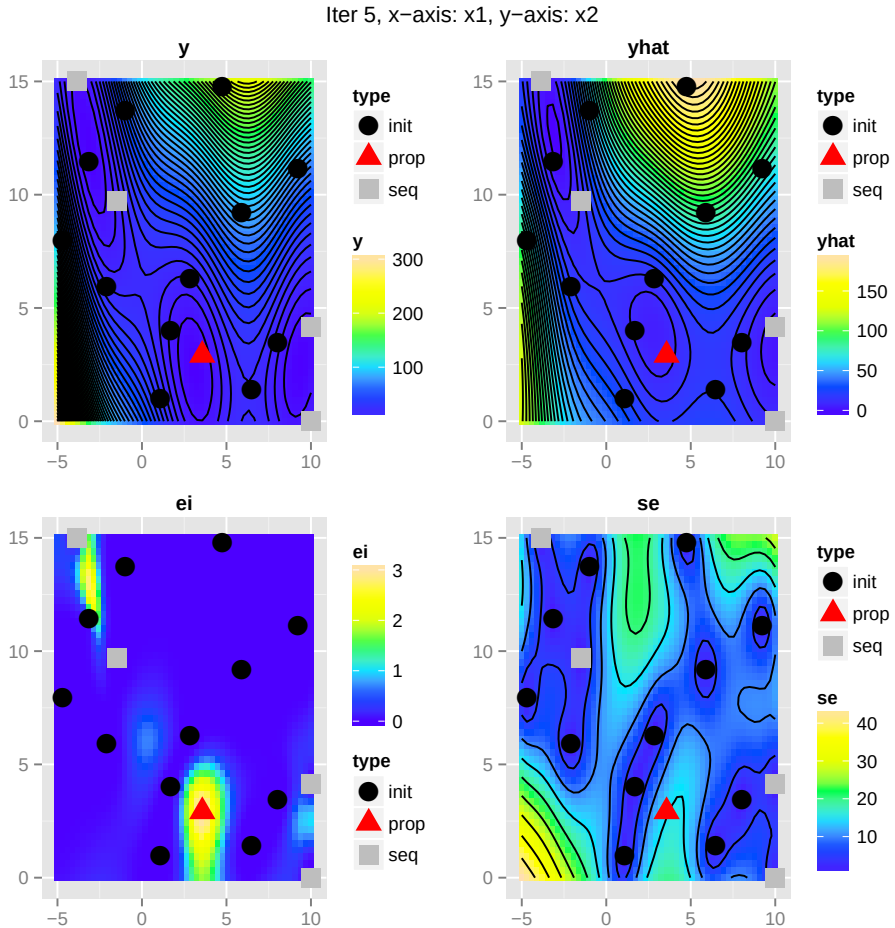
Goal: Data dependent + Automatic + Efficient

Adaptive tuning



mlrMBO: Model-Based Optimization Toolbox

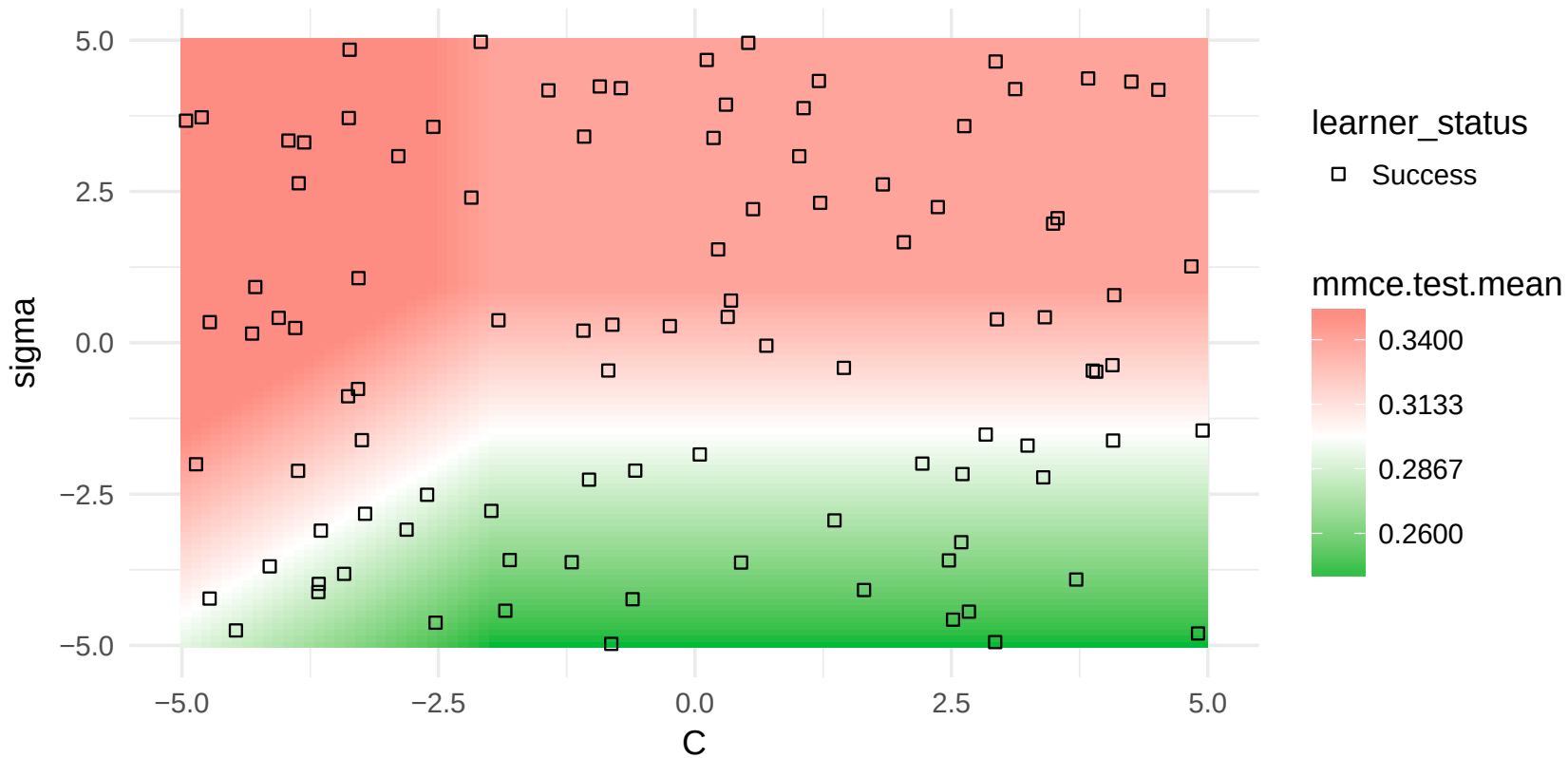
- Any regression from mlr
- Arbtritrary infill
- Single - or multi-crit
- Multi-point proposal
- Via parallelMap and batchtools runs on many parallel backends and clusters
- Algorithm configuration
- Active research



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

Tuning Example

```
> ps = makeParamSet(
+   makeNumericParam("C", lower = -5, upper = 5, trafo = function(x) 2^x),
+   makeNumericParam("sigma", lower = -5, upper = 5, trafo = function(x) 2^x)
+ )
> ctrl = makeTuneControlRandom(maxit = 100L)
> rdesc = makeResampleDesc("CV", iters = 2L)
> res = tuneParams("classif.ksvm", task = pid.task, control = ctrl,
+   resampling = rdesc, par.set = ps, show.info = FALSE)
```



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"

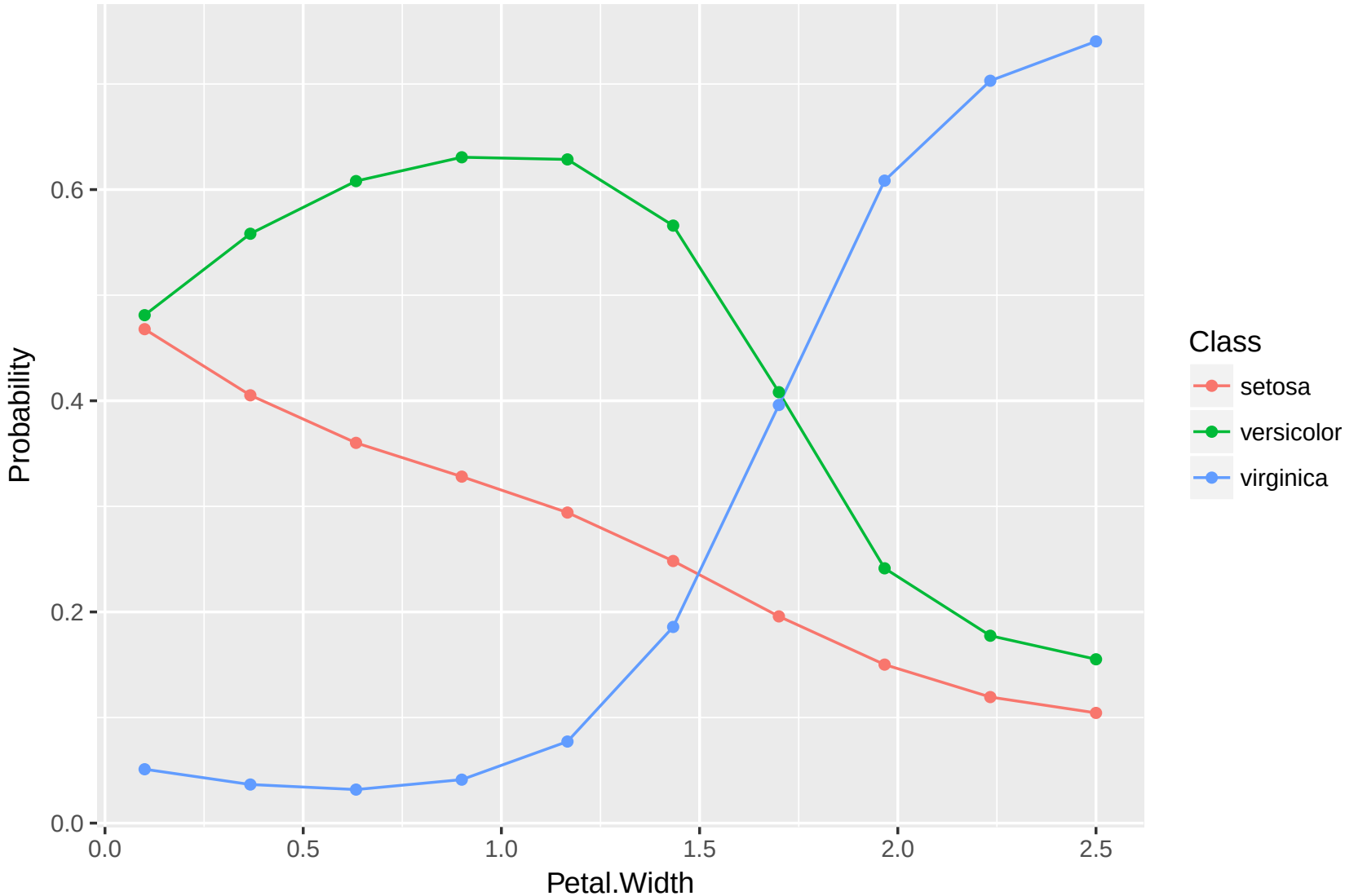
Partial Predictions Plots I

Partial Predictions

- Estimate how the learned prediction function is affected by one or more features.
- Displays marginalized version of the predictions of one or multiple effects.
- Reduce high dimensional function estimated by the learner.

```
> library(kernlab)
> lrn.classif = makeLearner("classif.svm", predict.type = "prob")
> fit.classif = train(lrn.classif, iris.task)
> pd = generatePartialDependenceData(fit.classif, iris.task, "Petal.Width")
>
> plotPartialDependence(pd)
```

Partial Predictions Plots II



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!

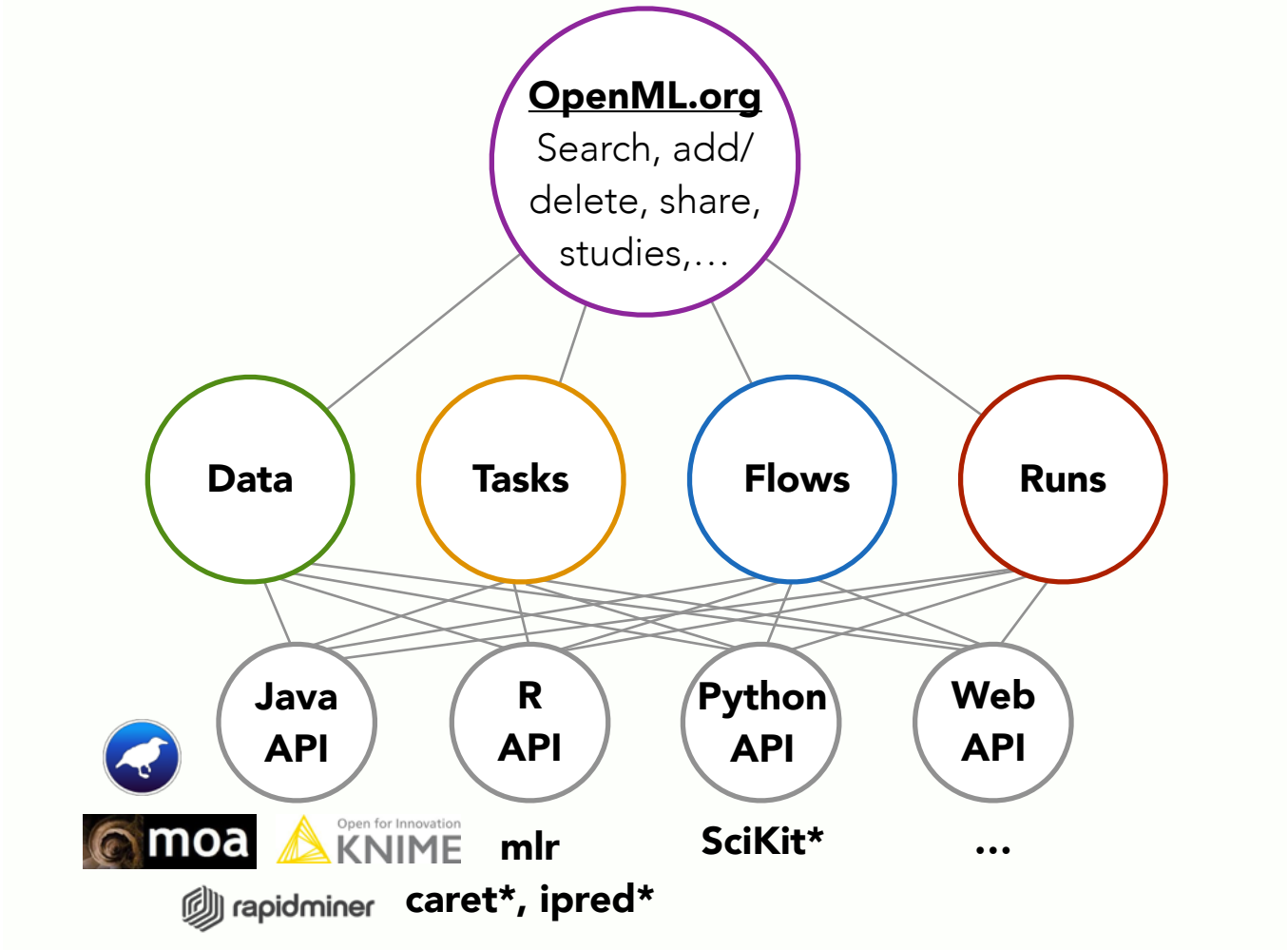
mlr Learner Wrappers II

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

OpenML

Main idea: Make ML experiments reproducible, computer-readable and allow collaboration with others.



OpenML R-Package I

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

Tutorial

- `http://openml.github.io/openml-r`
- Caution: Work in progress

Current API in R

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

There is more ...

- Clustering and Survival analysis
- Regular cost-sensitive learning (class-specific costs)
- Cost-sensitive learning (example-dependent costs)
- ROC and learning curves
- Imbalancy correction
- Multi-Label learning
- Bayesian optimization
- Multi-criteria optimization
- Ensembles, generic bagging and stacking
- Some interactive plots with `ggvis`
- ...

Outlook

We are working on

- Even better tuning system
- Composable preprocessing operators: mlrCPO
- More interactive and 3D plots
- Large-Scale learning on databases
- Keeping the data on hard disk & distributed storage
- Time-Series tasks
- Large-Scale usage of OpenML
- `auto-mlr`
- ...

Thanks!

Useful Links:

- https://github.com/mlr-org/mlr
- https://mlr-org.github.io/
- https://mlr-org.github.io/mlr-tutorial/devel/html/
- https://github.com/mlr-org/mlrCPO
- https://github.com/mlr-org/mlrMBO
- https://github.com/openml/openml-r