Machine Learning in R: Package mlr

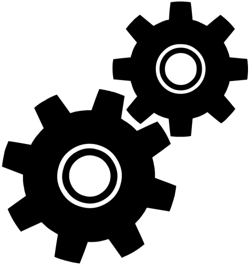
Bernd Bischl
Computational Statistics, LMU

Workshop documentation

goo.gl/DYzSmA

Agenda

- About `mlr`
- Features of `mlr`
  - Tasks and Learners
  - Train, Test, Resample
  - Benchmarking
  - Hyperparameter Tuning
  - Nested Resampling
  - Performance Visualization
  - Parallelization
- `iml` - Interpretable Machine Learning
- `mlrMBO` - Bayesian Optimization
- `mlrCPO` - Composable Preprocessing
- OpenML
- Outlook and `mlr` contribution



Machine Learning is a method of teaching computers to make predictions based on some data.

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!

About

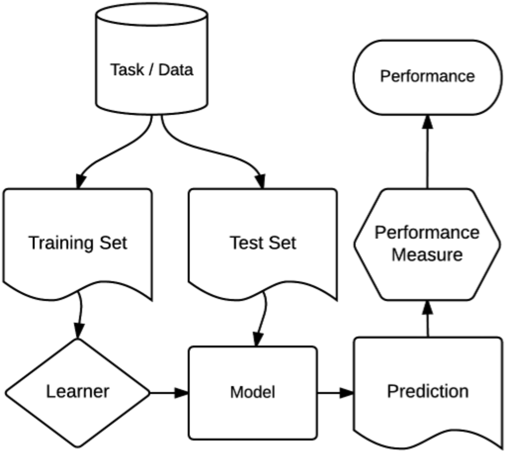
- Project home page

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

  - Cheatsheet for an quick overview
  - Tutorial for mlr documentation with many code examples
  - 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: mlr

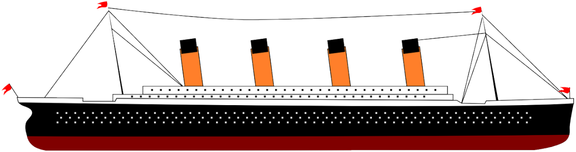
- Unified interface for the basic building blocks: tasks, learners, hyperparameters, ...



The mlr process

Titanic: Machine Learning from Disaster

- Titanic sinking on April 15, 1912
- Data provided on our website `goo.gl/DYzSmA`
- 809 out of 1309 passengers got killed
- Task
  - Can we predict who survived?
  - Why did people die / Which groups?



R Example: Data set

- Data Dictionary

`Survived`	Survived, 0 = No, 1 = Yes
`Pclass`	Ticket class, from 1st to 3rd
`Sex`	Sex
`Age`	Age in years
`Sibsp`	# of siblings/ spouses
`Parch`	# of parents/ children
`Ticket`	Ticket number
`Fare`	Passenger fare
`Cabin`	Cabin number
`Embarked`	Port of Embarkation

Preprocessing I

- Load the input data

```
> load("data.rda")
> print(summarizeColumns(data)[, -c(5, 6, 7)], digits = 0)

##         name      type  na mean min  max nlevs
## 1     Pclass    factor   0   NA 277  709     3
## 2   Survived    factor   0   NA 500  809     2
## 3       Name character   0   NA   1    2  1307
## 4        Sex    factor   0   NA 466  843     2
## 5        Age   numeric 263   30   0   80     0
## 6      Sibsp   numeric   0    0   0    8     0
## 7      Parch   numeric   0    0   0    9     0
## 8     Ticket    factor   0   NA   1   11   929
## 9       Fare   numeric   1   33   0  512     0
## 10     Cabin    factor   0   NA   1 1014   187
## 11 Embarked    factor   0   NA   2  914     4
```

Preprocessing II

- NB: All preprocessing steps are really naive, later we show better preprocessing with `mlrCPO`
- Set empty factor levels to NA

```
> data$Embarked[data$Embarked == ""] = NA
> data$Embarked = droplevels(data$Embarked)
> data$Cabin[data$Cabin == ""] = NA
> data$Cabin = droplevels(data$Cabin)
```

Preprocessing III

```r
> # Price per person, multiple tickets bought by one
> # person
> data$farePp = data$Fare / (data$Parch + data$Sibsp + 1)
>
> # The deck can be extracted from the the cabin number
> data$deck = as.factor(stri_sub(data$Cabin, 1, 1))
>
> # Starboard had an odd number, portside even cabin
> # numbers
> data$portside = stri_sub(data$Cabin, 3, 3)
> data$portside = as.numeric(data$portside) %% 2
>
> # Drop stuff we cannot easily model on
> data = dropNamed(data,
+   c("Cabin","PassengerId", "Ticket", "Name"))
```

Preprocessed data

```
> print(summarizeColumns(data)[, -c(5, 6, 7)], digits = 0)

##        name    type   na mean min max nlevs
## 1    Pclass  factor    0   NA 277 709     3
## 2  Survived  factor    0   NA 500 809     2
## 3       Sex  factor    0   NA 466 843     2
## 4       Age numeric  263   30   0  80     0
## 5     Sibsp numeric    0    0   0   8     0
## 6     Parch numeric    0    0   0   9     0
## 7      Fare numeric    1   33   0 512     0
## 8  Embarked  factor    2   NA 123 914     3
## 9    farePp numeric    1   21   0 512     0
## 10     deck  factor 1014   NA   1  94     8
## 11 portside numeric 1059    0   0   1     0
```

Imputation

- Remove missing values
- Impute numerics with median and factors with a seperate category
- NB: This is really naive, we should probably use multiple imputation and embed this in cross-valdiation

```
> data = impute(data, cols = list(
+   Age = imputeMedian(),
+   Fare = imputeMedian(),
+   Embarked = imputeConstant("__miss__"),
+   farePp = imputeMedian(),
+   deck = imputeConstant("__miss__"),
+   portside = imputeConstant("__miss__")
+ ))
>
> data = data$data
> data = convertDataFrameCols(data, chars.as.factor = TRUE)
```

Tasks I

- Create classification problem

```
> task = makeClassifTask(id = "titanic", data = data,
+   target = "Survived", positive = "1")
```

Tasks II

```
> print(task)

## Supervised task: titanic
## Type: classif
## Target: Survived
## Observations: 1309
## Features:
##    numerics     factors     ordered functionals
##           5           5           0           0
## Missings: FALSE
## Has weights: FALSE
## Has blocking: FALSE
## Has coordinates: FALSE
## Classes: 2
##   0   1
## 809 500
## Positive class: 1
```

What Learners are available? I

Classification (84)

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

Regression (61)

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

Clustering (9)

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

Survival (12)

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

What Learners are available? II

- Explore all learners via tutorial

mlr 2.13 Get Started Basics Advanced Extending Appendix mlr-org Packages Search...

Class / Short Name / Name	Packages	Num.	Fac.	Ord.	NAs	Weights	Props	Note
classif.ada *ada* ada Boosting	ada rpart	X	X				prob twoclass	`xval` has been set to `0` by default for spee
classif.adaboostm1 *adaboostm1* ada Boosting M1	RWeka	X	X				prob twoclass multiclass	NAs are directly passed to WEKA with `na.ac`
classif.bartMachine *bartmachine* Bayesian Additive Regression Trees	bartMachine	X	X		X		prob twoclass	`use_missing_data` has been set to `TRUE`
classif.binomial *binomial* Binomial Regression	stats	X	X			X	prob twoclass	Delegates to `glm` with freely choosable bin

What Learners are available? III

- Or ask `mlr`

```
> listLearners("classif", properties = c("prob",
+   "multiclass"))[1:5, c(1,4,13,16)]

##                   class       package prob multiclass
## 1 classif.adaboostm1         RWeka TRUE       TRUE
## 2   classif.boosting adabag,rpart TRUE       TRUE
## 3        classif.C50           C50 TRUE       TRUE
## 4    classif.cforest         party TRUE       TRUE
## 5      classif.ctree         party TRUE       TRUE
```

Train model I

- Create a learner
- Output prosterior probs – instead of a factor of class labels

```
> lrn = makeLearner("classif.randomForest",
+   predict.type = "prob")
```

- Split data into a training and test data set (neccessary for performance evaluation)
- And train a model

```
> n = nrow(data)
> train = sample(n, size = 2/3 * n)
> test = setdiff(1:n, train)
>
> mod = train(lrn, task, subset = train)
```

Predictions I

- Make predictions for new data

```
> pred = predict(mod, task = task, subset = test)
> head(as.data.frame(pred))

##    id truth prob.0 prob.1 response
## 3   3     0  0.500  0.500        1
## 8   8     0  0.814  0.186        0
## 9   9     1  0.016  0.984        1
## 12 12     1  0.176  0.824        1
## 13 13     1  0.018  0.982        1
## 15 15     1  0.868  0.132        0
```

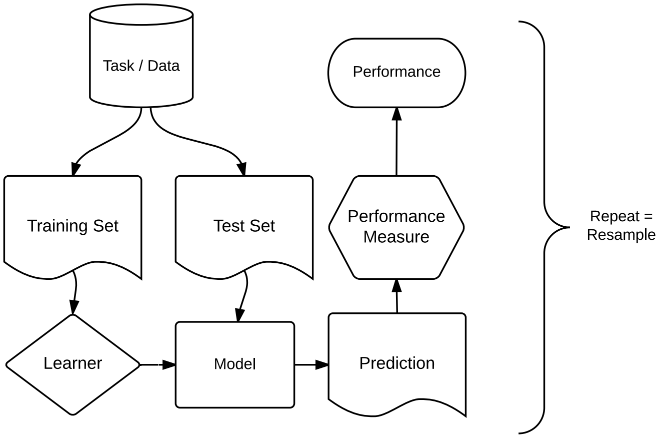
Predictions II

- Evaluate predictive performance

```
> performance(pred, measures = list(mlr::acc, mlr::auc))

##       acc       auc
## 0.8169336 0.8760819
```

Resampling

- Aim: Assess the performance of a learning algorithm
- Uses the data more efficiently then simple train-test
- Repeatedly split in train and test, then aggregate results.



Cross Validation

- Most popular resampling strategy: Cross validation with 5 or 10 folds
- Split the data into k roughly equally-sized partitions
- Use each part once as test set and joint $k-1$ other parts to train
- Obtain k test errors and average them

Example of 3-fold cross-validation

Iteration 1	Test	Train	Train
Iteration 2	Train	Test	Train
Iteration 3	Train	Train	Test

Crossvalidation in mlr I

```
> rdesc = makeResampleDesc("CV", iters = 3,
+   stratify = TRUE)
>
> r = resample(lrn, task, rdesc,
+   measures = list(mlr::acc, mlr::auc))
> print(r)

## Resample Result
## Task: titanic
## Learner: classif.randomForest
## Aggr perf: acc.test.mean=0.7906791,auc.test.mean=0.8548651
## Runtime: 4.30436
```

Crossvalidation in mlr II

```
> head(r$measures.test)

##   iter       acc       auc
## 1    1 0.7917620 0.8678310
## 2    2 0.7912844 0.8518965
## 3    3 0.7889908 0.8448679

> head(as.data.frame(r$pred))

##   id truth prob.0 prob.1 response iter  set
## 1  3     0  0.522  0.478        0    1 test
## 2  4     0  0.462  0.538        1    1 test
## 3 10     0  0.926  0.074        0    1 test
## 4 26     0  0.566  0.434        0    1 test
## 5 35     0  0.448  0.552        1    1 test
## 6 39     0  0.402  0.598        1    1 test
```

Resampling methods in mlr

Methods	Parameter
CV	`iters`
	`stratify`
LOO	
RepCV	`reps`
	`folds`
	`stratify`
Bootstrap	`iters`
	`stratify`
Subsample	`iters`
	`split`
	`stratify`
Holdout	`split`
	`stratify`

Benchmarking and Model Comparison I

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

```
> bmr = benchmark(list.of.learners, list.of.tasks, rdesc)
```

R Example: Algorithms I

- Benchmark experiment - Compare 4 algorithms

```
> set.seed(3)
>
> learners = c("glmnet", "naiveBayes", "randomForest",
+   "ksvm")
> learners = makeLearners(learners, type = "classif",
+   predict.type = "prob")
>
> bmr = benchmark(learners, task, rdesc,
+   measures = mlr::auc)
```

R Example: Algorithms II

- Access aggregated results

```
> getBMRAggrPerformances(bmr, as.df = TRUE)

##   task.id          learner.id auc.test.mean
## 1 titanic      classif.glmnet     0.8402273
## 2 titanic  classif.naiveBayes     0.8011408
## 3 titanic classif.randomForest     0.8571534
## 4 titanic        classif.ksvm     0.8292053
```

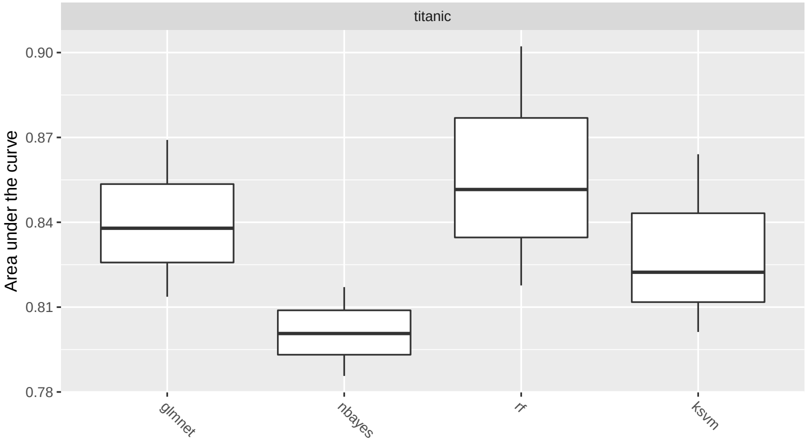
R Example: Algorithms III

- Access more fine-grained results
- Many more getters for predictions, models, etc.

```
> head(getBMRPerformances(bmr, as.df = TRUE), 4)

##   task.id         learner.id iter       auc
## 1 titanic     classif.glmnet    1 0.8378909
## 2 titanic     classif.glmnet    2 0.8136701
## 3 titanic     classif.glmnet    3 0.8691209
## 4 titanic classif.naiveBayes    1 0.8006653
```

R Example: Algorithms IV

```
> plotBMRBoxplots(bmr)
```



Performance measures

- Different performance measures for different types of learning problems
- `mlr` has 71 performance measures implemented
- See all via `https://mlr-org.github.io/mlr/articles/tutorial/devel/measures.html` or `listMeasures()`

mlr 2.13 Get Started Basics Advanced Extending Appendix mlr-org Packages Search...

Classification

ID / Name	Minim.	Best	Worst	Multi	Pred.	Truth	Probs	Model	Task	Feats	Aggr.	Note
acc Accuracy		1	0	X	X	X					test.mean	Defined as: mean(response == truth)
auc Area under the curve		1	0		X	X	X				test.mean	Integral over the graph that results fr and tpr for many different thresholds
bac Balanced accuracy		1	0	X	X	X					test.mean	For binary tasks, mean of true positiv negative rate.
ber Balanced error rate	X	0	1	X	X	X					test.mean	Mean of misclassification error rates classes.
brier Brier score	X	0	1		X	X	X				test.mean	The Brier score is defined as the qua between the probability and the valu That means we use the numeric repr for our target classes. It is similiar to error in regression. multiclass.brier is one vs. all comparisons and for a bin brier.
brier.scaled Brier scaled		1	0		X	X	X				test.mean	Brier score scaled to [0,1], see http://www.ncbi.nlm.nih.gov/pmc/ar
f1 F1 measure		1	0		X	X					test.mean	Defined as: 2 * tp/ (sum(truth == posi sum(response == positive))

Performance measure for Classification I

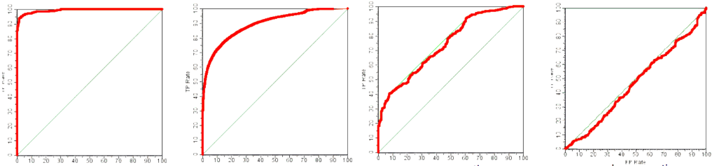
- In our Titanic example we have a classification problem
- Confusion matrix:
  contingency table of predictions $\hat{y}$ and true labels y

Diagnostic Testing Measures

		Actual Class y		
		Positive	Negative	
$\hat{y}$ **Test outcome**	Test outcome positive	**True positive** (TP)	**False positive** (FP, Type I error)	Precision = $\frac{\#\mathrm{TP}}{\#\mathrm{TP} + \#\mathrm{FP}}$
	Test outcome negative	**False negative** (FN, Type II error)	**True negative** (TN)	Negative predictive value = $\frac{\#\mathrm{TN}}{\#\mathrm{FN} + \#\mathrm{TN}}$
		Sensitivity = $\frac{\#\mathrm{TP}}{\#\mathrm{TP} + \#\mathrm{FN}}$	Specificity = $\frac{\#\mathrm{TN}}{\#\mathrm{FP} + \#\mathrm{TN}}$	Accuracy = $\frac{\#\mathrm{TP} + \#\mathrm{TN}}{\#\mathrm{TOTAL}}$

Performance measure for Classification II

- For classification performance measure the True Positive Rate (TPR) and the False Positive Rate (FPR) are plotted $\rightarrow$ ROC Curve (Receiver Operating Characteristic)



good separation | reasonable | poor separation | random separation

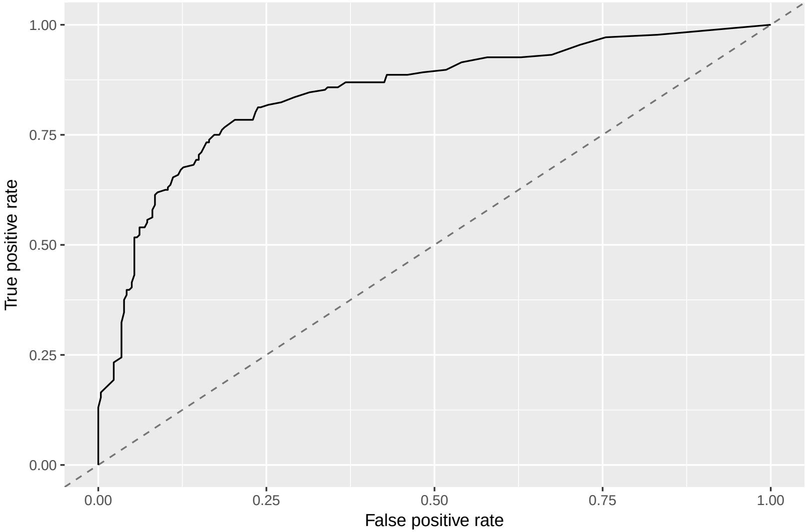
- For measuring the performance we can caluculate the area under the ROC curve (AUC)

R Example: Random Forest I

- The Random Forest seems to work best, lets have a closer look

```
> res = holdout(lrn, task)
> df = generateThreshVsPerfData(res$pred,
+   list(fpr, tpr, acc))
> plotROCCurves(df)
```

R Example: Random Forest II



R Example: Random Forest III

```
> print(calculateROCMeasures(pred), abbreviations = FALSE)

##      predicted
## true 0         1
##    0 238       30        tpr: 0.7  fnr: 0.3
##    1 50        119       fpr: 0.11 tnr: 0.89
##      ppv: 0.8 for: 0.17 lrp: 6.29 acc: 0.82
##      fdr: 0.2 npv: 0.83 lrm: 0.33 dor: 18.88
```

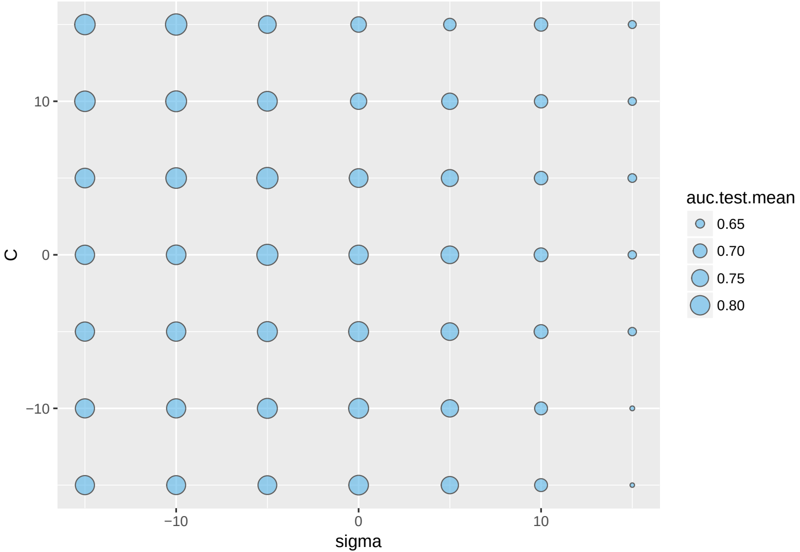
Hyperparameter Tuning

- Optimize parameters or decisions for ML algorithm w.r.t. the estimated prediction error
- Tuner proposes configuration, eval by resampling, tuner receives performance, iterate

Grid search

Try all combinations of finite grid

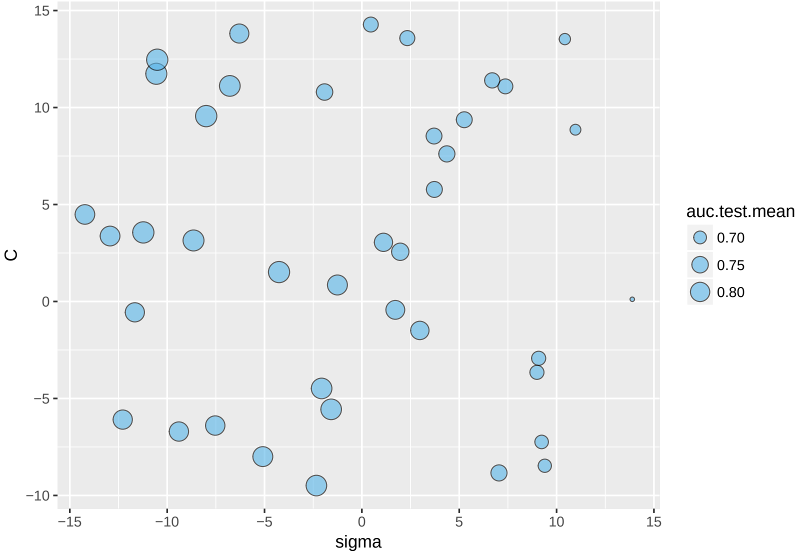
$\rightsquigarrow$ Inefficient, combinatorial explosion, searches irrelevant areas



Random search

Unformly randomly draw configurations,

$\rightsquigarrow$ Scales better then grid search, easily extensible



mlr

Tuning in mlr I

- Create a set of parameters
- Here we optimize an RBF SVM on logscale

```
> lrn.ksvm = makeLearner("classif.ksvm",
+   predict.type = "prob")
>
> par.set = makeParamSet(
+   makeNumericParam("C", lower = -8, upper = 8,
+     trafo = function(x) 2^x),
+   makeNumericParam("sigma", lower = -8, upper = 8,
+     trafo = function(x) 2^x)
+ )
```

Tuning in mlr II

- Optimize the hyperparameter of learner

```
> tune.ctrl = makeTuneControlRandom(maxit = 10L)
> tr = tuneParams(lrn.ksvm, task = task, par.set = par.set,
+   resampling = rdesc, control = tune.ctrl,
+   measures = mlr::auc)
```

Tuning in mlr III

```
> head(as.data.frame(tr$opt.path))[, c(1,2,3,7)]

##            C      sigma auc.test.mean exec.time
## 1   7.803771  2.0060031     0.7570655      2.54
## 2  -4.374242 -0.3324129     0.8160881      0.82
## 3  -5.417617  3.5509443     0.7770489      0.89
## 4   2.076026 -1.9390989     0.8135859      0.77
## 5   1.887830 -4.4571549     0.8321945      0.82
## 6  -2.167479  2.1372494     0.7861856      0.86
```

R Example: Tuning I

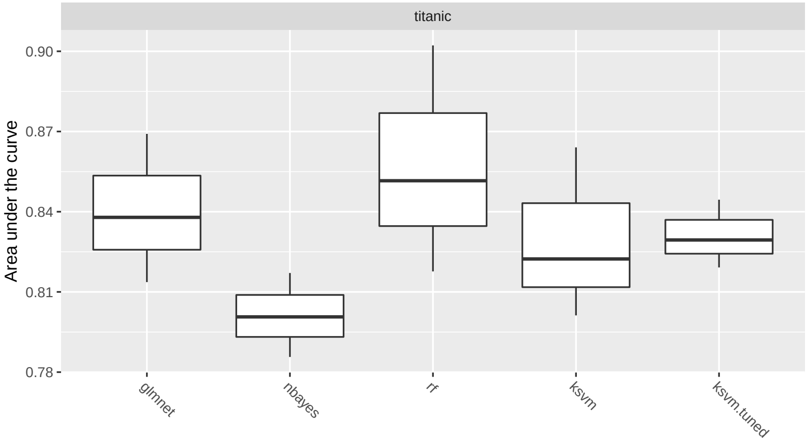
- We used all algorithms in their default settings
- Hopefully tuning will improve the performance
- Nested cross validation to get true out-of-sample predictions

```
> classif.ksvm.tuned = makeTuneWrapper(
+   lrn.ksvm, resampling = rdesc,
+   par.set = par.set, control = tune.ctrl)
> bmr2 = benchmark(classif.ksvm.tuned, task, rdesc)
```

- `makeTuneWrapper`: Fuses a base learner with a search strategy to select its hyperparameters

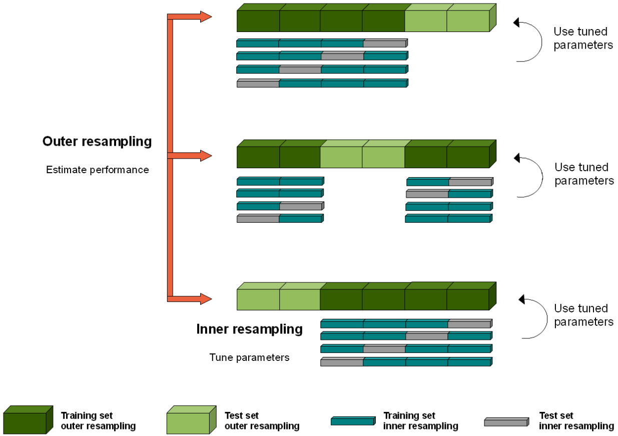
R Example: Tuning II

```
> plotBMRBoxplots(mergeBenchmarkResults(list(bmr, bmr2)))
```



Nested resampling I

- Danger of overfitting if hyperparameter and performance evaluation on the same test set
- Solution: 3-way split into training, optimization and test set



Nested Resampling Example I

- We show nested resampling with tuning
- Therefore we need an additional inner resampling loop

```
> inner.rdesc = makeResampleDesc("Subsample", iters = 2)
>
> classif.ksvm.inner = makeTuneWrapper(
+   lrn.ksvm, resampling = inner.rdesc,
+   par.set = par.set, control = tune.ctrl,
+   measures = mlr::auc)
```

Nested Resampling Example II

- We use `rdesc` for the outer loop

```
> r.nest = resample(classif.ksvm.inner, task,
+   resampling = rdesc, extract = getTuneResult,
+   measures = mlr::auc)
> r.nest

## Resample Result
## Task: titanic
## Learner: classif.ksvm.tuned
## Aggr perf: auc.test.mean=0.8373767
## Runtime: 13.2617
```

Nested Resampling Example III

```
> r.nest$extract

## [[1]]
## Tune result:
## Op. pars: C=3.39; sigma=0.0317
## auc.test.mean=0.8448094
##
## [[2]]
## Tune result:
## Op. pars: C=0.859; sigma=0.0454
## auc.test.mean=0.8439726
##
## [[3]]
## Tune result:
## Op. pars: C=173; sigma=0.00819
## auc.test.mean=0.8462711
```

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"

Interpretable Machine Learning

- `iml` - Interpretable Machine Learning - `https://github.com/christophM/iml`
- Background
  - Machine learning has a huge potential
  - Lack of explanation hurts trusts and creates barrier for machine learning adoption
  - Interpretation of the behaviour and explanation of predictions of machine learning model with **Interpretable Machine Learning**

Supported methods

- Model-agnostic interpretability methods for **any** kind of machine learning model
- Supported are
  - Feature importance
  - Partial dependence plots
  - Individual conditional expectation plots
  - Tree surrogate
  - Local interpretable model-agnostic explanations
  - Shapley value

One IML model for all methods I

- Use `iml` package

```
> library(iml)
```

- We use our trained model `mod`
- We need training data from the index vector `train`

```
> mod

## Model for learner.id=classif.randomForest; learner.class=clas
## Trained on: task.id = titanic; obs = 872; features = 10
## Hyperparameters:
```

One IML model for all methods II

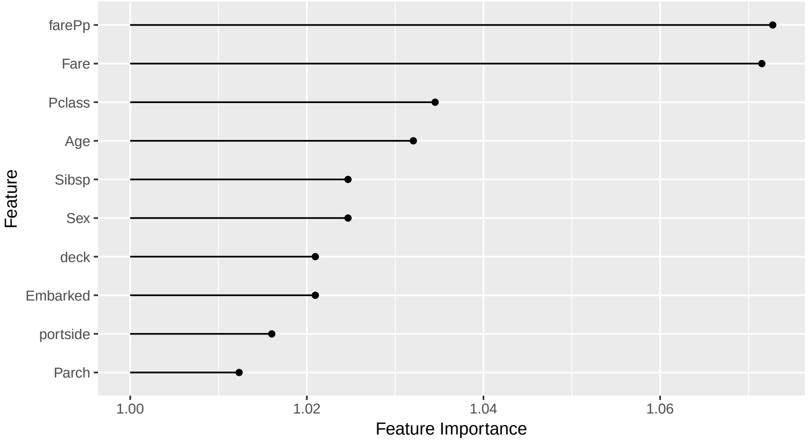
- Extract features
- Create IML model

```
> X = dropNamed(train.data, "Survived")
> iml.mod = Predictor$new(mod, data = X,
+   y = train.data$Survived, class = 2)
```

Feature Importance

- What were the most important features?

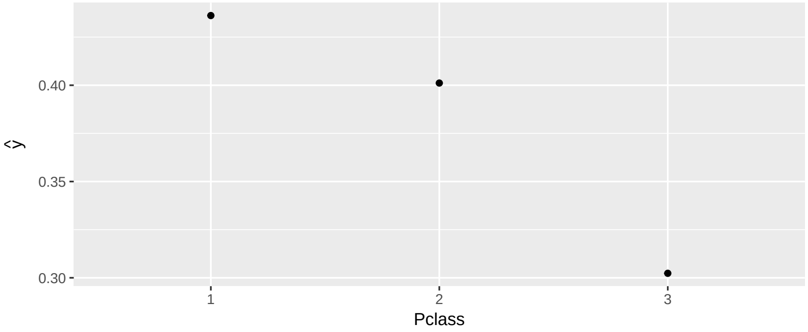
```
> imp = FeatureImp$new(iml.mod, loss = "ce")
> plot(imp)
```



Partial Dependence Plots

- How does the "passenger class" influence the prediction on average?

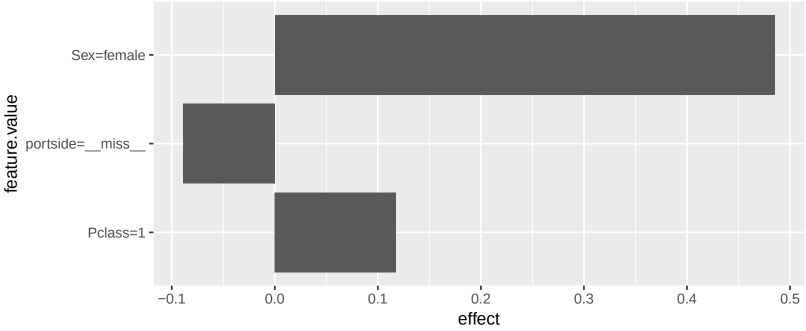
```r
> pdp = PartialDependence$new(iml.mod, feature = "Pclass")
> plot(pdp)
```



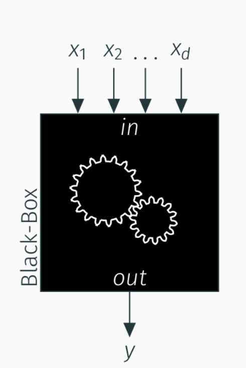
Local Linear Models (LIME)

- Explain a single prediction with LIME

```
> X[1,]

##   Pclass    Sex Age Sibsp Parch     Fare Embarked   farePp de
## 1      1 female  29     0     0 211.3375        S 211.3375

> lime = LocalModel$new(iml.mod, x.interest = X[1,])
> plot(lime)
```



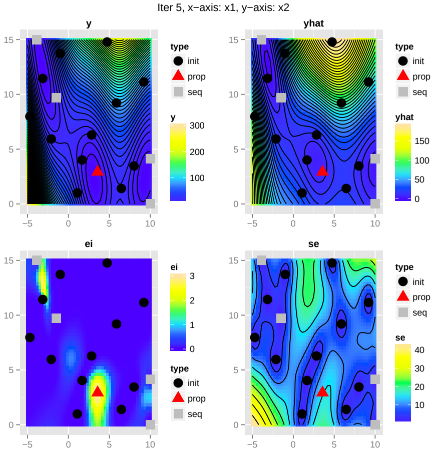
Expensive Black-Box Optimization



- `mlrMBO` - Bayesian Optimization and Model-Based Optimization - `https://github.com/mlr-org/mlrMBO`
- Goal: optimize *expensive black box functions* by *model-based optimization* (aka 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 I

Create an unified interface with the general `mlrMBO` workflow

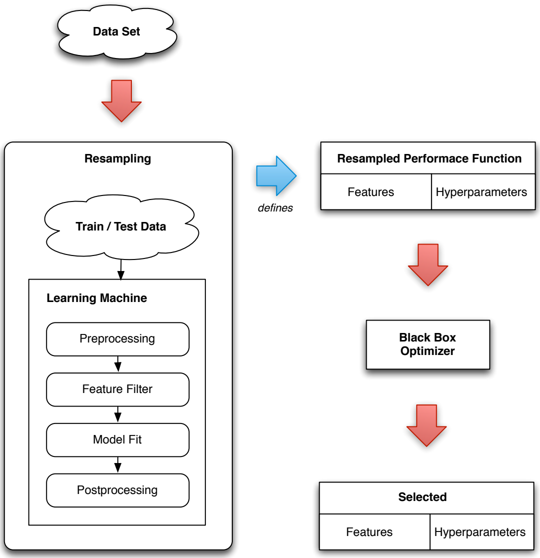
1. Define **objective function** and its parameters using the package `smoof`
2. Generate **initial design** (optional)
3. Define mlr' learner for **surrogate model** (optional)
4. Set up a **MBO control** object
5. Start the optimization with `mbo()`

mlrMBO II

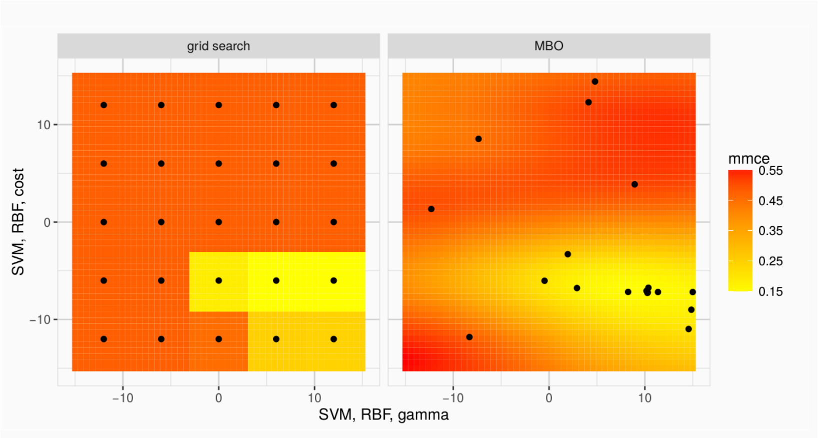
Supported are

- Efficient global optimization (EGO) of problems with numerical domain and Kriging as surrogate
- Using arbitrary regression models from mlr as surrogates
- Built-in parallelization using multi-point proposals
- Mixed-space optimization with categorical and subordinate parameters, for parameter configuration and tuning
- Multi-criteria optimization

From normal MBO to Hyperparameter Tuning



Hyperparameter Tuning



Example: Packages and Configuration

```
> library(mlrMBO) # Bayesian Optimization in R
> library(ParamHelpers) # Objects for parameter spaces
> library(smoof) # Interface for objective functions
> set.seed(2)
```

- We run all optimization with VERY fey evals to reduce time and log output

```
> iters = 5
```

Example: Mixed Space Optimization

- Extend our parameter set from Titanic example

```
> par.set = makeParamSet(
+   makeNumericParam("C", lower = -8, upper = 8,
+   trafo = function(x) 2^x),
+   makeNumericParam("sigma", lower = -8, upper = 8,
+   trafo = function(x) 2^x)
+   )
```

Objective Functon

- We use our Titanic task
- Create a single objective function

```
> svm = makeSingleObjectiveFunction(name = "svm.tuning",
+   fn = function(x) {
+   # remove inactive parameters coded with `NA`
+   x = x[!vlapply(x, is.na)]
+   lrn = makeLearner("classif.ksvm", par.vals = x)
+   crossval(lrn, task, iters = 2, show.info = FALSE)$aggr
+   },
+   par.set = par.set,
+   noisy = TRUE,
+   has.simple.signature = FALSE,
+   minimize = TRUE
+   )
> ctrl = makeMBOControl()
> ctrl = setMBOControlTermination(ctrl, iters = iters)
```

MBO Learner I

- Parameter set is not a suitable surrogate
- Use random Forest with imputation for non-active parameters

```
> makeMBOLearner(ctrl, svm)

## Learner regr.km from package DiceKriging
## Type: regr
## Name: Kriging; Short name: km
## Class: regr.km
## Properties: numerics,se
## Predict-Type: se
## Hyperparameters: jitter=TRUE,covtype=matern3_2,optim.method=g
```

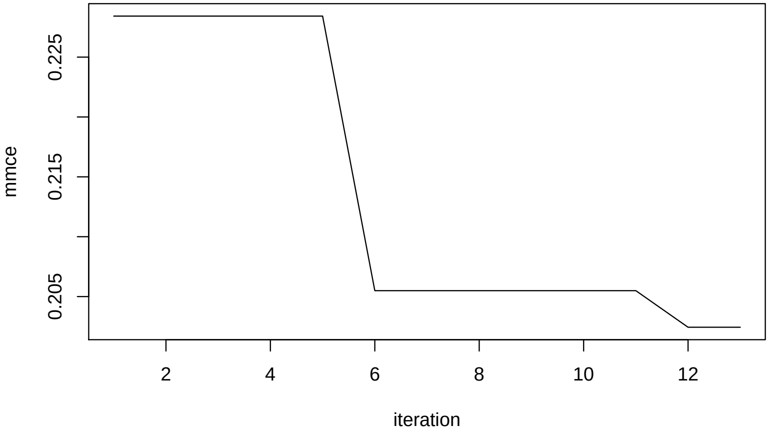
MBO Learner II

```
> res = mbo(svm, control = ctrl)
> print(res)

## Recommended parameters:
## C=-0.305; sigma=-2.19
## Objective: y = 0.202
##
## Optimization path
## 8 + 5 entries in total, displaying last 10 (or less):
##             C     sigma         y dob eol error.message exec.
## 4  -7.7416820 -6.333102 0.3819584   0  NA          <NA>
## 5   4.9360370  3.099967 0.3307725   0  NA          <NA>
## 6  -0.8946519 -3.522210 0.2054917   0  NA          <NA>
## 7  -4.4789734  4.361640 0.3819677   0  NA          <NA>
## 8   6.8105644 -4.292903 0.2177382   0  NA          <NA>
## 9   4.2962856 -2.585426 0.2207811   1  NA          <NA>
## 10  7.9947749 -3.508953 0.2360599   2  NA          <NA>
## 11  1.8919236 -4.051119 0.2207683   3  NA          <NA>
## 12 -0.3054272 -2.186221 0.2024371   4  NA          <NA>
## 13 -0.1544471 -2.866955 0.2024441   5  NA          <NA>
##    error.model train.time  prop.type propose.time se
```

mlr

Results

```
> op = as.data.frame(res$opt.path)
> plot(cummin(op$y), type = "l", ylab = "mmce",
+   xlab = "iteration")
```



References

- `mlrMBO` Paper on arXiv (under review) `https://arxiv.org/abs/1703.03373`
- Bischl, Wessing et al:*MOI-MBO: Multiobjective infill for parallel model-based optimization*, LION 2014
- Horn, Wagner, Bischl et al:*Model-based multi-objective optimization: Taxonomy, multi-point proposal, toolbox and benchmark*, EMO 2014

mlrCPO I

- `mlrCPO` - Composable Preprocessing Operators for mlr - `https://github.com/mlr-org/mlrCPO`

```
> library(mlrCPO)
```

- Preprocessing operations (e.g. imputation or PCA) as R objects with their own hyperparameters

```
> operation = cpoScale()
> print(operation)

## scale(center = TRUE, scale = TRUE)
```

mlrCPO II

- Objects are handled using the "piping" operator %>>%:
- Composition:

```
> imputing.pca = cpoImputeMedian() %>>% cpoPca()
```

- Application to data

```
> task %>>% imputing.pca
```

- Combination with a `Learner` to form a machine learning pipeline

```
> pca.rf = imputing.pca %>>%
+   makeLearner("classif.randomForest")
```

mlrCPO Example: Titanic I

The feature engineering and preprocessing steps done on the Titanic dataset, using `mlrCPO`:

```
> # Add interesting columns
> newcol.cpo = cpoAddCols(
+   farePp = Fare / (Parch + Sibsp + 1),
+   deck = stri_sub(Cabin, 1, 1),
+   side = {
+   digit = stri_sub(Cabin, 3, 3)
+   digit = suppressWarnings(as.numeric(digit))
+   c("port", "starboard")[digit %% 2 + 1]
+   })
```

mlrCPO Example: Titanic II

```
> # drop uninteresting columns
> dropcol.cpo = cpoSelect(names = c("Cabin",
+   "Ticket", "Name"), invert = TRUE)
>
> # impute
> impute.cpo = cpoImputeMedian(affect.type = "numeric") %>>%
+   cpoImputeConstant("__miss__", affect.type = "factor")
```

mlrCPO Example: Titanic III

```
> train.task = makeClassifTask("Titanic", train.data,
+   target = "Survived")
>
> pp.task = train.task %>>% newcol.cpo %>>%
+   dropcol.cpo %>>% impute.cpo
```

- Advantage: Different preprocessing steps can be tried by preparing different CPO objects (→ "strategy pattern").

Transformation of New Data

- New data (e.g. for testing, prediction) must also be preprocessed, in same order and with same hyperparameters
- Preprocessing parameters (e.g. PCA matrix) should only depend on training data
- Use `retrafo()` to get retrafo information to use on test data
- Object of type `CPOTRained`, behaves very similar to `CPO`

```
> # get retransformation
> ret = retrafo(pp.task)
> # can be applied to data using the %>>% operator,
> # just as a normal CPO
> pp.test = test.data %>>% ret
```

Combination with Learners

- Attach one or more CPO to a Learner to build machine learning pipelines
- Autotmatically handles preprocessing of test data

```
> learner = newcol.cpo %>>% dropcol.cpo %>>%
+   impute.cpo %>>% makeLearner("classif.randomForest",
+   predict.type = "prob")
>
> # the new object is a "CPOLearner", subclass of "Learner"
> inherits(learner, "CPOLearner")

## [1] TRUE

> # train using the task that was not preprocessed
> ppmod = train(learner, train.task)
```

Tuning with mlrCPO I

- CPO hyperparameters can be tuned in combination with Learner parameters
- Tuning can be done using `tuneParams()` function from `mlr`

```
> class(learner)[[1]]

## [1] "CPOLearner"

> ps = makeParamSet(
+   makeIntegerParam("ntree", lower = 1, upper = 500),
+   makeIntegerParam("mtry", lower = 1, upper = 10)
+ )
```

Tuning with mlrCPO II

```
> tuneParams(learner, train.task, cv3, par.set = ps,
+   control = makeTuneControlRandom(maxit = 10L),
+   measures = mlr::auc)

## Tune result:
## Op. pars: ntree=460; mtry=3
## auc.test.mean=0.8604592
```

mlrCPO III

- `listCPO()` to show available CPOs
- Currently 69 CPOs, and growing: imputation, feature type conversion, target value transformation, over/undersampling, ...
- "cbind" CPO combines different preprocessing outputs of the same data

```
> scale = cpoSelect(pattern = "Fare", id = "first") %>>%
+   cpoScale(id = "scale")
> scale.pca = scale %>>% cpoPca()
> cbinder = cpoCbind(scale, scale.pca, cpoSelect(
+   pattern = "Age", id = "second"))
> result = train.data %>>% cbinder
> result[1:3, ]

##       Fare      PC1     Age
## 1 3.768222 3.768222 29.0000
## 2 2.512035 2.512035  0.9167
## 4 2.512035 2.512035 30.0000
```

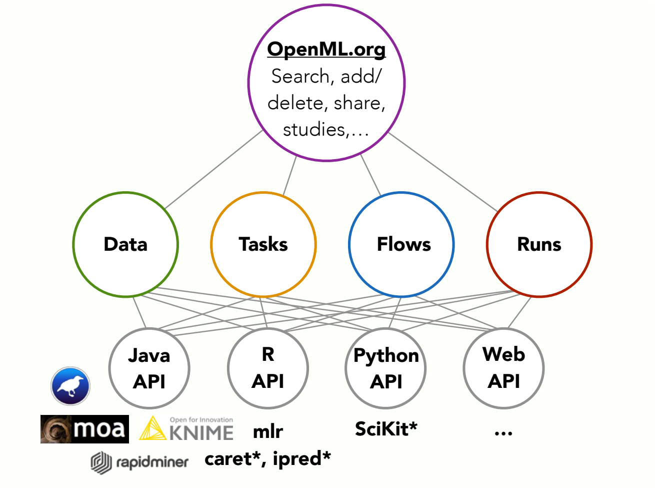
mlrCPO IV

- CPO "multiplexer" enables tuning over different distinct preprocessing operations
- Custom `CPOs` can be created using `makeCPO()`
- Further documentation in the vignettes:

```
> vignette("a_1_getting_started")
```

OpenML

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



OpenML R-Package

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

Tutorial

- 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

OpenML Account

- Install the openML package and either `farff` or `RWeka`

```r
> library("OpenML")
```

- You need an openML API key to talk to the server
- Create an account on `https://www.openml.org/register`

```r
> setOMLConfig(apikey = "c1994bdb7ecb3c6f3c8f3b35f4b47f1f")
>
> # Permanently save your API disk to your config file
> saveOMLConfig(apikey = "c1994...47f1f", overwrite=TRUE)
```

- Find your own API key in account settings `API Authentication`

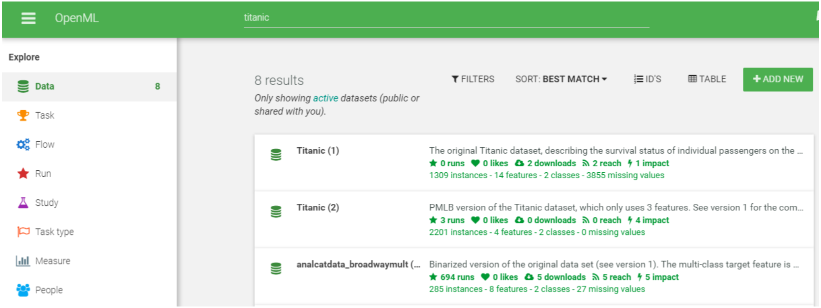
OpenML data and tasks I

- You can access all datasets or tasks

```
> datasets = listOMLDataSets()
> datasets[1:3, c(1,2,11)]

##   data.id     name number.of.features
## 1       2   anneal                 39
## 2       3 kr-vs-kp                 37
## 3       4    labor                 17

> tasks = listOMLTasks()
> tasks[1:3, 1:4]

##   task.id                 task.type data.id     name
## 1       2 Supervised Classification       2   anneal
## 2       3 Supervised Classification       3 kr-vs-kp
## 3       4 Supervised Classification       4    labor
```

OpenML data and tasks II

- Search for data on `https://www.openml.org/home`



OPENML TITANIC DATASET

- We download the Titanic dataset from OpenML

```
> listOMLDataSets(data.name = "titanic")[, 1:5]

##   data.id    name version status format
## 1   40704 Titanic       2 active   ARFF
## 2   40945 Titanic       1 active   ARFF

> titanic = getOMLDataSet(data.id = 40945L)
```

openML Titanic task

- We also can directly load the Titanic classification task

```
> listOMLTasks(data.name = "titanic")[1:2, 1:4]

##   task.id                     task.type data.id    name
## 1  145769                    Clustering   40704 Titanic
## 2  146230 Supervised Classification   40704 Titanic

> titanic.task = getOMLTask(task.id = 146230)
> titanic.task

##
## OpenML Task 146230 :: (Data ID = 40704)
##   Task Type            : Supervised Classification
##   Data Set             : Titanic :: (Version = 2, OpenML ID =
##   Target Feature(s)    : class
##   Estimation Procedure : Stratified crossvalidation (1 x 10 f
##   Evaluation Measure(s): precision
```

OpenML and mlr

- We can use OpenML and mlr together
- Use mlr for learner and use the task that we've got from OpenML

```
> lrn = makeLearner("classif.randomForest", mtry = 2)
> run.mlr = runTaskMlr(titanic.task, lrn)
> run.mlr$bmr$results

## $Titanic
## $Titanic$classif.randomForest
## Resample Result
## Task: Titanic
## Learner: classif.randomForest
## Aggr perf: ppv.test.join=0.7692308,timetrain.test.sum=3.94000
## Runtime: 4.17194
```

OpenML Upload

- You can upload your own data sets to OpenML
- Three steps are neccessary
    1. `makeOMLDataSetDescription`: create the description object of an OpenML data set
    2. `makeOMLDataSet`: convert the data set into an OpenML data set
    3. `uploadOMLDataSet`: upload the data set to the server
- We can upload our Titanic data set to OpenML

```
> titanic.desc = makeOMLDataSetDescription(name = "titanic",
+   description = "Titanic data set ...")
>
> titanic.data = makeOMLDataSet(desc = titanic.desc,
+   data = data, target.features = "Survived")
>
> # titanic.id = uploadOMLDataSet(titanic.data)
```

There is more ...

- Regression, Clustering and Survival analysis
- Cost-sensitive learning
- Multi-Label learning
- Imbalancy correction
- Wrappers
- Bayesian optimization
- Multi-criteria optimization
- Ensembles, generic bagging and stacking
- ...

We are working on

- Even better tuning system
- More interactive and 3D plots
- Large-Scale learning on databases
- Time-Series tasks
- Large-Scale usage of OpenML
- `auto-mlr`
- ...

mlr Contribution

- Write an issue on Git
- We are founding an association - **Machine Learning in R e.V**
  subscribe for updates `contact.mlr.org@gmail.com`

Thanks!