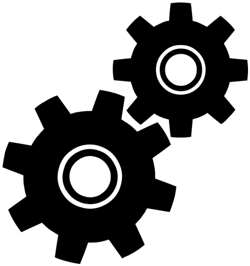
Machine Learning in R

Bernd Bischl
Computational Statistics, LMU

Material here: `goo.gl/DYzSmA`

Agenda

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



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

Motivation: mlr I

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 ML concepts!

Motivation: mlr II

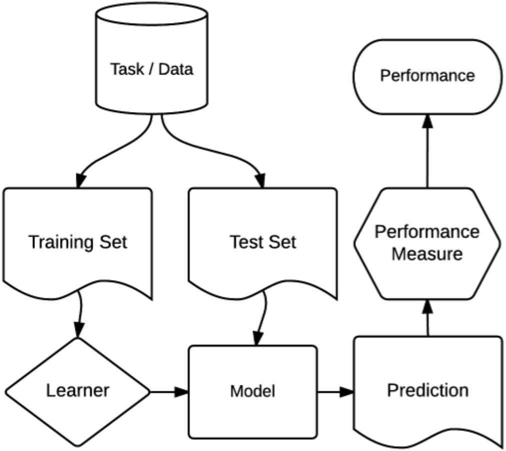
- Project home page

  `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

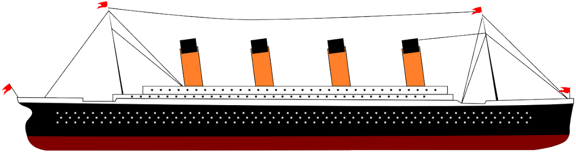
Motivation: `mlr` III

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



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

```
> # 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

- 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)
- ...

Clustering (9)

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

Regression (61)

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

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
## 2   2     1  0.566  0.434        0
## 10 10     0  0.884  0.116        0
## 11 11     0  0.868  0.132        0
## 12 12     1  0.110  0.890        1
## 16 16     0  0.518  0.482        0
## 20 20     0  0.908  0.092        0
```

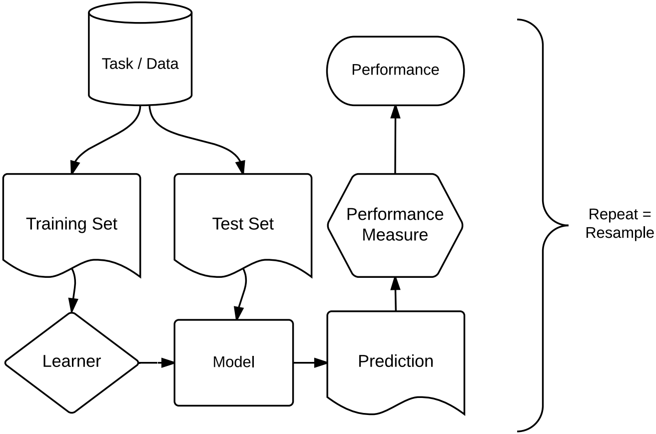
Predictions II

- Evaluate predictive performance

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

##    acc    auc
## 0.7963 0.8515
```

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=0.7998,auc.test.mean=0.8534
## Runtime: 1.53608
```

Crossvalidation in mlr II

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

##   iter    acc    auc
## 1    1 0.8165 0.8575
## 2    2 0.8146 0.8693
## 3    3 0.7683 0.8332

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

##   id truth prob.0 prob.1 response iter  set
## 1 31     0  0.584  0.416        0    1 test
## 2 39     0  0.420  0.580        1    1 test
## 3 53     0  0.822  0.178        0    1 test
## 4 59     0  0.930  0.070        0    1 test
## 5 71     0  0.946  0.054        0    1 test
## 6 75     0  0.450  0.550        1    1 test
```

Resampling methods in mlr

Method	Parameters
Holdout	`split`
	`stratify`
CV	`iters`
	`stratify`
LOO	
RepCV	`reps`
	`folds`
	`stratify`
Subsample	`iters`
	`split`
	`stratify`
Bootstrap	`iters`
	`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        0.8402
## 2 titanic    classif.naiveBayes        0.8011
## 3 titanic classif.randomForest        0.8583
## 4 titanic          classif.ksvm        0.8326
```

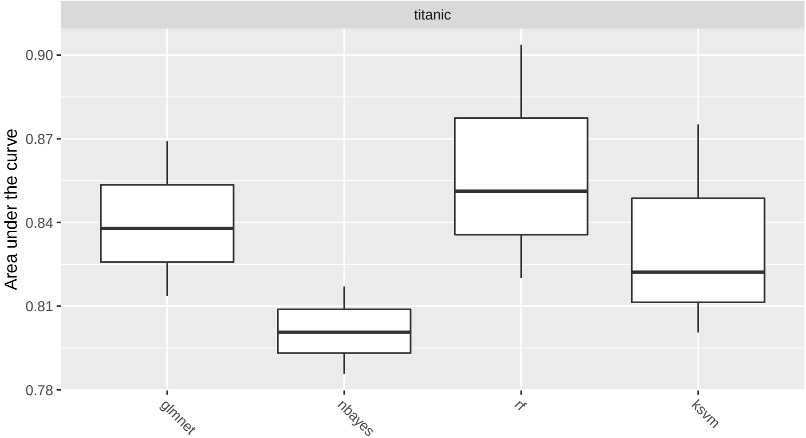
R Example: Algorithms III

- 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 0.8379
## 2 titanic     classif.glmnet    2 0.8137
## 3 titanic     classif.glmnet    3 0.8691
## 4 titanic classif.naiveBayes    1 0.8007
```

R Example: Algorithms IV

```
> plotBMRBoxplots(bmr)
```



Performance measures I

- `mlr` has 71 performance measures implemented
- See all via `https://mlr-org.github.io/mlr/articles/tutorial/devel/measures.html`
- Or ask via `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 measures II

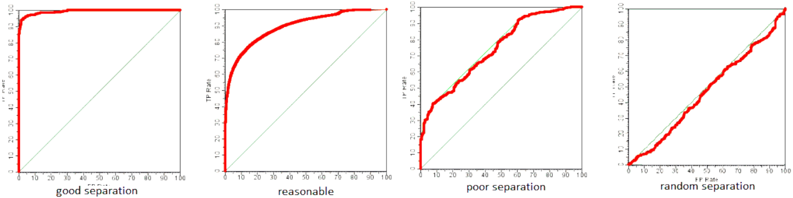
- Titanic is binary classification
- 2x2 confusion matrix: true labels y vs.predictions $\hat{y}$:

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 measures III

- Most classifiers are scoring systems
- Every threshold on that score induces a binary system
- Measure TPR and FPR for all, then put them in a ROC plot



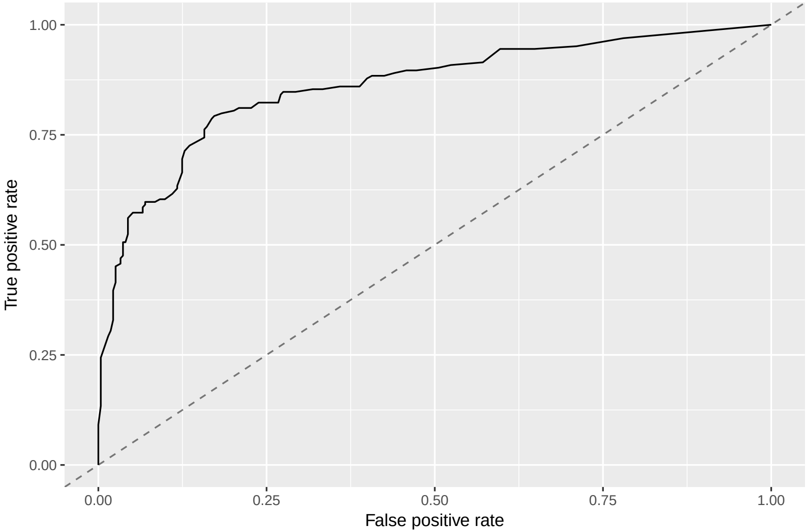
- AUC is the area under such a ROC curve (between 0.5 and 1)

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
##    0 229        46         tpr: 0.73 fnr: 0.27
##    1 43         119        fpr: 0.17 tnr: 0.83
##      ppv: 0.72 for: 0.16 lrp: 4.39 acc: 0.8
##      fdr: 0.28 npv: 0.84 lrm: 0.32 dor: 13.78
```

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

Hyperparameters in mlr I

```
> lrn = makeLearner("classif.rpart")
> getParamSet(lrn)

##                    Type len  Def    Constr Req Tunable Trafo
## minsplit        integer   -   20 1 to Inf   -    TRUE     -
## minbucket       integer   -    - 1 to Inf   -    TRUE     -
## cp              numeric   - 0.01   0 to 1   -    TRUE     -
## maxcompete      integer   -    4 0 to Inf   -    TRUE     -
## maxsurrogate    integer   -    5 0 to Inf   -    TRUE     -
## usesurrogate   discrete   -    2    0,1,2   -    TRUE     -
## surrogatestyle discrete   -    0      0,1   -    TRUE     -
## maxdepth        integer   -   30  1 to 30   -    TRUE     -
## xval            integer   -   10 0 to Inf   -   FALSE     -
## parms           untyped   -    -        -   -    TRUE     -
```

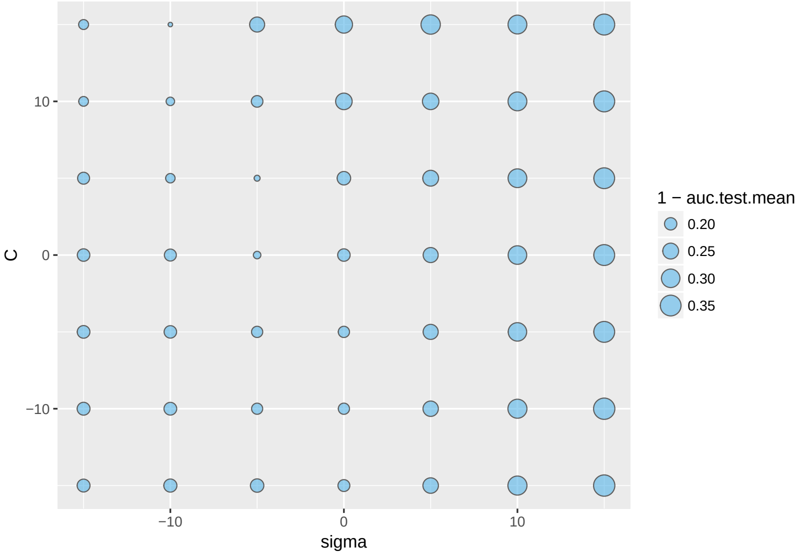
Hyperparameters in mlr II

- Either set them in constructor or change them later

```
> lrn = makeLearner("classif.ksvm", C = 5, sigma = 3)
> lrn = setHyperPars(lrn, C = 1, sigma = 2)
```

Grid search

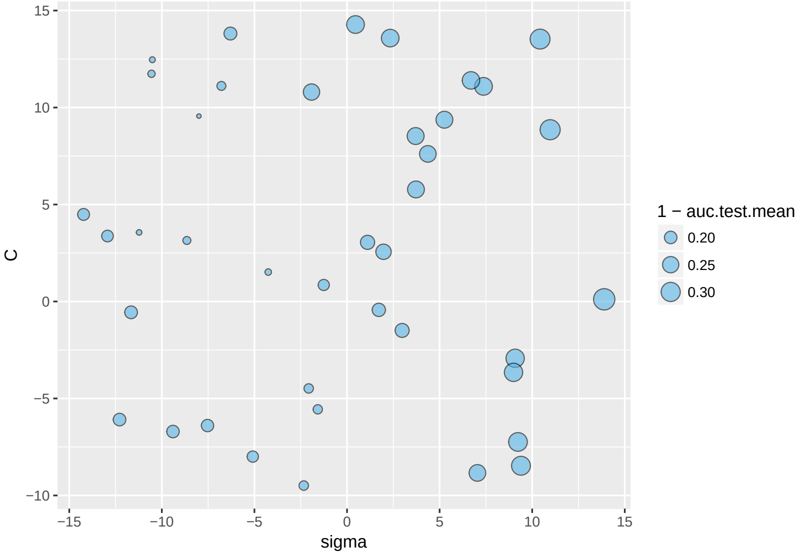
Try all combinations of finite grid
$\leadsto$ Inefficient, combinatorial explosion, searches irrelevant areas



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 = 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

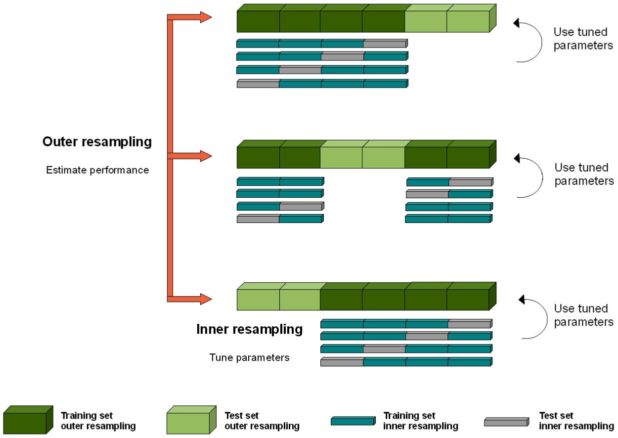
```
> tune.ctrl = makeTuneControlRandom(maxit = 50L)
> tr = tuneParams(lrn, task = task, par.set = par.set,
+   resampling = rdesc, control = tune.ctrl,
+   measures = mlr::auc)
```

Tuning in mlr III

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

##        C   sigma auc.test.mean exec.time
## 1  7.804  2.0060        0.7571     1.684
## 2 -4.374 -0.3324        0.8161     0.695
## 3 -5.418  3.5509        0.7770     0.673
## 4  2.076 -1.9391        0.8136     0.668
## 5  1.888 -4.4572        0.8322     0.625
## 6 -2.167  2.1372        0.7862     0.713
```

Nested resampling I

- Continuous tuning on the same data can lead to overfitting
- Unbiased evaluation with split into train, optimization and test set



Nested Resampling Example I

- `makeTuneWrapper`: Fuses a base learner with a search strategy to select its hyperparameters
- Therefore we need an additional inner resampling loop
- Tuning settings are like before (par.set and ctrl)

```
> inner = makeResampleDesc("Subsample", iters = 4)
> lrn = makeLearner("classif.ksvm", predict.type = "prob")
> lrn.autosvm = makeTuneWrapper(
+   lrn, resampling = inner,
+   par.set = par.set, control = tune.ctrl,
+   measures = mlr::auc)
```

Nested Resampling Example II

- We use `rdesc` for the outer loop

```
> r = resample(lrn.autosvm, task,
+   resampling = rdesc, extract = getTuneResult,
+   measures = mlr::auc)
> r

## Resample Result
## Task: titanic
## Learner: classif.ksvm.tuned
## Aggr perf: auc.test.mean=0.8402
## Runtime: 101.106
```

Nested Resampling Example III

```
> r$extract

## [[1]]
## Tune result:
## Op. pars: C=34.5; sigma=0.0105
## auc.test.mean=0.8403
##
## [[2]]
## Tune result:
## Op. pars: C=1.53; sigma=0.0237
## auc.test.mean=0.8268
##
## [[3]]
## Tune result:
## Op. pars: C=47.7; sigma=0.00936
## auc.test.mean=0.8364
```

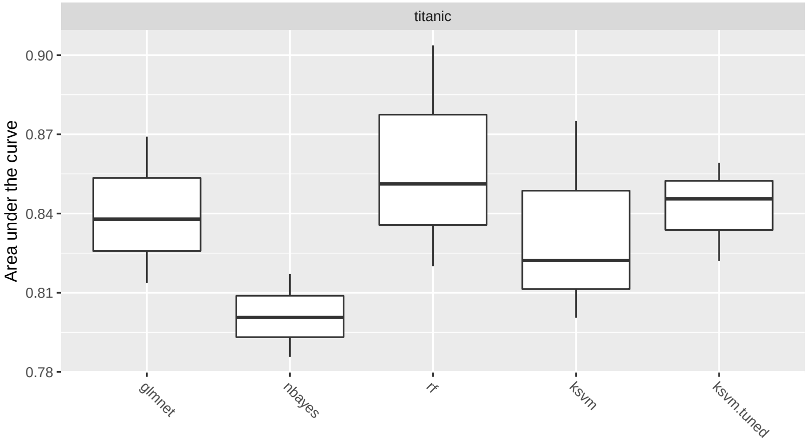
R Example: Tuning + Nested I

- Let's add our auto-tuned SVM to the benchmark

```
> bmr2 = benchmark(lrn.autosvm, task, rdesc)
```

R Example: Tuning + Nested II

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



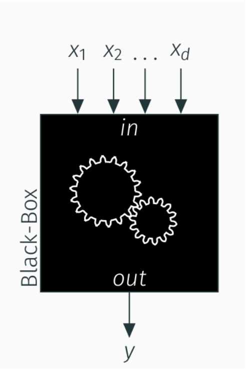
Parallelization

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

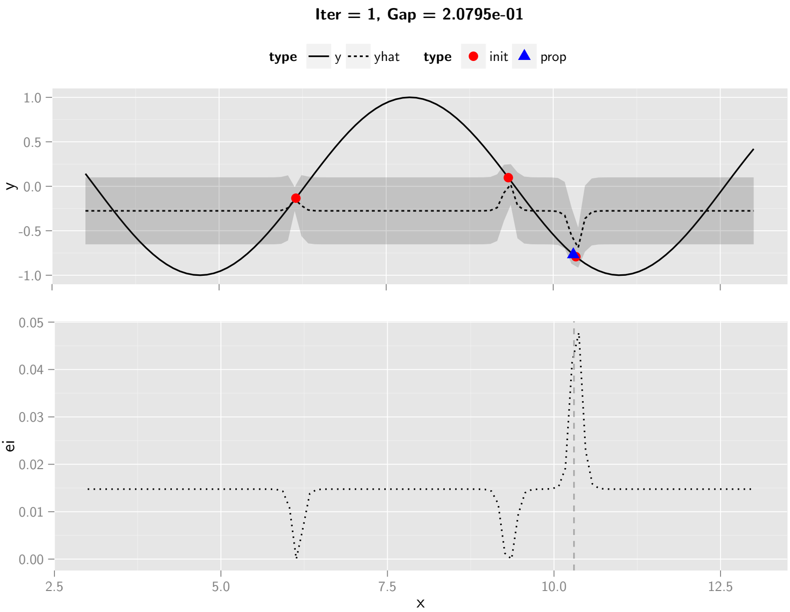
```
> parallelStart("multicore")
> benchmark(...)
> parallelStop()
```

- 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"

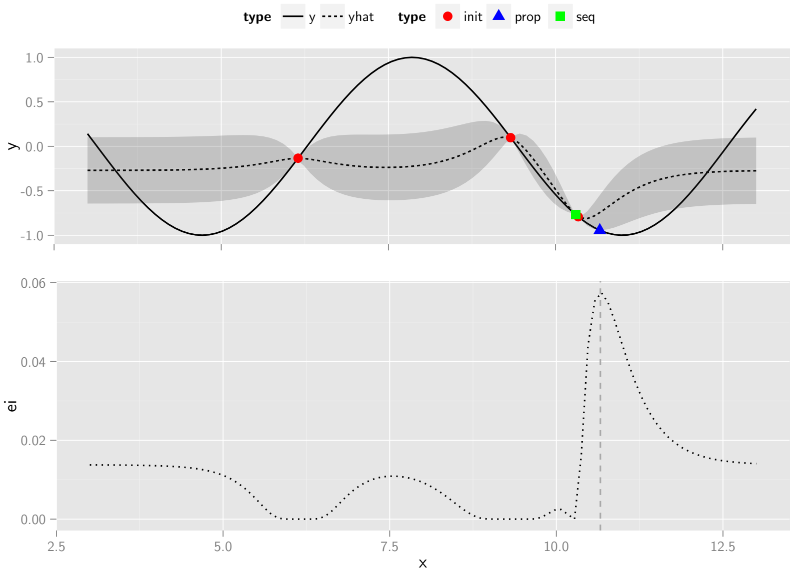
Expensive Black-Box Optimization



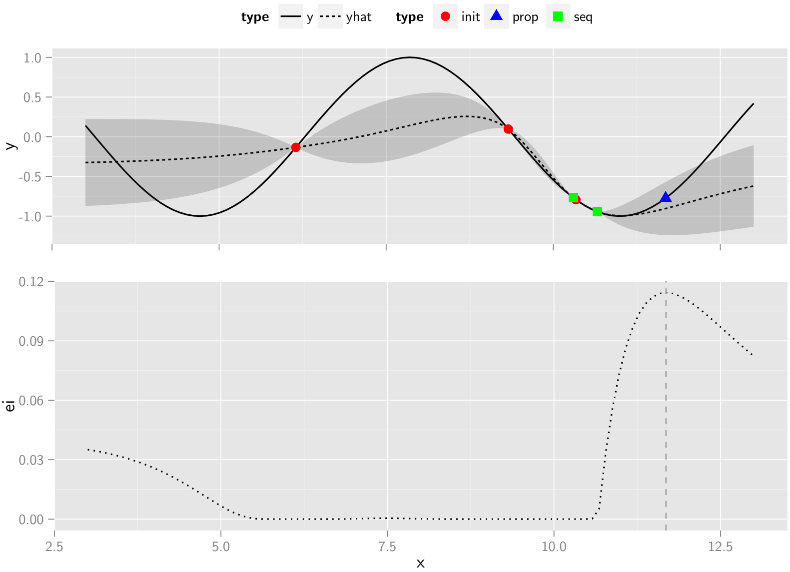
- `mlrMBO` - Bayesian Optimization and Model-Based Optimization - `https://github.com/mlr-org/mlrMBO`
- General idea:
  - Do some experiments on the black box
  - Measure performance
  - Model relationship between params and performance by regression
  - Optimize surrorgate model to get a new interesting configuration
  - Evaluate
  - Iterate


Iter = 1, Gap = 2.0795e-01
type y yhat
type init prop
y
1.0
0.5
0.0
-0.5
-1.0
ei
0.05
0.04
0.03
0.02
0.01
0.00
2.5
5.0
7.5
10.0
12.5
x

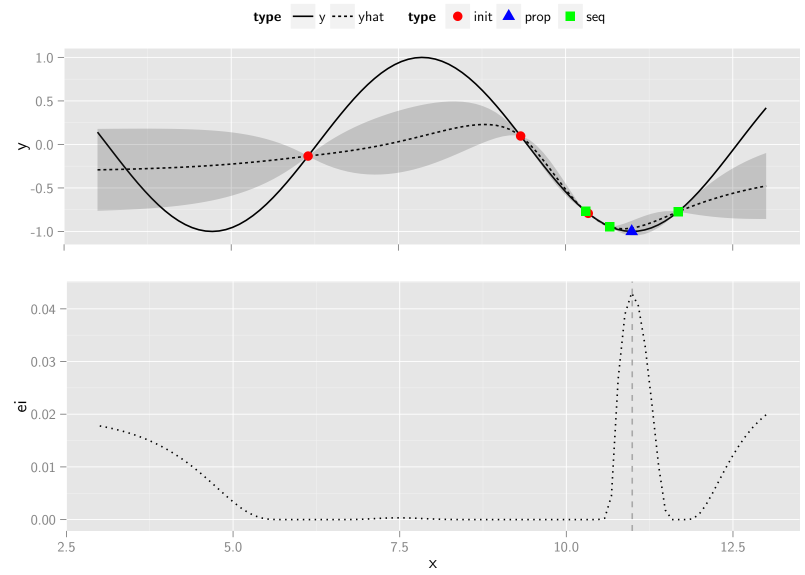
Iter = 2, Gap = 5.5410e-02

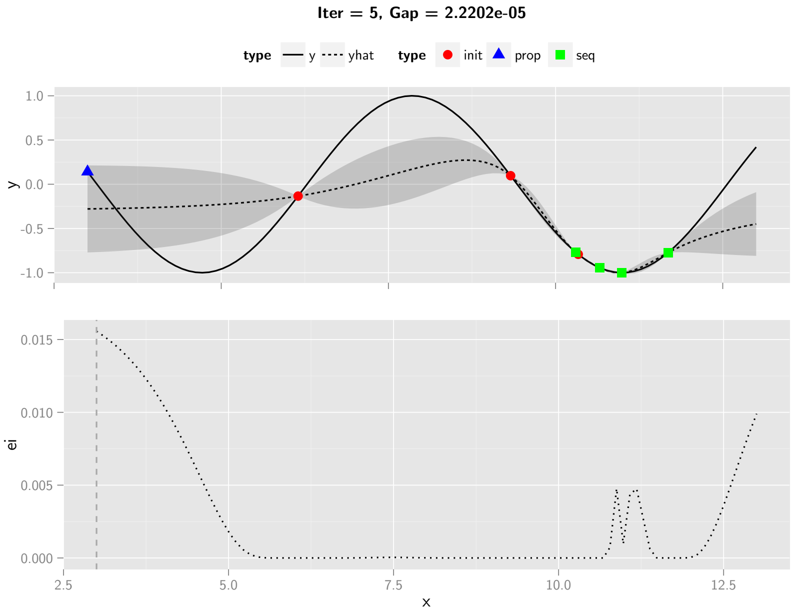


Iter = 3, Gap = 5.5410e-02

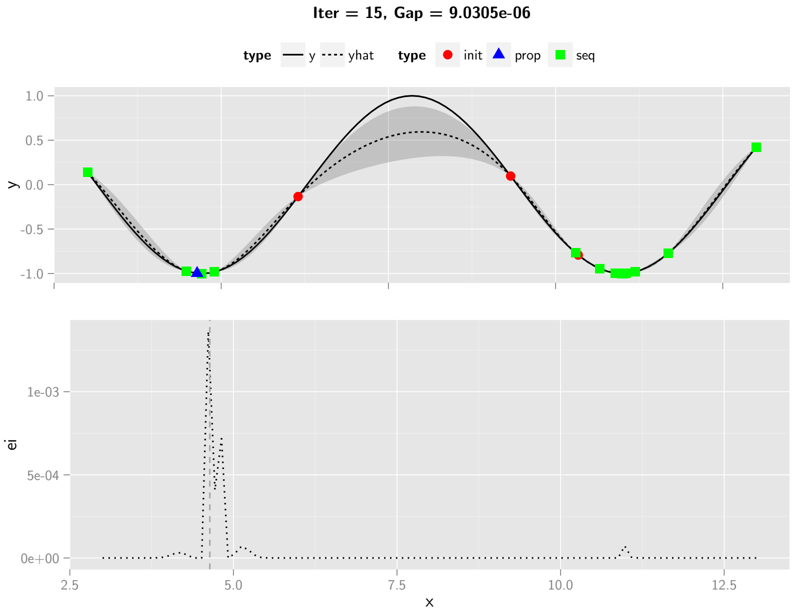


Iter = 4, Gap = 2.2202e-05



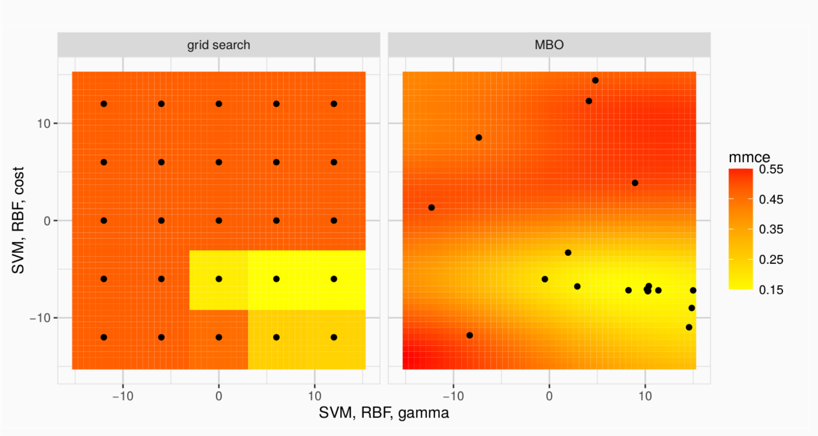

Iter = 5, Gap = 2.2202e-05
type
y
yhat
type
init
prop
seq
1.0
0.5
0.0
-0.5
-1.0
y
0.015
0.010
0.005
0.000
ei
2.5
5.0
7.5
10.0
12.5
x

mlr



Iter = 15, Gap = 9.0305e-06
type
y
yhat
type
init
prop
seq
1.0
0.5
0.0
-0.5
-1.0
y
1e-03
5e-04
0e+00
ei
2.5
5.0
7.5
10.0
12.5
x

Hyperparameter Tuning



mlrMBO

General `mlrMBO` workflow:

1. Define **objective function** and its parameters
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()`

Or use mlr's really simple tuning interface with mbo!

Machine Learning

- Successful, but requires human labor and expertise
  - Pre-process data
  - Select/ engineer features
  - Select a model family
  - Optimize hyperparameters (algorithm parameters)
  - $\cdots$
- Deep learning lets us automatically learn features
  - Automates feature engineering step, with large amount of data
  - Even more sensitive to architectures, hyperparameters, $\cdots$

Automatic Machine Learning I

- Can algorithms be trained to automatically build end-to- end machine learning systems?

Use machine learning to do better machine learning

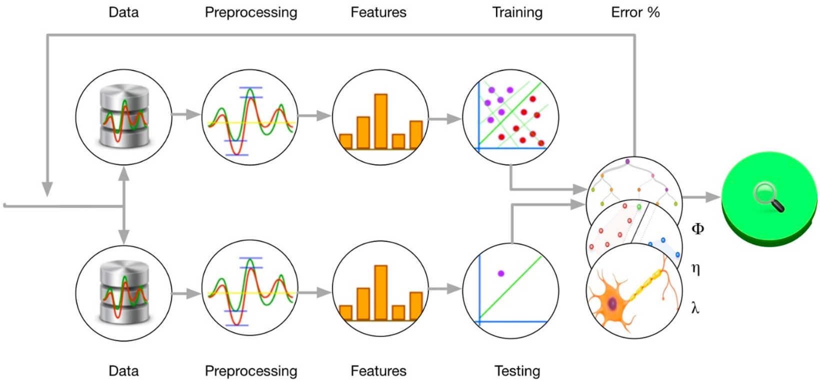
- Can we turn
  Solution = data + manual exploration + computation
- Into
  Solution = data + computation (x100)

Automatic Machine Learning II

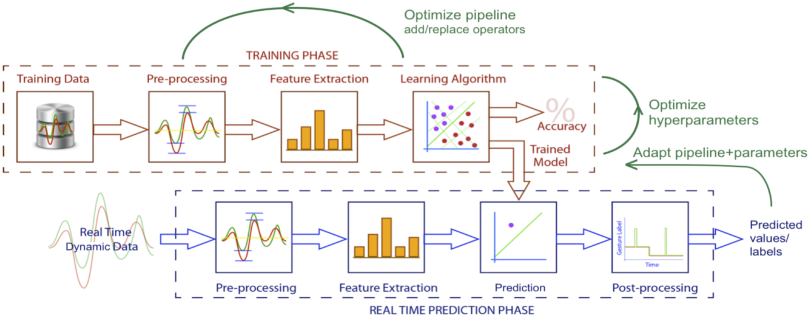
Not about automating data scientists

- Efficient exploration of techniques
  - Automate the tedious aspects (inner loop)
  - Make every data scientist a super data scientist
- Democratisation
  - Allow individuals, small companies to use machine learning effectively (at lower cost)
  - Open source tools and platforms
- Data Science
  - Better understand algorithms, develop better ones
  - Self-learning algorithms

Machine Learning Pipelines



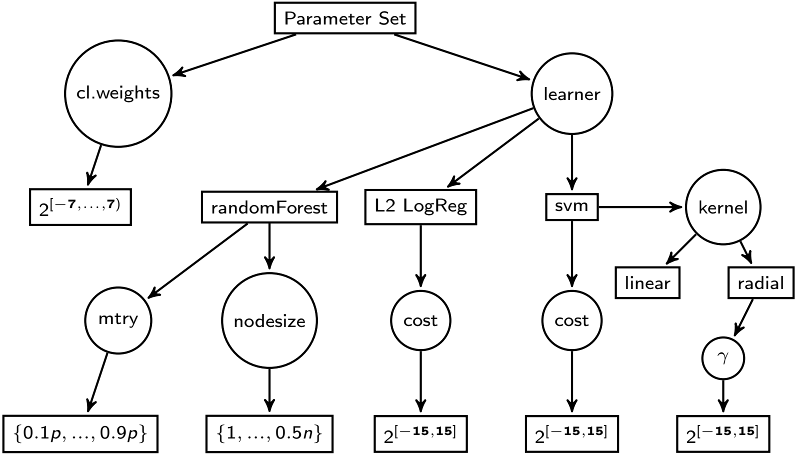
Automating Machine Learning Pipelines



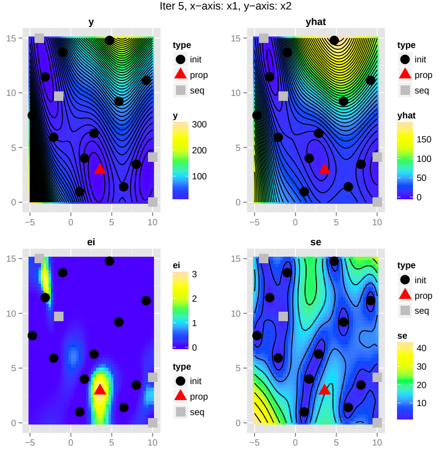
Automatic Machine Learning: Techniques

- **Bayesian Optimization:** Intelligently optimize pipelines/ architectures by iteratively choosing better ones
- **Genetic algorithms:** Evolve pipelines/architectures to work better for a given application
- **Meta-learning:** learn from previous applications to predict useful pipelines/ architectures for new problems
- **Transfer Learning:** train models on one problem, then transfer (parts) of good solutions to solve new problems.
- **Reinforcement Learning:** Train many models, use performance as "reward" for certain approaches
- **Combinations of all of these**

Automatic Machine Learning: Parameters



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



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 jointly, and jointly with Learner parameters
- Tuning can be done using `tuneParams()` function from `mlr` or nested resampling, without any problem

```
> lrn = cpoFilterFeatures(abs = 2L) %>>%
+   makeLearner("classif.randomForest")
>
>
> ps = makeParamSet(
+   makeDiscreteParam("filterFeatures.method",
+     values = c("anova.test", "chi.squared")),
+   makeIntegerParam("mtry", lower = 1, upper = 10)
+ )
> ctrl = makeTuneControlRandom(maxit = 10L)
> tr = tuneParams(lrn, iris.task, cv3, par.set = ps,
+   control = ctrl)
```

mlrCPO III

- "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
## 2  2.1137  2.1137  0.9167
## 4  2.1137  2.1137 30.0000
## 6 -0.1458 -0.1458 48.0000
```

mlrCPO IV

- `listCPO()` to show available `CPOs`
- Currently 69 CPOs, and growing: imputation, feature type conversion, target value transformation, over/undersampling, ...
- CPO "multiplexer" enables tuning over different distinct preprocessing operations
- Custom `CPOs` can be created using `makeCPO()`
- Further documentation in the vignettes

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

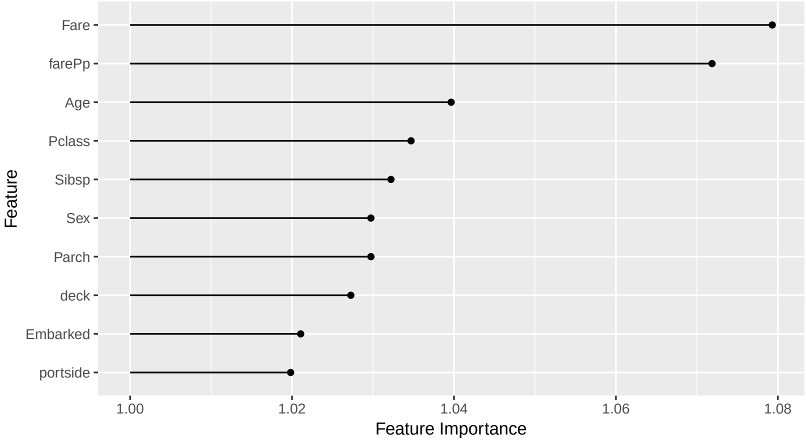
- Extract features
- Create IML model

```
> 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?

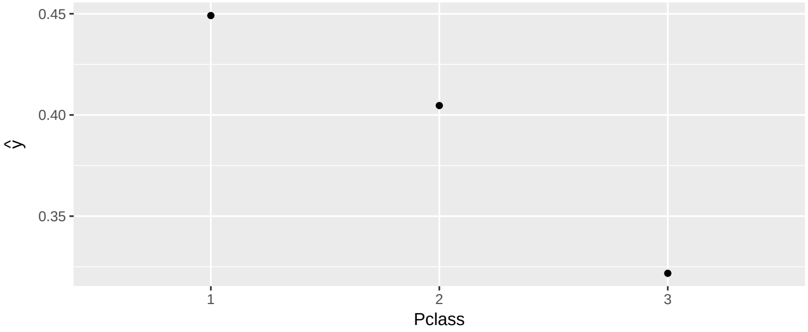
```
> imp = FeatureImp$new(iml.mod, loss = "ce")
> plot(imp)
```



Partial Dependence Plots

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

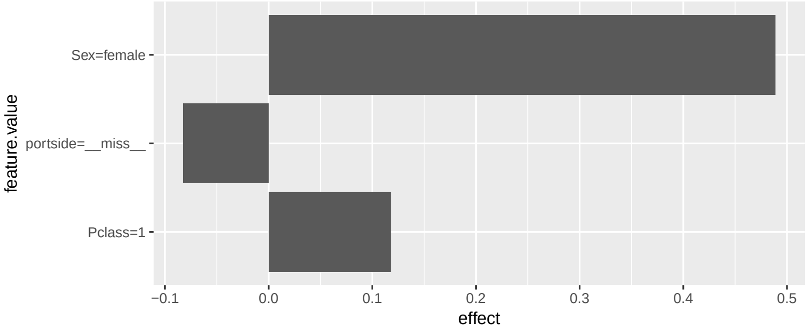
```
> pdp = PartialDependence$new(iml.mod, feature = "Pclass")
> plot(pdp)
```



Local Linear Models (LIME)

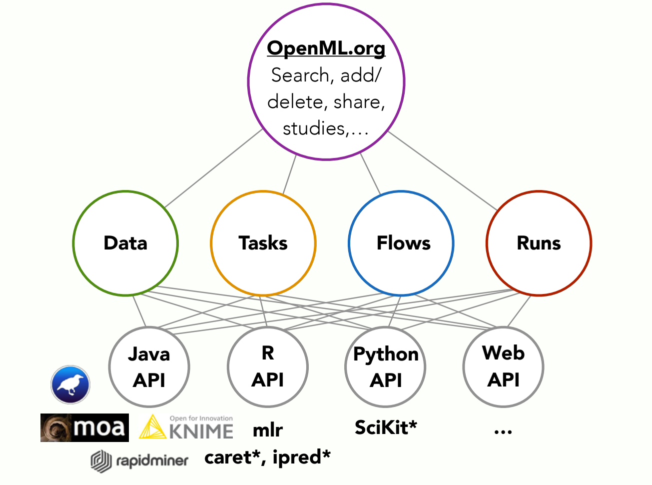
- Explain a single prediction with LIME

```
> X[1,]

##   Pclass    Sex Age Sibsp Parch  Fare Embarked farePp deck po
## 1      1 female  29     0     0 211.3        S  211.3    B __

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



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`

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

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

```
> 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`

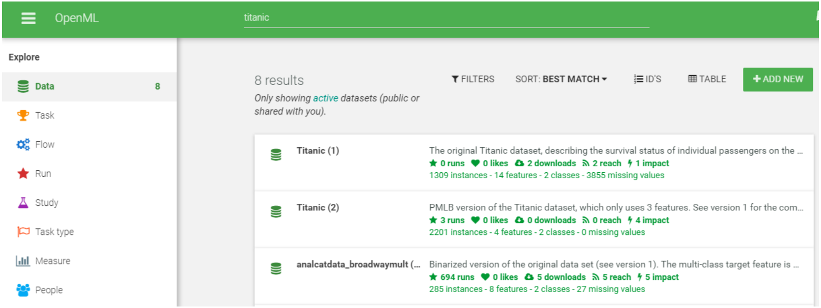
OpenML data and tasks I

- 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=0.7692,timetrain.test.sum=3.0720,tim
## Runtime: 3.17739

> # uploadOMLRun(run.mlr)
```

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!