Machine Learning in R: Package mlr

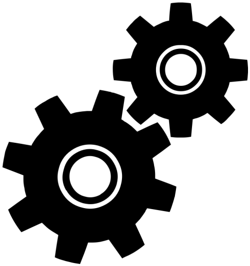
Bernd Bischl
Computational Statistics, LMU

Agenda

- About mlr
- Features of mlr
  - Tasks and Learners
  - Benchmarking
  - Parallelization
  - Hyperparameter Tuning
  - Performance Visualization
- iml - Interpretable Machine Learning
- mlrMBO - Bayesian Optimization
- mlrCPO - Composable Preprocessing
- OpenML
- mlr Contribution

Workshop documentation

https://github.com/mlr-org/mlr/wiki/mlr-for-openML-2018



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

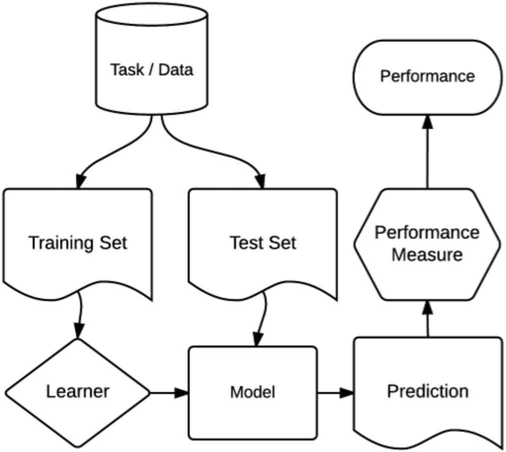
- Project home page

  `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, ...

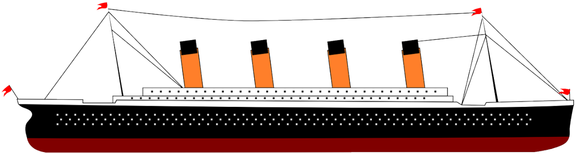


The mlr process

R Example: Titanic

Titanic: Machine Learning from Disaster

- Titanic sinking on April 15, 1912
- 1502 out of 2224 passengers got killed
- Task
  - What sorts of people were likely to survive? $\Rightarrow$ target variable `Survived`
  - Also important to answer: Why did people die?



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

R Example: Get the data

- Download the training and test data set from `https://www.kaggle.com/c/titanic`
- Install & load the neccessary libraries

```
> library(mlr)
> library(BBmisc)
> library(stringi)
> library(ggplot2)
```

Preprocessing I

- Load the input data
- Combine training and test data

```
> train = read.table("train.csv", header = TRUE, sep = ",",
+   colClasses = c("integer", "factor", "factor", "character", "factor",
+     "numeric", "numeric", "numeric", "factor", "numeric", "factor",
+     "factor"))
> train$train = TRUE
>
> test = read.table("test.csv", header = TRUE, sep = ",",
+   colClasses = c("integer", "factor", "character", "factor", "numeric",
+     "numeric", "numeric", "factor", "numeric", "factor", "factor"))
> test$Survived = NA
> test$train = FALSE
>
> data = rbind(train, test)
> rm(train, test)
```

Preprocessing II

```
> summary(data)

##   PassengerId   Survived   Pclass     Name               Sex     
##  Min.   :   1   0   :549   1:323   Length:1309        female:466  
##  1st Qu.: 328   1   :342   2:277   Class :character   male  :843  
##  Median : 655   NA's:418   3:709   Mode  :character               
##  Mean   : 655                                                     
##  3rd Qu.: 982                                                     
##  Max.   :1309                                                     
##                                                                   
##       Age            SibSp            Parch            Ticket    
##  Min.   : 0.17   Min.   :0.0000   Min.   :0.000   CA. 2343:  11  
##  1st Qu.:21.00   1st Qu.:0.0000   1st Qu.:0.000   1601    :   8  
##  Median :28.00   Median :0.0000   Median :0.000   CA 2144 :   8  
##  Mean   :29.88   Mean   :0.4989   Mean   :0.385   3101295 :   7  
##  3rd Qu.:39.00   3rd Qu.:1.0000   3rd Qu.:0.000   347077  :   7  
##  Max.   :80.00   Max.   :8.0000   Max.   :9.000   347082  :   7  
##  NA's   :263                                      (Other) :1261  
##       Fare                     Cabin      Embarked   train        
##  Min.   :  0.000                  :1014    :  2    Mode :logical  
##  1st Qu.:  7.896   C23 C25 C27    :   6   C:270    FALSE:418      
##  Median : 14.454   B57 B59 B63 B66:   5   Q:123    TRUE :891      
##  Mean   : 33.295   G6             :   5   S:914                   
##  3rd Qu.: 31.275   B96 B98        :   4                           
##  Max.   :512.329   C22 C26        :   4                           
##  NA's   :1         (Other)        : 271                           
```

Preprocessing III

- 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)
```

Feature Engineering I

- Extract possible information from the names

```
> # First split first and last names and save it as a list
> names = stri_split(data$Name, fixed = ", ")
>
> # Possible information from the titles of the persons
> data$titles = vapply(names, function(name) {
+   stri_split(name[2], fixed = " ", simplify = TRUE)[1]
+ }, character(1))
>
> data$titles = forcats::fct_collapse(data$titles,
+   Noble = c("Capt.", "Col.", "Major.", "Sir.", "Lady.", "Rev.", "Dr.",
+     "Don.", "Dona.", "Jonkheer."),
+   # "the" is for "the countess" and got butchered by the splitting earlier
+   Mrs = c("Mrs.", "Ms.", "the"),
+   Mr = c("Mr."),
+   Miss = c("Mme.", "Mlle.", "Miss."),
+   Master = c("Master."))
>
> # "children and women first" we generate a variable to account for this
> data$dibs = data$Sex == "female" | data$Age < 15
>
> # Price per person, since multiple ticket prices are bought by one person
> data$farePp = data$Fare / (data$Parch + data$SibSp + 1)
```

Feature Engineering II

```
> # 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$portside = as.numeric(stri_sub(data$Cabin, 3, 3)) %% 2 == 0
>
> # Drop PassengerId, Ticket, Name and Cabin
> data = dropNamed(data, c("Cabin","PassengerId", "Ticket", "Name"))
```

Imputation

- Remove missing values
- Impute numerics with median and factors with a seperate category

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

Tasks

- Split back into train and test
- Create a unified task for classification problem

```
> train = data[data$train, ]
> train$train = NULL
>
> test = data[!data$train, ]
> test$train = NULL
>
> task.train = makeClassifTask(id = "titanic", data = train,
+   target = "Survived", positive = "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
- ...

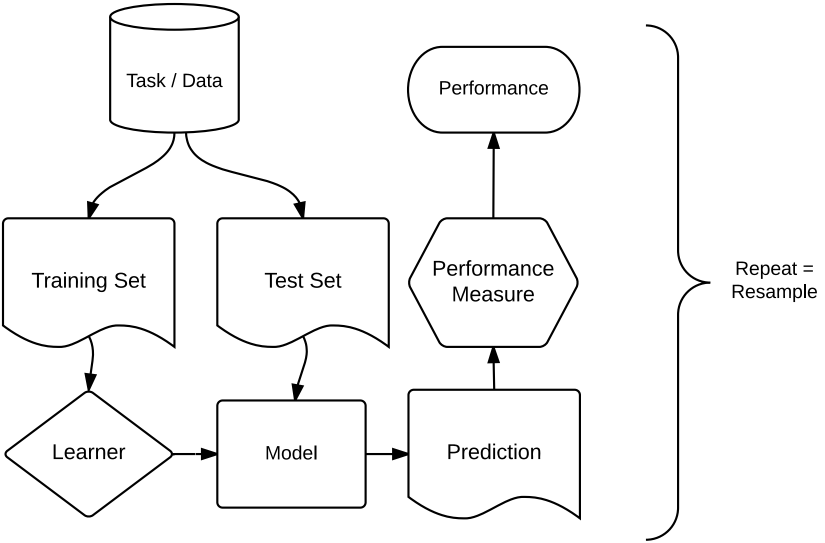
What Learners are available? II

- Explore all learners via tutorial
- Or ask mlr

```
> listLearners("classif",
+   properties = c("prob", "multiclass"))[1:3, 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
##   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
##   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
##   lcens rcens icens
## 1 FALSE FALSE FALSE
## 2 FALSE FALSE FALSE
## 3 FALSE FALSE FALSE
## ... (#elements: 22)
```

Resampling in `mlr` process



Resampling

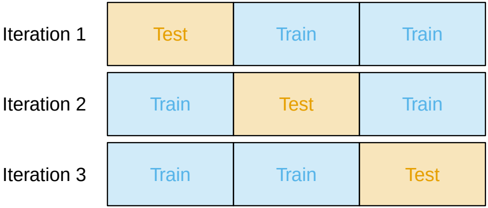
- Aim: Assess the performance of a learning algorithm
- We need to construct a better performance estimator through *resampling*, that uses the data more efficiently.
- All resampling variants operate similar: The data set is split repeatedly into training and tests sets, and we later aggregate (e.g. average) the results.
- The usual trick is to make training sets quite larger (to keep the pessimistic bias small), and to handle the variance introduced by smaller test sets through many repetitions and averaging of results.
- `mlr` provides differnet resample strategies
  - Cross Validation `CV`
  - Leave-one-out CV `LOO`
  - Out-of-bag bootstrap and other methods `Bootstrap`
  - Holdout `Holdout`
  - Subsample (Monte-Carlo CV) `Subsample`

```
> rdesc = makeResampleDesc("CV", iters = 5L, stratify = TRUE)
```

Cross Validation I

- Most popular resampling strategy: Cross validation with 5 or 10 folds
- Process
  - Split the data into k roughly equally-sized partitions
  - Use each of the k partitions once as a test set and the remaining $k-1$ training sets to fit the model
  - Obtain k test error and average them

Example of 3-fold cross-validation



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)
```

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"

R Example: Algorithms I

- Benchmark experiment - Compare 4 algorithms: glmnet, naiveBayes, randomForest & ksvmn

```
> learners = makeLearners(c("glmnet", "naiveBayes", "randomForest", "ksvm"),
+   type = "classif", predict.type = "prob")
>
> set.seed(3)
> rdesc = makeResampleDesc("CV", iters = 10L, stratify = TRUE)
> rinst = makeResampleInstance(rdesc, task.train)
```

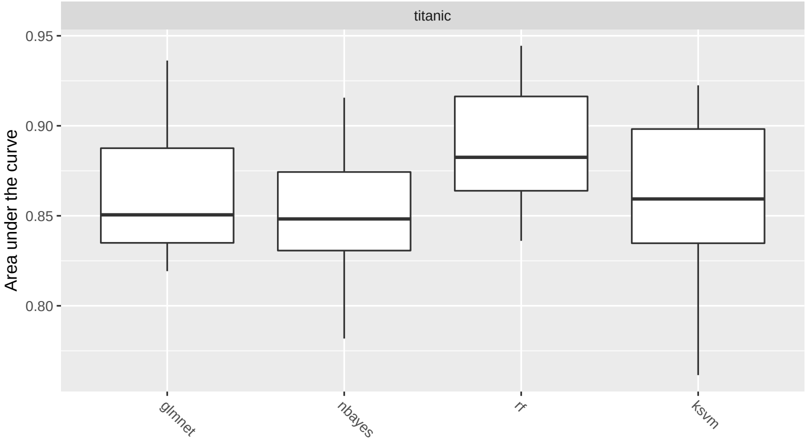
- Calculation can take a while $\rightarrow$ parallelization

```
> library(parallelMap)
>
> parallelStart()
> bmr = benchmark(learners, task.train, rinst, measures = auc)
> parallelStop()
```

R Example: Algorithms II

- Plot the benchmark results

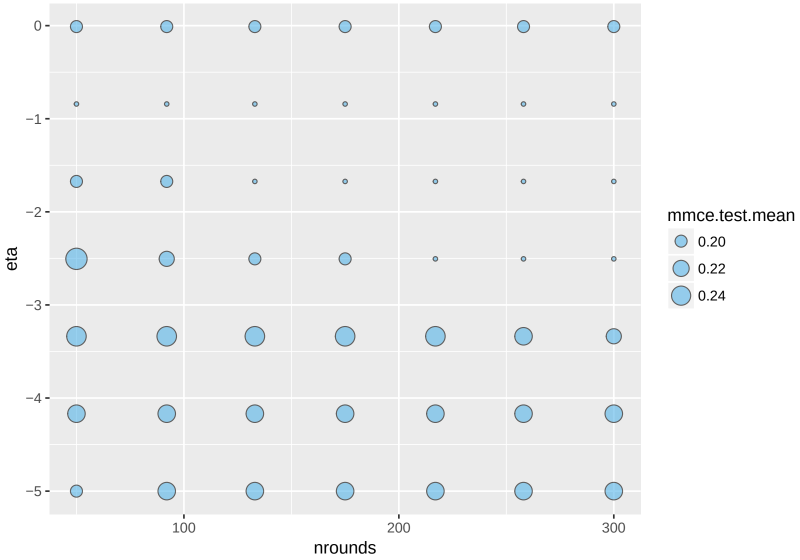
```
> plotBMRBoxplots(bmr)
```



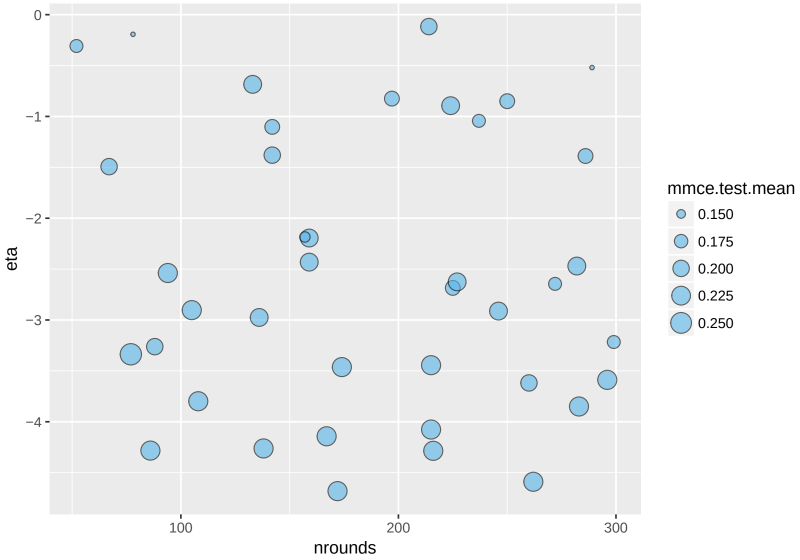
Hyperparameter Tuning

- Aim: Optimize parameters or decisions for an machine learning algorithm w.r.t. the estimated prediction error
- 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 - 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

Grid search



Random search



Tuning in mlr

- Create a set of parameters

```
> ps = makeParamSet(
+   makeIntegerParam("nrounds", lower = 50, upper = 300),
+   makeNumericParam("eta", lower = -5, upper = -0.01,
+     trafo = function(x) 2^x)
+ )
> ctrl = makeTuneControlGrid(resolution = 7)
```

- Optimize the hyperparameter of learner

```
> res.grid = tuneParams(lrn, task = sonar.task, par.set = ps,
+   resampling = rdesc, control = ctrl,
+   measures = mmce)
```

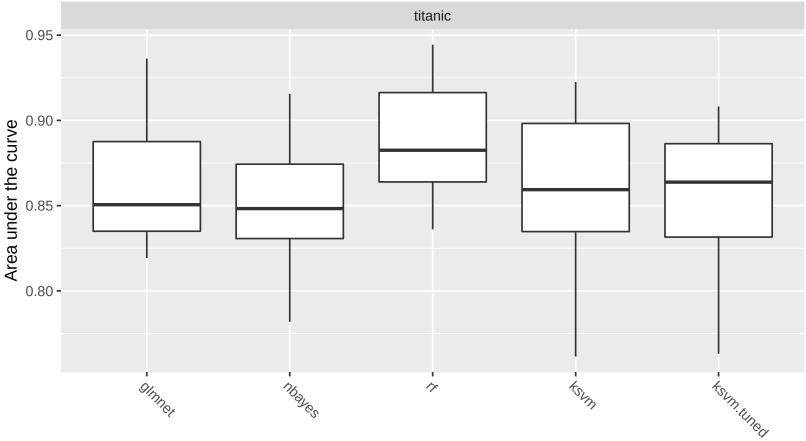
R Example: Tuning I

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

```
> 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)
+ )
> tune.ctrl = makeTuneControlRandom(maxit = 10L)
> classif.ksvm.tuned = makeTuneWrapper(learners$classif.ksvm, resampling = cv3,
+                                      par.set = par.set, control = tune.ctrl)
> bmr2 = benchmark(classif.ksvm.tuned, task.train, rinst)
```

R Example: Tuning II

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



Performance measures

- Different performance measures for different types of learning problems
- In `mlr` you can check out all implemented measures via `https://mlr-org.github.io/mlr/articles/tutorial/devel/measures.html`

Performance measure for Classification I

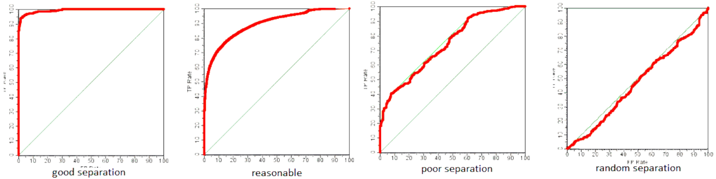
- In our Titanic example we have a classification problem
- Confusion matrix:
  contingency table of predictions $\hat{y}$ and true labels y

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 → ROC Curve (Receiver Operating Characteristic)



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

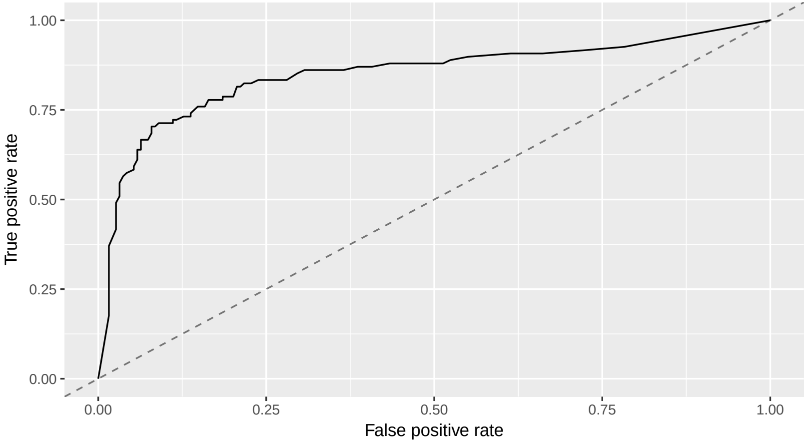
R Example: Random Forest I

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

```
> split = makeResampleInstance(hout, task.train)
> mod = train(learners$classif.randomForest, task.train,
+   subset = split$train.inds[[1]])
> pred = predict(mod, task.train, subset = split$test.inds[[1]])
> df = generateThreshVsPerfData(pred, list(fpr, tpr, acc))
```

R Example: Random Forest II

```
> plotROCCurves(df)
```



R Example: Random Forest III

```
> calculateROCMeasures(pred)

##     predicted
## true 0         1
##    0 163       26        tpr: 0.73 fnr: 0.27
##    1 29        79        fpr: 0.14 tnr: 0.86
##      ppv: 0.75 for: 0.15 lrp: 5.32 acc: 0.81
##      fdr: 0.25 npv: 0.85 lrm: 0.31 dor: 17.08
##
##
## Abbreviations:
## tpr - True positive rate (Sensitivity, Recall)
## fpr - False positive rate (Fall-out)
## fnr - False negative rate (Miss rate)
## tnr - True negative rate (Specificity)
## ppv - Positive predictive value (Precision)
## for - False omission rate
## lrp - Positive likelihood ratio (LR+)
## fdr - False discovery rate
## npv - Negative predictive value
## acc - Accuracy
## lrm - Negative likelihood ratio (LR-)
## dor - Diagnostic odds ratio
```

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

- Use `iml` package

```
> library(iml)
```

- We use our trained model `mod`

```
> mod

## Model for learner.id=classif.randomForest; learner.class=classif.randomForest
## Trained on: task.id = titanic; obs = 594; features = 12
## Hyperparameters:
```

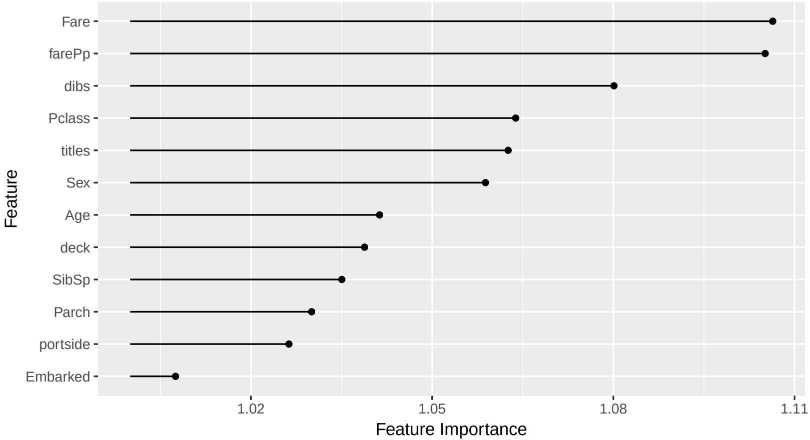
- Extract features
- Create IML model

```
> X = train[which(names(train) != "Survived")]
> iml.mod = Predictor$new(mod, data = X, y = train$Survived, class = 2)
```

Feature Importance

- What were the most important features?

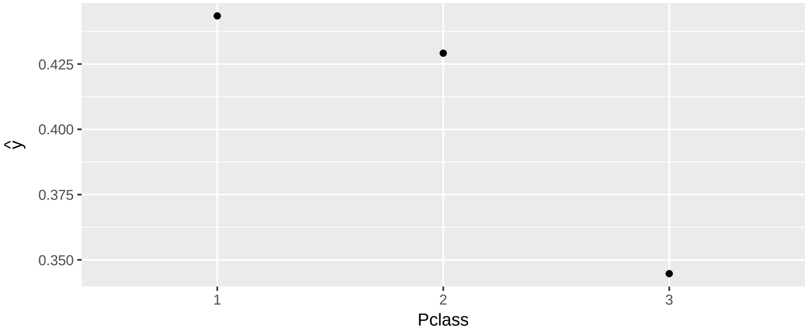
```
> imp = FeatureImp$new(iml.mod, loss = "ce")
> plot(imp)
```



Partial Dependence Plots

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

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



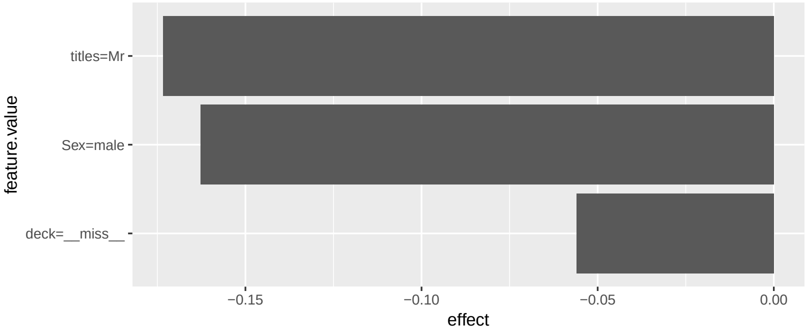
Local Linear Models (LIME)

- Explain a single prediction with LIME

```
> X[1,]

##   Pclass  Sex Age SibSp Parch Fare Embarked titles  dibs farePp
## 1      3 male  22     1     0 7.25        S     Mr FALSE  3.625
##        deck portside
## 1 __miss__ __miss__

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



mlr

mlrMBO I

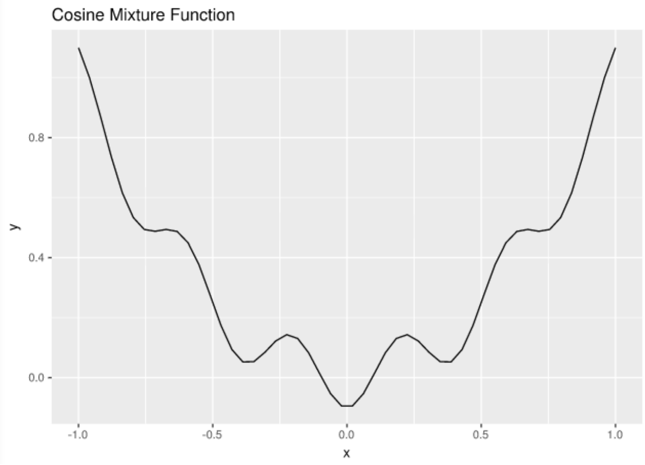
- `mlrMBO` - Bayesian Optimization and Model-Based Optimization - `https://github.com/mlr-org/mlrMBO`
- Goal: optimize *expensive black box functions* by *model-based optimization* (aka Bayesian optimization)
- Create an unified interface with the general `mlrMBO` workflow
  1. Define **objective function** 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

mlrMBO III

- Example: Minimize a cosine-like function with an initial design of 5 points and 10 sequential MBO iterations



mlrMBO IV

- Define **objective function** and its parameters using the package `smoof`

```
> obj.fun = makeSingleObjectiveFunction(
+   name = "my_sphere",
+   fn = function(x) {
+     sum(x*x) + 7
+   },
+   par.set = makeParamSet(
+     makeNumericVectorParam("x", len = 2L, lower = -5, upper = 5)
+   ),
+   minimize = TRUE
+ )
```

- Generate **initial design** (optional)

```
> des = generateDesign(n = 5,
+   par.set = getParamSet(obj.fun), fun = lhs::randomLHS)
> des$y = apply(des, 1, obj.fun)
```

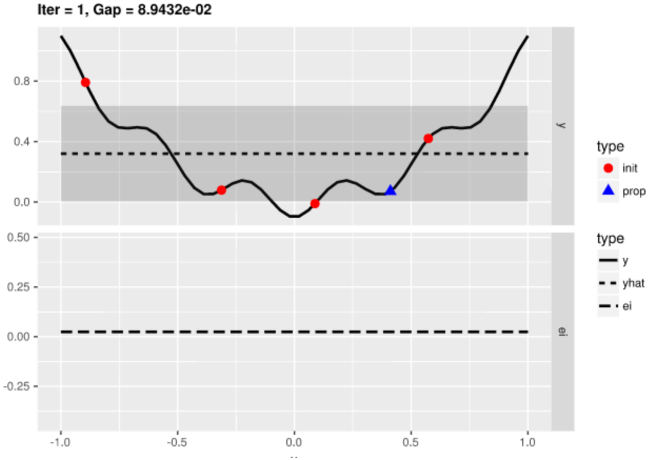
mlrMBO V

- Define mlr' learner for **surrogate model** (optional)

```
> surr.km = makeLearner("regr.km",
+   predict.type = "se", covtype = "matern3_2",
+   control = list(trace = FALSE))
```

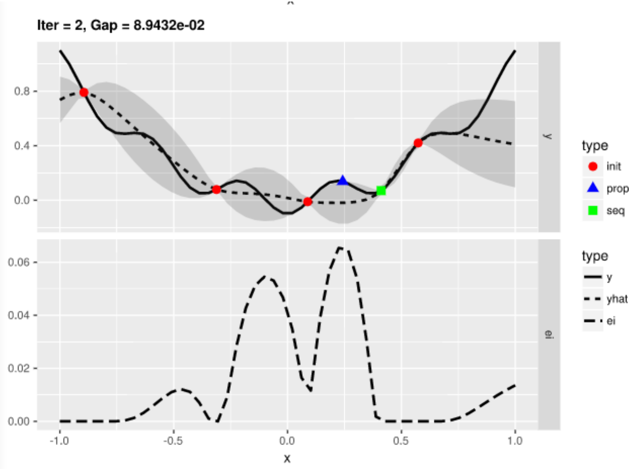
- Set up a **MBO control** object
- Start the optimization with `mbo()`
- Visualization of the optimization

mlrMBO VI



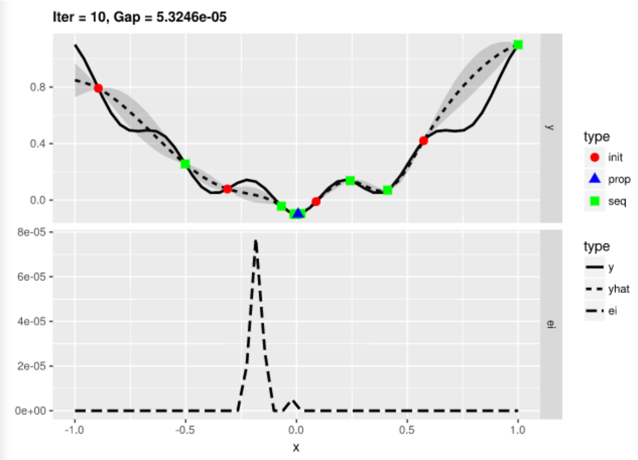
Iter = 1, Gap = 8.9432e-02
0.8
0.4
0.0
0.50
0.25
0.00
-0.25
-1.0
-0.5
0.0
0.5
1.0
y
ei
type
init
prop
type
y
yhat
ei

mlrMBO VII



Iter = 2, Gap = 8.9432e-02
0.8
0.4
0.0
y
0.06
0.04
0.02
0.00
ei
-1.0
-0.5
0.0
0.5
1.0
x
type
init
prop
seq
type
y
yhat
ei

mlrMBO VIII



Iter = 10, Gap = 5.3246e-05
0.8
0.4
0.0
y
8e-05
6e-05
4e-05
2e-05
0e+00
ei
-1.0
-0.5
0.0
0.5
1.0
x
type
init
prop
seq
type
y
yhat
ei

mlrCPO

- `mlrCPO` - Composable Preprocessing Operators for mlr - `https://github.com/mlr-org/mlrCPO`
- Preprocessing operators (e.g. imputation or PCA) in the form of R objects
- Generate machine learning piplines that combine preprocessing and model fitting

```
> task = iris.task
> task %<>>% cpoScale(scale = FALSE) %>>% cpoPca() %>>%  # pca
+   cpoFilterChiSquared(abs = 3) %>>%  # filter
+   cpoModelMatrix(~ 0 + .^2)  # interactions
```

There is more ... I

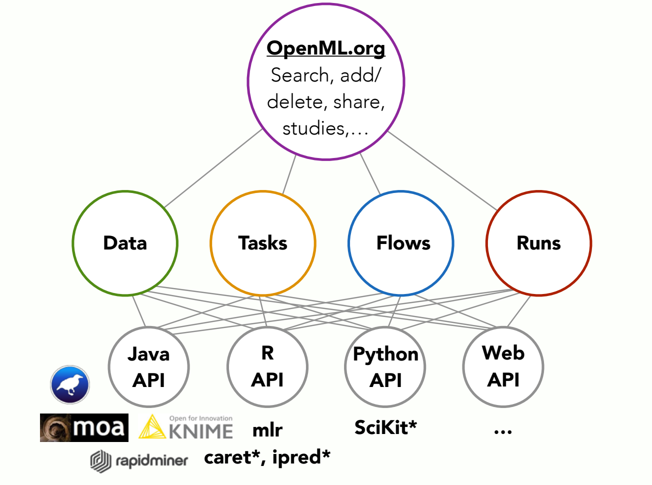
- Regression, Clustering and Survival analysis
- Regular cost-sensitive learning (class-specific costs)
- Cost-sensitive learning (example-dependent costs)
- Imbalancy correction
- Wrappers
- Multi-Label learning

There is more … II

- Bayesian optimization
- Multi-criteria optimization
- Ensembles, generic bagging and stacking
- Automatic model selection
- Adaptive tuning
- Some interactive plots with `ggvis`
- …

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

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`

Outlook

We are working on

- Even better tuning system
- 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!