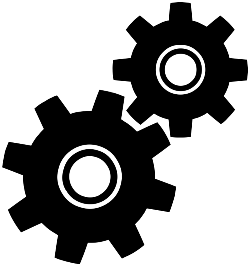
Machine Learning in R: Package mlr

Bernd Bischl
Computational Statistics, LMU

Agenda

- About `mlr`
- Features of `mlr`
  - Tasks and Learners
  - Train, Test, Resample
  - Benchmarking
  - Hyperparameter Tuning
  - Performance Visualization
- `iml` - Interpretable Machine Learning
- OpenML



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

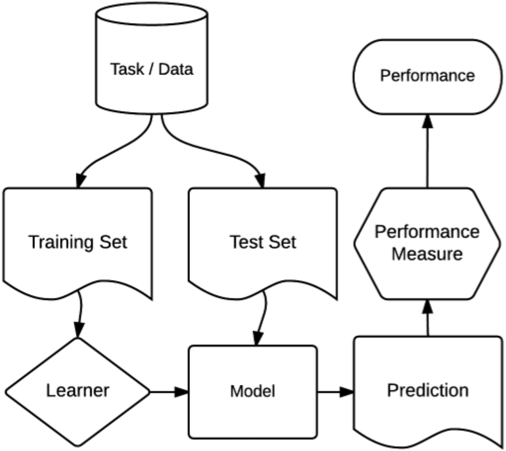
- Project home page

  `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

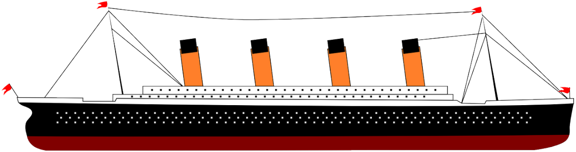
- Unified interface for the basic building blocks: tasks, learners, hyperparameters, ...



The mlr process

Titanic: Machine Learning from Disaster

- Titanic sinking on April 15, 1912
- Data on Kaggle: `https://www.kaggle.com/c/titanic`
- 1502 out of 2224 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
- 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)
```

Preprocessing II

- 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

```
> summarizeColumns(data)[, -c(5, 6,7)]

##           name      type   na     mean    min    max nlevs
## 1  PassengerId   integer    0 655.0000   1.00 1309.0     0
## 2     Survived    factor  418       NA 342.00  549.0     2
## 3       Pclass    factor    0       NA 277.00  709.0     3
## 4         Name character    0       NA   1.00    2.0  1307
## 5          Sex    factor    0       NA 466.00  843.0     2
## 6          Age   numeric  263  29.8811   0.17   80.0     0
## 7        SibSp   numeric    0   0.4989   0.00    8.0     0
## 8        Parch   numeric    0   0.3850   0.00    9.0     0
## 9       Ticket    factor    0       NA   1.00   11.0   929
## 10        Fare   numeric    1  33.2955   0.00  512.3     0
## 11       Cabin    factor 1014       NA   1.00    6.0   186
## 12    Embarked    factor    2       NA 123.00  914.0     3
## 13       train   logical    0       NA 418.00  891.0     2
```

Preprocessing I

```
> # 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

## Warning:  NAs introduced by coercion

> # Drop stuff we cannot easily model on
> data = dropNamed(data,
+   c("Cabin","PassengerId", "Ticket", "Name"))
```

Imputation

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

```
> 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

- Split back into train and test
- Create classification problem

```
> dtrain = data[data$train, ]
> dtrain$train = NULL
> dtest = data[!data$train, ]
> dtest$train = NULL
>
> task = makeClassifTask(id = "titanic", data = dtrain,
+   target = "Survived", positive = "1")
```

Tasks II

```
> print(task)

## Supervised task: titanic
## Type: classif
## Target: Survived
## Observations: 891
## Features:
##    numerics     factors     ordered functionals
##           5           5           0           0
## Missings: FALSE
## Has weights: FALSE
## Has blocking: FALSE
## Has coordinates: FALSE
## Classes: 2
##   0   1
## 549 342
## Positive class: 1
```

mlr

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"))[1:5, 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 - Test - Eval I

```
> lrn = makeLearner("classif.rpart", predict.type = "prob")
> mod = train(lrn, task)
> pred = predict(mod, newdata = dtest)
> head(as.data.frame(pred))

##     truth prob.0 prob.1 response
## 892  <NA> 0.8981 0.1019        0
## 893  <NA> 0.8571 0.1429        0
## 894  <NA> 0.8981 0.1019        0
## 895  <NA> 0.8981 0.1019        0
## 896  <NA> 0.3297 0.6703        1
## 897  <NA> 0.8981 0.1019        0
```

Train - Test - Eval II

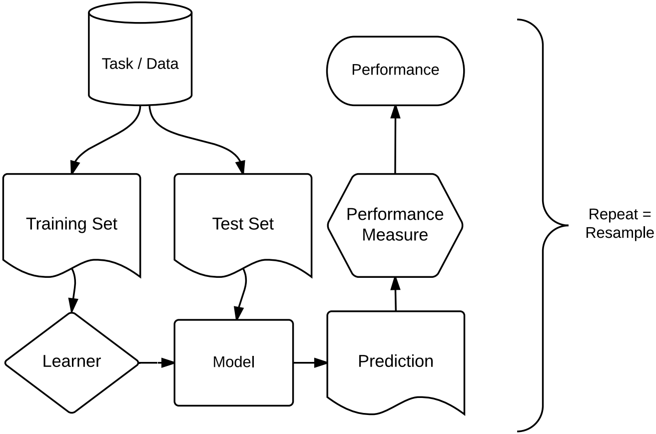
- We don't have labels for the true test set here
- Let's eval on train set now
- Which is dangerous in practice and not recommended in general!

```
> pred = predict(mod, newdata = dtrain)
> performance(pred, measures = list(mlr::acc, mlr::auc))

##    acc    auc
## 0.8541 0.8843
```

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 I

- 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

```
> lrn = makeLearner("classif.rpart", predict.type = "prob")
> 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.rpart
## Aggr perf: acc.test.mean=0.8013,auc.test.mean=0.8439
## Runtime: 0.0378931
```

Crossvalidation in mlr II

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

##   iter    acc    auc
## 1    1 0.8148 0.8184
## 2    2 0.8182 0.8667
## 3    3 0.7710 0.8467

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

##   id truth prob.0 prob.1 response iter  set
## 1  8     0 0.8831 0.1169        0    1 test
## 2 15     0 0.1613 0.8387        1    1 test
## 3 17     0 0.8831 0.1169        0    1 test
## 4 21     0 0.8831 0.1169        0    1 test
## 5 25     0 0.2500 0.7500        1    1 test
## 6 27     0 0.8831 0.1169        0    1 test
```

Resampling in mlr

- Holdout (Train-Test): “Holdout”
- Cross Validation: “CV”
- Leave-one-out: “LOO”
- Subsample (Monte-Carlo CV) “Subsample”
- Out-of-bag bootstrap and other methods “Bootstrap”

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)
```

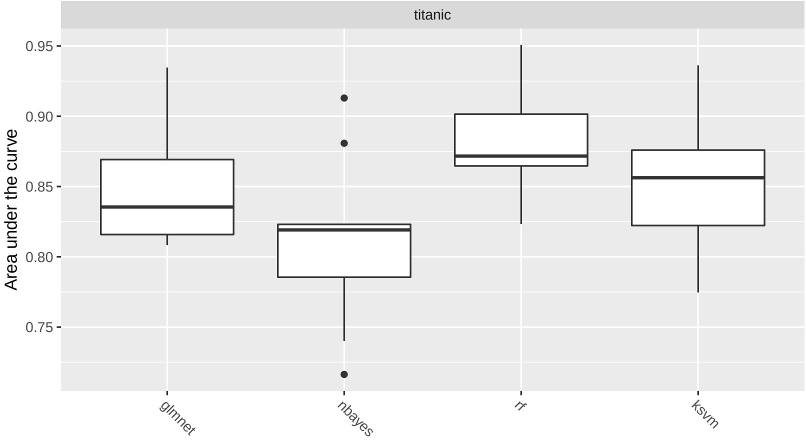
R Example: Algorithms I

- 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, rinst, measures = mlr::auc)
```

R Example: Algorithms II

```
> plotBMRBoxplots(bmr)
```

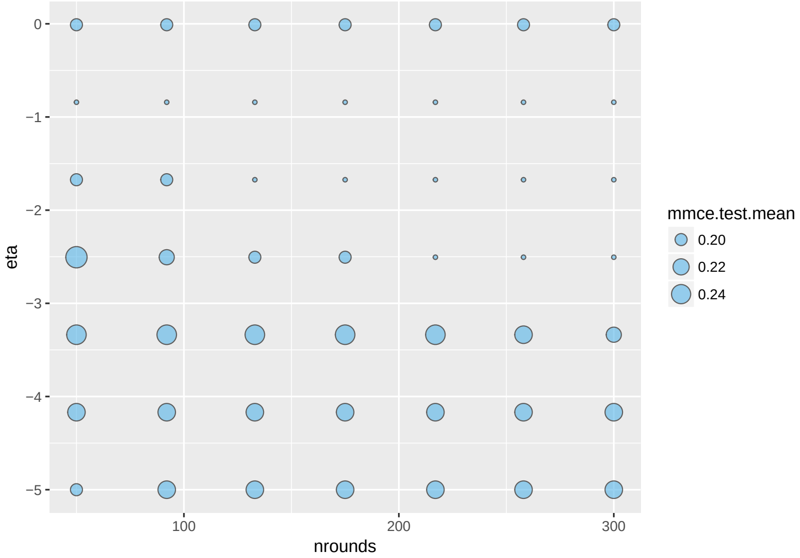


Hyperparameter Tuning

- 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

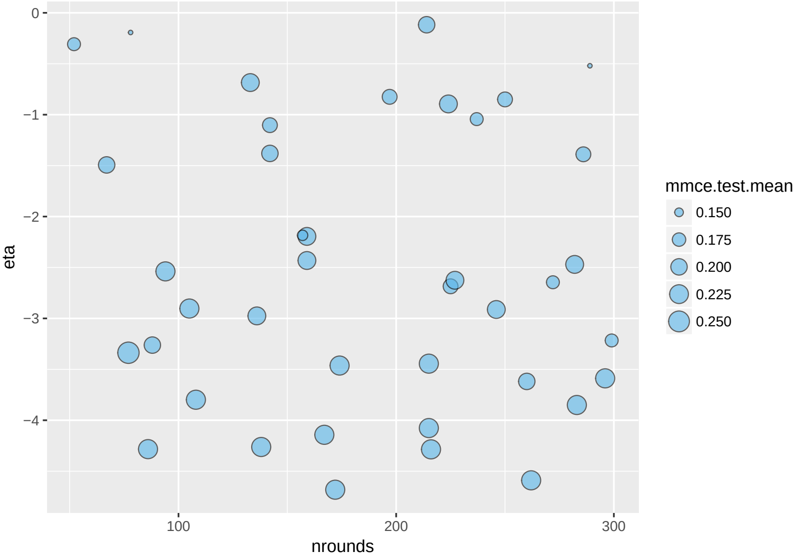
Try all combinations of finite grid
$\rightsquigarrow$ Inefficient, combinatorial explosion, searches irrelevant areas



Random search

Unformly randomly draw configurations,

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



Tuning in mlr I

- Create a set of parameters

```
> 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

```
> ctrl = makeTuneControlGrid(resolution = 7)
> tr = tuneParams(lrn, task = task, par.set = par.set,
+   resampling = rdesc, control = 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 -8.000    -8        0.8143     0.385
## 2 -5.333    -8        0.8152     0.397
## 3 -2.667    -8        0.8150     0.408
## 4  0.000    -8        0.8375     0.381
## 5  2.667    -8        0.8449     0.359
## 6  5.333    -8        0.8413     0.368
```

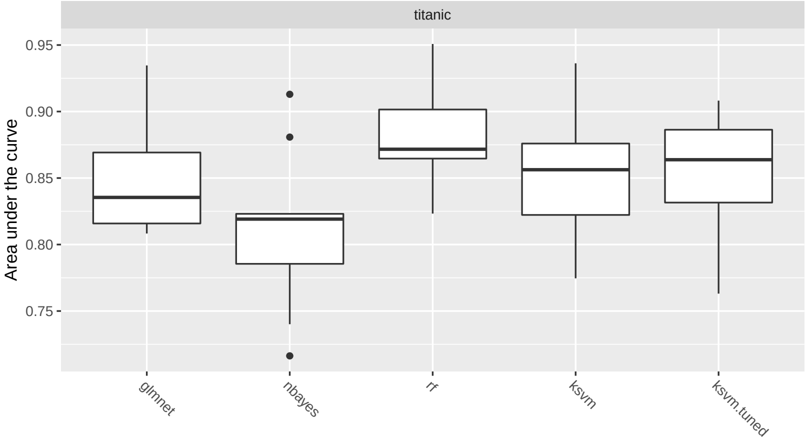
R Example: Tuning I

- 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

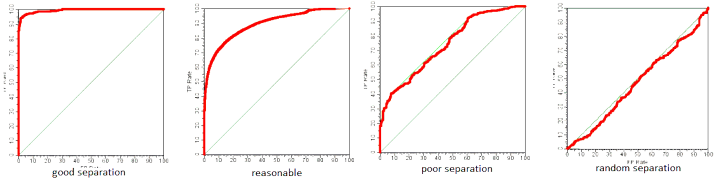
- In our Titanic example we have a classification problem
- Confusion matrix:
  contingency table of predictions $\hat{y}$ and true labels y

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 $\rightarrow$ ROC Curve (Receiver Operating Characteristic)



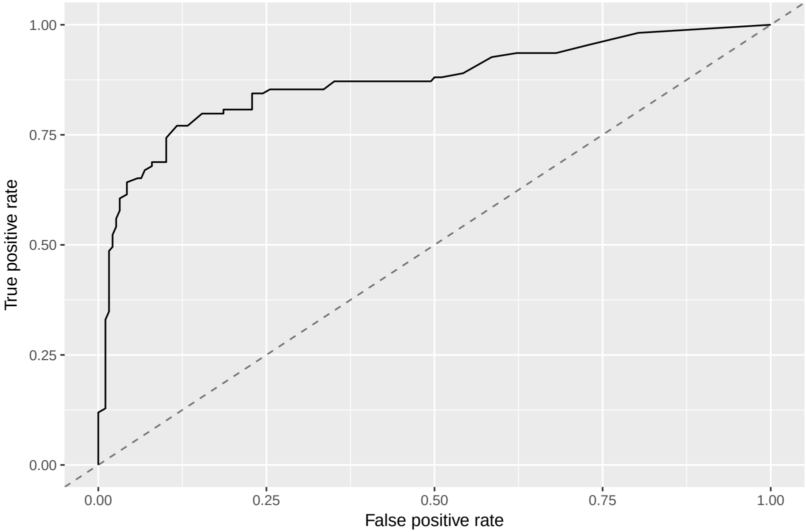
- For measuring the performance we can caluculate the area under the ROC curve (AUC)

R Example: Random Forest I

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

```
> r = holdout(learners$classif.randomForest, task)
> df = generateThreshVsPerfData(r$pred, list(fpr, tpr, acc))
> plotROCCurves(df)
```

R Example: Random Forest II



R Example: Random Forest III

```
> calculateROCMeasures(pred)

##     predicted
## true 0         1
##    0 504       45        tpr: 0.75 fnr: 0.25
##    1 85        257       fpr: 0.08 tnr: 0.92
##      ppv: 0.85 for: 0.14 lrp: 9.17 acc: 0.85
##      fdr: 0.15 npv: 0.86 lrm: 0.27 dor: 33.86
##
##
## 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
```

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

- Use `iml` package

```
> library(iml)
```

- We use our trained model `mod`

```
> mod

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

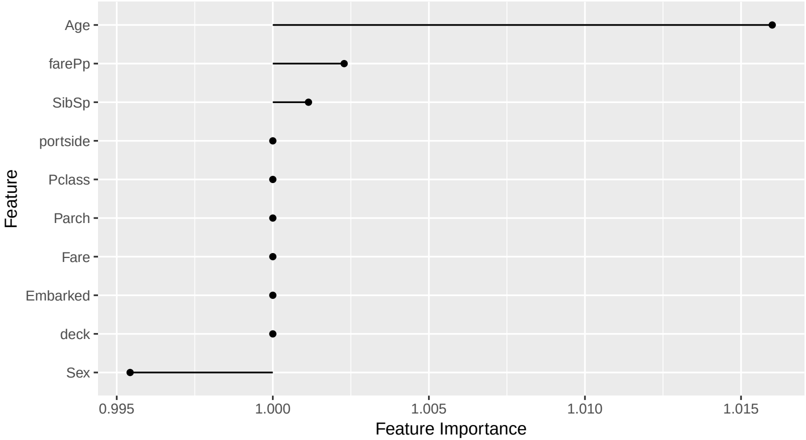
- Extract features
- Create IML model

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

Feature Importance

- What were the most important features?

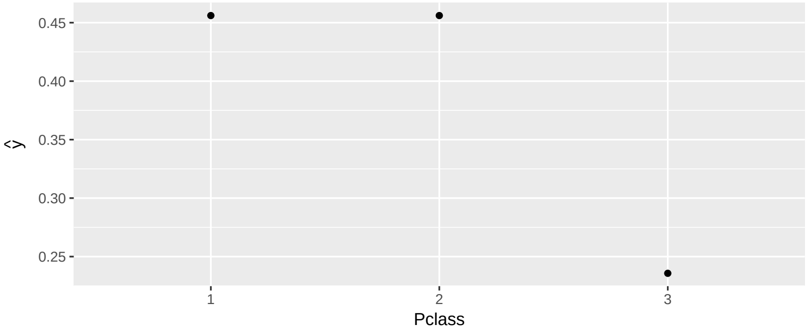
```
> imp = FeatureImp$new(iml.mod, loss = "ce")
> plot(imp)
```



Partial Dependence Plots

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

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



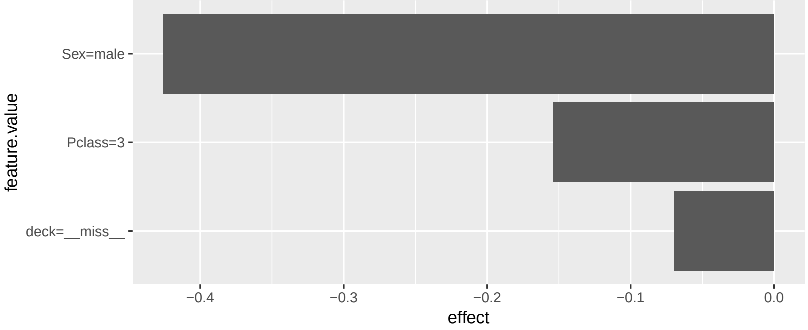
Local Linear Models (LIME)

- Explain a single prediction with LIME

```
> X[1,]

##   Pclass  Sex Age SibSp Parch Fare Embarked farePp     deck p
## 1      3 male  22     1     0 7.25        S  3.625 __miss__ _

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



There is more ...

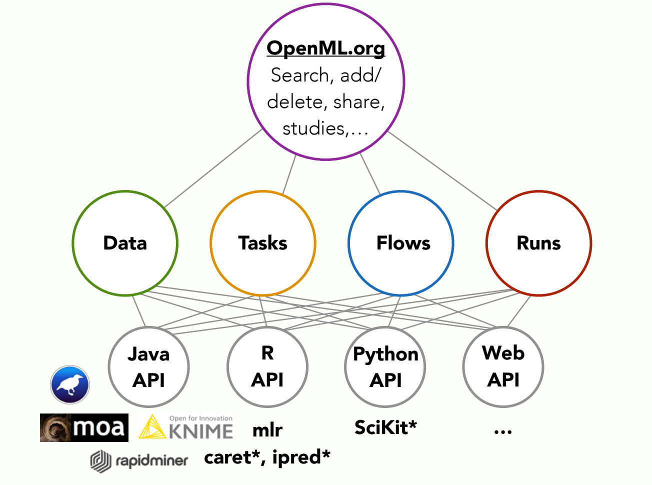
- Regression, Clustering and Survival analysis
- Cost-sensitive learning
- Multi-Label learning
- Imbalancy correction
- Wrappers
- Bayesian optimization
- Multi-criteria optimization
- Ensembles, generic bagging and stacking
- ...

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`
- ...

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`

Thanks!