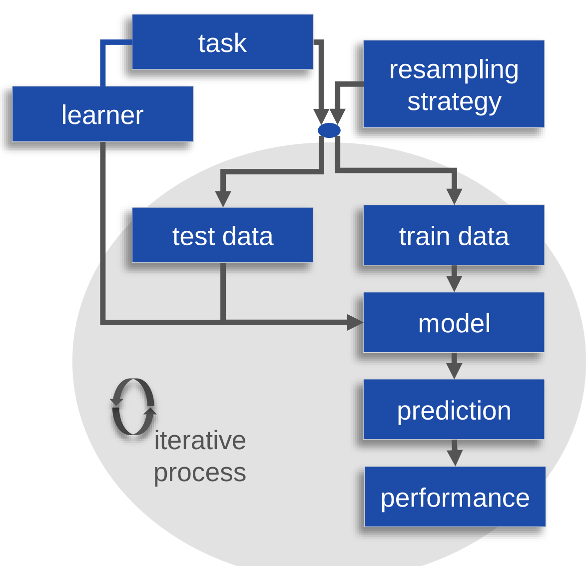
Machine Learning with R

Introduction

***mlr* offers a unified interface for the basic building blocks: tasks, learners, hyperparameters, …**

mlr package provides a generic, object-oriented, and extensible framework for **classification**, **regression**, survival analysis and **clustering**. It provides a unified interface to more than 160 basic **learners** and includes meta-algorithms and model selection techniques to improve and extend the functionality of basic learners with, e.g., **hyperparameter tuning**, feature selection, and ensemble construction. Parallel high-performance computing is natively supported

For more help, please see the mlr tutorial



Quickstart

```
task = makeClassifTask(data = iris,
  target = "Species")

learner = makeLearner("classif.lda")

train(learner, task,
  subset = seq(1, n, by = 2))
```

Basics

Task

Encapsulate data set and specify target variable

Task types

```
makeRegrTask(id = "bh",
  data = BostonHousing,
  target = "medv")
makeClassifTask(id =
  "BreastCancer", data = df,
  target = "Class")
makeSurvTask(data = lung,
  target = c("time", "status"))
makeClusterTask(data = mtcars)
makeMultilabelTask(id = "multi",
  data = yeast, target = labels)
makeCostSensTask(data = df,
  cost = cost)
```

Example tasks can be found here, e.g. `bh.task` for regression

Predict

Predict target

a) If training & test data set available →
```
predict(mod, task = bh.task,
  subset = test.set)
```
b) If **new** data should be predicted →
```
predict(mod, newdata = iris.test)
```

Performance

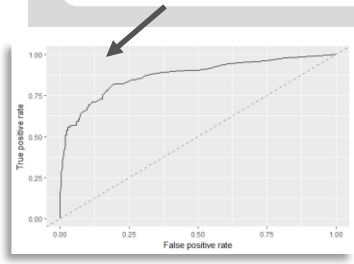
Check performance (example)
```
performance(pred, measures = auc)
```

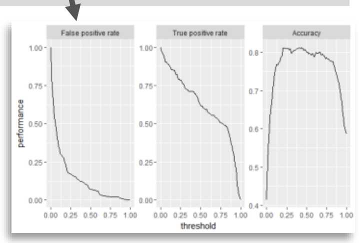
View all performance measures
```
listMeasures()
```

```
df = generateThreshVsPerfData(pred,
  list(fpr, tpr, acc))
plotThreshVsPerf(df)

plotROCCurves(df)
```

Use single measure or list





Learner

Specify learning method and set hyperparameters

```
makeLearner(“classif.randomForest”,
  predict.type = “prob")
makeLearner(“regr.gbm",
  predict.type = “se”)
makeLearner(“cluster.kmeans",
  centers = 5,
  predict.type = “response”)
```

You can also create learners for survival analysis, multilabel classification and cost-sensitve classification

Learners are always constructed like
`“task_type.<R_method_name>”`

View all possible learners depending on your task `listLearners()`

Resampling

Define a resampling strategy (example)
```
rdesc = makeResampleDesc("CV",
  iters = 3, predict = both,
  stratify = TRUE)
```

Resampling strategies
- Cross-validation `("CV")`
- Leave-one-out cross-validation `("LOO")`
- Repeated cross-validation `("RepCV")`
- Out-of-bag bootstrap and other variants like *b632* `("Bootstrap")`
- Subsampling `("Subsample")`
- Holdout (training/test) `("Holdout")`

Predict: Get prediction on `“training”` / `“test”` data set or on `“both”`

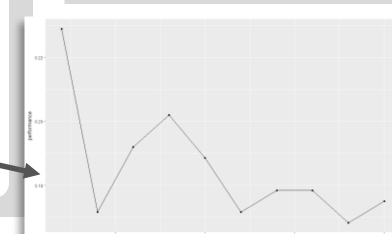
`Stratify = TRUE` to conduct stratified sampling

Access the results (example)
```
resample(learner, task, rdesc)
```

Visualization

mlr relies on generating functions naming is always like
`generate<FunctionPurpose>Data`
e.g. `generateThreshVsPerfData(pred, measures = list(fpr, fnr, mmce))`

```
d = generateLearningCurveData(lrn,
  task)
plotLearningCurve(d)
```



Train

Fit model to given data set
```
mod = train(learner, task)
```

Accessing learner models
- train returns object of class WrapperModel → contains also information about learner, task, features and observations

```
names(mod)
getLearnerModel(mod)
```

use training data to fit the model
```
n = getTaskSize(task)
trainSet = seq(1, n, by = 2)
train(learner, task,
  subset = trainSet)
```

Benchmarking

Evaluate multiple tasks / learners
```
bmr = benchmark(learnersList, task,
  rdesc)
```

Accessing benchmark experiment
```
getBMRAggrPerformances(bmr,
  as.df = TRUE)
getBMRPerformances(bmr,
  as.df = TRUE)
getBMRPredictions(bmr)
getBMRLearners(bmr)
getBMRMeasures(bmr)
```

Merging benchmark results
```
mergeBenchmarkResults(list(bmr,
  bmr2))
```

Imputation

Techniques
- Using simple functions, e.g. mode and mean
```
impute(data, classes =
  list(integer = imputeMean(),
  factor = imputeMode()),
  dummy.classes = "integer")
```
- Using supervised learning algorithms for imputation
```
impute(airquality, target = “y",
  cols = list(Wind =
  imputeLearner("classif.rpart")),
  dummy.cols = c("Solar.R",
  "Wind"))
```

Advanced

Tuning

In order to tune a machine learning algorithm, you have to specify

- the search space
```
ps = makeParamSet(
  makeNumericParam("C",
  lower = 0.01, upper = 0.1))
```
- the optimization algorithm
```
ctrl = makeTuneControlRandom(
  maxit = 100L)
```
For numeric search space and no randomizazion:
```
ctrl = makeTuneControlGrid(
  resolution = 15L)
```
- an evaluation method, i.e., a resampling strategy and a performance measure
```
rdesc = makeResampleDesc("CV",
  iters = 3L)
measure = acc
```

Performing the tuning
```
res = tuneParams(learner, task,
  rdesc, ps, ctrl, measure)
```

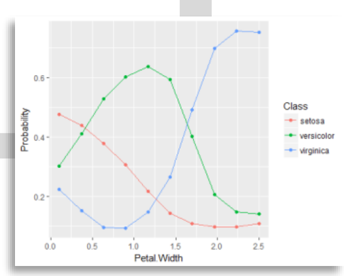
Accessing the tuning result
- Best found settings `res$x`
- Corresponding performance `res$y`

Further options
- Multiplexer model
```
lrn = makeModelMultiplexer(
  learnersList)
tuneParams(lrn, iris.task, rdesc,
ps, ctrl)
```
- Multi-criteria evaluation and optimization
```
tuneParamsMultiCrit("classif.ksvm"
  , task = sonar.task,
  resampling = rdesc,
  par.set = ps,
  measures = list(fpr, fnr),
  control = ctrl)
```
- Accessing results
```
plotTuneMultiCritResultGGVIS(res)
```

Partial Dependence Plots

Generate a grid for the chosen feature
```
pd = generatePartialDependenceData(
mod, task, "feature")
```

Plot the partial dependencies
```
plotPartialDependence(pd)
```



Nested Resampling

1. Specify inner resampling method
```
inner = makeResampleDesc(...)
```
2. Create learner with inner resampling method
- Tuning wrapper
```
lrn = makeTuneWrapper(...,
resampling=inner, ...)
```
- feature selection wrapper approach
```
lrn = makeFeatSelWrapper(...,
resampling=inner, ...)
```
- feature selection with filter approach
```
lrn = makeFilterWrapper(...)
lrn = makeTuneWrapper(lrn, ...,
  resampling = inner, ...)
```
3. Specify outer resampling method
```
outer = makeResampleDesc(...)
```
4. Call resample function
```
r = resample(learner = lrn, ...,
  resampling=outer, ...)
r$extract
```

Benchmarking with Nested resampling
Given learners with specified inner resampling
```
tasks = list(task1, task2)
lrns = list(lrn1, lrn2)
outer = list(outer1, outer2)
benchmark(lrns, tasks, outer, ...)
```

Feature Selection

Filter methods: assign importance values to features
```
fv = generateFilterValuesData(task,
  method = "information.gain")
plotFilterValues(fv)
filtered.task = filterFeatures(task,
  fval = fv, abs=2)
```

Options: specify `method` instead of `fval` and `perc` or `threshold` instead of `abs`
Fuse learner with filter method:
```
lrn = makeFilterWrapper(learner =
  „classif.fnn", fw.method =
  „information.gain", fw.abs = 2)
getFilteredFeatures(mod)
```

Wrapper methods: use performance of learning strategies: *GA, Exhaustive, Random, Sequential*
```
ctrl = makeFeatSelControlGA(...)
sfeats = selectFeatures(...,
  control = ctrl)
analyzeFeatSelResult(sfeats)
```

Fuse learner with feature selection
```
lrn = makeFeatSelWrapper(
  "surv.coxph",..., control = ctrl)
mod = train(lrn, task = task)
getFeatSelResult(mod)
```

Machine Learning with R

Configuration

Set global options for the current R session.

```
configureMlr(show.info = TRUE,
on.learner.error =
  'stop',
on.par.without.desc =
  'stop',
show.learner.output = TRUE, … )
```

Handle Errors

If an error is caught, R exception is generated

Set ‚quiet' to ignore error and ‚warn' to produce a warning

Check Parameters

Example:
```
configureMlr(on.learner.error =
"warn")
```
Warning instead of exception is issued and object (mod) of class FailureModel is created
- `isFailureModel(mod)`
- `getFailureModelMsg(mod)`