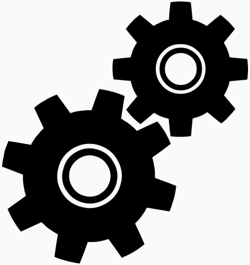
Machine Learning with R at LRZ

Introduction to mlr

Janek Thomas, Johannes Albert-von der Gönna

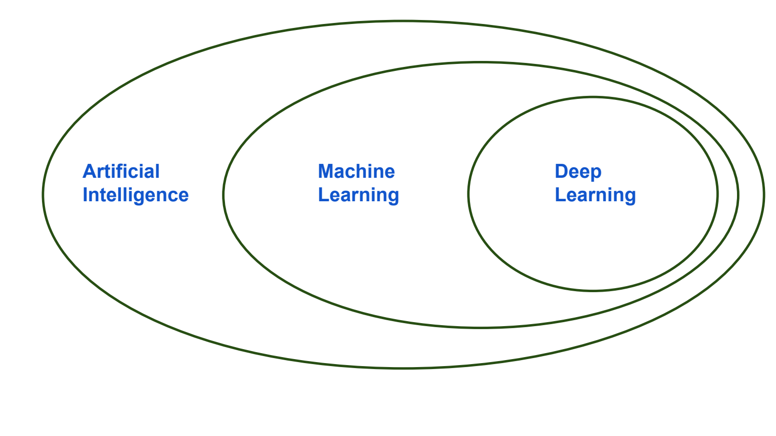
2018-11-27

WHAT IS MACHINE LEARNING

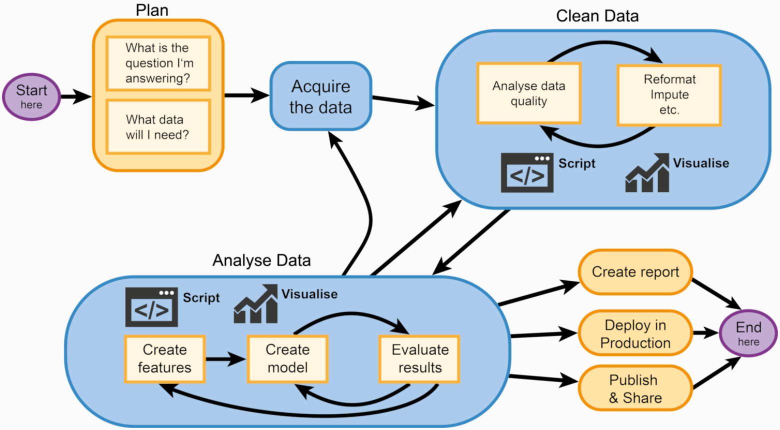


Machine Learning is a method of teaching computers to make predictions based on some data.

DATA SCIENCE AND MACHINE LEARNING



TYPICAL WORKFLOW



INTRODUCTION

MOTIVATION: MACHINE LEARNING IN R

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

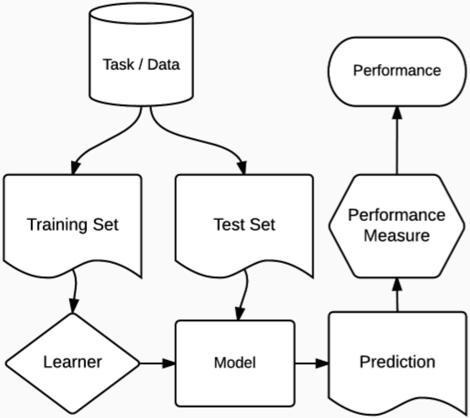
Our goal: A domain-specific language for ML concepts!

MOTIVATION: MLR

- 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 in 2017
- About 30K lines of code, 8K lines of unit tests

MOTIVATION: MLR

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



BASIC FEATURES OF MLR

- Tasks and Learners
- Train, Test, Resample
- Performance
- Benchmarking
- Hyperparameter Tuning
- Nested Resampling
- Parallelization

LEARN MORE

- Extensive Tutorial covers *all* features in mlr: https://mlr-org.github.io/mlr/
- Tuning
- Resampling (with blocking)
- Visualization Topics
- Multilabel Classification, Survival Analysis, Clustering
- Handling Spatial Data
- Functional Data
- Create Custom Learners and Measures
- ...

GETTING HELP

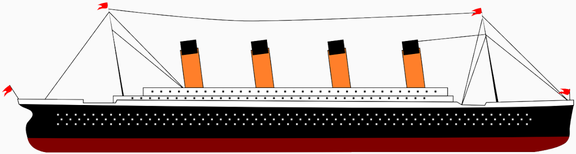
- Ask questions on Stackoverflow: https://stackoverflow.com/questions/tagged/mlr
- Found bugs? Report them: https://github.com/mlr-org/mlr/issues

You want to contribute? - Open a PR on github and join our slack: https://mlr-org.slack.com/

FIRST DATA ANALYSIS

TITANIC - MACHINE LEARNING FROM DISASTER

- Titanic sinking on April 15, 1912
- Data provided on Kaggle:
  https://www.kaggle.com/c/titanic
- 809 out of 1309 passengers died
- Task:
  - Can we predict who survived?
  - Why did people die / Which groups?



TITANIC - 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

TITANIC - DATA SET

```r
load("titanic.rda")
str(data)

## 'data.frame':    1309 obs. of  11 variables:
##  $ Pclass  : Factor w/ 3 levels "1","2","3": 1 1 1 1 1 1 1 1 ...
##  $ Survived: Factor w/ 2 levels "0","1": 2 2 1 1 1 2 2 1 ...
##  $ Name    : chr  "Allen, Miss. Elisabeth Walton" "Allison, Ma"..
##  $ Sex     : Factor w/ 2 levels "female","male": 1 2 1 2 1 2 1 ..
##  $ Age     : num  29 0.917 2 30 ...
##  $ Sibsp   : num  0 1 1 1 1 0 1 0 ...
##  $ Parch   : num  0 2 2 2 2 0 0 0 ...
##  $ Ticket  : Factor w/ 929 levels "110152","110413",..: 188 50 ..
##  $ Fare    : num  211 152 152 152 ...
##  $ Cabin   : Factor w/ 187 levels "","A10","A11",..: 45 81 81 8..
##  $ Embarked: Factor w/ 4 levels "","C","Q","S": 4 4 4 4 4 4 4 4..
```

TITANIC - DATA SET

```
library(mlr)
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
```

TITANIC - PREPROCESSING

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)
```

TITANIC - PREPROCESSING

```r
library(BBmisc)
library(stringi)

# 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$portside = stri_extract_last_regex(data$Cabin, "[0-9]")
data$portside = as.numeric(data$portside) %% 2

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

TITANIC - PREPROCESSING

```
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 1020    0   0   1     0
```

TITANIC - PREPROCESSING

- Impute missing numeric values with median, missing factor values with a separate category
- NB: This is really naive, we should probably embed this in cross-validation

```
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)
```

TITANIC - TASK

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

Classification

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

Regression

- Linear, lasso and ridge
- Boosting
- Trees and forests
- Gaussian processes
- (Deep) Neural Networks
- ...

Clustering

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

Survival

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

WHAT LEARNERS ARE AVAILABLE?

We can explore them on the webpage:

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?

Or ask mlr

```
tab = listLearners(task, warn.missing.packages = FALSE)
tab[1:5, c("class", "package")]

##                   class      package
## 1           classif.ada    ada,rpart
## 2    classif.adaboostm1        RWeka
## 3  classif.bartMachine   bartMachine
## 4      classif.binomial        stats
## 5      classif.boosting adabag,rpart
```

TITANIC - LEARNER

```
lrn = makeLearner("classif.kknn", k = 3, predict.type = "prob")
print(lrn)

## Learner classif.kknn from package kknn
## Type: classif
## Name: k-Nearest Neighbor; Short name: kknn
## Class: classif.kknn
## Properties: twoclass,multiclass,numerics,factors,prob
## Predict-Type: prob
## Hyperparameters: k=3
```

TITANIC - TRAIN

```
set.seed(123)
n = getTaskSize(task)
train.i = sample(n, size = 2/3 * n)
test.i = setdiff(1:n, train.i)

str(train.i)

##  int [1:872] 377 1032 535 1154 1228 60 689 1162 ...

str(test.i)

##  int [1:437] 4 6 8 11 12 18 22 27 ...

mod = train(lrn, task, subset = train.i)
```

TITANIC - MODEL

```
print(mod)

## Model for learner.id=classif.kknn; learner.class=classif.kknn
## Trained on: task.id = titanic; obs = 872; features = 10
## Hyperparameters: k=3

# retrieve model as returned from the third party package
# [NB: knn does not have a training step, mlr just returns the
#  training data which is required in the predict step]
rmodel = getLearnerModel(mod)
```

TITANIC - PREDICT

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

##    id truth prob.0 prob.1 response
## 4   4     0 0.6621  0.338        0
## 6   6     1 0.7358  0.264        0
## 8   8     0 1.0000  0.000        0
## 11 11     0 0.7358  0.264        0
## 12 12     1 0.0000  1.000        1
## 18 18     1 0.0737  0.926        1

head(getPredictionProbabilities(pred))

## [1] 0.338 0.264 0.000 0.264 1.000 0.926
```

TITANIC - PERFORMANCE

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

##   acc   auc
## 0.725 0.786
```

TITANIC - EXTERNAL VALIDATION SET

You can also predict on data not included in the task:

```
test.data = dropNamed(data[test.i, ], "Survived")
pred = predict(mod, newdata = data[test.i, ])
performance(pred, measures = list(mlr::acc, mlr::auc))

##   acc   auc
## 0.725 0.786
```

EXERCISE 1

RESAMPLING

HOLD-OUT IN MLR

```r
task = iris.task
n = getTaskSize(task)
ratio = 2/3

set.seed(123)
train.inds = sample(1:n, n * ratio)
test.inds = setdiff(1:n, train.inds)

lrn.knn1 = makeLearner("classif.knn", k = 1)
mod = train(lrn.knn1, task, subset = train.inds)
preds = predict(mod, task, subset = test.inds)
preds = predict(mod, newdata = iris[test.inds, ]) # alternative
mlr::performance(preds, mmce)

## mmce
## 0.08
```

CROSS-VALIDATION IN MLR

```r
# Define learner:
lrn = makeLearner("classif.randomForest", predict.type = "prob")

# Define resampling strategy:
rdesc = makeResampleDesc("CV", iters = 3, stratify = TRUE)
r = resample(lrn, spam.task, rdesc,
  measure = list(mlr::acc, mlr::auc))
print(r)

## Resample Result
## Task: spam-example
## Learner: classif.randomForest
## Aggr perf: acc.test.mean=0.952,auc.test.mean=0.985
## Runtime: 14.5019
```

CROSS-VALIDATION IN MLR

```
head(r$measures.test)

##   iter   acc   auc
## 1    1 0.949 0.982
## 2    2 0.956 0.988
## 3    3 0.950 0.986

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

##     id   truth prob.nonspam prob.spam response iter  set
## 1 1814 nonspam        0.684     0.316  nonspam    1 test
## 2 1818 nonspam        0.960     0.040  nonspam    1 test
## 3 1822 nonspam        0.922     0.078  nonspam    1 test
## 4 1828 nonspam        0.982     0.018  nonspam    1 test
## 5 1829 nonspam        0.958     0.042  nonspam    1 test
## 6 1843 nonspam        0.948     0.052  nonspam    1 test
```

PARAMETER OF MAKERESAMPLINGDESC

Methods	Parameter
`CV`	`iters` (Number of iterations)
`LOO`	
`RepCV`	`reps` (Repeats for repeated CV)
	`folds` (Folds in the repeated CV)
`Bootstrap`	`iters` (Number of iterations)
`Subsample`	`iters` (Number of iterations)
	`split` (Proportion of training cases)
`Holdout`	`split` (Proportion of training cases)

For instance 10-fold cross validation:

```
makeResampleDesc(method = "CV", iters = 10)

## Resample description: cross-validation with 10 iterations.
## Predict: test
## Stratification: FALSE
```

POSSIBLE WAYS TO USE CROSS VALIDATION

1. Explicitly define resampling:

```
rdesc = makeResampleDesc("CV", iters = 10)
rdesc = cv10

res1 = resample("classif.randomForest", iris.task, resampling = rdesc,
  show.info = FALSE)
res2 = resample("classif.randomForest", iris.task, resampling = cv10,
  show.info = FALSE)

res1
```

```
## Resample Result
## Task: iris-example
## Learner: classif.randomForest
## Aggr perf: mmce.test.mean=0.047
## Runtime: 0.228311
```

Other pre defined objects are `cv2`, `cv3` and `cv5`.

POSSIBLE WAYS TO USE CROSS VALIDATION

2. Use `crossval`:

```
res3 = crossval("classif.randomForest", iris.task, iters = 10,
  show.info = FALSE)
res3
```

```
## Resample Result
## Task: iris-example
## Learner: classif.randomForest
## Aggr perf: mmce.test.mean=0.053
## Runtime: 0.449039
```

Similar functions are `repcv`, `holdout`, `subsample`, `bootstrapOOB`, `bootstrapB632` and `bootstrapB632plus`.

CLASSIFICATION ANALYSIS: BENCHMARKING

```r
# quick way to compare learners with identical train/test splits
task = iris.task
learners = list(
  makeLearner("classif.knn", k = 3),
  makeLearner("classif.lda"),
  makeLearner("classif.naiveBayes")
)
benchmark(learners, task, resamplings = cv3)

##        task.id         learner.id mmce.test.mean
## 1 iris-example        classif.knn         0.0533
## 2 iris-example        classif.lda         0.0200
## 3 iris-example classif.naiveBayes         0.0400
```

CLASSIFICATION ANALYSIS: BENCHMARKING

```
tasks = list(iris.task, sonar.task, pid.task)
bm = benchmark(learners, tasks, resampling = cv3)
print(bm)

##                          task.id         learner.id mmce.test.mean
## 1                   iris-example        classif.knn         0.0467
## 2                   iris-example        classif.lda         0.0200
## 3                   iris-example classif.naiveBayes         0.0400
## 4 PimaIndiansDiabetes-example        classif.knn         0.2930
## 5 PimaIndiansDiabetes-example        classif.lda         0.2357
## 6 PimaIndiansDiabetes-example classif.naiveBayes         0.2500
## 7                  Sonar-example        classif.knn         0.1877
## 8                  Sonar-example        classif.lda         0.2788
## 9                  Sonar-example classif.naiveBayes         0.3028
```

CLASSIFICATION ANALYSIS: BENCHMARKING

```r
# aggregated data:
getBMRAggrPerformances(bm, as.df = TRUE)

##                         task.id          learner.id mmce.test.mean
## 1                  iris-example         classif.knn         0.0467
## 2                  iris-example         classif.lda         0.0200
## 3                  iris-example  classif.naiveBayes         0.0400
## 4 PimaIndiansDiabetes-example         classif.knn         0.2930
## 5 PimaIndiansDiabetes-example         classif.lda         0.2357
## 6 PimaIndiansDiabetes-example  classif.naiveBayes         0.2500
## 7                 Sonar-example         classif.knn         0.1877
## 8                 Sonar-example         classif.lda         0.2788
## 9                 Sonar-example  classif.naiveBayes         0.3028
```

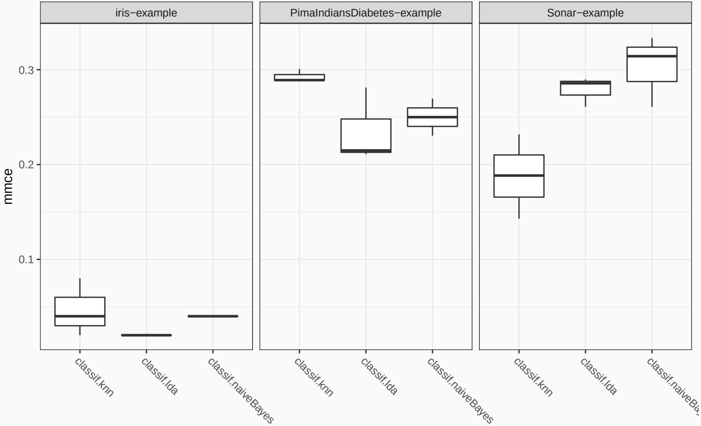
CLASSIFICATION ANALYSIS: BENCHMARKING

```
# complete data:
head(as.data.frame(bm), 10)

##                          task.id          learner.id iter  mmce
## 1                   iris-example         classif.knn    1 0.040
## 2                   iris-example         classif.knn    2 0.020
## 3                   iris-example         classif.knn    3 0.080
## 4                   iris-example         classif.lda    1 0.020
## 5                   iris-example         classif.lda    2 0.020
## 6                   iris-example         classif.lda    3 0.020
## 7                   iris-example classif.naiveBayes    1 0.040
## 8                   iris-example classif.naiveBayes    2 0.040
## 9                   iris-example classif.naiveBayes    3 0.040
## 10 PimaIndiansDiabetes-example         classif.knn    1 0.301
```

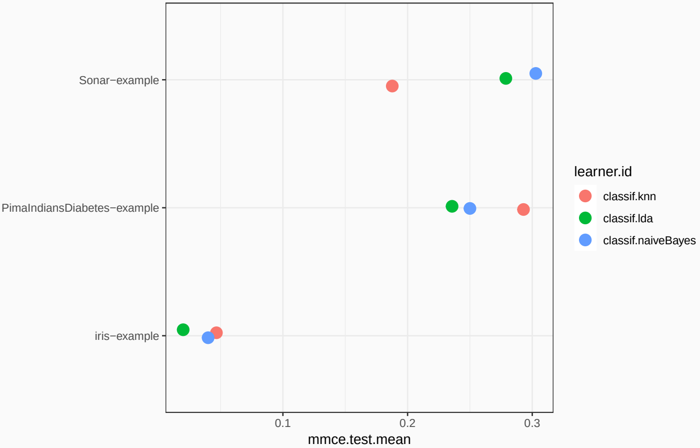
CLASSIFICATION ANALYSIS: BENCHMARKING

```
plotBMRBoxplots(bm, pretty.names = FALSE)
```



CLASSIFICATION ANALYSIS: BENCHMARKING

```r
plotBMRSummary(bm, pretty.names = FALSE)
```



TUNING

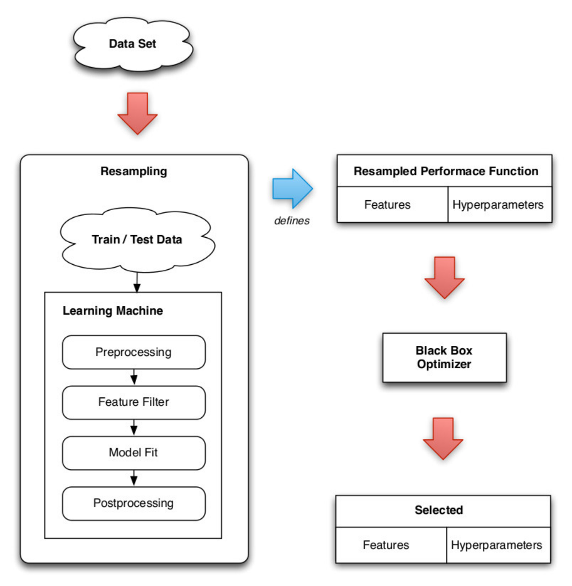
HYPERPARAMETER TUNING

- Many parameters or decisions for an ML algorithm are not decided by the (usually loss-minimizing) fitting procedure
- Our goal is to optimize these w.r.t. the estimated prediction error (often this implies an independent test set), or by cross-validation
- The same applies to preprocessing, feature construction and other model-relevant operations. In general we might be interested in optimizing a machine learning "pipeline"

HYPERPARAMETER TUNING

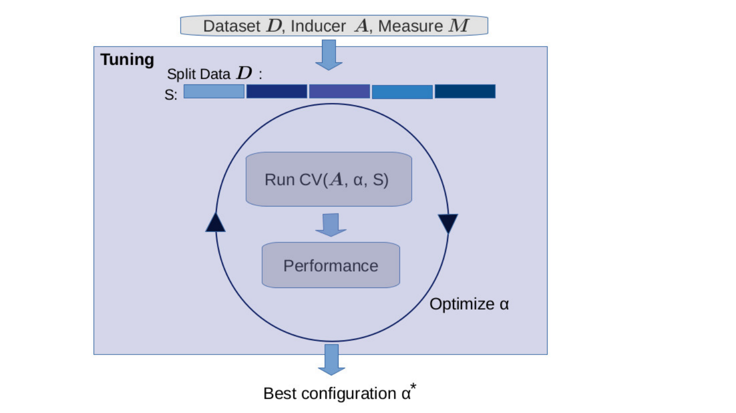
- Our learning method – or are there actually several?
- The performance measure – determined by the application
- Resampling procedure for measuring the performance
- Hyperparameters plus regions-of-interest

HYPERPARAMETER TUNING



HYPERPARAMETER TUNING

- Optimize hyperparameters for learner w.r.t. prediction error
- Tuner proposes configuration, eval by resampling, tuner receives performance, iterate

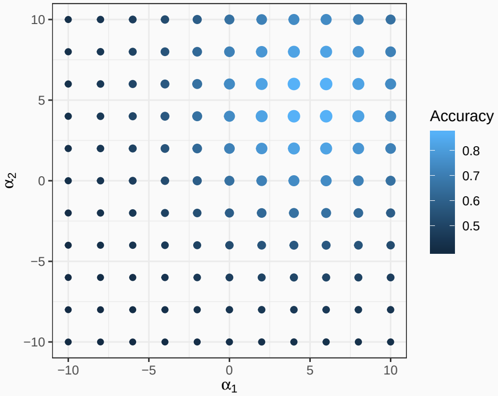


HYPERPARAMETER TUNING

- Our optimization problem is derivative-free, we can only ask for the quality of selected points (black-box problem)
- Our optimization problem is stochastic in principle. We want to optimize expected performance and use resampling
- Evaluation of our target function will probably take quite some time; Parallelization is often mandatory
- Categorical and dependent parameters complicate the problem

GRID SEARCH

- Try all hyperparameters combinations on a multi-dimensional discretized grid



GRID SEARCH

Advantages

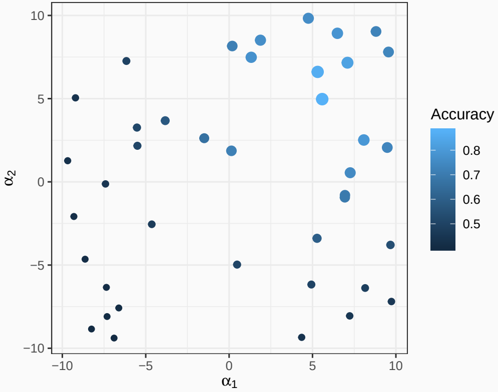
- Very easy to implement, therefore very popular
- All parameter types possible
- Parallelization is trivial

Disadvantages

- Combinatorial explosion, inefficient
- Searches large irrelevant areas
- Which values / discretization?

RANDOM SEARCH

- Small variation of grid search
- Instead of evaluating all hyper-parameter configurations on the grid, we uniformly sample from the region-of-interest



Advantages

- Very easy to implement, therefore very popular
- All parameter types possible
- Parallelization is trivial
- Anytime algorithm - we can always increase the budget when we are not satisfied
- Often better than grid search, as each individual parameter has been tried with m different values, when the search budget was m. Mitigates the problem of discretization.

Disadvantages

- As for grid search, many evaluations in areas with low likelihood for improvement

ADVANCED TUNING TECHNIQUES

- Simulated Annealing
- Genetic Algorithm / CMAES
- Iterated F-Racing
- Model-based Optimization / Bayesian Optimization

HYPERPARAMETERS IN MLR

```
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

Either set them in constructor, or change them later:

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

TUNING IN MLR

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

```
lrn = makeLearner("classif.ksvm",
  predict.type = "prob")

# this is actually a bad way to encode the SVM space, see a few slides later
# how to do this properly
par.set = makeParamSet(
  makeNumericParam("C", lower = 0.001, upper = 100),
  makeNumericParam("sigma", lower = 0.001, upper = 100)
)
```

TUNING IN MLR

Optimize the hyperparameter of learner

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

TUNING IN MLR

```
tr$x

## $C
## [1] 5.23
##
## $sigma
## [1] 3.05

tr$y

## auc.test.mean
##         0.633

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

##      C sigma auc.test.mean exec.time
## 1 73.8  86.0           0.5     0.047
## 2 71.7  91.9           0.5     0.040
## 3 60.4  44.6           0.5     0.046
```

PARAMETER TYPES

```r
makeNumericParam("x" ,lower = -1, upper = 1)
makeIntegerParam("x" ,lower = -1L, upper = 1L)
makeDiscreteParam("x" ,values = c("a", "b", "c"))
makeLogicalParam("x")
```

and vector-types exist for all param types

```r
makeNumericVectorParam("x" , len = 3L, lower = -1, upper = 1)

##              Type len Def  Constr Req Tunable Trafo
## 1 numericvector   3   - -1 to 1   -    TRUE     -
```

DEPENDENT PARAMS AND TRAFOS

```
lrn = makeLearner("classif.ksvm")
ps = makeParamSet(
  makeDiscreteParam("kernel", values = c("polydot", "rbfdot")),
  makeNumericParam("C", lower = -15, upper = 15,
    trafo = function(x) 2^x),
  makeNumericParam("sigma", lower = -15, upper = 15,
    trafo = function(x) 2^x,
   requires = quote(kernel == "rbfdot")),
  makeIntegerParam("degree", lower = 1, upper = 5,
   requires = quote(kernel == "polydot"))
)
```

PARALLEL MLR

PARALLELIZATION

- Many tasks in statistics are embarrassingly parallel (independence assumptions, resampling, ...)
- `R` is mostly single-threaded (matrix operations may be parallel, depending on your installation)
- Multiple backends for explicit parallelization available:
  - Multicore (packages parallel/multicore)
  - Socket and MPI cluster (packages parallel/snow/Rmpi)
  - HPC-Clusters (package `batchtools`): SLURM, Torque/PBS, SGE, LSF, Docker, SSH makeshift clusters, ...

PARALLELIZATION

- We use `parallelMap` in `mlr` an abstraction for all backends
- Initialize with `parallelStart()`
- Parallelize function call with `parallelMap()`/`parallelLapply()`/...
- Stop with `parallelStop()`

```
parallelStartSocket(4)
parallelMap(function(x) x^2, 1:10)
parallelStop()
```

PARALLELIZATION

- The first loop which is marked as parallel executable will be automatically parallelized
- Which loop is suited best for parallelization depends on the number of iterations
- Levels allow fine grained control over the parallelization
  - `mlr.resample`: Each resampling iteration (a train / test step) is a parallel job.
  - `mlr.benchmark`: Each experiment "run this learner on this data set" is a parallel job.
  - `mlr.tuneParams`: Each evaluation in hyperparameter space "resample with these parameter settings" is a parallel job. How many of these can be run independently in parallel depends on the tuning algorithm.
  - `mlr.selectFeatures`: Each evaluation in feature space "resample with this feature subset" is a parallel job.

PARALLELIZATION

```
lrns = list(makeLearner("classif.rpart"), makeLearner("classif.svm"))
rdesc = makeResampleDesc("Bootstrap", iters = 100)

parallelStartSocket(4)

## Starting parallelization in mode=socket with cpus=4.

bm = benchmark(learners = lrns, tasks = iris.task, resamplings = rdesc)

## Exporting objects to slaves for mode socket: .mlr.slave.options

## Mapping in parallel: mode = socket; cpus = 4; elements = 2.

parallelStop()

## Stopped parallelization. All cleaned up.
```

PARALLELIZATION

Parallelize the bootstrap instead:

```
parallelStartSocket(4, level = "mlr.resample")

## Starting parallelization in mode=socket with cpus=4.

bm = benchmark(learners = lrns, tasks = iris.task, resamplings = rdesc)

## Task: iris-example, Learner: classif.rpart

## Exporting objects to slaves for mode socket: .mlr.slave.options

## Resampling: OOB bootstrapping

## Measures:             mmce

## Mapping in parallel: mode = socket; cpus = 4; elements = 100.

##

## Aggregated Result: mmce.test.mean=0.059

##

## Task: iris-example, Learner: classif.svm

## Exporting objects to slaves for mode socket: .mlr.slave.options

## Resampling: OOB bootstrapping
```

EXERCISE 2