technische universität
dortmund

SFB 876 Providing Information by Resource-Constrained Data Analysis

mlrHyperopt: Effortless and collaborative hyperparameter optimization experiments

Jakob Richter

July 5, 2017

Faculty of Statistics, TU Dortmund University

Contents

1. Motivation for `caret`[1] Users
2. Motivation for `mlr`[2] Users
3. Website and API
4. Parameter Tuning
5. Lessons learned

[1] `https://topepo.github.io/caret`
[2] `https://mlr-org.github.io/mlr`

Motivation

caret

`caret` automatically performs a grid search for all learners.

```
library(caret)
system.time({m.c = train(iris[,1:4], iris[,5], method = "rf")})
##    user  system elapsed
##   4.533   0.016   4.552
system.time({m.r = randomForest(iris[,1:4], iris$Species)})
##    user  system elapsed
##   0.025   0.000   0.026
```

How to find out what is going on?

```
m.c$results
##   mtry  Accuracy     Kappa AccuracySD    KappaSD
## 1    2 0.9485003 0.9218204 0.02473556 0.03739386
## 2    3 0.9490167 0.9226138 0.02537238 0.03837125
## 3    4 0.9499133 0.9239744 0.02897600 0.04377608
```

caret

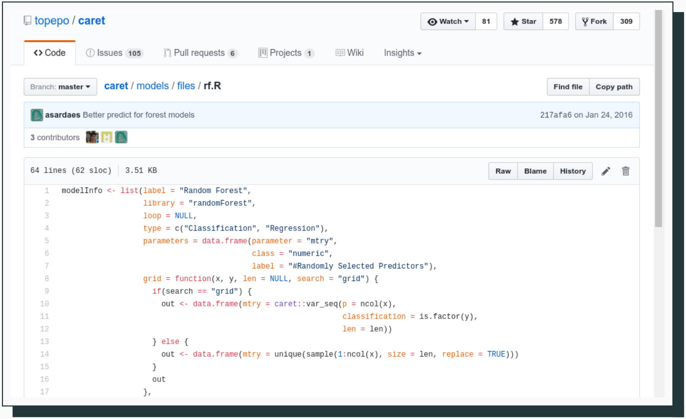
Can I find out in advance which parameters will be tuned?

`modelLookup("rf")` gives some information.

```
modelLookup("rf")
##   model parameter                          label forReg forClass probModel
## 1    rf      mtry #Randomly Selected Predictors   TRUE     TRUE      TRUE
```

caret

Can I find out in advance which parameters will be tuned?



`http://github.com/topepo/caret/blob/master/models/files` reveals all details.

caret: gbm

Extract from `models/files/gbm.R`:

```
out <- expand.grid(
  interaction.depth = seq(1, len), #<- parameter range depends on tuning budget
  n.trees = floor((1:len) * 50), #<- ..
  shrinkage = .1,
  n.minobsinnode = 10)
# ...
# Random Search
out <- data.frame(
  n.trees = floor(runif(len, min = 1, max = 5000)),
  interaction.depth = sample(1:10, replace = TRUE, size = len),
  shrinkage = runif(len, min = .001, max = .6),
  n.minobsinnode = sample(5:25, replace = TRUE, size = len) )
    out <- out[!duplicated(out),]
```

mlr

`mlr` provides parameter definitions for all learners.

```
library(mlr)
lrn = makeLearner("classif.randomForest")
filterParams(getParamSet(lrn), tunable = TRUE)
##                       Type  len   Def   Constr Req Tunable Trafo
## ntree              integer    -   500 1 to Inf   -    TRUE     -
## mtry               integer    -     - 1 to Inf   -    TRUE     -
## replace            logical    -  TRUE        -   -    TRUE     -
## classwt      numericvector <NA>     - 0 to Inf   -    TRUE     -
## cutoff       numericvector <NA>     -   0 to 1   -    TRUE     -
## sampsize     integervector <NA>     - 1 to Inf   -    TRUE     -
## nodesize           integer    -     1 1 to Inf   -    TRUE     -
## maxnodes           integer    -     - 1 to Inf   -    TRUE     -
## importance         logical    - FALSE        -   -    TRUE     -
## localImp           logical    - FALSE        -   -    TRUE     -
```

But `ParamSets` are unconstrained and include possibly unimportant parameters.

Necessary to define own `ParamSets` for tuning:

```
ps = makeParamSet(
  makeIntegerParam("mtry", lower = 1, upper = 4),
  makeIntegerParam("nodesize", lower = 1, upper = 10)
)
tuneParams(lrn, iris.task, cv10, measures = acc,
  par.set = ps, makeTuneControlGrid(resolution = 3L))
## Tune result:
## Op. pars: mtry=1; nodesize=6
## acc.test.mean=0.953
```

caret

Deviate from the defaults in `caret`:

```
grid = expand.grid(mtry = 2:4, nodesize = c(1,5,10))
m = caret::train(iris[,1:4], iris[,5],
  method = "rf", tuneGrid = grid)
## Error: The tuning parameter grid should have columns
mtry
```

It seems you have to write you own custom method[3].

[3] https://stackoverflow.com/questions/38625493/tuning-two-parameters-for-random-forest-in-caret-package

mlr vs. caret

In caret...

- + tuning is the default.
- + tuning with defaults is easy.
- – deviating from defaults is a hassle and needs expert knowledge.

In mlr...

- + train works like the default of the package.
- – tuning needs expert knowledge.
- + deviating from defaults is easy.

To solve this problem in `mlr` we want to share the expert knowledge with...

mlrHyperopt

mlrHyperopt

`mlrHyperopt` enables access to a web database of Parameter Configurations for many machine learning methods in R.

Why an online database?

- Defaults in packages will always be controversial.
- Knowledge changes over time but R packages have to maintain reproducibility.
- Defaults differ for different scenarios. (data set size *etc.*)

mlrHyperopt: ParConfigs

`mlrHyperopt` stores tuning parameters in `ParConfigs`:

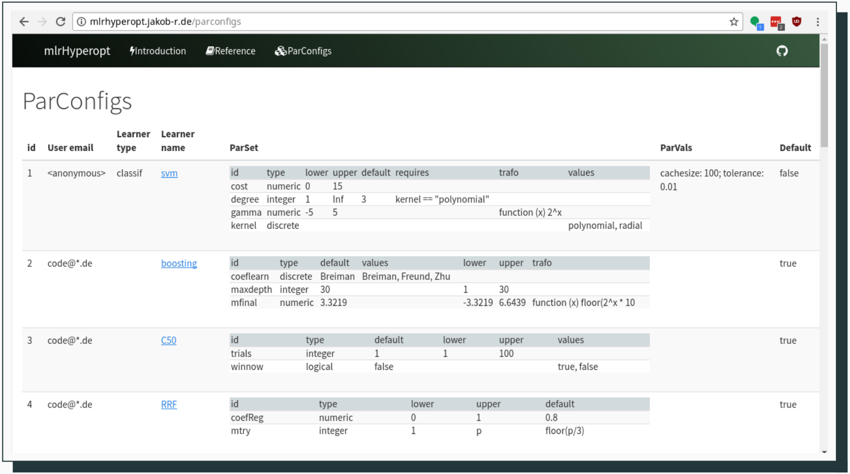
- *Parameter Set* of tunable parameters
- fixed *Parameter Values* to overwrite defaults
- associated learner and note

Features of the Parameter Set[4]:

- Parameter values can be: real-valued, integer, discrete, logical, ...
- Parameters can have:
  - transformations (to account non-uniform distribution of interesting regions)
  - requirements on other parameters (to represent hierarchical structures)
- Bounds and defaults can depend on the task size, number of features, *etc.*

[4] `https://github.com/berndbischl/ParamHelpers`

API Examples



mlrhyperopt.jakob-r.de/parconfigs

mlrHyperopt Introduction Reference ParConfigs

ParConfigs

<table>
<tr><th>id</th><th>User email</th><th>Learner type</th><th>Learner name</th><th>ParSet</th><th>ParVals</th><th>Default</th></tr>
<tr><td>1</td><td><anonymous></td><td>classif</td><td>svm</td><td>
<table>
<tr><th>id</th><th>type</th><th>lower</th><th>upper</th><th>default</th><th>requires</th><th>trafo</th><th>values</th></tr>
<tr><td>cost</td><td>numeric</td><td>0</td><td>15</td><td></td><td></td><td></td><td></td></tr>
<tr><td>degree</td><td>integer</td><td>1</td><td>Inf</td><td>3</td><td>kernel == "polynomial"</td><td></td><td></td></tr>
<tr><td>gamma</td><td>numeric</td><td>-5</td><td>5</td><td></td><td></td><td>function (x) 2^x</td><td></td></tr>
<tr><td>kernel</td><td>discrete</td><td></td><td></td><td></td><td></td><td></td><td>polynomial, radial</td></tr>
</table>
</td><td>cachesize: 100; tolerance: 0.01</td><td>false</td></tr>
<tr><td>2</td><td>code@*.de</td><td></td><td>boosting</td><td>
<table>
<tr><th>id</th><th>type</th><th>default</th><th>values</th><th>lower</th><th>upper</th><th>trafo</th></tr>
<tr><td>coeflearn</td><td>discrete</td><td>Breiman</td><td>Breiman, Freund, Zhu</td><td></td><td></td><td></td></tr>
<tr><td>maxdepth</td><td>integer</td><td>30</td><td></td><td>1</td><td>30</td><td></td></tr>
<tr><td>mfinal</td><td>numeric</td><td>3.3219</td><td></td><td>-3.3219</td><td>6.6439</td><td>function (x) floor(2^x * 10</td></tr>
</table>
</td><td></td><td>true</td></tr>
<tr><td>3</td><td>code@*.de</td><td></td><td>C50</td><td>
<table>
<tr><th>id</th><th>type</th><th>default</th><th>lower</th><th>upper</th><th>values</th></tr>
<tr><td>trials</td><td>integer</td><td>1</td><td>1</td><td>100</td><td></td></tr>
<tr><td>winnow</td><td>logical</td><td>false</td><td></td><td></td><td>true, false</td></tr>
</table>
</td><td></td><td>true</td></tr>
<tr><td>4</td><td>code@*.de</td><td></td><td>RRF</td><td>
<table>
<tr><th>id</th><th>type</th><th>lower</th><th>upper</th><th>default</th></tr>
<tr><td>coefReg</td><td>numeric</td><td>0</td><td>1</td><td>0.8</td></tr>
<tr><td>mtry</td><td>integer</td><td>1</td><td>p</td><td>floor(p/3)</td></tr>
</table>
</td><td></td><td>true</td></tr>
</table>

Overview of all `ParConfigs` uploaded to `http://mlrhyperopt.jakob-r.de/parconfigs`

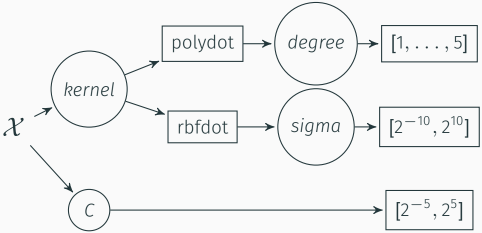
API: Download and Use ParConfigs

Tune the parameters for the `ranger` Random Forest with `mlr`[5].

```
library(mlrHyperopt)
lrn = makeLearner("classif.ranger")
(pc = downloadParConfigs(learner.class = getLearnerClass(lrn)))
## [[1]]
## Parameter Configuration
##   Parameter Values: num.threads=1, verbose=FALSE, respect.unordered.factors=T
##   Associated Learner: classif.ranger
##   Parameter Set:
##                    Type len            Def   Constr Req Tunable Trafo
## min.node.size integer   -                1 1 to 10    -    TRUE     -
## mtry          integer   - floor(sqrt(p))  1 to p    -    TRUE     -
ps = getParConfigParSet(pc[[1]])
ps = evaluateParamExpressions(ps, dict = getTaskDictionary(iris.task))
lrn = setHyperPars(lrn, par.vals = getParConfigParVals(pc[[1]]))
tuneParams(lrn, iris.task, resampling = cv10, par.set = ps,
  measures = acc, control = makeTuneControlRandom(maxit = 10))
## Tune result:
## Op. pars: min.node.size=3; mtry=1
## acc.test.mean=0.96
```

[5] http://mlr-org.github.io/mlr-tutorial/devel/html/tune/

API: Upload ParConfigs



Dependent search space for the tuning of a support vector machine.

```r
ps = makeParamSet(
  makeDiscreteParam("kernel", c("rbfdot", "polydot")),
  makeNumericParam("C", -5, 5, trafo = function(x) 2^x),
  makeNumericParam("sigma", lower = -10, upper = 10,
    trafo = function(x) 2^x, requires = quote(kernel == "rbfdot")),
  makeNumericParam("degree", lower = 1, upper = 5,
    requires = quote(kernel == "polydot"))
)
pc = makeParConfig(ps, learner.name = "ksvm")
uploadParConfig(pc)
## [1] "23"
```

Bonus: Use ParConfigs in caret

With the following `ParamHelpers` functions we can generate grids for `caret`

- `generateGridDesign`
- `generateRandomDesign`
- `generateDesign` (Latin Hypercube Sample)
- `generateDesignOfDefaults` (to be used in combination)

```
pc = downloadParConfigs(learner.name = "nnet")
grid = generateRandomDesign(n = 10L, par.set = pc[[1]]$par.set,
  trafo = TRUE)
tr = caret::train(iris[,1:4], iris[,5], method = "nnet",
  tuneGrid = grid, trace = FALSE)
tr$bestTune
##   size     decay
## 8   14 0.4467496
```

Tuning with `mlrHyperopt`

Tuning parameters with `mlrHyperopt`

A heuristic decides for tuning method:

Tuning Methods:

- **grid search:** 1 parameter, 2 mixed parameters
- **random search:** $>$ 2 mixed parameters
- **Bayesian optimization with `mlrMBO`[6]:** all parameters numeric

Default parameter sets from `mlrHyperopt` are used:

```
(h.res = hyperopt(task = iris.task, learner = "classif.ksvm"))
## Tune result:
## Op. pars: C=101; sigma=0.0432
## mmce.test.mean=0.0267
m = mlr::train(h.res$learner, iris.task)
```

[6] https://mlr-org.github.io/mlrMBO/

Benchmark

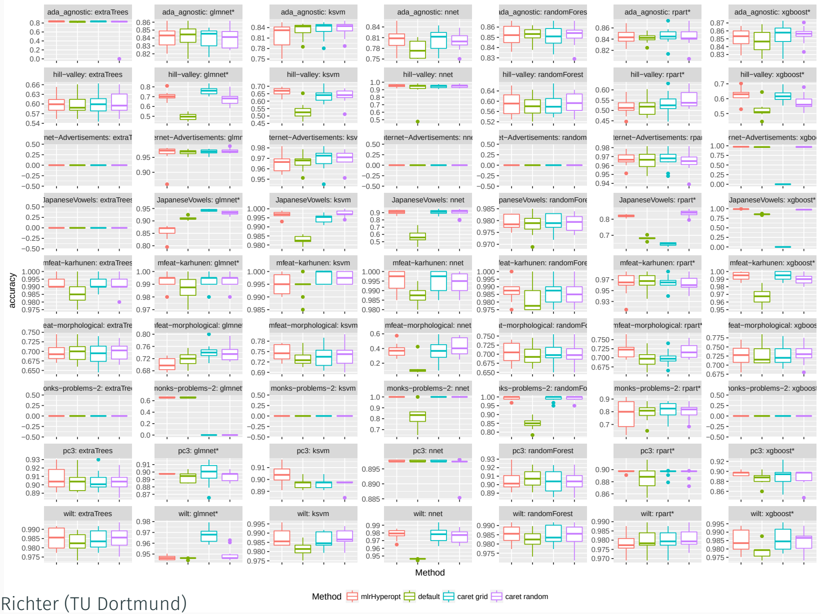
OpenML[7] Data Sets

OpenML_ID	Name	p	n
18	mfeat-morphological	6	2000
3493	monks-problems-2	6	601
3510	JapaneseVowels	14	9961
3883	mfeat-karhunen	64	2000
3896	ada_agnostic	48	4562
3903	pc3	37	1563
9914	wilt	5	4839
9970	hill-valley	100	1212
34536	Internet-Advertisements	1558	3279

Algorithms: `caret` with *grid* and *random* search and `mlrHyperopt`.
Each with a budget of 10 and 50 CV10-evaluations.

[7] `https://www.openml.org/`

All Results



ada_agnostic: extraTrees
ada_agnostic: glmnet*
ada_agnostic: ksvm
ada_agnostic: nnet
ada_agnostic: randomFores
ada_agnostic: rpart*
ada_agnostic: xgboost*
hill−valley: extraTrees
hill−valley: glmnet*
hill−valley: ksvm
hill−valley: nnet
hill−valley: randomForest
hill−valley: rpart*
hill−valley: xgboost*
net−Advertisements: extraT
ernet−Advertisements: glmn
ternet−Advertisements: ksv
nternet−Advertisements: nn
et−Advertisements: random
ternet−Advertisements: rpa
rnet−Advertisements: xgbo
JapaneseVowels: extraTrees
JapaneseVowels: glmnet*
JapaneseVowels: ksvm
JapaneseVowels: nnet
paneseVowels: randomFore
JapaneseVowels: rpart*
JapaneseVowels: xgboost*
mfeat−karhunen: extraTrees
mfeat−karhunen: glmnet*
mfeat−karhunen: ksvm
mfeat−karhunen: nnet
feat−karhunen: randomFore
mfeat−karhunen: rpart*
mfeat−karhunen: xgboost*
eat−morphological: extraTre
ifeat−morphological: glmne
mfeat−morphological: ksvm
mfeat−morphological: nnet
t−morphological: randomFo
mfeat−morphological: rpart*
feat−morphological: xgboos
onks−problems−2: extraTre
monks−problems−2: glmnet
monks−problems−2: ksvm
monks−problems−2: nnet
ks−problems−2: randomFo
monks−problems−2: rpart*
ionks−problems−2: xgboos
pc3: extraTrees
pc3: glmnet*
pc3: ksvm
pc3: nnet
pc3: randomForest
pc3: rpart*
pc3: xgboost*
wilt: extraTrees
wilt: glmnet*
wilt: ksvm
wilt: nnet
wilt: randomForest
wilt: rpart*
wilt: xgboost*
accuracy
Method
Method mlrHyperopt default caret grid caret random

Performance: Dominance

Performance with a budget of 10 10CV-Evaluations.

	caret grid	caret random	mlrHyperopt	default
caret grid	0.00	0.09	0.11	0.40
caret random	0.09	0.00	0.09	0.49
mlrHyperopt	0.14	0.12	0.00	0.47
default	0.09	0.02	0.04	0.00

The table gives the fractions of instances where $H_0 : acc_A \leq acc_B$ is rejected by the paired *Wilcoxon*-test to level $\alpha = 0.05$. A column, B rows.

i.e.: `mlrHyperopt` is significantly better than the default settings in 47% of the cases.

Performance: Dominance

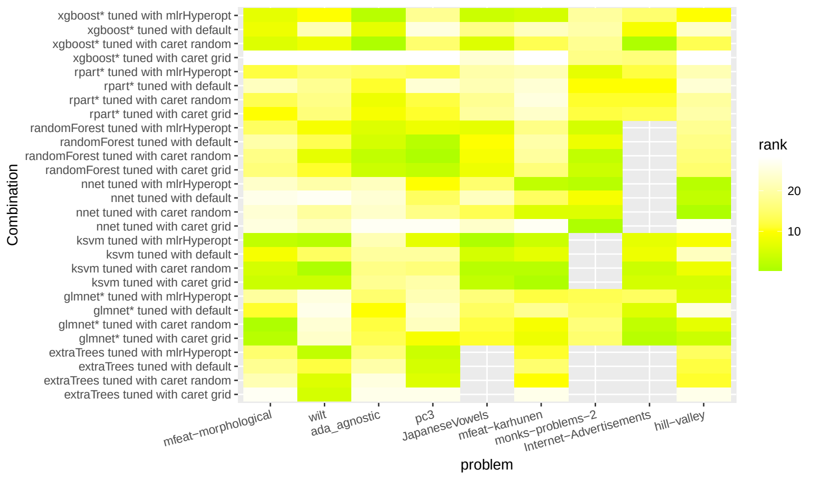
Performance with a budget of 50 10CV-Evaluations.

	caret grid	caret random	mlrHyperopt	default
caret grid	0.00	0.09	0.21	0.30
caret random	0.40	0.00	0.12	0.54
mlrHyperopt	0.40	0.16	0.00	0.53
default	0.33	0.02	0.05	0.00

The table gives the fractions of instances where $H_0 : acc_A \leq acc_B$ is rejected by the paired *Wilcoxon*-test to level $\alpha = 0.05$. *A* column, *B* rows.

i.e.: `mlrHyperopt` is significantly better than the default settings in 53% of the cases.

Which Learner Tuner Combination is a Good Choice?



Rankings of averaged performances of each combination on each dataset.

Lessons Learned

Lessons Learned

- Parameter Tuning is only beneficial on some data and for some methods.
- `caret`s grid search has performance problems on big data sets (ksvm, nnet).
- `caret`s grid search sub model trick is beneficial (glmnet).
- The benchmark indicates that *random search* is better than the grid search.

`mlrHyperopt`

Benefits

- Transparent and reproducible benchmarks in combination with *OpenML*:
  e.g. Tune ml method A on parameter space with id 123 on Open ML Task 456.

Outlook

- Implement voting system / advanced statistics

Find us on GitHub

- `github.com/jakob-r/mlrHyperopt`
- `github.com/mlr-org/mlr`