Machine Learning in R

The mlr package

Lars Kotthoff\textsuperscript{1}

University of Wyoming
larsko@uwyo.edu

St Andrews, 24 July 2018

\textsuperscript{1}with slides from Bernd Bischl
Outline

▷ Overview
▷ Basic Usage
▷ Wrappers
▷ Preprocessing with mlrCPO
▷ Feature Importance
▷ Parameter Optimization
Don’t reinvent the wheel.
Motivation

The good news

▷ hundreds of packages available in R
▷ often high-quality implementation of state-of-the-art methods

The bad news

▷ no common API (although very similar in many cases)
▷ not all learners work with all kinds of data and predictions
▷ what data, predictions, hyperparameters, etc are supported is not easily available

→ mlr provides a domain-specific language for ML in R
Overview

▷ https://github.com/mlr-org/mlr
▷ 8-10 main developers, >50 contributors, 5 GSoC projects
▷ unified interface for the basic building blocks: tasks, learners, hyperparameters...

![Diagram of machine learning process](image-url)
Basic Usage

head(iris)

## Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1 5.1 3.5 1.4 0.2 setosa
## 2 4.9 3.0 1.4 0.2 setosa
## 3 4.7 3.2 1.3 0.2 setosa
## 4 4.6 3.1 1.5 0.2 setosa
## 5 5.0 3.6 1.4 0.2 setosa
## 6 5.4 3.9 1.7 0.4 setosa

# create task
task = makeClassifTask(id = "iris", iris, target = "Species")

# create learner
learner = makeLearner("classif.randomForest")
# build model and evaluate

`holdout(learner, task)`

```r
## Resampling: holdout
## Measures:           mmce
## [Resample] iter 1:   0.0400000
##
## Aggregated Result:  mmce.test.mean=0.0400000
##
## Resample Result
## Task: iris
## Learner: classif.randomForest
## Aggr perf: mmce.test.mean=0.0400000
## Runtime: 0.0425465
```
Basic Usage

```r
# measure accuracy
holdout(learner, task, measures = acc)
```

```r
## Resampling: holdout
## Measures:
## [Resample] iter 1: 0.9800000
##
## Aggregated Result: acc.test.mean=0.9800000
##
## Resample Result
## Task: iris
## Learner: classif.randomForest
## Aggr perf: acc.test.mean=0.9800000
## Runtime: 0.0333493
```
Basic Usage

# 10 fold cross-validation

```r
crossval(learner, task, measures = acc)
```

```r
## Resampling: cross-validation
## Measures: acc
## [Resample] iter 1: 1.0000000
## [Resample] iter 2: 0.9333333
## [Resample] iter 3: 1.0000000
## [Resample] iter 4: 1.0000000
## [Resample] iter 5: 0.8000000
## [Resample] iter 6: 1.0000000
## [Resample] iter 7: 1.0000000
## [Resample] iter 8: 0.9333333
## [Resample] iter 9: 1.0000000
## [Resample] iter 10: 0.9333333

## Aggregated Result: acc.test.mean=0.9600000

## Resample Result
## Task: iris
## Learner: classif.randomForest
## Aggr perf: acc.test.mean=0.9600000
## Runtime: 0.530509
```
# more general -- resample description

```r
rdesc = makeResampleDesc("CV", iters = 8)
resample(learner, task, rdesc, measures = list(acc, mmce))
```

```r
## Resampling: cross-validation
## Measures: acc mmce
## [Resample] iter 1: 0.9473684 0.0526316
## [Resample] iter 2: 0.9473684 0.0526316
## [Resample] iter 3: 0.9473684 0.0526316
## [Resample] iter 4: 1.0000000 0.0000000
## [Resample] iter 5: 0.9473684 0.0526316
## [Resample] iter 6: 1.0000000 0.0000000
## [Resample] iter 7: 0.9444444 0.0555556
## [Resample] iter 8: 0.8947368 0.1052632

## Aggregated Result:
acc.test.mean=0.9535819, mmce.test.mean=0.0464181

## Resample Result
## Task: iris
## Learner: classif.randomForest
## Aggr perf: acc.test.mean=0.9535819, mmce.test.mean=0.0464181
## Runtime: 0.28359
```
listLearners(task)[1:5, c(1,3,4)]

<table>
<thead>
<tr>
<th></th>
<th>class</th>
<th>short.name</th>
<th>package</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>classif.adaboostm1</td>
<td>adaboostm1</td>
<td>RWeka</td>
</tr>
<tr>
<td>2</td>
<td>classif.boosting</td>
<td>adabag</td>
<td>adabag,rpart</td>
</tr>
<tr>
<td>3</td>
<td>classif.C50</td>
<td>C50</td>
<td>C50</td>
</tr>
<tr>
<td>4</td>
<td>classif.cforest</td>
<td>cforest</td>
<td>party</td>
</tr>
<tr>
<td>5</td>
<td>classif.ctree</td>
<td>ctree</td>
<td>party</td>
</tr>
</tbody>
</table>

listMeasures(task)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&quot;featperc&quot;</td>
</tr>
<tr>
<td>2</td>
<td>&quot;mmce&quot;</td>
</tr>
<tr>
<td>3</td>
<td>&quot;lsr&quot;</td>
</tr>
<tr>
<td>4</td>
<td>&quot;bac&quot;</td>
</tr>
<tr>
<td>5</td>
<td>&quot;qsr&quot;</td>
</tr>
<tr>
<td>6</td>
<td>&quot;timeboth&quot;</td>
</tr>
<tr>
<td>7</td>
<td>&quot;multiclass.aunp&quot;</td>
</tr>
<tr>
<td>8</td>
<td>&quot;timetrain&quot;</td>
</tr>
<tr>
<td>9</td>
<td>&quot;multiclass.au1p&quot;</td>
</tr>
<tr>
<td>10</td>
<td>&quot;ber&quot;</td>
</tr>
<tr>
<td>11</td>
<td>&quot;timepredict&quot;</td>
</tr>
<tr>
<td>12</td>
<td>&quot;multiclass.brier&quot;</td>
</tr>
<tr>
<td>13</td>
<td>&quot;ssr&quot;</td>
</tr>
<tr>
<td>14</td>
<td>&quot;acc&quot;</td>
</tr>
<tr>
<td>15</td>
<td>&quot;logloss&quot;</td>
</tr>
<tr>
<td>16</td>
<td>&quot;wkappa&quot;</td>
</tr>
<tr>
<td>17</td>
<td>&quot;multiclass.au1p&quot;</td>
</tr>
<tr>
<td>18</td>
<td>&quot;multiclass.au1u&quot;</td>
</tr>
<tr>
<td>19</td>
<td>&quot;kappa&quot;</td>
</tr>
</tbody>
</table>
Integrated Learners

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

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

Regression
- Linear, lasso and ridge
- Boosting
- Trees and forests
- Gaussian processes
- ...

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

12
### Learner Hyperparameters

```r
getParamSet(learner)
```

<table>
<thead>
<tr>
<th>##</th>
<th>Type</th>
<th>len</th>
<th>Def</th>
<th>Constr</th>
<th>Req</th>
<th>Tunable</th>
<th>Trafo</th>
</tr>
</thead>
<tbody>
<tr>
<td>ntree</td>
<td>integer</td>
<td></td>
<td>500</td>
<td>1 to Inf</td>
<td></td>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td>mtry</td>
<td>integer</td>
<td></td>
<td></td>
<td>1 to Inf</td>
<td></td>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td>replace</td>
<td>logical</td>
<td></td>
<td>TRUE</td>
<td>-</td>
<td></td>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td>classwt</td>
<td>numericvector</td>
<td>&lt;NA&gt;</td>
<td>-0 to Inf</td>
<td>-</td>
<td></td>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td>cutoff</td>
<td>numericvector</td>
<td>&lt;NA&gt;</td>
<td>0 to 1</td>
<td>-</td>
<td></td>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td>strata</td>
<td>untyped</td>
<td></td>
<td></td>
<td>-</td>
<td></td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>sampsize</td>
<td>integervector</td>
<td>&lt;NA&gt;</td>
<td>-1 to Inf</td>
<td>-</td>
<td></td>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td>nodesize</td>
<td>integer</td>
<td></td>
<td>1 1 to Inf</td>
<td>-</td>
<td></td>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td>maxnodes</td>
<td>integer</td>
<td></td>
<td>- 1 to Inf</td>
<td>-</td>
<td></td>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td>importance</td>
<td>logical</td>
<td></td>
<td>FALSE</td>
<td>-</td>
<td></td>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td>localImp</td>
<td>logical</td>
<td></td>
<td>FALSE</td>
<td>-</td>
<td></td>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td>proximity</td>
<td>logical</td>
<td></td>
<td>FALSE</td>
<td>-</td>
<td></td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>oob.prox</td>
<td>logical</td>
<td></td>
<td>-</td>
<td>-</td>
<td>Y</td>
<td>FALSE</td>
<td>-</td>
</tr>
<tr>
<td>norm.votes</td>
<td>logical</td>
<td></td>
<td>TRUE</td>
<td>-</td>
<td></td>
<td>FALSE</td>
<td>-</td>
</tr>
<tr>
<td>do.trace</td>
<td>logical</td>
<td></td>
<td>FALSE</td>
<td>-</td>
<td></td>
<td>FALSE</td>
<td>-</td>
</tr>
<tr>
<td>keep.forest</td>
<td>logical</td>
<td></td>
<td>TRUE</td>
<td>-</td>
<td></td>
<td>FALSE</td>
<td>-</td>
</tr>
<tr>
<td>keep.inbag</td>
<td>logical</td>
<td></td>
<td>FALSE</td>
<td>-</td>
<td></td>
<td>FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>
Learner Hyperparameters

```
lrn = makeLearner("classif.randomForest", ntree = 100, mtry = 10)
```

```
lrn = setHyperPars(lrn, ntree = 100, mtry = 10)
```
Wrappers

▷ extend the functionality of learners
▷ e.g. wrap a learner that cannot handle missing values with an impute wrapper
▷ hyperparameter spaces of learner and wrapper are joined
▷ can be nested
Wrappers

Available Wrappers

▷ Preprocessing: PCA, normalization (z-transformation)
▷ Parameter Tuning: grid, optim, random search, genetic algorithms, CMAES, iRace, MBO
▷ Filter: correlation- and entropy-based, $\chi^2$-test, mRMR, ...
▷ Feature Selection: (floating) sequential forward/backward, exhaustive search, genetic algorithms, ...
▷ Impute: dummy variables, imputations with mean, median, min, max, empirical distribution or other learners
▷ Bagging to fuse learners on bootstrapped samples
▷ Stacking to combine models in heterogenous ensembles
▷ Over- and Undersampling for unbalanced classification
Composable Preprocessing Operators for mlr – https://github.com/mlr-org/mlrCPO

- separate R package due to complexity, mlrCPO
- preprocessing operations (e.g. imputation or PCA) as R objects with their own hyperparameters

```r
operation = cpoScale()
print(operation)
## scale(center = TRUE, scale = TRUE)
```
Preprocessing with mlrCPO

▷ objects are handled using the “piping” operator %>>%
▷ composition:

```r
imputing.pca = cpoImputeMedian() %>>% cpoPca()
```

▷ application to data:

```r
task %>>% imputing.pca
```

▷ combination with a Learner to form a machine learning pipeline:

```r
pca.rf = imputing.pca %>>%
makeLearner("classif.randomForest")
```
# drop uninteresting columns
dropcol.cpo = cpoSelect(names = c("Cabin", 
"Ticket", "Name"), invert = TRUE)

# impute
impute.cpo = cpoImputeMedian(affect.type = "numeric") %>>% 
cpoImputeConstant("__miss__", affect.type = "factor")
mlrCPO Example: Titanic

```r
train.task = makeClassifTask("Titanic", train.data, 
    target = "Survived")

pp.task = train.task %>>% dropcol.cpo %>>% impute.cpo
print(pp.task)
```

```r
## Supervised task: Titanic
## Type: classif
## Target: Survived
## Observations: 872
## Features:
##   numerics   factors   ordered   functionals
##        4         3        0         0
## Missings: FALSE
## Has weights: FALSE
## Has blocking: FALSE
## Has coordinates: FALSE
## Classes: 2
##   0  1
##  541 331
## Positive class: 0
```
Combination with Learners

▷ attach one or more CPOs to a learner to build machine learning pipelines
▷ automatically handles preprocessing of test data

```r
learner = dropcol.cpo %>>% impute.cpo %>>%
    makeLearner("classif.randomForest", predict.type = "prob")

# train using the task that was not preprocessed
pp.mod = train(learner, train.task)
```
mlrCPO Summary

▷ `listCP0()` to show available CPOs
▷ currently 69 CPOs, and growing: imputation, feature type conversion, target value transformation, over/undersampling, ...
▷ CPO “multiplexer” enables combination of different distinct preprocessing operations selectable through hyperparameter
▷ custom CPOs can be created using `makeCP0()`
model = train(makeLearner("classif.randomForest"), iris.task)
getFeatureImportance(model)

## FeatureImportance:
## Task: iris-example
##
## Learner: classif.randomForest
## Measure: NA
## Contrast: NA
## Aggregation: function (x) x
## Replace: NA
## Number of Monte-Carlo iterations: NA
## Local: FALSE
##    Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1        9.857828     2.282677    42.51918    44.58139
Feature Importance

```r
model = train(makeLearner("classif.xgboost"), iris.task)
getFeatureImportance(model)
```

```r
## FeatureImportance:
## Task: iris-example
##
## Learner: classif.xgboost
## Measure: NA
## Contrast: NA
## Aggregation: function (x) x
## Replace: NA
## Number of Monte-Carlo iterations: NA
## Local: FALSE
##
## Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1 0 0 0.4971064 0.5028936
```
Partial Dependence Plots

Partial Predictions

▷ estimate how the learned prediction function is affected by features
▷ marginalized version of the predictions for one or more features

```
lrn = makeLearner("classif.randomForest", predict.type = "prob")
fit = train(lrn, iris.task)
pd = generatePartialDependenceData(fit, iris.task, "Petal.Width")

plotPartialDependence(pd)
```
Partial Dependence Plots

Class
- setosa
- versicolor
- virginica

Probability

Petal Width

0.1
0.2
0.3
0.4
0.5
0.6
0.0 0.5 1.0 1.5 2.0 2.5

26
Partial Dependence Plots

```
pd = generatePartialDependenceData(fit, iris.task, c("Petal.Width", "Petal.Length"), interaction = TRUE)
plotPartialDependence(pd, facet = "Petal.Length")
```
Hyperparameter Tuning

- often important to get good performance
- humans are really bad at it
- mlr supports many different methods for hyperparameter optimization

```
ps = makeParamSet(makeIntegerParam("ntree", lower = 10, upper = 500))
tune.ctrl = makeTuneControlRandom(maxit = 3)
rdesc = makeResampleDesc("CV", iters = 10)
tuneParams(makeLearner("classif.randomForest"), task = iris.task, par.set = ps, resampling = rdesc, control = tune.ctrl)
```

```
## [Tune] Started tuning learner classif.randomForest for parameter set:
##       Type len Def Constr Req Tunable Trafo
##  ntree integer - - 10 to 500 - TRUE -
## With control class: TuneControlRandom
## Imputation value: 1
## [Tune-x] 1: ntree=287
## [Tune-y] 1: mmce.test.mean=0.0466667; time: 0.0 min
## [Tune-x] 2: ntree=315
## [Tune-y] 2: mmce.test.mean=0.0400000; time: 0.0 min
## [Tune-x] 3: ntree=181
## [Tune-y] 3: mmce.test.mean=0.0400000; time: 0.0 min
## [Tune] Result: ntree=315 : mmce.test.mean=0.0400000

## Tune result:
## Op. pars: ntree=315
## mmce.test.mean=0.0400000
```
Automatic Hyperparameter Tuning

▷ combine learner with tuning wrapper (and nested resampling)

```r
ps = makeParamSet(makeIntegerParam("ntree", lower = 10, upper = 500))
tune.ctrl = makeTuneControlRandom(maxit = 3)
learner = makeTuneWrapper(makeLearner("classif.randomForest"), par.set = ps,
resampling = makeResampleDesc("CV", iters = 10), control = tune.ctrl)
resample(learner, iris.task, makeResampleDesc("Holdout"))
```

```r
## Resampling: holdout
## Measures: mmce
## [Tune] Started tuning learner classif.randomForest for parameter set:
##       Type len Def Constr Req Tunable Trafo
## ntree integer  -  -  10 to 500  -  TRUE  -
## [Tune] Result: ntree=125 : mmce.test.mean=0.0300000
## [Resample] iter 1: 0.0400000
## Aggregated Result: mmce.test.mean=0.0400000
##
## Resample Result
## Task: iris-example
## Learner: classif.randomForest.tuned
## Aggr perf: mmce.test.mean=0.0400000
## Runtime: 0.595004
```
Tuning of Joint Hyperparameter Spaces

\[
\text{lrn} = \text{cpoFilterFeatures}(\text{abs} = 2L) \gg>\gg \text{makeLearner}("\text{classif.randomForest}"
)\]

\[
\text{ps} = \text{makeParamSet(}
  \text{makeDiscreteParam("filterFeatures.method",}
  \text{values} = \text{c("anova.test", "chi.squared"))},
  \text{makeIntegerParam("ntree", lower = 10, upper = 500)}
)\]

\[
\text{ctrl} = \text{makeTuneControlRandom(maxit = 3L)}
\]

\[
\text{tr} = \text{tuneParams(lrn, iris.task, cv3, par.set = ps, control = ctrl)}
\]

---

## [Tune] Started tuning learner classif.randomForest.filterFeatures for parameter set:

<table>
<thead>
<tr>
<th>Type</th>
<th>len</th>
<th>Def</th>
<th>Constr</th>
<th>Req</th>
<th>Tunable</th>
</tr>
</thead>
<tbody>
<tr>
<td>filterFeatures.method</td>
<td>discrete</td>
<td>-</td>
<td>anova.test,chi.squared</td>
<td>-</td>
<td>TRUE</td>
</tr>
<tr>
<td>ntree</td>
<td>integer</td>
<td>-</td>
<td>10 to 500</td>
<td>-</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

## Trafo

## With control class: TuneControlRandom

## Imputation value: 1

## [Tune-x] 1: filterFeatures.method=chi.squared; ntree=343
## [Tune-y] 1: mmce.test.mean=0.0533333; time: 0.0 min

## [Tune-x] 2: filterFeatures.method=chi.squared; ntree=23
## [Tune-y] 2: mmce.test.mean=0.0533333; time: 0.0 min

## [Tune-x] 3: filterFeatures.method=chi.squared; ntree=397
## [Tune-y] 3: mmce.test.mean=0.0533333; time: 0.0 min

## [Tune] Result: filterFeatures.method=chi.squared; ntree=343 : mmce.test.mean=0.0533333
Available Hyperparameter Tuning Methods

▷ grid search
▷ random search
▷ population-based approaches (racing, genetic algorithms, simulated annealing)
▷ Bayesian model-based optimization (MBO)
▷ custom design
Grid Search Example
Random Search Example

![Random Search Example Diagram]
Simulated Annealing Example
Model-Based Search Example
There is more...

- benchmark experiments
- visualization of learning rates, ROC, ...
- parallelization
- cost-sensitive learning
- handling of imbalanced classes
- multi-criteria optimization
- ...

36
Resources

▷ project page: https://github.com/mlr-org/mlr
▷ tutorial: https://mlr-org.github.io/mlr/
▷ mlrCPO: https://github.com/mlr-org/mlrCPO
▷ mlrMBO: https://github.com/mlr-org/mlrMBO
I’m hiring!

Several funded graduate positions available.