# Machine Learning in R

## The mlr package

Lars Kotthoff[1]

University of Wyoming
larsko@uwyo.edu

St Andrews, 24 July 2018

[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

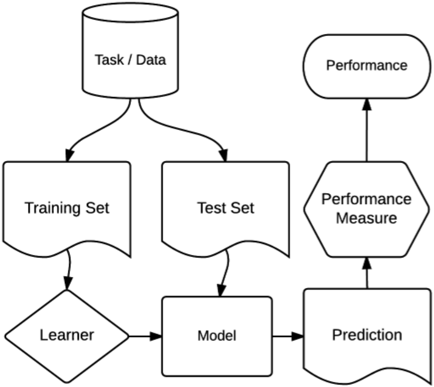
## The bad news

- 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...



# Basic Usage

```r
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")
```

# Basic Usage

```
# build model and evaluate
holdout(learner, task)

## 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)

## Resampling: holdout
## Measures:             acc
## [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
crossval(learner, task, measures = acc)

## 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
```

# Basic Usage

```
# more general -- resample description
rdesc = makeResampleDesc("CV", iters = 8)
resample(learner, task, rdesc, measures = list(acc, mmce))

## 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
```

# Finding Your Way Around

```
listLearners(task)[1:5, c(1,3,4)]

##                  class short.name      package
## 1 classif.adaboostm1 adaboostm1        RWeka
## 2   classif.boosting     adabag adabag,rpart
## 3        classif.C50        C50          C50
## 4    classif.cforest    cforest        party
## 5      classif.ctree      ctree        party

listMeasures(task)

##  [1] "featperc"          "mmce"              "lsr"
##  [4] "bac"               "qsr"               "timeboth"
##  [7] "multiclass.aunp"  "timetrain"        "multiclass.aunu"
## [10] "ber"              "timepredict"     "multiclass.brier"
## [13] "ssr"               "acc"               "logloss"
## [16] "wkappa"           "multiclass.au1p"  "multiclass.au1u"
## [19] "kappa"
```

# 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
- ...

# Learner Hyperparameters

```
getParamSet(learner)

##                     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     -
## strata           untyped    -     -        -   -   FALSE     -
## 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     -
## proximity        logical    - FALSE        -   -   FALSE     -
## oob.prox         logical    -     -        -   Y   FALSE     -
## norm.votes       logical    -  TRUE        -   -   FALSE     -
## do.trace         logical    - FALSE        -   -   FALSE     -
## keep.forest      logical    -  TRUE        -   -   FALSE     -
## keep.inbag       logical    - FALSE        -   -   FALSE     -
```

# 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, $\mathcal{X}^2$-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 bootstraped samples
- Stacking to combine models in heterogenous ensembles
- Over- and Undersampling for unbalanced classification

# Preprocessing with mlrCPO

- 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

```
operation = cpoScale()
print(operation)

## scale(center = TRUE, scale = TRUE)
```

# Preprocessing with mlrCPO

- objects are handled using the "piping" operator %>>%
- composition:

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

- application to data:

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

- combination with a `Learner` to form a machine learning pipeline:

```
pca.rf = imputing.pca %>>%
  makeLearner("classif.randomForest")
```

# mlrCPO Example: Titanic

```r
# 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

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

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

## 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

```
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

- `listCPO()` 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 `makeCPO()`

# Feature Importance

```r
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)

## 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
```

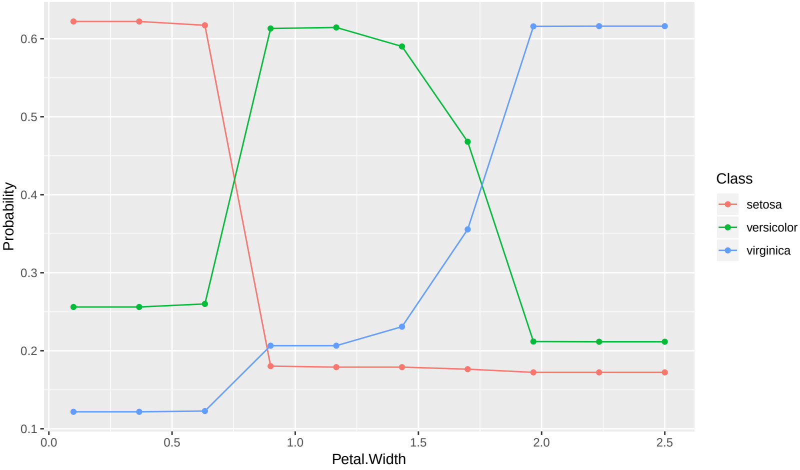
# Partial Dependence Plots

## Partial Predictions

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

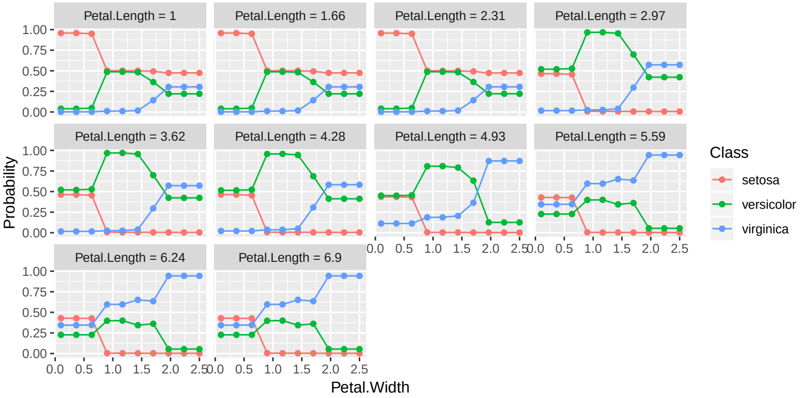
```
lrn = makeLearner("classif.randomForest", predict.type = "prob")
fit = train(lrn, iris.task)
pd = generatePartialDependenceData(fit, iris.task,
  "Petal.Width")

plotPartialDependence(pd)
```

# Partial Dependence Plots



# Partial Dependence Plots

```r
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)

```
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"))
```

```
## 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     -
## With control class: TuneControlRandom
## Imputation value: 1
## [Tune-x] 1: ntree=351
## [Tune-y] 1: mmce.test.mean=0.0300000; time: 0.0 min
## [Tune-x] 2: ntree=125
## [Tune-y] 2: mmce.test.mean=0.0300000; time: 0.0 min
## [Tune-x] 3: ntree=369
## [Tune-y] 3: mmce.test.mean=0.0300000; time: 0.0 min
## [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

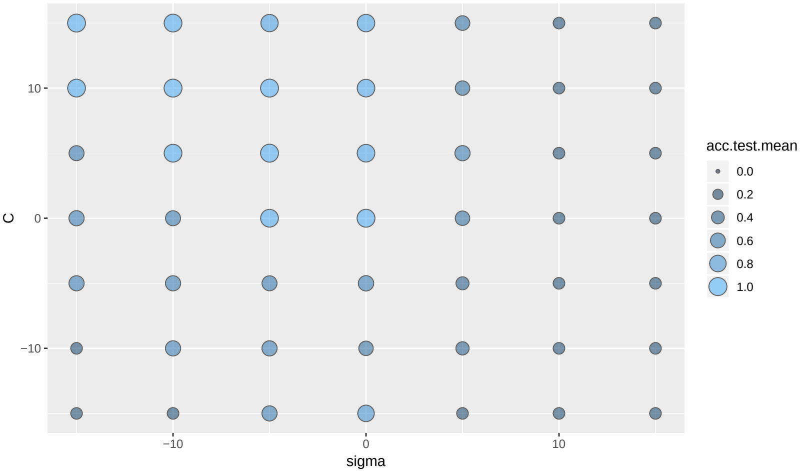
```r
lrn = cpoFilterFeatures(abs = 2L) %>>% makeLearner("classif.randomForest")

ps = makeParamSet(
  makeDiscreteParam("filterFeatures.method",
    values = c("anova.test", "chi.squared")),
  makeIntegerParam("ntree", lower = 10, upper = 500)
)
ctrl = makeTuneControlRandom(maxit = 3L)
tr = tuneParams(lrn, iris.task, cv3, par.set = ps, control = ctrl)
```

```
## [Tune] Started tuning learner classif.randomForest.filterFeatures for parameter
set:
##                          Type len Def                 Constr Req Tunable
## filterFeatures.method discrete   -   - anova.test,chi.squared   -    TRUE
## ntree                  integer   -   -             10 to 500   -    TRUE
##                        Trafo
## filterFeatures.method     -
## ntree                     -
## 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
```

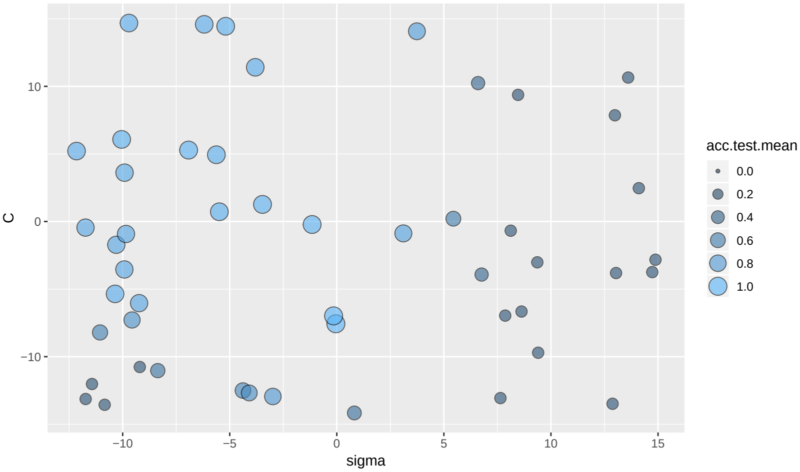
# Available Hyperparameter Tuning Methods

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

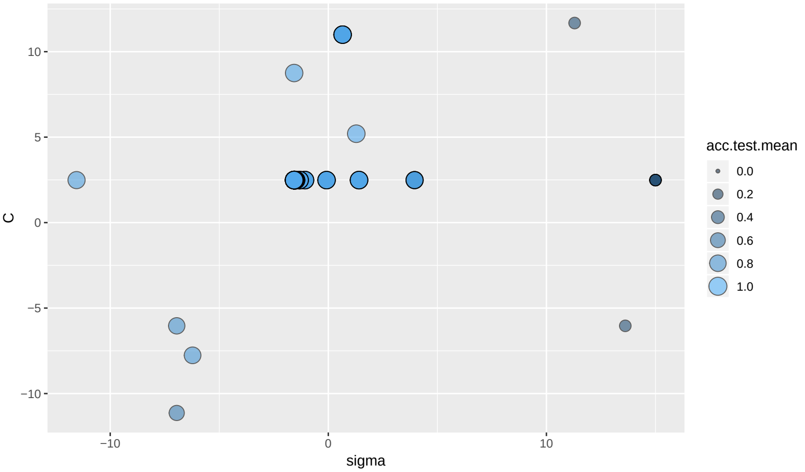
# Grid Search Example



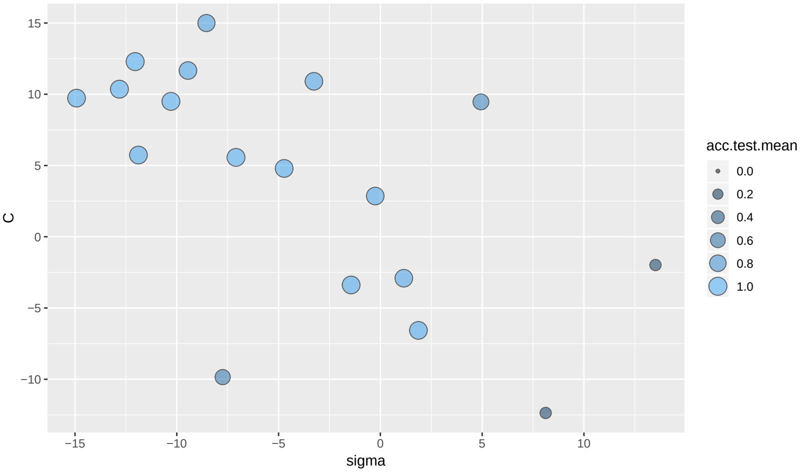
# Random Search Example



# 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
- ...

# Resources

- project page: `https://github.com/mlr-org/mlr`
- tutorial: `https://mlr-org.github.io/mlr/`
- cheat sheet: `https://github.com/mlr-org/mlr/blob/master/vignettes/tutorial/cheatsheet/MlrCheatsheet.pdf`
- mlrCPO: `https://github.com/mlr-org/mlrCPO`
- mlrMBO: `https://github.com/mlr-org/mlrMBO`

# I'm hiring!

Several funded graduate positions available.