

# Modern Machine Learning in R

## mlr3

Department of Statistics – LMU Munich

September 24, 2019



# Intro

# SO YOU WANT TO DO ML IN R

```
library("mlr3")
```

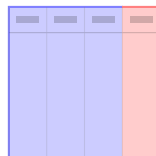
Ingredients:

- Data
- Learning Algorithms
- Performance Evaluation
- Performance Comparison

# Data

# DATA

- Tabular data
  - Features
  - Target / outcome to predict
    - discrete for classification
    - continuous for regression
- ⇒ data determines the machine learning “Task”



```
print(iris) # included in R
```

```
#>   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
#> ...
```

Task ID

data

target name

```
task = TaskClassif$new("iris", iris, "Species")
```

# DATA

```
task = TaskClassif$new("iris", iris, "Species")
```

```
print(task)

# <TaskClassif:iris> (150 x 5)
# * Target: Species
# * Properties: multiclass
# * Features (4):
#   - dbl (4): Petal.Length, Petal.Width, Sepal.Length, Sepal.Width
```

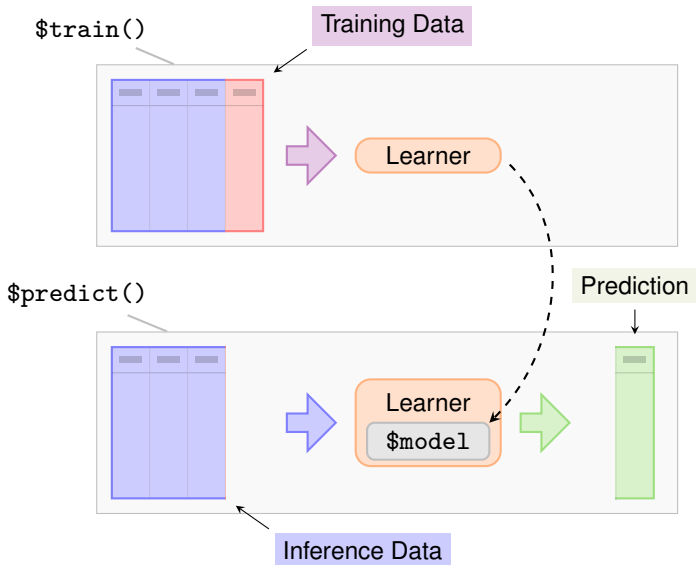
```
task$ncol
task$nrow
task$feature_names
task$target_names
```

```
task$head(n = )
task$truth(row_ids = )
task$data(rows = ,
           cols = )
```

```
task$select(cols = )
task$filter(rows = )
task$cbind(data = )
task$rbind(data = )
```

# Learning Algorithms

# LEARNING ALGORITHMS





# LEARNING ALGORITHMS

- Get a Learner provided by `mlr`

```
learner = lrn("classif.rpart")
```

- Train the Learner

```
learner$train(task)
```

- The `$model` is the `rpart` model: a decision tree

```
print(learner$model)
```

```
#> n= 150
#>
#> node), split, n, loss, yval, (yprob)
#>      * denotes terminal node
#>
#> 1) root 150 100 setosa (0.333 0.333 0.333)
#>   2) Petal.Length< 2.5 50   0 setosa (1.000 0.000 0.000) *
#>   3) Petal.Length>=2.5 100  50 versicolor (0.000 0.500 0.500)
#>     6) Petal.Width< 1.8 54   5 versicolor (0.000 0.907 0.093) *
#>     7) Petal.Width>=1.8 46   1 virginica (0.000 0.022 0.978) *
```

# HYPERPARAMETERS

- Learners have *hyperparameters*

```
learner$param_set

#> ParamSet:
#>      id      class lower upper levels default value
#> 1:  minsplit ParamInt     1   Inf        20
#> 2:      cp ParamDbl     0     1         0.01
#> 3: maxcompete ParamInt     0   Inf         4
#> 4: maxsurrogate ParamInt     0   Inf         5
#> 5:  maxdepth ParamInt     1    30        30
#> 6:      xval ParamInt     0   Inf        10      0
```

- Changing them changes the Learner behavior

```
learner$param_set$values = list(maxdepth = 1)

learner$train(task)
```

# HYPERPARAMETERS

- This gives a smaller decision tree

```
print(learner$model)

#> n= 150
#>
#> node), split, n, loss, yval, (yprob)
#>      * denotes terminal node
#>
#> 1) root 150 100 setosa (0.33 0.33 0.33)
#>   2) Petal.Length< 2.5 50   0 setosa (1.00 0.00 0.00) *
#>   3) Petal.Length>=2.5 100  50 versicolor (0.00 0.50 0.50) *
```

- Instead of assigning `$values` a `list()`, we can change individual parameters

```
learner$param_set$values$maxdepth = 10
```

# PREDICTION

- Lets make a prediction

```
new_data  
  
#   Sepal.Length Sepal.Width Petal.Length Petal.Width  
# 1           4           3           2           1  
# 2           2           2           3           2
```

- Call `$predict_newdata()` with the data and the old **Task**

```
prediction = learner$predict_newdata(task, new_data)
```

- We get a Prediction object:

```
prediction  
  
#> <PredictionClassif> for 2 observations:  
#>   row_id truth response  
#>    151  <NA>   setosa  
#>    152  <NA> versicolor
```

# PREDICTION

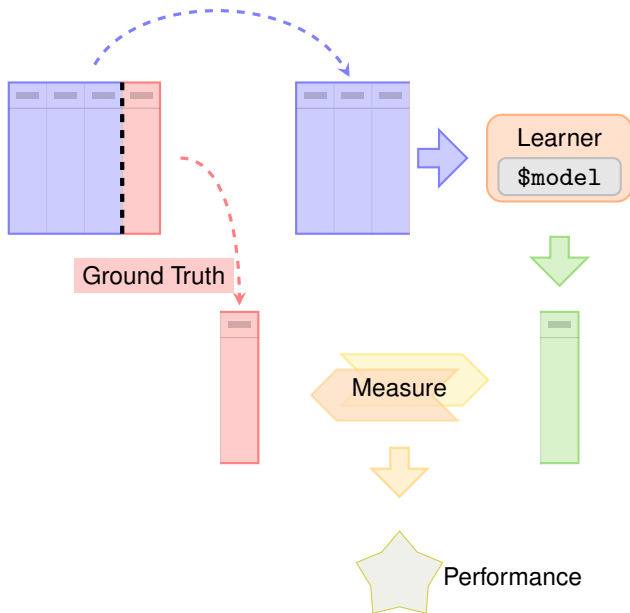
- We can make the Learner predict *probabilities* when we set `predict_type`:

```
learner$predict_type = "prob"
learner$predict_newdata(task, new_data)

# <PredictionClassif> for 2 observations:
#   row_id truth   response prob.setosa prob.versicolor
#     151  <NA>    setosa           1           0.0
#     152  <NA> versicolor          0           0.5
#   prob.virginica
#             0.0
#             0.5
```

# Performance

# PERFORMANCE EVALUATION



# PERFORMANCE EVALUATION

- Prediction 'Task' with known data

```
known_truth_task$data()

#   Species Petal.Length Petal.Width Sepal.Length Sepal.Width
# 1:  setosa           2           1           4           3
# 2:  setosa           3           2           2           2
```

- Predict again

```
pred = learner$predict(known_truth_task)
pred

#> <PredictionClassif> for 2 observations:
#> row_id truth  response
#>      1 setosa   setosa
#>      2 setosa virginica
```

- Score the prediction

```
pred$score(msr("classif.ce"))

#> classif.ce
#>          0.5
```



# PERFORMANCE EVALUATION

- Confusion Matrix

```
pred
```

```
#> <PredictionClassif> for 2 observations:
```

```
#>  row_id  truth  response
```

```
#>      1 setosa   setosa
```

```
#>      2 setosa virginica
```

```
pred$confusion
```

```
#>                truth
```

```
#> response      setosa versicolor virginica
```

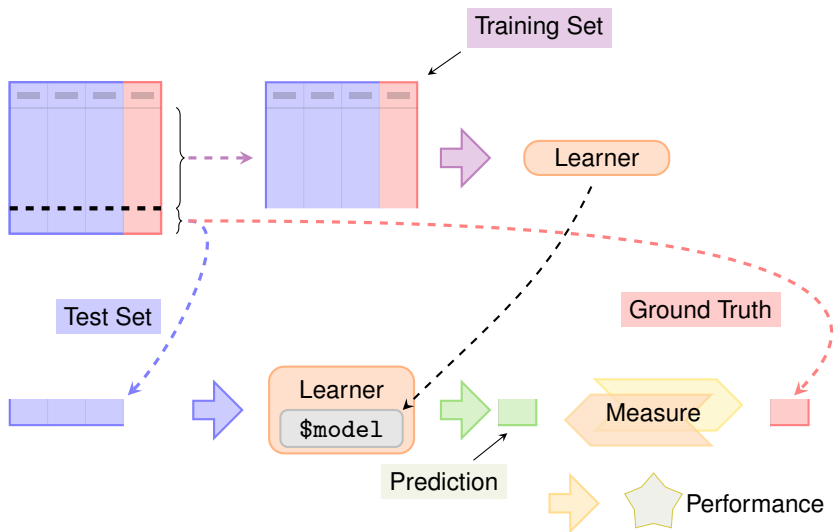
```
#>   setosa          1           0           0
```

```
#> versicolor       0           0           0
```

```
#> virginica        1           0           0
```

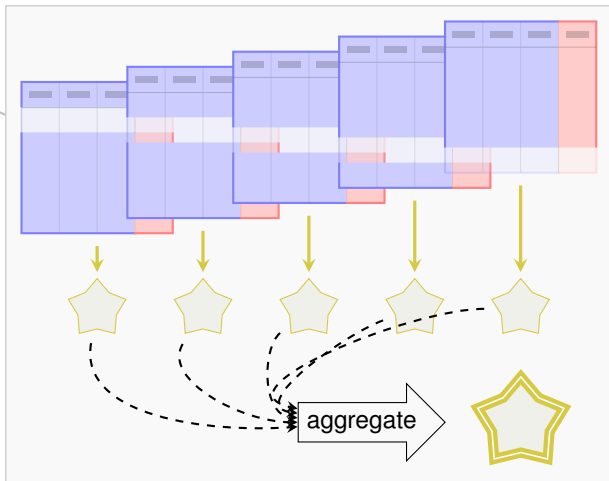
# Resampling

# RESAMPLING



# RESAMPLING

`resample()`



# RESAMPLING

- Resample description: How to split the data

```
cv5 = rsmp("cv", folds = 5)
```

- Use the `resample()` function for resampling:

```
result = resample(task, learner, cv5)
```

- The resampling result:

```
print(result)

#> <ResampleResult> of 5 iterations
#> * Task: iris
#> * Learner: classif.rpart
#> * Performance: 0.087 [classif.ce]
#> * Warnings: 0 in 0 iterations
#> * Errors: 0 in 0 iterations
```

# RESAMPLING

- Get performance:

```
result$aggregate(msr("classif.ce"))  
#> classif.ce  
#>      0.087
```

- Get predictions

```
result$prediction()  
#> <PredictionClassif> for 150 observations:  
#>      row_id      truth  response  
#>          1    setosa    setosa  
#>          8    setosa    setosa  
#>         15    setosa    setosa  
#> ---  
#>        145 virginica virginica  
#>        148 virginica virginica  
#>        150 virginica virginica
```

# RESAMPLING

- Predictions of individual folds

```
result$predictions()[[1]]  
  
#> <PredictionClassif> for 30 observations:  
#>      row_id      truth  response  
#>          1    setosa    setosa  
#>          8    setosa    setosa  
#>         15    setosa    setosa  
#> ---  
#>        133 virginica  virginica  
#>        134 virginica  versicolor  
#>        138 virginica  virginica
```

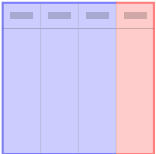



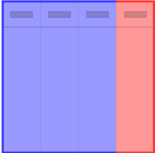



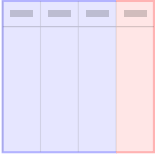



- Score of individual folds

```
result$score()[, .(iteration, classif.ce)]  
  
#>      iteration classif.ce  
#> 1:          1      0.067  
#> 2:          2      0.133  
#> 3:          3      0.133  
#> 4:          4      0.067  
#> 5:          5      0.033
```

# Benchmark



# PERFORMANCE COMPARISON

	Learner 1	Learner 2	Learner 3
			
			
			

# PERFORMANCE COMPARISON

- Multiple Learners, multiple Tasks:

```
library("mlr3learners")  
learners = list(lrn("classif.rpart"), lrn("classif.kknn"))  
tasks = list(tsk("iris"), tsk("sonar"), tsk("wine"))
```

- Set up the *design* and execute benchmark:

```
design = benchmark_grid(tasks, learners, cv5)  
bmr = benchmark(design)
```

- The benchmark result shows that kknn outperforms rpart:

```
bmr$aggregate()[, .(task_id, learner_id, classif.ce)]  
  
#>   task_id learner_id classif.ce  
#> 1:   iris classif.rpart    0.073  
#> 2:   iris classif.kknn    0.047  
#> 3:  sonar classif.rpart    0.337  
#> 4:  sonar classif.kknn    0.168  
#> 5:   wine classif.rpart    0.140  
#> 6:   wine classif.kknn    0.039
```

# Short Forms

# SHORT FORMS

- Ordinary constructors: `LearnerClassifRpart$new()`

⇒ Short form constructors

- They access Dictionary of objects:

Object	Dictionary	Short Form
Task	<code>mlr_tasks</code>	<code>tsk()</code>
Learner	<code>mlr_learners</code>	<code>lrn()</code>
Measure	<code>mlr_measures</code>	<code>msr()</code>
Resampling	<code>mlr_resamplings</code>	<code>rsmp()</code>

```
mlr_measures
```

```
#> <DictionaryMeasure> with 30 stored values
#> Keys: classif.acc, classif.auc, classif.ce, classif.costs,
#>   classif.f1, classif.fdr, classif.fn, classif.fnr,
#>   classif.for, classif.fp, classif.fpr, classif.npv,
#>   classif.ppv, classif.precision, classif.recall,
#>   classif.sensitivity, classif.specificity, classif.tn,
#>   classif.tnr, classif.tp, classif.tpr, debug, oob_error,
#>   regr.mae, regr.mse, regr.rmse, selected_features,
#>   time_both, time_predict, time_train
```

# Details for Nerds

# MLR3 PHILOSOPHY

- Overcome limitations of S3 with the help of **R6**
  - Truly object-oriented: data and methods live in the same object
  - Make use of inheritance
  - Reference semantics
- Embrace **data.table**, both for arguments and internally
  - Fast operations for tabular data
  - List columns to arrange complex objects in tabular structure
- Be **light on dependencies**:
  - R6, data.table, Metrics, lgr, uuid, mlbench, digest
  - Plus some of our own packages (backports, checkmate, ...)

# INTERNAL DATA STRUCTURE

Results objects (`resample()`, `benchmark()`, ...) share the same structure

```
print(as.data.table(result))
```

```
#>           task           learner   resampling iterati...
#> 1: <TaskClassif> <LearnerClassifRpart> <ResamplingCV>    ...
#> 2: <TaskClassif> <LearnerClassifRpart> <ResamplingCV>    ...
#> 3: <TaskClassif> <LearnerClassifRpart> <ResamplingCV>    ...
#> 4: <TaskClassif> <LearnerClassifRpart> <ResamplingCV>    ...
#> 5: <TaskClassif> <LearnerClassifRpart> <ResamplingCV>    ...
```

## Combining R6 and data.table

- Not the objects are stored, but pointers to them
- Inexpensive to work on:
  - `rbind()`: copying R6 objects  $\leftrightarrow$  copying pointers
  - `cbind()`: `data.table()` over-allocates columns, no copies
  - `[i, ]`: lookup row (possibly hashed), create a list of pointers
  - `[, j]`: direct access to list elements

# CONTROL OF EXECUTION

## Parallelization

```
future::plan("multicore")
```

- runs each resampling iteration as a job
- also allows nested resampling (although not needed here)

## Encapsulation

```
learner$encapsulate = c(train = "callr", predict = "callr")
```

- Spawns a separate R process to train the learner
- Learner may segfault without tearing down the session
- Logs are captured
- Possibility to have a fallback to create predictions



# OUT-OF-MEMORY DATA

- Task stores data in a `DataBackend`:
  - `DataBackendDataTable`: Default backend for dense data (in-memory)
  - `DataBackendMatrix`: Backend for sparse numerical data (in-memory)
  - `DataBackendDplyr`: Backend for many DBMS (out-of-memory)
  - `DataBackendCbind`: Combine backends in a `cbind()` fashion (virtual)
  - `DataBackendRbind`: Combine backends in a `rbind()` fashion (virtual)
- Backends are immutable
  - Filtering rows or selecting columns just modifies the "view" on the data
  - Multiple tasks can share the same backend
- Example: Interface a read-only MariaDB with `DataBackendDplyr`, add generated features via `DataBackendDataTable`

# Outro

# SO YOU WANT TO DO ML IN R

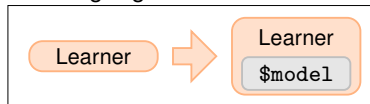
Ingredients:

Data



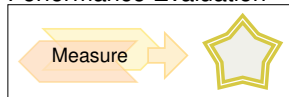

```
TaskClassif,  
TaskRegr,  
tsk()
```

Learning Algorithms



```
lrn(),  
$train(),  
$predict()
```

Performance Evaluation



```
msr(),  
resample(),  
$aggregate()
```

Performance Comparison




```
benchmark_grid(),  
benchmark()
```