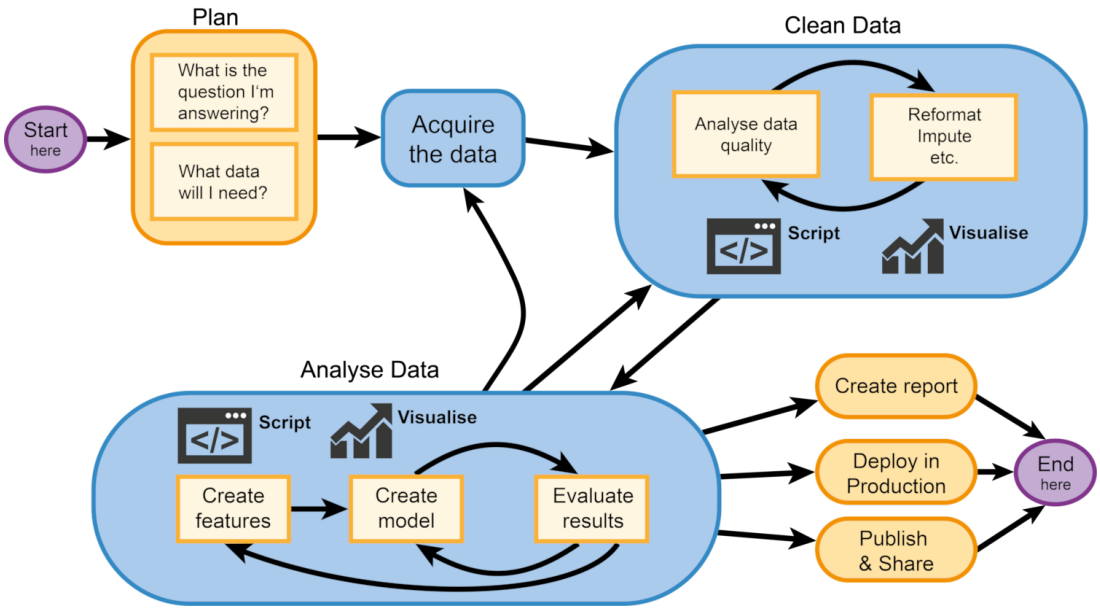
Machine Learning with R at LRZ

2020-10-07

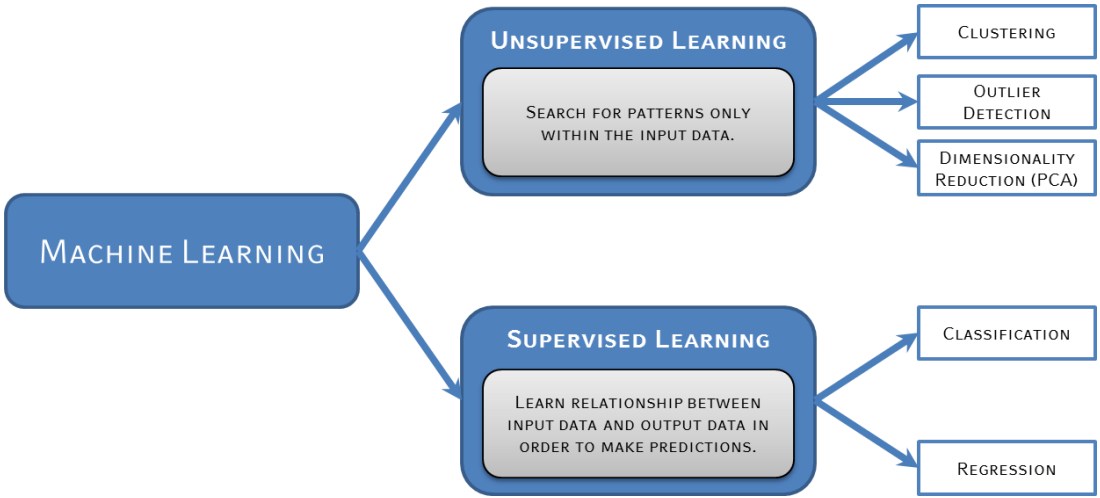
WHAT IS MACHINE LEARNING

Typical Workflow



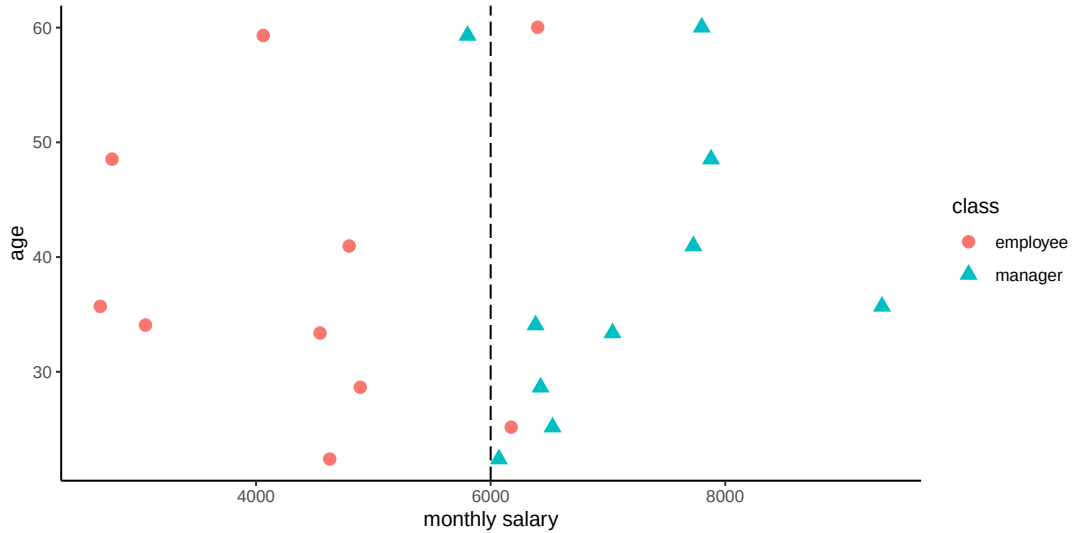
Machine Learning Tasks

Machine learning (ML) can be seen as the intersection between computer science and computational statistics in which computer algorithms learn to solve different tasks based on data (e.g. making predictions, finding groups, . . .).



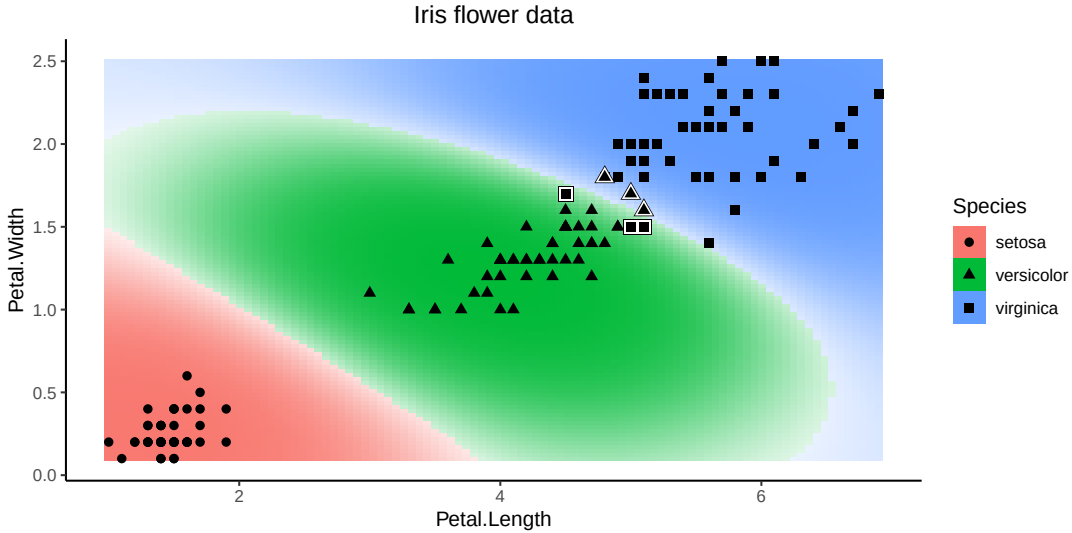
Binary Classification Task

- y is a categorical variable (with two values)
- E.g., sick/healthy, or credit/no credit
- **Goal**: Predict a class (or membership probabilities)



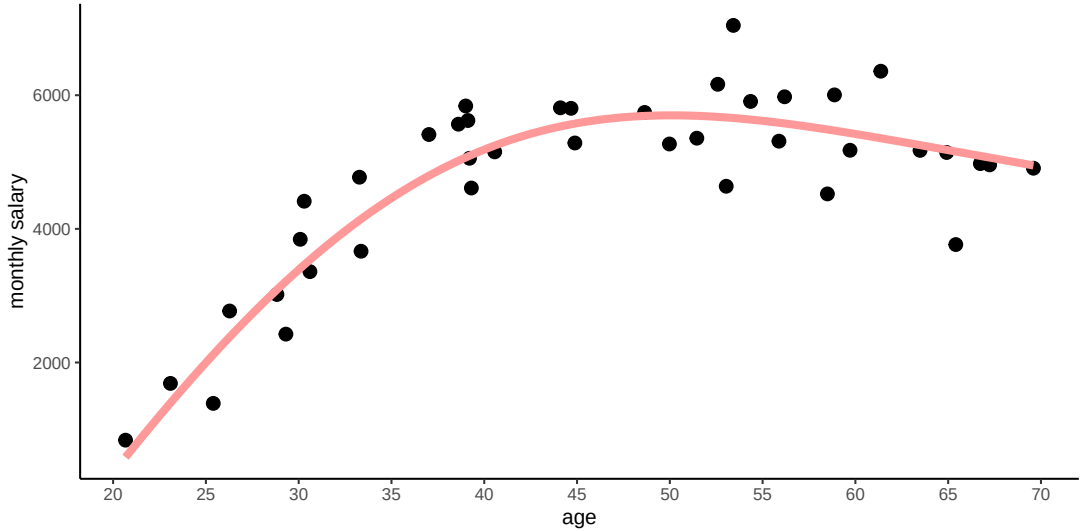
Multiclass Classification Task

- y is a categorical variable with > 2 unordered values
- Each instance belongs to only one class
- **Goal**: Predict a class (or membership probabilities)



Regression Task

- **Goal**: Predict a continuous output
- y is a metric variable (with values in $\mathbb{R}$)
- Regression model can be constructed by different methods, e.g., linear regression, trees or splines



`mlr3`: Modern Machine Learning in R

SO YOU WANT TO DO ML IN R

- R gives you access to many machine learning methods

SO YOU WANT TO DO ML IN R

- R gives you access to many machine learning methods
- . . . but without a unified interface

SO YOU WANT TO DO ML IN R

- R gives you access to many machine learning methods
- . . . but without a unified interface
- things like performance evaluation are cumbersome

SO YOU WANT TO DO ML IN R

- R gives you access to many machine learning methods
- . . . but without a unified interface
- things like performance evaluation are cumbersome

Example:

```
# Specify what we want to model in a formula: target ~ features
svm_model = e1071::svm(Species ~ ., data = iris)
```

SO YOU WANT TO DO ML IN R

- R gives you access to many machine learning methods
- . . . but without a unified interface
- things like performance evaluation are cumbersome

Example:

```
# Specify what we want to model in a formula: target ~ features
svm_model = e1071::svm(Species ~ ., data = iris)
```

vs.

```
# Pass the features as a matrix and the target as a vector
xgb_model = xgboost::xgboost(data = as.matrix(iris[1:4]),
  label = iris$Species, nrounds = 10)
```

SO YOU WANT TO DO ML IN R

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

Ingredients:

- Data / Task
- Learning Algorithms
- Performance Evaluation
- Performance Comparison

R6

R6 – ALL YOU NEED TO KNOW

`mlr3` uses the *R6* class system. Some things may seem unusual if you see them for the first time.

- *Objects* are created using `<Class>$new()`.

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

R6 – ALL YOU NEED TO KNOW

`mlr3` uses the *R6* class system. Some things may seem unusual if you see them for the first time.

- *Objects* are created using `<Class>$new()`.

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

- Objects have *fields* that contain information about the object.

```
task$nrow
#> [1] 150
```

R6 – ALL YOU NEED TO KNOW

`mlr3` uses the *R6* class system. Some things may seem unusual if you see them for the first time.

- *Objects* are created using `<Class>$new()`.

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

- Objects have *fields* that contain information about the object.

```
task$nrow
#> [1] 150
```

- Objects have *methods* that are called like functions:

```
task$filter(rows = 1:10)
```

R6 – ALL YOU NEED TO KNOW

`mlr3` uses the *R6* class system. Some things may seem unusual if you see them for the first time.

- *Objects* are created using `<Class>$new()`.

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

- Objects have *fields* that contain information about the object.

```
task$nrow
#> [1] 150
```

- Objects have *methods* that are called like functions:

```
task$filter(rows = 1:10)
```

- Methods may change ("mutate") the object (reference semantics)!

```
task$nrow
#> [1] 10
```

R6 AND ACTIVE BINDINGS

Some fields of R6-objects may be *"Active Bindings"*. Internally they are realized as functions that are called whenever the value is set or retrieved.

- Active bindings for read-only fields

```
task$nrow = 11
#> Error:  Field/Binding is read-only
```

R6 AND ACTIVE BINDINGS

Some fields of R6-objects may be *"Active Bindings"*. Internally they are realized as functions that are called whenever the value is set or retrieved.

- Active bindings for read-only fields

```
task$nrow = 11
#> Error:  Field/Binding is read-only
```

- Active bindings for argument checking

```
task$properties = NULL
#> Error in assert_set(rhs, .var.name = "properties"):
Assertion on 'properties' failed:  Must be of type
'character', not 'NULL'.
task$properties = c("property1", "property2")  # works
```

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

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

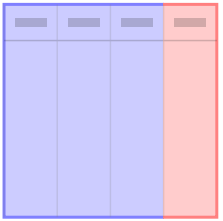
MLR3 PHILOSOPHY

- 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`, `lgr`, `uuid`, `mlbench`, `digest`
  - Plus some of our own packages (`backports`, `checkmate`, ...)

Data

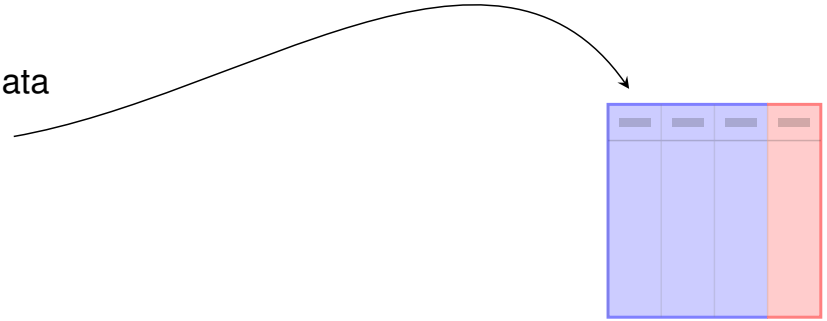
DATA

- Tabular data



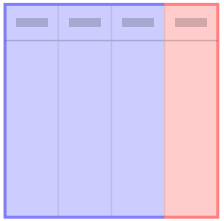
DATA

- Tabular data
- Features



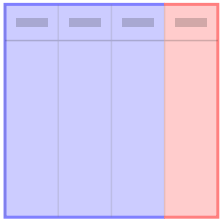
DATA

- Tabular data
- Features
- Target / outcome to predict



DATA

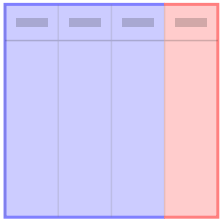
- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression



DATA

- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression

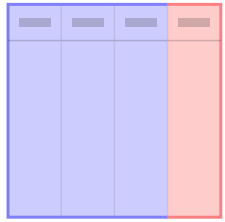
  $\Rightarrow$ target determines the machine learning "Task"



DATA

- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression

  $\Rightarrow$ target determines the machine learning "Task"

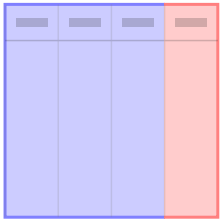


```
print(iris)  # included in R

#>   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
#> ...
```

DATA

- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression

  $\Rightarrow$ target determines the machine learning "Task"

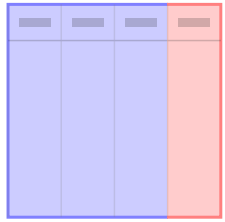


```
print(iris)  # included in R

#>   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
#> ...
```

DATA

- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression

  $\Rightarrow$ target determines the machine learning "Task"



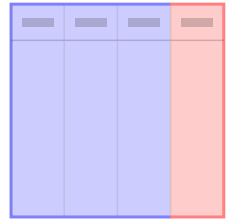
```
print(iris)  # included in R

#>   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 = TaskClassif$new("iris", iris, "Species")
```

DATA

- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression

  $\Rightarrow$ target 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
#> ...
```

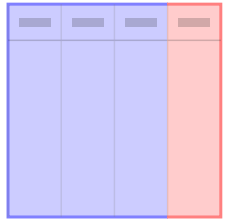
Task ID

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

DATA

- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression

  $\Rightarrow$ target 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

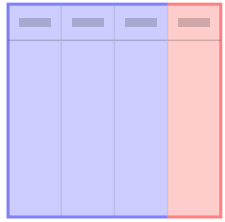
data

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

DATA

- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression

  $\Rightarrow$ target 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 → "iris", data → iris, target name → "Species"

```
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 = )
```

Dictionaries

DICTIONARIES

- Ordinary constructors: `TaskClassif$new()` / `LearnerClassifRpart$new()`

DICTIONARIES

- Ordinary constructors: `TaskClassif$new()` / `LearnerClassifRpart$new()`

$\Rightarrow$ `mlr3` offers *Short Form Constructors* that are less verbose

DICTIONARIES

- Ordinary constructors: `TaskClassif$new()` / `LearnerClassifRpart$new()`

$\Rightarrow$ `mlr3` offers *Short Form Constructors* that are less verbose

- They access `Dictionary` of objects:

DICTIONARIES

- Ordinary constructors: `TaskClassif$new()` / `LearnerClassifRpart$new()`

$\Rightarrow$ `mlr3` offers *Short Form Constructors* that are less verbose

- They access `Dictionary` of objects:

Object	Dictionary	Short Form
`Task`	`mlr_tasks`	`tsk()`
`Learner`	`mlr_learners`	`lrn()`
`Measure`	`mlr_measures`	`msr()`
`Resampling`	`mlr_resamplings`	`rsmp()`

Dictionaries can get populated by add-on packages (e.g. `mlr3learners`)

DICTIONARIES

```
# list items
tsk()

#> <DictionaryTask> with 10 stored values
#> Keys: boston_housing, breast_cancer, german_credit, iris,
#>   mtcars, pima, sonar, spam, wine, zoo

# retrieve object
tsk("iris")

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

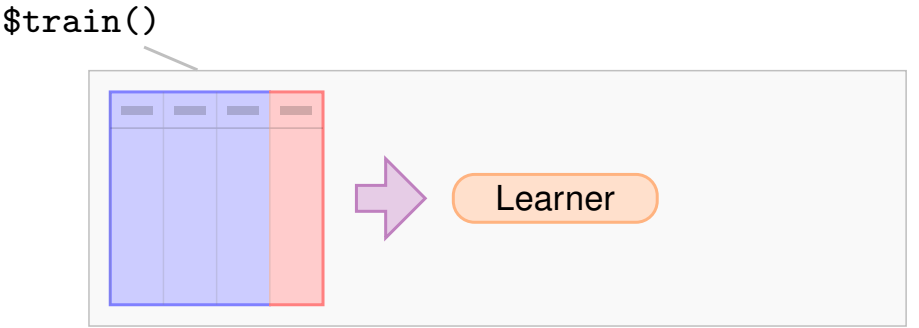
SHORT FORMS AND DICTIONARIES

`as.data.table(<DICTIONARY>)` creates a `data.table` with metadata about objects in dictionaries:

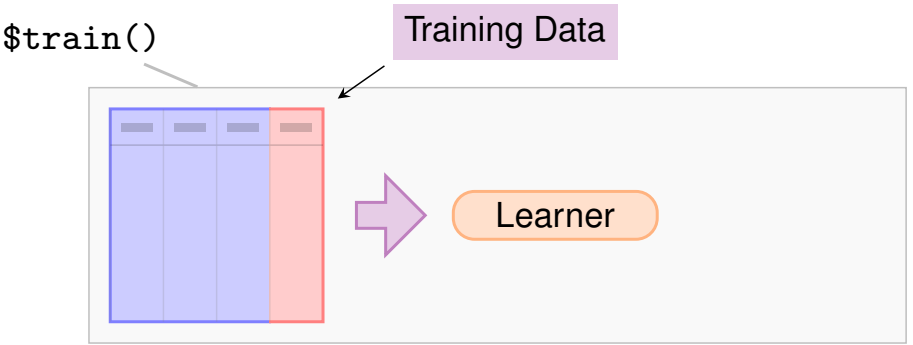
```
mlr_learners_table = as.data.table(mlr_learners)

mlr_learners_table[1:10, c("key", "packages", "predict_types")]

#                    key packages predict_types
#  1:   classif.cv_glmnet   glmnet response,prob
#  2:       classif.debug          response,prob
#  3: classif.featureless          response,prob
#  4:      classif.glmnet   glmnet response,prob
#  5:        classif.kknn     kknn response,prob
#  6:         classif.lda     MASS response,prob
#  7:     classif.log_reg    stats response,prob
#  8:    classif.multinom     nnet response,prob
#  9: classif.naive_bayes    e1071 response,prob
# 10:         classif.qda     MASS response,prob
```

Learning Algorithms

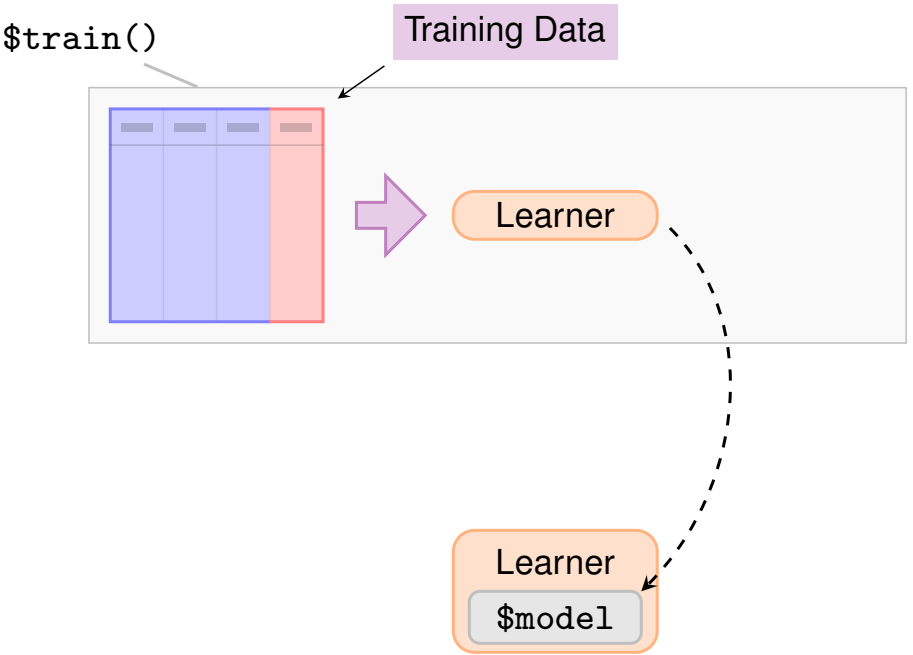
LEARNING ALGORITHMS



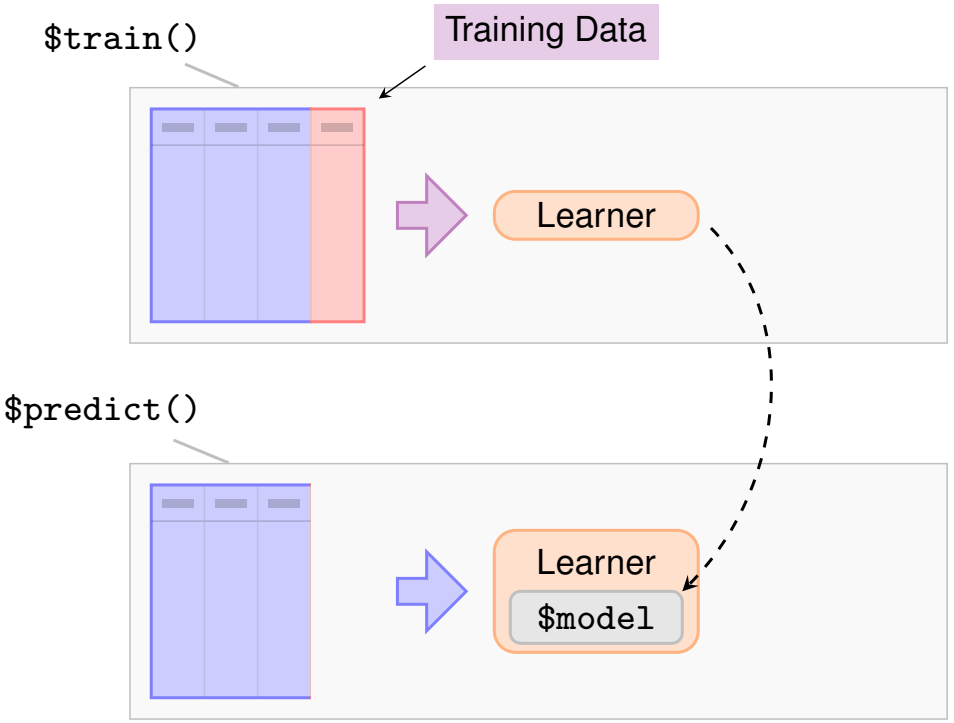
LEARNING ALGORITHMS



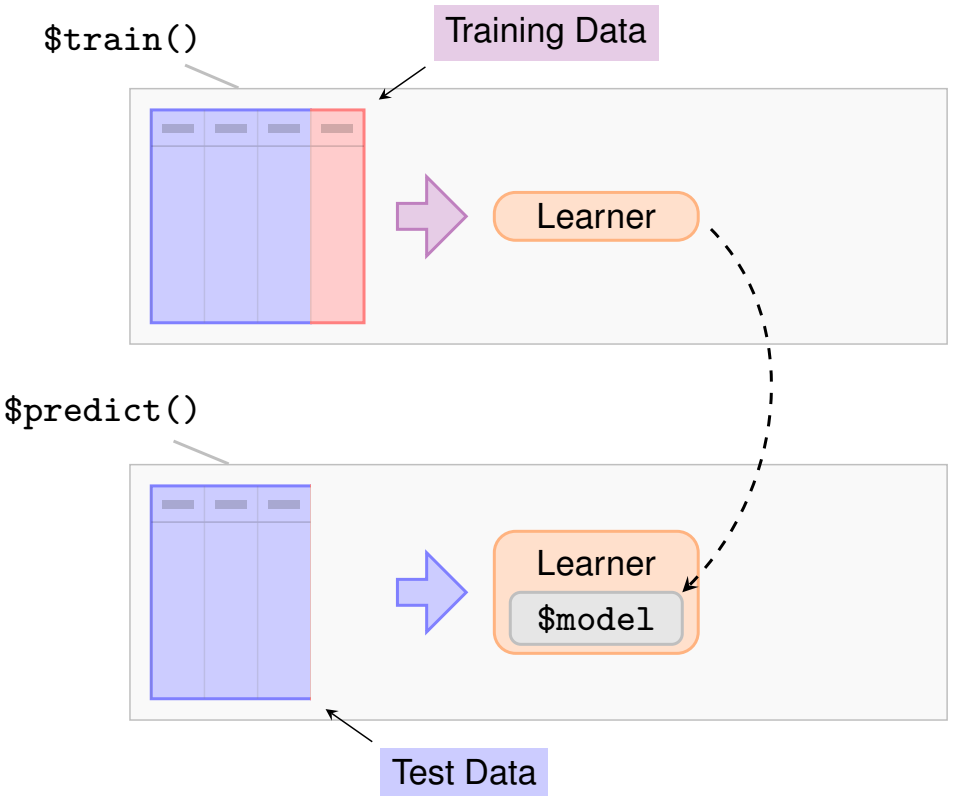
LEARNING ALGORITHMS



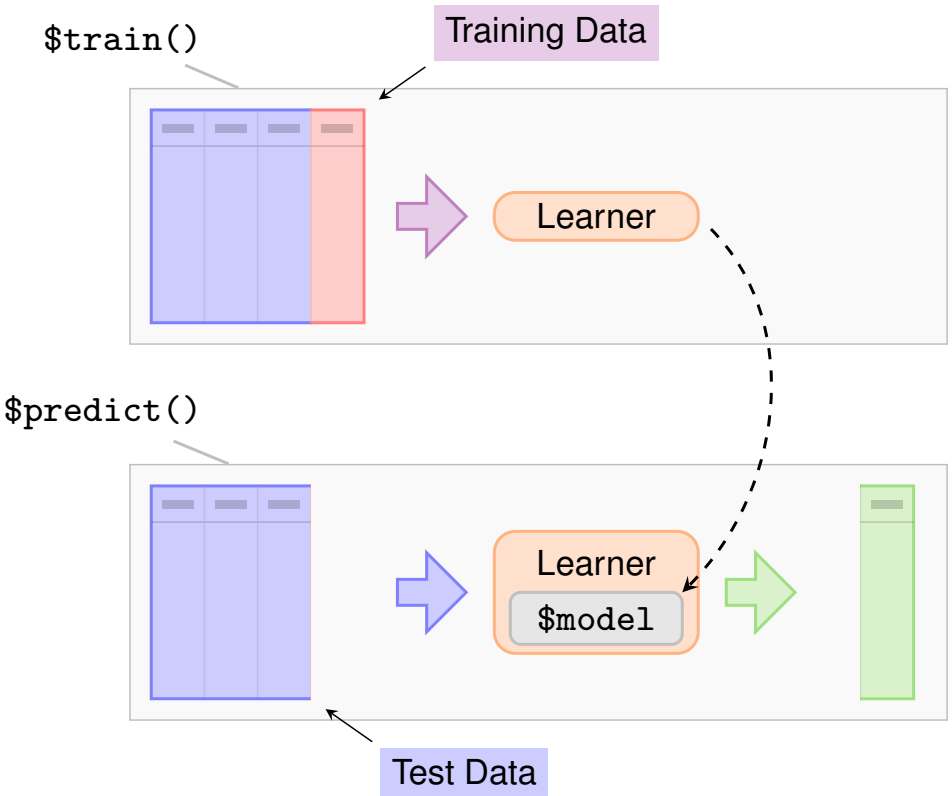
LEARNING ALGORITHMS



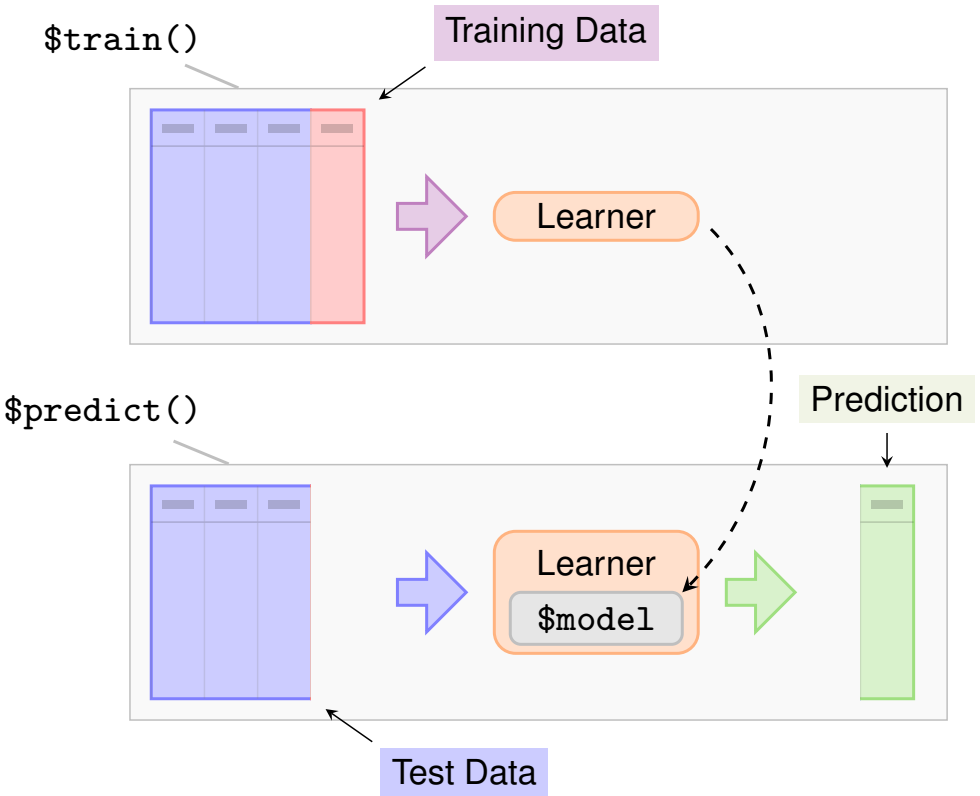
LEARNING ALGORITHMS



LEARNING ALGORITHMS



LEARNING ALGORITHMS



LEARNING ALGORITHMS

- Get a `Learner` provided by `mlr`

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

LEARNING ALGORITHMS

- Get a `Learner` provided by `mlr`

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

- Train the `Learner`

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

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< 2.4 50   0 setosa (1.000 0.000 0.000) *
#>   3) Petal.Length>=2.4 100  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*

```
as.data.table(learner$param_set)[, 1:6]
```

```
#>                 id    class lower upper      levels nlevels
#>  1:       minsplit ParamInt     1   Inf                 Inf
#>  2:      minbucket ParamInt     1   Inf                 Inf
#>  3:             cp ParamDbl     0     1                 Inf
#>  4:     maxcompete ParamInt     0   Inf                 Inf
#>  5:   maxsurrogate ParamInt     0   Inf                 Inf
#>  6:       maxdepth ParamInt     1    30                  30
#>  7:   usesurrogate ParamInt     0     2                   3
#>  8: surrogatestyle ParamInt     0     1                   2
#>  9:           xval ParamInt     0   Inf                 Inf
#> 10:     keep_model ParamLgl    NA    NA  TRUE,FALSE       2
```

HYPERPARAMETERS

- Learners have *hyperparameters*

```
as.data.table(learner$param_set)[, 1:6]

#>                 id    class lower upper      levels nlevels
#>  1:       minsplit ParamInt     1   Inf                 Inf
#>  2:      minbucket ParamInt     1   Inf                 Inf
#>  3:             cp ParamDbl     0     1                 Inf
#>  4:     maxcompete ParamInt     0   Inf                 Inf
#>  5:   maxsurrogate ParamInt     0   Inf                 Inf
#>  6:       maxdepth ParamInt     1    30                  30
#>  7:   usesurrogate ParamInt     0     2                   3
#>  8: surrogatestyle ParamInt     0     1                   2
#>  9:           xval ParamInt     0   Inf                 Inf
#> 10:     keep_model ParamLgl    NA    NA  TRUE,FALSE       2
```

- Changing them changes the `Learner` behavior

```
learner$param_set$values = list(maxdepth = 1, xval = 0)

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< 2.4 50   0 setosa (1.00 0.00 0.00) *
#>   3) Petal.Length>=2.4 100  50 versicolor (0.00 0.50 0.50) *
```

PREDICTION

- Let's make a prediction for some new data, e.g.:

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

PREDICTION

- Let's make a prediction for some new data, e.g.:

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

- To do so, we call the `$predict_newdata()` method using the new data:

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

PREDICTION

- Let's make a prediction for some new data, e.g.:

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

- To do so, we call the `$predict_newdata()` method using the new data:

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

- We get a `Prediction` object:

```
prediction
#> <PredictionClassif> for 2 observations:
#>  row_id truth   response
#>       1  <NA>     setosa
#>       2  <NA> versicolor
```

PREDICTION

- Let's make a prediction for some new data, e.g.:

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

- To do so, we call the `$predict_newdata()` method using the new data:

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

- We get a `Prediction` object:

```
prediction
#> <PredictionClassif> for 2 observations:
#>  row_id truth   response
#>       1  <NA>     setosa
#>       2  <NA> versicolor
```

PREDICTION

- Let's make a prediction for some new data, e.g.:

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

- To do so, we call the `$predict_newdata()` method using the new data:

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

- We get a `Prediction` object:

```
prediction
#> <PredictionClassif> for 2 observations:
#>  row_id truth  response
#>       1  <NA>     setosa
#>       2  <NA> versicolor
```

PREDICTION

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

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

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

PREDICTION

What exactly is a `Prediction` object?

- Contains predictions and offers useful access fields / methods

PREDICTION

What exactly is a `Prediction` object?

- Contains predictions and offers useful access fields / methods

$\Rightarrow$ Use `as.data.table()` to extract data

```
as.data.table(prediction)
#>    row_id truth   response
#> 1:      1  <NA>     setosa
#> 2:      2  <NA> versicolor
```

PREDICTION

What exactly is a `Prediction` object?

- Contains predictions and offers useful access fields / methods

⇒ Use `as.data.table()` to extract data

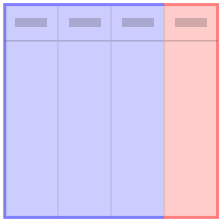
```
as.data.table(prediction)
#>    row_id truth   response
#> 1:      1  <NA>     setosa
#> 2:      2  <NA> versicolor
```

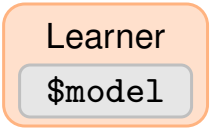
⇒ Active bindings and functions that give further information: `$response`, `$truth`, ...

```
prediction$response
#> [1] setosa     versicolor
#> Levels: setosa versicolor virginica
```

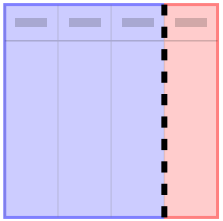
Performance

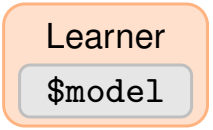
PERFORMANCE EVALUATION



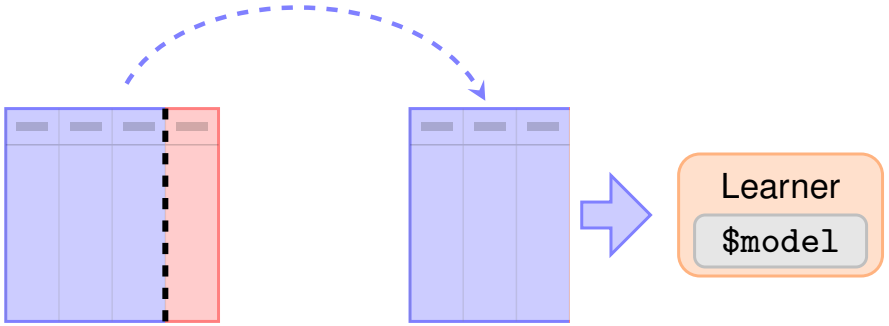


PERFORMANCE EVALUATION

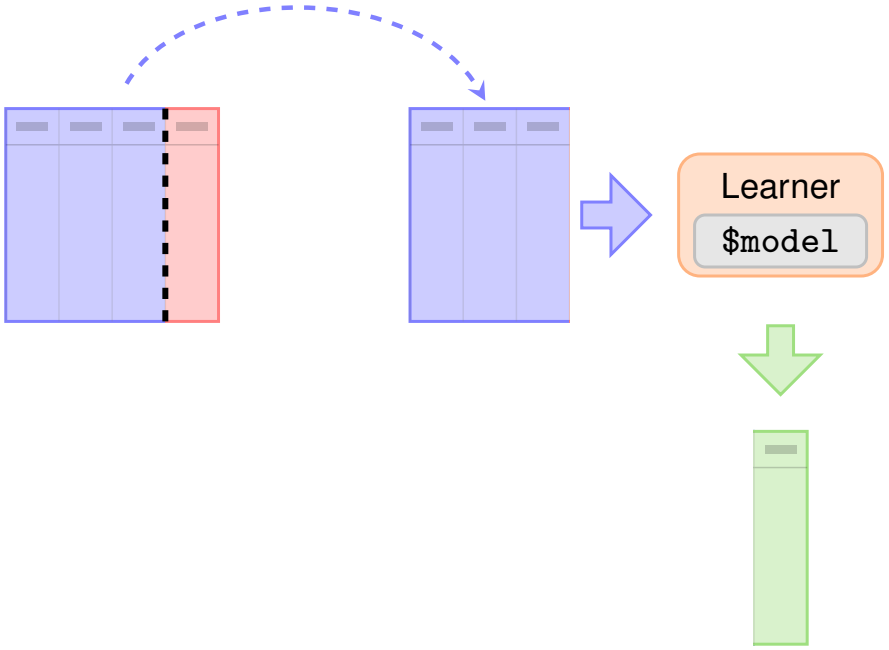




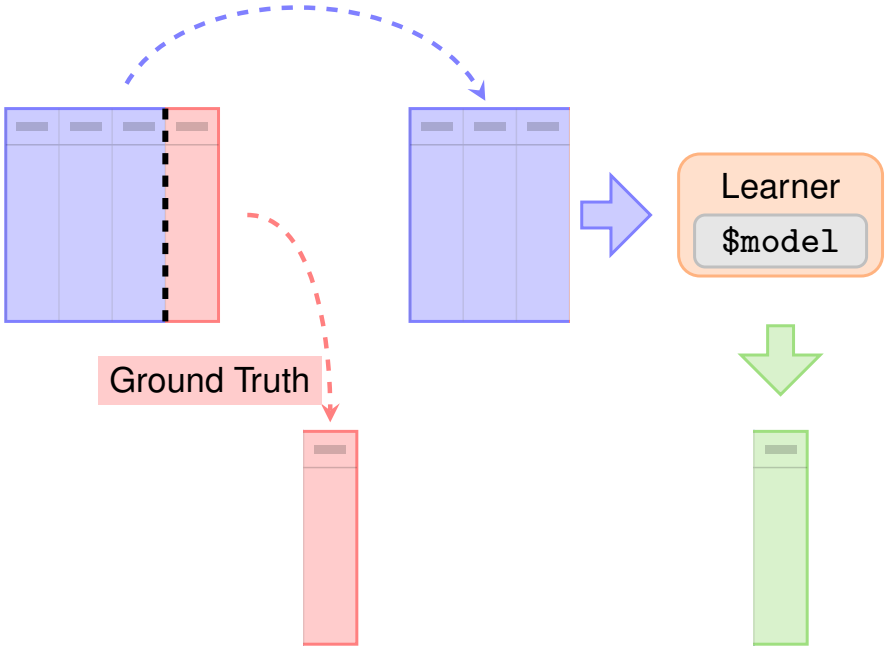
PERFORMANCE EVALUATION



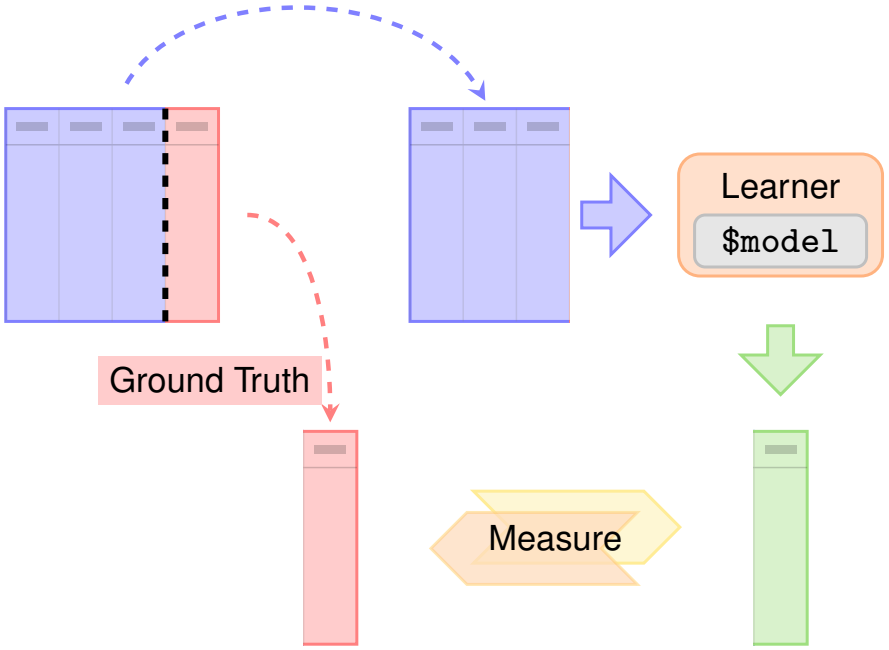
PERFORMANCE EVALUATION



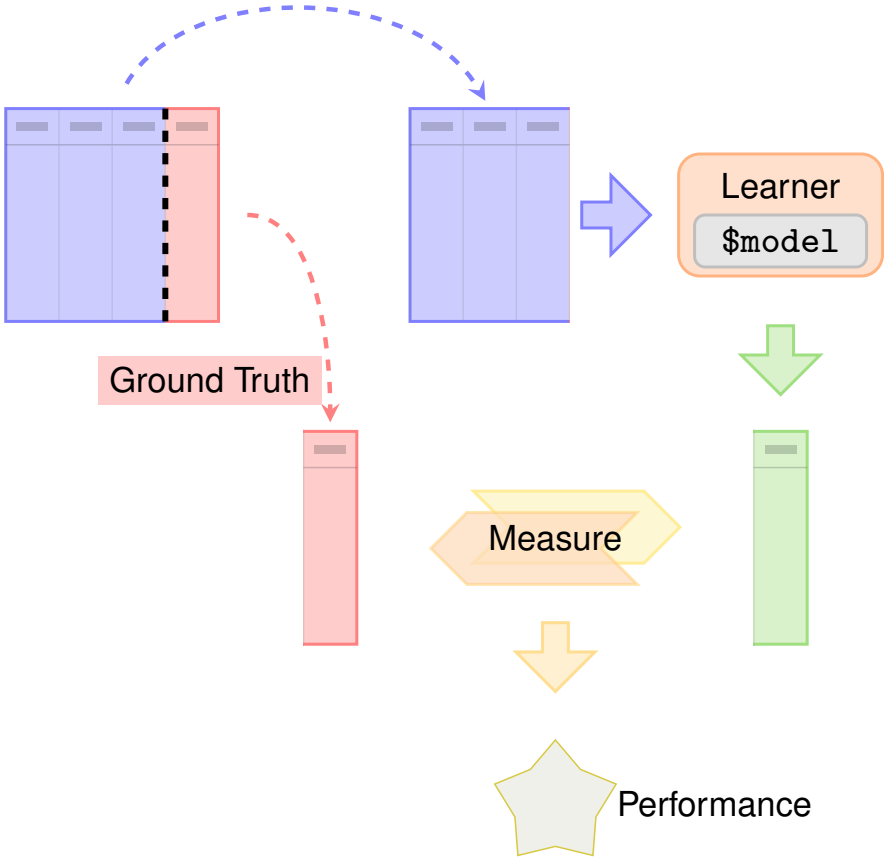
PERFORMANCE EVALUATION



PERFORMANCE EVALUATION



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
```

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
```

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

- 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

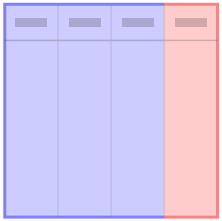
```
pred = learner$predict(known_truth_task)
pred
#> <PredictionClassif> for 2 observations:
#>  row_id  truth  response
#>       1 setosa    setosa
#>       2 setosa virginica
```

- Score the prediction

```
pred$score(msr("classif.ce"))
#> classif.ce
#>        0.5
```

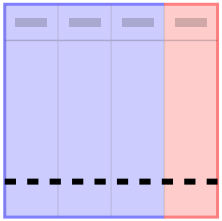
Resampling

RESAMPLING



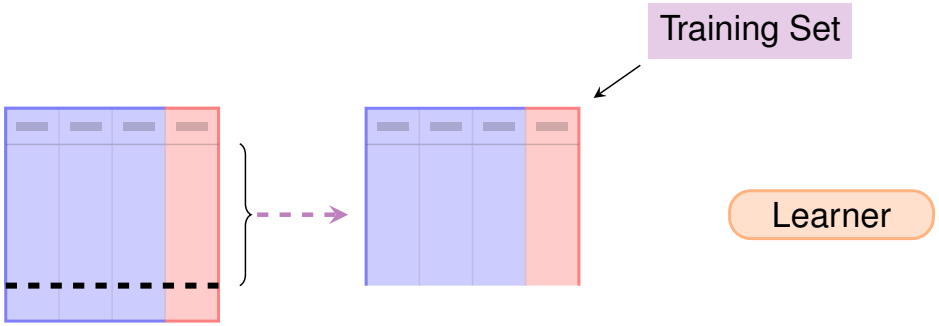
Learner

RESAMPLING

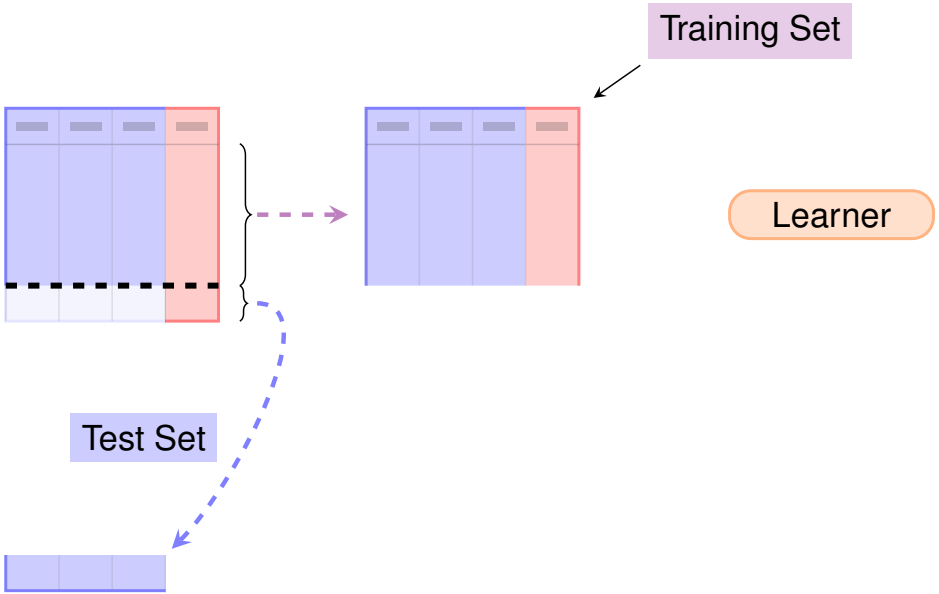


Learner

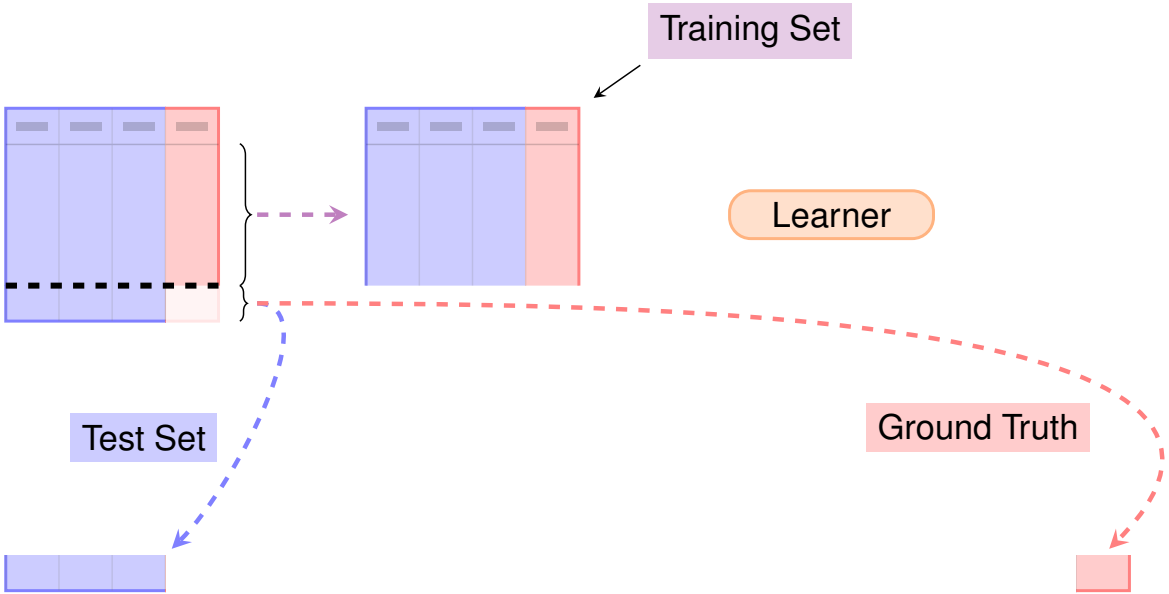
RESAMPLING



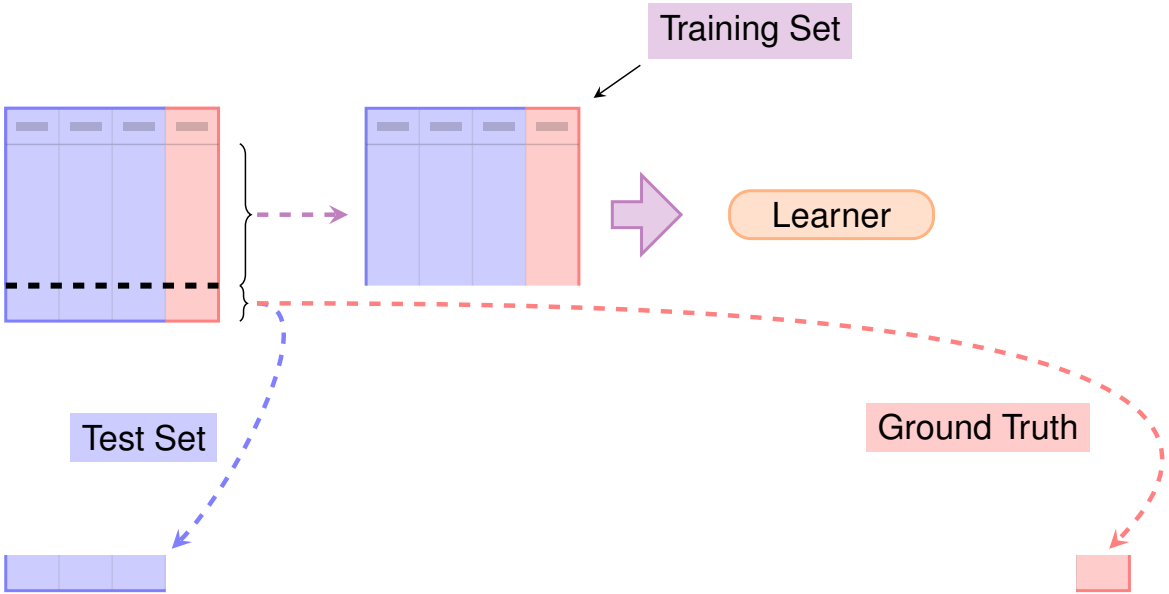
RESAMPLING



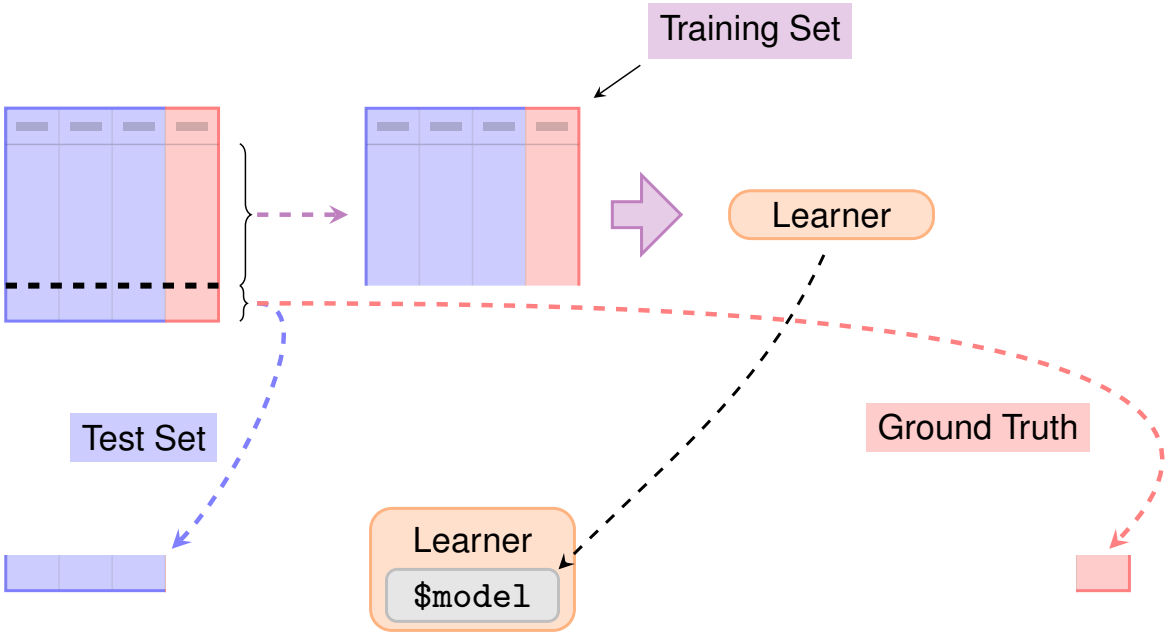
RESAMPLING



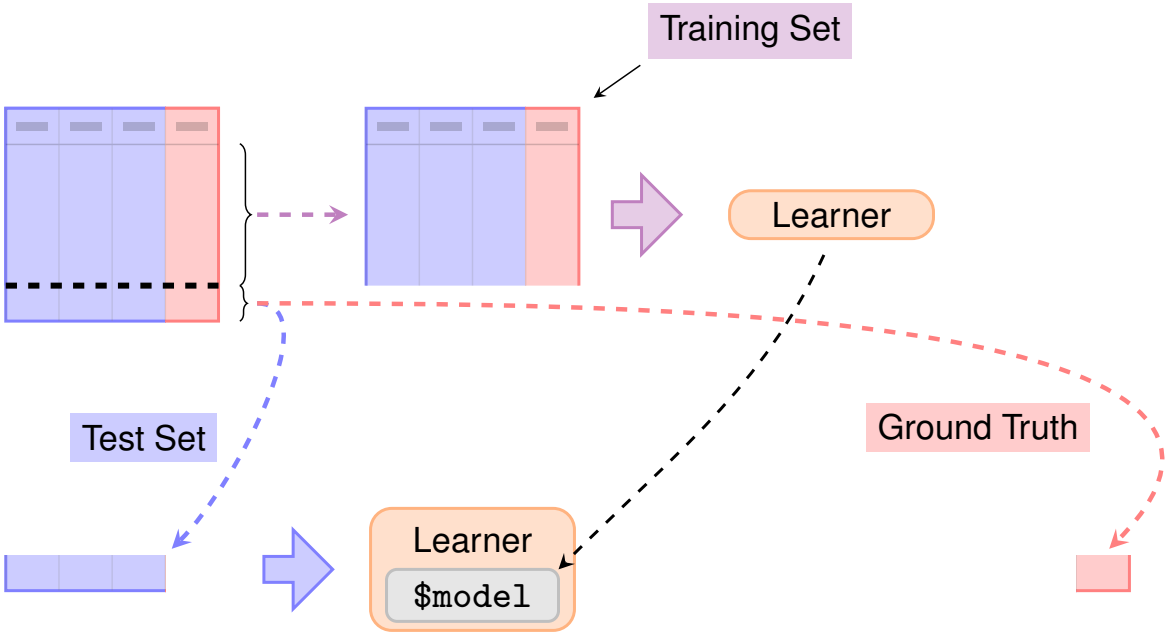
RESAMPLING



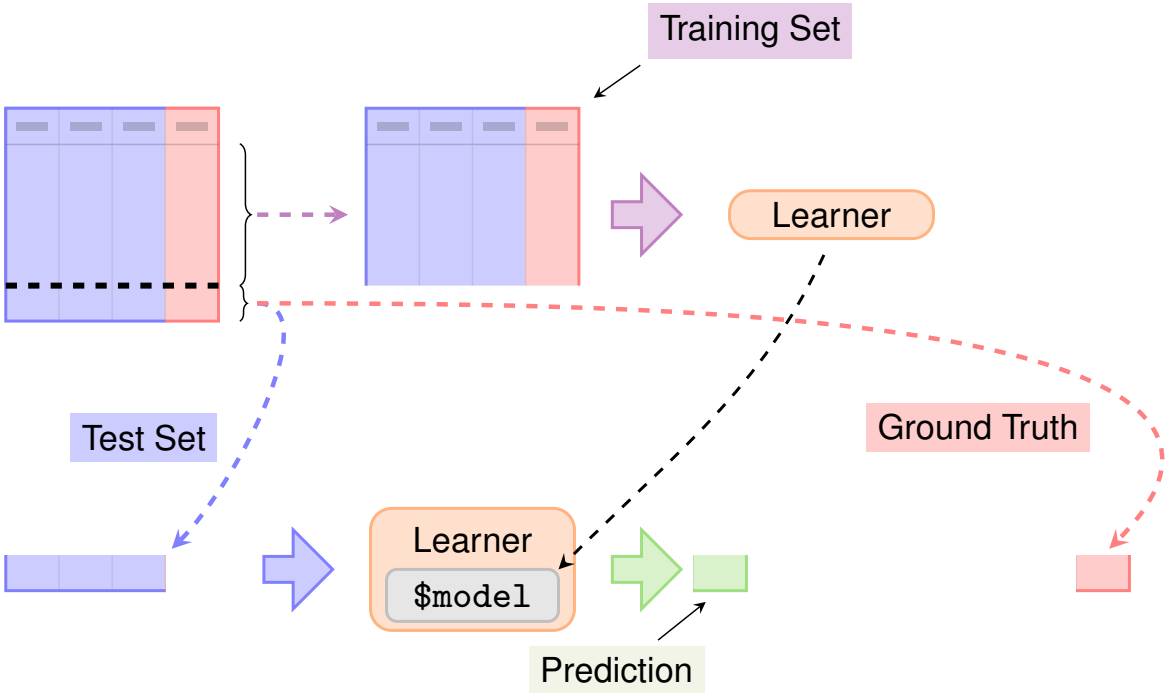
RESAMPLING



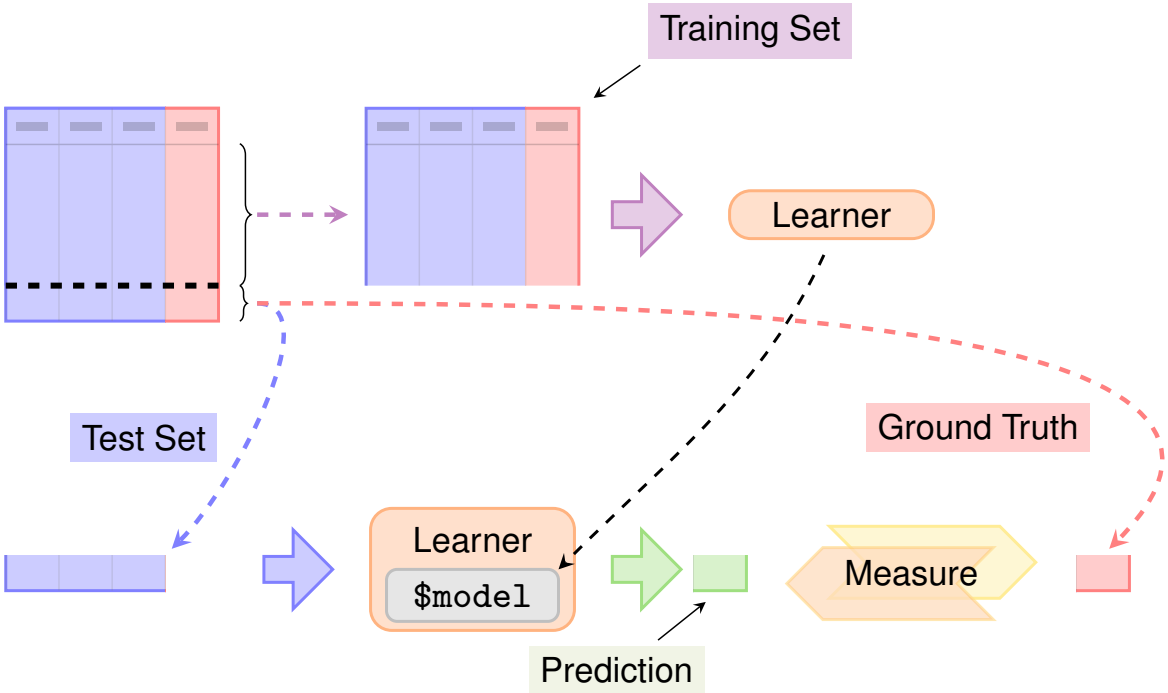
RESAMPLING



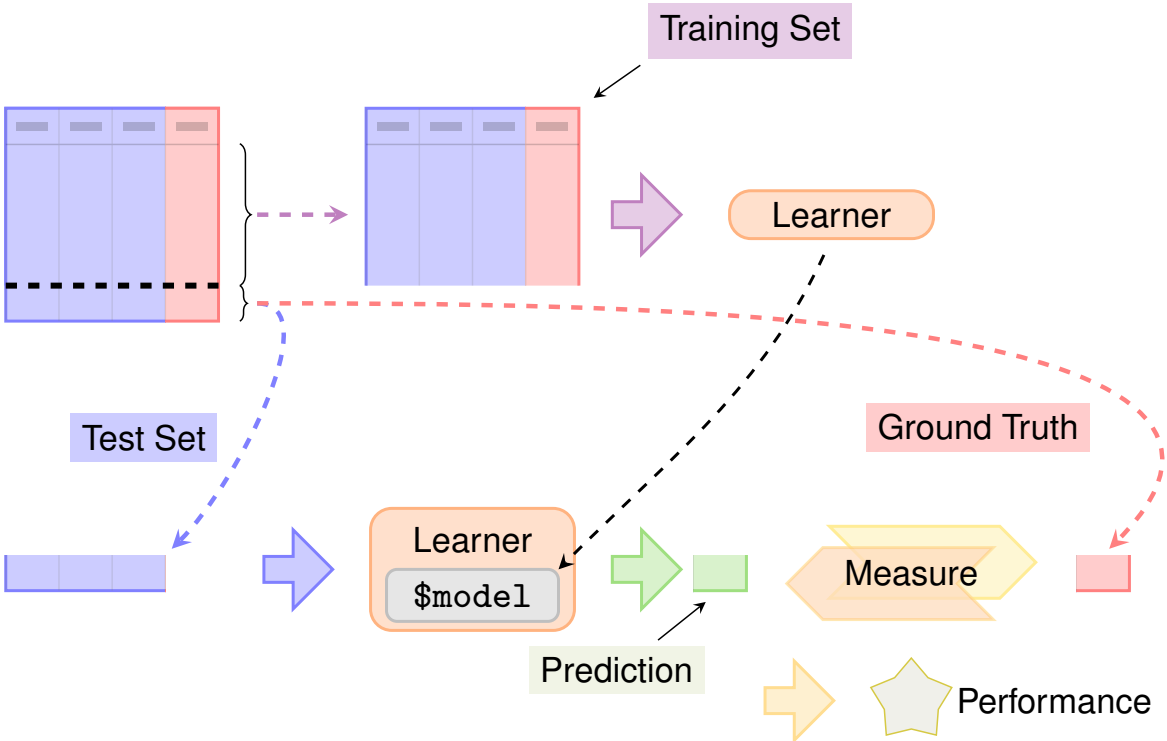
RESAMPLING


Training Set
Learner
Test Set
Ground Truth
Learner
$model
Prediction

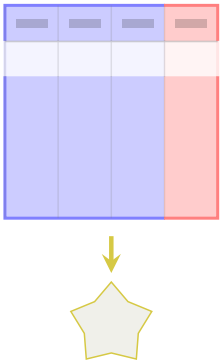
RESAMPLING


Training Set
Learner
Test Set
Ground Truth
Learner
$model
Measure
Prediction

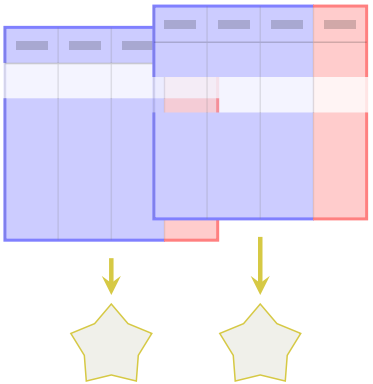
RESAMPLING


Training Set
Learner
Test Set
Ground Truth
Learner
$model
Measure
Prediction
Performance

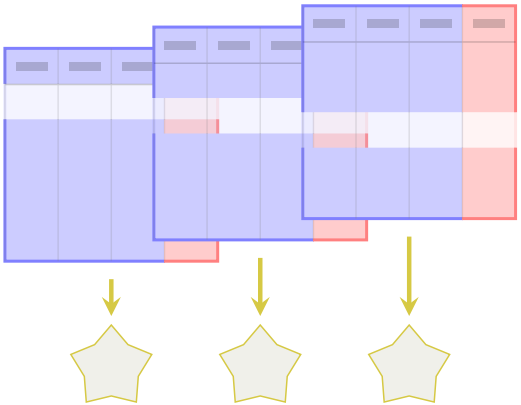
RESAMPLING



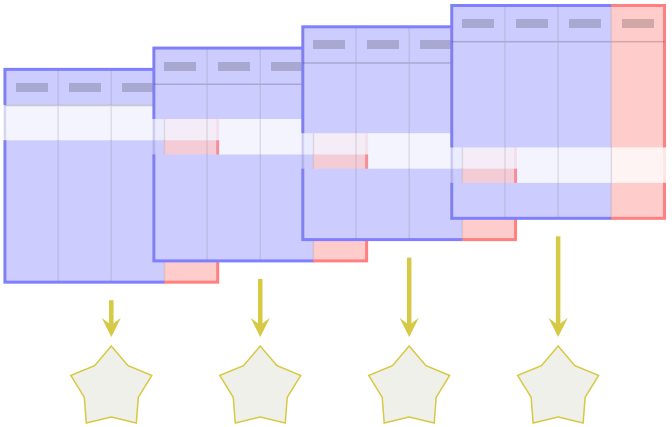
RESAMPLING



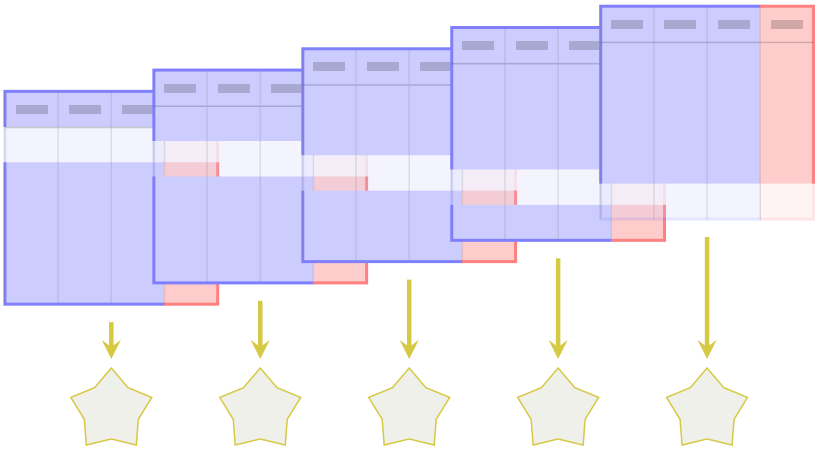
RESAMPLING



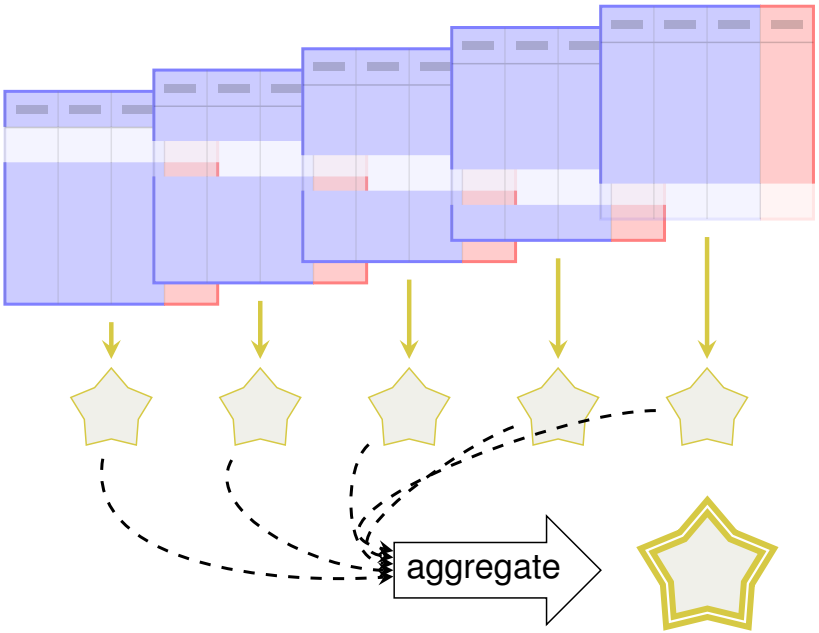
RESAMPLING



RESAMPLING

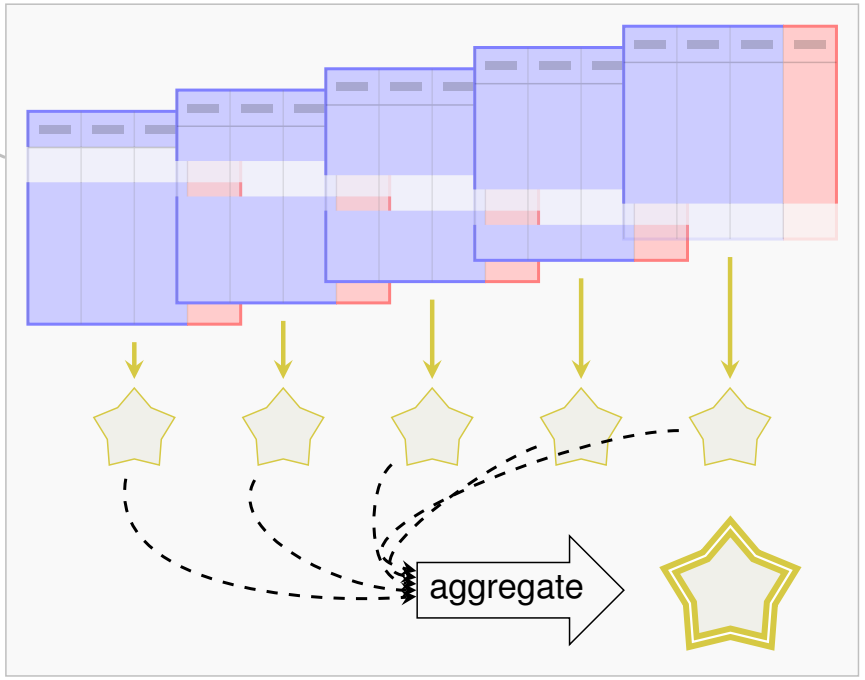


RESAMPLING



RESAMPLING

`resample()`



RESAMPLING

- Resample description: How to split the data

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

RESAMPLING

- Resample description: How to split the data

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

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

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

RESAMPLING

- Resample description: How to split the data

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

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

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

- We get a `ResamplingResult` object:

```
print(rr)
#> <ResampleResult> of 5 iterations
#> * Task: iris
#> * Learner: classif.rpart
#> * Warnings: 0 in 0 iterations
#> * Errors: 0 in 0 iterations
```

RESAMPLING RESULTS

What exactly is a `ResamplingResult` object?

RESAMPLING RESULTS

What exactly is a `ResamplingResult` object?
Remember `Prediction`:

RESAMPLING RESULTS

What exactly is a `ResamplingResult` object?
Remember `Prediction`:

- Get a table representation using `as.data.table()`

```
rr_table = as.data.table(rr)

print(rr_table)
#                  task                    learner         resampling
# 1: <TaskClassif[44]> <LearnerClassifRpart[32]> <ResamplingCV[19]>
# 2: <TaskClassif[44]> <LearnerClassifRpart[32]> <ResamplingCV[19]>
# 3: <TaskClassif[44]> <LearnerClassifRpart[32]> <ResamplingCV[19]>
# 4: <TaskClassif[44]> <LearnerClassifRpart[32]> <ResamplingCV[19]>
# 5: <TaskClassif[44]> <LearnerClassifRpart[32]> <ResamplingCV[19]>
#    iteration prediction
# 1:         1  <list[1]>
# 2:         2  <list[1]>
# 3:         3  <list[1]>
# 4:         4  <list[1]>
# 5:         5  <list[1]>
```

RESAMPLING RESULTS

What exactly is a `ResamplingResult` object?
Remember `Prediction`:

- Get a table representation using `as.data.table()`

```
rr_table = as.data.table(rr)

print(rr_table)
#                  task                    learner         resampling
# 1: <TaskClassif[44]> <LearnerClassifRpart[32]> <ResamplingCV[19]>
# 2: <TaskClassif[44]> <LearnerClassifRpart[32]> <ResamplingCV[19]>
# 3: <TaskClassif[44]> <LearnerClassifRpart[32]> <ResamplingCV[19]>
# 4: <TaskClassif[44]> <LearnerClassifRpart[32]> <ResamplingCV[19]>
# 5: <TaskClassif[44]> <LearnerClassifRpart[32]> <ResamplingCV[19]>
#    iteration prediction
# 1:         1  <list[1]>
# 2:         2  <list[1]>
# 3:         3  <list[1]>
# 4:         4  <list[1]>
# 5:         5  <list[1]>
```

- Active bindings and functions that make information easily accessible

RESAMPLING RESULTS

- Calculate performance:

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

RESAMPLING RESULTS

- Calculate performance:

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

- Get predictions

```
rr$prediction()
#> <PredictionClassif> for 150 observations:
#>     row_id     truth   response
#>          2    setosa     setosa
#>          6    setosa     setosa
#>         10    setosa     setosa
#> ---
#>        128 virginica versicolor
#>        147 virginica  virginica
#>        150 virginica  virginica
```

RESAMPLING

- Predictions of individual folds

```
predictions = rr$predictions()
predictions[[1]]
```

```
#> <PredictionClassif> for 30 observations:
#>     row_id     truth  response
#>          2    setosa    setosa
#>          6    setosa    setosa
#>         10    setosa    setosa
#> ---
#>        136 virginica virginica
#>        139 virginica virginica
#>        140 virginica virginica
```

RESAMPLING

- Predictions of individual folds

```
predictions = rr$predictions()
predictions[[1]]
```

```
#> <PredictionClassif> for 30 observations:
#>     row_id     truth  response
#>          2    setosa    setosa
#>          6    setosa    setosa
#>         10    setosa    setosa
#> ---
#>        136 virginica virginica
#>        139 virginica virginica
#>        140 virginica virginica
```

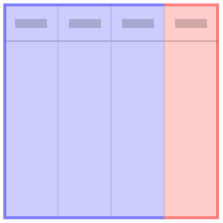
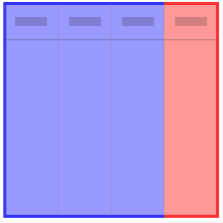
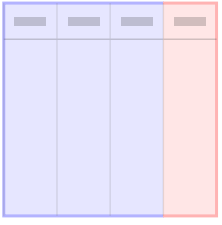
- Score of individual folds

```
scores = rr$score()
scores[1:3, c("iteration", "classif.ce")]
```

```
#>    iteration classif.ce
#> 1:         1      0.000
#> 2:         2      0.067
#> 3:         3      0.133
```

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"))
```

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)
```

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)
```

- We get a `BenchmarkResult` object which shows that `kknn` outperforms `rpart`:

```
bmr_ag = bmr$aggregate()
bmr_ag[, c("task_id", "learner_id", "classif.ce")]
```

```
#>    task_id    learner_id classif.ce
#> 1:    iris classif.rpart      0.053
#> 2:    iris  classif.kknn      0.033
#> 3:   sonar classif.rpart      0.327
#> 4:   sonar  classif.kknn      0.158
#> 5:    wine classif.rpart      0.136
#> 6:    wine  classif.kknn      0.045
```

BENCHMARK RESULT

What exactly is a `BenchmarkResult` object?

BENCHMARK RESULT

What exactly is a `BenchmarkResult` object?
Just like `Prediction` and `ResamplingResult`!

BENCHMARK RESULT

What exactly is a `BenchmarkResult` object?
Just like `Prediction` and `ResamplingResult`!

- Table representation using `as.data.table()`

BENCHMARK RESULT

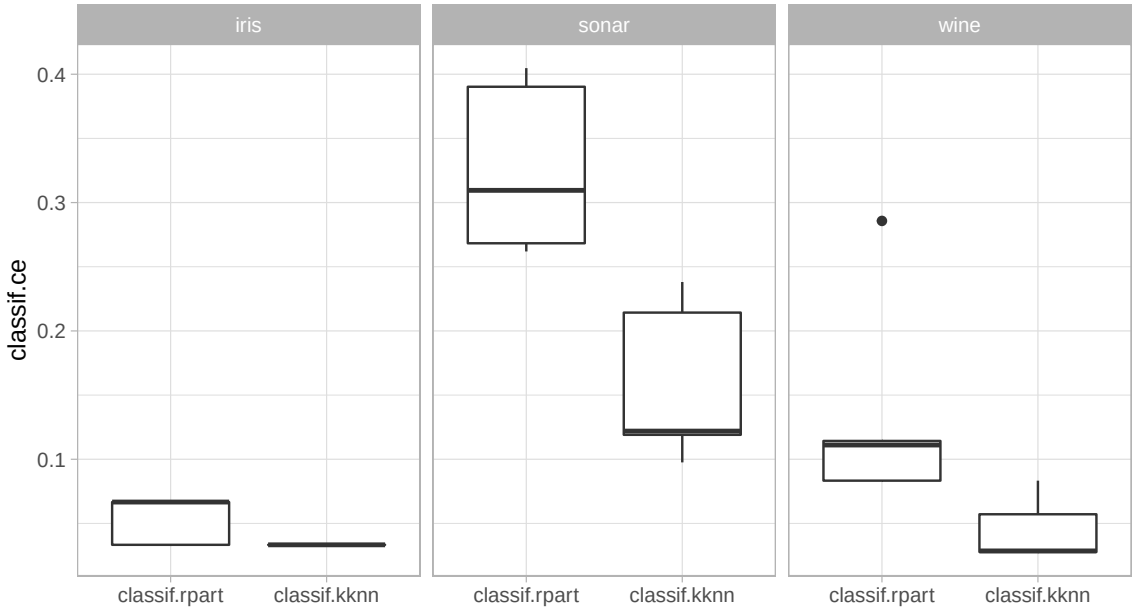
What exactly is a `BenchmarkResult` object?
Just like `Prediction` and `ResamplingResult`!

- Table representation using `as.data.table()`
- Active bindings and functions that make information easily accessible

BENCHMARK RESULT

The `mlr3viz` package contains `autoplot()` functions for many mlr3 objects

```
library(mlr3viz)
autoplot(bmr)
```



Control of Execution

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
- Possibilty to have a fallback to create predictions

How to get Help

HOW TO GET HELP

- Where to start?
  - Check these slides
  - **Check the mlr3book https://mlr3book.mlr-org.com**

HOW TO GET HELP

- Where to start?
  - Check these slides
  - **Check the mlr3book https://mlr3book.mlr-org.com**
- Get help for R6 objects?
  1. Find out what kind of R6 object you have:

```
class(bmr)
#> [1] "BenchmarkResult" "R6"
```

  2. Go to the corresponding help page:

```
?BenchmarkResult
```

  New: open the corresponding man page with

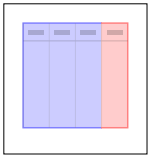
```
learner$help()
```

Outro

OVERVIEW

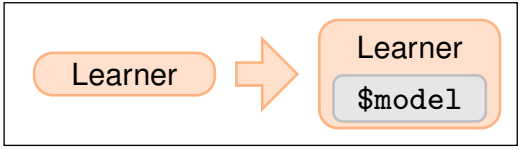
Ingredients:

Data



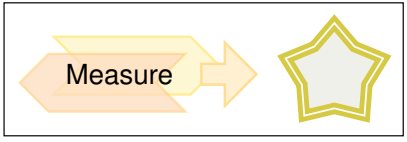
`TaskClassif`,
`TaskRegr`,
`tsk()`

Learning Algorithms



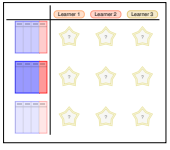
`lrn()` ⇒ `Learner`,
`$train()`,
`$predict()` ⇒ `Prediction`

Performance Evaluation



`rsmp()` ⇒ `Resampling`,
`msr()` ⇒ `Measure`,
`resample()` ⇒ `ResamplingResult`,
`$aggregate()`

Performance Comparison



`benchmark_grid()`,
`benchmark()` ⇒ `BenchmarkResult`

Introduction to `batchtools`

Supported Systems

- Torque/PBS based systems
- Sun Grid Engine / Oracle Grid Engine
- Load Sharing Facility (LSF)
- SLURM
- DockerSwarm

Other modes:

- Interactive: Jobs executed in current interactive **R** session
- Multicore: local multicore execution with spawned processes
- SSH: distributed computing on loosely connected machines which are accessible via SSH (makeshift cluster)

Links and references

https://github.com/mllg/batchtools

- Installation infos
- R documentation
- Vignettes
- Issue tracker
- Recent development version in git

Paper:

```
batchtools: Tools for R to work on batch systems.
The Journal of Open Source Software 2.10 (2017).
Lang, Michel, Bernd Bischl, and Dirk Surmann.
```

Create a registry

- Object used to access and exchange informations: file paths, job parameters, computational events, ...
- All information is stored in a single, portable directory
- Initialization of a new registry:

```
library(batchtools)
reg = makeRegistry(
  file.dir = "registry",  # accessible on all nodes
  seed = 1                # initial seed for first job
)
```

- `loadRegistry(dir)` to resume working with an existing registry

Define Jobs

`batchMap`:

- Like `lapply` or `mapply`
- $(x_1, x_2) \times (y_1, y_2) \to (f(x_1, y_1), f(x_2, y_2))$
- 10 jobs to calculate $1 + 9$, $2 + 8$, $\ldots$, $9 + 1$

```
map = function(i, j)  i + j
ids = batchMap(fun = map, i = 1:9, j = 9:1, reg = reg)
```

- Stores function on file system
- Creates jobs as rows in a **data.table**
- Parameters also serialized into the **data.table** for fast access
- All jobs get unique positive integers as IDs
- `reg =` can be omitted in most cases. See `?getDefaultRegistry`.

Subset Jobs

- Query job IDs by computational status: `find*` functions `findSubmitted`, `findRunning`, `findDone`, ...
- Query job IDs by parameters: `findJobs(pars)`

```
findJobs(j==1)
findNotSubmitted()
findDone()
```

- Set operations on `job.id` **data.tables**: `merge`
- **data.table** of `job.id`'s can be passed to basically all functions interacting with the batch system

Submit Jobs

- Creates **R** script files and job description files on the fly
- Resources can be provided as named list

```
# 1 hour maximal execution time, about 2 GB of RAM
res = list(walltime = 60*60, memory = 2000)

# ... and submit
submitJobs(resources = res)
```

- Submits all jobs per default
- Subsets of jobs can be providing as **data.table** or `vector`

```
submitJobs(ids = 1:5, ressources = res)
```

Supervise and Debug

- Quick overview of what is going on: `getStatus()`

```
## Status for 9 jobs at 2019-10-10 17:49:48:
##   Submitted    : 9 (100.0%)
##   -- Queued    : 0 (  0.0%)
##   -- Started   : 9 (100.0%)
##   ---- Running : 0 (  0.0%)
##   ---- Done    : 9 (100.0%)
##   ---- Error   : 0 (  0.0%)
##   ---- Expired : 0 (  0.0%)
```

- Display log files with a customizable pager (less, vi, ...): `showLog(findErrors()[1])`
- You can also `grepLogs(pattern)`
- Found a bug? `killJobs(findRunning())`
- Run a job in the current **R** session: `testJob(id)`

Collect Results

Reduce:

```
# combine in numeric vector
reduceResults(ids = findDone(), init = numeric(0),
  fun = function(aggr, job, res) c(aggr, res))
```

- Convenience wrappers around `reduceResults`: `reduceResults[DataTable|List]`

Simple Loading:

```
loadResult(id = 1)
```

More Configuration

Configuration file `~/.batchtools.conf.R`:

```
cluster.functions = makeClusterFunctionsSlurm("~/slurm_lmulrz.tmpl",
  clusters = "serial")
default.resources = list(walltime = 3600, memory = 1024,
  ntasks = 1)
debug = FALSE
max.concurrent.jobs = 999
```

Experiments in batchtools

Intended as abstraction for typical statistical tasks:

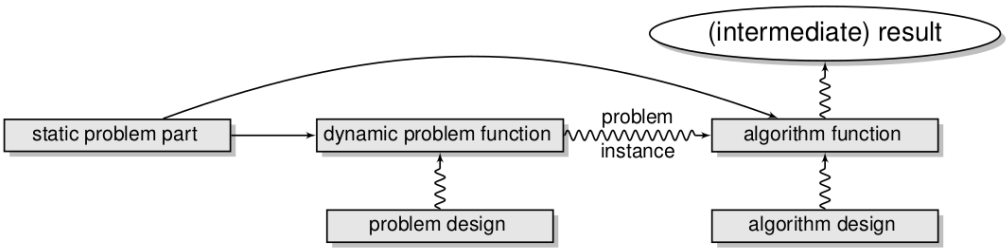
Applying algorithms on problems

- More aimed at the end user
- Convenient for simulation studies, comparison and benchmark experiments, sensitivity analysis, ...
- Workflow differs only in job definition

Scenarios:

- Compare machine learning algorithms on many data sets
- Compare one/many estimation procedure(s) on simulated data
- Compare optimizers on objective functions
- ...

Abstraction of Computer Experiments



- Problem definition split into static and dynamic part
  - Immutable **R** objects: matrix, data frames, ...
  - Arbitrary **R** function: transformations of static part, extraction of data from external sources, ...
- Parametrization through specifying experimental designs for both problems and algorithms
- Each step automatically seeded, random seeds stored in a database

Experiment definition steps

- Add problems to registry: `addProblem`
  - Efficient storage: Separation of static (`data`) and dynamic (`instance`) problem parts.
- Add algorithms to registry: `addAlgorithm`
  - Problem instance gets passed to algorithm
  - Can be connected with an experimental design (function parameters)
  - Return value will be saved on the file system
- Add experiments to registry: `addExperiments`
  - Experiment: problem instance + algorithm + algorithm parameters
  - Job: Experiment + replication number

A simple Example

```
reg = makeExperimentRegistry("test_reg")
addProblem(name = "p1", data = 1, seed = 1,
  fun = function(data, job) runif(data))
addAlgorithm(name = "a1",
  fun = function(job, data, instance) 2 * instance)
addAlgorithm(name = "a2",
  fun = function(job, data, instance) data + instance)
addExperiments(repls = 2)
submitJobs()
res = reduceResultsDataTable()
getJobPars()[res]

##    job.id problem prob.pars algorithm algo.pars result
## 1:      1      p1    <list>        a1    <list>  0.531
## 2:      2      p1    <list>        a1    <list>   0.37
## 3:      3      p1    <list>        a2    <list>   1.27
## 4:      4      p1    <list>        a2    <list>   1.18
```

Summary

- Reproducibility: Every computation is seeded, seeds are stored in a **data.table**
- Portability: Data, algorithms, results and job information reside in a single directory
- Extensibility: Add more problems or algorithms, try different parameters or increase the replication numbers at any computational state
- Exchangeability: Share your file directory to allow others to extend your study with their data sets and algorithms
- Greatly simplifies the work with batch systems
- Interactively control batch systems from within R (no shell required)
- Do reproducible research
- Exchange code and results with others

BATCHTOOLS DEMO

Tuning Machine Learning Algorithms with `mlr3`

TUNING

- Behavior of most methods depends on *hyperparameters*

TUNING

- Behavior of most methods depends on *hyperparameters*
- We want to choose them so our algorithm performs well

TUNING

- Behavior of most methods depends on *hyperparameters*
- We want to choose them so our algorithm performs well
- Good hyperparameters are data-dependent

TUNING

- Behavior of most methods depends on *hyperparameters*
- We want to choose them so our algorithm performs well
- Good hyperparameters are data-dependent

$\Rightarrow$ We do *black box optimization* (“Try stuff and see what works”)

TUNING

- Behavior of most methods depends on *hyperparameters*
- We want to choose them so our algorithm performs well
- Good hyperparameters are data-dependent

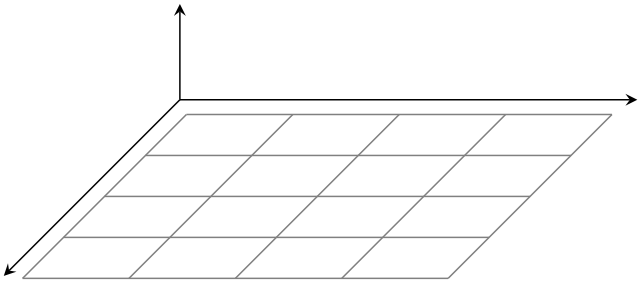
$\Rightarrow$ We do *black box optimization* (“Try stuff and see what works”)

Tuning toolbox for `mlr3`:

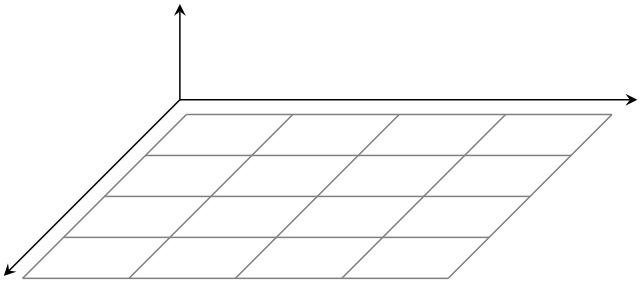
```
library("bbotk")
library("mlr3tuning")
```

Tuning

TUNING

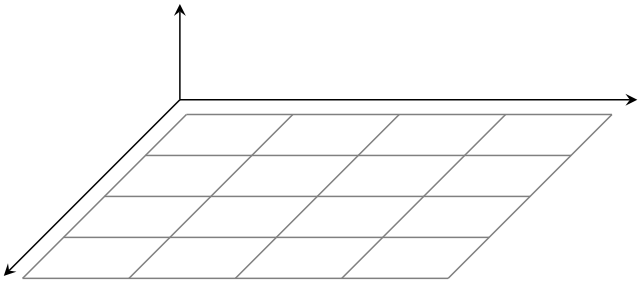


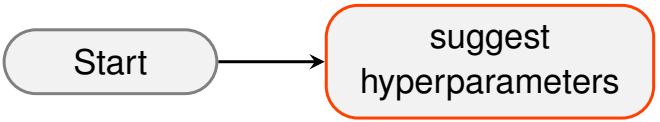
TUNING



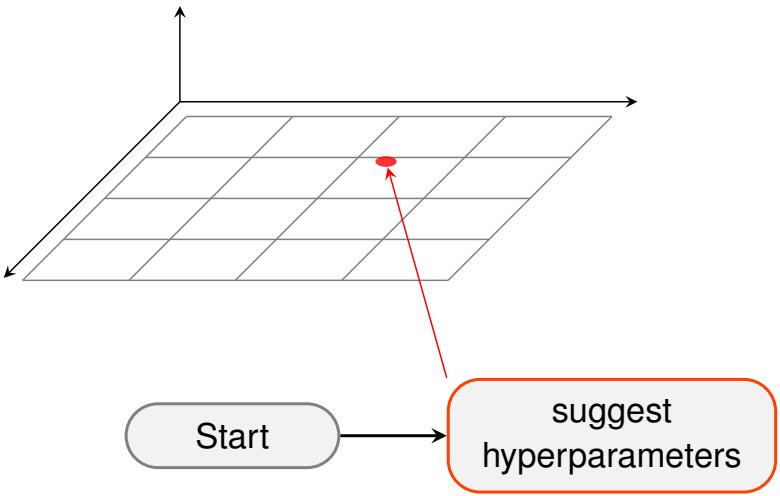
Start

TUNING

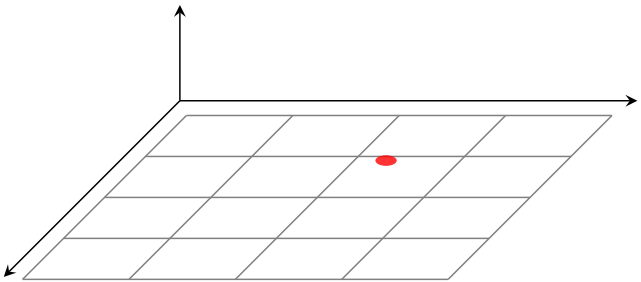


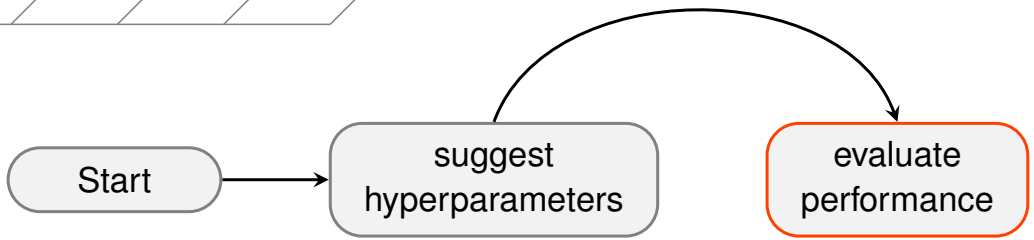


TUNING

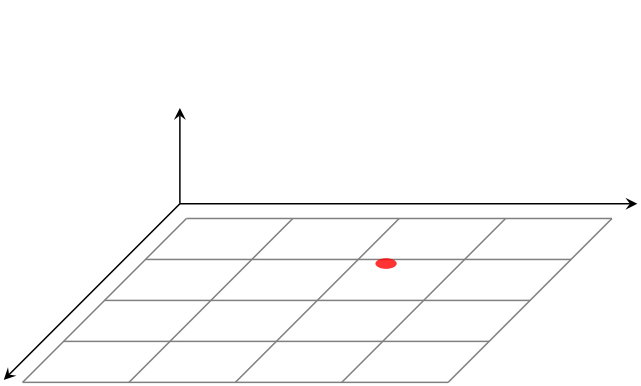


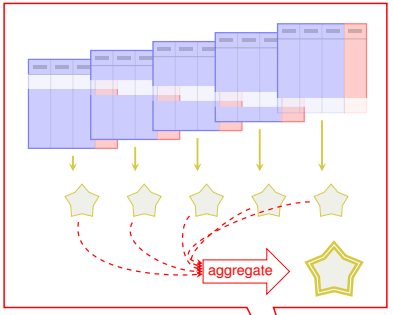
TUNING





TUNING



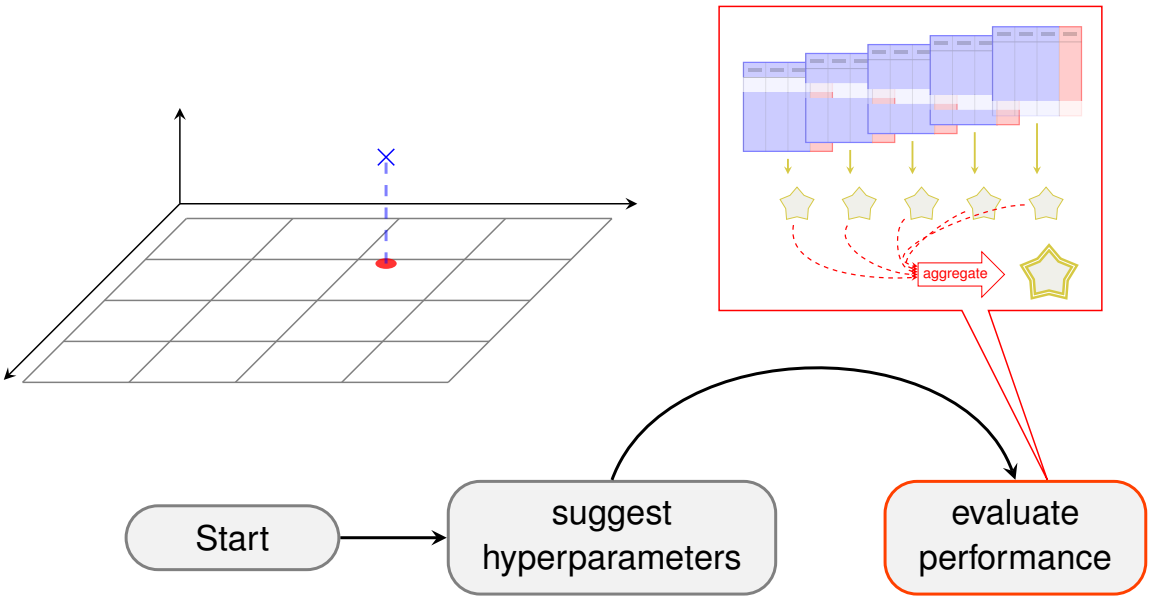


Start

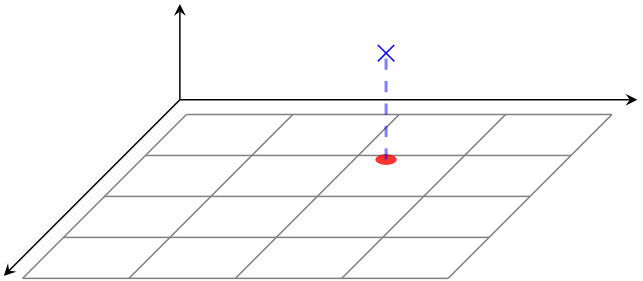
suggest hyperparameters

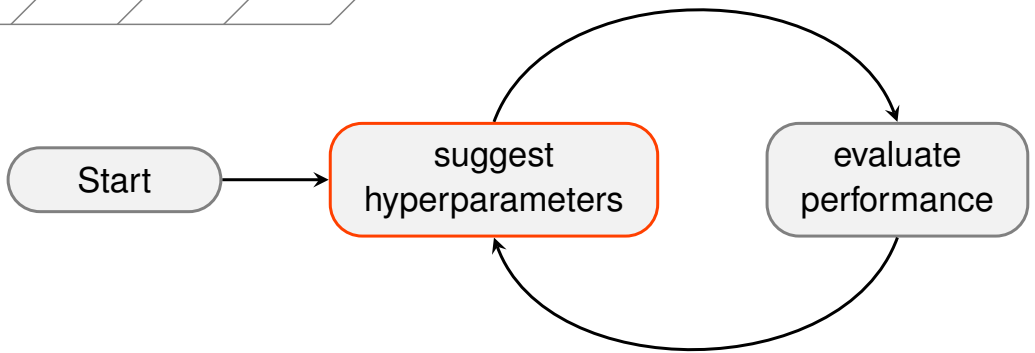
evaluate performance

TUNING

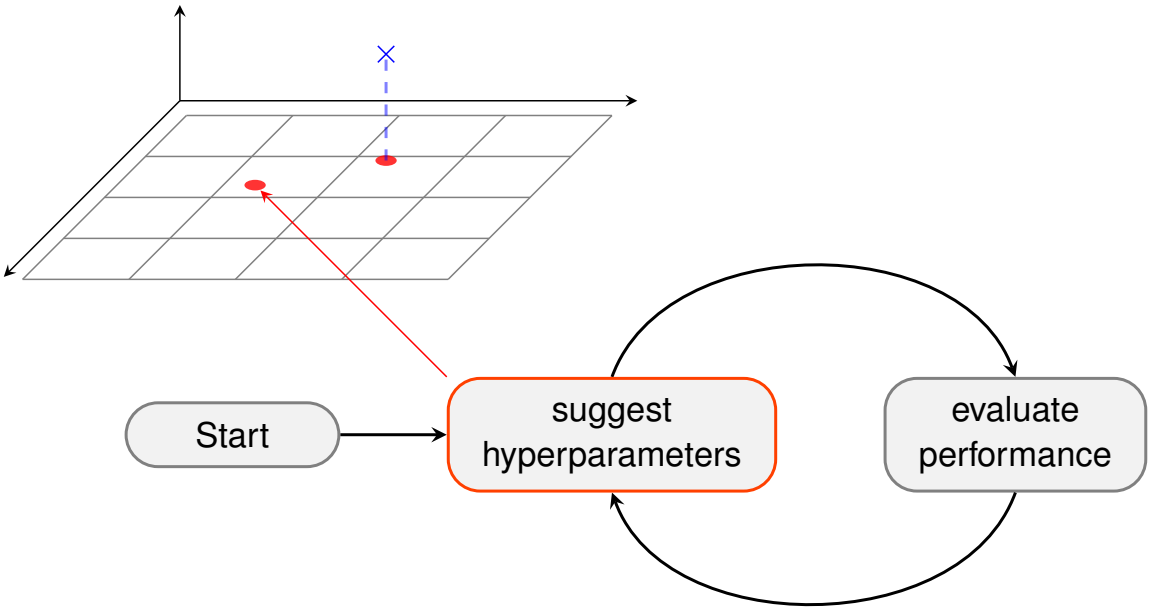

aggregate
Start
suggest hyperparameters
evaluate performance

TUNING

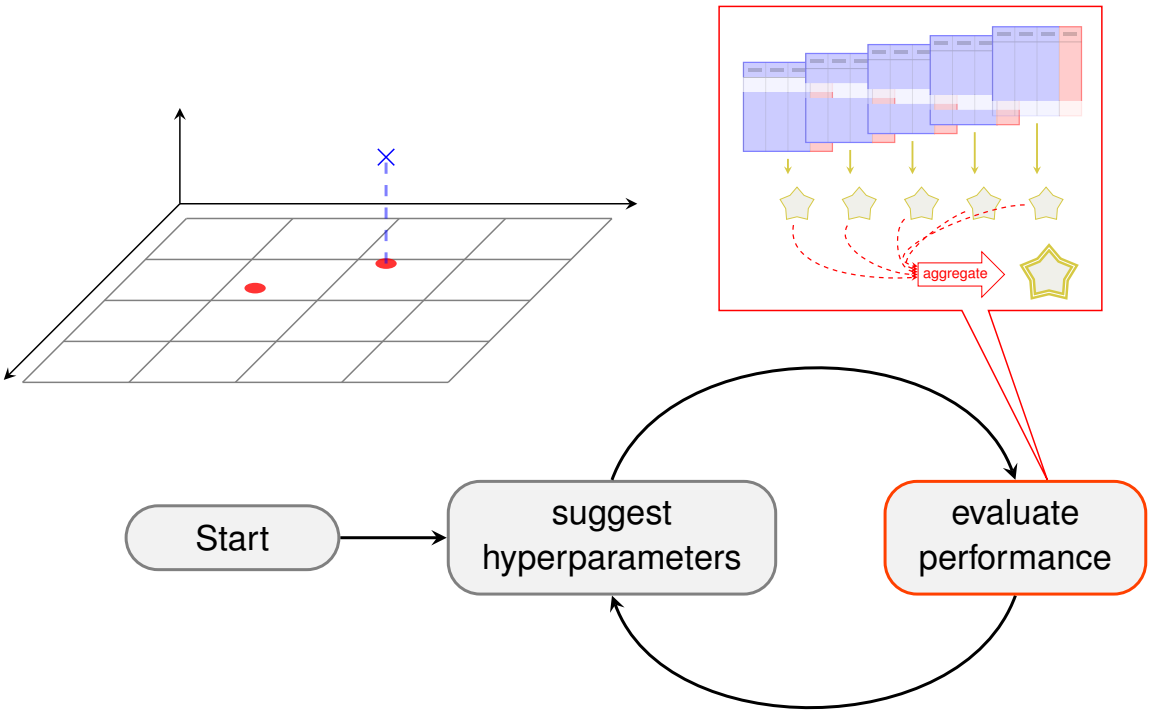




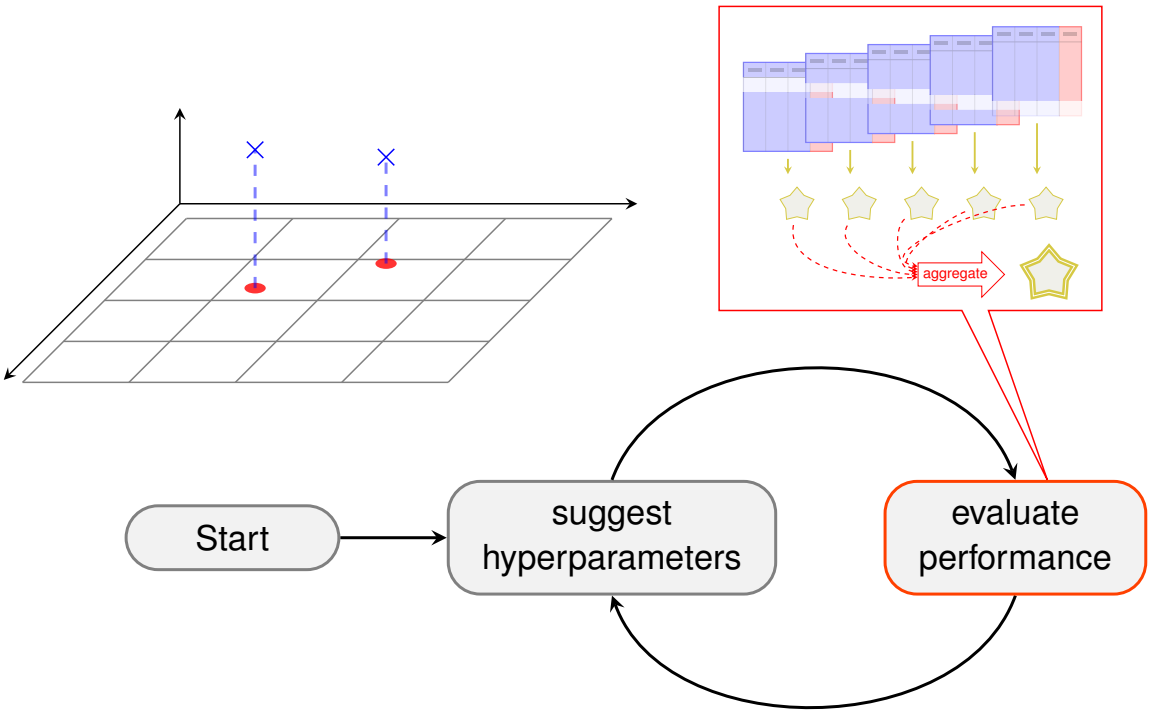
TUNING



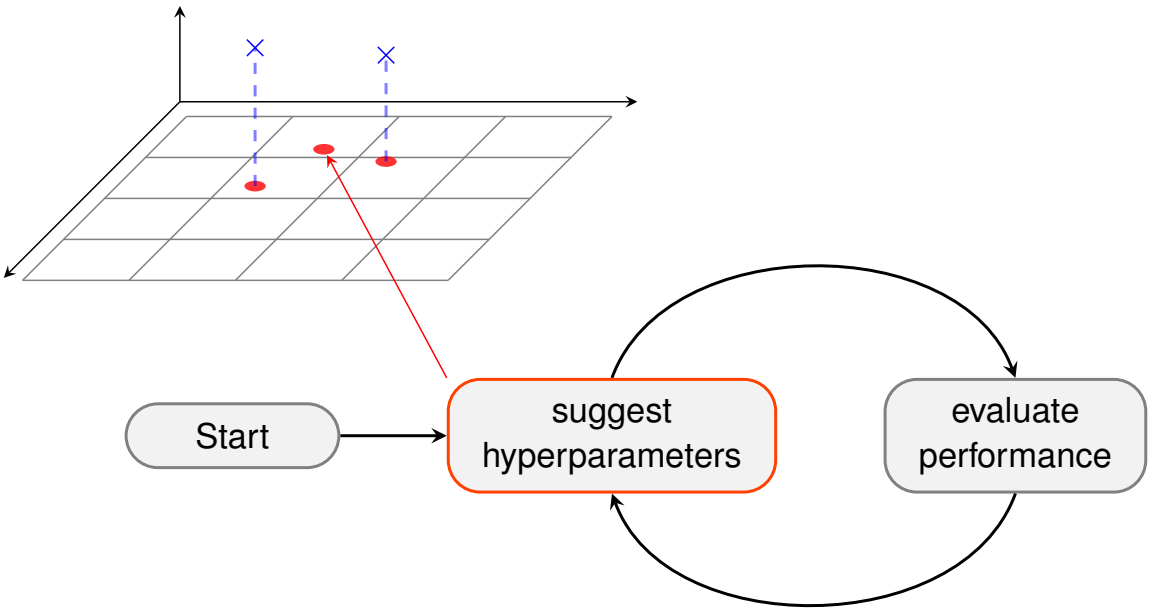
TUNING


aggregate
Start
suggest hyperparameters
evaluate performance

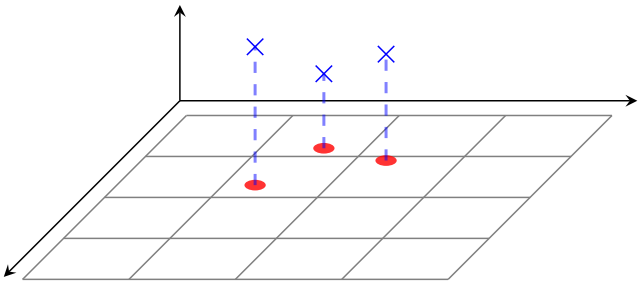
TUNING

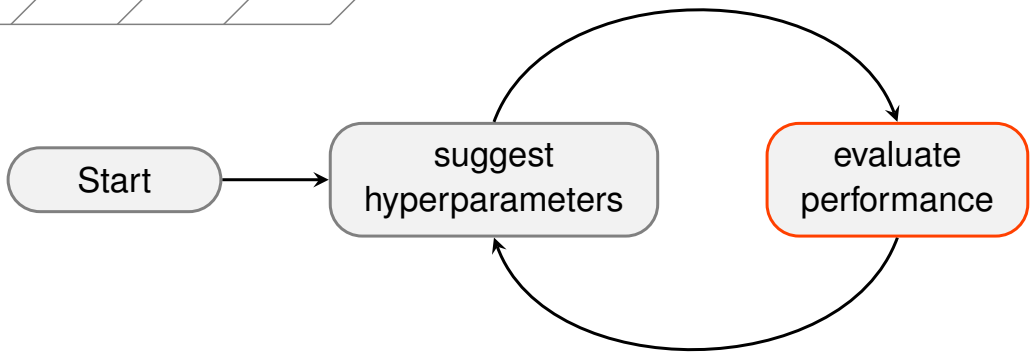

aggregate
Start
suggest hyperparameters
evaluate performance

TUNING

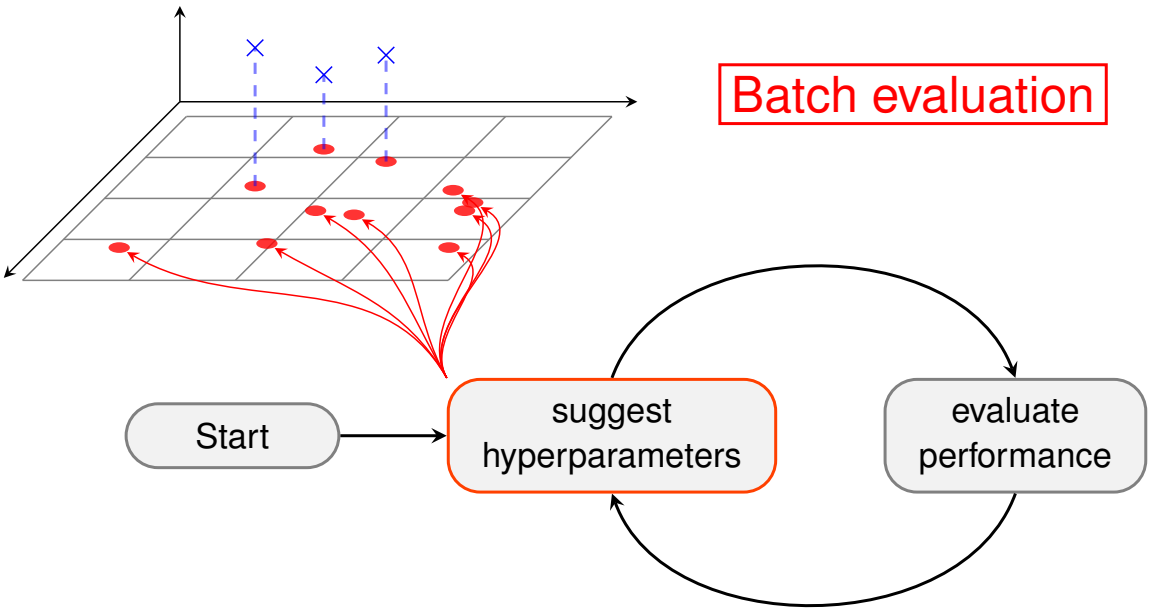


TUNING

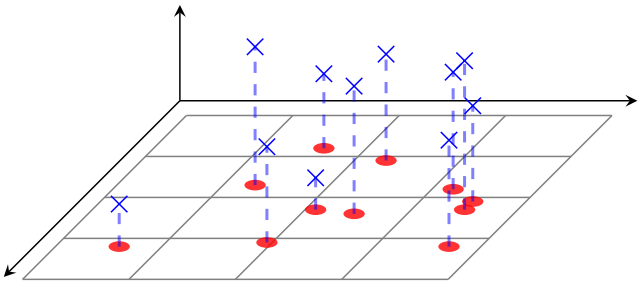




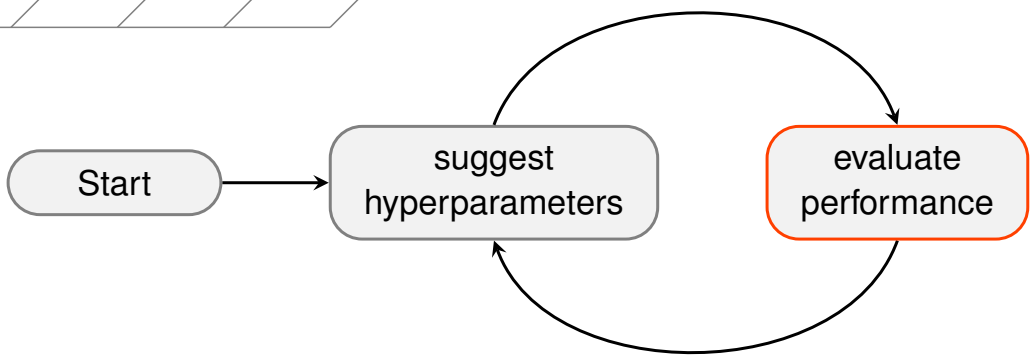
TUNING


Batch evaluation
Start
suggest hyperparameters
evaluate performance

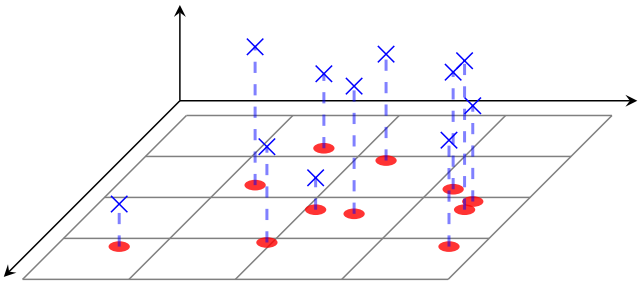
TUNING

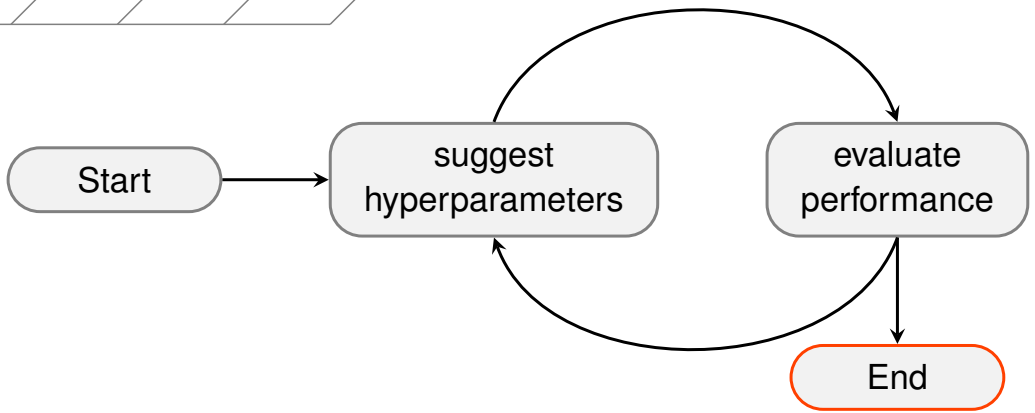


Batch evaluation

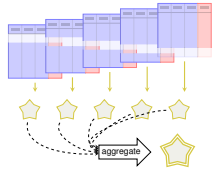


TUNING

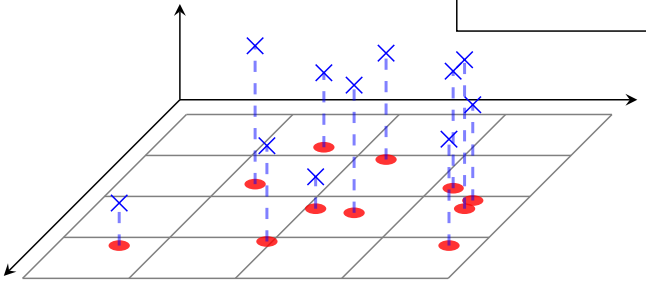


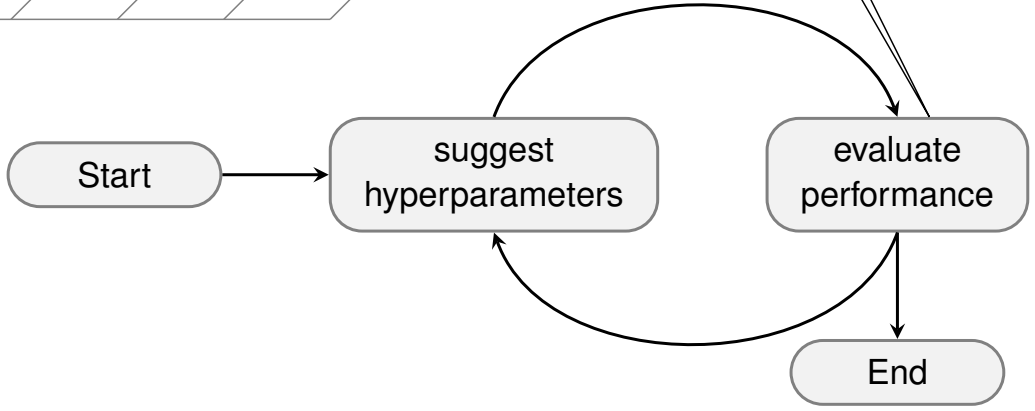


TUNING

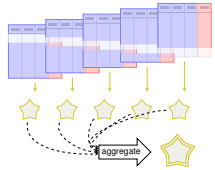


`Learner` (`lrn()`)
`Task`
`Resampling` (`rsmp()`)
`Measure` (`msr()`)

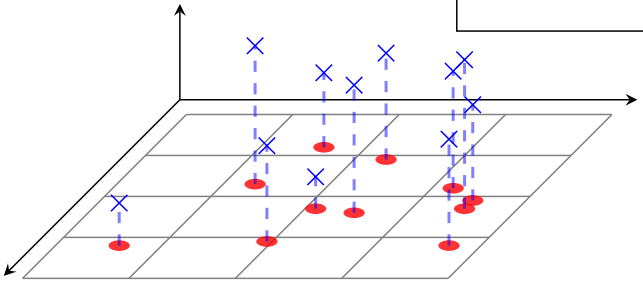




TUNING



`Learner` (`lrn()`)
`Task`
`Resampling` (`rsmp()`)
`Measure` (`msr()`)



Start

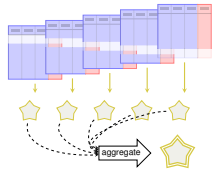
suggest hyperparameters

evaluate performance

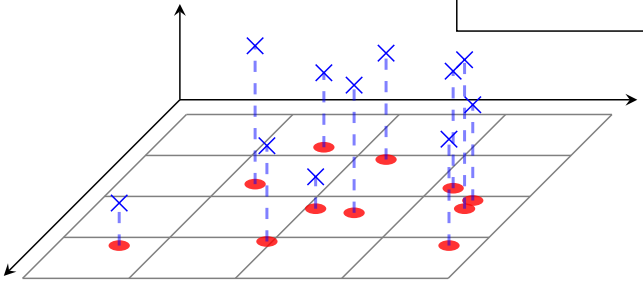
search space
`Tuner`

End

TUNING



`Learner` (`lrn()`)
`Task`
`Resampling` (`rsmp()`)
`Measure` (`msr()`)



Start

suggest hyperparameters

evaluate performance

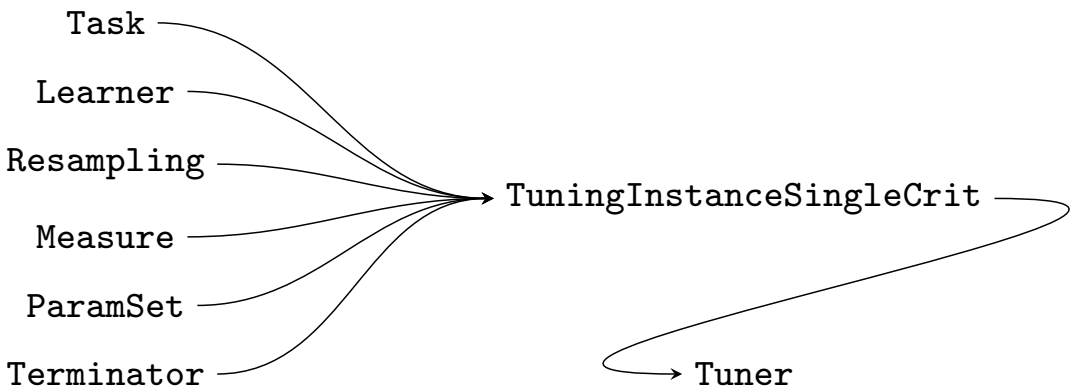
search space
`Tuner`

`Terminator` (when to finish)

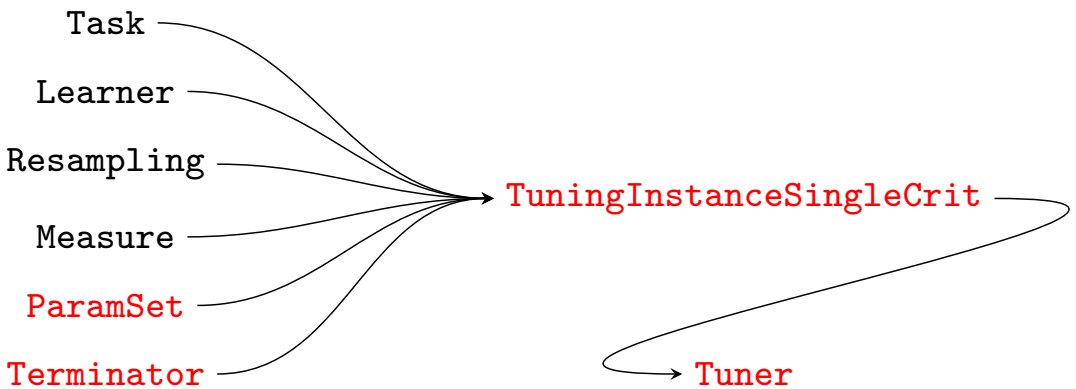
End

Tuning in mlr3

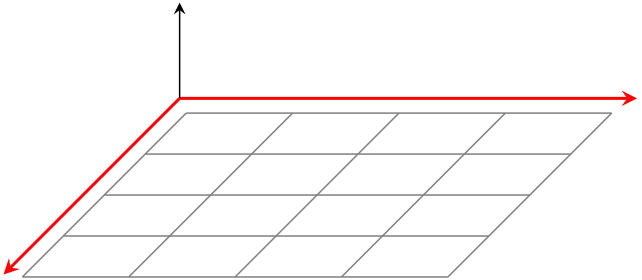
OBJECTS IN TUNING



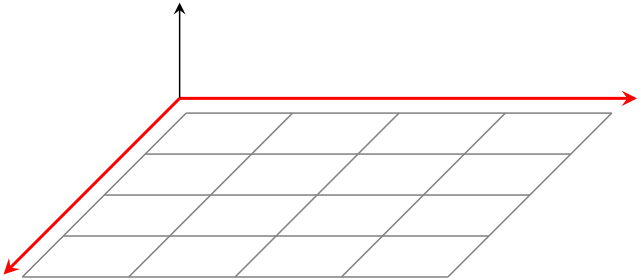
OBJECTS IN TUNING



SEARCH SPACE

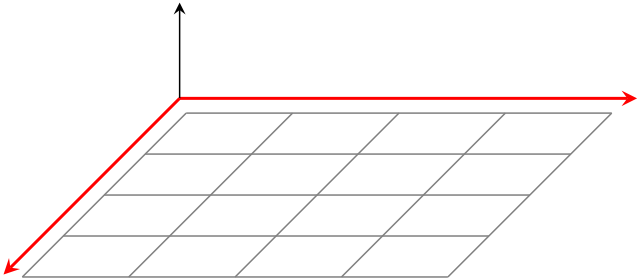


SEARCH SPACE



```
ParamSet$new(list(param1, param2, ...))
```

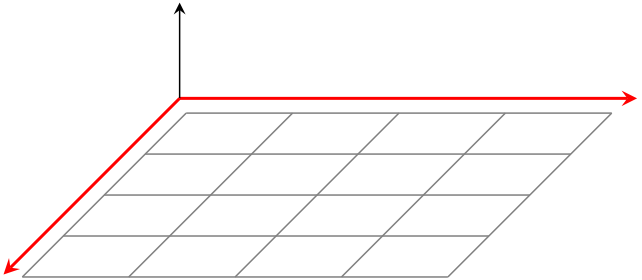
SEARCH SPACE



```
ParamSet$new(list(param1, param2, ...))
```

Numerical parameter	`ParamDbl$new(id, lower, upper)`
Integer parameter	`ParamInt$new(id, lower, upper)`
Discrete parameter	`ParamFct$new(id, levels)`
Logical parameter	`ParamLgl$new(id)`
Untyped parameter	`ParamUty$new(id)`

SEARCH SPACE



```
ParamSet$new(list(param1, param2, ...))
```

Numerical parameter	`ParamDbl$new(id, lower, upper)`
Integer parameter	`ParamInt$new(id, lower, upper)`
Discrete parameter	`ParamFct$new(id, levels)`
Logical parameter	`ParamLgl$new(id)`
Untyped parameter	`ParamUty$new(id)`

```
library("paradox")
searchspace_knn = ParamSet$new(list(
  ParamInt$new("k", 1, 20)
))
```

TERMINATION

- Tuning needs a *termination condition*: when to finish

TERMINATION

- Tuning needs a *termination condition*: when to finish
- `Terminator` class

TERMINATION

- Tuning needs a *termination condition*: when to finish
- `Terminator` class
- `mlr_terminators` dictionary, `trm()` short form

TERMINATION

- Tuning needs a *termination condition*: when to finish
- `Terminator` class
- `mlr_terminators` dictionary, `trm()` short form
- ```
  as.data.table(mlr_terminators)
  #>                 key
  #> 1:       clock_time
  #> 2:            combo
  #> 3:            evals
  #> 4:             none
  #> 5:     perf_reached
  #> 6:         run_time
  #> 7:       stagnation
  #> 8: stagnation_batch
  ```
  ```

TERMINATION

- Tuning needs a *termination condition*: when to finish
- `Terminator` class
- `mlr_terminators` dictionary, `trm()` short form

```
as.data.table(mlr_terminators)
#>                 key
#> 1:       clock_time
#> 2:            combo
#> 3:            evals
#> 4:             none
#> 5:     perf_reached
#> 6:         run_time
#> 7:       stagnation
#> 8: stagnation_batch
```

```
trm("evals", n_evals = 20)
#> <TerminatorEvals>
#> * Parameters: n_evals=20
```

TUNING METHOD

- need to choose a *tuning method*

TUNING METHOD

- need to choose a *tuning method*
- `Tuner` class

TUNING METHOD

- need to choose a *tuning method*
- `Tuner` class
- `mlr_tuners` dictionary, `tnr()` short form

TUNING METHOD

- need to choose a *tuning method*
- `Tuner` class
- `mlr_tuners` dictionary, `tnr()` short form
- ```
  as.data.table(mlr_tuners)
  #>              key
  #> 1: design_points
  #> 2:         gensa
  #> 3:   grid_search
  #> 4:        nloptr
  #> 5: random_search
  ```
  ```

TUNING METHOD

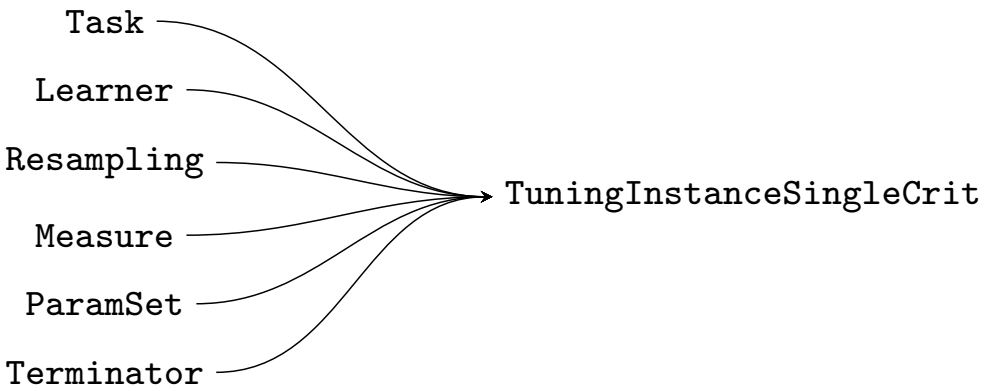
- load `Tuner` with `tnr()`, set parameters

TUNING METHOD

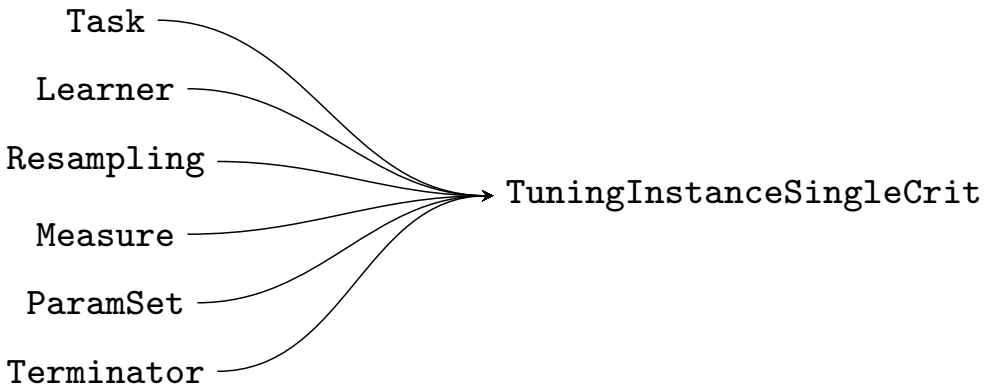
- load `Tuner` with `tnr()`, set parameters
- ```
  gsearch = tnr("grid_search", resolution = 3)

  print(gsearch)
  #> <TunerGridSearch>
  #> * Parameters: resolution=3, batch_size=1
  #> * Parameter classes: ParamLgl, ParamInt, ParamDbl, ParamFct
  #> * Properties: dependencies, single-crit, multi-crit
  #> * Packages: -
  ```
  ```

TUNING METHOD

- load `Tuner` with `tnr()`, set parameters
- ```
  gsearch = tnr("grid_search", resolution = 3)

  print(gsearch)
  #> <TunerGridSearch>
  #> * Parameters: resolution=3, batch_size=1
  #> * Parameter classes: ParamLgl, ParamInt, ParamDbl, ParamFct
  #> * Properties: dependencies, single-crit, multi-crit
  #> * Packages: -
  ```
- common parameter `batch_size` for parallelization
  ```

CALLING THE TUNER



CALLING THE TUNER



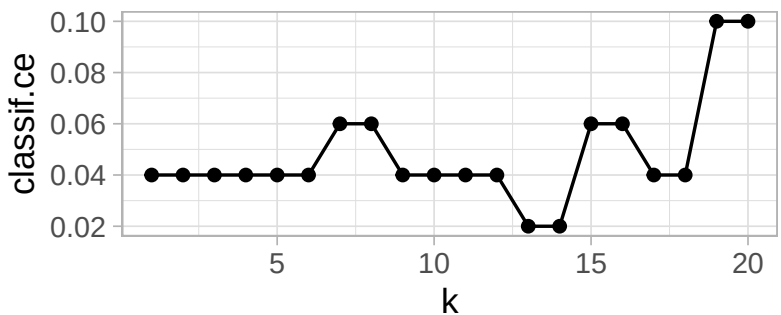

```
inst = TuningInstanceSingleCrit$new(
  tsk("iris"), lrn("classif.kknn", kernel="rectangular"),
  rsmp("holdout"), msr("classif.ce"),
  searchspace_knn, trm("none")
)
```

CALLING THE TUNER

```
gsearch$optimize(inst)

#> INFO  [08:23:19.807] Starting to optimize 1 parameter(s) with '<Opti
#> INFO  [08:23:19.853] Evaluating 1 configuration(s)
#> INFO  [08:23:21.138] Result of batch 1:
#> INFO  [08:23:21.140]   k classif.ce                                  u
#> INFO  [08:23:21.140]  10       0.04 3a0f7bd0-61c7-42a6-83e0-
d2a63d5b5efb
#> INFO  [08:23:21.142] Evaluating 1 configuration(s)
#> INFO  [08:23:21.230] Result of batch 2:
#> INFO  [08:23:21.232]  k classif.ce                                 ul
#> INFO  [08:23:21.232]  1       0.06 33732d74-49fb-41e0-9302-
7ff287db2e82
#> INFO  [08:23:21.234] Evaluating 1 configuration(s)
#> INFO  [08:23:21.300] Result of batch 3:
#> INFO  [08:23:21.302]   k classif.ce                                  u
#> INFO  [08:23:21.302]  20       0.08 bd9d8935-80f5-4caa-a236-
ec0e3f967f13
#> INFO  [08:23:21.310] Finished optimizing after 3 evaluation(s)
#> INFO  [08:23:21.311] Result:
#> INFO  [08:23:21.313]   k learner_param_vals  x_domain classif.ce
#> INFO  [08:23:21.313]  10          <list[2]> <list[1]>       0.04
```

TUNING RESULTS

```
gsearch = tnr("grid_search", resolution = 20)

inst = TuningInstanceSingleCrit$new(
  tsk("iris"), lrn("classif.kknn", kernel="rectangular"), rsmp("holdout"),
  msr("classif.ce"),searchspace_knn, trm("none"))

gsearch$optimize(inst)

#>     k learner_param_vals  x_domain classif.ce
#> 1: 14          <list[2]> <list[1]>       0.02

ggplot(inst$archive$data(),
  aes(x = k, y = classif.ce)) + geom_line() + geom_point()
```



RECAP

```
inst = TuningInstanceSingleCrit$new(
  tsk("iris"), lrn("classif.kknn", kernel="rectangular"),
  rsmp("holdout"), msr("classif.ce"),
  searchspace_knn, trm("evals", n_evals = 2)
)

gsearch = tnr("grid_search", resolution = 3)

gsearch$optimize(inst)

#>    k learner_param_vals  x_domain classif.ce
#> 1: 1          <list[2]> <list[1]>       0.06
```