Introduction to mlr3 for Machine Learning in R

Poo Kuan Hoong, Ph.D

@kuanhoong

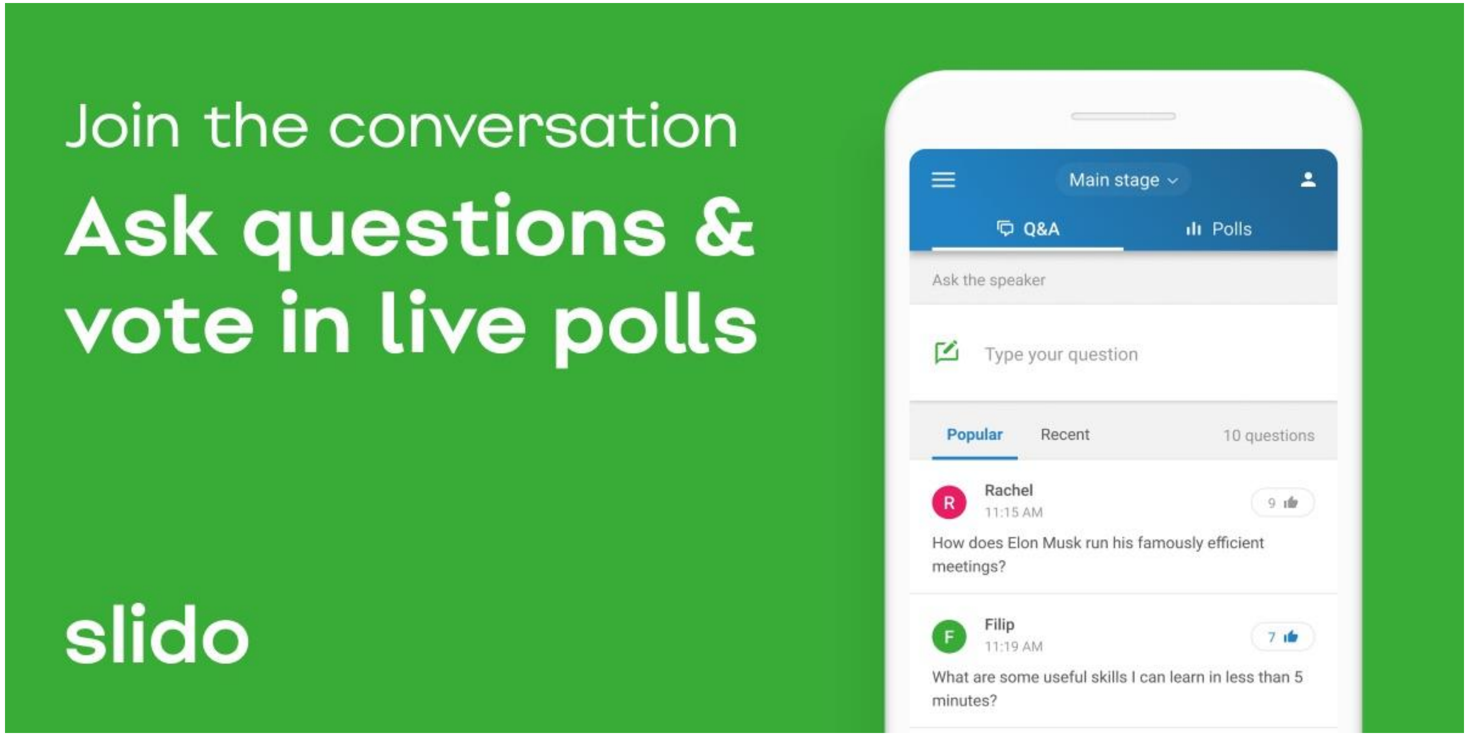
Slides

https://bit.ly/mlr3-slides

Questions

Slido: https://bit.ly/mlr3-slido

Event Code: **3287901**



About Me

Poo Kuan Hoong, Ph.D (Linkedin)

- Lead Data Scientist, BAT
- Google Developer Expert (GDE) in Machine Learning
- TensorFlow & Deep Learning Malaysia
- Malaysia R User Group
- Malaysia R Ladies
- AI/ML & Data Talks Podcast

Agenda

- Introduction to machine learning
- Overview of mlr3 package
- Getting started with mlr3
  - Creating a task
  - Creating a learner
  - Training a model
  - Making predictions
- Evaluating machine learning models
- Demo
- Conclusion

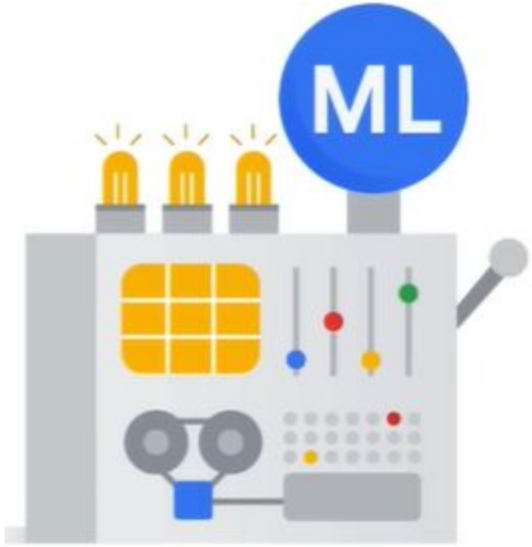
In this webinar...

- Participants should have basic knowledge about R programming, Object Oriented programming and Machine Learning
- The webinar will cover the introduction of mlr3 and a short demo of the usage mlr3 for machine learning.

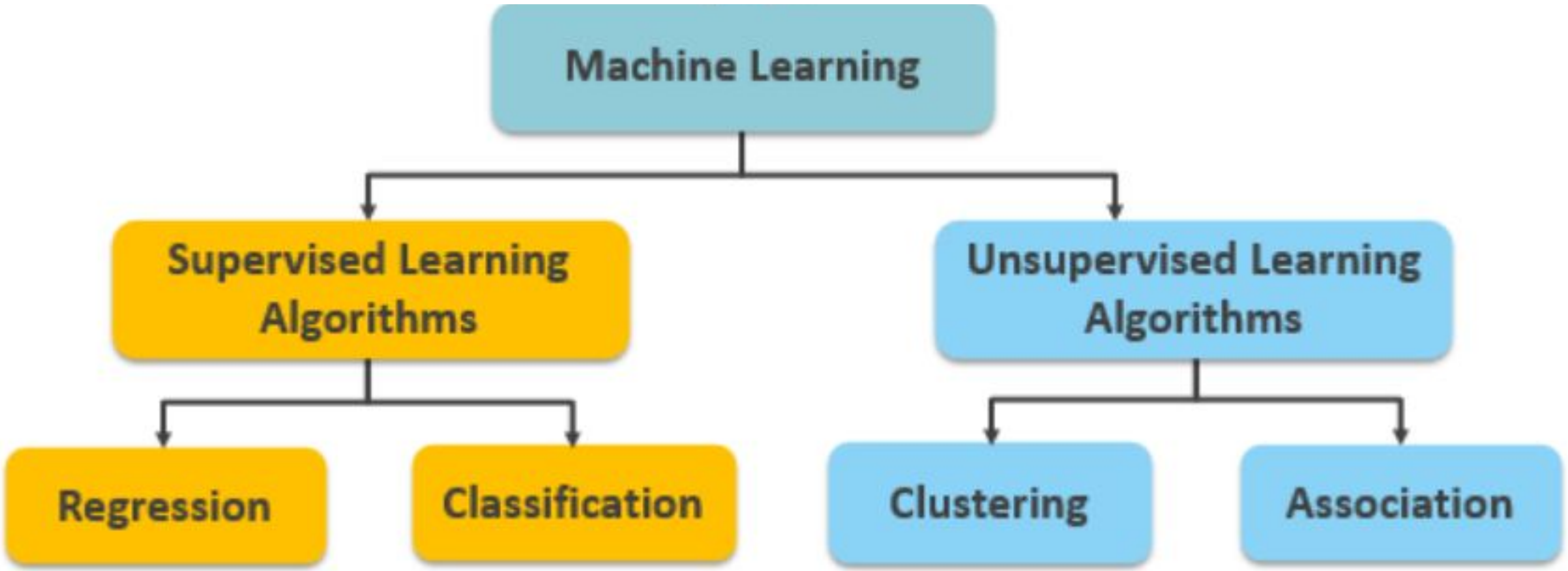
Introduction to Machine Learning

Introduction to Machine Learning

Machine learning is a type of artificial intelligence that allows computers to **learn without being explicitly programmed**. This means that computers can learn to perform tasks by **analyzing data** and **identifying patterns**.



Supervised vs Unsupervised Learning



Supervised Learning

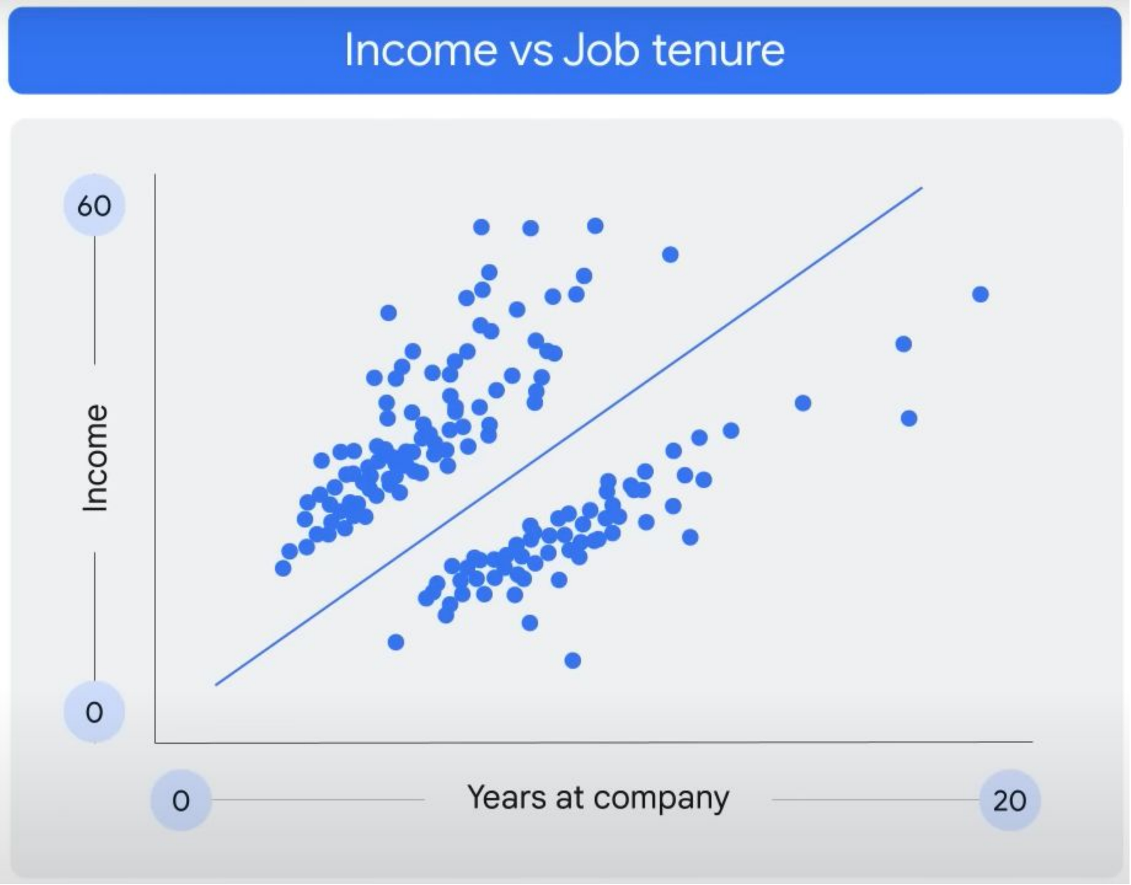
Supervised learning implies the data is **already labelled**.

In supervised learning we are learning from past examples to predict future values.

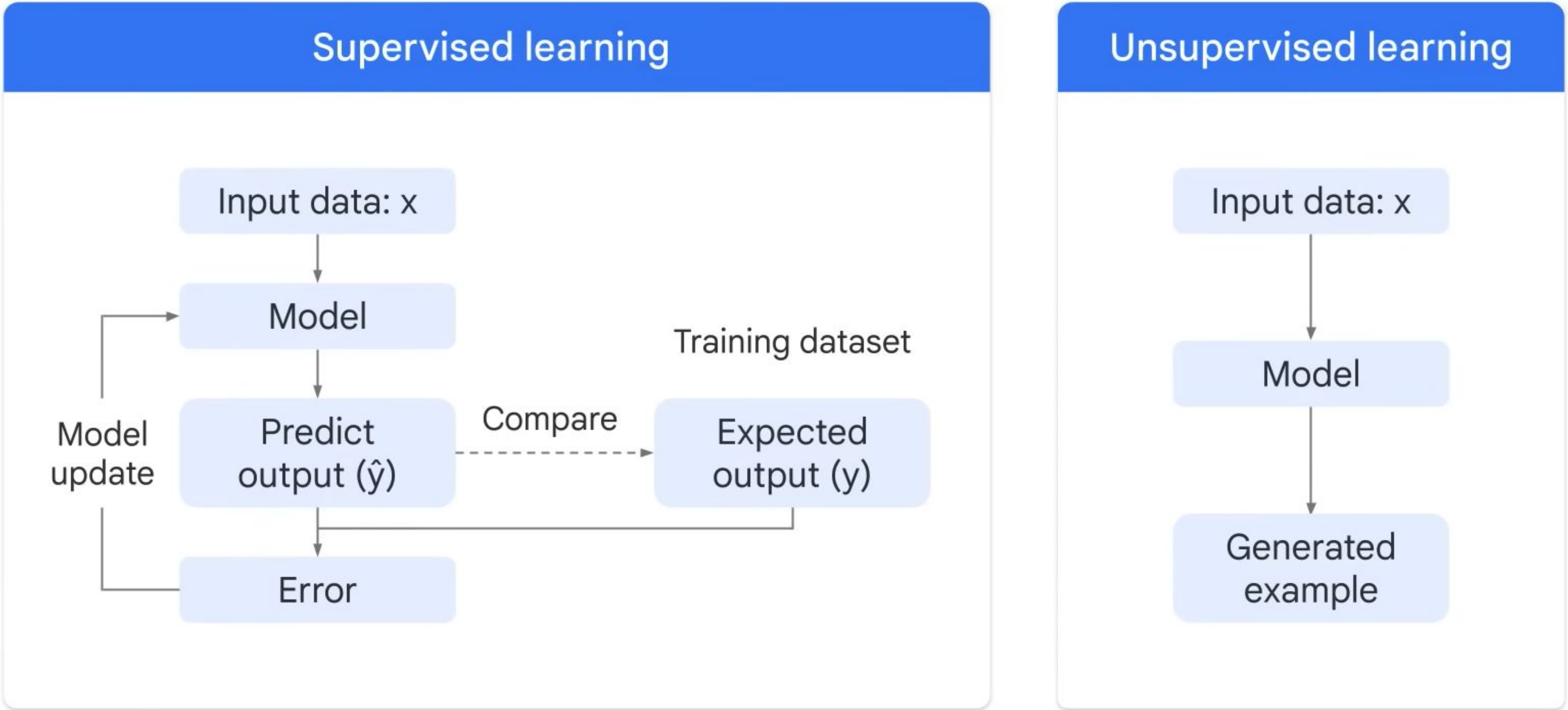
Unsupervised Learning

Unsupervised learning implies the data is **not labelled**.

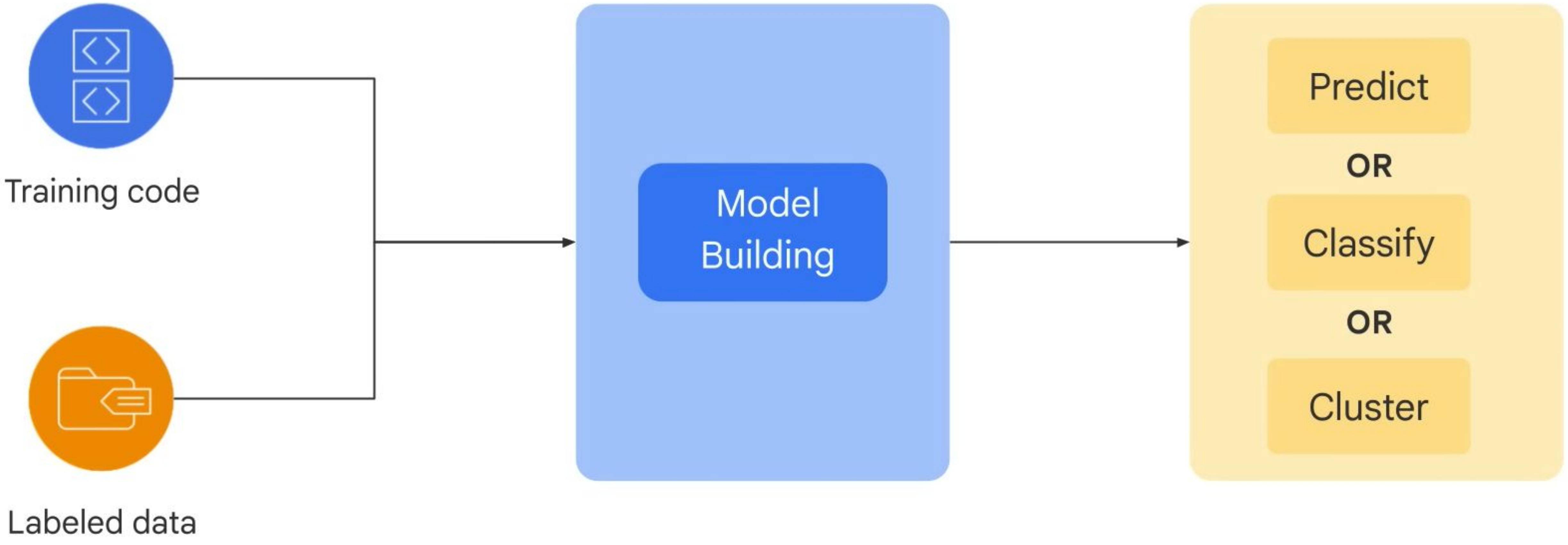
Unsupervised problems are all about looking at the raw data, and seeing if it naturally falls into groups.

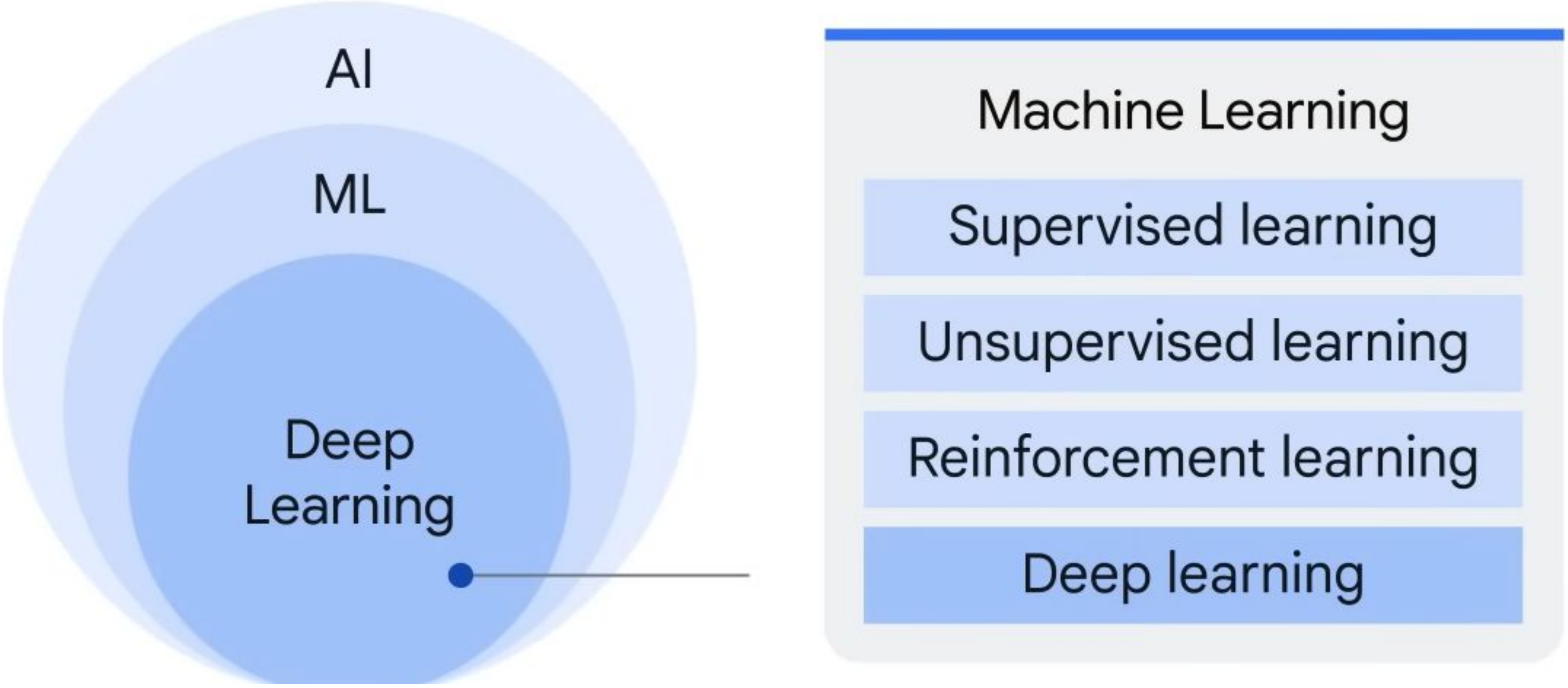


Example Model: Clustering
Is this employee on the “fast-track” or not?

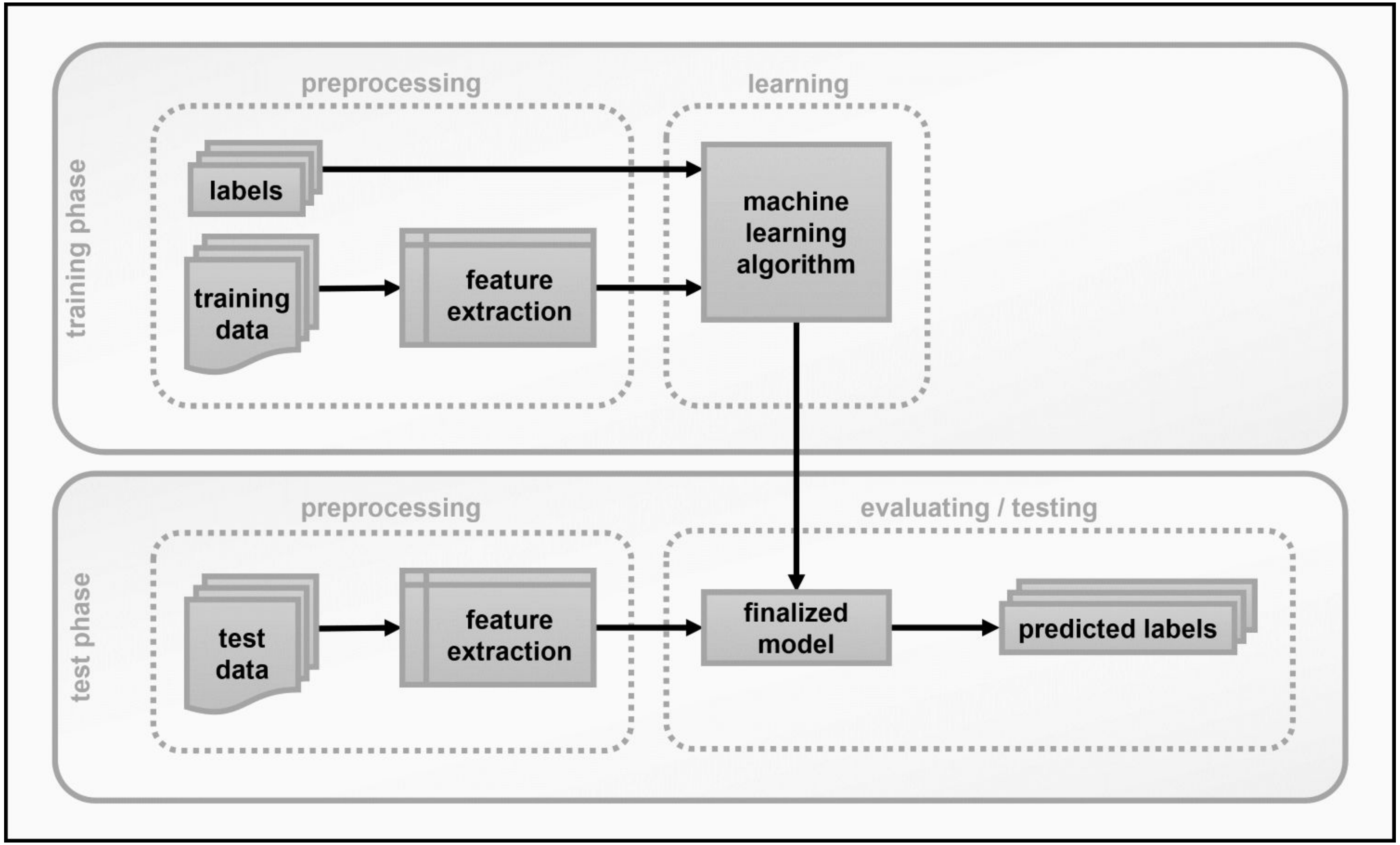

Supervised learning
Input data: x
Model
Model update
Predict output (ŷ)
Compare
Training dataset
Expected output (y)
Error
Unsupervised learning
Input data: x
Model
Generated example

Supervised vs Unsupervised Learning




AI
ML
Deep
Learning
Machine Learning
Supervised learning
Unsupervised learning
Reinforcement learning
Deep learning

Machine Learning Workflow



Machine Learning 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)
```

Machine Learning in R - mlr3

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

Ingredients:

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

R6

mlr3 - R6

`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
```

mlr3 - 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
```

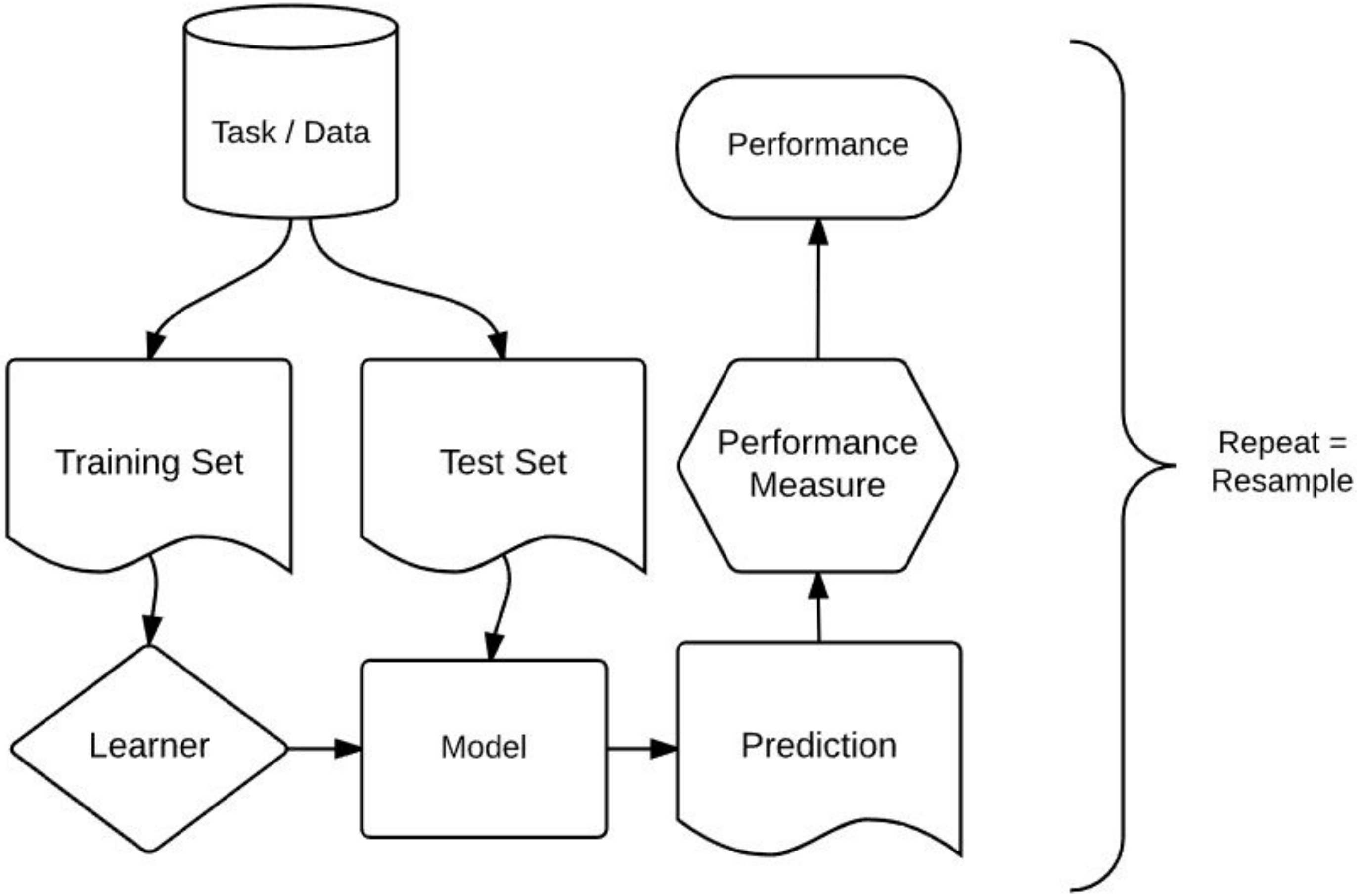
mlr3 philosophy

- 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
- 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,...`)

Machine Learning Workflow

mlr3 workflow

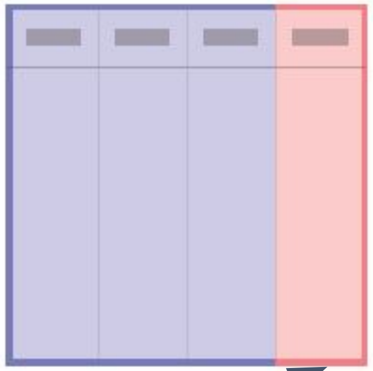
1. Load data
2. Split data into training and test sets
3. Create a task
4. Choose a learner
5. Train
6. Predict
7. Assess
8. Interpret



Data

Data

- Tabular data
- Features
- Target/outcome to predict
  - discrete for classification
  - continuous for regression
    - 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** **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 = )
```

Dictionaries

Dictionaries

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

- `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 15 stored values
#> Keys: boston_housing, breast_cancer, faithful,
#>   german_credit, iris, lung, mtcars, pima, precip, rats,
#>   sonar, spam, unemployment, 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:

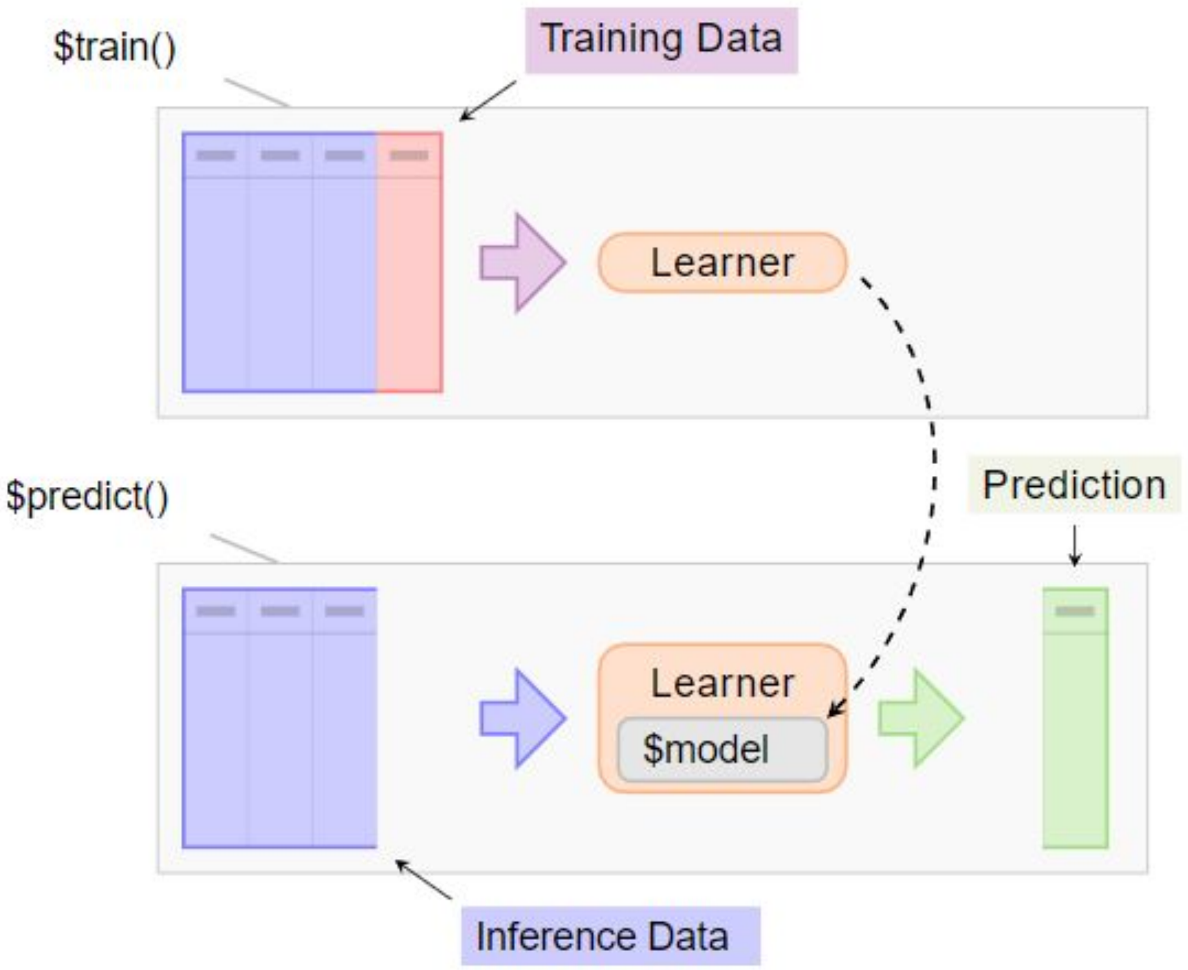
```
mlr_learners_table = as.data.table(mlr_learners)

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



Learner

- Each learner provides the following meta-data:
  - **`$feature_types`:** the type of features the learner can deal with.
  - **`$packages`:** the packages required to train a model with this learner and make predictions.
  - **`$properties`:** additional properties and capabilities. For example, a learner has the property “missings” if it is able to handle missing feature values, and “importance” if it computes and allows to extract data on the relative importance of the features.
  - **`$predict_types`:** possible prediction types. For example, a regression learner can predict numerical values (“response”) and may be able to predict the standard error of a prediction (“se”).
  - **`$param_set`:** the set of hyperparameters.

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`

```
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< 2.5 50   0 setosa (1.00 0.00 0.00) *
#>   3) Petal.Length>=2.5 100  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
```

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

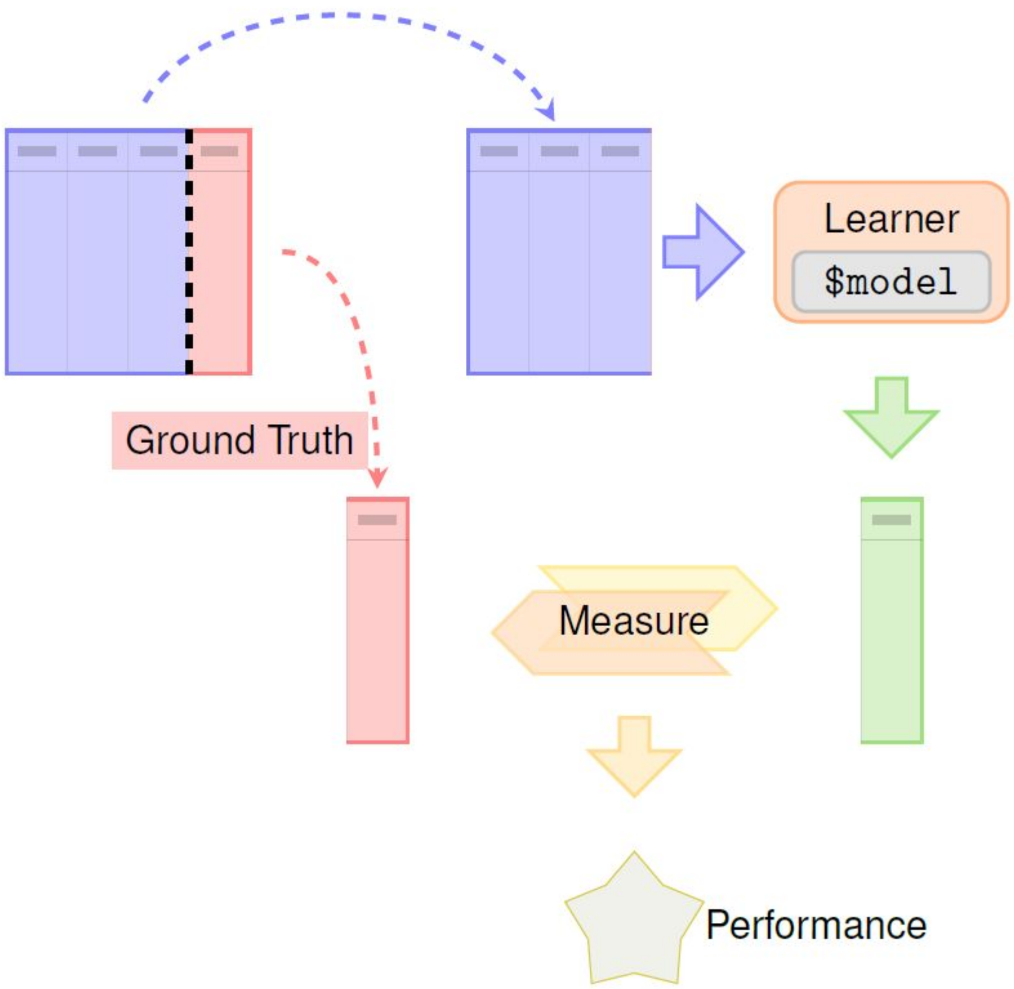
```
as.data.table(prediction)
#>     row_id truth   response
#> 1:       1  <NA>     setosa
#> 2:       2  <NA> versicolor
```

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

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

Performance

Performance Evaluation



Performance Evaluation

- 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

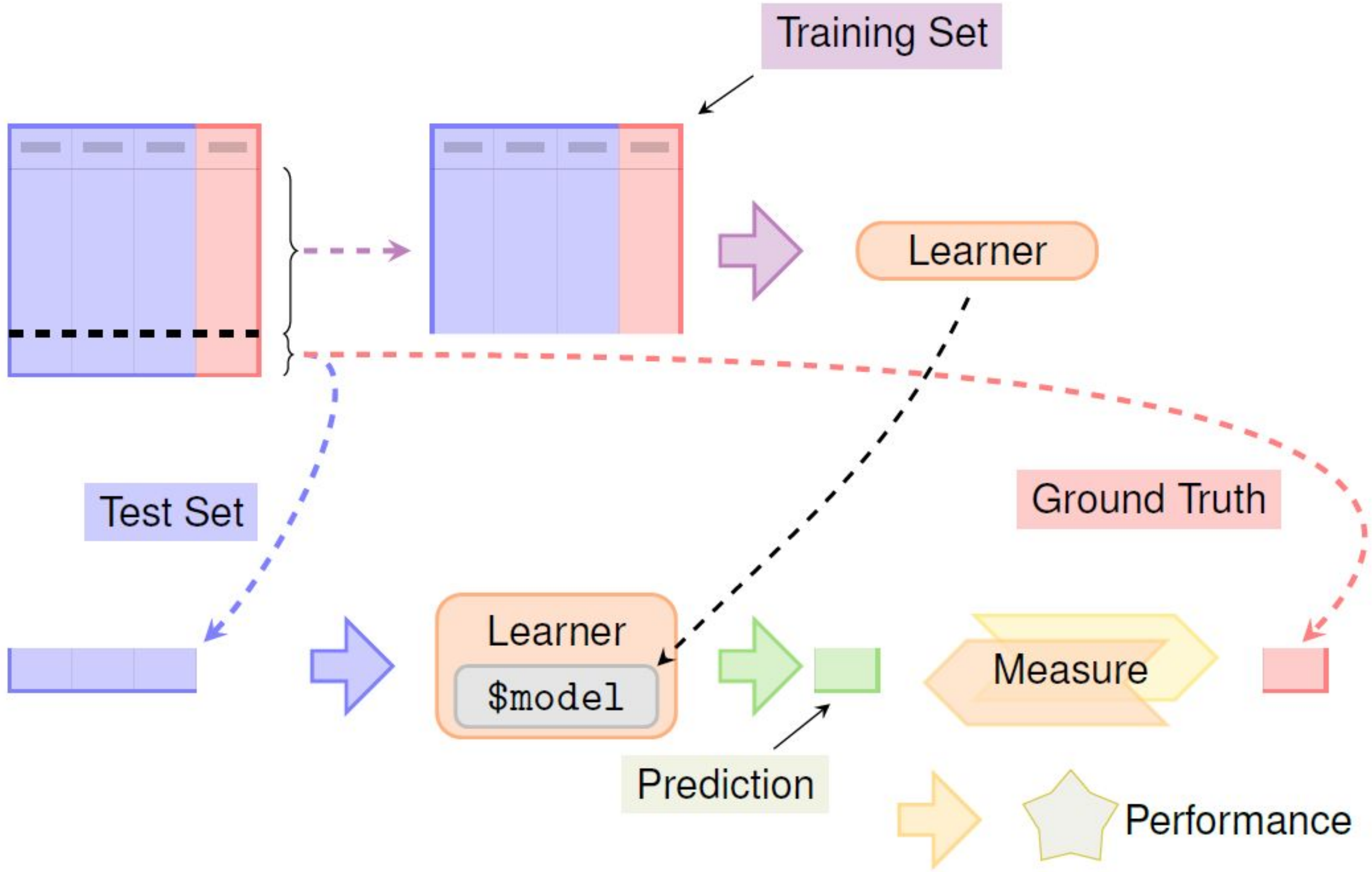
```
pred = learner$predict(known_truth_task)
pred
#> <PredictionClassif> for 2 observations:
#>  row_id  truth  response
#>       1 setosa    setosa
#>       2 setosa virginica
```

- Score the prediction

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

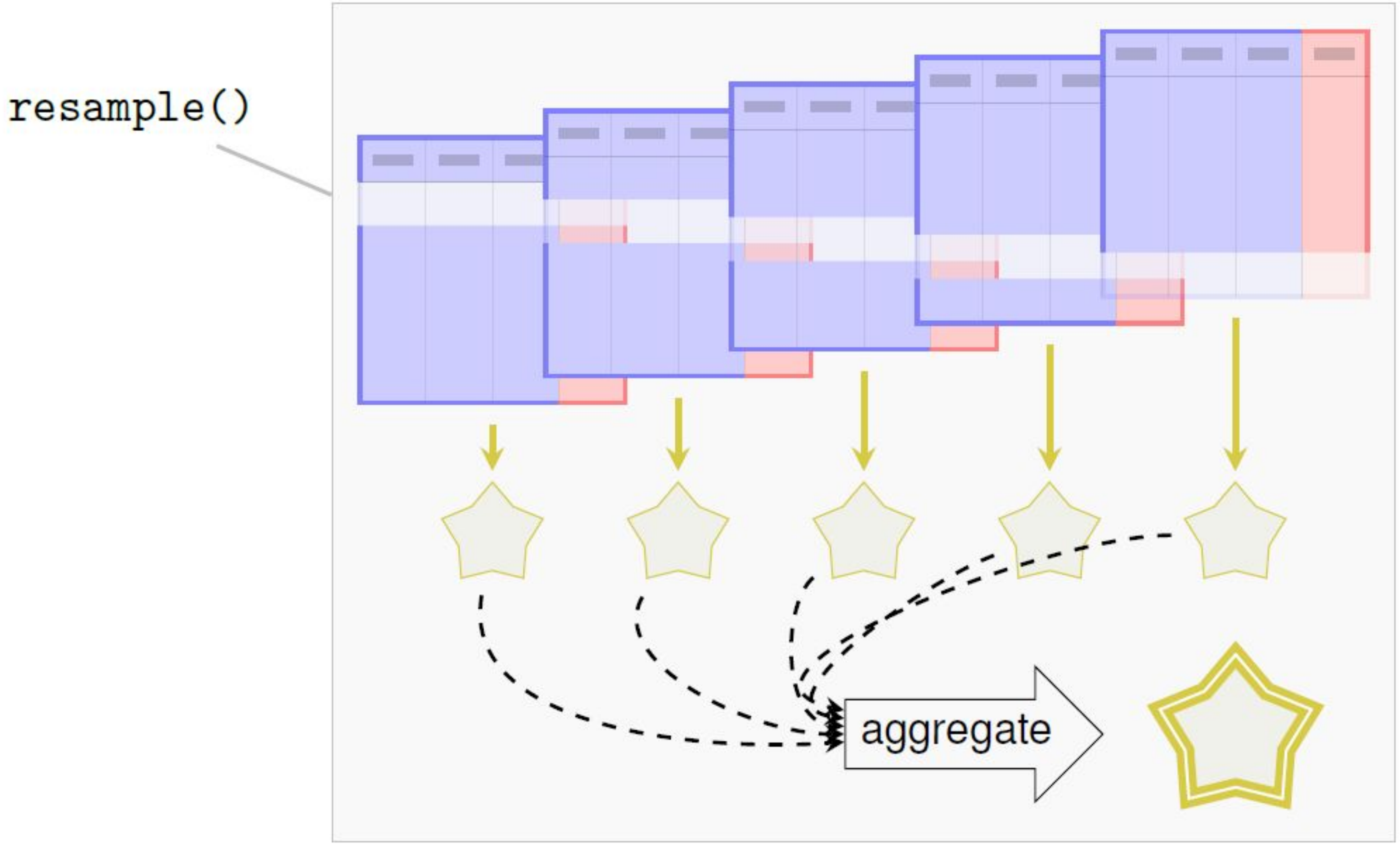
Resampling

Resampling

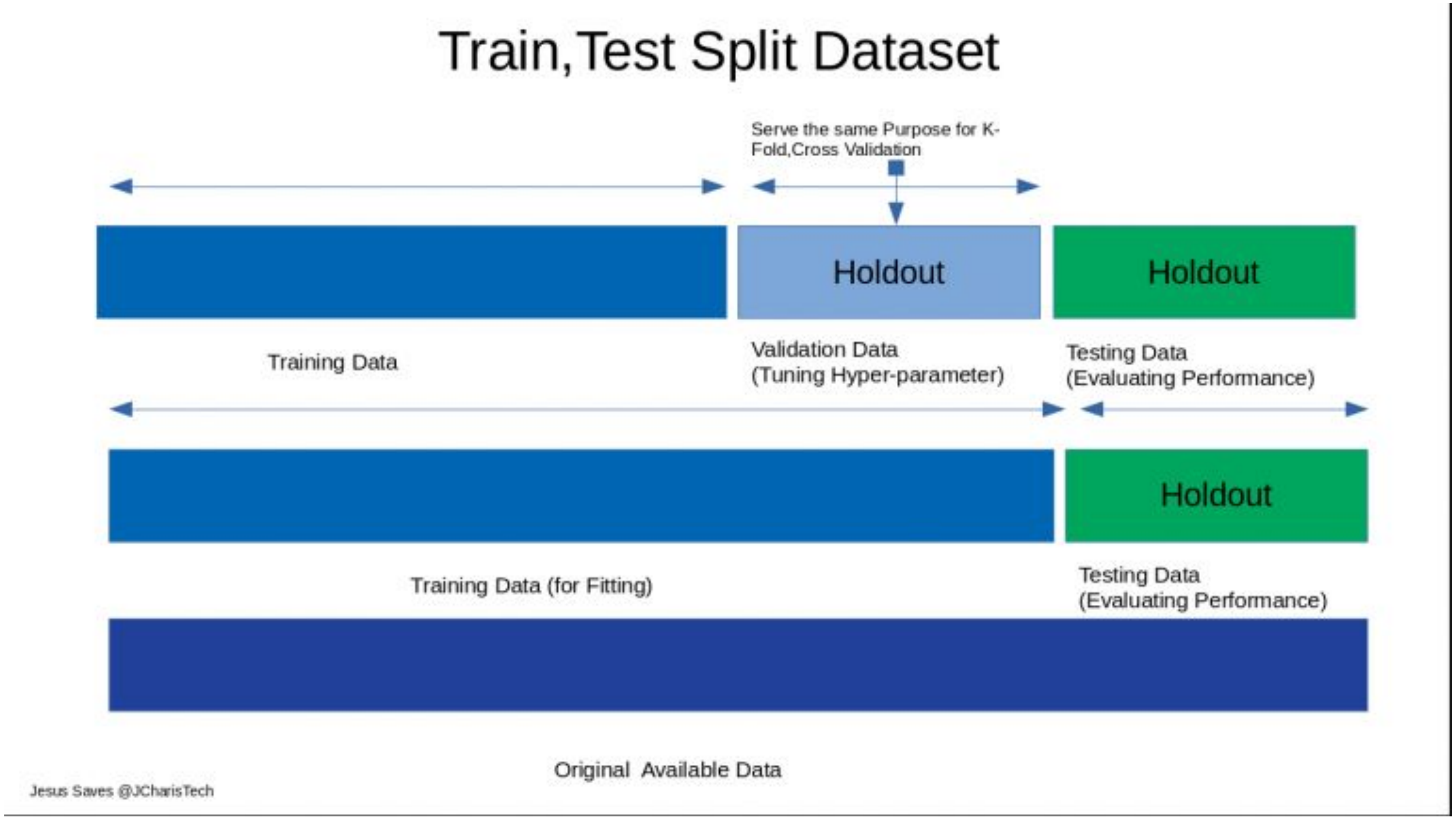


Training Set
Learner
Test Set
Ground Truth
Learner
$model
Measure
Prediction
Performance

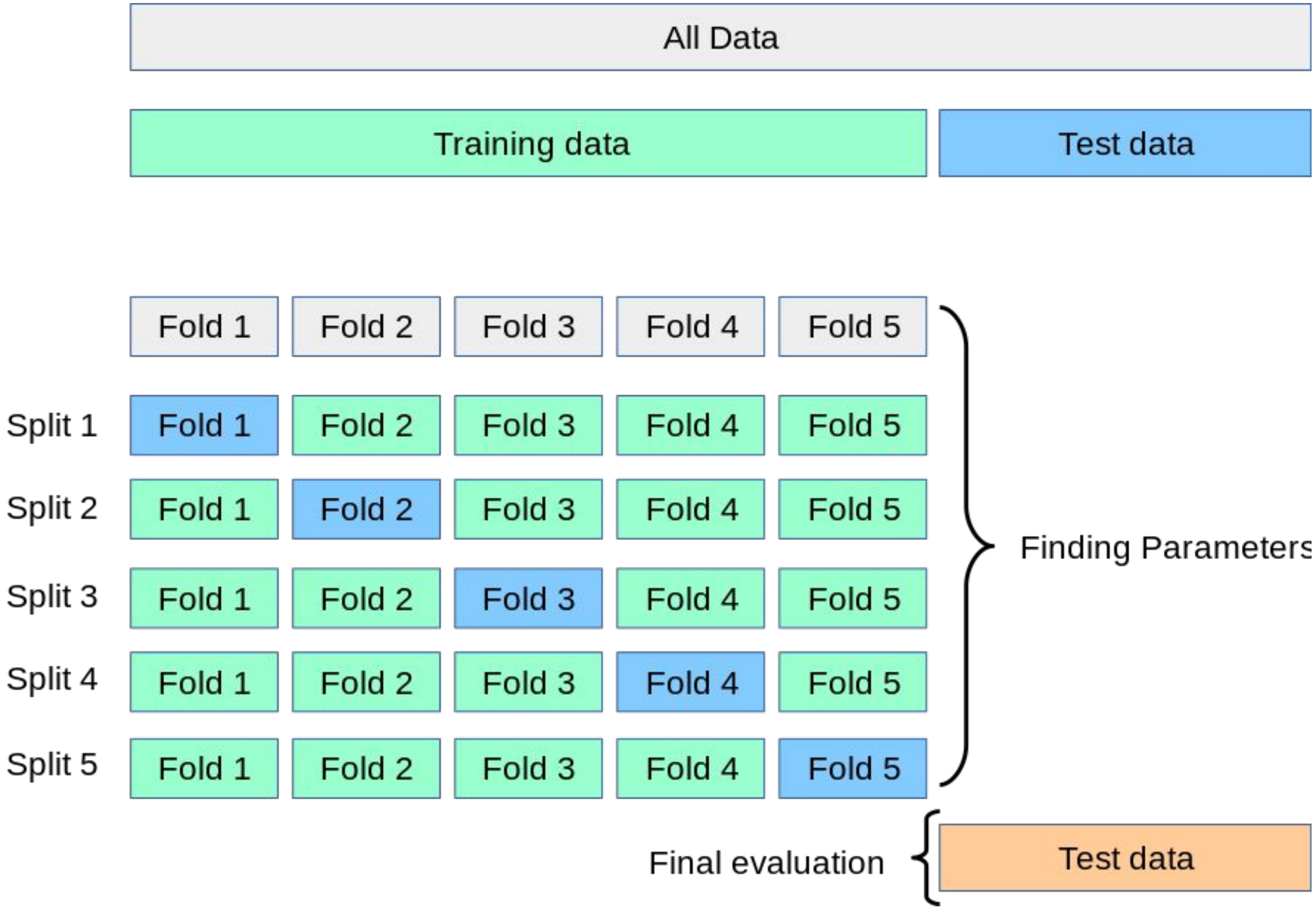
Resampling



Train Test Split Dataset



Cross-validation



Resampling

- Resample description: How to split the data

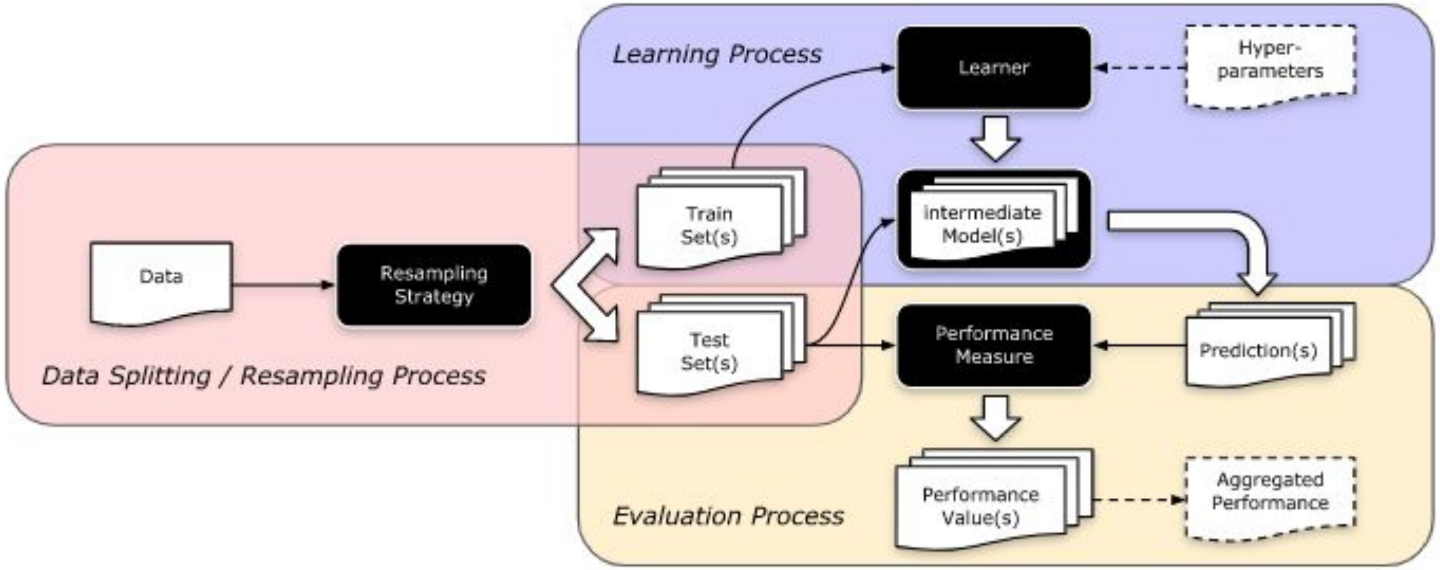
```
cv5 = rsmp("cv", folds = 5)
```

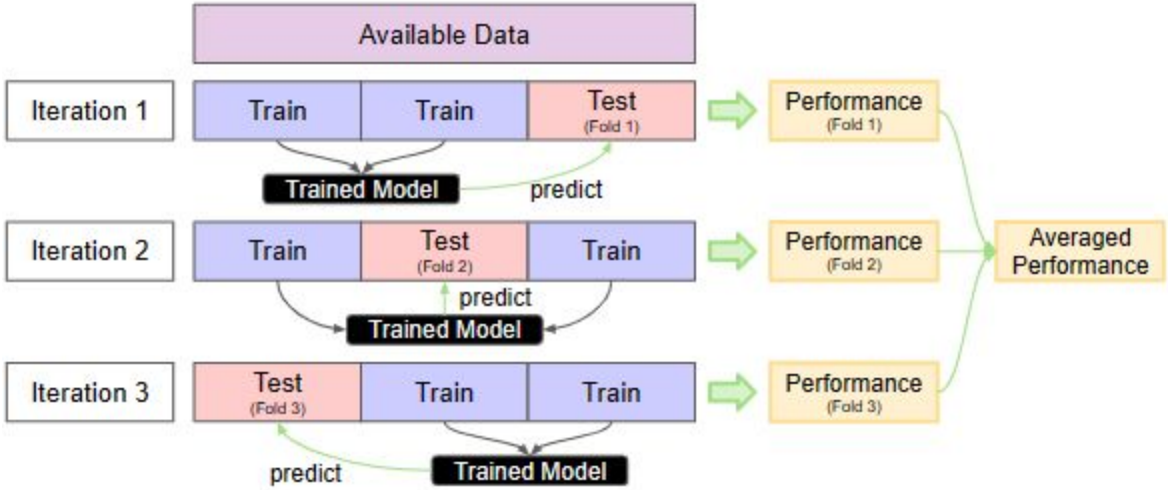
- Use the `resample()` function for resampling:

```
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
```





: Illustration of a 3-fold cross-validation.

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 i...
# 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]>   ...
```

- Active bindings and functions that make information easily accessible

Resampling results

- Calculate performance:

```
rr$aggregate(msr("classif.ce"))
#> classif.ce
#>      0.073
```

- Get predictions

```
rr$prediction()
#> <PredictionClassif> for 150 observations:
#>     row_id     truth  response
#>          5    setosa    setosa
#>         14    setosa    setosa
#>         18    setosa    setosa
#> ---
#>        139 virginica virginica
#>        145 virginica virginica
#>        146 virginica virginica
```

Resampling

- Predictions of individual folds

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

#> <PredictionClassif> for 30 observations:
#>     row_id     truth  response
#>          5    setosa    setosa
#>         14    setosa    setosa
#>         18    setosa    setosa
#> ---
#>        132 virginica virginica
#>        137 virginica virginica
#>        147 virginica virginica
```

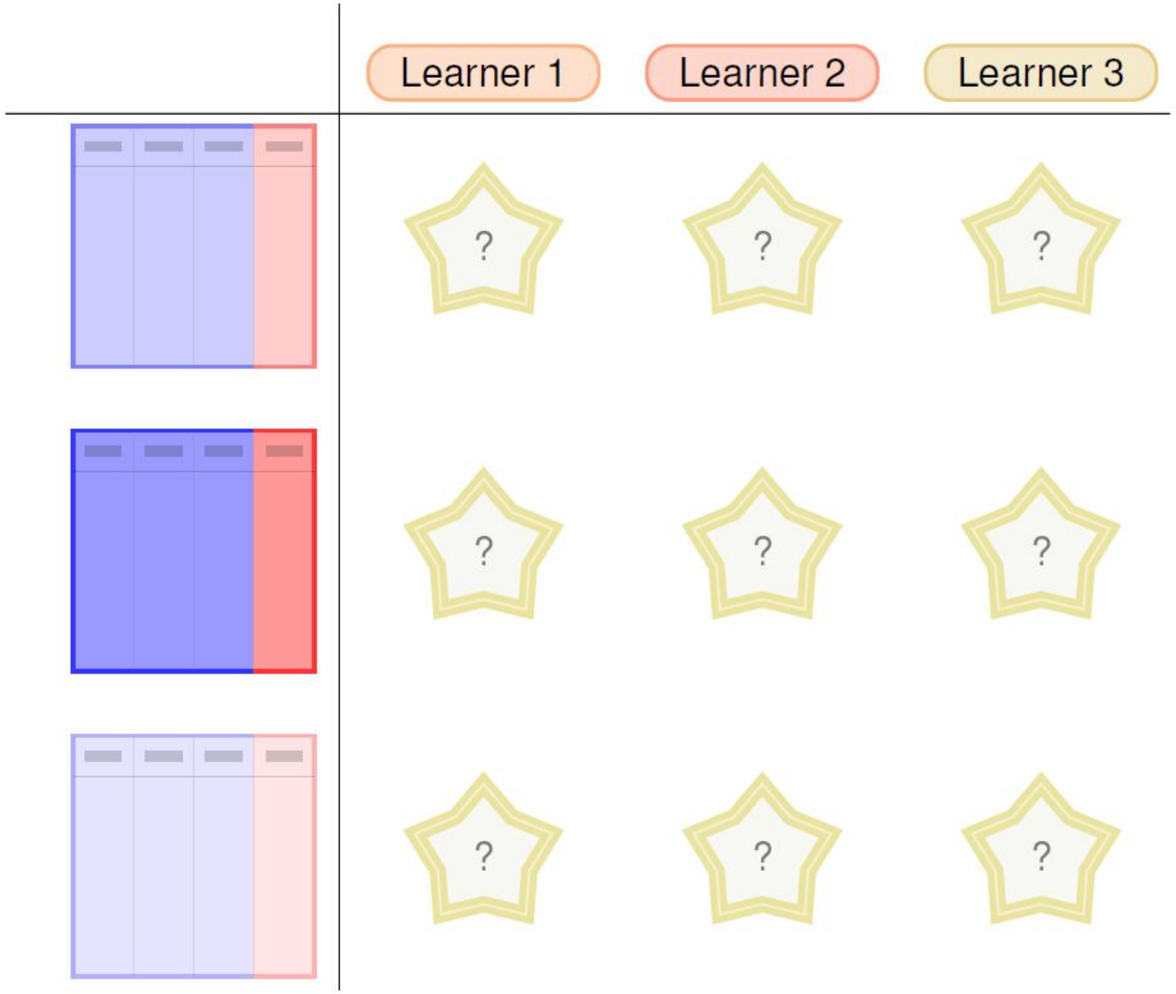
- Score of individual folds

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

#>    iteration classif.ce
#> 1:         1      0.033
#> 2:         2      0.033
#> 3:         3      0.100
```

Benchmark

Performance Comparison



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      0.060
#> 2:    iris  classif.kknn      0.060
#> 3:   sonar classif.rpart      0.279
#> 4:   sonar  classif.kknn      0.168
#> 5:    wine classif.rpart      0.101
#> 6:    wine  classif.kknn      0.051
```

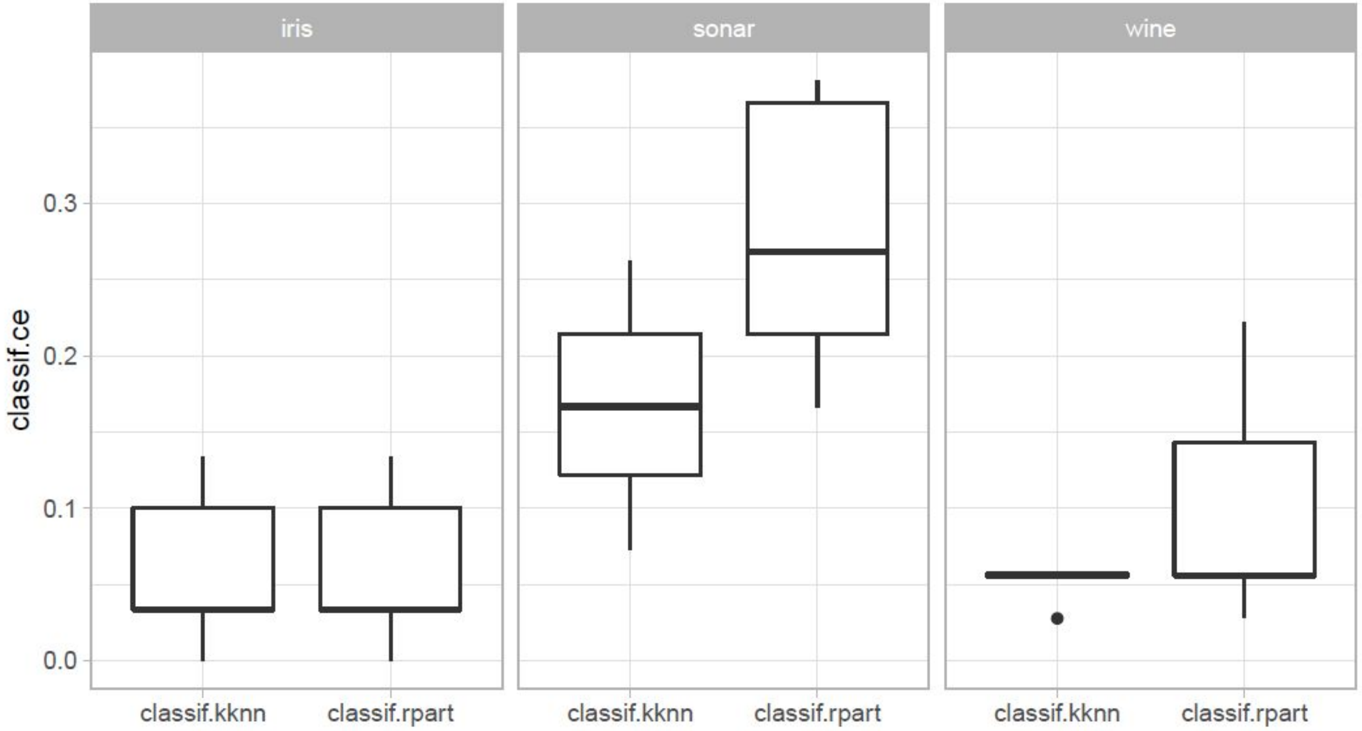
Benchmark result

- 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
- Possibility 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

- Get help for R6 objects?
  - Find out what kind of R6 object you have:

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

  - Go to the corresponding help page:

```
?BenchmarkResult
```

New: open the corresponding man page with

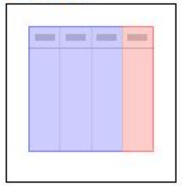
```
learner$help()
```

Outro

Overview

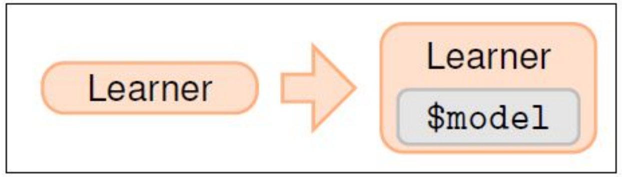
Ingredients:

Data



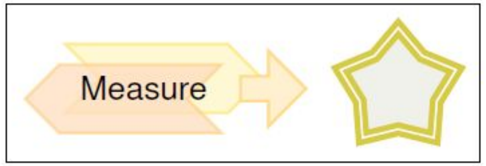
`TaskClassif`,
`TaskRegr`,
`tsk()`

Learning Algorithms



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

Performance Evaluation

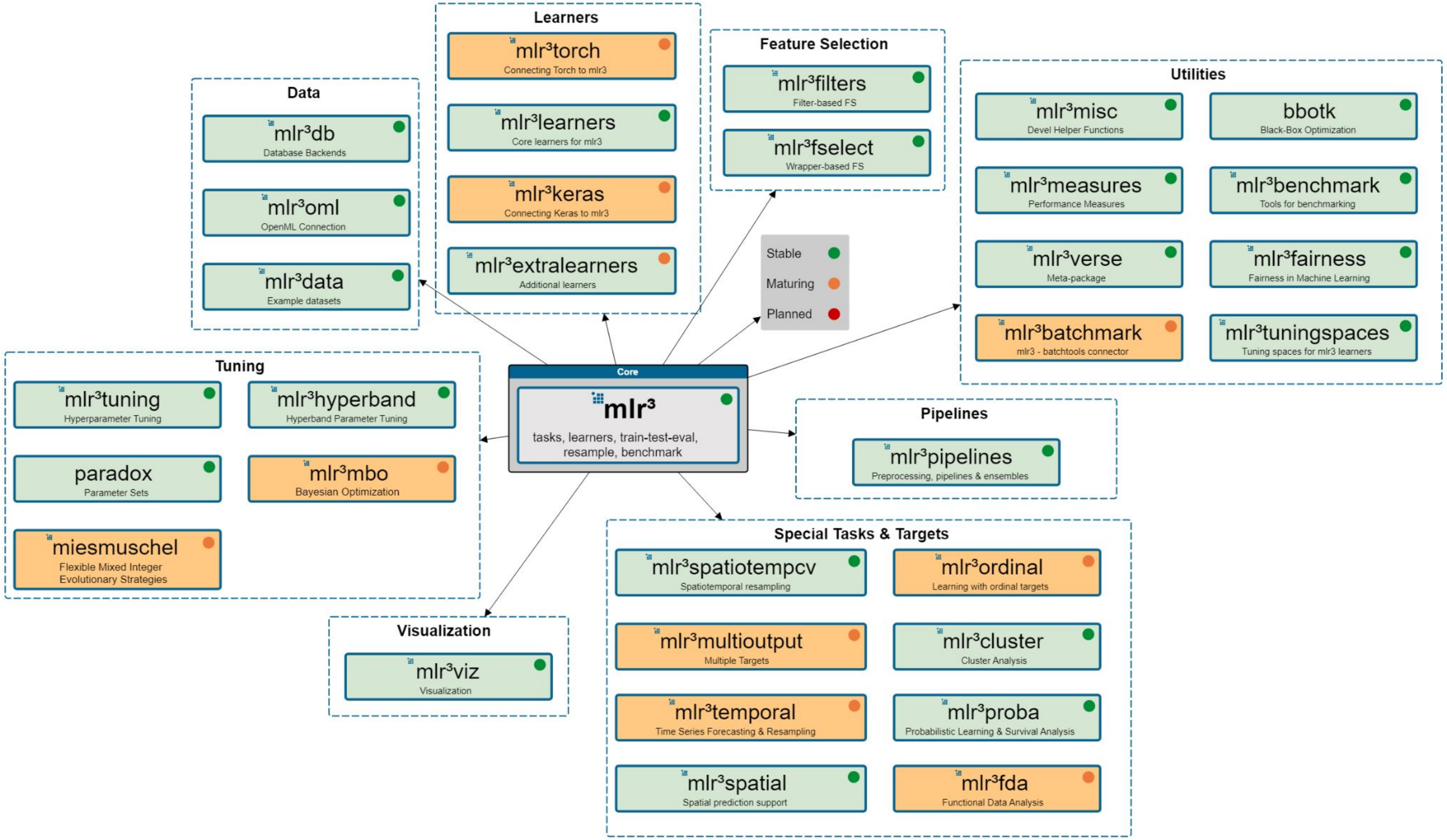


`rsmp()` ⇒ `Resampling`,
`msr()` ⇒ `Measure`,
`resample()` ⇒ `ResamplingResult`,
`$aggregate()`

Performance Comparison

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

mlr3 ecosystem



Conclusion

- Machine learning is a powerful tool that can be used to solve a wide variety of problems.
- mlr3 is a **powerful machine learning library** that makes it easy to build and train machine learning models with R.

Demo

https://bit.ly/mlr3-demo

LinkedIn: https://www.linkedin.com/in/kuanhoong/
Twitter: https://www.twitter.com/kuanhoong

Resources

- mlr3 website
- Flexible and Robust Machine Learning Using mlr3 in R (ebook)
- Exploring the World of Machine Learning with mlr3 in R
- Building ML models using mlr3
- mlr3 cheatsheets
- Introduction to Machine Learning (I2ML)