

# Introduction to mlr

## Beginner Workshop

---

Jane Thomas, Daniel Schalk

2018-07-03

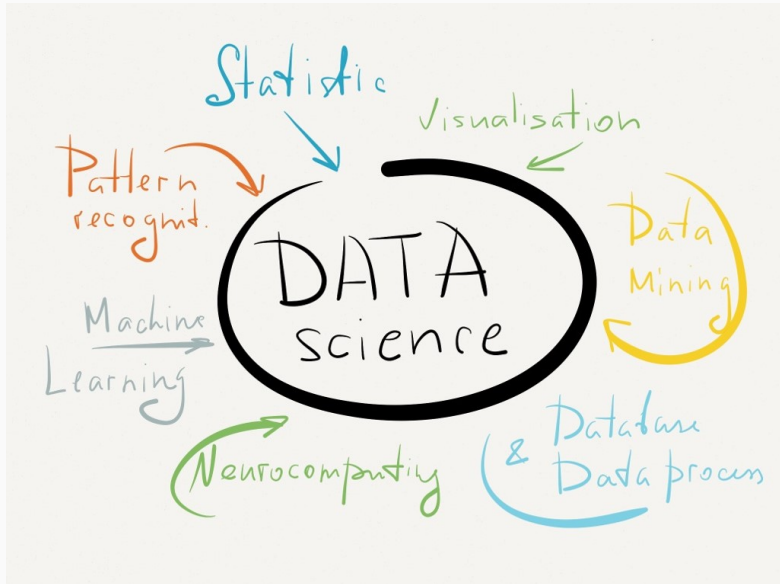


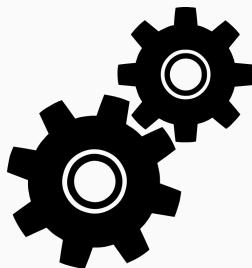


# WHAT IS MACHINE LEARNING

---

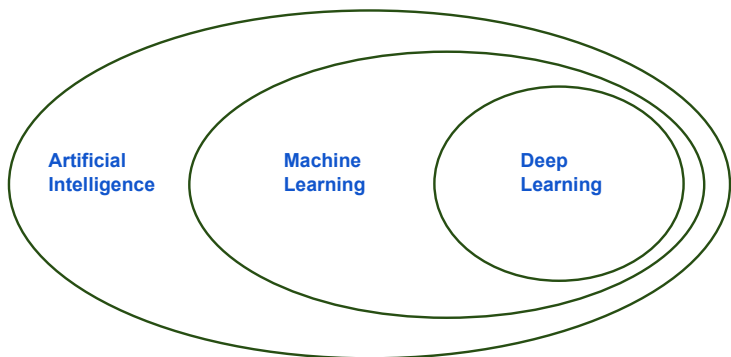
# DATA SCIENCE AND MACHINE LEARNING





Machine Learning is a method of teaching computers to make predictions based on some data.

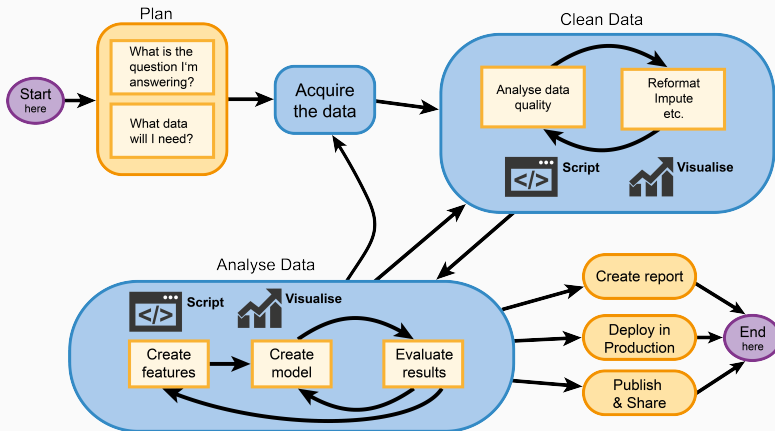
# DATA SCIENCE AND MACHINE LEARNING



# MACHINE LEARNING IS CHANGING OUR WORLD

- Search engines learn what you want
- Recommender systems learn your taste in books, music, movies, . . .
- Algorithms do automatic stock trading
- Elections are won by understanding voters
- Google Translate learns how to translate text
- Siri learns to understand speech
- DeepMind beats humans at Go
- Cars drive themselves
- Medicines are developed faster
- Smartwatches monitor your health
- Data-driven discoveries are made in Physics, Biology, Genetics, Astronomy, Chemistry, Neurology, . . .

# MOTIVATION





# MLR AND A FIRST EXAMPLE

---

# MOTIVATION: MACHINE LEARNING IN R

The **good** news:

- CRAN serves hundreds of packages for machine learning
- Often compliant to the unwritten interface definition:

```
model = fit(target ~ ., data = train.data, ...)  
predictions = predict(model, newdata = test.data, ...)
```

The **bad** news:

- Some packages' API is “just different”
- Functionality is always package or model-dependent, even though the procedure might be general
- No meta-information available or buried in docs

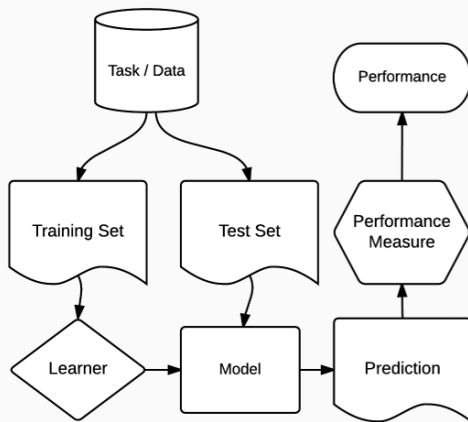
**Our goal: A domain-specific language for ML concepts!**



- Project home page: <https://github.com/mlr-org/mlr>
  - Cheatsheet for an quick overview
  - Tutorial for mlr documentation with many code examples
  - Ask questions in the GitHub issue tracker
- 8-10 main developers, quite a few contributors, 4 GSOC projects in 2015/16 and one coming in 2017
- About 30K lines of code, 8K lines of unit tests

# MOTIVATION: MLR

- Unified interface for the basic building blocks: tasks, learners, hyperparameters, ...



# FEATURES OF MLR

- **Tasks and Learners**
- **Train, Test, Resample**
- **Performance**
- **Benchmarking**
- Hyperparameter Tuning
- Nested Resampling
- Parallelization

- Extensive Tutorial covers *all* features in mlr:  
<https://mlr-org.github.io/mlr/>
- Tuning
- Resampling (with blocking)
- Visualization Topics
- Multilabel Classification, Survival Analysis, Clustering
- Handling Spatial Data
- Functional Data
- Create Custom Learners and Measures
- ...

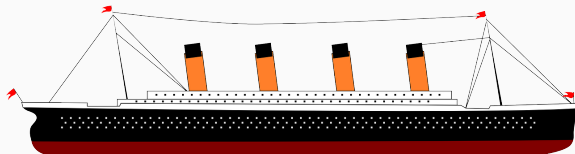
# GETTING HELP

- Ask questions on Stackoverflow:  
<https://stackoverflow.com/questions/tagged/mlr>
- Found bugs? Report them:  
<https://github.com/mlr-org/mlr/issues>

You want to contribute? - Open a PR on github and join our slack: <https://mlr-org.slack.com/>

# TITANIC - MACHINE LEARNING FROM DISASTER

- Titanic sinking on April 15, 1912
- Data provided on Kaggle:  
<https://www.kaggle.com/c/titanic>
- 809 out of 1309 passengers died
- Task:
  - Can we predict who survived?
  - Why did people die / Which groups?





# TITANIC - DATA SET

## Data Dictionary:

Survived	Survived, 0 = No, 1 = Yes
Pclass	Ticket class, from 1st to 3rd
Sex	Sex
Age	Age in years
Sibsp	# of siblings/ spouses
Parch	# of parents/ children
Ticket	Ticket number
Fare	Passenger fare
Cabin	Cabin number
Embarked	Port of Embarkation

# TITANIC - DATA SET

```
load("titanic.rda")
str(data)
```

```
## 'data.frame':    1309 obs. of  11 variables:
## $ Pclass   : Factor w/ 3 levels "1","2","3": 1 1 1 1 1 1 1 1 ...
## $ Survived : Factor w/ 2 levels "0","1": 2 2 1 1 1 2 2 1 ...
## $ Name     : chr  "Allen, Miss. Elisabeth Walton" "Allison, Ma"..
## $ Sex      : Factor w/ 2 levels "female","male": 1 2 1 2 1 2 1 ..
## $ Age      : num  29 0.917 2 30 ...
## $ Sibsp    : num  0 1 1 1 1 0 1 0 ...
## $ Parch    : num  0 2 2 2 2 0 0 0 ...
## $ Ticket   : Factor w/ 929 levels "110152","110413",...: 188 50 ..
## $ Fare     : num  211 152 152 152 ...
## $ Cabin    : Factor w/ 187 levels "", "A10", "A11",...: 45 81 81 8..
## $ Embarked : Factor w/ 4 levels "", "C", "Q", "S": 4 4 4 4 4 4 4 4..
```

# TITANIC - DATA SET

```
library(mlr)
print(summarizeColumns(data)[, -c(5, 6, 7)], digits = 0)
```

##	name	type	na	mean	min	max	nlevs
## 1	Pclass	factor	0	NA	277	709	3
## 2	Survived	factor	0	NA	500	809	2
## 3	Name	character	0	NA	1	2	1307
## 4	Sex	factor	0	NA	466	843	2
## 5	Age	numeric	263	30	0	80	0
## 6	Sibsp	numeric	0	0	0	8	0
## 7	Parch	numeric	0	0	0	9	0
## 8	Ticket	factor	0	NA	1	11	929
## 9	Fare	numeric	1	33	0	512	0
## 10	Cabin	factor	0	NA	1	1014	187
## 11	Embarked	factor	0	NA	2	914	4

# TITANIC - PREPROCESSING

Set empty factor levels to NA:

```
data$Embarked[data$Embarked == ""] = NA
data$Embarked = droplevels(data$Embarked)
data$Cabin[data$Cabin == ""] = NA
data$Cabin = droplevels(data$Cabin)
```

# TITANIC - PREPROCESSING

```
library(BBmisc)
library(stringi)

# Price per person, multiple tickets bought by one person
data$farePp = data$Fare / (data$Parch + data$Sibsp + 1)

# The deck can be extracted from the the cabin number
data$deck = as.factor(stri_sub(data$Cabin, 1, 1))

# Starboard had an odd number, portside even cabin numbers
data$portside = stri_extract_last_regex(data$Cabin, "[0-9]")
data$portside = as.numeric(data$portside) %% 2

# Drop stuff we cannot easily model on
data = dropNamed(data,
  c("Cabin", "PassengerId", "Ticket", "Name"))
```

# TITANIC - PREPROCESSING

```
print(summarizeColumns(data)[, -c(5, 6, 7)], digits = 0)
```

##	name	type	na	mean	min	max	nlevs
## 1	Pclass	factor	0	NA	277	709	3
## 2	Survived	factor	0	NA	500	809	2
## 3	Sex	factor	0	NA	466	843	2
## 4	Age	numeric	263	30	0	80	0
## 5	Sibsp	numeric	0	0	0	8	0
## 6	Parch	numeric	0	0	0	9	0
## 7	Fare	numeric	1	33	0	512	0
## 8	Embarked	factor	2	NA	123	914	3
## 9	farePp	numeric	1	21	0	512	0
## 10	deck	factor	1014	NA	1	94	8
## 11	portside	numeric	1020	0	0	1	0

# TITANIC - PREPROCESSING

- Impute missing numeric values with median, missing factor values with a separate category
- NB: This is really naive, we should probably embed this in cross-validation

```
data = impute(data, cols = list(  
  Age = imputeMedian(),  
  Fare = imputeMedian(),  
  Embarked = imputeConstant("__miss__"),  
  farePp = imputeMedian(),  
  deck = imputeConstant("__miss__"),  
  portside = imputeConstant("__miss__")  
))  
  
data = data$data  
data = convertDataFrameCols(data, chars.as.factor = TRUE)
```

# TITANIC - TASK

```
task = makeClassifTask(id = "titanic", data = data,  
  target = "Survived", positive = "1")  
print(task)
```

```
## Supervised task: titanic  
## Type: classif  
## Target: Survived  
## Observations: 1309  
## Features:  
##      numerics      factors    ordered functionals  
##           5           5           0           0  
## Missings: FALSE  
## Has weights: FALSE  
## Has blocking: FALSE  
## Has coordinates: FALSE  
## Classes: 2  
##    0    1  
## 809 500  
## Positive class: 1
```



# WHAT LEARNERS ARE AVAILABLE?

## Classification

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

## Clustering

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

## Regression

- Linear, lasso and ridge
- Boosting
- Trees and forests
- Gaussian processes
- (Deep) Neural Networks
- ...

## Survival

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

# WHAT LEARNERS ARE AVAILABLE?

We can explore them on the webpage:

mlr 2.13 Get Started Basics ▾ Advanced ▾ Extending ▾ Appendix ▾ mlr-org Packages ▾ Search...								
Class / Short Name / Name	Packages	Num.	Fac.	Ord.	NAs	Weights	Props	Note
<b>classif.ada</b> <i>ada</i>  ada Boosting	<a href="#">ada</a> <a href="#">rpart</a>	X	X				prob twoclass	<code>xval</code> has been set to <code>0</code> by default for spe
<b>classif.adaboostm1</b> <i>adaboostm1</i>  ada Boosting M1	<a href="#">RWeka</a>	X	X				prob twoclass multiclass	NAs are directly passed to WEKA with <code>na.ac</code>
<b>classif.bartMachine</b> <i>bartmachine</i>  Bayesian Additive Regression Trees	<a href="#">bartMachine</a>	X	X		X		prob twoclass	<code>use_missing_data</code> has been set to <code>TRUE</code>
<b>classif.binomial</b> <i>binomial</i>  Binomial Regression	<a href="#">sjats</a>	X	X			X	prob twoclass	Delegates to <code>glm</code> with freely choosable bin

# WHAT LEARNERS ARE AVAILABLE?

Or ask mlr

```
tab = listLearners(task, warn.missing.packages = FALSE)
tab[1:5, c("class", "package")]
```

##	class	package
## 1	classif.ada	ada,rpart
## 2	classif.adaboostm1	RWeka
## 3	classif.bartMachine	bartMachine
## 4	classif.binomial	stats
## 5	classif.blackboost	mboost,party

# TITANIC - LEARNER

```
lrn = makeLearner("classif.kknn", k = 3, predict.type = "prob")  
print(lrn)
```

```
## Learner classif.kknn from package kknn  
## Type: classif  
## Name: k-Nearest Neighbor; Short name: kknn  
## Class: classif.kknn  
## Properties: twoclass,multiclass,numerics,factors,prob  
## Predict-Type: prob  
## Hyperparameters: k=3
```

# TITANIC - TRAIN

```
set.seed(123)
n = getTaskSize(task)
train = sample(n, size = 2/3 * n)
test = setdiff(1:n, train)

head(sort(train))

## [1] 1 2 3 5 7 9

head(sort(test))

## [1] 4 6 8 11 12 18

mod = train(lrn, task, subset = train)
```

# TITANIC - MODEL

```
print(mod)

## Model for learner.id=classif.kknn; learner.class=classif.kknn
## Trained on: task.id = titanic; obs = 872; features = 10
## Hyperparameters: k=3

# retrieve model as returned from the third party package
# [NB: knn does not have a training step, mlr just returns the
# training data which is required in the predict step]
rmodel = getLearnerModel(mod)
```

# TITANIC - PREDICT

```
pred = predict(mod, task = task, subset = test)
head(as.data.frame(pred))
```

```
##      id truth prob.0 prob.1 response
## 4     4     0 0.6621 0.338         0
## 6     6     1 0.7358 0.264         0
## 8     8     0 1.0000 0.000         0
## 11    11     0 0.7358 0.264         0
## 12    12     1 0.0000 1.000         1
## 18    18     1 0.0737 0.926         1
```

```
head(getPredictionProbabilities(pred))
```

```
## [1] 0.338 0.264 0.000 0.264 1.000 0.926
```

# TITANIC - PERFORMANCE

```
performance(pred, measures = list(mlr::acc, mlr::auc))
```

```
##    acc    auc
```

```
## 0.725 0.786
```



# TITANIC - EXTERNAL VALIDATION SET

You can also predict on data not included in the task:

```
test.data = dropNamed(data[test, ], "Survived")
pred = predict(mod, newdata = data[test, ])
performance(pred, measures = list(mlr::acc, mlr::auc))

##   acc   auc
## 0.725 0.786
```

# NOTEBOOK 1

---

# RESAMPLING

---

# PARAMETER OF MAKERESAMPLINGDESC

Methods	Parameter
CV	iters (Number of iterations)
L00	
RepCV	reps (Repeats for repeated CV) folds (Folds in the repeated CV)
Bootstrap	iters (Number of iterations)
Subsample	iters (Number of iterations) split (Proportion of training cases)
Holdout	split (Proportion of training cases)

For instance 10-fold cross validation:

```
makeResampleDesc(method = "CV", iters = 10)  
  
## Resample description: cross-validation with 10 iterations.  
## Predict: test  
## Stratification: FALSE
```

# POSSIBLE WAYS TO USE CROSS VALIDATION

## 1. Explicitly define resampling:

```
rdesc = makeResampleDesc("CV", iters = 10)
rdesc = cv10

res1 = resample("classif.randomForest", iris.task, resampling = rdesc,
  show.info = FALSE)
res2 = resample("classif.randomForest", iris.task, resampling = cv10,
  show.info = FALSE)

res1
```

```
## Resample Result
## Task: iris-example
## Learner: classif.randomForest
## Aggr perf: mmce.test.mean=0.047
## Runtime: 0.642717
```

Other pre defined objects are `cv2`, `cv3` and `cv5`.

# POSSIBLE WAYS TO USE CROSS VALIDATION

## 2. Use crossval:

```
res3 = crossval("classif.randomForest", iris.task, iters = 10,  
  show.info = FALSE)  
res3
```

```
## Resample Result  
## Task: iris-example  
## Learner: classif.randomForest  
## Aggr perf: mmce.test.mean=0.053  
## Runtime: 0.367938
```

Similar functions are `repcv`, `holdout`, `subsample`, `bootstrapOOB`, `bootstrapB632` and `bootstrapB632plus`.

# CLASSIFICATION ANALYSIS: BENCHMARKING

*# quick way to compare learners with identical train/test splits*

```
task = iris.task
```

```
learners = list(
```

```
  makeLearner("classif.knn", k = 3),
```

```
  makeLearner("classif.lda"),
```

```
  makeLearner("classif.naiveBayes")
```

```
)
```

```
benchmark(learners, task, resamplings = cv3)
```

##	task.id	learner.id	mmce.test.mean
## 1	iris-example	classif.knn	0.04
## 2	iris-example	classif.lda	0.02
## 3	iris-example	classif.naiveBayes	0.04

# CLASSIFICATION ANALYSIS: BENCHMARKING

```
tasks = list(iris.task, sonar.task, pid.task)
bm = benchmark(learners, tasks, resampling = cv3)
print(bm)
```

##	task.id	learner.id	mmce.test.mean
## 1	iris-example	classif.knn	0.0333
## 2	iris-example	classif.lda	0.0200
## 3	iris-example	classif.naiveBayes	0.0400
## 4	PimaIndiansDiabetes-example	classif.knn	0.3021
## 5	PimaIndiansDiabetes-example	classif.lda	0.2253
## 6	PimaIndiansDiabetes-example	classif.naiveBayes	0.2448
## 7	Sonar-example	classif.knn	0.2207
## 8	Sonar-example	classif.lda	0.2355
## 9	Sonar-example	classif.naiveBayes	0.3226



# CLASSIFICATION ANALYSIS: BENCHMARKING

*# aggregated data:*

```
getBMRAggrPerformances(bm, as.df = TRUE)
```

##	task.id	learner.id	mmce.test.mean
## 1	iris-example	classif.knn	0.0333
## 2	iris-example	classif.lda	0.0200
## 3	iris-example	classif.naiveBayes	0.0400
## 4	PimaIndiansDiabetes-example	classif.knn	0.3021
## 5	PimaIndiansDiabetes-example	classif.lda	0.2253
## 6	PimaIndiansDiabetes-example	classif.naiveBayes	0.2448
## 7	Sonar-example	classif.knn	0.2207
## 8	Sonar-example	classif.lda	0.2355
## 9	Sonar-example	classif.naiveBayes	0.3226

# CLASSIFICATION ANALYSIS: BENCHMARKING

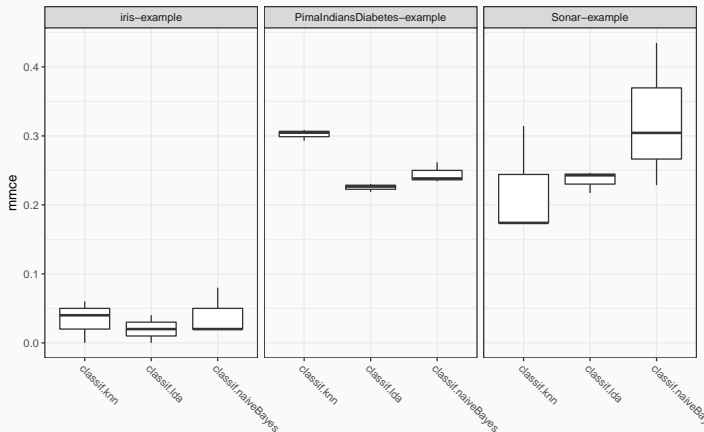
*# complete data:*

```
head(as.data.frame(bm), 10)
```

##	task.id	learner.id	iter	mmce
## 1	iris-example	classif.knn	1	0.040
## 2	iris-example	classif.knn	2	0.060
## 3	iris-example	classif.knn	3	0.000
## 4	iris-example	classif.lda	1	0.020
## 5	iris-example	classif.lda	2	0.040
## 6	iris-example	classif.lda	3	0.000
## 7	iris-example	classif.naiveBayes	1	0.020
## 8	iris-example	classif.naiveBayes	2	0.080
## 9	iris-example	classif.naiveBayes	3	0.020
## 10	PimaIndiansDiabetes-example	classif.knn	1	0.309

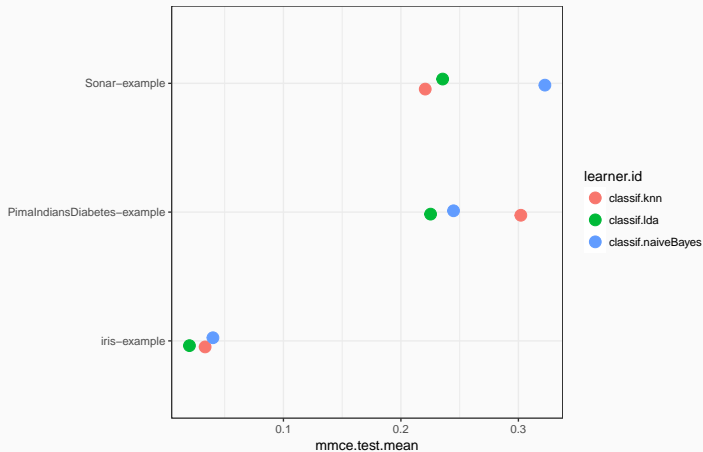
# CLASSIFICATION ANALYSIS: BENCHMARKING

```
plotBMRBoxplots(bm, pretty.names = FALSE)
```



# CLASSIFICATION ANALYSIS: BENCHMARKING

```
plotBMRSummary(bm, pretty.names = FALSE)
```



# NOTEBOOK 2

---