Munich Datageeks

Stefan Coors

July 27, 2017

Contents

- Introduction
- What is shiny, mlr and shinyMlr?
- LIVE DEMO with Kaggle's Titanic Data

Introduction

About me

- B.Sc. and M.Sc in mathematics and economics at Bielefeld University
- Right now: 2nd Master in statistics at LMU and research associate at the institute of Psychological methods and Diagnostics

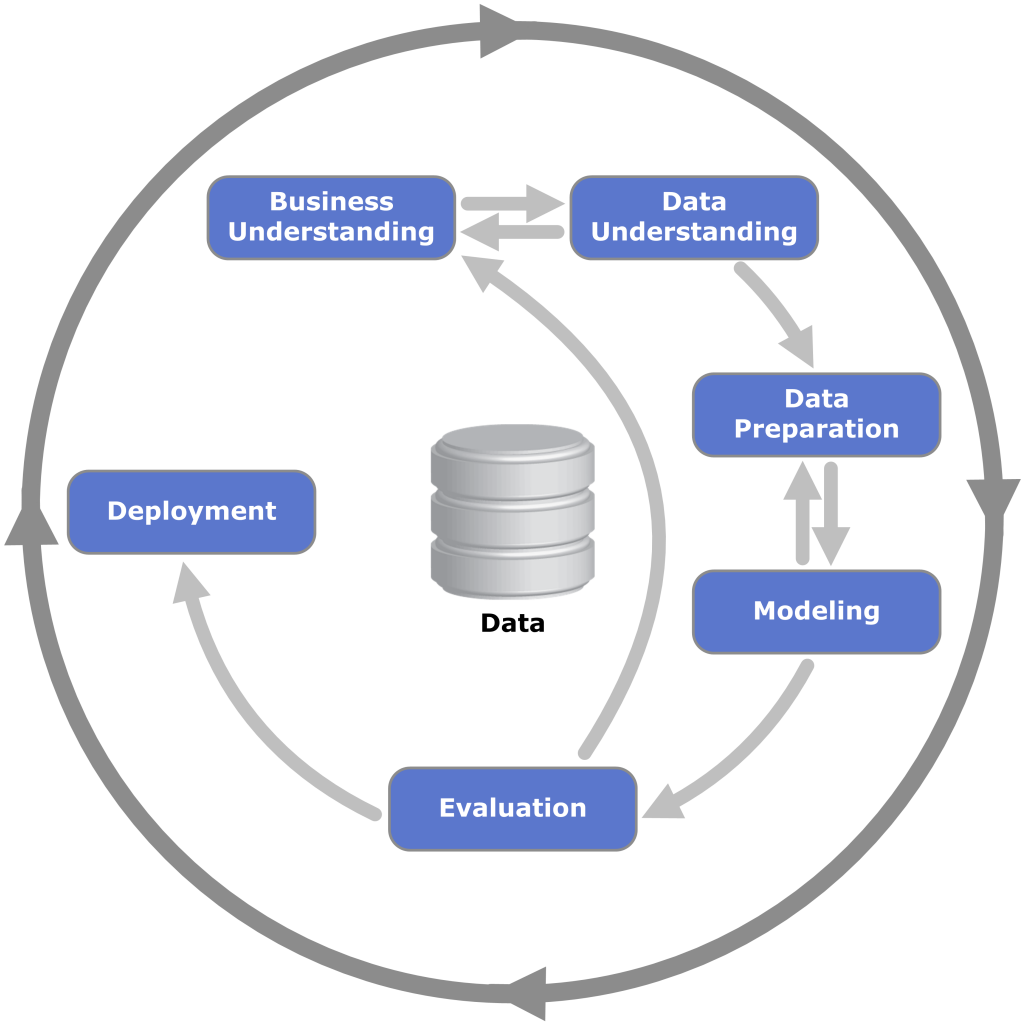
Introduction / 2

Machine learning experiments consist of different blocks:

- data exploring and visualisation
- preprocessing
- specifying ML task (classification/Regression?)
- choosing suitable learning methods and
- tuning their hyperparameters
- evaluating trained models and
- predicting on new data
- visualising the results

Introduction / 3

CRoss-**I**ndustry **S**tandard **P**rocess for **D**ata **M**ining



Introduction / 4

R provides many different packages for machine learning tasks. Of course, each package has its own API:

Example: Decision Tree (rpart, ctree)

```
rpart(formula, data, weights, subset,
  na.action = na.rpart, method, model = FALSE,
  x = FALSE, y = TRUE, parms, control, cost, ...)

ctree(formula, data, subset = NULL, weights = NULL,
  controls = ctree_control(), xtrafo = ptrafo,
  ytrafo = ptrafo, scores = NULL)
```

Introduction / 5

- To conduct a whole machine-learning experiment you need lots of different packages and learn the API for each one
- Imagine a benchmark experiment with 5 learners and cross-validation: You need a package for each learner, a package for the evaluation measure (or you code it yourself) and at least one package to perform resampling
- $\Rightarrow$ This makes coding complicated, slow and error prone

But we can do better:
CRAN hosts two packages that combine many popular packages for all basic machine-learning-blocks and provide a unified API (application programming interface) to minimize coding effort:

caret (**C**lassification **A**nd **RE**gression **T**raining)

mlr (**M**achine **L**earning for **R**)

What is mlr?

- Unified interface for the basic building blocks: tasks, learners, resampling, hyper parameters, . . .
- Reflections: nearly all objects are queryable (i.e. you can ask them for their properties and program on them)
- Tasks encapsulate data and meta-information about it
- Regression, classification, clustering, survival tasks
- Data is stored inside an environment to save memory

$\Rightarrow$ Let's learn mlr with an example!

Titanic survival data

- April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg
- 1502 out of 2224 passengers and crew died
- Surely, there was some element of luck involved in surviving the sinking
- However: some groups of people were more likely to survive than others, such as women, children, and the upper-class.

Aim of analysis: predict which passengers survived the tragedy!

Titanic survival data / 2

Variable	Name Description
`Survived`	Survived (1) or died (0)
`Pclass`	Passenger's class
`Name`	Passenger's name
`Sex`	Passenger's sex
`Age`	Passenger's age in years
`SibSp`	Number of siblings/spouses aboard
`Parch`	Number of parents/children aboard
`Ticket`	Ticket number
`Fare`	Fare
`Cabin`	Cabin
`Embarked`	Port of embarkation: C = Cherbourg, Q = Queenstown, S = Southampton

Creating a classification task

```
task = makeClassifTask(id = "titanic-task", data = train, target = "Survived")
print(task)

## Supervised task: titanic-task
## Type: classif
## Target: Survived
## Observations: 594
## Features:
## numerics  factors  ordered
##        5        2        0
## Missings: FALSE
## Has weights: FALSE
## Has blocking: FALSE
## Classes: 2
##   0   1
## 366 228
## Positive class: 0
```

Creating a learner

- We can choose between several different learning methods
- Depends on the underlying task

Classification (79)

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

Regression (60)

- Linear, lasso and ridge
- Boosting
- Trees and forests
- Gaussian processes
- ...

Clustering (9)

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

Survival (13)

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

Creating a learner / 2

```
lrn = makeLearner("classif.randomForest", predict.type = "prob")
print(lrn)

## Learner classif.randomForest from package randomForest
## Type: classif
## Name: Random Forest; Short name: rf
## Class: classif.randomForest
## Properties: twoclass,multiclass,numerics,factors,ordered,prob,class.weights,oo
## Predict-Type: prob
## Hyperparameters:
```

Set learner parameters

- Extensive meta-information for hyper parameters available: storage type, constraints, defaults, dependencies
- Automatically checked for feasibility

```
getParamSet(lrn)
```

```
##                     Type  len   Def   Constr Req Tunable Trafo
## ntree            integer    -   500 1 to Inf   -    TRUE     -
## mtry             integer    -     - 1 to Inf   -    TRUE     -
## replace          logical    -  TRUE        -   -    TRUE     -
## classwt     numericvector <NA>     - 0 to Inf   -    TRUE     -
## cutoff      numericvector <NA>     -   0 to 1   -    TRUE     -
## strata           untyped    -     -        -   -   FALSE     -
## sampsize    integervector <NA>     - 1 to Inf   -    TRUE     -
## nodesize         integer    -     1 1 to Inf   -    TRUE     -
## maxnodes         integer    -     - 1 to Inf   -    TRUE     -
## importance       logical    - FALSE        -   -    TRUE     -
## localImp         logical    - FALSE        -   -    TRUE     -
## proximity        logical    - FALSE        -   -   FALSE     -
## oob.prox         logical    -     -        -   Y   FALSE     -
## norm.votes       logical    -  TRUE        -   -   FALSE     -
## do.trace         logical    - FALSE        -   -   FALSE     -
## keep.forest      logical    -  TRUE        -   -   FALSE     -
## keep.inbag       logical    - FALSE        -   -   FALSE     -
```

Train learner

```
mod = train(learner = lrn, task = task)
print(mod)

## Model for learner.id=classif.randomForest; learner.class=classif.randomForest
## Trained on: task.id = titanic-task; obs = 594; features = 7
## Hyperparameters:
```

Predict on new data

```
pred = predict(obj = mod, newdata = test)
print(pred)
```

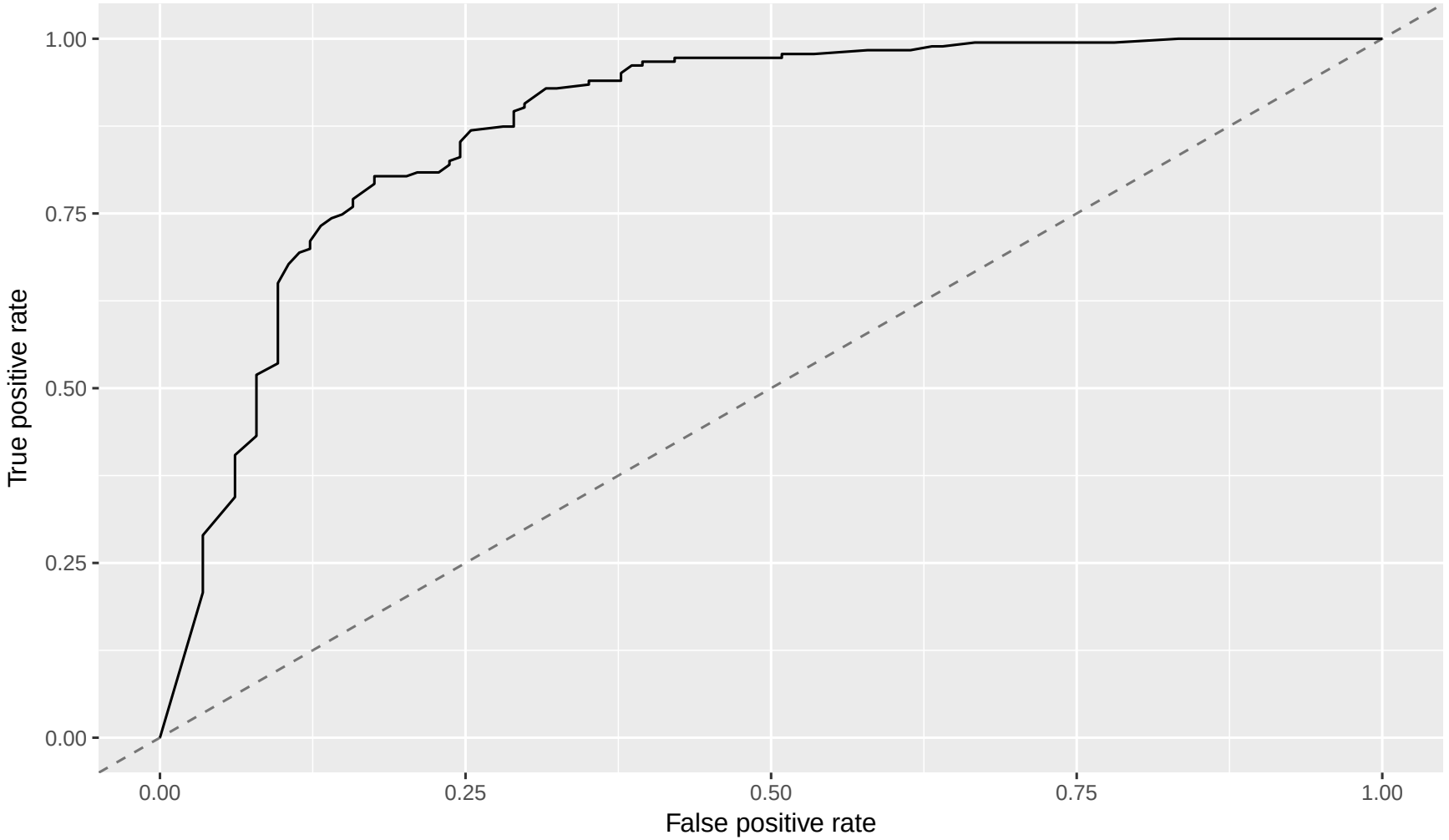
```
## Prediction: 297 observations
## predict.type: prob
## threshold: 0=0.50,1=0.50
## time: 0.08
##     truth prob.0 prob.1 response
## 154     0  0.834  0.166        0
## 499     0  0.108  0.892        1
## 833     0  0.946  0.054        0
## 29      1  0.282  0.718        1
## 724     0  0.958  0.042        0
## 107     1  0.426  0.574        1
## ... (297 rows, 4 cols)
```

```
calculateConfusionMatrix(pred)
```

```
##         predicted
## true       0  1 -err.-
##   0      172 11     11
##   1       40 74     40
##   -err.-  40 11     51
```

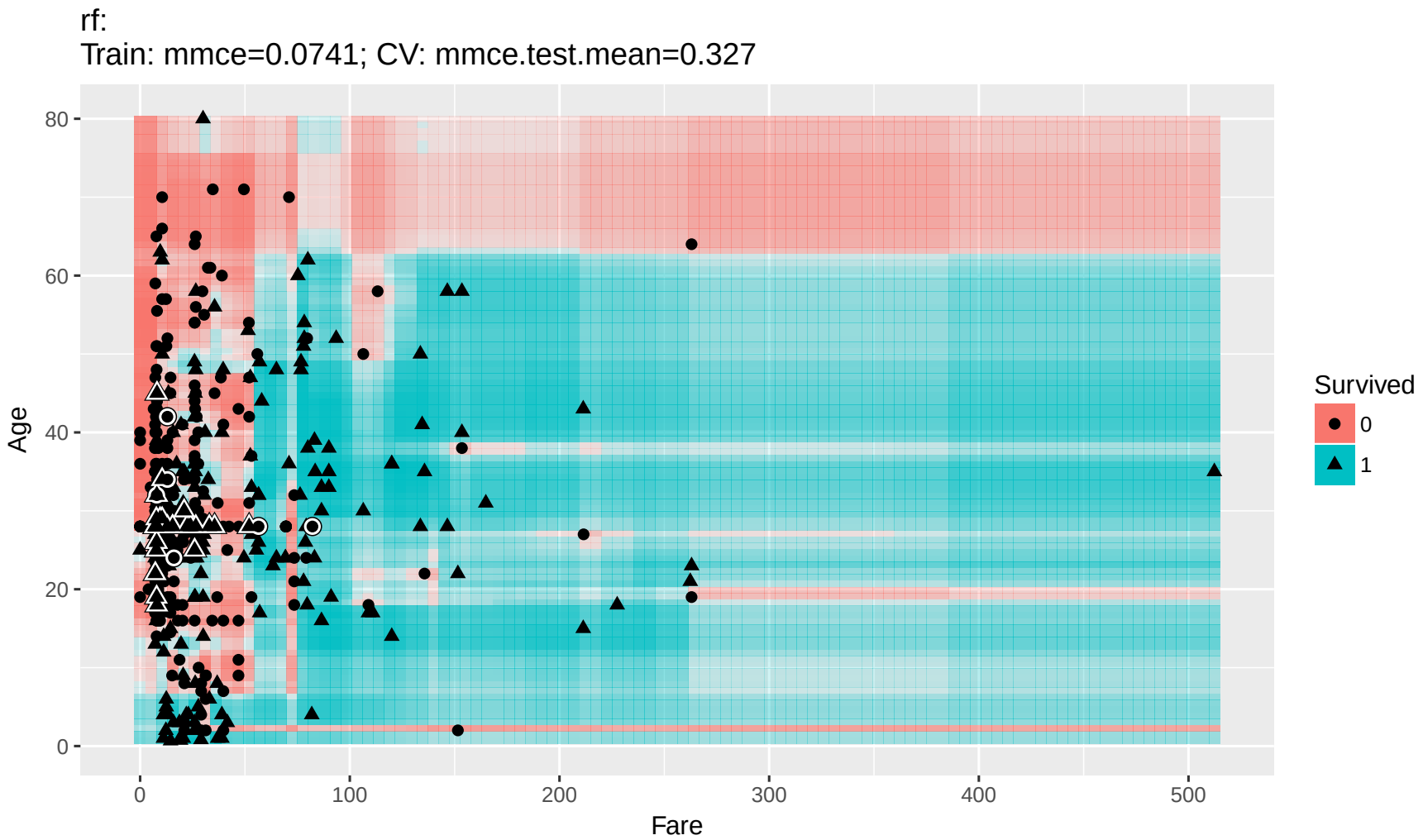
Plot ROC curves

```
ROC.data = generateThreshVsPerfData(pred, measures = list(fpr, tpr, mmce))
plotROCCurves(ROC.data)
```



Plot learner prediction

```
plotLearnerPrediction(lrn, task, features = c("Fare", "Age"))
```



So, are we happy now?

- One consistent UI makes coding a lot easier
- However, `CODING` is still necessary!
- Things are still complicated for users not affine to coding!

$\Rightarrow$ User should be able to do machine learning without ONE SINGLE LINE of code!

The shiny package for R

shiny: is an **R** package which makes it easy to build interactive web applications (apps) straight from R!

Provides tools to create a **GUI** (Graphical User Interface) and consists of two parts

server	ui	
your normal R-code which creates all objects of your app	**input:** sliders, buttons, texts, numeric values, ...	**output:** boxes, tables, plots, ...

There exist many extension packages:

- **shinydashboard** for nice dashboards
- **shinyjs** for easily adding Java Script functionality

The shinyMlr App

The shinyMlr App / 2

- started $\sim$1 year ago as a consultancy project which was part of our studies
- core developers: Florian Fendt and me
- GUI for the **mlr** package
- Includes major machine learning functionalities:
  - Data import
  - Data exploration and preprocessing
  - Creating regression or classification tasks
  - Making use of any **mlr** learner
  - Tuning of learner hyper parameters
  - Training and predicting a model
  - Benchmark experiments with different learners and measures
  - Many visualisations

The shinyMlr App / 3

Availability and installation:

- **Windows:** start the App directly from github:

```
shiny::runGitHub('mlr-org/shinyMlr', 'YourGithubAcountName')
```

- **Linux and Mac:** simple installation via `install_github`

```
devtools::install_github("mlr-org/shinyMlr/package")
runShinyMlr()
```

The shinyMlr App / 4

LIVE DEMO with Kaggle's Titanic Data

Alternatives

Other ML/Data Mining Apps:

- rattle (R)
- Weka (Java)
- KNIME (Java)
- Orange (Python)
- MLJAR (Python)

Alternatives / 2

Advantages of shinyMlr:

- completely free
- open source
- easy extendable
- based on a well known and solid framework (mlr)

Thank you very much for your attention!

Detailed tutorials on our YouTube channel!

If you like our project, follow and star us on github:

`https://github.com/mlr-org/shinyMlr`