SlideR01

Table of contents

おすすめ教科書

- Advanced R
  - 非常に簡潔かつわかりやすく、“プログラミング言語” としての R を解説
  - 仕事でデータ分析をする可能性があるのであれば、学生時代に読むことを強くお勧め
  - Python などにも共通する解説
- より一般的な解説書として、R for Data Science

例題

- object の複製を活用し、defaul と浅い木を推定したい

```
library(mlr3verse)

Tree <- lrn("regr.rpart")

ShallowTree <- Tree

ShallowTree$param_set$values$maxdepth <- 2
```

例題

```
Tree
```

```
<LearnerRegrRpart:regr.rpart>: Regression Tree
* Model: -
* Parameters: xval=0, maxdepth=2
* Packages: mlr3, rpart
* Predict Types:  [response]
* Feature Types: logical, integer, numeric, factor, ordered
* Properties: importance, missings, selected_features, weights
```

- “勝手に変わった!!!”

R の基本文法

```
a <- 1                                                                    ①
```

① “1 を a という名前で保存”

- 正確には、“メモリ-上のどこか (Location) に保存された 1 を、a という Reference(参照名) で保存”

Copy

- 一般に複製 (copy) には、複数の意味がある
- 参照名の複製と object の複製を区別することは、しばしば重要
  - mlr3 のような object 志向では非常に重要

Location と Reference (参照名)

```
a <- 1

b <- a                                                    ①

lobstr::obj_addr(a)                                       ②

lobstr::obj_addr(b)
```

① a を複製?

```
[1] "0x141476a00"
[1] "0x141476a00"
```

- 同じ object に、異なる参照名 (a,b) を付与している” だけ”

2. lobstr パッケージを局所的にロード

Copy on modification

- object に操作を加えると、object そのものが異なる location にコピーされる

```
a <- 1

b <- a

b <- b + 1                                                ①

lobstr::obj_addr(a)

lobstr::obj_addr(b)
```

① deep copy

```
[1] "0x13469c400"
[1] "0x133ca5408"
```

まとめ

- R のほとんどの package において、copy-on-modification が default

  - object を操作した場合、異なる location に複製 (deep copy) を自動で行う
- プログラムに詳しくなくても、予期せぬ挙動を抑止できる
  - 不要な deep copy が行われてしまい、計算速度が犠牲になる場合も、、、
- R6 class を採用している package(mlr3 など) は例外

mlr3 objects

```
library(mlr3verse)

Tree <- lrn("regr.rpart")

Tree1 <- Tree

lobstr::obj_addr(Tree)
```

```
[1] "0x136847eb0"
```

```
lobstr::obj_addr(Tree1)
```

```
[1] "0x136847eb0"
```

mlr3 objects

```
Tree <- lrn("regr.rpart")

Tree1 <- Tree

Tree1$param_set$values$maxdepth <- 2                                    ①

Tree$param_set$values$maxdepth

Tree1$param_set$values$maxdepth
```

① ハードコピーが行われない!!!

```
[1] 2
[1] 2
```

mlr3 objects

```
lobstr::obj_addr(Tree)
```

```
[1] "0x144d5e468"
```

```
lobstr::obj_addr(Tree1)
```

```
[1] "0x144d5e468"
```

- object 自体は複製されない!!!

clone 関数 (deep copy)

```
Tree <- lrn("regr.rpart")

Tree1 <- Tree$clone()                                                    ①

lobstr::obj_addr(Tree)

lobstr::obj_addr(Tree1)
```

① ハードコピー

```
[1] "0x146196f98"
[1] "0x146b79298"
```

Method chain

- Object oriented programming
  - object にデータと関数 (method) を格納
  - method により、object 内データを操作
  - R では$で呼び出す
- Method chain: method を$で繋げることで、逐次的な処理が可能
  - 慣れると、code の読みやすさが改善
  - 関数志向における pipe 演算子に似た利点

例

```
library(tidyverse)

Data <- read_csv("Public/Example.csv")

Task <- as_task_regr(
  Data,
  "Price"
)

EstTree <- Tree$clone()$train(Task)$model # 複製/推定ずみ決定木を保存
```