Pakiet `mlr` jako implementacja procesu uczenia maszynowego

Wykład 10

Maj 2020

Outline

Tworzenie modelu uczenia maszynowego

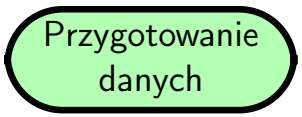


Figure: Tworzenie modelu uczenia maszynowego

Tworzenie modelu uczenia maszynowego

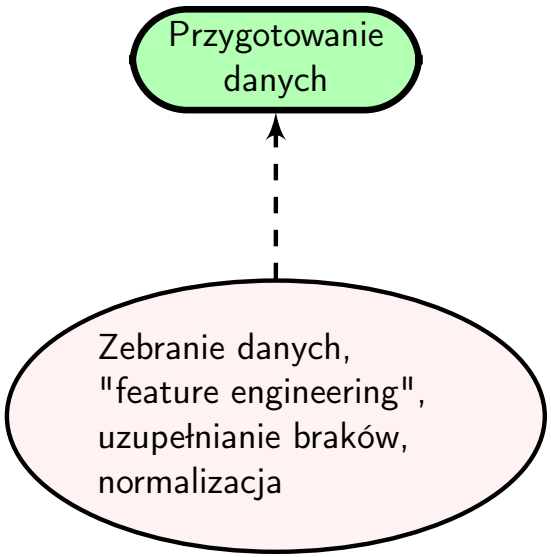


Figure: Tworzenie modelu uczenia maszynowego

Tworzenie modelu uczenia maszynowego

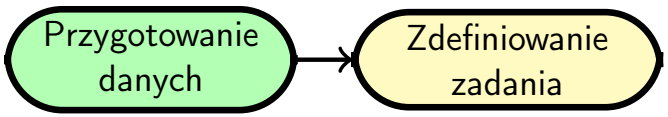


Figure: Tworzenie modelu uczenia maszynowego

Tworzenie modelu uczenia maszynowego

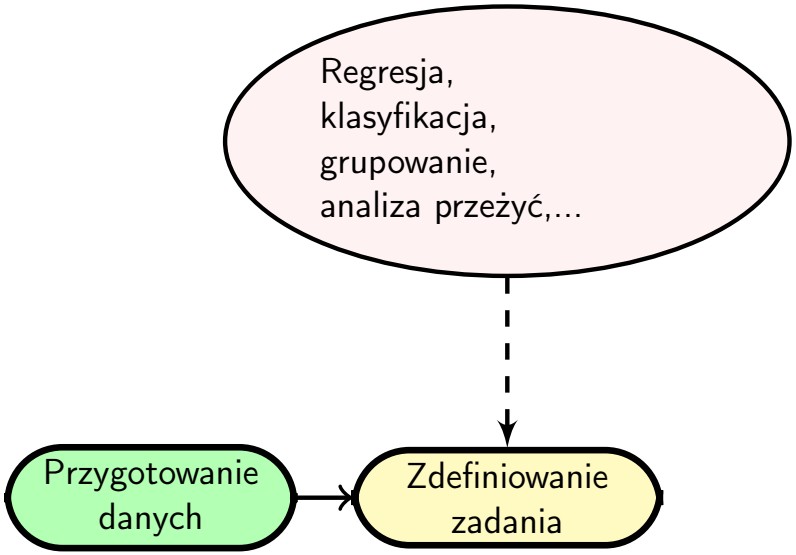


Figure: Tworzenie modelu uczenia maszynowego

Tworzenie modelu uczenia maszynowego

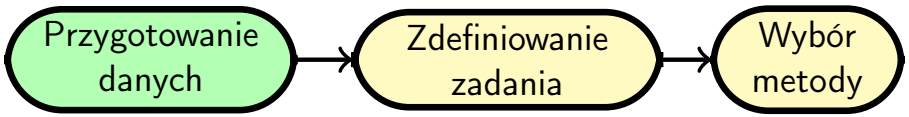


Figure: Tworzenie modelu uczenia maszynowego

Tworzenie modelu uczenia maszynowego

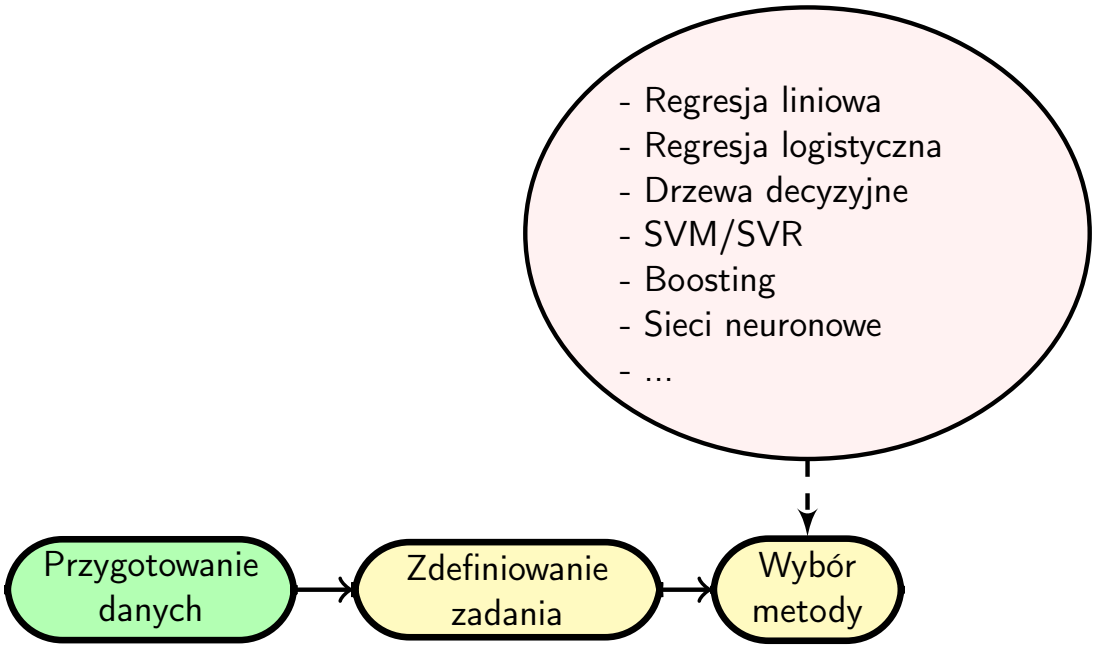


Figure: Tworzenie modelu uczenia maszynowego

Tworzenie modelu uczenia maszynowego

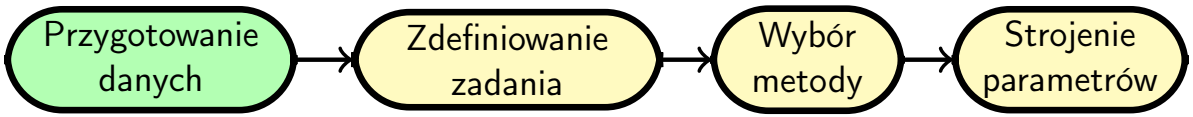


Figure: Tworzenie modelu uczenia maszynowego

Tworzenie modelu uczenia maszynowego

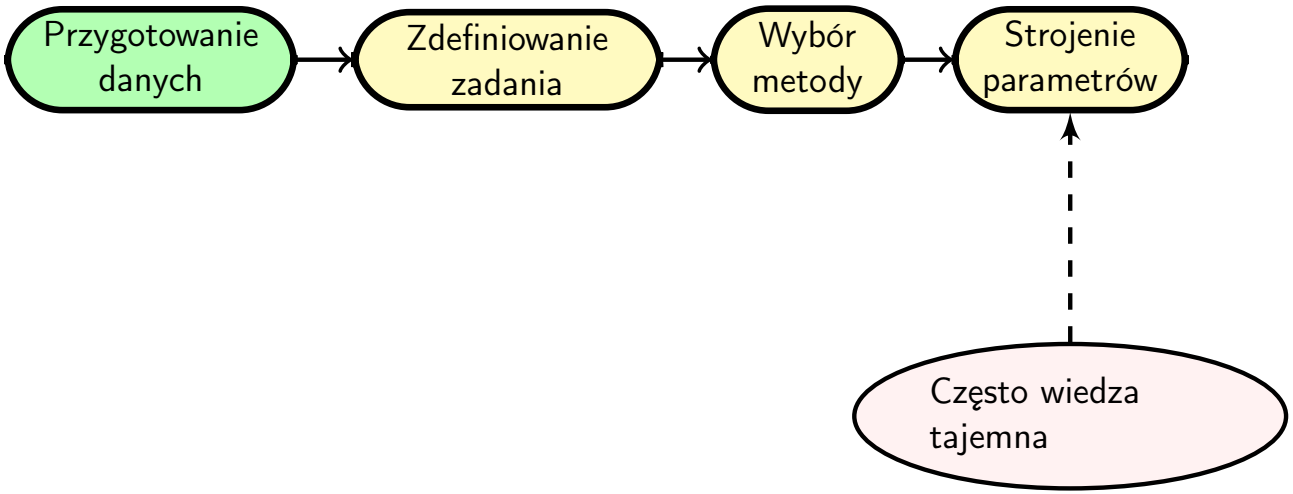


Figure: Tworzenie modelu uczenia maszynowego

Tworzenie modelu uczenia maszynowego

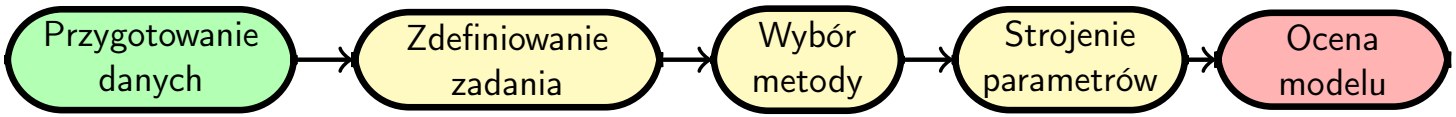


Figure: Tworzenie modelu uczenia maszynowego

Tworzenie modelu uczenia maszynowego

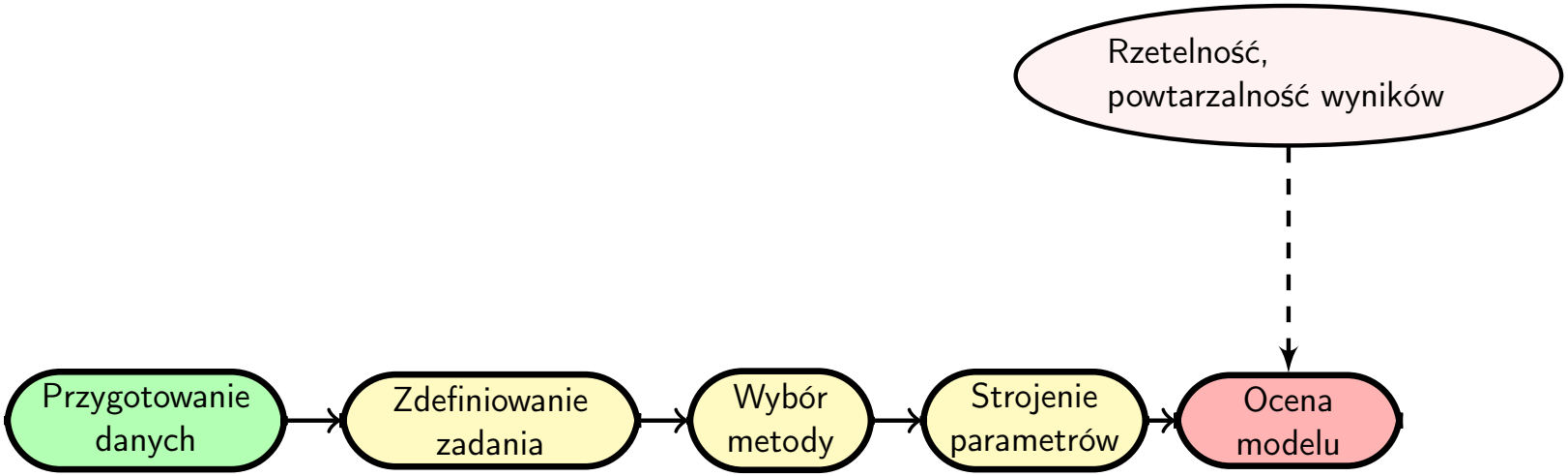


Figure: Tworzenie modelu uczenia maszynowego

Tworzenie modelu uczenia maszynowego

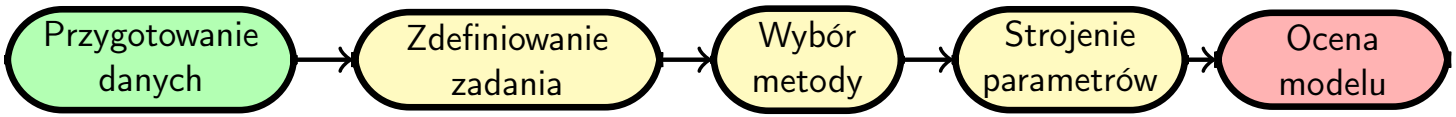


Figure: Tworzenie modelu uczenia maszynowego

Przykłady w R

- ahp - analityczna hierarchizacja
- neuralnet - sieci neoronowe
- C50 - drzewa decyzyjne, reguły klasyfikacyjne
- mlr - ?

Sprawdzian krzyżowy I

Definition

Sprawdzian krzyżowy (walidacja krzyżowa, kroswalidacja, sprawdzanie krzyżowe) – metoda statystyczna, polegająca na podziale próby statystycznej na podzbiory, a następnie przeprowadzaniu wszelkich analiz na niektórych z nich (zbiór uczący), podczas gdy pozostałe służą do potwierdzenia wiarygodności jej wyników (zbiór testowy, zbiór walidacyjny).

Teoria sprawdzianu krzyżowego została zapoczątkowana przez Seymoura Geissera. Pozwala ona bronić się przed tzw. błędem trzeciego rodzaju i właściwie ocenić trafność prognostyczną modelu predykcyjnego. Bez jej zastosowania nie można być pewnym czy model będzie dobrze działał dla danych, które nie były wykorzystywane do jego konstruowania (overfitting).

Sprawdzian krzyżowy II

Overfitting

Nadmierne dopasowanie, przeuczenie, przetrenowanie, overfitting – różne, stosowane w statystyce nazwy tego samego zjawiska, zachodzącego gdy model statystyczny ma zbyt dużo parametrów w stosunku do rozmiaru próby na podstawie której był konstruowany.
Absurdalne i fałszywe modele mogą świetnie pasować do danych uczących gdy model ma wystarczającą złożoność, jednak będą dawały gorsze wyniki, gdy zastosujemy je do danych, z którymi się nie zetknęły podczas uczenia.

Sprawdzian krzyżowy III

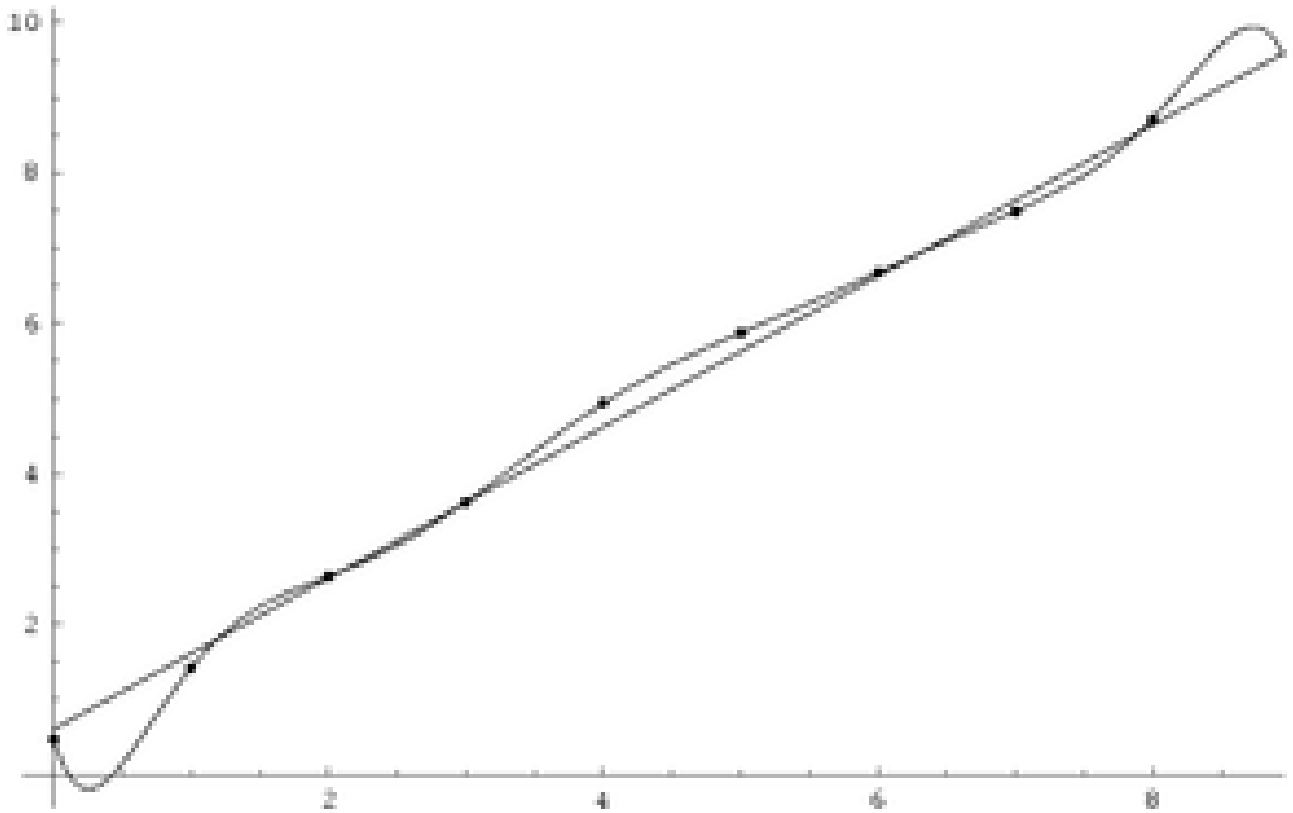


Figure: Overfitting

Rodzaje CV

Sprawdzian krzyżowy IV

- Prosta walidacja. Jest to najbardziej typowy rodzaj walidacji, w którym próbę dzieli się losowo na rozłączne zbiory: uczący i testowy. Zwykle zbiór testowy stanowi mniej niż 1/3 próby. Niektórzy nie zaliczają tego typu walidacji do metody sprawdzianu krzyżowego.
- K-krotna walidacja (k-fold validation, cv3, cv5, cv10 etc.). W tej metodzie, oryginalna próba jest dzielona na K podzbiorów. Następnie kolejno każdy z nich bierze się jako zbiór testowy, a pozostałe razem jako zbiór uczący i wykonuje analizę. Analiza jest więc wykonywana K razy. K rezultatów jest następnie uśrednianych (lub łączonych w inny sposób) w celu uzyskania jednego wyniku.
- Leave-one-out. Jest to odmiana walidacji K-krotnej, gdy N-elementowa próba jest dzielona na N podzbiorów, zawierających po jednym elemencie. Stosowana często dla małych zbiorów danych.

Sprawdzian krzyżowy V

- Kroswalidacja stratyfikowana. Nie jest to w zasadzie osobna odmiana kroswalidacji, a odnosi się do wszystkich jej rodzajów wymienionych powyżej. Kroswalidacja stratyfikowana (ang. stratified cross-validation) polega na takim podziale obiektów pomiędzy zbiór treningowy i zbiór testowy, aby zachowane były oryginalne proporcje pomiędzy klasami decyzyjnymi. Zastosowanie kroswalidacji stratyfikowanej jest szczególnie ważne w przypadku, gdy w oryginalnym zbiorze danych występują znaczne dysproporcje w liczebności przykładów należących do poszczególnych klas decyzyjnych.

Sprawdzian krzyżowy VI

Przykład implementacji k-krotnej walidacji w R

```
k <- 10
for(i in 1:k){
  index <- sample(1:nrow(data),round(0.9*nrow(data)))
  train.cv <- scaled[index,]
  test.cv <- scaled[-index,]
  nn <- neuralnet(f,data=train.cv,hidden=c(18),
                                  linear.output=T)
  pr.nn <- compute(nn,test.cv[,-28]) #without y15
  pr.nn <- pr.nn$net.result*(max(data$y15)-min(data$y15))
                    +min(data$y15) #skalowanie
  test.cv.r <- (test.cv$y15)*(max(data$y15)-min(data$y15))
                    +min(data$y15)  # skalowanie
  cv.error[i] <- sum((test.cv.r - pr.nn)^2)/nrow(test.cv)
}
```

Pakiet mlr jako implementacja procesu uczenia maszynowego I

Dlaczego mlr?

```
library(randomForest)
randomForest(formula, data=NULL, ..., subset, na.action=na.fail
```

Work flow

- Przygotowanie danych (preprocessing)
- Zadanie (task)
- Wybór modelu (benchmark)
- Strojenie parametrów (tuning)
- Ocena modelu

Dane I

```
library(mlr)
library(dplyr)
library(ggplot2)
mieszkania <- read.csv("~/Dokumenty/Projekty/eRementarz4/
                        mieszkania_wroclaw_ceny.csv")
head(mieszkania)
```

	n_pokoj	metraz	cena_m2	rok	pietro	pietro_maks	dzielnica
1	3	89.00	5270	2007	2	2	Krzyki
2	4	163.00	6687	2002	2	2	Psie Pole
3	3	52.00	6731	1930	1	2	Srodmiescie
4	4	95.03	5525	2016	1	2	Krzyki
5	4	88.00	5216	1930	3	4	Srodmiescie
6	2	50.00	5600	1915	3	4	Krzyki

Przygotowanie danych

Typowe zadania:

- standaryzacja danych - `normalizeFeatures`
- łączenie mało licznych poziomów zmiennych jakościowych - `mergeSmallFactorLevels`
- wybranie części obserewacji - subsetTask
- imputacja danych brakujących - `impute`
- i inne...

Zadania (task) I

Obsługiwane klasy problemów

`makeClassifTask()`
`makeRegrTask()`
`makeClusterTask()`
`makeCostSensTask()`
`makeMultilabelTask()`
`makeSurvTask()`

Nasz problem I

```
m2_task <- makeRegrTask(id = "mieszkanie",
data = mieszkania,
target = "cena_m2")
```

Alternatywnie

```
m2_task_ndz <- makeRegrTask(id = "mieszkanie_ndz",
data = select(mieszkania, -dzielnica),
target = "cena_m2")
```

Uwaga

- `fixup.data = "warn"`: czyszczenie danych (aktualnie tylko usuwanie pustych poziomów)
- `check.data = TRUE`: sprawdzanie poprawności danych (aktualnie: NA i puste poziomy zmiennej odpowiedzi)

Metody uczenia (learner) I

Ogromna liczba dostępnych metod

```
listLearners(obj = "regr")[1:6, c(1, 3:4)]
```

```
  class              short.name  package
1 regr.cforest       cforest     party
2 regr.ctree         ctree       party
3 regr.cvglmnet      cvglmnet    glmnet
4 regr.featureless   featureless mlr
5 regr.gausspr       gausspr     kernlab
6 regr.glm           glm         stats
```

- Jak radzi sobie zwykła regresja liniowa?

  ```
  reg_lm <- makeLearner("regr.lm")
  ```

- A jak inne popularne metody?

Metody uczenia (learner) II

```
reg_rf <- makeLearner("regr.randomForest")
reg_nnet <- makeLearner("regr.nnet")
```

- Inaczej:

```
lrns <- makeLearners(c("lm", "randomForest", "nnet"),
type = "regr")
```

- takie wywołania tworzą obiekty typu `Learner`
- metody są zaimplementowane w odpowiednich pakietach - mlr jest nakładką
- różne metody - różne wsparcie dla brakujących wartości, wag itd
- Uwaga: w ten sposób wszystkie hiperparametry mają ustawione wartości domyślne

Metody uczenia (learner) III

Informacje o metodzie

```
getLearnerProperties(reg_rf)
```

```
[1] "numerics" "factors" "ordered" "se" "oobpreds" "featimp"
```

```
getLearnerParamSet(reg_rf)
```

```
 Type len Def Constr Req
ntree integer - 500 1 to Inf -
se.ntree integer - 100 1 to Inf Y
se.method discrete - jackknife bootstrap,jackknife,sd Y
se.boot integer - 50 1 to Inf -
mtry integer - - 1 to Inf -
replace logical - TRUE - -
```

Metody uczenia (learner) IV

```
strata untyped - - - -
sampsize integervector <NA> - 1 to Inf -
nodesize integer - 5 1 to Inf -
maxnodes integer - - 1 to Inf -
importance logical - FALSE - -
localImp logical - FALSE - -
nPerm integer - 1 -Inf to Inf -
proximity logical - FALSE - -
oob.prox logical - - - Y
do.trace logical - FALSE - -
keep.forest logical - TRUE - -
keep.inbag logical - FALSE - -
 Tunable Trafo
ntree TRUE -
se.ntree TRUE -
se.method TRUE -
```

Metody uczenia (learner) V

```
se.boot TRUE -
mtry TRUE -
replace TRUE -
strata FALSE -
sampsize TRUE -
nodesize TRUE -
maxnodes TRUE -
importance TRUE -
localImp TRUE -
nPerm TRUE -
proximity FALSE -
oob.prox FALSE -
do.trace FALSE -
keep.forest FALSE -
keep.inbag FALSE -
```

Metody uczenia (learner) VI

Ustawianie hiperparametrów

- przy tworzeniu learnera:

```
reg_rf2 <- makeLearner("regr.randomForest",
             par.vals = list(ntree = 1000))
```

  po utworzeniu learnera:

```
reg_rf2 <- setHyperPars(reg_rf, ntree = 1000)
```

- `getHyperPars(reg_rf2)`

```
$ntree
[1] 1000
```

Outline

Porównywanie modeli I

Porównywanie modeli

```
porownanie <- benchmark(learners = list(reg_lm, reg_rf, reg_nne
tasks = list(m2_task, m2_task_ndz),
resampling = cv5)
save(porownanie, file = "porownanie.rda")
```

```
load("porownanie.rda")
porownanie
```

```
  task.id          learner.id          mse.test.mean
1 mieszkanie       regr.lm             2084932
2 mieszkanie       regr.randomForest   1119241
3 mieszkanie       regr.nnet           2674150
4 mieszkanie_ndz   regr.lm             2481207
5 mieszkanie_ndz   regr.randomForest   1748378
```

Porównywanie modeli II

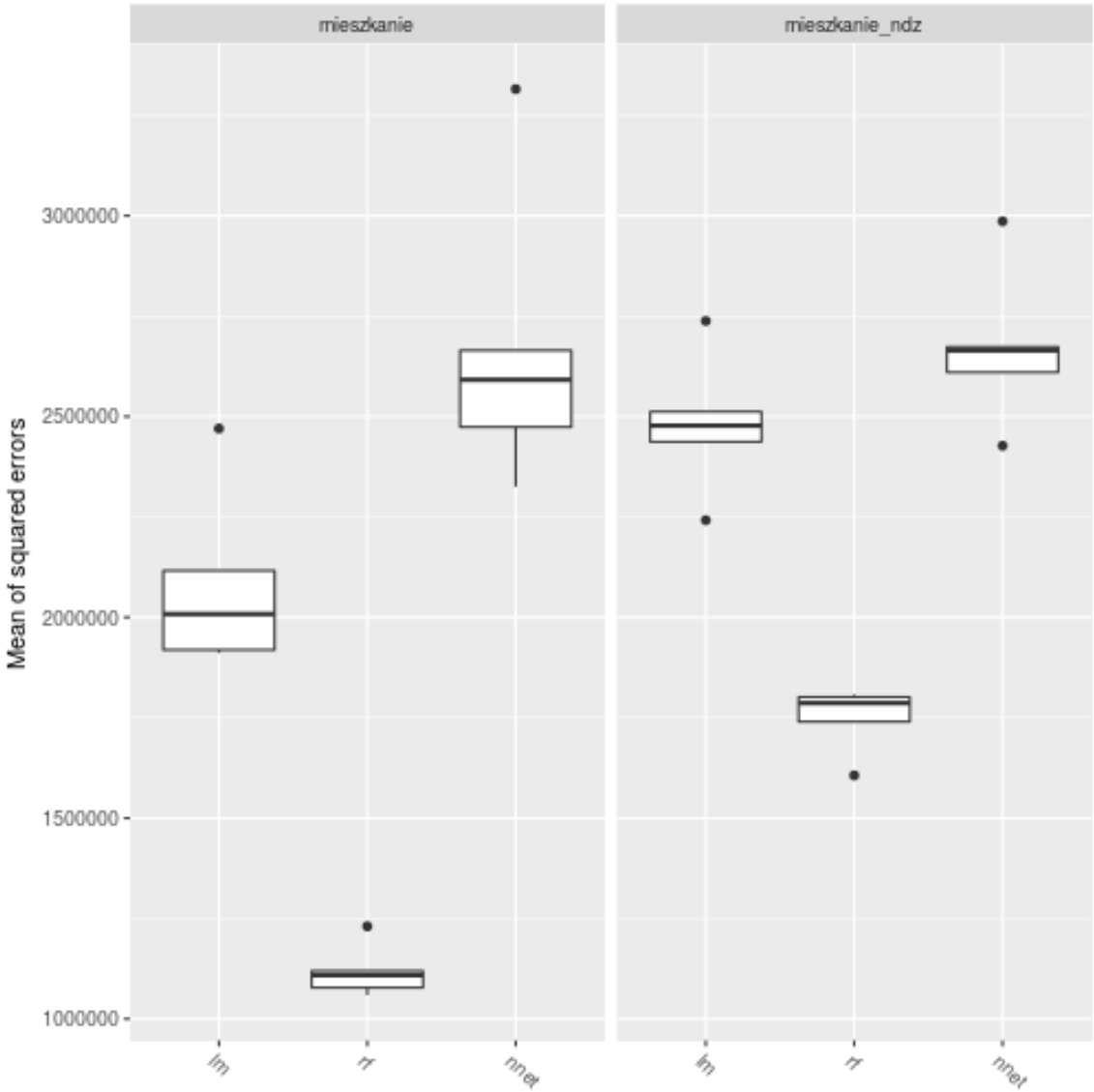
```
6 mieszkanie_ndz  regr.nnet          2672237
```

Porównywanie modeli III

Wyniki porównania

```
# getBMRAggrPerformances(porownanie)
# getBMRPerformances(porownanie)
plotBMRBoxplots(porownanie)
```

Porównywanie modeli IV



Porównywanie modeli V

Różne kryteria

```
listMeasures(obj = "regr")
```

```
 [1] "rsq" "rae" "rmsle" "mse" "rrse"
 [6] "medse" "mae" "timeboth" "timepredict" "medae"
[11] "featperc" "mape" "rmse" "kendalltau" "sse"
[16] "arsq" "expvar" "msle" "sae" "timetrain"
[21] "spearmanrho"
```

Outline

Wytrenowanie pojedynczego modelu I

Wytrenowanie pojedynczego modelu

podstawowe wywołanie:

```
m2_rf <- train(reg_rf2, m2_task)
```

uwaga: można samodzielnie zdefiniować zbiór uczący

```
uczacy <- sample(1:nrow(mieszkania), floor(0.7*nrow(mieszkania)
testowy <- setdiff(1:nrow(mieszkania), uczacy)
m2_rf_czesc <- train(reg_rf2, m2_task, subset = uczacy)
```

przewidywane wartości:

Wytrenowanie pojedynczego modelu II

```
pred <- predict(m2_rf, task = m2_task)
head(getPredictionResponse(pred))
[1] 5353.419 6598.072 6426.211 5378.307 5241.464 5547.251
pred2 <- predict(m2_rf_czesc, newdata = mieszkania[testowy, ])
head(getPredictionResponse(pred2))
[1] 5383.544 6566.714 5296.223 4756.715 7551.108 5929.577
```

Outline

Strojenie parametrów I

Strojenie parametrów

```
all_params <- makeParamSet(
makeDiscreteParam("mtry", values = 2:5),
makeDiscreteParam("nodesize", values = seq(5, 45, by = 10))
)
m2_params <- tuneParams(reg_rf2, task = m2_task,
resampling = cv3,
par.set = all_params,
control = makeTuneControlGrid())
```

Strojenie parametrów II

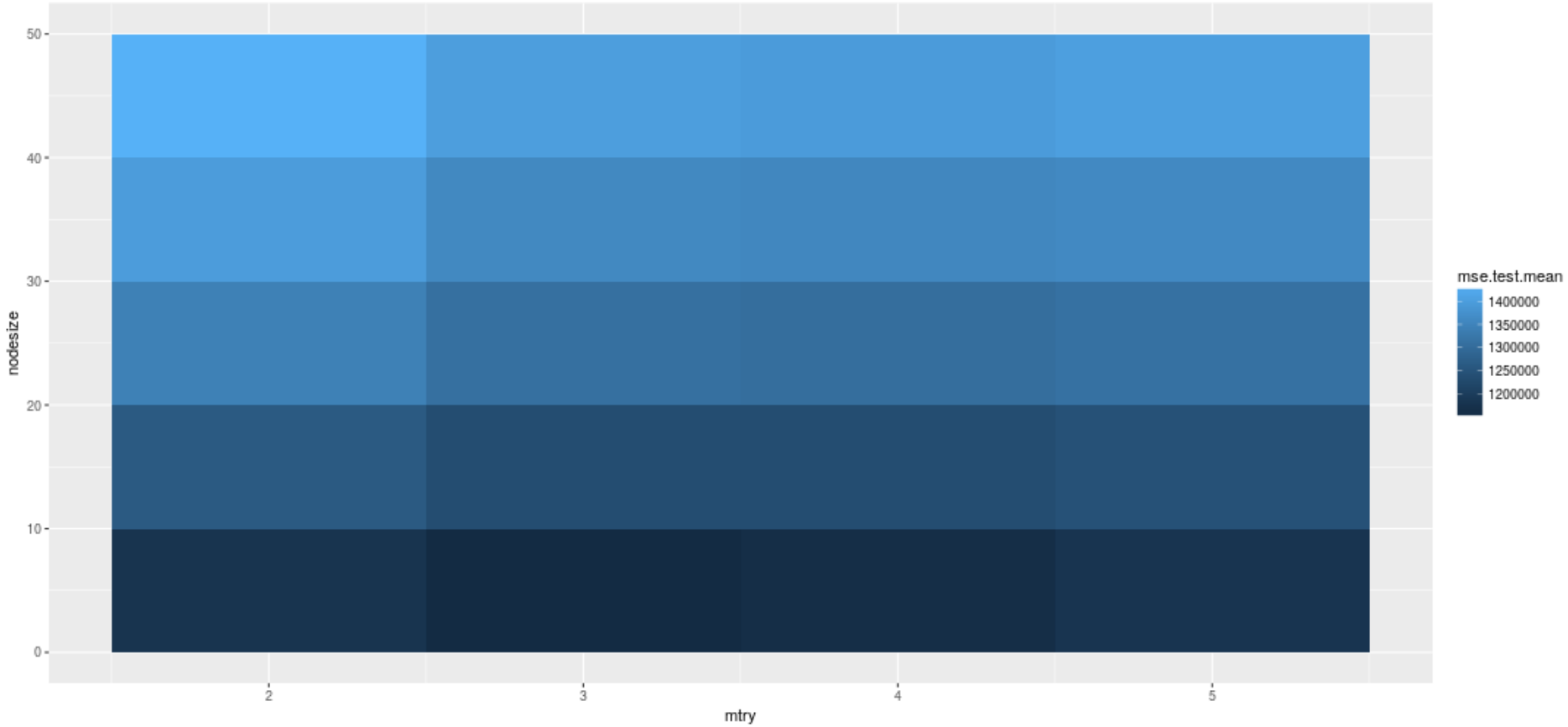
Wynik

```
m2_params

Tune result:
Op. pars: mtry=3; nodesize=5
mse.test.mean=1.15e+06

par_data <- generateHyperParsEffectData(m2_params)
plotHyperParsEffect(par_data, x = "mtry", y = "nodesize", z = "
                    plot.type = "heatmap")
```

Strojenie parametrów III




```
reg_rf2 <- setHyperPars(reg_rf2, mtry = 3)
```

```
m2_rf2 <- train(reg_rf2, m2_task)
```

Strojenie parametrów IV

Cena mieszkania

```
moje <- data.frame(n_pokoj = 3L,
metraz = 60.00,
rok = 2010L,
pietro = 3L,
pietro_maks = 5L,
dzielnica = "Srodmiescie")
moje$dzielnica <- factor(moje$dzielnica,
levels = levels(mieszkania$dzielnica))
predict(m2_rf2, newdata = moje)

Prediction: 1 observations
predict.type: response
threshold:
time: 0.09
```

Strojenie parametrów V

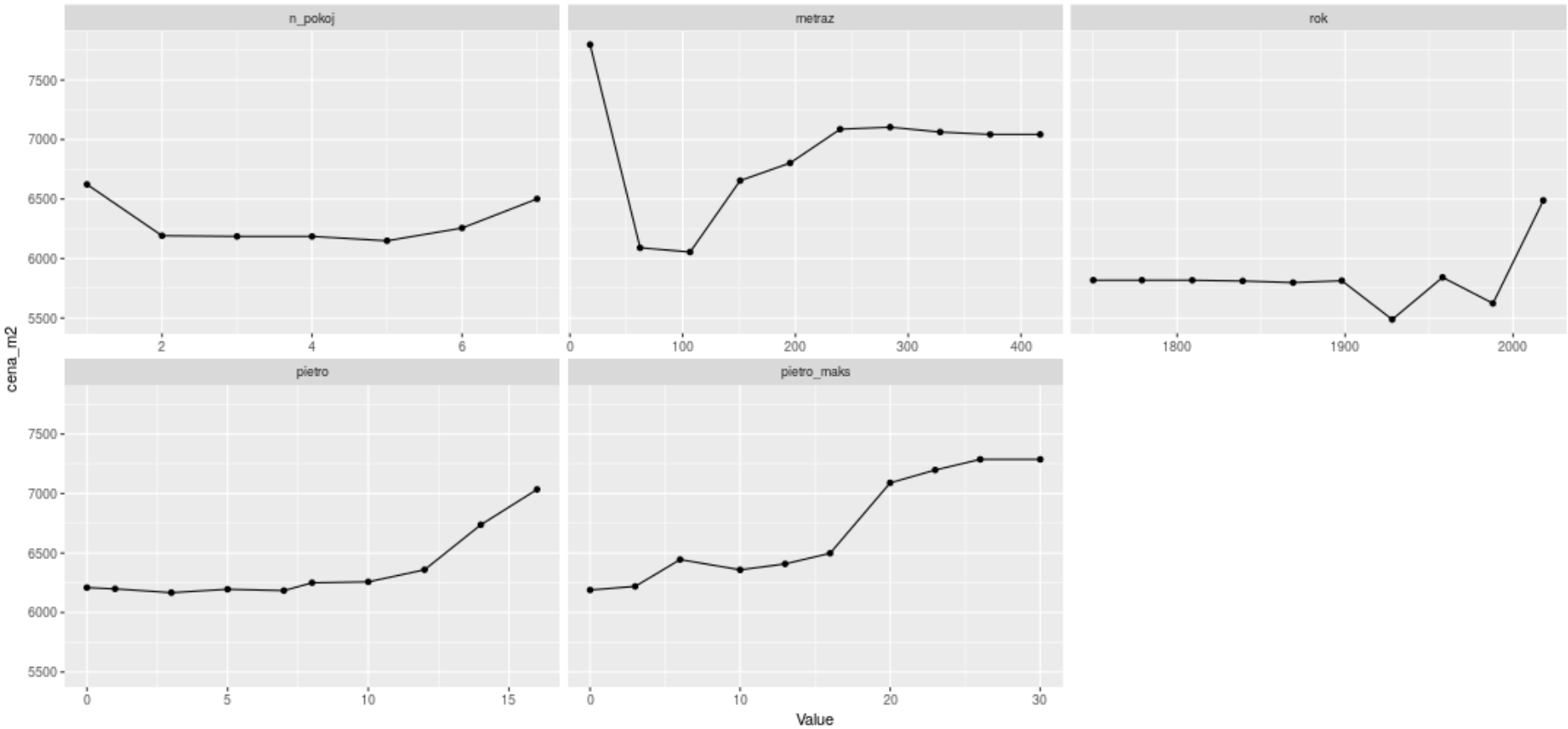
```
  response
1 7503.985
```

Wizualizacja modelu I

Wizualizacja modelu

```
pdp <- generatePartialDependenceData(m2_rf2,
m2_task,
features = colnames(mieszkania)[-c(3, 7)])
plotPartialDependence(pdp)}
```

Wizualizacja modelu II



Outline

Pakiet mlr jako implementacja procesu uczenia maszynowego I

Przykład iris

```
library(mlr)
data(iris)
## Define the task
task = makeClassifTask(id = "tutorial", data = iris,
                       target = "Species")

## Define the learner
lrn = makeLearner("classif.lda")

## Define the resampling strategy
rdesc = makeResampleDesc(method = "CV", stratify = TRUE)
```

Pakiet mlr jako implementacja procesu uczenia maszynowego II

Przykład iris

```
## Do the resampling
r = resample(learner = lrn, task = task, resampling = rdesc,
             show.info = FALSE)

## Get the mean misclassification error
r$aggr
#> mmce.test.mean
#> 0.02
```

Pakiet mlr jako implementacja procesu uczenia maszynowego I

Przykład klasyfikacji złamania kości

```
library(mlr)

df <- read.table("Dane.csv", na.strings=c("", "NA"),
                 header = TRUE, dec=",", sep = ";")

df <- df[,!(names(df) %in% c("last_name","first_name",
                             "BirthTme","DateOfExam",
                             "Gluc","LF","Ca_plus_plus","Mn"))]

df1 <- na.omit(df)

df1[] <- lapply(df1, as.numeric)
```

Pakiet mlr jako implementacja procesu uczenia maszynowego II

Przykład klasyfikacji złamania kości

```
df1$Fractures <- factor(df1$Fractures)

## Define the task
task = makeClassifTask(id = "fractures_classification",
                       data = df1, target = "Fractures")

## Define the learner from package MASS
library(MASS)
lrn = makeLearner("classif.lda")


## Define the resampling strategy
```

Pakiet mlr jako implementacja procesu uczenia maszynowego III

Przykład klasyfikacji złamania kości

```
rdesc = makeResampleDesc(method = "CV", stratify = TRUE)

## Do the resampling
r = resample(learner = lrn, task = task, resampling = rdesc,
             show.info = FALSE)

## Get the mean misclassification error
r$aggr
#> mmce.test.mean
#> 0.02


lrns <- makeLearners(c("lda","rpart", "C50","rFerns",
```

Pakiet mlr jako implementacja procesu uczenia maszynowego IV

Przykład klasyfikacji złamania kości

```
                        "randomForestSRC"), type = "classif")
porownanie <- benchmark(learners = lrns,
                        tasks = task,
                        resampling = cv5)#reper
save(porownanie, file = "porownanie.rda")
load("porownanie.rda")
porownanie

plotBMRBoxplots(porownanie)
```

Wykaz literatury podstawowej

1. Drew Conway, John Myles White, Uczenie maszynowe dla programistów, Helion, 2015
2. Peter Flach, Machine Learning: The Art and Science of Algorithms that Make Sense of Data, 2012
3. Rob Kabacoff. R in Action. Manning, 2010
4. Paul Teetor, 25 Recipes for Getting Started with R, 2011
5. John Verzani, Getting Started with RStudio, 2011
6. Ian H. Witten, Eibe Frank, Mark A. Hall, Data Mining: Practical Machine Learning Tools and Techniques, 2011

Wykaz literatury uzupełniającej:

1. Szeliga Marcin, Data Science i uczenie maszynowe, Wydawnictwo Naukowe PWN, 2017, ISBN: 9788301192327
2. Kirk Matthew, Python w uczeniu maszynowym. Podejście sterowane testami, Wydawnictwo: APN PROMISE, 2018, ISBN: 9788375413571