Kombajn do modelowania w R - pakiet MLR w pigułce

Data Science Warsaw, 2017.11.14

dr inż. Paweł Zawistowski

Instytut Informatyki, WEiTI, PW / Adform

Źródło obrazu: https://cdn.pixabay.com/photo/2013/06/16/13/41/cotton-harvester-139651_1280.jpg

Plan prezentacji

- Do czego służy i co potrafi pakiet MLR?
- Przygotowanie danych/sformułowanie zadania.
- Uczenie modelu, strojenie parametrów.
- Wizualizacje i rozszerzanie MLR.

Do czego służy i co potrafi pakiet MLR?

Źródło obrazu: https://cdn.pixabay.com/photo/2013/06/16/13/41/cotton-harvester-139651_1280.jpg

Tworzenie modelu z lotu ptaka

Tworzenie modelu z lotu ptaka

Tworzenie modelu z lotu ptaka

Tworzenie modelu z lotu ptaka

Tworzenie modelu z lotu ptaka

Tworzenie modelu z lotu ptaka

Przygotowanie danych

Zdefiniowanie zadania

Tworzenie modelu z lotu ptaka

Tworzenie modelu z lotu ptaka

Przygotowanie danych

Zdefiniowanie zadania

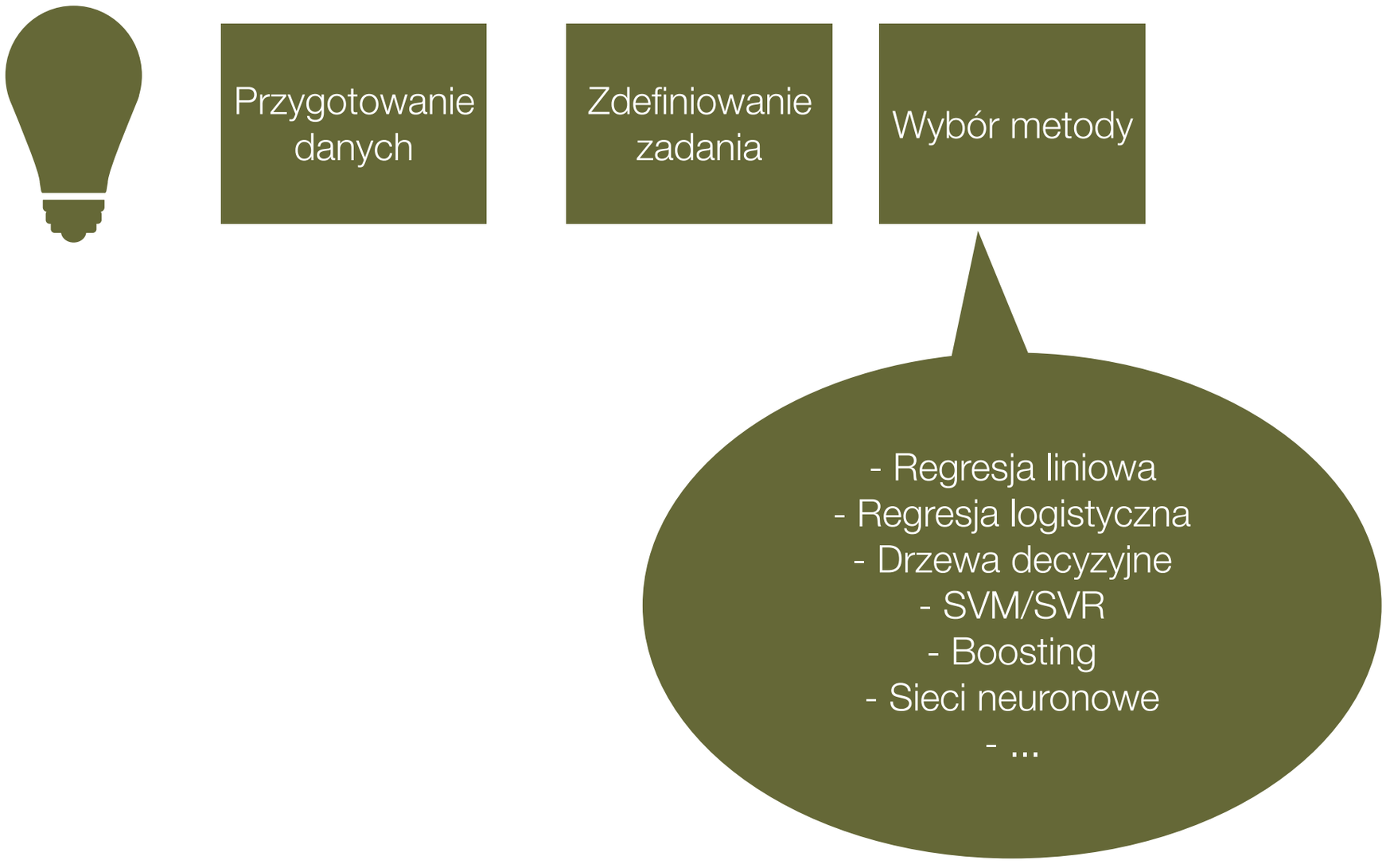
Tworzenie modelu z lotu ptaka

Przygotowanie danych

Zdefiniowanie zadania

Wybór metody

Tworzenie modelu z lotu ptaka



Tworzenie modelu z lotu ptaka

Przygotowanie danych

Zdefiniowanie zadania

Wybór metody

Tworzenie modelu z lotu ptaka

Przygotowanie danych

Zdefiniowanie zadania

Wybór metody

Strojenie parametrów

Tworzenie modelu z lotu ptaka

Tworzenie modelu z lotu ptaka

Przygotowanie danych

Zdefiniowanie zadania

Wybór metody

Strojenie parametrów

Tworzenie modelu z lotu ptaka

Przygotowanie danych

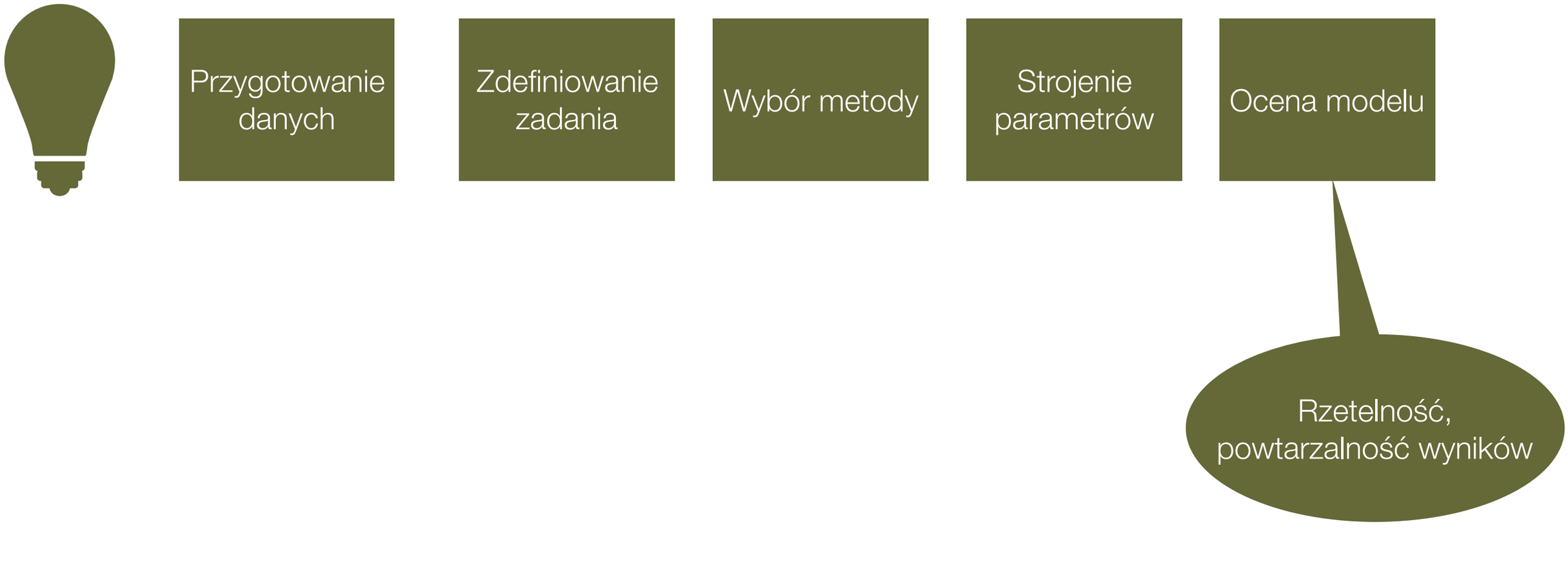
Zdefiniowanie zadania

Wybór metody

Strojenie parametrów

Ocena modelu

Tworzenie modelu z lotu ptaka



Tworzenie modelu z lotu ptaka

Przygotowanie danych

Zdefiniowanie zadania

Wybór metody

Strojenie parametrów

Ocena modelu

Tworzenie modelu z lotu ptaka

MLR **ułatwia modelowanie** w R
ujednolicając sposób korzystania
z innych pakietów.

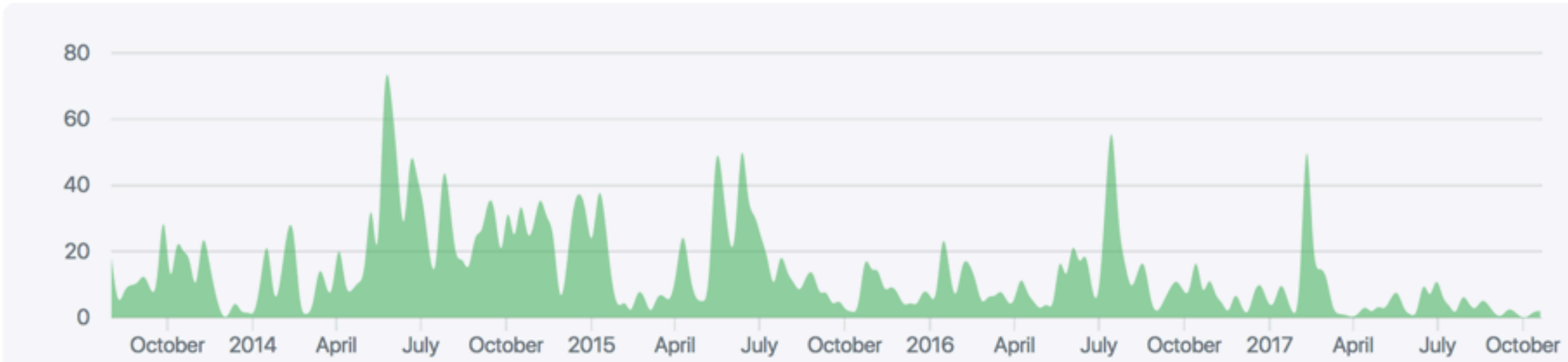
mlR : Machine Learning in R

- Pierwszy commit sierpień 2013.
- > 3800 commits
- 49 osób ma swoje kontrybucje

Aug 25, 2013 – Nov 14, 2017

Contributions: **Commits**

Contributions to master, excluding merge commits



Co potrafi MLR?

Motywujący przykład

```
library(mlr); library(tidyverse); data(iris)

parallelMap::parallelStartMulticore(level = 'mlr.resample')
results = makeLearner("classif.knn") %>%
          makeTuneWrapper(resampling = cv3,
                          measures = acc,
                          par.set = makeParamSet(makeIntegerParam("k", lower=1, upper=20)),
                          control = makeTuneControlGrid()) %>%
          resample(task = makeClassifTask(data = iris, target = "Species"),
                   resampling = cv3,
                   measures = list(acc, setAggregation(acc, test.sd)))
parallelMap::parallelStop()
calculateConfusionMatrix(results$pred)
```

Motywujący przykład

```
library(mlr); library(tidyverse); data(iris)

parallelMap::parallelStartMulticore(level = 'mlr.resample')
results = makeLearner("classif.knn") %>%
          makeTuneWrapper(resampling = cv3,
                          measures = acc,
                          par.set = makeParamSet(makeIntegerParam("k", lower=1, upper=20)),
                          control = makeTuneControlGrid()) %>%
          resample(task = makeClassifTask(data = iris, target = "Species"),
                   resampling = cv3,
                   measures = list(acc, setAggregation(a      tes
parallelMap::parallelStop()
calculateConfusionMatrix(results$pred)
```

Zadanie klasyfikacji

Motywujący przykład

Wybór metody

```
library(mlr); library(                    )

parallelMap::parallelStartMu                (level = 'mlr.resample')
results = makeLearner("classif.knn") %>%
          makeTuneWrapper(resampling = cv3,
                          measures = acc,
                          par.set = makeParamSet(makeIntegerParam("k", lower=1, upper=20)),
                          control = makeTuneControlGrid()) %>%
          resample(task = makeClassifTask(data = iris, target = "Species"),
                   resampling = cv3,
                   measures = list(acc, setAggregation(acc, test.sd)))
parallelMap::parallelStop()
calculateConfusionMatrix(results$pred)
```

Motywujący przykład

```
library(mlr); library(tidyverse); data(iris)

parallelMap::parallelStartMulticore(level = 'mlr.resample')
results = makeLearner("classif.knn") %>%
          makeTuneWrapper(resampling = cv3,
                          measures = acc,
                          par.set = makeParamSet(makeIntegerParam("k", lower=1, upper=20)),
                          control = makeTuneControlGrid()) %>%
        r          k = makeClassifTask(data = iris, target = "Species"),
                 ling = cv3,
                      = list(acc, setAggregation(acc, test.sd)))
para
calc                 sults$pred)
```

Strojenie parametrów

Motywujący przykład

```
library(mlr); library(tidyverse); data(iris)

parallelMap::parallelStartMulticore(level = 'mlr.resamp
results = makeLearner("classif.knn") %>%
          makeTuneWrapper(resampling = cv3,
                          measures = acc,
                          par.set = makeParamSet(makeInteg                er=1, upper=20)),
                          control = makeTuneControlGrid()) %>%
          resample(task = makeClassifTask(data = iris, target = "Species"),
                   resampling = cv3,
                   measures = list(acc, setAggregation(acc, test.sd)))
parallelMap::parallelStop()
calculateConfusionMatrix(results$pred)
```

Walidacja krzyżowa

Walidacja krzyżowa

Motywujący przykład

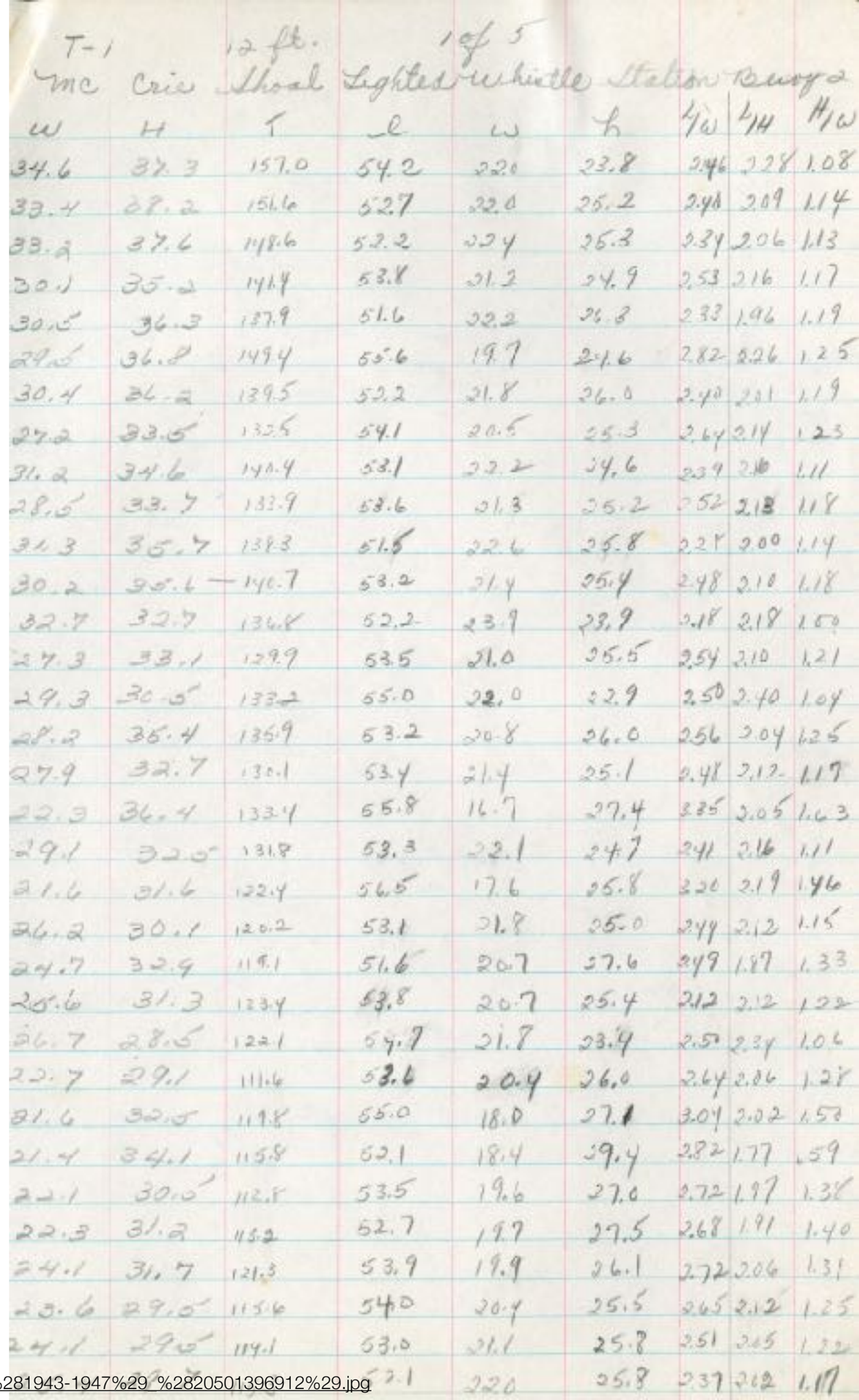
```
library(mlr); library(tidyverse); data(iris)

parallelMap::parallelStartMulticore(level = 'mlr.resample')
results = makeLearner("classif.knn") %>%
          makeTuneWrapper(resampling = cv3,
                          measures = acc,
                          par.set = makeParamSet(makeIntegerParam("k", lower=1, upper=20)),
                          control = makeTuneControlGrid()) %>%
          resample(makeClassifTask(data = iris, target = "Species"),
                   resampling = cv3,
                   extract = list(acc, setAggregation(acc, test.sd)))
parallelMap::parallelStop()
calculateConfusionMatrix(results$pred)
```

Obliczenia Równoległe

Motywujący przykład

```
library(mlr); library(tidyverse); data(iris)

parallelMap::parallelStartMulticore(level = 'mlr.resample')
results = makeLearner("classif.knn") %>%
          makeTuneWrapper(resampling = cv3,
                          measures = acc,
                          par.set = makeParamSet(makeIntegerParam("k", lower=1, upper=20)),
                          control = makeTuneControlGrid()) %>%
          resample(task = makeClassifTask(data = iris, target = "Species"),
                   resampling = cv3,
                   measures = list(acc, setAggregation(acc, test.sd)))
parallelMap::parallelStop()
calculateConfusionMatrix(results$pred)
```

Przygotowanie danych



T-1 1 of 5

Mc W	Cric H	12 ft. Shoal T	Lighted l	Whistle w	Station h	Buoy 4/w	Buoy 4/H	Buoy H/w
34.6	37.3	157.0	54.2	22.0	23.8	2.46	2.28	1.08
33.4	28.2	151.6	52.7	22.0	25.2	2.48	2.09	1.14
33.9	37.6	148.6	52.2	22.4	25.3	2.34	2.06	1.13
30.1	35.2	141.4	53.8	21.2	24.9	2.53	2.16	1.17
30.5	36.3	137.9	51.6	22.2	26.8	2.33	1.96	1.19
29.5	36.8	149.4	55.6	19.7	24.6	2.82	2.26	1.25
30.4	26.2	139.5	52.2	21.8	26.0	2.40	2.21	1.19
27.2	33.5	133.5	54.1	20.5	25.3	2.64	2.14	1.23
31.2	34.6	141.4	53.1	22.2	24.6	2.39	2.16	1.11
28.5	33.7	133.9	53.6	21.3	25.2	2.52	2.13	1.18
31.3	35.7	134.3	51.6	22.6	25.8	2.28	2.00	1.14
30.2	35.6	140.7	53.2	21.4	25.4	2.48	2.10	1.18
32.7	32.7	136.8	52.2	23.9	23.9	2.18	2.18	1.00
27.3	33.1	129.9	53.5	21.0	25.5	2.54	2.10	1.21
29.3	30.5	133.2	55.0	22.0	22.9	2.50	2.40	1.04
28.2	35.4	135.9	53.2	20.8	26.0	2.56	2.04	1.25
27.9	32.7	130.1	53.4	21.4	25.1	2.48	2.12	1.17
22.3	36.4	133.4	55.8	16.7	27.4	3.35	2.05	1.63
29.1	32.0	131.8	53.3	22.1	24.7	2.41	2.16	1.11
21.6	31.6	122.4	56.5	17.6	25.8	3.20	2.19	1.46
26.2	30.1	120.2	53.1	21.8	25.0	2.44	2.12	1.15
24.7	32.9	119.1	51.6	20.7	27.6	2.49	1.87	1.33
25.6	31.3	123.4	53.8	20.7	25.4	2.12	2.12	1.22
26.7	28.5	122.1	54.7	21.8	23.4	2.50	2.34	1.06
22.7	29.1	111.6	53.6	20.4	26.0	2.64	2.86	1.28
21.6	30.5	119.8	55.0	18.0	27.1	3.04	2.02	1.58
21.4	34.1	115.8	52.1	18.4	29.4	2.82	1.77	1.59
22.1	30.0	112.5	53.5	19.6	27.0	2.72	1.97	1.38
22.3	31.2	115.2	52.7	19.7	27.5	2.68	1.91	1.40
24.1	31.7	121.3	53.9	19.9	26.1	2.72	2.06	1.31
23.6	29.5	115.6	54.0	20.4	25.5	2.65	2.12	1.25
24.1	29.5	114.1	53.0	21.1	25.8	2.51	2.05	1.22
			52.1	22.0	25.8	2.37	2.02	1.17

Źródło obrazu: https://upload.wikimedia.org/wikipedia/commons/8/8d/Buoy_fouling_survey%2C_research_data_%2815%29_%281943-1947%29_%2820501396912%29.jpg

Konkretne **zadanie**, wraz z jego danymi, reprezentuje obiekt dziedziczący z klasy **Task**.

Zadania wspierane przez MLR

- `makeClassifTaskmakeMultilabelTask` klasyfikacja binarna/wieloklasowa
- `makeClusterTask` - klastrowanie
- `makeRegrTask` - regresja
- `makeSurvTask` - analiza przeżycia
- `makeCostSensTask` - klasyfikacja z różnymi kosztami pomyłek

Przekształcenia danych

- capLargeValues - usuwanie/zamiana wartości odstających
- createDummyFeatures - “one hot encoding”
- dropFeatures - usuwanie atrybutów
- joinClassLevels - łączenie “małych” klas w większe
- mergeSmallFactorLevels - łączenie rzadko spotykanych wartości atrybutów
- normalizeFeatures - normalizacja atrybutów
- removeConstantFeatures - usuwanie jednowartościowych atrybutów
- subsetTask - usuwanie obserwacji/atrybutów
- selekcja cech
- uzupełnianie brakujących danych

Przykładowe zadanie regresji

```
autoMpgTask <- makeRegrTask(data=autoMpgDf, target = "mpg", id = "auto_mpg") %>%
                  dropFeatures("car_name") %>%
                  createDummyFeatures(cols = "origin") %>%
                  normalizeFeatures(method = "standardize")

summarizeColumns(autoMpgTask)
```

```
Supervised task: auto_mpg
Type: regr
Target: mpg
Observations: 398
Features:
numerics  factors  ordered
       9        0        0
Missings: TRUE
Has weights: FALSE
Has blocking: FALSE
```

Uczenie modelu, strojenie parametrów

Źródło obrazu: https://upload.wikimedia.org/wikipedia/commons/e/e5/Comp._rack_%28Supernatural%29.jpg

Metody uczące

- MLR jest tylko nakładką - implementacje są w osobnych pakietach.

- Reprezentowane przez potomków klasy `Learner`.

```
makeLearner(“classif.randomForest"
   , predict.type = “prob"
   , fix.factors.prediction = TRUE)

makeLearner(“regr.gbm"
   , id = “gmb_model”
   , par.vals = list(n.trees = 500, interaction.depth = 3))
```

Dostępne metody uczące

- Dokumentacja lub np. `listLearners(tsk)`

Class / Short Name / Name	Packages	Num.	Fac.	Ord.	NAs	Weights	Props	Note
classif.ada *ada* ada Boosting	ada rpart	X	X				prob twoclass	`xval` has been set to `0` by default for speed.
classif.bartMachine *bartmachine* Bayesian Additive Regression Trees	bartMachine	X	X		X		prob twoclass	`use_missing_data` has been set to `TRUE` by default to allow missing data support.
classif.binomial *binomial* Binomial Regression	stats	X	X			X	prob twoclass	Delegates to `glm` with freely choosable binomial link function via learner parameter `link`. We set 'model' to FALSE by default to save memory.

Ustawianie parametrów metody

- Podczas tworzenia obiektu `Learner`:

```
makeLearner(“regr.gbm"
   , par.vals = list(n.trees = 500, interaction.depth = 3))
makeLearner("regr.gbm", n.trees = 500, interaction.depth = 3)
```

- Na utworzonym wcześniej obiekcie

```
lrn = makeLearner(“regr.gbm”)
lrn = setHyperPars(lrn, n.trees = 500, interaction.depth = 3)
```

- W świecie tidyverse

```
lrn = makeLearner(“regr.gbm”) %>%
      setHyperPars(n.trees = 500, interaction.depth = 3)
```

Strojenie parametrów

- Ustalanie przestrzeni przeszukiwania

```
par.set = makeParamSet(
    makeDiscreteParam("S", values = c("linear","ranked"))
    , makeDiscreteVectorParam(id = "C", values = c('dp','radial','exp'),)
    , makeNumericParam("scal.lambda", lower = 0.0, upper = 1.0)
    , makeFunctionParam(“weights”,
        default = function(y, w.par){ 0.01+dnorm(y,sd=abs(w.par))})
    , makeIntegerParam("optim.maxit", lower=10, upper = 100000)
    , makeLogicalParam("verbose", default = FALSE, tunable = FALSE)
)
```

- Strategie optymalizacji - obiekty TuneControl

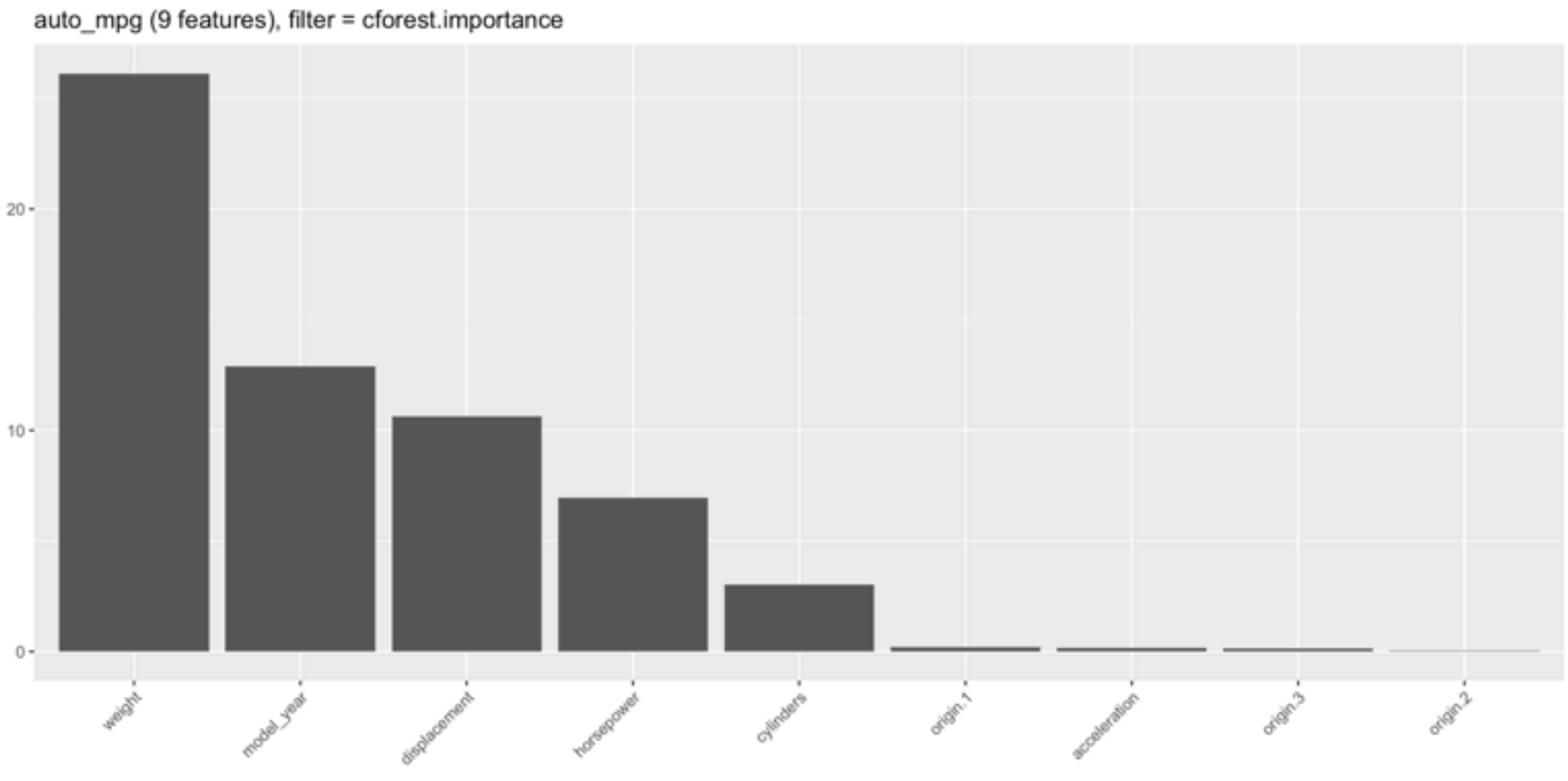
Strategie optymalizacji

- `makeTuneControlDesign` – podajemy ramkę danych z parametrami
- `makeTuneControlGrid` – przeszukiwanie po hipersiatce
- `makeTuneControlRandom` – przeszukiwanie po losowe
- `makeTuneControlCMAES` – zastosowanie algorytmu CMA-ES
- `makeTuneControlGenSA` – symulowane wyżarzanie

- `makeTuneControlIrace` – metoda “iterated F-Racing”

Selekcja atrybutów - filtry

```
featureImportance = generateFilterValuesData(autoMpgPreprocessedTask, method = c("cforest.importance"))
plotFilterValues(featureImportance)
```



Selekcja atrybutów - wrapper

- Exhaustive search (makeFeatSelControlExhaustive),
- Genetic algorithm (makeFeatSelControlGA),
- Random search (makeFeatSelControlRandom),
- Deterministic forward or backward search (makeFeatSelControlSequential)

Uczenie modelu

- “Zwyczajne”:

```
model <-train(lrn, task)
```

- Z dodatkowym próbkowaniem:

```
results =  lrn %>%  resample(task = tsk,
                   resampling = cv3,
                   measures = list(acc, setAggregation(acc, test.sd)))
```

- W ramach benchmarku:

```
benchmarkResults <- benchmark(list(lrn1, lrn2), list(task1, task2),
                   resampling = cv3,
                   measures = list(acc, setAggregation(acc, test.sd)))
```

Próbkowanie

- Dostępne strategie:
  - CV - walidacja krzyżowa (`cv2, cv3, cv5, cv10`),
  - RepCV - powtarzana walidacja krzyżowa,
  - LOO - “leave-one-out”,
  - Bootstrap - bootstrapping,
  - Subsample - losowo wybrany zbiór testowy,
  - Holdout - ustalony zbiór testowy (`hout`).

“Gotowych” miar błędu jest wiele

- featperc
- timetrain
- timepredict
- timeboth
- sse
- mse
- rmse
- medse
- sae
- mae
- medae
- rsq
- expvar
- arsq
- rrse
- rae
- mape
- msle
- rmsle
- kendalltau
- spearmanrho
- mmce
- acc
- ber
- multiclass.aunp
- multiclass.au1u
- multiclass.au1p
- multiclass.brier
- logloss
- ssr
- qsr
- lsr
- kappa
- wkappa
- auc
- brier
- brier.scaled
- bac
- tp
- tn
- fp
- fn
- tpr
- tnr
- fpr
- fnr
- ppv
- npv
- fdr
- mcc
- f1
- gmean
- gpr
- multilabel.hamloss
- multilabel.subset01
- multilabel.f1
- multilabel.acc
- multilabel.ppv
- multilabel.tpr
- cindex
- meancosts
- mcp
- db
- dunn
- G1
- G2
- silhouette

… a można tworzyć własne

Obliczenia równoległe

```
parallelMap::parallelStartMulticore(level = 'mlr.resample')
```

level	działanie
mlr.resample	instancje próbkowania równolegle
mlr.benchmark	pojedyncze eksperymenty równolegle
mlr.tuneParams	obliczenia dla pojedynczych zestawów parametrów równolegle (zależy od strategii optymalizacji)
mlr.selectFeatures	pojedyncze zestawy atrybutów równolegle
mlr.ensemble	w przypadku modeli złożonych - np. bagging.

Kompletny przykład - raz jeszcze

```
parallelMap::parallelStartMulticore(level = 'mlr.resample')
results = makeLearner("classif.knn") %>%
          makeTuneWrapper(resampling = cv3,
                          measures = acc,
                          par.set = makeParamSet(makeIntegerParam("k", lower=1, upper=20)),
                          control = makeTuneControlGrid()) %>%
          resample(task = makeClassifTask(data = iris, target = "Species"),
                   resampling = cv3,
                   measures = list(acc, setAggregation(acc, test.sd)))
parallelMap::parallelStop()
```

Rozszerzanie MLR

Źródło obrazu: https://cdn.pixabay.com/photo/2013/06/16/13/41/cotton-harvester-139651_1280.jpg

Własne rozszerzenia MLR

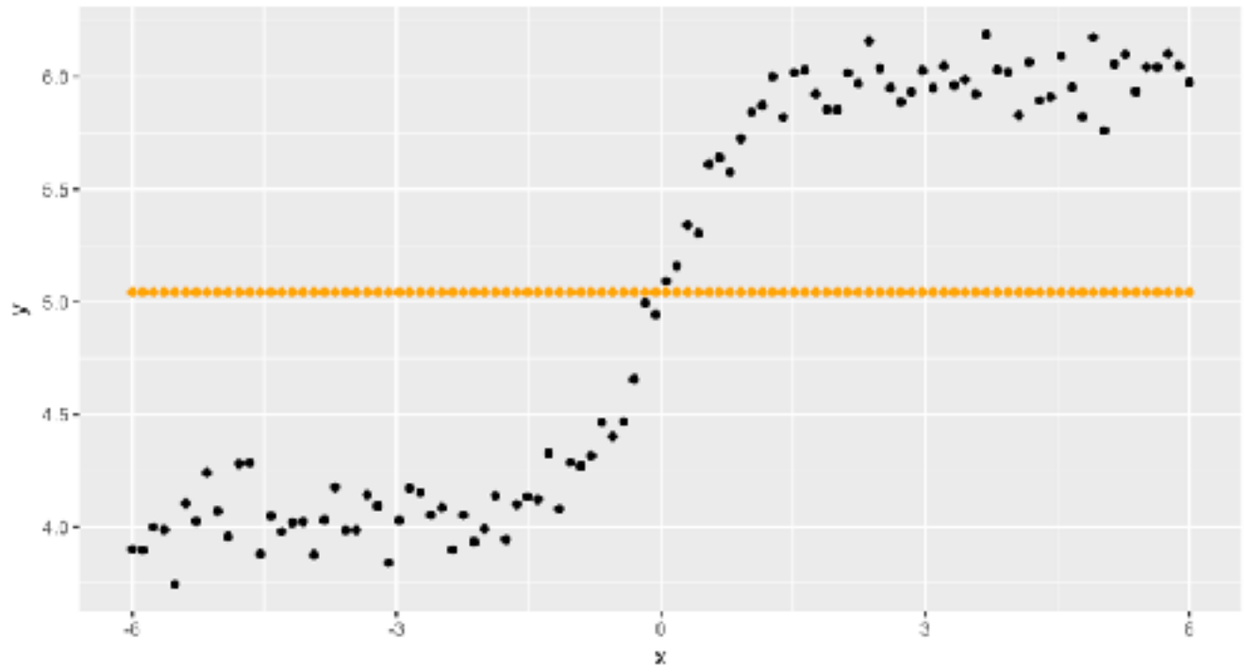
- Możemy dodawać własne:
  - miary jakości/błędu,
  - funkcje do uzupełniania danych,
  - algorytmy selekcji cech - tzw. metody filtrujące,
  - metody uczenia.

Własna metoda ucząca

```r
## Obiekt z opisem metody
makeRLearner.regr.foo <- function() {
  makeRLearnerRegr(
    cl = "regr.foo",
    package = "stats",
    par.set = makeParamSet(),
    properties = list('numerics'),
    name = "Simple median model",
    short.name = "foo"
  )
}

## Uczenie
trainLearner.regr.foo = function(.learner, .task, .subset, .weights = NULL, ...) {
  trainData <- getTaskData(.task,.subset, target.extra = TRUE)
  list(targetMedian = median(trainData$target))
}

## Predykcja przy pomocy gotowego modelu
predictLearner.regr.foo = function(.learner, .model, .newdata, ...) {
  model <- .model$learner.model

  rep(model$targetMedian, nrow(.newdata))
}
```

Własna metoda - użycie

```r
x <- seq(-6,6,length.out=100); y <- tanh(x) + rnorm(100, sd=0.1) + 5
trainingData <- data.frame(x,y)

tsk <- makeRegrTask(data=trainingData, target = "y")
m <- makeLearner('regr.foo') %>%  train(tsk)
```

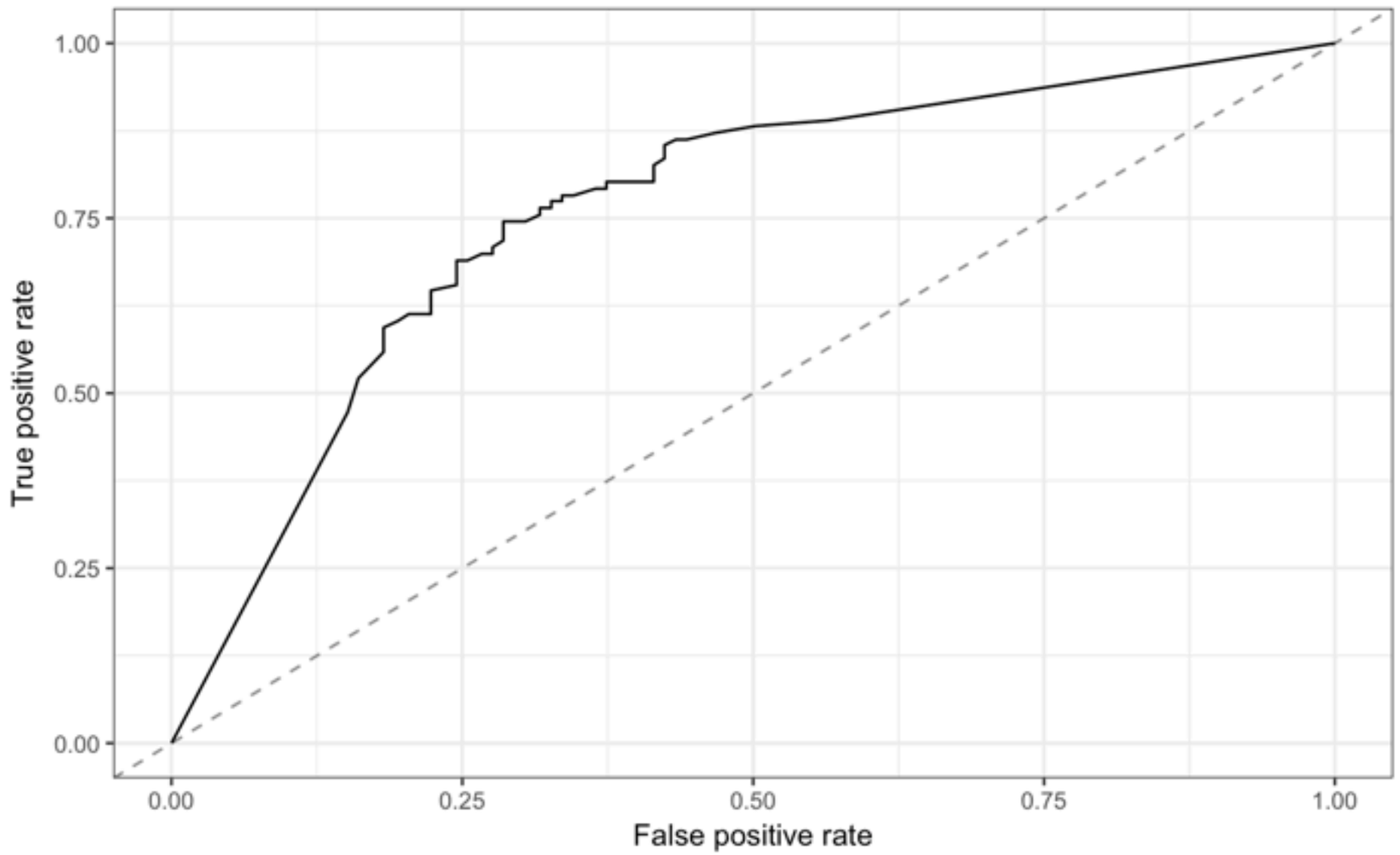


Diagnostyka i wizualizacja wyników

Źródło obrazu: https://cdn.pixabay.com/photo/2013/06/16/13/41/cotton-harvester-139651_1280.jpg

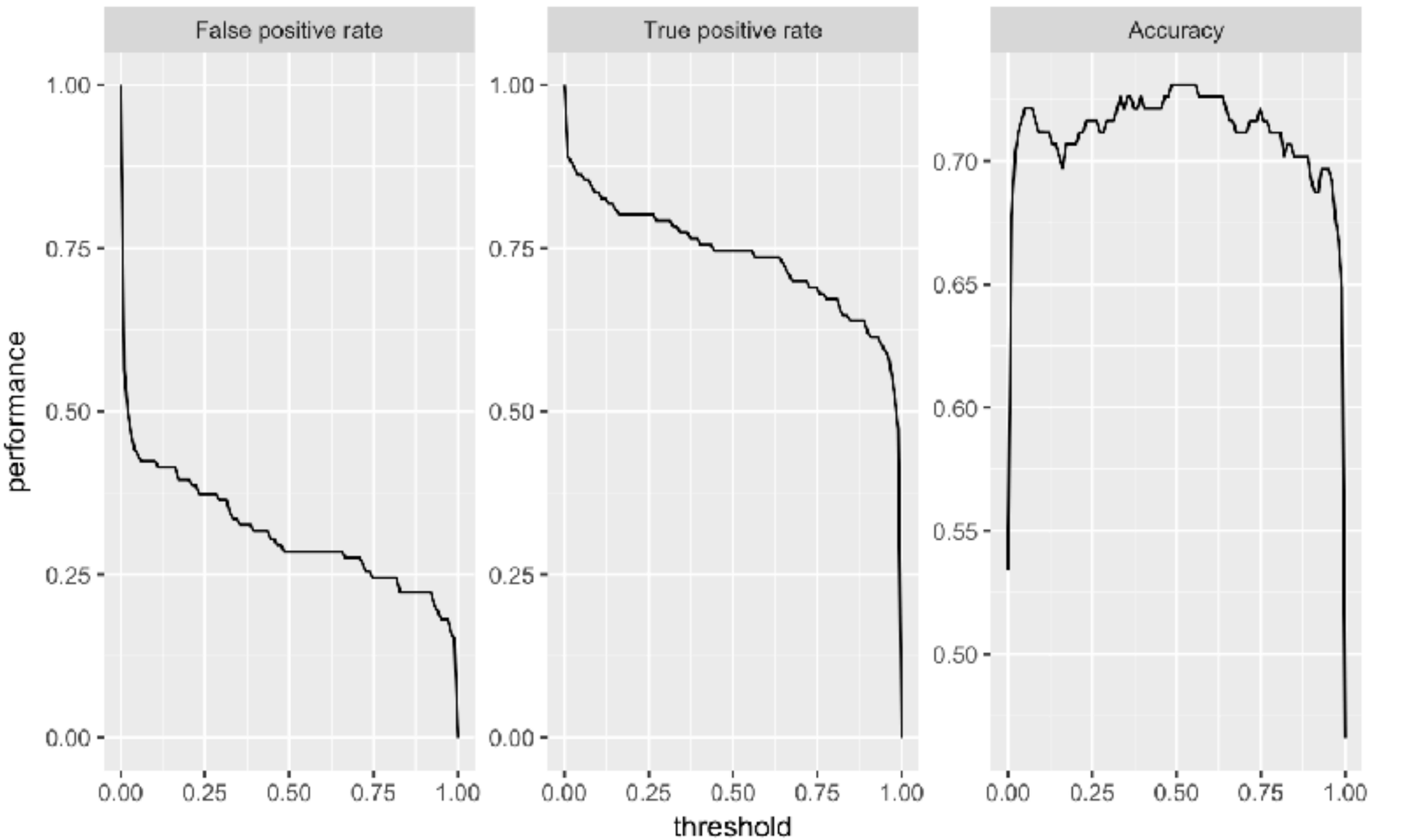
Krzywa ROC

```
df = generateThreshVsPerfData(results, measures = list(fpr, tpr, acc))

plotROCCurves(df) + theme_bw()
```



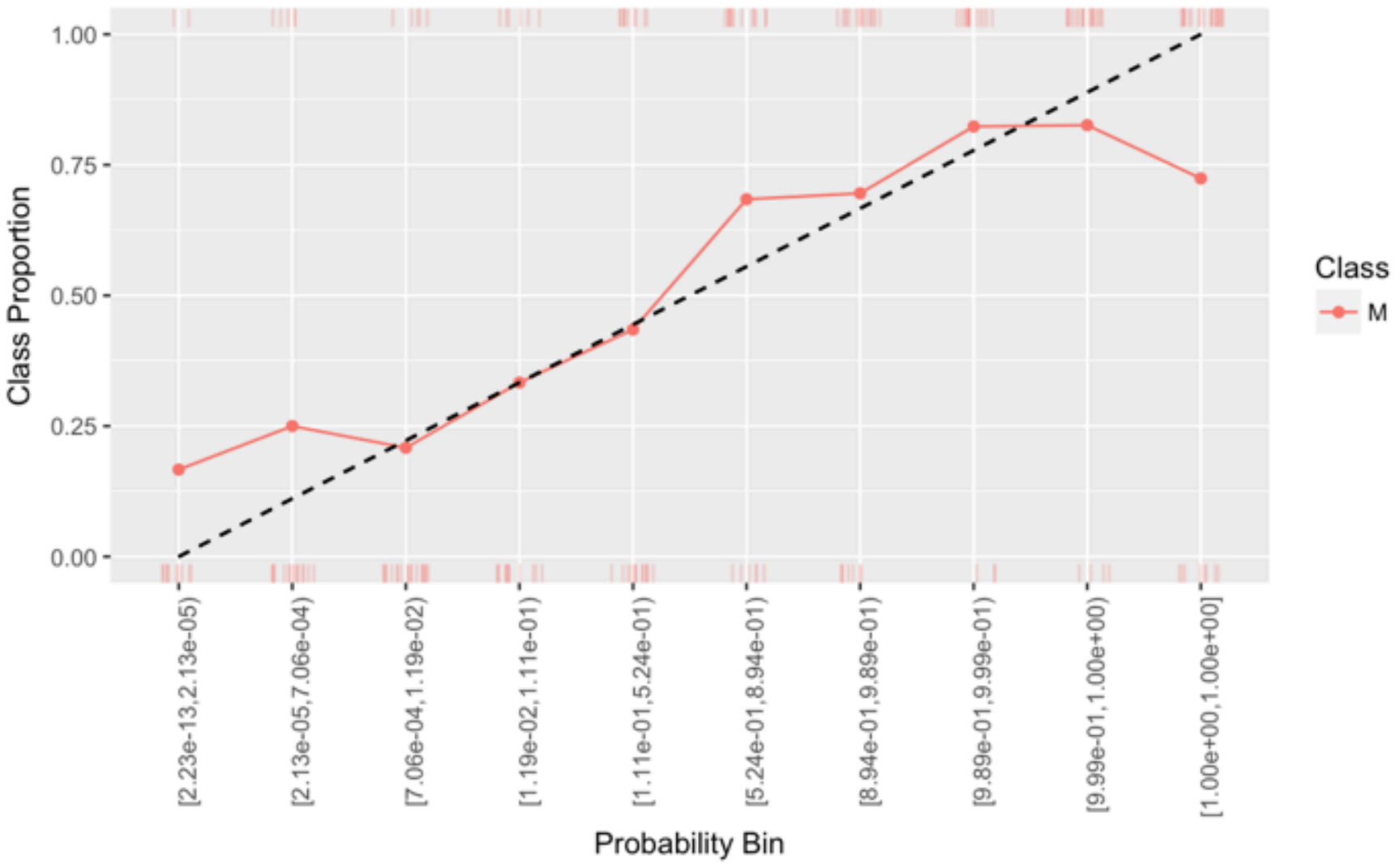
Miara jakości a wartość progu

```
df = generateThreshVsPerfData(results, measures = list(fpr, tpr, acc))
plotThreshVsPerf(df)
```



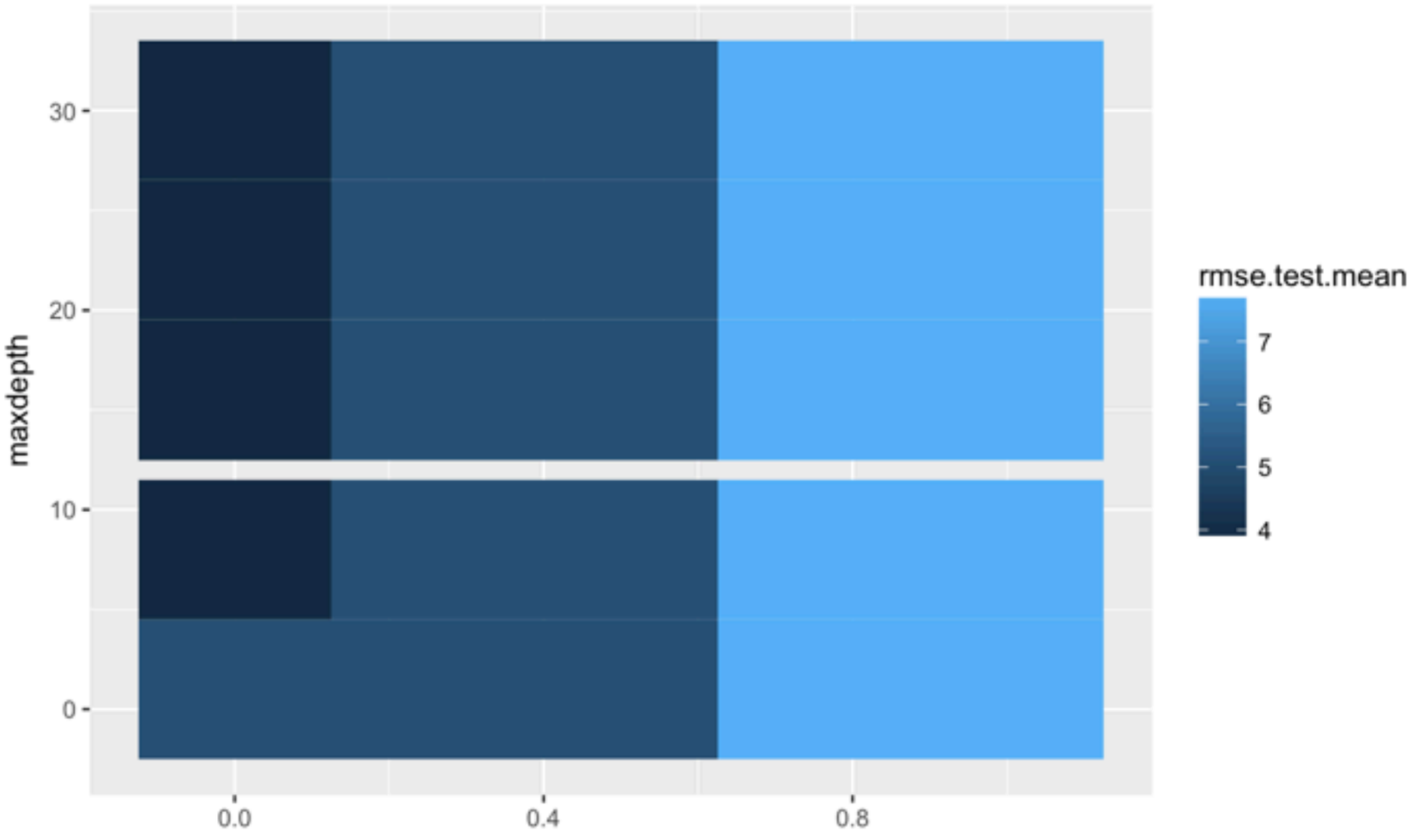
Kalibracja klasyfikatora

```
cal = generateCalibrationData(results, groups = 10)
plotCalibration(cal)
```



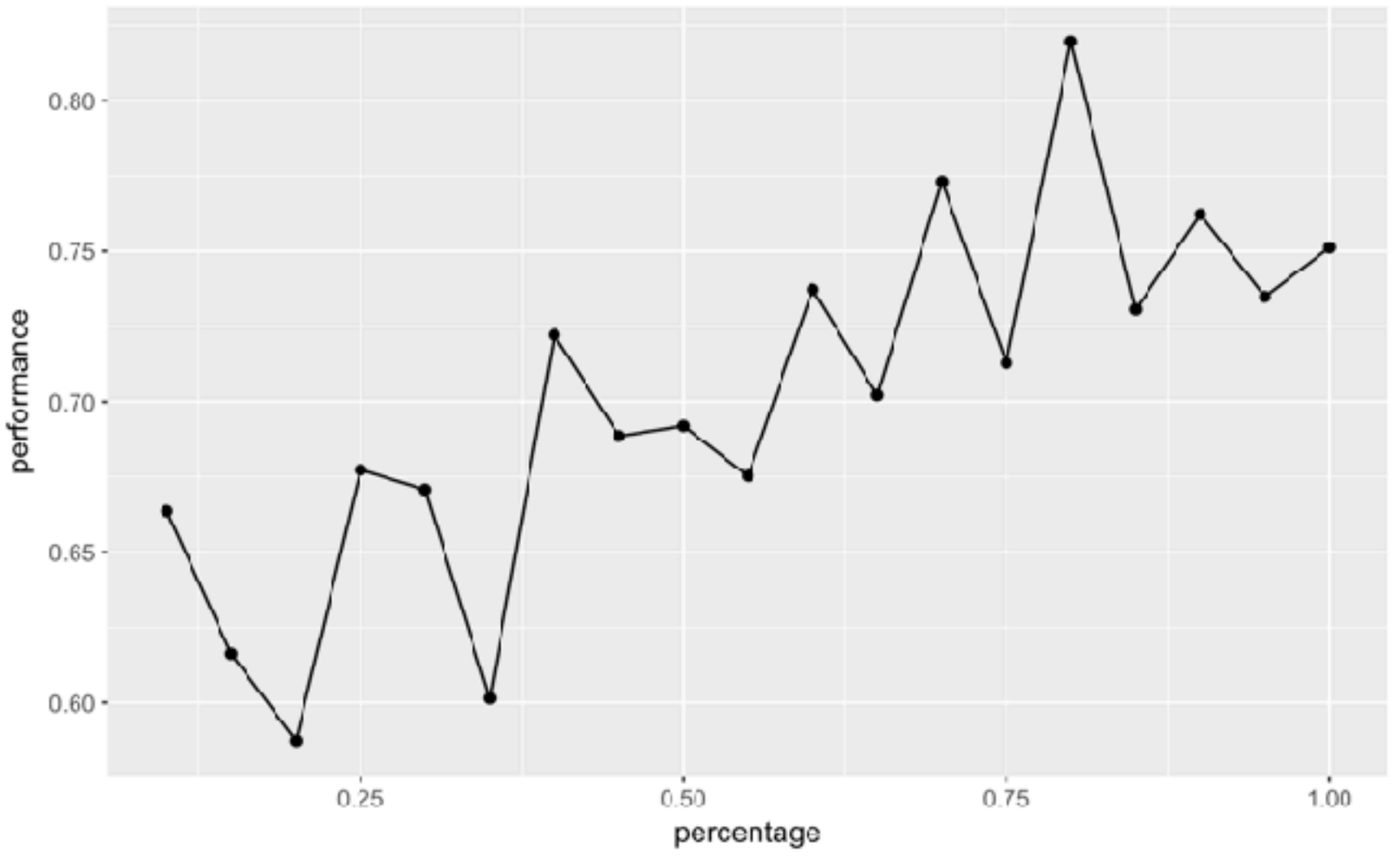
Wpływ wartości hiperparametrów na model

```
data  <- tuneParams(rpartLearner,
                    task = autoMpgTask,
                    control = makeTuneControlGrid(resolution = 5),
                    measures = setAggregation(rmse, test.mean),
                    resampling = makeResampleDesc("Holdout"),
                    par.set = ps, show.info = FALSE) %>%
                    generateHyperParsEffectData()

plotHyperParsEffect(data, x = "cp", y = "maxdepth", z = "rmse.test.mean", plot.type = "heatmap")
```



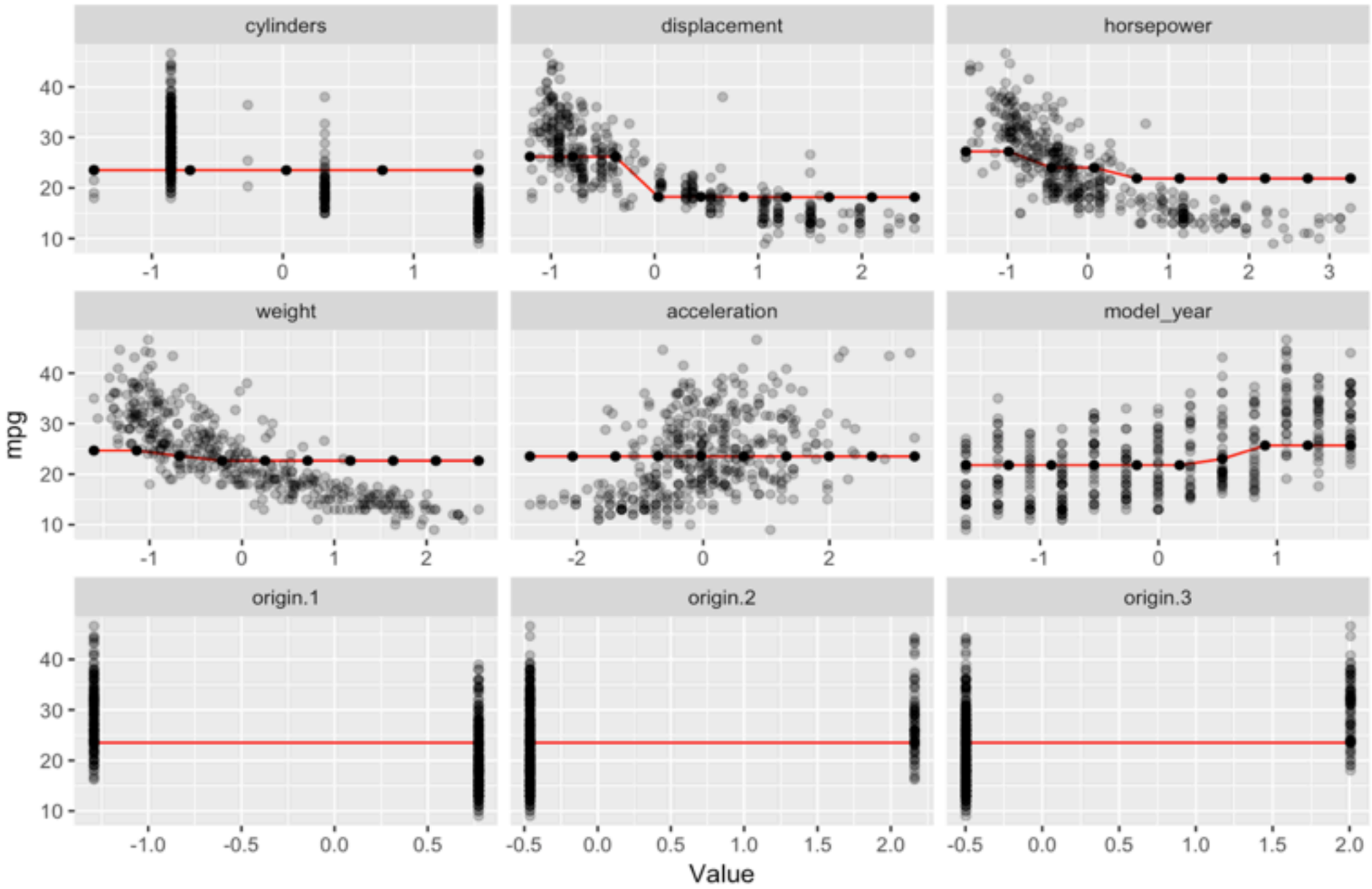
Krzywa uczenia

```
r = generateLearningCurveData(
    learners = lrn,
    task = sonar.task,
    percs = seq(0.1, 1, by = 0.05),
    measures = list(tpr),
    resampling = cv10)
plotLearningCurve(r)
```



Wpływ atrybutów wejściowych na model

```
pd = generatePartialDependenceData(train('regr.rpart', autoMpgTask), autoMpgTask)
plotPartialDependence(pd, data = getTaskData(autoMpgTask))
```



Podsumowanie…

MLR - podsumowanie

- Zadania: klasyfikacja, regresja, klastrowanie i analizy przeżycia.
- Możliwość kodowania typów danych i ich ograniczeń.
- Bootstrapping, walidacja krzyżowa, próbkowanie - również zagnieżdżone.
- Wizualizacje: krzywe ROC, predykcje.
- Możliwość tworzenia “benchmarków”.
- Strojenie parametrów przy pomocy różnych strategii optymalizacyjnych - np. F-racing.
- Selekcja zmiennych.
- Wsparcie dla ważenia przypadków/nierównomiernego rozkładu klas.
- Wbudowane wsparcie dla zrównoleglania.

Dziękuję za uwagę!

Źródło obrazu: https://cdn.pixabay.com/photo/2013/06/16/13/41/cotton-harvester-139651_1280.jpg