LABORATORIUM 10

- Ocena skuteczności klasyfikatora
- Krzywa ROC
- Walidacja krzyżowa

Eksploracja danych

Eksploracja danych (*data mining*) to proces odkrywania znaczacych nowych powiązań, wzorców i trendów poprzez przeszukiwanie dużych ilości danych zgromadzonych w bazach danych przy użyciu metod matematycznych. Na zajęciach zajmiemy się problemem klasyfikacji, czyli jak na podstawie cech pewnych obiektów przypisać je do określonych klas (inaczej: jak dokonać podziału na poszczególne klasy), jak również kwestią redukcji wymiaru. W ogólności możemy wyróżnić dwa typu klasyfikacji: **pod nadzorem** (*supervised learning*) i **bez nadzoru** (*unsupervised learning*). W pierwszym przypadku musimy dysponować pewnym zbiorem danych, w którym istnieją już dane sklasyfikowane (czyli posiadające przypisaną klasę, tzw. “próba ucząca”). Algorytm “uczy się” cech związanych z obiektami, a następnie korzystając z takiej wiedzy, klasyfikuje nowe dane. W przypadku klasyfikacji bez nadzoru, rozdział na klasy następuje bez uprzedniego procesu uczenia.

Ocena skuteczności klasyfikatora

Aby oszacować skuteczność danego klasyfikatora (modelu predykcyjnego) niezbędna jest do tego próba testowa, która powinna być rozdzielna z próbą uczącą. Przepuszczamy obserwacje z próby testowej przez model i odczytujemy przewidziane klasy dla tych obserwacji, a następnie porównujemy je z rzeczywistymi klasami. W ten sposób możemy skontruować macierz pomyłek (*confusion matrix*), której pola mają nazwy jak na rysunku poniżej (numenklatura jest związana z testami medycznymi, mającymi na celu określenie nosicielstwa danej choroby).

	sklasyfikowani jako CHORZY	**sklasyfikowani jako ZDROWI**
faktycznie CHORZY	TRUE POSITIVE (**TP**)	FALSE NEGATIVE (**FN**)
faktycznie ZDROWI	FALSE POSITIVE (**FP**)	TRUE NEGATIVE (**TN**)

Wszystkie $n = P + N$ przypadki dzielą się na osobników chorych $P = TP + FN$ i zdrowych $N = TN + FP$. Na podstawie macierzy pomyłek można zdefiniować następujące wielkości pomocne w porównywaniu klasyfikatorów:

- czułość (ang. *sensitivity, recall, true positive rate*) – odsetek poprawnie wykrytych osobników chorych

  $TPR = \frac{TP}{P}$,

- specyficzność (ang. *specificity, selectivity, true negative rate*) - odsetek poprawnie odrzuconych osobników zdrowych

  $TNR = \frac{TN}{N}$,

- precyzja (ang. *precision, positive predictive value*) - proporcja prawdziwie pozytywnych wyników wśród wszystkich wyników pozytywnych

  $PPV = \frac{TP}{TP+FP}$,

- wartość predykcyjna ujemna (ang. *negative predictive value*) - proporcja prawdziwie negatywnych wyników wśród wszystkich wyników negatywnych

$$NPV = \frac{TN}{TN+FN}$$

- dokładność (ang. *accurracy*) - odsetek poprawnie sklasyfikowanych obserwacji

$$ACC = \frac{TN+TP}{n},$$

- miara F_1 (ang. *F1 score*) - średnia harmoniczna precyzji i czułości

$$F_1 = \frac{2 \cdot PPV \cdot TPR}{PPV+TPR} = \frac{2TP}{2TP+FP+FN}.$$

Dokładność określa na ile dobrze algorytm przewiduje dowolną klasę. Czułość i specyficzność są prawdopodobieństwami warunkowymi dobrej klasyfikacji, pod warunkiem, że obiekt faktycznie należy do danej klasy. Mają one szczególne znaczenie, jeśli rozkład liczebności klas jest nierówny. Dla przykładu obliczmy powyższe miary dla modelu regresji logistycznej dla zbioru danych `Pima` z biblioteki `MASS`.

```
library(MASS)
# Regresja liniowa oparta o statystycznie istotne czynniki: glu i ped
pima.lr <- glm (type~glu+ped, data=Pima.tr, family = binomial(link = "logit"))

# Predykcja
pima.predict <- predict (pima.lr, newdata = Pima.te, type = "response")
pima.result <- ifelse (pima.predict>0.50, "Yes", "No")
pima.true <- Pima.te[,8]

# Macierz pomyłek
pima.cm <- table(pima.true,pima.result)
pima.cm
```

```
##          pima.result
## pima.true  No Yes
##       No  205  18
##       Yes  50  59
```

```
TN <- pima.cm[1,1]
TP <- pima.cm[2,2]
FP <- pima.cm[1,2]
FN <- pima.cm[2,1]
P <- TP+FN
N <- TN+FP
# Czułość
TP/P
```

```
## [1] 0.5412844
```

```
# Specyficzność
TN/N
```

```
## [1] 0.9192825
```

```
# Precyzja
TP/(TP+FP)
```

```
## [1] 0.7662338
```

```
# Wartość predykcyjna ujemna
TN/(TN+FN)
```

```
## [1] 0.8039216
```

```
# Dokładność
(TN+TP)/sum(pima.cm)
```

```
## [1] 0.7951807
```

```
# Miara F1
2*TP/(2*TP+FP+FN)
```

```
## [1] 0.6344086
```

Szybszym sposobem na policzenie powyższych wartości jest skorzystanie z funkcji `confusionMatrix()` z biblioteki `caret`. Argumentami tej funkcji może być zarówno macierz współwystępowania zwracana przez funkcję `table` jak i wektory przynależności do klas (przewidywane i rzeczywiste).

```
library(caret)
pima.conf <- confusionMatrix (as.factor(pima.result), pima.true, positive = "Yes")
pima.conf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  205  50
##        Yes  18  59
##
##                Accuracy : 0.7952
##                  95% CI : (0.7477, 0.8373)
##     No Information Rate : 0.6717
##     P-Value [Acc > NIR] : 4.282e-07
##
##                   Kappa : 0.4979
##
##  Mcnemar's Test P-Value : 0.0001704
##
##             Sensitivity : 0.5413
##             Specificity : 0.9193
##          Pos Pred Value : 0.7662
##          Neg Pred Value : 0.8039
##              Prevalence : 0.3283
##          Detection Rate : 0.1777
##    Detection Prevalence : 0.2319
##       Balanced Accuracy : 0.7303
##
##        'Positive' Class : Yes
##
```

Powyższa funkcja zwraca również przedziały ufności dla dokładności i testuje hipotezę statystyczną czy dokładność jest większa niż “no information rate”, który jest równy procentowemu udziałowi liczniejszej klasy. Miara F1 nie jest domyślnie pokazana i trzeba się do niej dostać przez pole `byClass`.

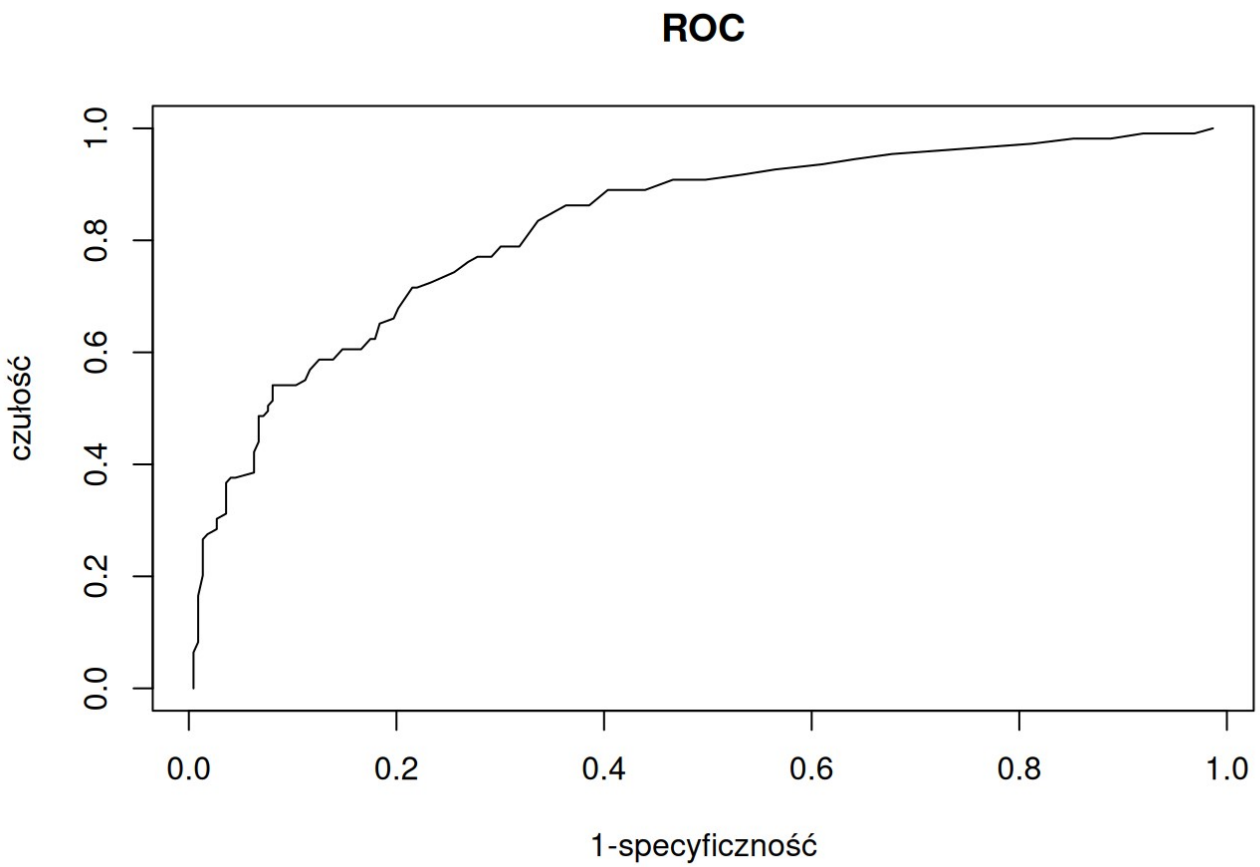
```
pima.conf$byClass
```

```
##          Sensitivity          Specificity       Pos Pred Value
##            0.5412844            0.9192825            0.7662338
##       Neg Pred Value            Precision               Recall
##            0.8039216            0.7662338            0.5412844
##                   F1           Prevalence       Detection Rate
##            0.6344086            0.3283133            0.1777108
## Detection Prevalence    Balanced Accuracy
##            0.2319277            0.7302835
```

Krzywa ROC (Receiver Operating Characteristic)

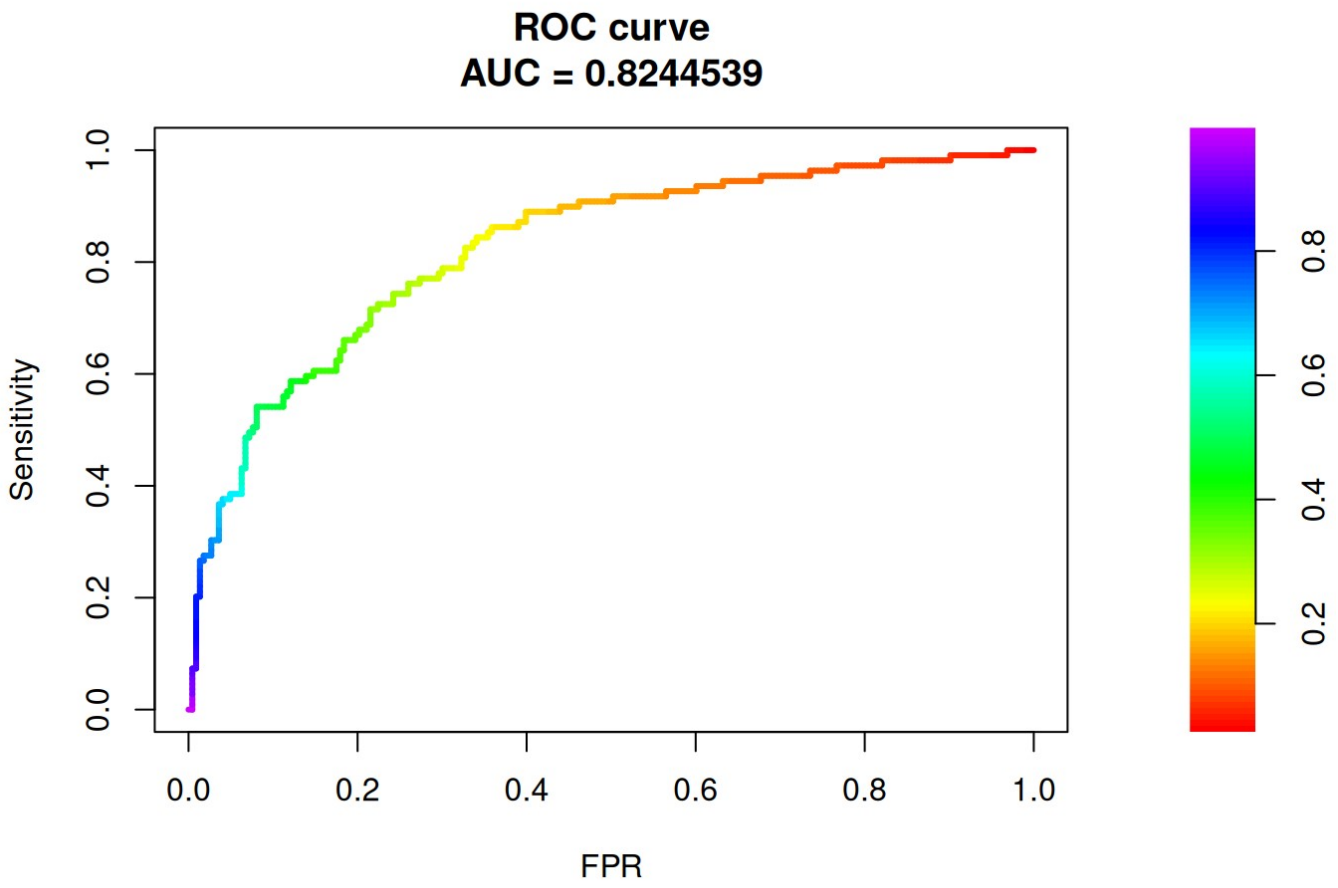
W powyższym przykładzie założyliśmy, że progiem prawdopodobieństwa, od którego uznajemy osobnika za diabetyka jest 0.5. Właściwszym podejściem byłoby przetestowanie wielu progów i zbadanie czułości i specyficzności dla każdego z progów. Przy pomocy funkcji `sapply` można policzyć macierz pomyłek dla dowolnej liczby progów, przy czym próg 0.04 jest minimalny, przy którym występują obie klasy.

```
progi <- seq(0.04,0.99,0.01)
roc <- sapply (progi, function(p)
  confusionMatrix (as.factor(ifelse(pima.predict>p, "Yes", "No")), pima.true, positive = "
        Yes")$byClass[c("Sensitivity","Specificity")])
plot(1-roc["Specificity",], roc["Sensitivity",], xlab="1-specyficzność", ylab="czułość", m
        ain="ROC", type = "l")
```



Powyższa krzywa nosi nazwę ROC (Receiver Operating Characteristic) i ilustruje ona jakość klasyfikatora. Optymalny punkt podziału to ten, który maksymalizuje **Indeks Youdena**, dany wzorem $YI = TPR + TNR - 1$. Istnieje wiele bibliotek, które służą pomocą przy wykreślaniu krzywej ROC, a nawet obliczają optymalny punkt podziału i AUC (Area Under Curve), czyli pole pod wykresem krzywej ROC.

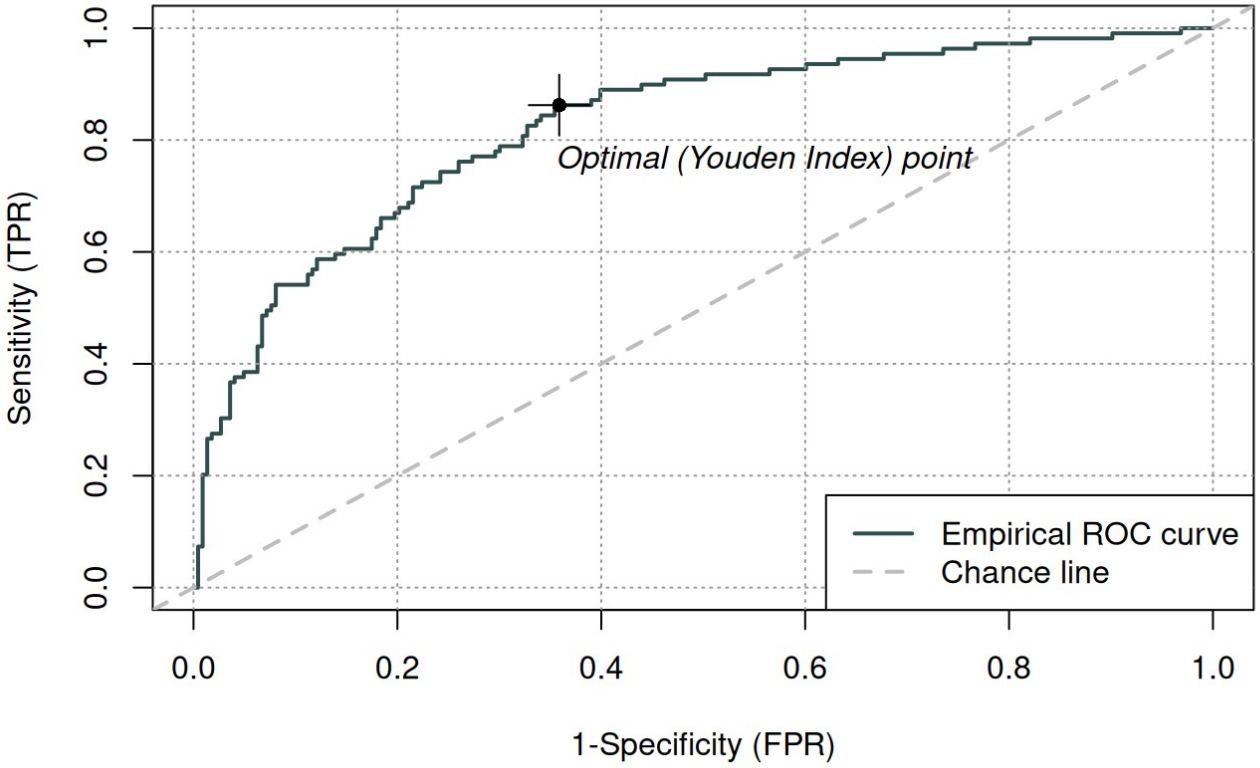
```
library(PRROC)
pima.true <- ifelse(Pima.te[,8]=="Yes", 1, 0)
prroc.obj <- roc.curve(scores.class0 = pima.predict, weights.class0 = pima.true, curve=TRU
        E)
plot(prroc.obj)
```




```
library(ROCit)
rocit.obj <- rocit(score=pima.predict,class=Pima.te[,8])
summary(rocit.obj)
```

```
##
##  Method used: empirical
##  Number of positive(s): 109
##  Number of negative(s): 223
##  Area under curve: 0.8245
```

```
plot(rocit.obj)
```



Niestety maksymalny Indeks Youdena nie jest przechowywany w obiekcie `rocit` , ale możemy łatwo go obliczyć.

```
best.yi.index <- which.max(rocit.obj$TPR-rocit.obj$FPR)
best.cutoff <- rocit.obj$Cutoff[best.yi.index]
best.tpr <- rocit.obj$TPR[best.yi.index]
best.fpr <- rocit.obj$FPR[best.yi.index]
sprintf("Best Cutoff = %.2f (TPR = %.3f, FPR = %.3f)", best.cutoff, best.tpr, best.fpr)
```

```
## [1] "Best Cutoff = 0.23 (TPR = 0.862, FPR = 0.359)"
```

Walidacja krzyżowa

Klasyczna walidacja modelu polega na podzieleniu zbioru danych na dwa rozłączne podzbiory: zbiór uczący i zbiór treningowy. Podział ten można w prosty sposób wykonać przy pomocy funkcji `createDataPartition()` z biblioteki `caret` .

```
data("swiss")
# Dane dotyczą płodności w 47 prowincjach Szwajcarii w roku 1888
head(swiss)
```

```
##              Fertility Agriculture Examination Education Catholic
## Courtelary        80.2        17.0          15        12     9.96
## Delemont          83.1        45.1           6         9    84.84
## Franches-Mnt      92.5        39.7           5         5    93.40
## Moutier           85.8        36.5          12         7    33.77
## Neuveville        76.9        43.5          17        15     5.16
## Porrentruy        76.1        35.3           9         7    90.57
##              Infant.Mortality
## Courtelary               22.2
## Delemont                 22.2
## Franches-Mnt             20.2
## Moutier                  20.3
## Neuveville               20.6
## Porrentruy               26.6
```

```
set.seed(123) # Ustawiamy ziarno generatora liczb losowych
swiss.samples <- createDataPartition(swiss$Fertility, p = 0.8, list = FALSE)
swiss.train <- swiss[swiss.samples,]
swiss.test <- swiss[-swiss.samples,]
swiss.lm <- lm(Fertility ~., data = swiss.train)
swiss.predict <- predict(swiss.lm, newdata=swiss.test)
data.frame (R2 = R2(swiss.predict, swiss.test$Fertility),
            RMSE = RMSE(swiss.predict, swiss.test$Fertility),
            MAE = MAE(swiss.predict, swiss.test$Fertility))
```

```
##          R2     RMSE      MAE
## 1 0.5946201 6.410914 5.651552
```

W powyższym przykładzie wykorzystaliśmy trzy miary oszacowania błędu: współczynnik R^2, pierwiastek błędu średniokwadratowego (RMSE, Root Mean Squared Error) oraz średni błąd bezwzględny (MAE, Mean Absolute Error). Jeśli dysponujemy bardzo dużym zbiorem danych, klasyczna walidacja powinna być wystarczająca do oszacowania skuteczności modelu. Jeśli jednak zebranych danych nie jest wiele, każda obserwacja jest cenna i może wpłynąć na charakterystykę modelu i dlatego wyłączanie części obserwacji ze zbioru treningowego nie jest wskazane. W takich sytuacjach stosuje się **walidację krzyżową** (ang. *cross validation*) w jednej z czterech odmian:

1. **Bootstrap** - polega na generowaniu wielu podzbiorów testowych poprzez losowanie ze zwracaniem z oryginalnego zbioru. Zwykle generuje się setki lub tysiące takich podzbiorów i dla każdego liczy się skuteczność modelu. Wynik końcowy jest uśrednieniem (lub inną funkcją) wszystkich obliczonych skuteczności.
2. **Leave one out cross validation (LOOCV)** - polega na wybraniu tylko jednej obserwacji do zbioru testowego. Pozostałe obserwacje są wykorzystywane do wytrenowania modelu. Operacja ta jest powtórzona dla wszystkich obserwacji ze zbioru, a końcowa skuteczność jest średnią wszystkich obliczonych skuteczności.
3. **K-krotna walidacja krzyżowa (CV, K-fold cross validation)** - dzielimy zbiór danych na K losowych podzbiorów, następnie uczymy i testujemy model K razy, za każdym razem wybierając inny podzbiór jako próbę testową, a pozostałe jako próbę uczącą.
4. **K-krotna walidacja krzyżowa z potwórzeniami (RCV, Repeated K-fold cross validation)** - polega na wielokrotnym wykonaniu K-krotnej walidacji krzyżowej, przy czym za każdym razem losuje się inne K podzbiory.

W bibliotece `caret` jest zaimplementowana funkcja `train()` będąca wrapperem do ponad dwustu różnych modeli statystycznych i pozwalająca na wykonanie walidacji krzyżowej w łatwy sposób.

```
# Wypisanie modeli kompatybilnych z funkcją train()
names(getModelInfo())
```

```
##   [1] "ada"                 "AdaBag"              "AdaBoost.M1"
##   [4] "adaboost"            "amdai"               "ANFIS"
##   [7] "avNNet"              "awnb"                "awtan"
##  [10] "bag"                 "bagEarth"            "bagEarthGCV"
##  [13] "bagFDA"              "bagFDAGCV"           "bam"
##  [16] "bartMachine"         "bayesglm"            "binda"
##  [19] "blackboost"          "blasso"              "blassoAveraged"
##  [22] "bridge"              "brnn"                "BstLm"
##  [25] "bstSm"               "bstTree"             "C5.0"
##  [28] "C5.0Cost"            "C5.0Rules"           "C5.0Tree"
##  [31] "cforest"             "chaid"               "CSimca"
##  [34] "ctree"               "ctree2"              "cubist"
##  [37] "dda"                 "deepboost"           "DENFIS"
##  [40] "dnn"                 "dwdLinear"           "dwdPoly"
##  [43] "dwdRadial"           "earth"               "elm"
##  [46] "enet"                "evtree"              "extraTrees"
##  [49] "fda"                 "FH.GBML"             "FIR.DM"
##  [52] "foba"                "FRBCS.CHI"           "FRBCS.W"
##  [55] "FS.HGD"              "gam"                 "gamboost"
##  [58] "gamLoess"            "gamSpline"           "gaussprLinear"
##  [61] "gaussprPoly"         "gaussprRadial"       "gbm_h2o"
##  [64] "gbm"                 "gcvEarth"            "GFS.FR.MOGUL"
##  [67] "GFS.LT.RS"           "GFS.THRIFT"          "glm.nb"
##  [70] "glm"                 "glmboost"            "glmnet_h2o"
##  [73] "glmnet"              "glmStepAIC"          "gpls"
##  [76] "hda"                 "hdda"                "hdrda"
##  [79] "HYFIS"               "icr"                 "J48"
##  [82] "JRip"                "kernelpls"           "kknn"
##  [85] "knn"                 "krlsPoly"            "krlsRadial"
##  [88] "lars"                "lars2"               "lasso"
##  [91] "lda"                 "lda2"                "leapBackward"
##  [94] "leapForward"         "leapSeq"             "Linda"
##  [97] "lm"                  "lmStepAIC"           "LMT"
## [100] "loclda"              "logicBag"            "LogitBoost"
## [103] "logreg"              "lssvmLinear"         "lssvmPoly"
## [106] "lssvmRadial"         "lvq"                 "M5"
## [109] "M5Rules"             "manb"                "mda"
## [112] "Mlda"                "mlp"                 "mlpKerasDecay"
## [115] "mlpKerasDecayCost"   "mlpKerasDropout"     "mlpKerasDropoutCost"
## [118] "mlpML"               "mlpSGD"              "mlpWeightDecay"
## [121] "mlpWeightDecayML"    "monmlp"              "msaenet"
## [124] "multinom"            "mxnet"               "mxnetAdam"
## [127] "naive_bayes"         "nb"                  "nbDiscrete"
## [130] "nbSearch"            "neuralnet"           "nnet"
## [133] "nnls"                "nodeHarvest"         "null"
## [136] "OneR"                "ordinalNet"          "ordinalRF"
## [139] "ORFlog"              "ORFpls"              "ORFridge"
## [142] "ORFsvm"              "ownn"                "pam"
## [145] "parRF"               "PART"                "partDSA"
## [148] "pcaNNet"             "pcr"                 "pda"
## [151] "pda2"                "penalized"           "PenalizedLDA"
## [154] "plr"                 "pls"                 "plsRglm"
## [157] "polr"                "ppr"                 "pre"
## [160] "PRIM"                "protoclass"          "qda"
```

```
## [163] "QdaCov"              "qrf"                "qrnn"
## [166] "randomGLM"           "ranger"             "rbf"
## [169] "rbfDDA"              "Rborist"            "rda"
## [172] "regLogistic"         "relaxo"             "rf"
## [175] "rFerns"              "RFlda"              "rfRules"
## [178] "ridge"               "rlda"               "rlm"
## [181] "rmda"                "rocc"               "rotationForest"
## [184] "rotationForestCp"    "rpart"              "rpart1SE"
## [187] "rpart2"              "rpartCost"          "rpartScore"
## [190] "rqlasso"             "rqnc"               "RRF"
## [193] "RRFglobal"           "rrlda"              "RSimca"
## [196] "rvmLinear"           "rvmPoly"            "rvmRadial"
## [199] "SBC"                 "sda"                "sdwd"
## [202] "simpls"              "SLAVE"              "slda"
## [205] "smda"                "snn"                "sparseLDA"
## [208] "spikeslab"           "spls"               "stepLDA"
## [211] "stepQDA"             "superpc"            "svmBoundrangeString"
## [214] "svmExpoString"       "svmLinear"          "svmLinear2"
## [217] "svmLinear3"          "svmLinearWeights"   "svmLinearWeights2"
## [220] "svmPoly"             "svmRadial"          "svmRadialCost"
## [223] "svmRadialSigma"      "svmRadialWeights"   "svmSpectrumString"
## [226] "tan"                 "tanSearch"          "treebag"
## [229] "vbmpRadial"          "vglmAdjCat"         "vglmContRatio"
## [232] "vglmCumulative"      "widekernelpls"      "WM"
## [235] "wsrf"                "xgbDART"            "xgbLinear"
## [238] "xgbTree"             "xyf"
```

```
# Bootstrap
train.control <- trainControl(method="boot", number=100)
swiss.boot <- train(Fertility ~ ., data = swiss, method = "lm", trControl = train.control)
print(swiss.boot)
```

```
## Linear Regression
##
## 47 samples
##  5 predictor
##
## No pre-processing
## Resampling: Bootstrapped (100 reps)
## Summary of sample sizes: 47, 47, 47, 47, 47, 47, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   8.437493  0.5887206  6.772303
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
# Leave one out cross validation (LOOCV)
train.control <- trainControl(method = "LOOCV")
swiss.loocv <- train(Fertility ~ ., data = swiss, method = "lm", trControl = train.contro
        l)
print(swiss.loocv)
```

```
## Linear Regression
##
## 47 samples
##  5 predictor
##
## No pre-processing
## Resampling: Leave-One-Out Cross-Validation
## Summary of sample sizes: 46, 46, 46, 46, 46, 46, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   7.738618  0.6128307  6.116021
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
# 10-krotna walidacja krzyżowa
train.control <- trainControl(method = "cv", number=10)
swiss.cv <- train(Fertility ~ ., data = swiss, method = "lm", trControl = train.control)
print(swiss.cv)
```

```
## Linear Regression
##
## 47 samples
##  5 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 43, 42, 43, 43, 44, 41, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   7.278777  0.7008316  6.080132
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
# 10-krotna walidacja krzyżowa powtórzona 3 razy
train.control <- trainControl(method = "repeatedcv", number=10, repeats=3)
swiss.repeatedcv <- train(Fertility ~ ., data = swiss, method = "lm", trControl = train.co
         ntrol)
print(swiss.repeatedcv)
```

```
## Linear Regression
##
## 47 samples
##  5 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 42, 41, 43, 42, 43, 41, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   7.320856  0.7033335  6.156657
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```