

Metody Odkrywania Wiedzy

Dokumentacja końcowa projektu

„Predykcja zużycia energii na podstawie danych czujnikowych”

Krzysztof Belewicz
Paweł Pińczuk

27 stycznia 2020

1. Opis projektu

Celem projektu było wyznaczenie całkowitego zużycia energii dla zadanej chwili czasu, tzn. sumy poborów sprzętów AGD (kolumna *Appliances*) i oświetlenia (kolumna *lights*). Zbiór danych został pozyskany z archiwum dostępnego na stronie: <https://archive.ics.uci.edu/ml/datasets/Appliances+energy+prediction>. Pojęciem docelowym jest wartość całkowitej pobieranej mocy przez gospodarstwo domowe. W ramach projektu zdecydowano się na oddzielne wykonania zadania regresji dla celu *Appliances* i celu *lights*, ze względu na hipotezę, że modele je wyznaczające mogą mieć inne właściwości.

Dokonano selekcji atrybutów za pomocą trzech algorytmów opisanych w rozdziale 3. Przeprowadzono procedurę oceny algorytmów liniowej regresji, drzew regresji oraz kawałkami liniowej regresji.

2. Opis danych

2.1. Charakterystyka danych

Dane wykorzystywane do eksperymentów zostały zebrane za pomocą sieci czujników w niewielkim domu w czasie 4.5 miesiąca. Składają się z:

- daty i godziny pomiaru,
- poboru energii sprzętów domowych [*Wh*],
- poboru energii oświetlenia [*Wh*],
- pomiarów temperatury i wilgotności dla 8 różnych pomieszczeń ($^{\circ}\text{C}$, %),
- pomiarów temperatury i wilgotności dla zewnętrznej, północnej strony budynku ($^{\circ}\text{C}$, %),
- danych z pobliskiej stacji pogodowej:
 - temperatura powietrza ($^{\circ}\text{C}$),
 - temperatura punktu rosy ($^{\circ}\text{C}$),
 - ciśnienie atmosferyczne [*mm Hg*],
 - wilgotność [%],
 - prędkość wiatru [*m/s*],
 - widoczność [*km*].

2.2. Przygotowanie danych

Każdy pomiar został uśredniony z 3 próbek wykonanych w równych odstępach co ok. 3,3 min. W ramach przygotowania danych, data i godzina pomiaru zostały rozdzielone na cztery oddzielne kolumny, zawierające miesiąc, dzień, godzinę i minutę pomiaru.

Liczba wszystkich obserwacji, zebranych w pliku *energydata_complete.csv* wynosi 19735. Celem przyspieszenia obliczeń, algorytmy przedstawione w zadaniu zostały wykonane na danych zawierających 2000 pierwszych rekordów zmienna *testDataLength*. Wszystkie operacje dot. przygotowania danych są wykonane w funkcji *data_org*.

3. Selekcja atrybutów

Aby zapobiec nadmiernemu dopasowaniu, stosuje się selekcję atrybutów, która wybiera kilka najważniejszych atrybutów do późniejszego stworzenia modeli. Po zastosowaniu selekcji, modele oparte o ograniczoną liczbę atrybutów zwykle są lepsze od opartych o wszystkie atrybuty. Istnieje wiele metod selekcji atrybutów; w ramach projektu zostało sprawdzone kilka metod (w nawiasach umieszczono opcję typu funkcji *feature_selection*):

- prosty filtr statystyczny („*simple*”) - pomiędzy każdym z atrybutów a celem regresji stosuje się miarę statystyczną, która określa zależność celu od danego atrybutu (dalej „miara zależności”). Następnie wybiera się kilka atrybutów o największej „mierze zależności”. W ramach regresji pomiędzy atrybutami ciągłymi zastosowano współczynnik korelacji (Pearsona);
- bazująca na drzewach losowych („*rf*”) - w tym celu wykorzystano pakiet *randomForest* i jego wbudowaną opcję zwracającą parametr *IMPORTANCE* (bazujący na mierze MSE), o możliwości konfiguracji ilości drzew, w badaniu wykorzystano generowanie 500 drzew;
- metoda *RRELIEF* („*relief*”) - wersja algorytmu *RELIEF* do zastosowań w zadaniu regresji. Algorytm *RELIEF*, początkowo zaprojektowany dla zadania klasyfikacji binarnej, polega na losowym wybraniu obserwacji (jednego rekordu klasy+atrybuty). Następnie wyszukuje się k najbardziej podobnych obserwacji tej samej klasy, oraz k klasy przeciwnej. Dla każdego atrybutu oblicza się wagę istotności. Po wykonaniu K operacji, wykonuje się średnią wag istotności. Atrybuty segreguje się według wag istotności. W zadaniu regresji stosuje się inne funkcje obliczające wagę np. funkcję rozkładu. W projekcie $k=3$, $K=50$.

W ramach projektu stosuje się następujące podejście: dla każdej wymienionej metody wykonuje się selekcję połowy atrybutów ($part=0.5$) atrybutów, następnie wyznaczoną formułę aplikuje się do stworzenia modelu *rpart()*, i procedurze oceny (10-krotnej walidacji krzyżowej *model_eval()*)¹. Następnie największy współczynnik korelacji Pearsona wyznacza najlepszą metodę selekcji atrybutów oraz formułę do stworzenia modelu.

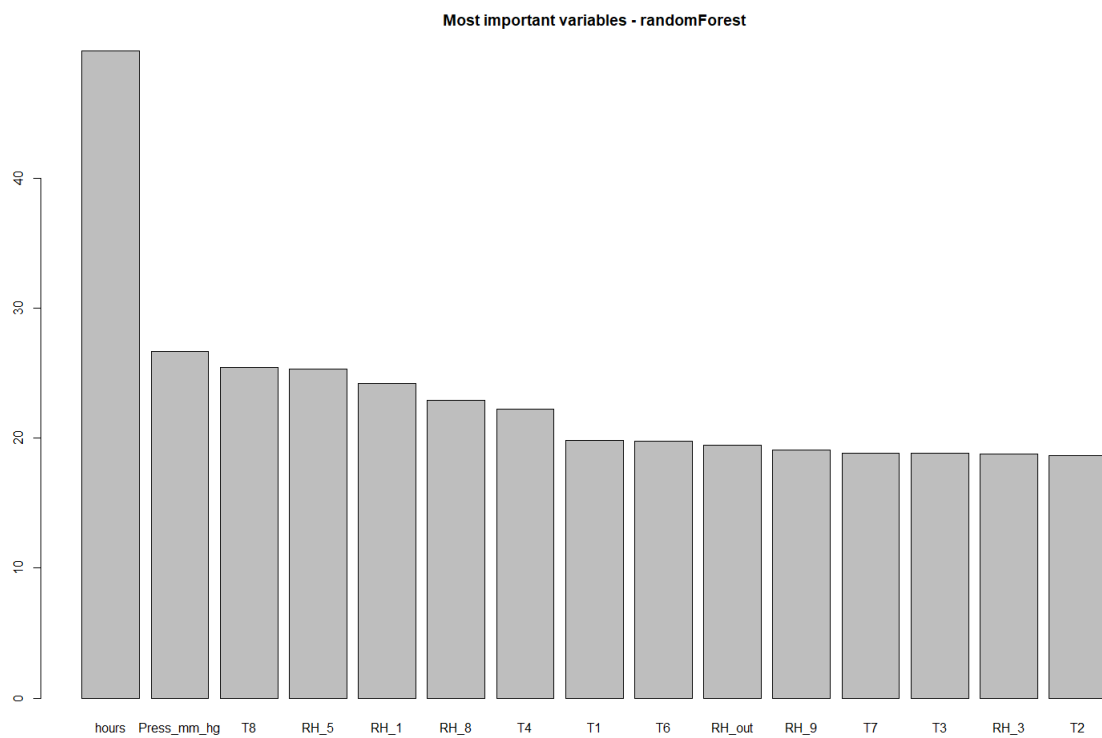
3.1. Wyniki selekcji atrybutów

Na rysunkach 3.4 – 3.6 przedstawiono wyniki każdej z selekcji. Dla *randomForest* i *simple.filter* atrybut *hours* dominuje nad pozostałymi. W tabeli 3.1 przedstawiono porównanie wyników każdej z selekcji. Wynika z tego, że atrybuty wyznaczone funkcją *simple.filter* pozwalają na najlepsze wyznaczenie modelu. Dla porównania przedstawiono też wynik walidacji krzyżowej dla modelu opartego o wszystkie atrybuty. Tylko selekcja atrybutów za pomocą *simple.filter* pozwala na poprawę dla obecnych warunków testowych (dostępne dane, ilość selekcionowanych argumentów, algorytm do walidacji krzyżowej). W związku z wynikami, atrybuty wyznaczono za pomocą prostego filtra statystycznego posłużą w dalszej konstrukcji modeli.

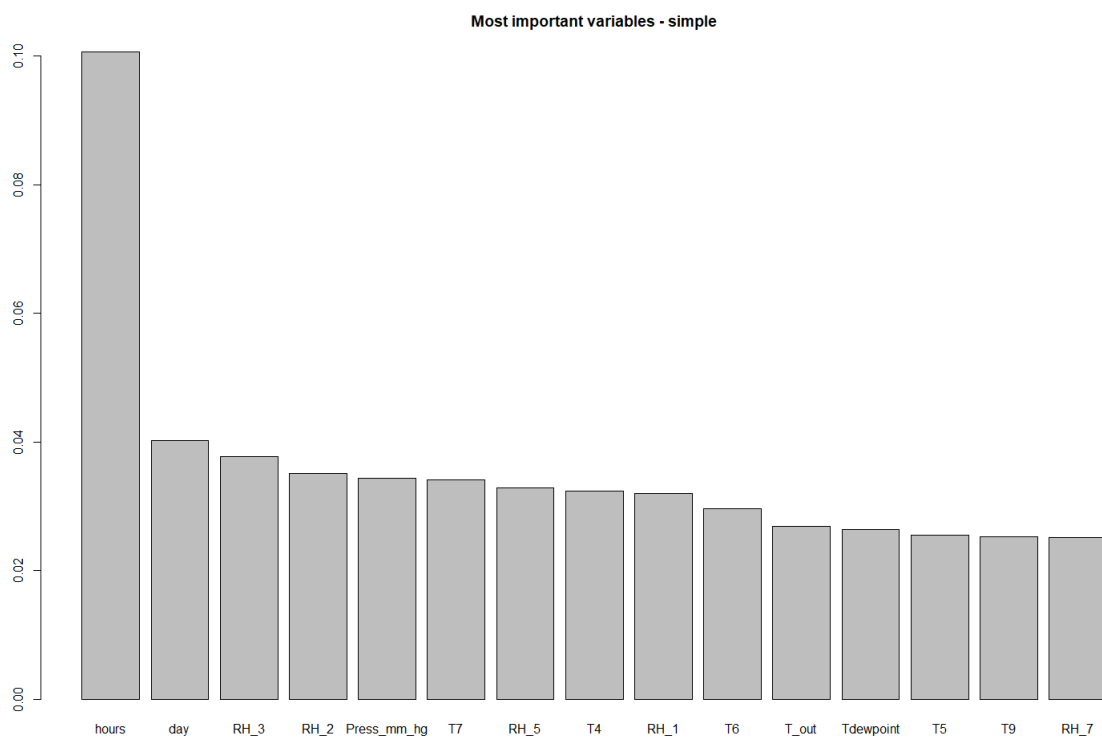
Tablica 3.1: Wyniki selekcji atrybutów — współczynniki korelacji

Parametr	<i>randomForest</i>	<i>simple</i>	<i>RELIEF</i>	bez selekcji
Appliances	0,555	0,616	0,553	0,585
lights	0,672	0,759	0,691	0,757

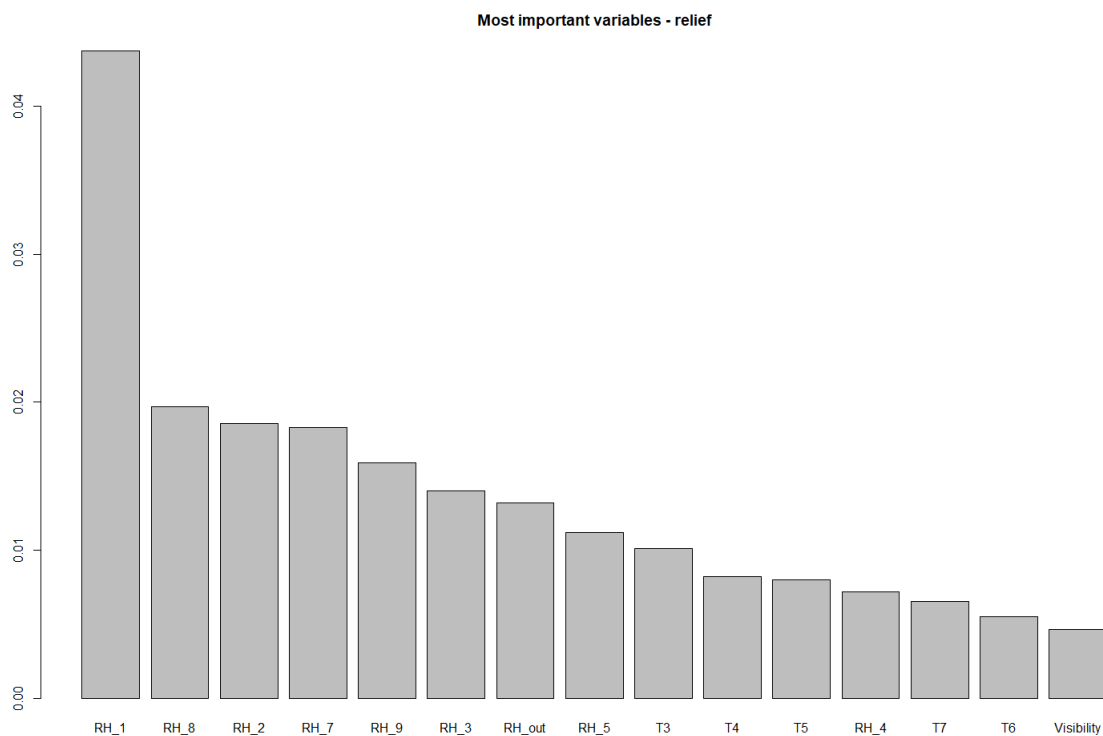
¹ Więcej nt. procedury walidacji krzyżowej w rozdziale 4.3



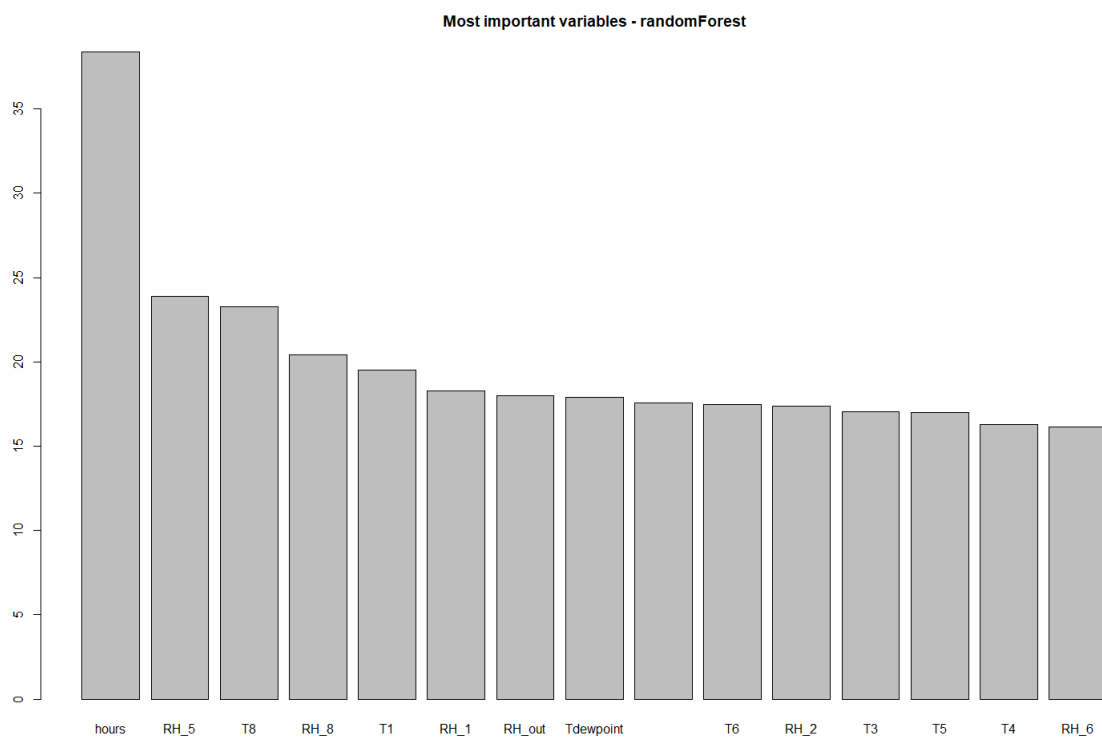
Rysunek 3.1: Wyniki selekcji z wykorzystaniem pakietu *randomForest*; cel: lights



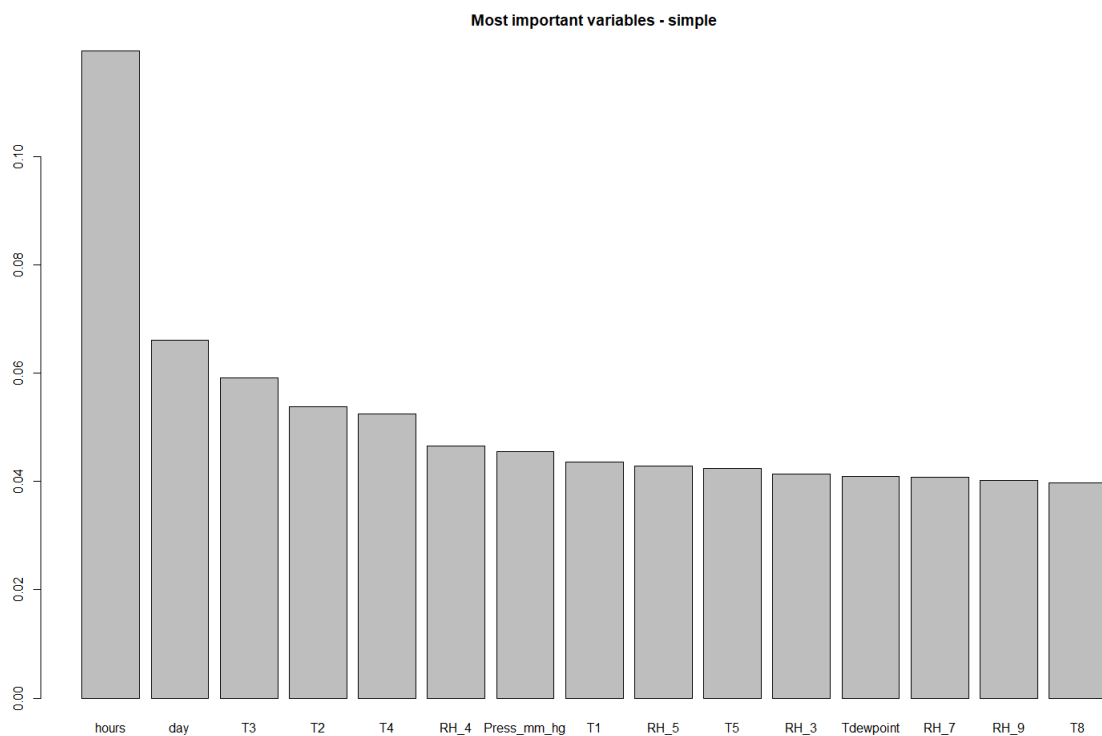
Rysunek 3.2: Wyniki selekcji z wykorzystaniem funkcji *simple.filter*; cel: lights



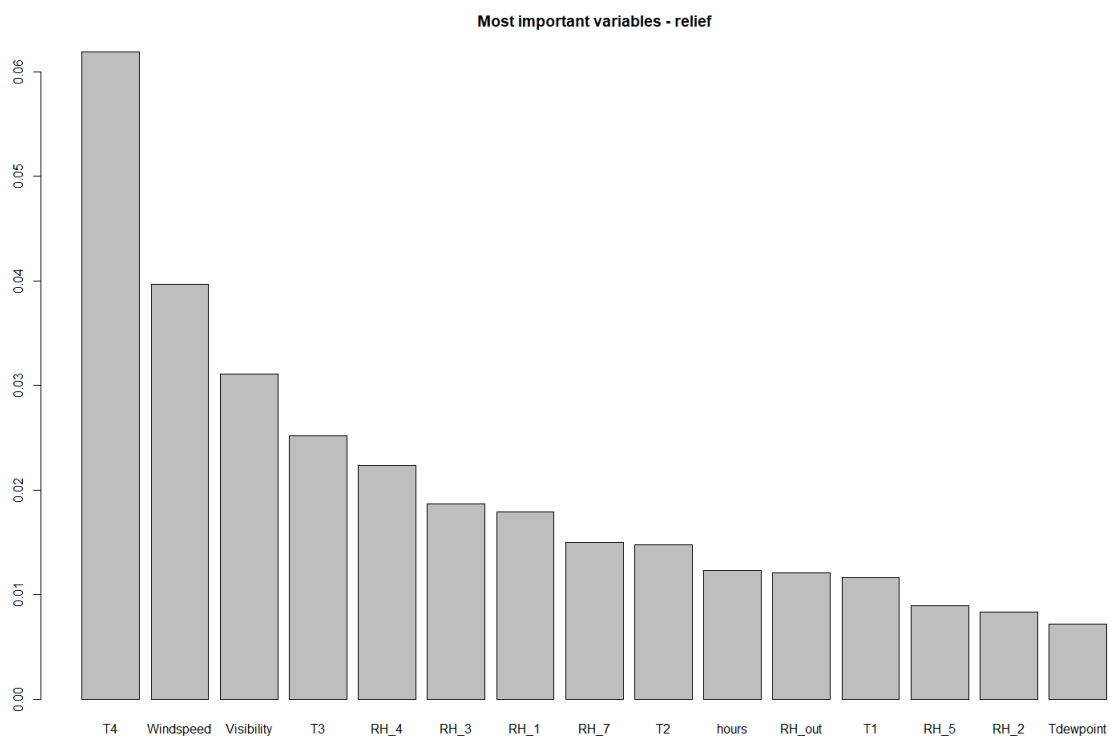
Rysunek 3.3: Wyniki selekcji z wykorzystaniem funkcji *rrelief.filter*; cel: lights



Rysunek 3.4: Wyniki selekcji z wykorzystaniem pakietu *randomForest*; cel: Appliances



Rysunek 3.5: Wyniki selekcji z wykorzystaniem funkcji *simple.filter*; cel: Appliances



Rysunek 3.6: Wyniki selekcji z wykorzystaniem funkcji *rrelief.filter*; cel: Appliances

4. Konstrukcja i ocena modeli

4.1. Metody konstrukcji modeli

W pierwszym kroku dokonano modelowania za pomocą liniowej regresji. Realizuje ją algorytm *lm()*. Argumentami algorytmu są tylko: formuła (cel+atrybuty) oraz zestaw danych w formie *data.frame*. Algorytm na podstawie średniej lub mediany wyznacza współczynniki funkcji liniowej tj. współczynnik kierunkowy i wyraz wolny. Następnie od każdego atrybutu wyznacza wagę współczynnika kierunkowego oraz jeden wyraz wolny. Zaletą tego algorytmu jest jego prostota i szybkość działania, zaś niewątpliwą wadą jest fakt, że większość procesów zachodzących w świecie nie da się opisać za pomocą liniowych zależności.

Następnie dokonano modelowania za pomocą drzew regresji (algorytm *rpart()*). Funkcja buduje model rekurencyjnie dzieląc zbiór na mniejsze i wyznaczając dla nich średnią. Dla metody „anova” dedykowanej do zadania regresji kryterium podziału wyznaczone jest na podstawie sum kwadratów dla danego węzła i dla jego potomnych. Algorytm może przyjąć także argumenty na minimalną liczbę podziałów *minsplit* oraz maksymalną głębokość drzewa *maxdepth* (czyli długość pomiędzy korzeniem a liśćmi).

Pojedyncze modele drzew regresji zazwyczaj cierpią z powodu wysokiej wariancji – jedną z metod jej redukcji jest tzw. Bagging (**B**ootstrap **a**ggregating). W ramach tej metody tworzona jest pewna liczba zbiorów "bootstrapowych". Dla każdego z tych zbiorów tworzy się nieprzycięte drzewo regresji. Następnie uśrednia się każdy z tych modeli, zmniejszając wariancję i redukując zbytnie dopasowanie. Bagging może zostać zrealizowany za pomocą pakietu *ipred* lub *caret*. Zasada działania modelowania przy pomocy tych pakietów jest podobna, pakiet *caret* pozwala natomiast na łatwą analizę istotności atrybutów. Celem porównania wyników realizowanych przez oba algorytmy, zdecydowano się na użycie ich obu.

Modele oparte o drzewo regresji cierpią z powodu faktu, że w liściach, które reprezentują pewny podzbiór przestrzeni (na której buduje się model), wynikiem jest pojedyncza liczba. Przykładowo dla zależności celu od jednego atrybutu, funkcja reprezentująca model może być nieciągła i stworzona z odcinków o zerowym współczynniku kierunkowym. Dużo lepiej byłoby aproksymować tę zależność funkcją kawałkami liniową. W tym celu wykorzystuje się regresję kawałkami liniową (*grow.modtree* z pakietu *dmr.regtree*). Za pomocą listy *plr_args* ograniczono głębokość drzewa do 10 oraz wymuszono co najmniej 2 podziały.

Modele opisane w tym podrozdziale zostały ocenione za pomocą procedury k-krotnej walidacji krzyżowej z wykorzystaniem miar jakości opisanych dalej.

4.2. Miary jakości

Dla zbudowanych modeli oblicza się następujące miary jakości:

1. CC - współczynnik korelacji liniowej Pearsona

$$CC = \frac{cov(P,A)}{var(P) \cdot var(A)}$$

2. MSE - błąd średniokwadratowy

$$MSE = \frac{(p_1 - a_1)^2 + \dots + (p_n - a_n)^2}{n}$$

3. RMSE - pierwiastek z błędu średniokwadratowego

$$RMSE = \sqrt{\frac{(p_1 - a_1)^2 + \dots + (p_n - a_n)^2}{n}}$$

4. MAE - średni błąd względny

$$MAE = \frac{|p_1 - a_1| + \dots + |p_n - a_n|}{n}$$

5. RSE - względny błąd kwadratowy

$$RSE = \frac{(p_1 - a_1)^2 + \dots + (p_n - a_n)^2}{(a_1 - \bar{a})^2 + \dots + (a_n - \bar{a})^2}$$

6. RRSE - pierwiastek ze względnego błędu kwadratowego

$$RRSE = \sqrt{\frac{(p_1 - a_1)^2 + \dots + (p_n - a_n)^2}{(a_1 - \bar{a})^2 + \dots + (a_n - \bar{a})^2}}$$

7. RAE - błąd względny

$$RAE = \frac{|p_1 - a_1| + \dots + |p_n - a_n|}{|a_1 - \bar{a}| + \dots + |a_n - \bar{a}|}$$

4.3. Procedury oceny

Aby móc ocenić model pod względem przydatności zastosowano metodę k-krotnej walidacji krzyżowej. Kod zawarto w funkcji `model_eval()`. Zbiór testowy jest dzielony losowo na k podzbiorów równej wielkości. W kolejnych iteracjach każdy ze zbiorów jest traktowany jako zbiór testowy, podczas gdy na reszcie danych buduje się model. Następnie modele są uśredniane i następuje predykcja. Po predykcji modelu na zbiorze testowym wyznacza się miary jakości opisane w 4.2.

4.4. Wyniki walidacji krzyżowej

Wyniki 10-krotnej walidacji krzyżowej zostały przedstawione w tabeli 4.1. Przedstawia ona miary jakości wyznaczone za pomocą tej procedury dla każdego algorytmu. Zauważa się przewagę metod *bootstrapowych* nad innymi. Błędy i współczynniki korelacji dla tych metod osiągają najmniejsze wartości. Ewentualne różnice pomiędzy wynikami algorytmów z pakietów *ipred* i *caret* można tłumaczyć losowością procesu tworzenia ich modelu i/lub procedury oceny. Zauważa się też różnicę pomiędzy predykcją celu *Appliances* a *lights*. Wydaje się to być zgodne z intuicją — zwykle światła włącza się w dzień, a wyłącza w nocy. Z kolei różne urządzenia AGD stosuje się w różnych okresach, więc znalezienie szczególnej zależności może być skomplikowanym zadaniem.

Tablica 4.1: Walidacja krzyżowa — porównanie parametrów

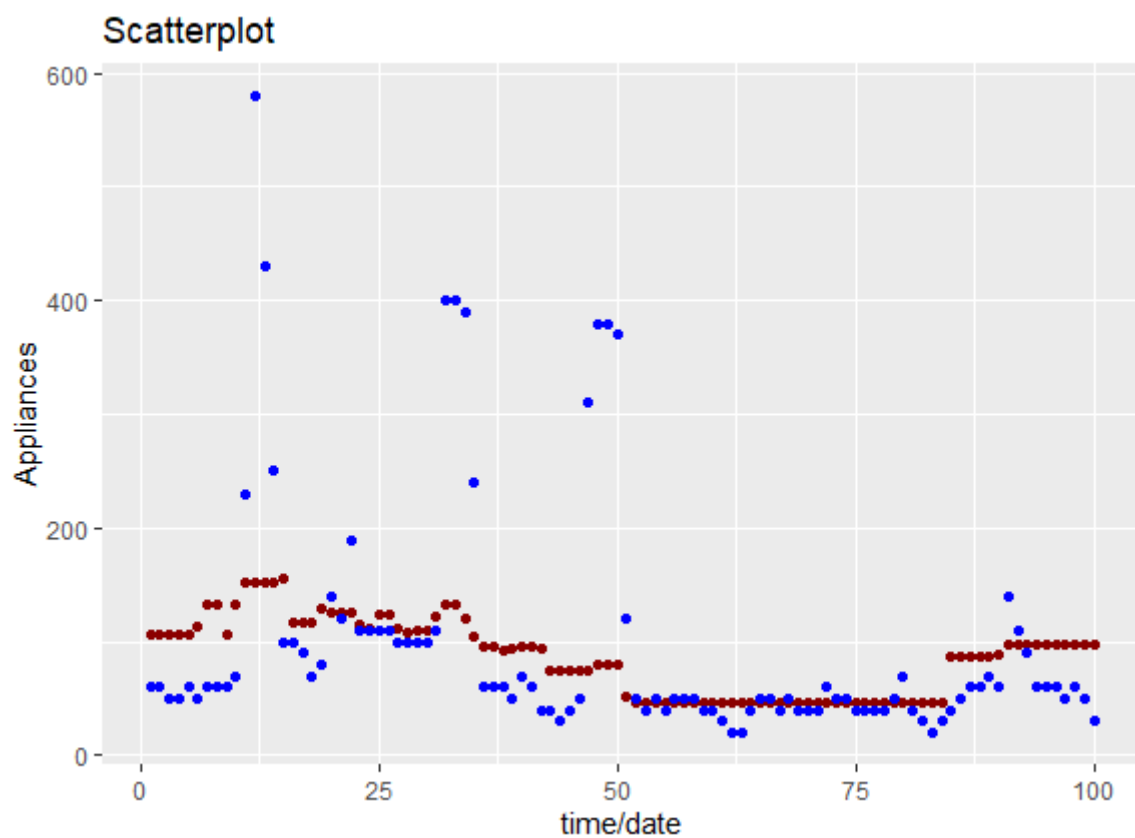
Appliances	MSE	RRSE	MAE	RMSE	RAE	CC	RSE
<i>lm()</i>	14914.800	2.5702027	75.20398	122.12616	1.9553172	0.3425775	6.6059421
<i>rpart()</i>	11646.595	1.2051757	58.13124	107.91939	1.0320860	0.5698647	1.4524484
<i>ipred</i>	9016.412	1.3234627	53.02221	94.95479	1.0308015	0.6983742	1.7515534
<i>PLR</i>	266453.137	0.9929399	78.68274	516.19099	0.8137909	0.1532704	0.9859296
<i>caret</i>	8958.720	1.2985949	52.82693	94.65051	1.0113670	0.6991588	1.6863487
lights	MSE	RRSE	MAE	RMSE	RAE	CC	RSE
<i>lm()</i>	63.61679	1.6605096	5.640188	7.976013	1.4319283	0.50520939	2.7572920
<i>rpart()</i>	38.27473	0.8563671	3.815467	6.186657	0.7194938	0.74393671	0.7333646
<i>ipred</i>	30.00860	0.8274880	3.578404	5.478011	0.7473722	0.81096864	0.6847364
<i>PLR</i>	7264.81589	0.9960550	9.868371	85.233889	0.7595684	0.09086541	0.9921255
<i>caret</i>	30.92202	0.8363734	3.593142	5.560757	0.7441449	0.80317142	0.6995204

5. Wnioski

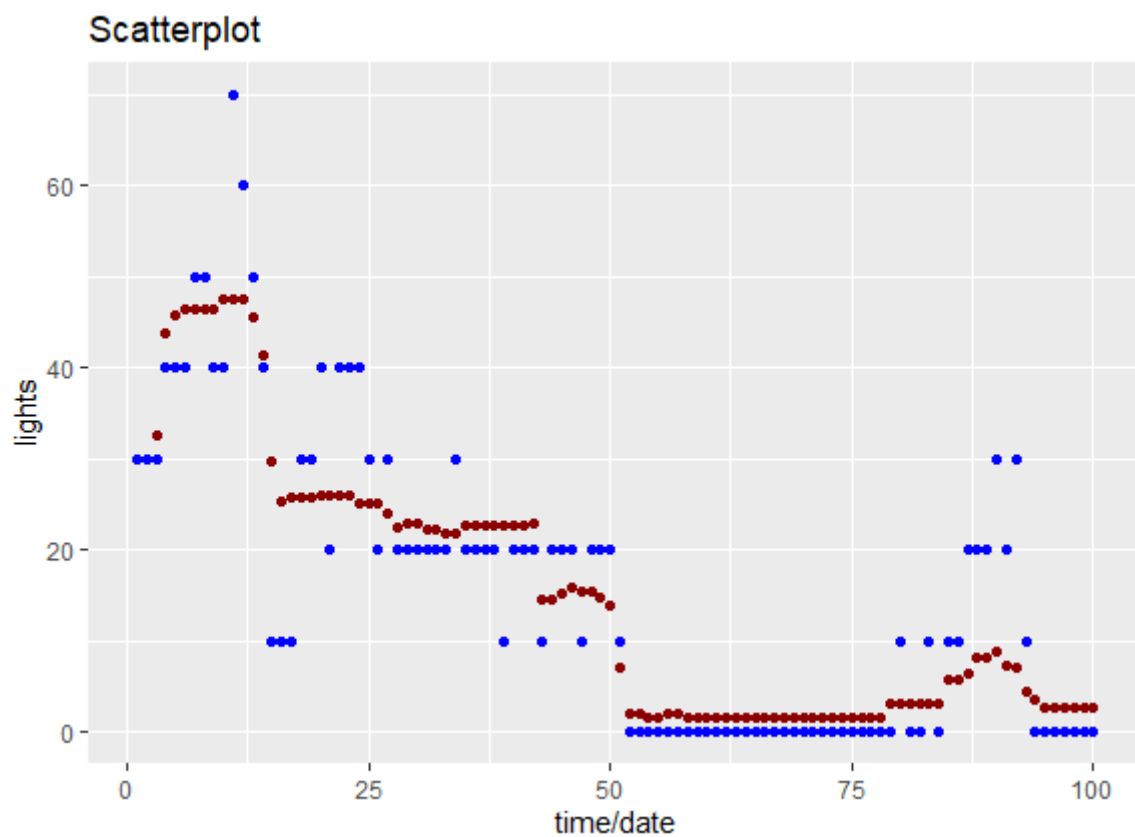
Wykonano analizę danych dotyczących pomiarów oddzielnie dla zużycia energii przez sprzęty AGD oraz dla zużycia energii przez oświetlenie. Dokonano ograniczenia zbioru danych i wstępnej obróbki danych.

Zrealizowano selekcję atrybutów. Modele *rpart* oparte o wyselekcjonowane atrybuty okazały się nieznacznie lepsze (prosty filtr statystyczny oparty o współczynnik korelacji Pearsona) lub gorsze (randomForest, RELIEF). Jednak dzięki mniejszemu zbiorowi atrybutów użytych do zadania regresji zaoszczędzono na obliczeniach.

Skonstruowano 5 różnych modeli: liniowy (*lm()*), oparty o drzewo regresji (*rpart*), dwa oparte o „bootstrapowane” drzewo regresji tzw. metoda Bagging (*ipred*, *caret*) oraz oparty o regresję kawałkami liniową (*grow.modtree*). Poddano je k-krotnej walidacji krzyżowej. Najlepsze rezultaty osiągnięto dla metody Bagging. Na rysunkach 5.1 i 5.2 przedstawiono przykład porównania wartości predykowanych i prawdziwych dla obu celów. Dla wybranych metod lepsze możliwości predykcji wykazuje cel *lights*.



Rysunek 5.1: Metoda: *bagging (ipred)*; cel: *Appliances*; czerwony — predykowane, niebieski — prawdziwe



Rysunek 5.2: Metoda: *bagging (ipred)*; cel: *lights*; czerwony — predykowane, niebieski — prawdziwe

```

1 #setwd("E:/Documents/Studies/MOW/MOW_project")
2 #setwd("Z:/Offtop/MOW/Project/MOW_project")
3 #setwd("F:/GitHub/MOW_project")
4 setwd(".")
5
6 # Main script
7 # Krzysztof Belewicz
8 # Pawel Pienczuk
9 # MOW - Metody odkrywania wiedzy
10 # FEIT WUT Class
11
12 library(rsample)      # data splitting
13 library(dplyr)        # data wrangling
14 library(ipred)        # bagging
15 library(caret)        # bagging
16 library(dmr.regtree)
17
18 rm(list = ls())
19
20 # to make the final comparision change line 122 "test_data$..."
21 target_t <- "Appliances"
22
23 # DATA COLLECTION AND ORGANIZATION GOES HERE
24 # 10 to try function, 2000 to test try algorithm, 19735 to full
25 source('R/data_org.R')
26
27 testDataLength <- 2000
28 complete_data <- read.csv("energydata_complete.csv")
29
30 test_data <- dataOrganization(complete_data, target_t, testDataLength)
31
32 # FEATURE SELECTION GOES HERE
33
34 source('R/feature_selection.R')
35 source("R/model_eval.R")
36
37 featSelTypes <- list("rf", "simple", "relief")
38 corMeasures <- NULL
39 formula <- NULL
40 # for every type of feature selection method
41 for (k in 1:length(featSelTypes)){
42   res <- feature_selection(formula_full = as.formula(paste(target_t, "~.")),
43                           target = target_t,
44                           test_data = test_data,
45                           type = featSelTypes[k],
46                           part = 0.5,
47                           trees_num=500
48   )
49
50   formula <- c(formula, as.vector(res$attr_part[1]))
51
52   # create and cross-validate rpart model based on particular feature sel. method
53   crossval.rpart <- model_eval(test_data = test_data,
54                               fun = rpart,
55                               formula = formula[k],
56                               crossval_number = 10,
57                               args = list(method="anova")
58   )
59   corMeasures <- c(corMeasures, crossval.rpart$COR)

```

```

60 }
61
62 # with all attributes
63 crossval.rpart <- model_eval(test_data = test_data,
64                             fun = rpart,
65                             formula = paste(target_t, "~."),
66                             args = list(method = "anova"),
67                             crossval_number = 10
68 )
69 corMeasures <- c(corMeasures, crossval.rpart$COR)
70 formula <- c(formula, paste(target_t, "~."))
71
72 #selection of best formula
73 k <- which.max(corMeasures)
74 formula <- formula[k]
75
76 # EVALUATION PROCEDURES GOES HERE
77
78 crossval.lm <- model_eval(test_data = test_data,
79                           fun = lm,
80                           formula = formula,
81                           crossval_number = 10
82 )
83 crossval.rpart <- model_eval(test_data = test_data,
84                              fun = rpart,
85                              formula = formula,
86                              crossval_number = 10
87 )
88
89 crossval.bagging <- model_eval(test_data = test_data,
90                               fun = bagging,
91                               formula = formula,
92                               crossval_number = 10
93 )
94 plr_args <- list(minsplit=2, maxdepth=8)
95
96 crossval.plr <- model_eval(test_data=test_data,
97                           fun = grow.modtree,
98                           formula = formula,
99                           crossval_number = 10,
100                          args = plr_args
101 )
102
103 ctrl <- trainControl(method = "cv", number = 10)
104 args_t <- c(method="treebag", trControl=ctrl)
105
106 crossval.caret <- model_eval(test_data = test_data,
107                              fun = train,
108                              formula = formula,
109                              crossval_number = 10,
110                              args=args_t
111 )
112
113 crossval.all <- rbind(crossval.lm, crossval.rpart, crossval.bagging, crossval.plr, crossval.caret)
114
115 bestModelInd <- which.max(crossval.all$COR)
116
117 # bagging was the best model, so there is a creation
118 bagmodel <- bagging(as.formula(formula), test_data)
119
120 pred = predict(bagmodel, test_data)

```

```
121
122 comparison = data.frame(k=1:nrow(test_data), pred=pred, true=test_data$Appliances)
123
124 ggplot(comparison[1:100,], x="time&date", y=target_t) +
125   geom_point(aes(x=k, y=pred), color = "darkred") +
126   geom_point(aes(x=k, y=true), color = "blue") +
127   labs(title="Scatterplot") +
128   xlab("time/date") +
129   ylab(target_t)
```

```

1 #data_org.R
2 # data collecting and organization
3
4 # training_data <- read.csv("training.csv")
5
6 #' Title
7 #'
8 #' @param data data to be organized
9 #' @param target "Appliances" or "lights"
10 #' @param testDataLength length of output subset (from 1 to testDataLength)
11 #'
12 #' @return organised data set (or subset, if testDataLength specified),
13 #'
14 dataOrganization <- function(data,target,testDataLength=nrow(data)){
15
16     #day_mon = day + 30 * month
17     #min_hour = minutes + 60 * hours
18     if (!is.data.frame(data)){
19         message("data_is_no_data.frame_type")
20         stop()
21     }
22     if ((target!="Appliances") & (target!="lights"))
23     {
24         message("Target_is_not_\\"Appliances\\"_neither_\\"lights\\"")
25         stop()
26     }
27     if (testDataLength<2|testDataLength>nrow(data)){
28         message("testDataLength_must_be_between_2_and_length(data)")
29         stop()
30     }
31
32
33     if (target == "Appliances"){
34         out_data <- data.frame(
35             Appliances = data$Appliances,
36             month = as.numeric(substring(data$date,6,7)),
37             day = as.numeric(substring(data$date,9,10)),
38             hours = as.numeric(substring(data$date,12,13)),
39             minutes = as.numeric(substring(data$date,15,16)),
40             # day_mon = day_mon,
41             # min_hour = min_hour,
42             T1 = data$T1,
43             T2 = data$T2,
44             T3 = data$T3,
45             T4 = data$T4,
46             T5 = data$T5,
47             T6 = data$T6,
48             T7 = data$T7,
49             T8 = data$T8,
50             T9 = data$T9,
51             RH_1 = data$RH_1,
52             RH_2 = data$RH_2,
53             RH_3 = data$RH_3,
54             RH_4 = data$RH_4,
55             RH_5 = data$RH_5,
56             RH_6 = data$RH_6,
57             RH_7 = data$RH_7,
58             RH_8 = data$RH_8,
59             RH_9 = data$RH_9,

```

```

60     T_out = data$T_out,
61     RH_out = data$RH_out,
62     Press_mm_hg = data$Press_mm_hg,
63     Windspeed = data$Windspeed,
64     Visibility = data$Visibility,
65     Tdewpoint = data$Tdewpoint,
66     rv1 = data$rv1,
67     rv2 = data$rv2
68 )
69 } else if(target == "lights"){
70   out_data <- data.frame(
71     lights = data$lights,
72     month = as.numeric(substring(data$date, 6, 7)),
73     day = as.numeric(substring(data$date, 9, 10)),
74     hours = as.numeric(substring(data$date, 12, 13)),
75     minutes = as.numeric(substring(data$date, 15, 16)),
76     # day_mon = day_mon,
77     # min_hour = min_hour,
78     T1 = data$T1,
79     T2 = data$T2,
80     T3 = data$T3,
81     T4 = data$T4,
82     T5 = data$T5,
83     T6 = data$T6,
84     T7 = data$T7,
85     T8 = data$T8,
86     T9 = data$T9,
87     RH_1 = data$RH_1,
88     RH_2 = data$RH_2,
89     RH_3 = data$RH_3,
90     RH_4 = data$RH_4,
91     RH_5 = data$RH_5,
92     RH_6 = data$RH_6,
93     RH_7 = data$RH_7,
94     RH_8 = data$RH_8,
95     RH_9 = data$RH_9,
96     T_out = data$T_out,
97     RH_out = data$RH_out,
98     Press_mm_hg = data$Press_mm_hg,
99     Windspeed = data$Windspeed,
100    Visibility = data$Visibility,
101    Tdewpoint = data$Tdewpoint,
102    rv1 = data$rv1,
103    rv2 = data$rv2
104  )
105 } else {
106   print("Target_can_be_only_lights_or_Appliances")
107   stop()
108 }
109
110 # checking scirpts on smaller dataset, comment to use full
111 out_data <- out_data[1:testDataLength,]
112
113 # unused data to forget
114 }

```

```

1  # feature_selection.R
2
3  library(randomForest)
4  library(rpart)
5  library(rpart.plot)  # plotting regression trees
6  library(Metrics)     # RMSE
7  library(dmr.disc)
8  library(dmr.stats)
9  library(dmr.attrsel)
10
11  #' @title Feature Selection
12  #'
13  #' @param test_data data to perform feature ranking
14  #' @param formula_full full formula, with targets and all attributes
15  #' @param target target (character type)
16  #' @param type "rf" - randomForest IMPORTANCE-based ranking (default),
17  #' "relief" - RELIEF algorithm
18  #' "simple" - simple filter algorithm
19  #' @param part part of attributes returned by feature selection, 0 < part <=1 (default=0.5)
20  #' @param trees_num numbers of trees (randomForest) or bootstrap sets (bootstrap)
21  #'
22  #' @return formula with selected attributes
23  #'
24  #'
25  #'
26  feature_selection <-< function(formula_full,target,test_data,type="rf",part=1,trees_num=10)
27
28  if ( !is.data.frame(test_data) ){
29    message("Pass_data_frame_format")
30    stop()
31  }
32  if (trees_num<1) {
33    message("Trees_number_should_be_>=1")
34    stop()
35  }
36  if (part>1|part<=0) {
37    message("Parts_of_atrrributes_must_be_>0_and_<=1")
38    stop()
39  }
40
41  print("FEATURE_SELECTION")
42  count <- ceiling((ncol(test_data)-1)*part)
43
44  if (type=="rf"){
45    full_model_RF <- randomForest(formula = formula_full,
46                                  data = test_data,
47                                  importance = TRUE,
48                                  ntree=trees_num)
49    importance_RF <- data.frame(importance(full_model_RF, type=1), "k"=1:(ncol(test_data)-
50    res <- data.frame("importance"=importance_RF[order(importance_RF[,1],decreasing = TRUE
51                    "k"=importance_RF$k[order(importance_RF[,1],decreasing = TRU
52                    "attr"=rownames(importance_RF)[order(importance_RF[,1], dechr
53
54    # plotting most important parameters
55    barplot(res$importance[1:count],main="Most_important_variables_-_randomForest",names.a
56
57    # varImpPlot(x=full_model_RF,
58    #             n.var=count,
59    #             type=1,

```



```

60     #           main="Most important variables - randomForest importance")
61
62     # passing most important attributes from feature selection
63     count <- ceiling((ncol(test_data)-1)*part)
64     attr_part <- paste(res$attr[2:count], collapse = "+")
65     attr_part <- paste(target, "~", attr_part)
66     out <- data.frame(attr_part, res)
67     return(out)
68 }
69 else if(type=="relief"){
70
71     res <- rrelief.filter(formula = formula_full,
72                           data = test_data,
73                           k=3,
74                           K=20)
75
76     names_t <- names(res)
77
78     # plotting most important parameters
79     barplot(as.vector(res)[1:count], main="Most_important_variables_-_relief", names.arg = n
80
81     attr_part <- paste(names_t[1:count], collapse = "+")
82     attr_part <- paste(target, "~", attr_part)
83     out <- data.frame(attr_part, res)
84     return(out)
85 }
86
87 else if (type=="simple"){
88
89     res <- simple.filter(formula = formula_full,
90                          data = discnm.eqfreq(~., test_data, 10),
91                          dd=symunc)
92
93     names_t <- names(res)
94
95     # plotting most important parameters
96     barplot(as.vector(res)[1:count], main="Most_important_variables_-_simple", names.arg = n
97
98     attr_part <- paste(names_t[1:count], collapse = "+")
99     attr_part <- paste(target, "~", attr_part)
100    out <- data.frame(attr_part, res)
101    return(out)
102 }
103
104 # else if (type=="wrapper"){
105 #
106 #     res <- wrapper.filter.select(formula = formula_full,
107 #                                  data = test_data,
108 #                                  utils = rpart(formula_full, test_data),
109 #                                  alg = rpart,
110 #                                  args = list(minsplit=2),
111 #                                  initf = asel.init.none,
112 #                                  nextf = asel.next.forward
113 #     )
114 #
115 #
116 #     names_t <- res$subset
117 #
118 #     # plotting most important parameters
119 #     barplot(res$eval[1:count], main="Most important variables - wrapper", names.arg = name
120 #

```

```
121 # attr_part <- paste(names_t, collapse = "+")
122 # attr_part <- paste(target, "~", attr_part)
123 # out <- data.frame(attr_part, res)
124 # return(out)
125 # }
126 else{
127   message("Wrong_type_of_feature_selection: \"relief\" \"rf\" or \"simple\"")
128   stop()
129 }
130 }
```

Plik: model_eval.R

```
1 # model evaluation
2
3 library(dmr.claseval)
4
5 #' @title Model Evaluation
6 #'
7 #' @param test_data - data to perform cross-validation
8 #' @param fun - regression algorithm (e.g. train,bagging,lm) default=rpart
9 #' @param formula - regression formula
10 #' @param crossval_number - number of cross-validations, default=10
11 #' @param args - passed arguments into regression algorithm, default=NULL
12 #'
13 #' @return data frame with measures in following order: MSE, RRSE, MAE, RMSE, RAE, COR, RSE
14 #'
15 #'
16 model_eval <-<= function(test_data, fun=rpart, formula, crossval_number=10, args=NULL) {
17
18   if ( !is.data.frame(test_data) ) {
19     message("Pass_data_frame_format")
20     stop()
21   }
22   if ( !is.character(formula) ) {
23     message("Pass_string_into_formula_parameter")
24     stop()
25   }
26   if (crossval_number<1) {
27     message("Cross_validations_number_must_be_greater_than_0")
28     stop()
29   }
30   if (crossval_number==1) {
31     warning("No_cross-validation")
32   }
33   temp_model <- crossval(fun,
34                           as.formula(formula),
35                           test_data,
36                           k=crossval_number,
37                           args = args)
38   MSE <- mse(temp_model$pred, temp_model$true)
39   RRSE <- rrse(temp_model$pred, temp_model$true)
40   MAE <- mae(temp_model$pred, temp_model$true)
41   RMSE <- rmse(temp_model$pred, temp_model$true)
42   RAE <- rae(temp_model$pred, temp_model$true)
43   COR <- cor(temp_model$pred, temp_model$true, method="pearson")
44   RSE <- rse(temp_model$pred, temp_model$true)
45
46   out <- data.frame(MSE, RRSE, MAE, RMSE, RAE, COR, RSE)
47   return(out)
48 }
```
