Metody Odkrywania Wiedzy

Dokumentacja końcowa projektu

„Predykcja zużycia energii na podstawie danych czujnikowych”

Krzysztof Belewicz
Paweł Pieńczuk

26 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 ([$^\circ C$], [$\%$]),
- pomiarów temperatury i wilgotności dla zewnętrznej, północnej strony budynku ([$^\circ C$], [$\%$]),
- danych z pobliskiej stacji pogodowej:
  - temperatura powietrza [$^\circ C$],
  - temperatura punktu rosy [$^\circ C$],
  - 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 type 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 zwracają 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.

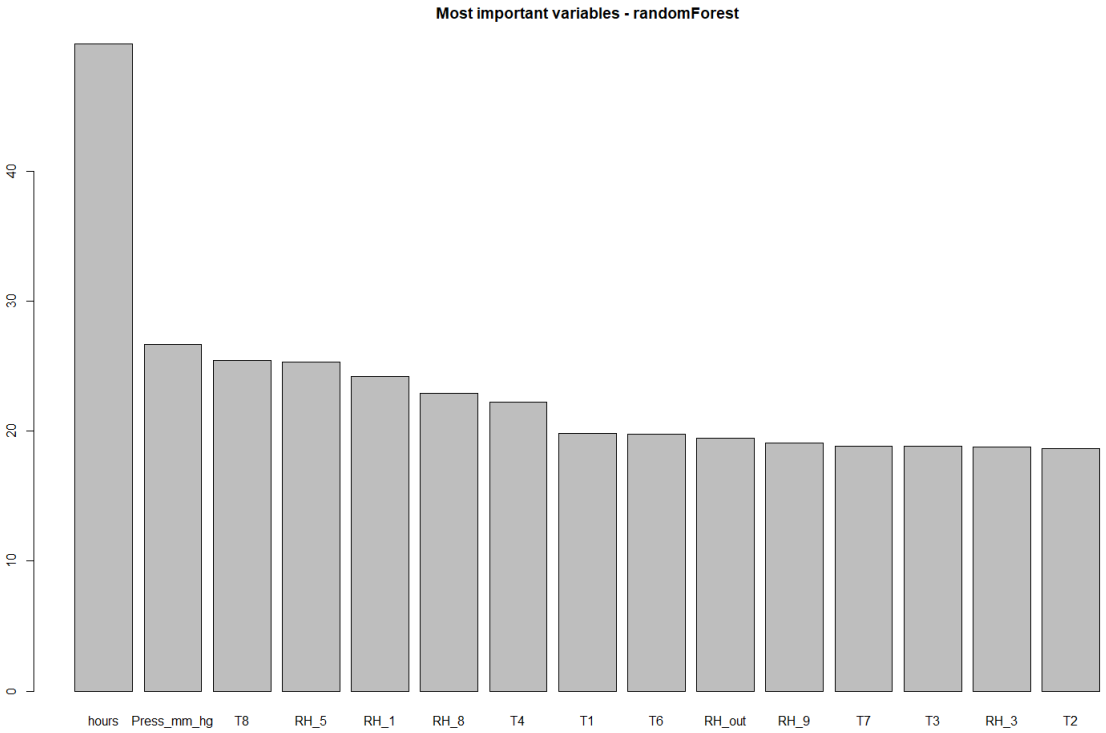
3.1. Wyniki selekcji atrybutów

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 selekcjonowanych argumentów, algorytm do walidacji krzyżowej). W związku z wynikami, atrybuty wyznaczono za pomocą prostego filtru statystycznego posłużą w dalszej konstrukcji modelów.

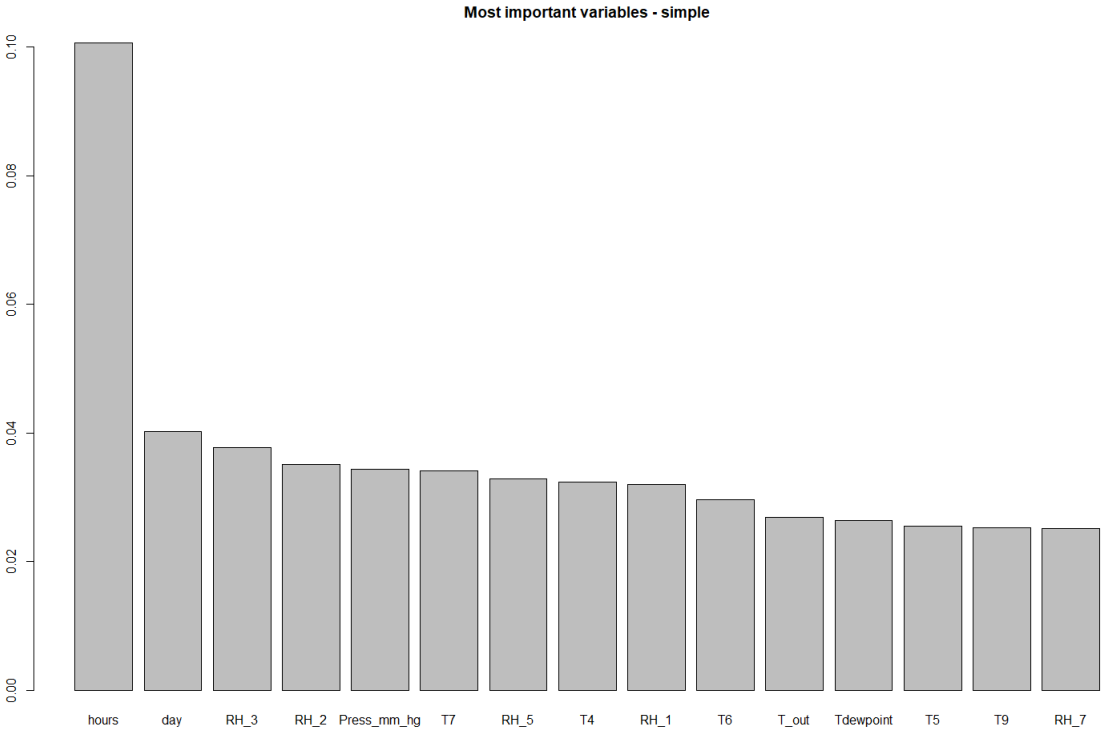
Tablica 3.1: Wyniki selekcji atrybutów — współczynniki korelacji

Parametr	***randomForest***	***simple***	***RELIEF***	**bez selekcji**
Appliances	0,555	0,616	0,553	0,585
lights	0,672	0,759	0,691	0,757

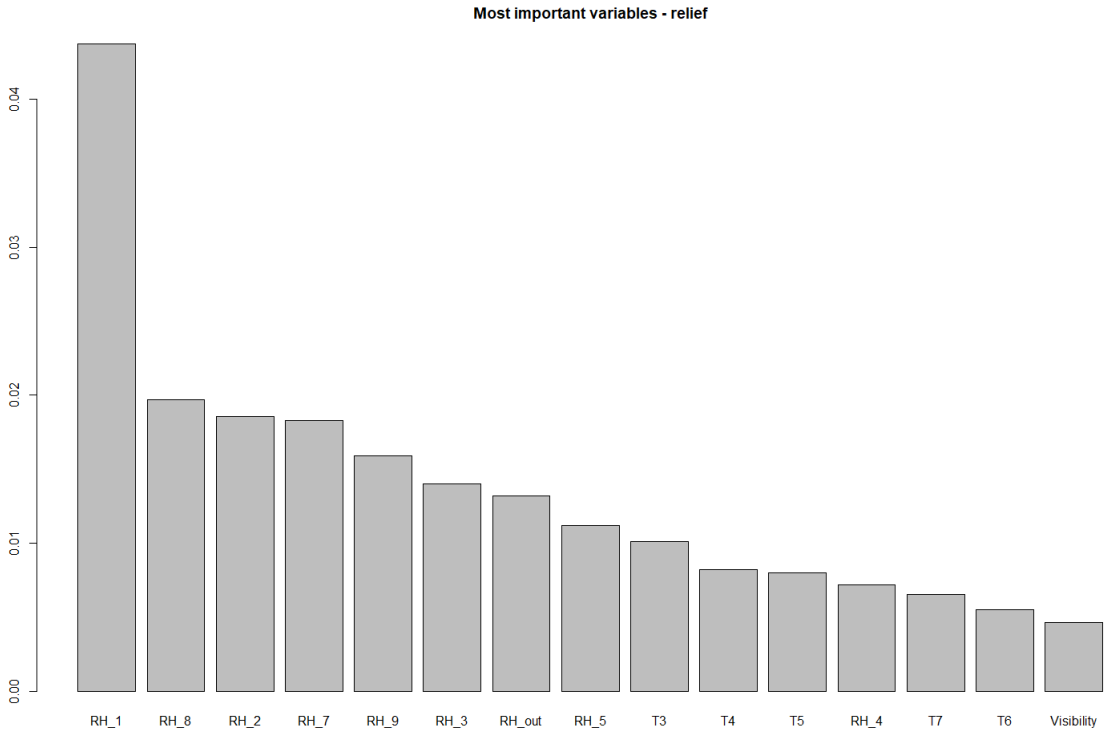
[1] Więcej nt. procedury walidacji krzyżowej w rozdziale 4.3



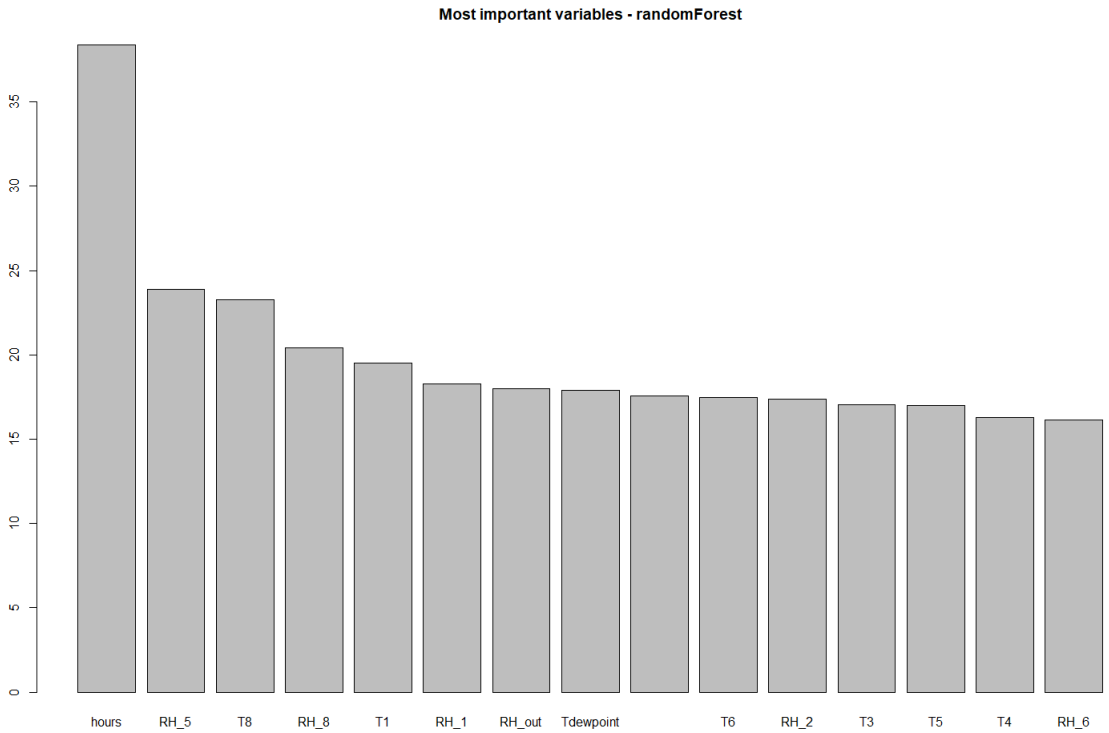
Rysunek 3.1: Wyniki selekcji z wykorzystaniem pakietu *randomForest*; cel: lights



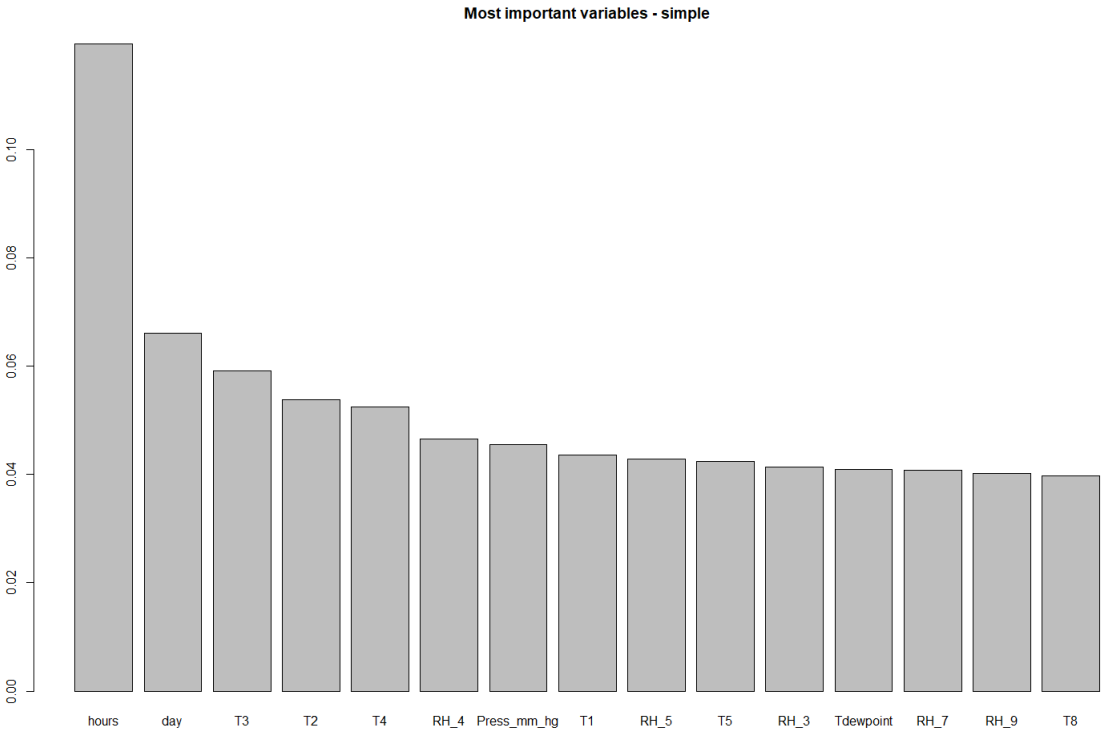
Rysunek 3.2: Wyniki selekcji z wykorzystaniem funkcji *simple.filter*; cel: lights



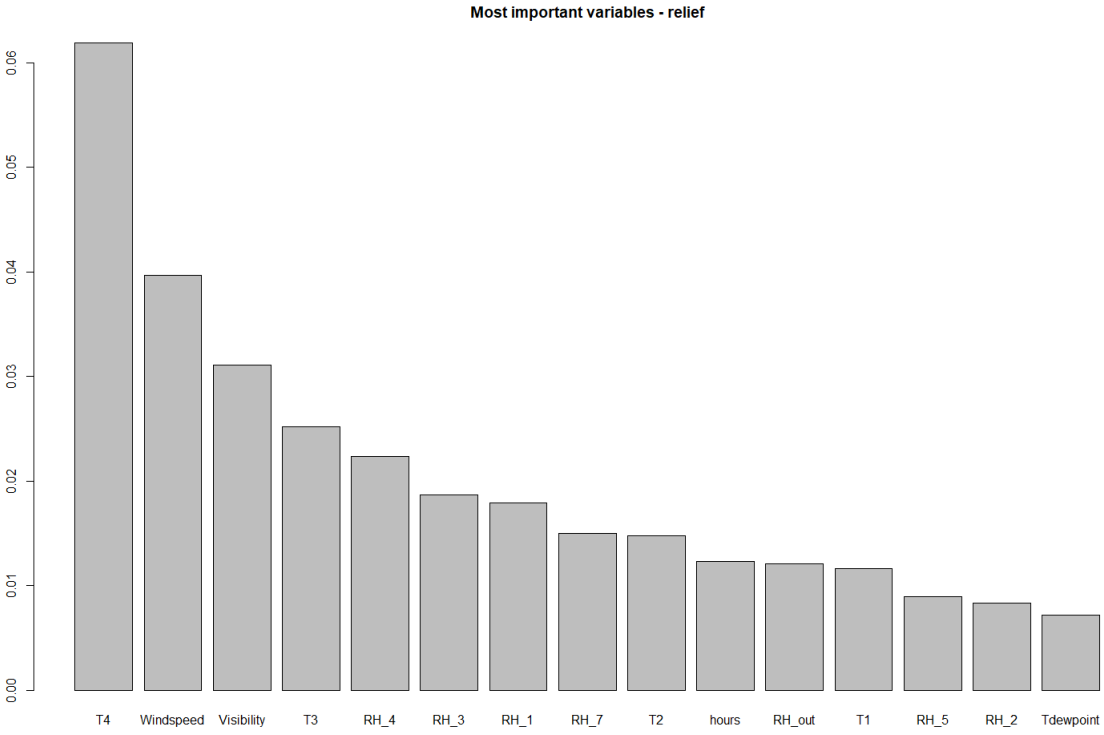
Rysunek 3.3: Wyniki selekcji z wykorzystaniem funkcji *rrelief.filter*; cel: lights



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 atrybuty wyznacza wagę współczynniku 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 **agg**regat**ing**). 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

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

3. RMSE - pierwiastek z błędu średniokwadratowego

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

4. MAE - średni błąd względny

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

5. RSE - względny błąd kwadratowy

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

6. RRSE - pierwiastek ze względnego błędu kwadratowego

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

7. RAE - błąd względny

$$RAE = \frac{|p_1 - a_1| + ... + |p_n - a_n|}{|a_1 - \overline{a}| + ... + |a_n - \overline{a}|}$$

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

5. Wnioski

TODO

Plik: main.r

```
# main v1.1
#setwd("E:/Documents/Studies/MOW/MOW_project")
#setwd("Z:/Offtop/MOW/Project/MOW_project")
#setwd("F:/GitHub/MOW_project")
setwd(".")

library(rsample)      # data splitting
library(dplyr)        # data wrangling
library(ipred)        # bagging
library(caret)        # bagging
library(dmr.regtree)

rm(list = ls())


target_t <- "lights"

# DATA COLLECTION AND ORGANIZATION GOES HERE
# 10 to try function, 1000 to test try algorithm, 14803 to full
source('R/data_org.R')

testDataLength <- 14803
complete_data <- read.csv("energydata_complete.csv")

test_data <- dataOrganization(complete_data, target_t,testDataLength)

# FEATURE SELECTION GOES HERE

source('R/feature_selection.R')
source("R/model_eval.R")

featSelTypes <- list("rf", "simple","relief")
corMeasures <- NULL
formula <- NULL
# for every type of feature selection method
for (k in 1:length(featSelTypes)){
  res <- feature_selection(formula_full = as.formula(paste(target_t, "~.")),
                           target = target_t,
                           test_data = test_data,
                           type = featSelTypes[k],
                           part = 0.5,
                           trees_num=500
  )

  formula <- c(formula,as.vector(res$attr_part[1]))

  # create and cross-validate rpart model based on particular feature sel. method
  crossval.rpart <- model_eval(test_data = test_data,
                               fun = rpart,
                               formula = formula[k],
                               crossval_number = 10,
                               args = list(method="anova")
  )
  corMeasures <- c(corMeasures,crossval.rpart$COR)
}

# with all attributes
crossval.rpart <- model_eval(test_data = test_data,
                             fun = rpart,
```

```
                              formula = as.formula(paste(target_t, "~.")),
                              args = list(method ="anova"),
                              crossval_number = 10
)
corMeasures <- c(corMeasures,crossval.rpart$COR)

#selection of best formula
k <- which.max(corMeasures)
formula <- formula[k]

# EVALUATION PROCEDURES GOES HERE

crossval.lm <- model_eval(test_data = test_data,
                              fun = lm,
                              formula = formula,
                              crossval_number = 10
)
crossval.rpart <- model_eval(test_data = test_data,
                             fun = rpart,
                             formula = formula,
                             crossval_number = 10
                             )

crossval.bagging <- model_eval(test_data = test_data,
                              fun = bagging,
                              formula = formula,
                              crossval_number = 10
)
plr_args <- list(minsplit=2, maxdepth=8)

crossval.plr <- model_eval(test_data=test_data,
                              fun = grow.modtree,
                              formula = formula,
                              crossval_number = 10,
                              args = plr_args
                              )

ctrl <- trainControl(method = "cv", number = 10)
args_t <- c(method="treebag",trControl=ctrl)

crossval.caret <- model_eval(test_data = test_data,
                              fun = train,
                              formula = formula,
                              crossval_number = 10,
                              args=args_t
)

crossval.all <- rbind(crossval.lm,crossval.rpart,crossval.bagging,crossval.plr,crossval.ca
```

Plik: data_org.R

```r
#data_org.R
# data collecting and organization

# training_data <- read.csv("training.csv")

#' Title
#'
#' @param data data to be organized
#' @param target "Appliances" or "lights"
#' @param testDataLength length of output subset (from 1 do testDataLength)
#'
#' @return organised data set (or subset, if testDataLength specified),
#'
dataOrganization <- function(data,target,testDataLength=nrow(data)){

  #day_mon = day + 30 * month
  #min_hour = minutes + 60 * hours
  if (!is.data.frame(data)){
    message("data is no data.frame type")
    stop()
  }
  if ((target!="Appliances")&(target!="lights"))
  {
    message("Target is not \"Appliances\" neither \"lights\" ")
    stop()
  }
  if (testDataLength<2|testDataLength>nrow(data)){
    message("testDataLength must be between 2 and length(data)")
    stop()
  }


  if (target == "Appliances"){
    out_data <- data.frame(
      Appliances = data$Appliances,
      month = as.numeric(substring(data$date,6,7)),
      day = as.numeric(substring(data$date,9,10)),
      hours = as.numeric(substring(data$date,12,13)),
      minutes = as.numeric(substring(data$date,15,16)),
      # day_mon = day_mon,
      # min_hour = min_hour,
      T1 = data$T1,
      T2 = data$T2,
      T3 = data$T3,
      T4 = data$T4,
      T5 = data$T5,
      T6 = data$T6,
      T7 = data$T7,
      T8 = data$T8,
      T9 = data$T9,
      RH_1 = data$RH_1,
      RH_2 = data$RH_2,
      RH_3 = data$RH_3,
      RH_4 = data$RH_4,
      RH_5 = data$RH_5,
      RH_6 = data$RH_6,
      RH_7 = data$RH_7,
      RH_8 = data$RH_8,
      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 }
```

Plik: feature_selection.R

```r
# feature_selection.R

library(randomForest)
library(rpart)
library(rpart.plot)  # plotting regression trees
library(Metrics)     # RMSE
library(dmr.disc)
library(dmr.stats)
library(dmr.attrsel)

#' @title Feature Selection
#'
#' @param test_data data to perform feature ranking
#' @param formula_full  full formula, with targets and all attributes
#' @param target target (character type)
#' @param type  "rf" - randomForest IMPORTANCE-based ranking (default),
#' "relief" - RELIEF algorithm
#' "simple" - simple filter algorithm
#' @param part part of attributes returned by feature selection, 0 < part <=1 (default=0.5
#' @param trees_num numbers of trees (randomForest) or bootstrap sets (bootstrap)
#'
#' @return formula with selected attributes
#'
#'
#'
feature_selection <- function(formula_full,target,test_data,type="rf",part=1,trees_num=10

  if ( !is.data.frame(test_data) ){
    message("Pass data frame format")
    stop()
  }
  if (trees_num<1) {
    message("Trees number should be >= 1")
    stop()
  }
  if (part>1|part<=0){
    message("Parts of atrributes must be >0 and <=1")
    stop()
  }

  print("FEATURE SELECTION")
  count <- ceiling((ncol(test_data)-1)*part)

  if (type=="rf"){
    full_model_RF <- randomForest(formula = formula_full,
                          data = test_data,
                          importance = TRUE,
                          ntree=trees_num)
    importance_RF <- data.frame(importance(full_model_RF, type=1), "k"=1:(ncol(test_data)-
    res <- data.frame("importance"=importance_RF[order(importance_RF[,1],decreasing = TRUE
                              "k"=importance_RF$k[order(importance_RF[,1],decreasing = TRU
                              "attr"=rownames(importance_RF)[order(importance_RF[,1], decr

    # plotting most important parameters
    barplot(res$importance[1:count],main="Most important variables - randomForest",names.a

    # varImpPlot(x=full_model_RF,
    #          n.var=count,
    #          type=1,
```

```
    #          main="Most important variables - randomForest importance")

    # passing most important attributes from feature selection
    count <- ceiling((ncol(test_data)-1)*part)
    attr_part <- paste(res$attr[2:count], collapse = "+")
    attr_part <- paste(target, "~", attr_part)
    out <- data.frame(attr_part,res)
    return(out)
  }
  else if(type=="relief"){

    res <- rrelief.filter(formula = formula_full,
                          data = test_data,
                          k=3,
                          K=20)

    names_t <- names(res)

    # plotting most important parameters
    barplot(as.vector(res)[1:count],main="Most important variables - relief",names.arg = na

    attr_part <- paste(names_t[1:count], collapse = "+")
    attr_part <- paste(target, "~", attr_part)
    out <- data.frame(attr_part,res)
    return(out)
  }

  else if (type=="simple"){

    res <- simple.filter(formula = formula_full,
                          data = discnm.eqfreq(~.,test_data,10),
                          dd=symunc)

    names_t <- names(res)

    # plotting most important parameters
    barplot(as.vector(res)[1:count],main="Most important variables - simple",names.arg = na

    attr_part <- paste(names_t[1:count], collapse = "+")
    attr_part <- paste(target, "~", attr_part)
    out <- data.frame(attr_part,res)
    return(out)
  }

  # else if (type=="wrapper"){
  #
  #   res <- wrapper.filter.select(formula = formula_full,
  #                        data = test_data,
  #                        utils = rpart(formula_full,test_data),
  #                        alg = rpart,
  #                        args = list(minsplit=2),
  #                        initf = asel.init.none,
  #                        nextf = asel.next.forward
  #   )
  #
  #
  #   names_t <- res$subset
  #
  #   # plotting most important parameters
  #   barplot(res$eval[1:count],main="Most important variables - wrapper",names.arg = name
  #
```

```
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: simple.filter.R

```
1
2  library(dmr.stats)
3  library(dmr.util)
4
5  dd.chi2 <- function(a1, a2) 1-chisq.test(a1, a2)$p.value
6  cd.kruskal <- function(a1, a2) 1-kruskal.test(a1, a2)$p.value
7  cc.spearman <- function(a1, a2) 1-cor.test(a1, a2, method="spearman")$p.value
8
9  #' @title Simple attribute filter
10 #'
11 #' @param formula
12 #' @param data data to perform feature selection
13 #' @param dd test to perform selection with discrete attributes and discrete target
14 #' @param cd test to perform selection with continuous attributes and discrete target
15 #' @param cc test to perform selection with continuous attributes and discrete target
16 #'
17 #' @return named by attribute list of test measures
18 #'
19 simple.filter <- function(formula, data, dd=dd.chi2, cd=cd.kruskal, cc=cc.spearman)
20 {
21   attributes <- x.vars(formula, data)
22   target <- y.var(formula)
23   utility <- function(a)
24   {
25     unname(switch(attr.type(data[[a]], data[[target]]),
26                   dd = dd(data[[a]], data[[target]]),
27                   cd = cd(data[[a]], data[[target]]),
28                   dc = cd(data[[target]], data[[a]]),
29                   cc = cc(data[[a]], data[[target]])))
30   }
31   sort(sapply(attributes, utility), decreasing=TRUE)
32 }
```

Plik: model_eval.R

```r
# model evaluation

library(dmr.claseval)

#' @title Model Evaluation
#'
#' @param test_data - data to perform cross-validation
#' @param fun - regression algorithm (e.g. train,bagging,lm) default=rpart
#' @param formula - regression formula
#' @param crossval_number - number of cross-validations, default=10
#' @param args - passed arguments into regression algorithm, default=NULL
#'
#' @return data frame with measures in following order: MSE, RRSE, MAE, RMSE, RAE, COR, RS
#'
#'
model_eval <- function(test_data,fun=rpart,formula,crossval_number=10,args=NULL){

  if ( !is.data.frame(test_data) ){
    message("Pass data frame format")
    stop()
  }
  if ( !is.character(formula)) {
    message("Pass string into attr parameter")
    stop()
  }
  if (crossval_number<1){
    message("Cross validations number must be greater than 0")
    stop()
  }
  if (crossval_number==1){
    warning("No cross-validation")
  }
  temp_model <- crossval(fun,
                         as.formula(formula),
                         test_data,
                         k=crossval_number,
                         args = args)
  MSE <- mse(temp_model$pred, temp_model$true)
  RRSE <- rrse(temp_model$pred, temp_model$true)
  MAE <- mae(temp_model$pred, temp_model$true)
  RMSE <- rmse(temp_model$pred, temp_model$true)
  RAE <- rae(temp_model$pred, temp_model$true)
  COR <- cor(temp_model$pred, temp_model$true, method="pearson")
  RSE <- rse(temp_model$pred, temp_model$true)

  out <- data.frame(MSE,RRSE,MAE,RMSE,RAE,COR,RSE)
  return(out)
}
```