Projekt2

Bogna Pawlus

2022-06-07

1. Opis

Dane X_train zawierają obserwacje 9000 genów (kolumny) w każdej z 3794 komórek (wiersze) – zmienne objaśniające

```
head(X_train[,1:8], n = 6L)
```

```
##   LINC01128     NOC2L HES4     ISG15 AGRN C1orf159 TNFRSF18 TNFRSF4
## 1         0 0.0000000    0 0.9530905    0        0        0       0
## 2         0 0.0000000    0 0.0000000    0        0        0       0
## 3         0 1.1452906    0 0.0000000    0        0        0       0
## 4         0 0.0000000    0 0.0000000    0        0        0       0
## 5         0 0.5644676    0 0.0000000    0        0        0       0
## 6         0 0.0000000    0 0.8929155    0        0        0       0
```

Dane y_train zawierają obserwację zmiennej objaśnianej – dla każdej komórki ilość białka

```
head(y_train, n = 6L)
```

```
##        CD36
## 1 0.0000000
## 2 2.1832790
## 3 0.0000000
## 4 0.2991487
## 5 0.0000000
## 6 0.0000000
```

Dane x_test zawierają obserwacje 9000 genów dla 670 komórek. Przewidzimy dla tych komórek ilość białka

```
head(X_test[,1:8], n = 6L)
```

```
##   LINC01128     NOC2L HES4     ISG15 AGRN C1orf159 TNFRSF18 TNFRSF4
## 1         0 0.0000000    0 0.8214592    0        0        0       0
## 2         0 0.0000000    0 0.0000000    0        0        0       0
## 3         0 0.6346151    0 0.0000000    0        0        0       0
## 4         0 0.0000000    0 0.0000000    0        0        0       0
## 5         0 0.0000000    0 0.0000000    0        0        0       0
## 6         0 0.0000000    0 0.2933263    0        0        0       0
```

Sprawdźmy brakujące dane

```
sum(is.na(X_train))
```

```
## [1] 0
```

```
sum(is.na(X_test))
```

```
## [1] 0
```

```
sum(is.na(y_train))
```

```
## [1] 0
```

To oznacza, że nie ma braków w danych.

Sprawdzimy jakiego typu są kolumny

```
SX = sapply(X_train, class)
SX[SX!="numeric"]
```

```
## named character(0)
```

```
SX2 = sapply(X_test, class)
SX2[SX2!="numeric"]
```

```
## named character(0)
```

```
class(y_train[,1])
```

```
## [1] "numeric"
```

To oznacza, że wszystkie kolumny są liczbowe, zatem nie dokonujemy żadnej konwersji danych.

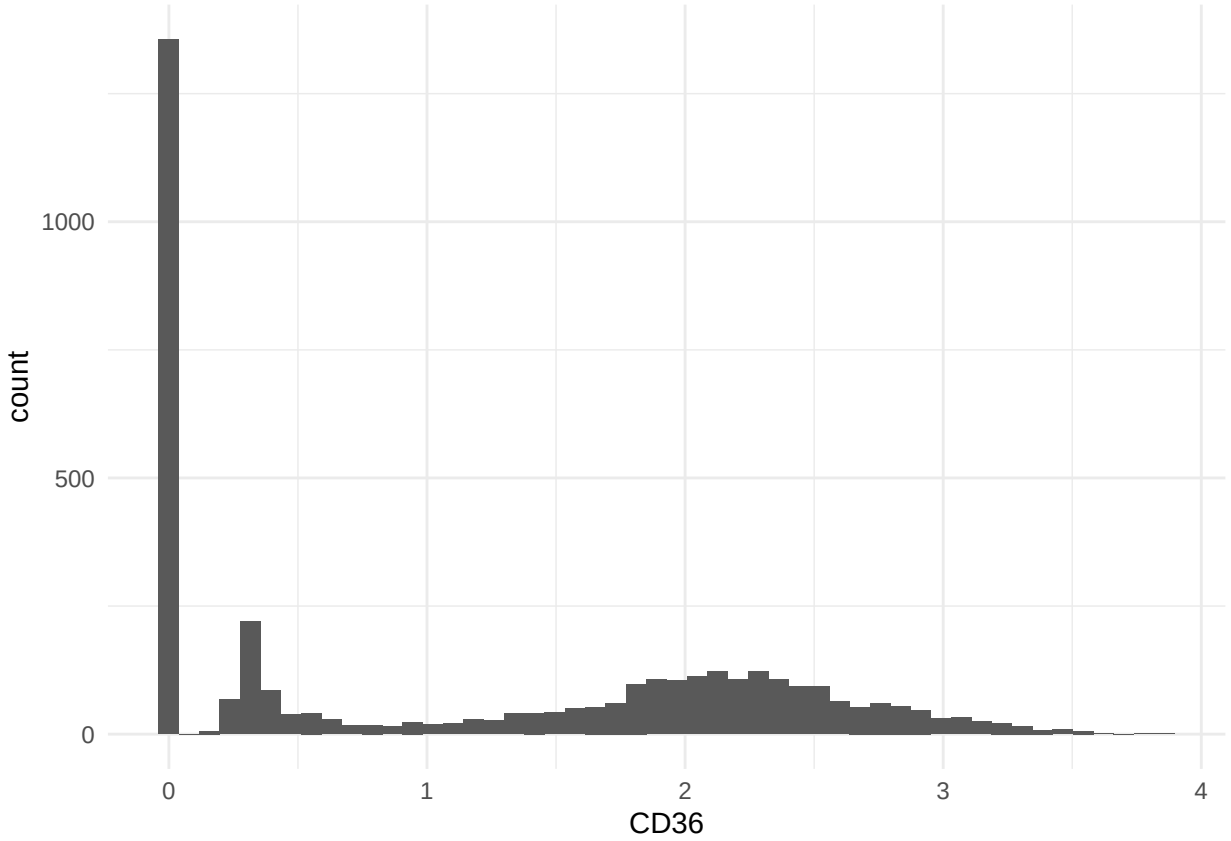
Podastwowe statystyki dla y_train:

```
summary(y_train)
```

```
##       CD36
##  Min.   :0.0000
##  1st Qu.:0.0000
##  Median :0.9147
##  Mean   :1.1454
##  3rd Qu.:2.1666
##  Max.   :3.8572
```

Histogram dla zmiennej objaśnianej y_train

```
ggplot(y_train) + geom_histogram(aes(x=CD36), bins=50) +theme_minimal()
```



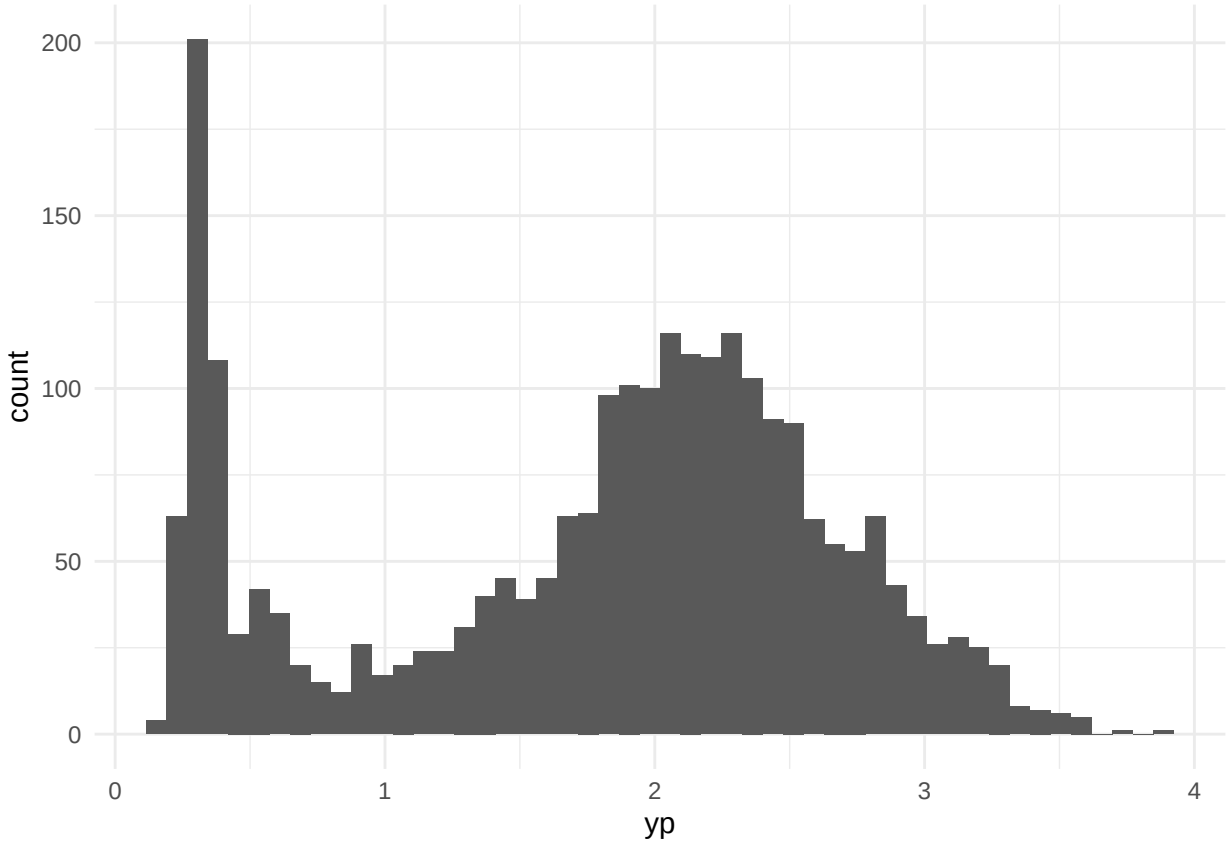

```
sum(y_train$CD36 == 0)
```

```
## [1] 1356
```

Dużo komórek ma zerową ilość białka, dlatego rozważmy też histogram zmiennej objaśnianej y_train dla niezerowych ilości białka, z którego widać, że wartość zmiennej najczęściej pojawia się wokół mniej więcej 0.3 i 2.3., natomiast w okolicach średniej 1.1454 znajduje się mała liczba danych.

```
yp = y_train[y_train$CD36>0,]
yp = data.frame(yp)

ggplot(yp) + geom_histogram(aes(x=yp), bins=50) +theme_minimal()
```



Policzymy korelacje wektora y_train z każdą z 9000 zmiennych objaśniających

```
cor_ytrain = apply(X_train, 2, function(x) cor(x, y_train))
```

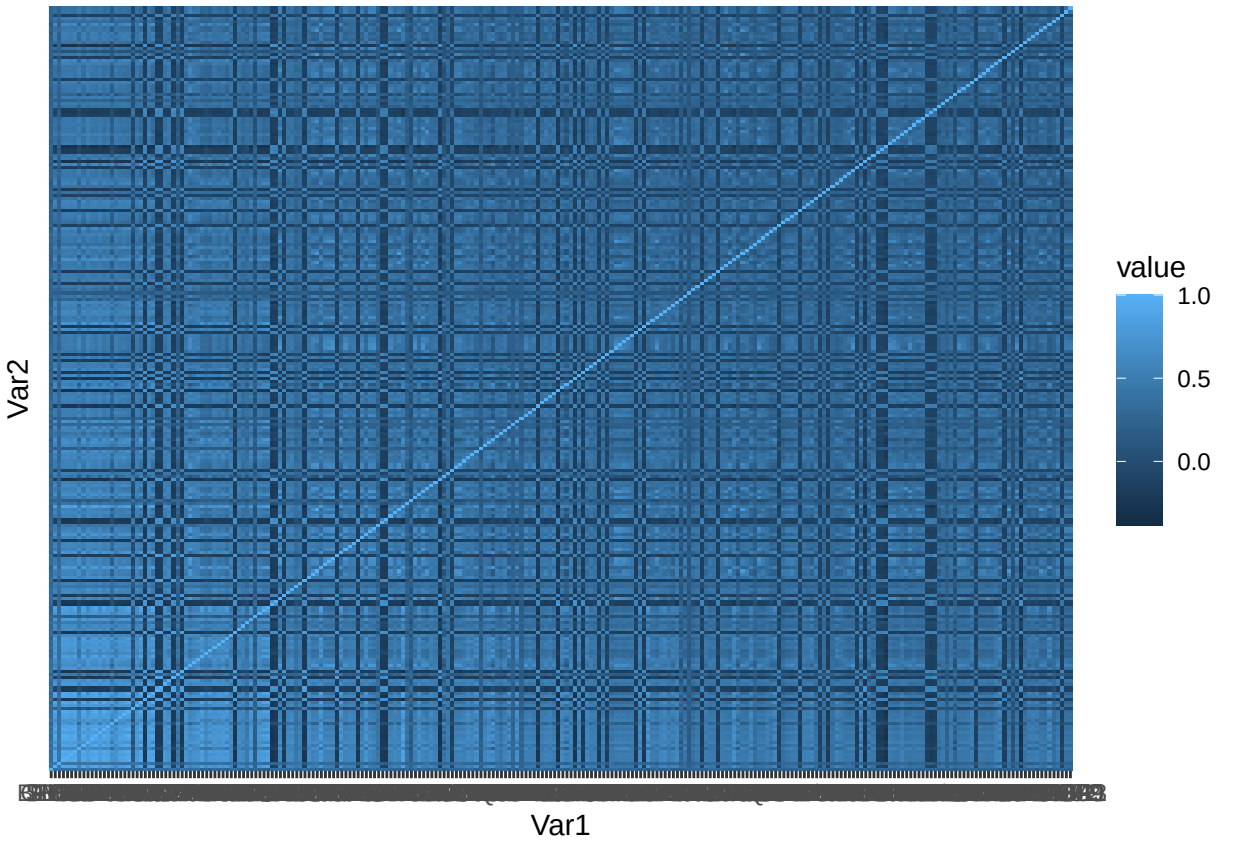
Mapa ciepła

Wybieramy 250 zmiennych objaśniających najbardziej skolerowaych z ilością białka

```
cor_ytrain = apply(X_train, 2, function(x) cor(x, y_train)) #wektor korelacji białka z każdym genem
nrows_maxcor = order(cor_ytrain, decreasing = "TRUE")[1:250] #współrzędne genów z największych korelacj
X_train_250 = X_train[nrows_maxcor] #kolumny genów z największą korelacją
```

Mapa ciepła, posortowana względem zmiennych najbardziej skolerowanych ze zmienną objaśniającą. (wnioskujemy niej, że jeśli koleracja zmiennych a, b ze zmienną objaśniającą jest duza, to średnio również cor(a, b) będzie duża)

```
matx2 = round(cor(X_train_250), 2)
meltmatx = melt(matx2)
ggplot(data = meltmatx, aes(x=Var1, y=Var2, fill=value)) + geom_tile() + theme(axis.text.y=element_blank
```



Źródło:

http://www.sthda.com/english/wiki/ggplot2-quick-correlation-matrix-heatmap-r-software-and-data-visualization

2.0 Walidacja krzyżowa

Ze względu na długi czas tworzenia modelów, w walidacji krzyżowej utworzymy jedynie 5 podzbiorów. Tworzymy wektor 'podział', zawierający 3794 liczby całkowite od 1 do 5, przy czym każda występuje podobną liczbę razy

```
podzial <- cut(1:nrow(X_train), 5, labels = F)
podzial <- sample(podzial)
podzial[1:40]
```

```
##  [1] 1 5 2 4 3 4 4 1 4 5 2 4 5 3 2 1 4 2 1 5 5 5 1 3 5 4 5 3 5 2 1 1 4 4 5 1 4 5
## [39] 1 1
```

$i-$ta współrzędna w wektorze 'podzial' odpowiada przynależności $i-$tej obserwacji ze X_train do odpowiedniego podzbioru wykonanej walidacji. Kolejne modele będziemy budować korzystając z tego podziału.

2. ElasticNet

W tym modelu próbujemy znaleść liniową zależność wektora zmiennej objaśnianej $y \in \mathbb{R}^n$ od macierzy $x \in \mathbb{R}^{n \times m}$ zmiennych objaśniających, tzn chcemy, żeby zachodziło

$$y \approx \beta x$$

Dopasowując współczynniki β minimalizujemy RSE + pewien błąd karny, dokładniej minimalizujemy

$$\sum_{i=1}^{n}\left(y_i - \sum_{j=0}^{p}\beta_j x_{ij}\right)^2 + (1-\alpha)\lambda\sum_{j=1}^{p}\beta_j^2 + \alpha\lambda\sum_{j=1}^{p}|\beta_j|$$

dla pewnych hiperparametrów α, λ. Dla $\alpha = 1$ otrzymujemy regresję lasso, dla $\alpha = 0$ regresję grzbietową.

Źródło:

https://daviddalpiaz.github.io/r4sl/elastic-net.html

https://rpubs.com/jmkelly91/881590

Zbudujemy modele dla $\alpha \in \{0, 0.5, 1\}$ i dla $\lambda \in \{1, 10, 100\}$. Będziemy korzystać ze zbiorów wybranych już wcześniej do walidacji krzyżowej w punkcie 2.0. Utworzymy macierz matx zawierającą błędy (testowe) modelu dla ustalonego alfa(wiersze) i lambda(kolumny). Dla każdej ustalonej pary (alfa, lambda) błąd testowy umieszczony w macierzy matx będzie średnią błędów średniokwadratowych z walidacji krzyżowej.

```
alfa = c(0, 0.5, 1)
lamb = c(1, 10, 100)

matx = matrix(rep(0, 9), nrow = 3) #macierz błędów
blad = rep(0, 5) #błędy dla każdego podzbioru cv, ustalone w, k

for(w in 1:3){
  for(k in 1:3){
    for(i in 1:5){#dla ustalonych w, k sprawdź wszystkie podzbiory z cv
    treningowe <- which(podzial != i) #indeksy obserwacji treningowych

      m0 = glmnet(X_train[treningowe,], y_train[treningowe,], alpha=alfa[w], lambda=lamb[k])
      przewid <- predict(m0, as.matrix(X_train[-treningowe,]))
      ytes = y_train[-treningowe, ]
      blad[i] = mean((przewid-ytes)^2) #mse dla podziału i
    }

    matx[w, k] = mean(blad) #ustalone w, k -- uśrednienie błędu dla podziałów
  }
}
```

Stąd otrzymujemy macierz błędów matx

```
matx
```

```
##           [,1]     [,2]      [,3]
## [1,] 0.1404644 0.137981 0.3306945
## [2,] 0.7811647 1.224191 1.2241911
## [3,] 1.2241911 1.224191 1.2241911
```

Widzimy, że najmniejszy błąd występuje w matx[1, 2], co oznacza, że najmniejszy błąd dostaliśmy dla alfa[1]=0 i lambda[2]=10. Widzimy, że przy alfa=0, niezależnie od lambda wystąpiły ogólnie mniejsze błędy, dlatego możemy spróbować poprawić siatkę wartości alf do wartości bliższych 0, np. $\alpha \in \{0.005, 0.01, 0.015\}$ (nie zmieniamy lambd) i wykonać ponownie ten sam kod, ze zmienioną siatką

Otrzymujemy macierz

```
matx
```

```
##           [,1]      [,2]     [,3]
```

```
## [1,] 0.1200669 0.2098575 1.135310
## [2,] 0.1196649 0.2877835 1.226196
## [3,] 0.1227764 0.3698298 1.226196
```

przy czym matx[2, 1] jest najmniejsza, czyli bardziej opłaca się wziąć alfa=0.01, lambda=1.

Możemy zauważyć jeszcze, że minimalne pomniejszenie alfa do 0.09 i lambda do 0.9 trochę pomniejszy błąd testowy. Policzmy jeszcze uśredniony błąd treningowy:

```
blad = rep(0, 5) #wektor błędów testowych w na odpowiednich podzbiorach z cv
bladb = rep(0, 5) #wektor błędów treningowych

for(i in 1:5){#dla ustalonych w, k sprawdź wszystkie podzbiory z cv
  treningowe <- which(podzial != i) #indeksy obserwacji treningowych

  m0 = glmnet(X_train[treningowe,], y_train[treningowe,], alpha=0.009, lambda=0.9)
  wsp = predict(m0, type="coef") #współczynniki w wektorze

  #błędy testowe
  vect = rep(1, nrow(X_train[-treningowe,])) #wektor z jedynkami
  przewid <- predict(m0, X_train[-treningowe,])
  ytes = y_train[-treningowe, ]
  blad[i] = mean((przewid-ytes)^2) #mse dla podziału i

  #błędy treningowe
  vectb = rep(1, nrow(X_train[treningowe,])) #wektor z jedynkami
  przewidb <- predict(m0, X_train[treningowe,])
  ytesb = y_train[treningowe, ]
  bladb[i] = mean((przewidb-ytesb)^2) #mse dla podziału i
}

#blad testowy
mean(blad)
```

```
## [1] 0.118934
```

```
#blad treningowy
mean(bladb)
```

```
## [1] 0.07183734
```

Lasy losowe

Korzystając z walidacji krzyżowej z punktu 2.0 przeprowadzimy algorytm lasów losowych. Ponieważ zwiększanie liczby drzew i zwiększanie (do pewnego momentu) liczby rozważanych cech w drzewach zwiększa dokładność modelu, przyjmiemy hiperparametr ntree = 20 i mtree = 5 (tylko tyle ze względu na długi czas liczenia), a walidacją krzyżową spróbujemy dobrać wartości parametrom: maxnodes ze zbioru {10, 50, 100}, sampsize, nodesize ze zbioru {5, 10, 20}. Dla każdej trójki (ii, jj, kk) błąd treningowy zapiszemy w trójwymiarowej macierzy 'bledy'.

```
library(randomForest)
ii = c(10, 50, 100) #w1
jj = c(5, 10, 20) #w2
kk = c(5, 10, 20) #w3
bladrf = rep(0, 5) #błędy w każdym podzbiorze cv
bledy = array(rep(1, 3*3*3), dim=c(3, 3, 3)) #macierz bledów dla kazdych ii, jj, kk
```

```
for(lii in 1:3){
  for(ljj in 1:3){
    for(lkk in 1:3){
      for(i in 1:5){
        treningowe <- which(podzial != i) #indeksy obserwacji treningowych

        clas = randomForest(x = X_train[treningowe,], y=y_train[treningowe,],
        importance = TRUE, mtry = 5, ntree = 20, maxnodes = ii[lii], nodesize = jj[ljj],
        stratasize=kk[lkk])
        y_pdk = predict(clas, newdata = X_train[-treningowe,]) #predykcja
        ytes = y_train[-treningowe, ] #bledy
        bladrf[i] = mean((y_pdk - ytes)^2)

      }
      bledy[lii, ljj, lkk] = mean(bladrf)
    }
  }
}
bledy
```

```
## , , 1
##
##           [,1]      [,2]      [,3]
## [1,] 0.7422433 0.7463069 0.7581396
## [2,] 0.5202477 0.4900391 0.5198943
## [3,] 0.4231806 0.4191353 0.4136565
##
## , , 2
##
##           [,1]      [,2]      [,3]
## [1,] 0.7486200 0.7502736 0.7964287
## [2,] 0.5381165 0.5000198 0.4930078
## [3,] 0.4419380 0.4164275 0.3692845
##
## , , 3
##
##           [,1]      [,2]      [,3]
## [1,] 0.7467344 0.7724126 0.7478780
## [2,] 0.5260512 0.5070175 0.4778665
## [3,] 0.4288089 0.4302310 0.3952944
```

Z powyższej macierzy widzimy, że najmniejsza współrzędna wystąpiła w [3,3,2], zatem tworzymy model z parametrami maxnodes=100, nodesize=20, stratasize=10,. Dla dokładności modelu zwiększymy jeszcze liczbę drzew i liczbę powtórzeń. Policzymy jeszcze błąd treningowy

```
bladrftr = rep(0, 5) #błędy walidacyjne
for(i in 1:5){
  treningowe <- which(podzial != i) #indeksy obserwacji treningowych

  clas = randomForest(x = X_train[treningowe,], y=y_train[treningowe,],
  importance = TRUE, mtry = 20, ntree = 100, maxnodes = 100,
  nodesize = 20, stratasize=10)

  #testowe
  y_pdk = predict(clas, newdata = X_train[-treningowe,]) #predykcja
```

```
  ytes = y_train[-treningowe, ] #bledy
  bladrf[i] = mean((y_pdk - ytes)^2) #uśredniony błąd testowy

  #treningowy
  y_pdktr = predict(clas, newdata = X_train[treningowe,]) #predykcja
  ytestr = y_train[treningowe, ] #bledy
  bladrftr[i] = mean((y_pdktr - ytestr)^2) #uśredniony błąd testowy

}
#błąd testowy modelu
mean(bladrf)
```

```
## [1] 0.1941342
```

```
#błąd treningowy modelu
mean(bladrftr)
```

```
## [1] 0.129573
```

Podstawowy model referencyjny

Wykonamy model ze średnią arytmetyczną zmiennej objaśnianej, korzystając ze wcześniejszej walidacji krzyżowej.

```
bladp = rep(0, 5) #błędy testowe
bladptr = rep(0, 5) #błędy treningowe
for(i in 1:5){
  treningowe <- which(podzial != i) #indeksy obserwacji treningowych

  predyk = mean(y_train[treningowe, ])

  #testowe
  y_pdkp = rep(predyk, nrow(as.matrix(y_train[-treningowe, ]))) #predykcja
  ytesp = y_train[-treningowe, ] #bledy
  bladp[i] = mean((y_pdkp - ytesp)^2) #uśredniony błąd testowy

  #treningowy
  y_pdkptr = rep(predyk, nrow(as.matrix(y_train[treningowe, ]))) #predykcja
  ytestptr = y_train[treningowe, ] #bledy
  bladptr[i] = mean((y_pdkptr - ytestptr)^2) #uśredniony błąd treningowy

}
#błąd testowy modelu
mean(bladp)
```

```
## [1] 1.226196
```

```
#błąd treningowy modelu
mean(bladptr)
```

```
## [1] 1.223805
```

Podsumowanie modeli

Zrobimy tabelę 'podsum' z błędami. Widzimy z niej, że przy ograniczeniach czasu i powyższym doborze parametrów, najlepszy błąd średniokwadratowy otrzymujemy z modelu Elastic Net, gorszy z RandomForest, jeszcze gorszy z podstawowego modelu.

```r
podsum = matrix(rep(0, 6), nrow = 2)
rownames(podsum) = c("b. testowy", "b. treningowy")
colnames(podsum) = c("eNet", "randomForest", "basic")
podsum[1,1] = mean(blad)
podsum[2,1] = mean(bladb)

podsum[1, 2] = mean(bladrf)
podsum[2, 2] = mean(bladrftr)

podsum[1, 3] = mean(bladp)
podsum[2, 3] = mean(bladptr)

podsum
```

```
##                      eNet randomForest    basic
## b. testowy    0.11893402    0.1941342 1.226196
## b. treningowy 0.07183734    0.1295730 1.223805
```

Ponieważ najlepszy okazuje się model Elastic Net, w ostatnim punkcie będziemy poprawiać ten model.

Dowolny model

Ponieważ usuwanie zmiennych o wysokiej korelacji może poprawiać jakoś modelu, rozważymy macierz korelacji, z której wybierzemy zmienne z silną korelacją

```r
#cormatx = cor(X_train)
#highlyCorrelated <- findCorrelation(cormatx, cutoff=0.95)
#print(highlyCorrelated)
#nie wykonamy kodu ze względu na długość działania, ale wiersze do odrzucenia to 4682 i 6599
X_train = X_train[,-c(4682,6599)]
X_test = X_test[,-c(4682,6599)]
```

Na tak zmodyfikowanych danych przeprowadzimy poprzedni model Elastic Net (analogicznie jak wcześniej dla zwykłego modelu Elastic Net, możemy spróbować różne siatki hiperparametrów, ale próbując siatkę alfa = c(0.007, 0.009, 0.01, 0.011), lamb = c(0.9, 1, 1.1, 1.5), dostaniemy, że najmniejszy błąd testowy w takiej siatce również tutaj będzie dla alfa=0.009, lambda=0.9, skąd decyzja o pozostaniu przy takich parametrach)

```r
for(i in 1:5){#dla ustalonych w, k sprawdź wszystkie podzbiory z cv
  treningowe <- which(podzial != i) #indeksy obserwacji treningowych

  m0 = glmnet(X_train[treningowe,], y_train[treningowe,], alpha=0.009, lambda=0.9)
  wsp = predict(m0, type="coef") #współczynniki w wektorze

  #błędy testowe
  vect = rep(1, nrow(X_train[-treningowe,])) #wektor z jedynkami
  przewid <- predict(m0, X_train[-treningowe,])
  ytes = y_train[-treningowe, ]
  blad[i] = mean((przewid-ytes)^2) #mse dla podziału i

  #błędy treningowe
  vectb = rep(1, nrow(X_train[treningowe,])) #wektor z jedynkami
  przewidb <- predict(m0, X_train[treningowe,])
  ytesb = y_train[treningowe, ]
  bladb[i] = mean((przewidb-ytesb)^2) #mse dla podziału i
}
```

```
#błąd testowy
mean(blad)
```

```
## [1] 0.1192544
```

```
#błąd treningowy
mean(bladb)
```

```
## [1] 0.07191825
```

Wartość zmiennej 'blad' oznaczająca uśredniony błąd średniokwadratowy modelu jest mniejsza, niż w poprzednich modelach.

Oszacowania dla obserwacji X_test:

```
#m1 = glmnet(X_train[,], y_train[,], alpha=0.009, lambda=0.9)
#wynik = predict(m1, as.matrix(X_test))
#wynik = data.frame(0:669, wynik)
#names(wynik) = c("ID", "Expected")
```