Modele regresji i ich zastosowania - raport 3.

Patryk Krukowski(249824)

5 maja 2021

Spis treści

1 Wstęp 1

2 Zadanie 1. 1

3 Zadanie 2. 2

4 Zadanie 3. 3

5 Zadanie 4. 5

6 Zadanie 5. 6

7 Zadanie 6. 7

8 Zadanie 7. 8

9 Zadanie 8. 9

1 Wstęp

W raporcie tym zajmiemy się zagadnieniem dotyczącym regresji nieliniowej, w szczególności zaimplementujemy i użyjemy algorytmu Gaussa-Newtona do estymacji współczynników regresji oraz sprawdzimy działanie tej metody. Na początek załóżmy, że:

- $n = 10000$ (rozmiar próby)
- $\boldsymbol{x} = (0.1, 0.2, 0.3, \ldots, 1000)$
- $\beta_1 = 80$
- $\beta_2 = 100$
- $\beta_3 = 0.005$
- $\sigma^2 = 0.5$

Przystąpmy teraz do wykonania zadań laboratoryjnych.

2 Zadanie 1.

Generujemy n obserwacji $Y_1, \ldots, Y_n$, takich że

$$Y_i = g\left(x_i, \boldsymbol{\beta}\right) + \epsilon_i,$$

gdzie $\epsilon_1, \ldots, \epsilon_n$ i.i.d. $\mathcal{N}\left(0, \sigma^2\right)$ oraz $\boldsymbol{\beta} = (\beta_1, \beta_2, \beta_3)$.

```
##
## Attaching package: 'calculus'
## The following objects are masked from 'package:pracma':
##
##     cross, gradient, hessian, integral, jacobian, laplacian, taylor
```

```
x <- seq(0.1, 1000, by=0.1)
n <- length(x)
beta_1 <- 80
beta_2 <- 100
beta_3 <- 0.005
war <- 0.5


eps <- rnorm(n, 0, sqrt(war))
y <- as.numeric(c(rep(0, n))) #inicjalizacja zerami
for (i in 1:n) {
  y[i] <- beta_1+beta_2*exp(-beta_3*x[i])+eps[i]
}
```

3 Zadanie 2.

Niech teraz $\beta_1, \beta_2, \beta_3$ oznaczają zmienne zamiast konkretnych wartości. Podajemy odpowiednie wzory:

- $$g\left(\boldsymbol{x}, \boldsymbol{\beta}\right) = \begin{pmatrix} g\left(x_1, \boldsymbol{\beta}\right) \\ \ldots \\ g\left(x_{10000}, \boldsymbol{\beta}\right) \end{pmatrix},$$

  gdzie $g\left(x_i, \boldsymbol{\beta}\right) = \beta_1 + \beta_2 e^{-\beta_3 x_i}$ oraz, przy takim podziale, $x_i = \frac{i}{10}$.

- Ustalmy indeks $i \in \{1, 2, \ldots, 10000\}$. Dalej mamy

  $$\nabla g\left(x_i, \boldsymbol{\beta}\right) = \left(1, e^{-\beta_3 x_i}, -\beta_2 x_i e^{-\beta_3 x_i}\right)$$

  Implementujemy wspomniane wyżej funkcje w R.

```r
g_funkcja <- function(beta, x) {
  if (length(beta) > 3) {
    print('Z■ly wymiar bety!')
  }
  g <- rep(0,length(x))
  for (i in 1:length(x)) {
    g[i] = beta[1]+beta[2]*exp(-beta[3]*x[i])
  }
  return(g)
}

g_gradient <- function(i, beta, x) {
  beta <- as.vector(beta)
  if (length(beta) > 3) {
    print('Z■ly wymiar bety!')
  }
  g <- rep(0, length(x))
  g <- c(1, exp(-beta[3]*x[i]), -x[i]*beta[2]*exp(-beta[3]*x[i]))
  return(g)
}
```

4 Zadanie 3.

Implementujemy algorytm Gaussa-Newtona, gdzie punkt startowy $\left(\beta_1^0, \beta_2^0, \beta_3^0\right) = (79, 101, 0.004)$ oraz funkcję odpowiedzialną za estymator wariancji, tj.

$$\hat{\sigma}^2 = \frac{\sum_{i=1}^{n} \left(y_i - g\left(\boldsymbol{x}_i, \hat{\boldsymbol{\beta}}\right)\right)^2}{n-3},$$

i wyznaczymy estymatory $\hat{\beta}_1, \hat{\beta}_2, \hat{\beta}_3, \hat{\sigma}^2$. Ponadto ustawiamy dokładność metody na 0.0001, tzn. kryterium stopu to

$$\left\| \hat{\boldsymbol{\beta}}^{(s+1)} - \hat{\boldsymbol{\beta}}^{(s)} \right\|_2 \leq 0.0001,$$

gdzie s to liczba kroków metody. Niech maksymalna liczba kroków to 100. Parametrami algorytmu jest wektor $\boldsymbol{x}$ oraz punkt startowy algorytmu $\left(\beta_1^0, \beta_2^0, \beta_3^0\right)$.

W trakcie wykonywania algorytmu może się zdarzyć sytuacja, w której macierz $\boldsymbol{G}^T\boldsymbol{G}$ (oznaczenia zachowane z wykładu) jest **nieodwracalna**. Wówczas zabezpieczamy się poniższym warunkiem.

```r
if (isTRUE(det(t(G_macierz(beta_0, x)) %*% G_macierz(beta_0, x)) > 0) == FALSE ) {
      print('Algorytm rozbie■ny!')
      break
    }
```

Gdyby okazał się on spełniony, to oznaczać to będzie, że algorytm jest **rozbieżny**. Wtedy przerywamy procedurę i printujemy ostatnią wartość $\hat{\boldsymbol{\beta}}$, którą mogliśmy wyliczyć.

```r
gauss_newton <- function(x, beta_0) {
  max_krok <- 100 #maksymalna liczba iteracji
  n <- length(x) #rozmiar próby
  beta_0 <- matrix(beta_0, nrow=3, ncol=1)
  G_macierz <- function(beta,x) { #macierz G z konspektu do wykladu nr 7
    G_macierz_0 <- matrix(c(0, 3*n), nrow=n, ncol=3)
  for (k in 1:n) {
    G_macierz_0[k,] <- g_gradient(k, beta, x)
  }
    return(G_macierz_0)
  }
  res_0 <- y-g_funkcja(beta_0, x) #reziduum
  k <- 1
  wyniki <- matrix(rep(0, 3*max_krok), nrow=max_krok, ncol=3) #inicjalizacja ma-
cierzy wyników
  wyniki[k,] <- beta_0
  repeat {
    if (isTRUE(det(t(G_macierz(beta_0, x)) %*% G_macierz(beta_0, x)) > 0) == FAL-
SE ) {
      print('Algorytm rozbie■ny!')
      break
    }
    #ciąg przybli■eń
    beta_next <- beta_0 + inv(t(G_macierz(beta_0, x)) %*% G_macierz(beta_0, x)) %*%
      t(G_macierz(beta_0, x)) %*% res_0
    k <- k+1
    #zadajemu warunki stopu
    if (k >= max_krok || Norm(beta_next-beta_0) <= 0.0001) break;
    beta_0 <- beta_next
    res_0 <- y-g_funkcja(beta_0, x)
    wyniki[k,] <- beta_next

  }
  wyniki <- as.data.frame(cbind(seq(0,99, by=1), wyniki))
  colnames(wyniki)[1] <- 'Iteracja'
  colnames(wyniki)[2] <- 'beta_1'
  colnames(wyniki)[3] <- 'beta_2'
  colnames(wyniki)[4] <- 'beta_3'
  return(wyniki)
}

#Uruchamiamy algorytm
GN <- gauss_newton(x, c(79,101,0.004))
GN[1:5,]

##   Iteracja   beta_1    beta_2      beta_3
## 1        0 79.00000 101.00000 0.004000000
## 2        1 80.78354  98.63311 0.004893729
```

```
## 3        2 79.99157  99.99639 0.004997070
## 4        3 79.98574 100.00771 0.004996694
## 5        4  0.00000   0.00000 0.000000000
```

Możemy uznać, że algorytm Gaussa-Newtona zadziałał przyzwalająco dobrze, ponieważ estymatory dla $\boldsymbol{\beta}$ w przybliżeniu wynoszą odpowiednio

$$\left(\hat{\beta}_1, \hat{\beta}_2, \hat{\beta}_3\right) = (79.986, 100.008, 0.005)\,.$$

Teraz implementujemy estymator wariancji i wyliczamy interesujący nas estymator.

```
#implementujemy estymator wariancji, p=3
wariancja_hat <- function(beta, x) {
  beta <- as.vector(beta)
  return(sum((y-g_funkcja(beta, x))^2)/(n-3))
}
war_hat <- wariancja_hat(c(as.matrix(GN[4,2:4])),x)
war_hat

## [1] 0.5019423
```

Zatem w przybliżeniu

$$\left(\hat{\beta}_1, \hat{\beta}_2, \hat{\beta}_3, \hat{\sigma}^2\right) = (79.986, 100.008, 0.005, 0.502)\,.$$

Ponadto algorytm jest zbieżny w trzech iteracjach. Zwróćmy jednak uwagę na to, że algorytm zadziałał tak dobrze głównie dlatego, że wartości inicjalizujące algorytm są bliskie tym oczekiwanym.

5 Zadanie 4.

Zilustrujmy teraz działanie procedury tabelką pokazującą, dla każdego kroku algorytmu s, wartości estymatorów $\left(\hat{\beta}_1^{(s)}, \hat{\beta}_2^{(s)}, \hat{\beta}_3^{(s)}, \left(\hat{\sigma}^2\right)^{(s)}\right)$ i odpowiadającą im wartość logarytmu funkcji wiarogodności $l\left(\boldsymbol{\beta}, \sigma^2\right)$ zdefiniowaną jako

$$l\left(\boldsymbol{\beta}, \sigma^2\right) = -\frac{n}{2}\log 2\pi - \frac{n}{2}\log\sigma^2 - \sum_{i=1}^{n}\frac{\left(y_i - g\left(\boldsymbol{x}_i, \hat{\boldsymbol{\beta}}\right)\right)^2}{2\sigma^2}.$$

Oczywiście, rozkłady Gaussowskie w funkcji wiarogodności biorą się stąd, że zakłócenia obecne w danych mają rozkład normalny. Teraz przygotujmy kod, by zaimplementować logarytm funkcji wiarogodności i wyliczyć jej odpowiednie wartości wraz z estymatorami wariancji.

```
loglikelihood <- function(beta,wariancja, x){
  v <- -n*log(2*pi)/2 -n/2 *log(wariancja) -1/(2*wariancja)*sum((y-g_funkcja(beta,x))^2)
  return(v)
}
```

```
war_hat_0 <- wariancja_hat(c(as.matrix(GN[1,2:4])), x)
war_hat_0

## [1] 22.12212

loglikelihood(c(as.matrix(GN[1,2:4])),war_hat_0,x)

## [1] -29670.78

war_hat_1 <-wariancja_hat(c(as.matrix(GN[2,2:4])), x)
war_hat_1

## [1] 1.441958

loglikelihood(c(as.matrix(GN[2,2:4])), war_hat_1, x)

## [1] -16017.89

war_hat_2 <-  wariancja_hat(c(as.matrix(GN[3,2:4])), x)
war_hat_2

## [1] 0.4897928

loglikelihood(c(as.matrix(GN[3,2:4])), war_hat_2, x)

## [1] -10619.02

loglikelihood(c(as.matrix(GN[4,2:4])),war_hat, x)

## [1] -10619.34
```

kroki algorytmu	$\hat{\beta}_1^{(s)}$	$\hat{\beta}_2^{(s)}$	$\hat{\beta}_3^{(s)}$	$\left(\hat{\sigma}^2\right)^{(s)}$	$l\left(\hat{\boldsymbol{\beta}}^{(s)}, \left(\hat{\sigma}^2\right)^{(s)}\right)$
0	79	101	0.004	22.122	-29670.78
1	80.784	98.633	0.0049	1.442	-16017.89
2	79.992	99.996	0.004997	0.4898	-10619.02
3	79.986	100.008	0.005	0.502	-10619.34

Tabela 1: Estymatory dla $\boldsymbol{\beta}$ oraz $\boldsymbol{\sigma}^2$ wraz z wartością logarytmu funkcji wiarogodności dla poszczególnych kroków algorytmu

Wniosek 1 *W ciągu kolejnych kroków algorytmu logarytm funkcji wiarogodności zwiększa się. Oznacza to, że z każdym krokiem metody jakość dopasowania modelu do danych jest lepsza. Ponadto estymatory zbiegają do prawdziwych wartości.*

6 Zadanie 5.

Zwizualizujmy porównanie przy pomocy tabelki.

współrzędne	beta1	beta2	beta3
$\hat{\boldsymbol{\beta}}$	79.986	100.008	0.005
$\boldsymbol{\beta}$	80	100	0.005

Tabela 2: Porównanie $\boldsymbol{\beta}$ oraz $\hat{\boldsymbol{\beta}}$

```
Norm(c(as.matrix(GN[4,2:4]))-c(80, 100, 0.005))

## [1] 0.01620841
```

Zatem widzimy, że otrzymane wartości estymatorów są bliskie tym prawdziwym.

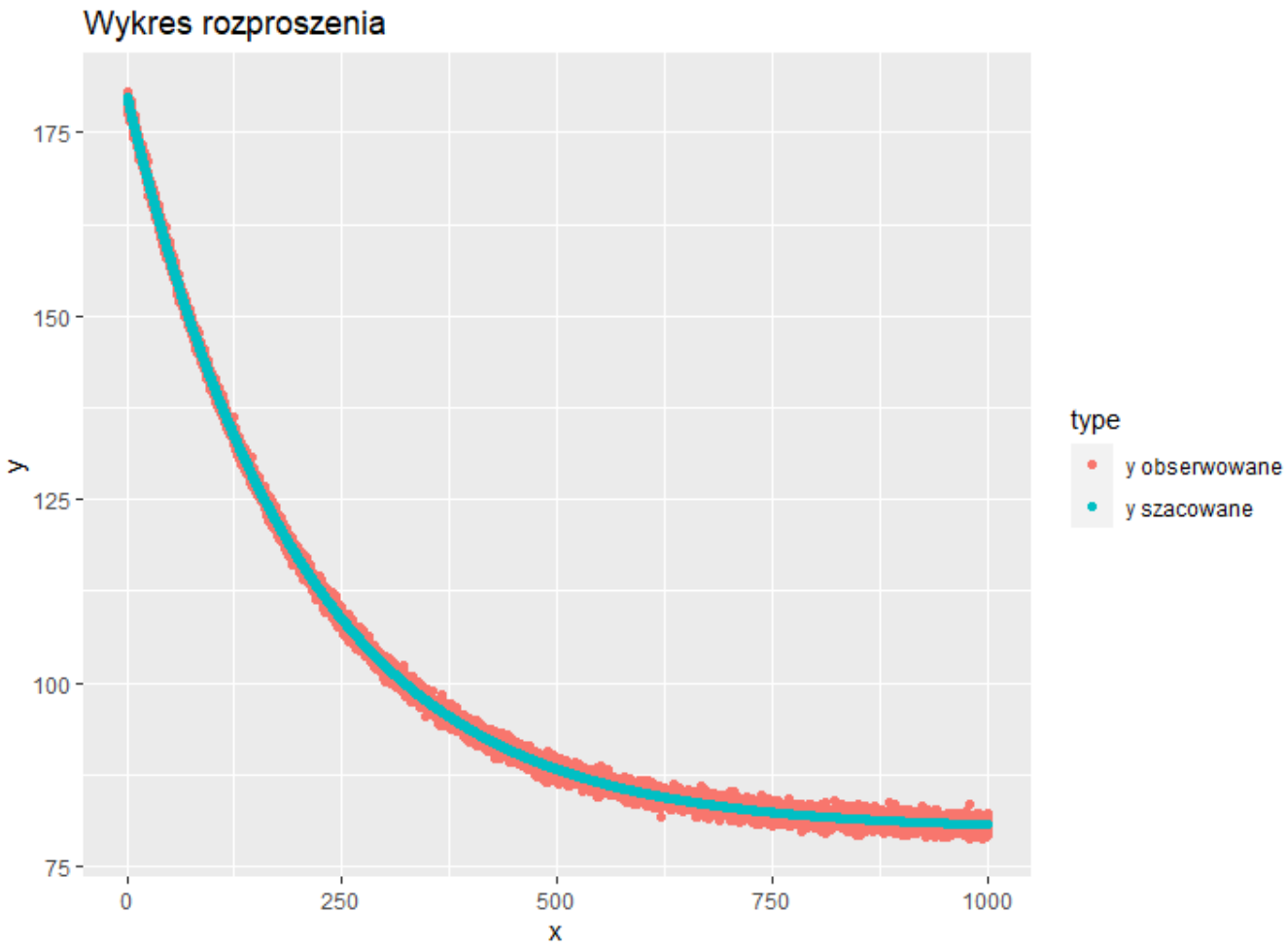
7 Zadanie 6.

```
beta_hat <- c(as.matrix(GN[4,2:4]))
beta <- c(beta_1, beta_2, beta_3)
x1 <- x
x2 <- x
df_1 <- data.frame(x=x1,y=y, type='y obserwowane')
df_2 <- data.frame(x=x2, y=g_funkcja(beta_hat), type='y szacowane')
df <- rbind(df_1, df_2)

ggplot(df) +
  geom_point(aes(x,y,colour=type)) +
  ggtitle('Wykres rozproszenia')
```

Narysujmy teraz wykresy dla $\{(x_i, y_i) : i = 1, \ldots, n\}$ oraz $\{(x_i, \hat{y}_i) : i = 1, \ldots, n\}$, gdzie

$$\hat{y}_i = \hat{\beta}_1 + \hat{\beta}_2 e^{-\hat{\beta}_3 x_i}.$$



Rysunek 1: Wykres rozproszenia dla $(x_1, y_1), \ldots, (x_n, y_n)$ oraz $(x_1, \hat{y}_1), \ldots, (x_n, \hat{y}_n)$

Wniosek 2 *Z rysunku nr 1 wynika, że model jest dobrze dopasowany do danych i dobrze uchwycił zmienność danych.*

8 Zadanie 7.

Szacujemy macierz kowariancji przy pomocy wzoru

$$\text{Cov}\left(\hat{\boldsymbol{\beta}}\right) = \hat{\sigma}^2 \left(\hat{\boldsymbol{G}}^T \hat{\boldsymbol{G}}\right)^{-1},$$

gdzie

$$\hat{\boldsymbol{G}} = \left[\frac{\delta g\left(\boldsymbol{x}_i, \hat{\boldsymbol{\beta}}\right)}{\delta \beta_j}\right].$$

```
#macierz G
G_macierz <- function(beta,x) { #macierz G z konspektu do wykladu nr 7
    G_macierz_0 <- matrix(c(0, 3*n), nrow=n, ncol=3)
  for (k in 1:n) {
```

```
    G_macierz_0[k,] <- g_gradient(k, beta, x)
  }
    return(G_macierz_0)
}

#estymowana macierz kowariancji
kow_beta_hat <- war_hat * inv(t(G_macierz(c(as.matrix(GN[4,2:4])), x)) %*%
                                G_macierz(c(as.matrix(GN[4,2:4])),x))

kow_beta_hat

##               [,1]          [,2]         [,3]
## [1,]  1.929041e-04 -2.424265e-05 3.587902e-08
## [2,] -2.424265e-05  1.009160e-03 4.587451e-08
## [3,]  3.587902e-08  4.587451e-08 1.170770e-11
```

9 Zadanie 8.

Teraz sprawdzimy działanie algorytmu Gaussa-Newtona w przypadku (a), kiedy punkt startowy nieznacznie różni się od wartości prawdziwych, oraz w przypadku (b), kiedy punkt startowy znacznie różni się od wartości prawdziwych.

(a) Niech $\left(\beta_1^0, \beta_2^0, \beta_3^0\right) = (73, 110, 0.01)$.

```
gauss_newton(x, c(73,110, 0.01))[1:5,]

##   Iteracja   beta_1    beta_2      beta_3
## 1        0 73.00000 110.00000 0.010000000
## 2        1 83.48982  88.82717 0.003185263
## 3        2 84.14709  93.78763 0.004913122
## 4        3 80.01491  99.95451 0.005005862
## 5        4  0.00000   0.00000 0.000000000
```

Dla punktu inicjalizującego, relatywnie bliskiego prawdziwemu punktowi, algorytm działa dobrze i w ciągu trzech iteracji procedury dostajemy wartości bliskie oczekiwanym.

(b) Niech $\left(\beta_1^0, \beta_2^0, \beta_3^0\right) = (50, 120, 0.4)$.

```
gauss_newton(x, c(50,120,0.4))[1:5,]

## [1] "Algorytm rozbie■ny!"
##   Iteracja   beta_1     beta_2     beta_3
## 1        0 50.00000 120.000000  0.4000000
## 2        1 99.07924  -1.548999 -0.6508504
## 3        2  0.00000   0.000000  0.0000000
## 4        3  0.00000   0.000000  0.0000000
## 5        4  0.00000   0.000000  0.0000000
```

Teraz widzimy, że dla punktu startowego znacznie różniącego się od prawdziwej $\hat{\boldsymbol{\beta}}$ algorytm Gaussa-Newtona zbiega do kompletnie innej wartości, wybuchając po drodze.

Wniosek 3 *Algorytm Gaussa-Newtona jest nieskuteczny w przypadku obrania zbyt dalekiego punktu startowego w stosunku do punktu oczekiwanego.*