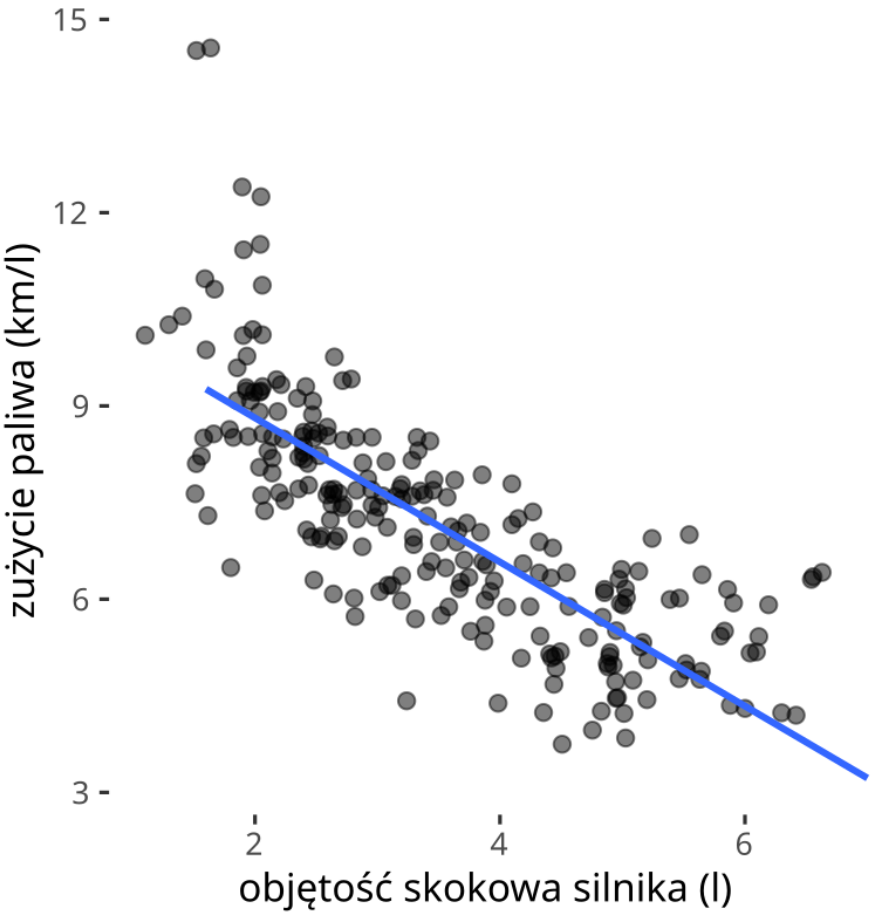
# Regresja liniowa

## 1 Definicja

**Regresja liniowa** – w modelowaniu statystycznym, metody oparte na liniowych kombinacjach zmiennych i parametrów dopasowujących model do danych. Dopasowana linia lub krzywa regresji reprezentuje oszacowaną wartość oczekiwaną zmiennej y przy konkretnych wartościach innej zmiennej lub zmiennych x. W najprostszym przypadku dopasowana jest stała lub funkcja liniowa.




Rysunek 1: Przykład danych z modelem liniowym dopasowanym metodą najmniejszych kwadratów.

## 2 Przykładowy kod PyTorch, Python

```python
import torch
import torch.nn as nn
import torch.optim as optim

# Dane wejściowe i wyjściowe
X = torch.tensor([[1.0], [2.0], [3.0], [4.0]], requires_grad=True)
y = torch.tensor([[2.0], [3.0], [4.0], [5.0]])

# Definicja modelu liniowego
model = nn.Linear(1, 1)

# Ustawienie innych wartości wagi i przesunięcia
model.weight.data.fill_(1.0)
model.bias.data.fill_(0.5)

# Funkcja strat
criterion = nn.MSELoss()

# Optymalizator
optimizer = optim.SGD(model.parameters(), lr=0.01)

# Liczba epok
epochs = 3

for epoch in range(epochs):
    # Przewidywanie
    y_pred = model(X)

    # Obliczanie straty
    loss = criterion(y_pred, y)

    # Zerowanie gradientów
    optimizer.zero_grad()

    # Obliczanie gradientów
    loss.backward()

    # Aktualizacja wag
    optimizer.step()

    # Wyświetlanie straty
    print(f'Epoch {epoch+1}, Loss: {loss.item()}')
```

# 3 Analiza kodu i obliczenia/wzory

Obliczanie regresji liniowej zaczyna się od utworzenia dwóch macierzy. Do analizy bierzemy dwie przykładowe macierze.

$$X = \begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \\ 4.0 \end{bmatrix}, \quad y = \begin{bmatrix} 2.0 \\ 3.0 \\ 4.0 \\ 5.0 \end{bmatrix}$$

Ta część kodu odpowiada za definiowanie funkcji liniowej. Parametry funkcji, czyli W (weight) oraz b (bias) ustawiane są losowo w domyślny sposób, ale możemy także je ustawić na starcie.

```
model = nn.Linear(1, 1)
```

Ustawianie wartości W i b dokonujemy dzięki poniższym komendą. Dzięki temu decydujemy z jakich wartości będą startować te zmienne.

```
model.weight.data.fill_(1.0)
model.bias.data.fill_(0.5)
```

Zatem w tym przypadku wartości te wyglądają następująco.

$$\text{W} = 1, \text{b} = 0.5$$

Zaś wzorem funkcji jest:

$$\text{y} = \text{W} * \text{X} + \text{b}$$

Ustawiamy używanie funkcji straty MSE, czyli obliczanie średniego kwadratowego błędu estymacji oraz współczynikka uczenia lr, który będzie odgrywał role w aktualizacji wag.

```
criterion = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)
```

Następnie tworzymy pętle, która będzie odpowiedzialna za trenowanie modelu. Model w pierwszym kroku tworzy przewidywania używając wzoru funkcji. Część kodu za to odpowiedzialna:

```
y_pred = model(X)
```

W tym przypadku obliczenia wyglądają następująco:

$$y_{pred} = \begin{bmatrix} 1.0 \cdot 1 + 0.5 \\ 2.0 \cdot 1 + 0.5 \\ 3.0 \cdot 1 + 0.5 \\ 4.0 \cdot 1 + 0.5 \end{bmatrix} = \begin{bmatrix} 1.5 \\ 2.5 \\ 3.5 \\ 4.5 \end{bmatrix}$$

Następnie trzeba obliczyć MSE. Odpowiada za to ta część kodu:

```
loss = criterion(y_pred, y)
```

Funkcja ta korzysta z poniższego wzoru.

$$\text{MSE} = \frac{1}{n} \cdot \sum (y - y_{pred})^2$$

Zatem obliczenia wyglądają następująco.

$$\text{MSE} = \frac{1}{4} \cdot \sum( \begin{bmatrix} 2.0 \\ 3.0 \\ 4.0 \\ 5.0 \end{bmatrix} - \begin{bmatrix} 1.5 \\ 2.5 \\ 3.5 \\ 4.5 \end{bmatrix})^2 = \frac{1}{4} \cdot \sum( \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{bmatrix})^2 = \frac{1}{4} \cdot \sum( \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix}) = \frac{1}{4} \cdot 1 = 0.25$$

Następnie, aby PyTorch nie dodał nowych wag do już istniejących tylko dał te nowe trzeba wyzerować gradienty.

```
optimizer.zero_grad()
```

Za pomocą następnej komendy obliczamy gradienty.

```
loss.backward()
```

Wzór potrzebny do obliczeń wygląda następująco.

**Gradient dla W.**

$$\text{W} = \frac{\partial Loss}{\partial W} = -\frac{2}{n} \sum_{i=1}^{n} (y_{pred} - y) \cdot X$$

**Gradient dla b.**

$$\text{b} = \frac{\partial Loss}{\partial b} = -\frac{2}{n} \sum_{i=1}^{n} (y_{pred} - y)$$

**Obliczenia gradientów**

$$\text{W} = \frac{2}{4} \cdot \sum( \begin{bmatrix} 1.5 \\ 2.5 \\ 3.5 \\ 4.5 \end{bmatrix} - \begin{bmatrix} 2.0 \\ 3.0 \\ 4.0 \\ 5.0 \end{bmatrix}) \cdot \begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \\ 4.0 \end{bmatrix} = \frac{1}{2} \cdot \sum( \begin{bmatrix} -0.5 \\ -0.5 \\ -0.5 \\ -0.5 \end{bmatrix}) \cdot \begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \\ 4.0 \end{bmatrix} = \frac{1}{2} \cdot (-0.5 - 1.0 - 1.5 - 2.0) = \frac{1}{2} \cdot (-5.0) = -2.5$$

$$\text{b} = \frac{2}{4} \cdot \sum( \begin{bmatrix} 1.5 \\ 2.5 \\ 3.5 \\ 4.5 \end{bmatrix} - \begin{bmatrix} 2.0 \\ 3.0 \\ 4.0 \\ 5.0 \end{bmatrix}) = \frac{1}{2} \cdot \sum( \begin{bmatrix} -0.5 \\ -0.5 \\ -0.5 \\ -0.5 \end{bmatrix}) = \frac{1}{2} \cdot (-2) = -1$$

Teraz trzeba zaktualizować wagi.

```
optimizer.step()
```

Wzory na aktualizacje wag.

$$\text{W} = W - lr \cdot \frac{\partial Loss}{\partial W}$$

$$\text{b} = b - lr \cdot \frac{\partial Loss}{\partial b}$$

Obliczenia wag.

$$\text{W} = 1 - 0.01 \cdot (-2.5) = 1 + 0.025 = 1.025$$

$$\text{b} = 0.5 - 0.01 \cdot (-1) = 0.5 + 0.01 = 0.51$$

W kolejne kroki są takie same, a po prostu zmienia się W oraz b. W kolejnych iteracjach dąży się do coraz bardziej dokładnego wyniku.

# 4 Inna ilość wejść i wyjść w regresji liniowej

Dane w przykładach będą losowe.

W każdym przypadku regresja liniowa ma wzór następujący.

$$y=Wx+b$$

W pierwszym przypadku mamy funkcję liniową z wejściem W o wymiarze 1 W (waga) to 3 wiersze x kolumny, a bias ma wymiar 3.

```
nn.Linear(1, 3)
```

Przykładowo weźmy niech macierz wejścia wygląda następująco.

$$X = \begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \end{bmatrix}$$

Wagi moga wyglądać tak.

$$W = \begin{bmatrix} 0.7645 \\ 0.83 \\ -0.2343 \end{bmatrix}$$

$$b = \begin{bmatrix} -0.9186 \\ = 0.2191 \\ 0.2018 \end{bmatrix}$$

Obliczenia będą wyglądać tak dla $y_{pred}$.

$$y_{pred} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$a_{11} = 0.7645 \cdot 1.0 + 0.9186 = 1.6831$

$a_{12} = 0.83 \cdot 1.0 - 0.2191 = 0.6109$

$a_{13} = -0.2343 \cdot 1.0 + 0.2018 = -0.0325$

$a_{21} = 0.7645 \cdot 2.0 + 0.9186 = 2.4477$

$a_{22} = 0.83 \cdot 2.0 - 0.2191 = 1.4409$

$a_{23} = -0.2343 \cdot 2.0 + 0.2018 = -0.2668$

$a_{31} = 0.7645 \cdot 3.0 + 0.9186 = 3.2122 = 3.2122$

$a_{32} = 0.83cdot3.0 - 0.2191 = 2.2709$

$a_{33} = -0.2343 \cdot 3.0 + 0.2018 = -0.5010$

Czyli macierz $y_{pred}$ wygląda następująco.

$$y_{pred} = \begin{bmatrix} 1.6831 & 0.6109 & -0.0325 \\ 2.4477 & 1.4409 & -0.2668 \\ 3.2122 & 2.2709 & -0.5010 \end{bmatrix}$$

W dalszej części obliczenia wyglądają podobnie. Trzeba obliczyć MSE, gradienty i zaktualizować wagi (używając dalej kodu).

Następny przypadek ma W, czyli 1 wiersz x 3 kolumny i bias o wymiarze 1.

```
nn.Linear(3, 1)
```

Przykładowa macierzy wejścia może wyglądać następująco.

$$X = \begin{bmatrix} 1.0 & 2.0 & 3.0 \\ 4.0 & 5.0 & 6.0 \\ 7.0 & 8.0 & 9.0 \end{bmatrix}$$

Wagi funkcji mogą wyglądać tak.

$$W = \begin{bmatrix} 0.4414 \\ 0.4792 \\ -0.1353 \end{bmatrix}$$

$$\mathrm{b} = 0.5304$$

$$y_{pred} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

$a_1 = 0.4414 \cdot 1.0 + 0.4792 \cdot 2.0 - 0.1353 \cdot 3.0 + 0.5304 = 1.5244$

$a_2 = 0.4414 \cdot 4.0 + 0.4792 \cdot 5.0 - 0.1353 \cdot 6.0 + 0.5304 = 3.8805$

$a_3 = 0.4414 \cdot 7.0 + 0.4792 \cdot 8.0 - 0.1353 \cdot 9.0 + 0.5304 = 6.2365$

A zatem $y_{pred}$ w tym przypadku wynosi.

$$y_{pred} = \begin{bmatrix} 1.5244 \\ 3.8805 \\ 6.2365 \end{bmatrix}$$

Warto także rozważyć jeszcze jeden przypadek. W poniższym przypadku są dla W 3 wiersze x 3 kolumny oraz b ma wymiar 3.

```
nn.Linear(3, 3)
```

Przykładowa macierz wejścia.

$$X = \begin{bmatrix} 1.0 & 2.0 & 3.0 \\ 4.0 & 5.0 & 6.0 \\ 7.0 & 8.0 & 9.0 \end{bmatrix}$$

Niech wagi wyglądają następująco.

$$W = \begin{bmatrix} 0.4414 & 0.4792 & -0.1353 \\ 0.5304 & -0.1265 & 0.1165 \\ -0.2811 & 0.3391 & 0.5090 \end{bmatrix}$$

$$b = \begin{bmatrix} -0.4236 \\ 0.5018 \\ 0.1081 \end{bmatrix}$$

Obliczenia wyglądają następująco

$$y_{pred} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$a11 = 0.4414 \cdot 1.0 + 0.4792 \cdot 2.0 - 0.1353 \cdot 3.0 - 0.4236 = 0.5705$

$a12 = 0.5304 \cdot 1.0 - 0.1265 \cdot 2.0 + 0.1165 \cdot 3.0 + 0.5018 = 1.1287$

$a13 = -0.2811 \cdot 1.0 + 0.3391 \cdot 2.0 + 0.5090 \cdot 3.0 + 0.1081 = 2.0320$

$a21 = 0.4414 \cdot 4.0 + 0.4792 \cdot 5.0 - 0.1353 \cdot 6.0 - 0.4236 = 2.9265$

$a22 = 0.5304 \cdot 4.0 - 0.1265 \cdot 5.0 + 0.1165 \cdot 6.0 + 0.5018 = 2.6898$

$a23 = -0.2811 \cdot 4.0 + 0.3391 \cdot 5.0 + 0.5090 \cdot 6.0 + 0.1081 = 3.7328$

$a31 = 0.4414 \cdot 7.0 + 0.4792 \cdot 8.0 - 0.1353 \cdot 9.0 - 0.4236 = 5.2826$

$a32 = 0.5304 \cdot 7.0 - 0.1265 \cdot 8.0 + 0.1165 \cdot 9.0 + 0.5018 = 4.2509$

$a33 = -0.2811 \cdot 1.0 + 0.3391 \cdot 2.0 + 0.5090 \cdot 3.0 + 0.1081 = 5.4336$

Ostateczny wynik dla $y_{pred}$ wynosi.

$$y_{pred} = \begin{bmatrix} 0.5705 & 1.1287 & 2.0320 \\ 2.9265 & 2.6898 & 3.7328 \\ 5.2826 & 4.2509 & 5.4336 \end{bmatrix}$$