Table of Contents



# Regresja logistyczna

## Wykład

Slajdy

## Regresja logistyczna jednej zmiennej

Proszę zaimportować potrzebne biblioteki oraz bazę danych: Dane:

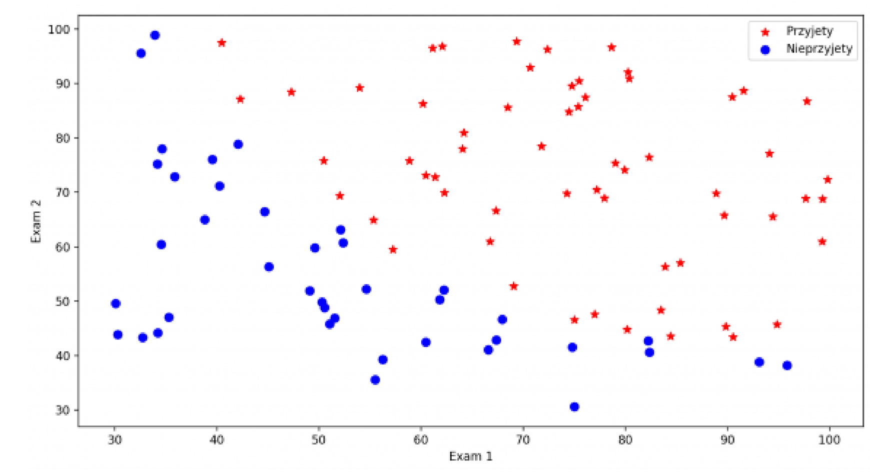
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import os
path = os.getcwd() + '/dane_egz.txt.pdf'
data = pd.read_csv(path, header=None, names=['Exam 1', 'Exam 2', 'Admitted'])
```

**Zad 0.:** Proszę zapoznać się z danymi.

**Cel:** W danych występują dwie zmienne ciągłe niezależne - `Exam 1` i `Exam 2`. Naszym celem jest określenie, czy student został przyjęty na uczelnię na podstawie wyników z egzaminów, czy też nie (target - `Admitted`). Przewidywanie/wyjaśnienie `Y` przyjmuje dwie możliwe wartości/klasy - mamy do czynienia z *klasyfikacją binarną*. Wartość 1 oznacza, że uczeń został przyjęty, a wartość 0 oznacza, że uczeń nie został przyjęty. **Zad 1.:** Proszę wykonać następujące kroki:

- dodać kolumnę z wartościami '1'
- podzielić dane na parametry (X) oraz etykiety/klasy (y)
- konwersja danych do `numpy` (`np.array`)

**Zad 2.:** Proszę przedstawić na wykresie wczytane dane (`scatter plot`).




**Zad 3.:** Wcześniej powiedzieliśmy, że chcemy aby nasz klasyfikator $h_\theta(x)$ spełniał właściwość:

$$0 \leq h_\theta(x) \leq 1$$

W regresji liniowej używaliśmy hipotezy:

$$h_\theta(x) = \theta^T x$$

Teraz dokonujemy lekkiej modyfikacji, a dokładnie składamy funkcję hipotezy z regresji liniowej z nową funkcją $sig$:

$$h_\theta(x) = sig(\theta^T x)$$

Funkcja $sig$ to tzw. funkcja logistyczna (sigmoida):

$$sig(z) = \frac{1}{1+e^{-z}}$$

Co sprowadza do tego że:

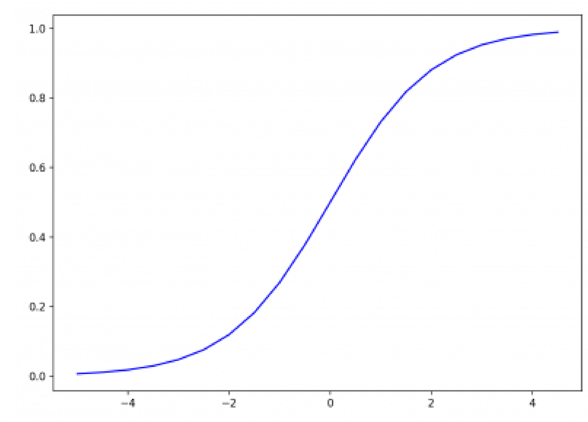
$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

Proszę zaimplementować funkcję logistyczną (sigmoida):

$$sig(t) = \frac{1}{1+e^{-t}}$$

```
def sig(t)
#TODO
```

**Zad 3.:** Przy pomocy funkcji `np.arrange` proszę wygenerować dane z zakresu `[-5,5]`, krok `0.5` i sprawdzić poprawne działanie zaimplementowanej funkcji.




Zauważamy, że sigmoida posiada asymptotę w $y = 1$ dążąca do $+\infty$, oraz w $y = 0$ dążącą do $-\infty$.

Mamy zbiór treningowy $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$, składający się z $m$ przykładów treningowych.
Jak wybrać parametry $\theta$ ?

Dla regresji liniowej, używaliśmy błędu średniokwadratowego

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$$

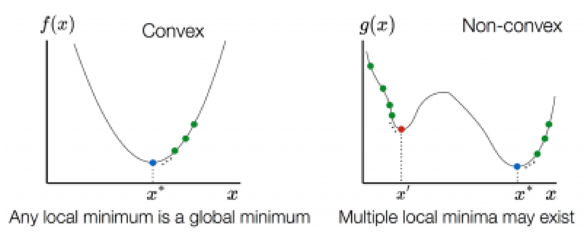
Zdefiniujmy to w następujący sposób:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)})$$

$$Cost(h_\theta(x), y) = \frac{1}{2}(h_\theta(x) - y)^2$$

Funkcja $Cost$ reprezentuje koszt, który musi “zapłacić” nasz algorytm, jeśli zwraca odpowiedź $h_\theta(x)$, gdy rzeczywistą wartością jest $y$.

Możemy użyć tej funkcji podczas minimalizacji regresji logistycznej, ale istnieje problem, mianowicie jest to funkcja niewypukła. Sigmoida w hipotezie wprowadza nieliniowość, która powoduje w powyższej funkcji kosztu powstanie wielu minimów lokalnych.




**Funkcja kosztu dla regresji logistycznej**

Użyjemy następującej funkcji kosztu:

$$Cost(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{gdy } y = 1 \\ -\log(1 - h_\theta(x)) & \text{gdy } y = 0 \end{cases}$$

Mamy

$$Cost(h_\theta(x), y) = -y log(h_\theta(x)) - (1-y) log(1 - h_\theta(x))$$

Możemy teraz podstawić powyższy wzoru na $J(\theta)$

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)}) = \frac{1}{m}\sum_{i=1}^{m}[-y^{(i)} log(h_\theta(x^{(i)})) - (1-y^{(i)}) log(1 - h_\theta(x^{(i)}))]$$

Można się zastanawiać dlaczego zdecydowaliśmy się właśnie na tą funkcję kosztu. Poza tym że ma tą pozytywną cechę że jest wypukła, może ona zostać wyprowadzona z praw statystyki przy użyciu zasady maksymalnego prawdopodobieństwa, czyli zasady mówiącej o tym jak wydajnie szukać parametrów różnych modeli.
**Zad 4.** Na podstawie powyższych wzorów proszę zaimplementować funkcję kosztu (rozwiązanie zwektoryzowane):

```
def cost(theta, X, y):
#TODO
```

W celu sprawdzenia działania funkcji `cost` prosze zainicjalizować wartości $\theta$:

```
theta = np.zeros(3)
```

Poprawna wartość funkcji kosztu dla analizowanych danych wynosi `0.69`.
**Metoda gradientu prostego**
Naszym dalszym celem będzie znalezienie takich parametrów $\theta$ dla których $J(\theta)$ będzie minimalne. Można to osiągnąć za pomocą metody gradientu prostego, czyli powtarzać

$$\theta_j = \theta_j - \alpha\frac{d}{d\theta_j}J(\theta)$$

Gdzie $\alpha$ oznacza długość kroku w każdej iteracji. Pochodna:

$$\frac{d}{d\theta_j}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

**Zad 5.**: Proszę zaimplementować funkcję gradientu prostego:

```
def gradient_prosty(X, y, theta, alpha, it):
    # it - liczba iteracji


    return theta, cost
```

Dla parametru `alpha=1` oraz 150 iteracji funkcja kosztu wynosi `0.20` a wartości $\theta$ [1.65947664],[3.8670477],[3.60347302]. W przypadku różnicy, proszę pamiętać o normalizacji danych. Wyniki mogą się różnić.

**Zad 6.:** Proszę przedstawić skuteczność (ang. accuracy) działania algorytmu. Wartości predykcji będą w zakresie [0;1]. Progowanie wartości 0.5.

## Regresja logistyczna - Python

Rozwiązaniem alternatywnym do jawnej implementacji problemu regresji liniowej jest skorzystanie z biblioteki `scikit-learn` z obiektu `LogisticRegression`.

### Zadanie 1 - regresja logistyczna dla problemu kilku klas

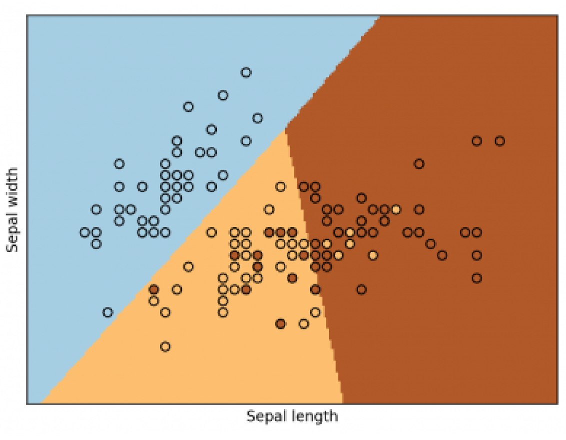
Proszę wczytać podstawowe biblioteki oraz bazę danych `Iris`:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn import datasets

iris = datasets.load_iris()
X = iris.data[:, :2]  # analizujemy tylko dwa parametry
Y = iris.target
```

**Zad 1.:** Proszę zapoznać się z obiektem sklearn.linear_model.LogisticRegression i wybrać algorytm optymalizacji, współczynnik regularyzacji i stworzyć odpowiedni model regresji liniowej.
**Zad 2.:** Utwórz instancję i dopasuj dane - `fit`.

Poniższy kod pozwoli wyświetlić podział regionów decyzyjnych:

```
# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, x_max]x[y_min, y_max].
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
h = .02  # step size in the mesh
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = logreg.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors='k', cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(())
plt.yticks(())

plt.show()
```




**Zad 3.:** Stosując metodę `predict_proba` proszę przewidzieć prawdopodobieństwo przynależności do danej klasy.
**Zad 4.:** Proszę określić skuteczność działania algorytmu.