

- AIR – Inf1
- AIR – PAOM
- AIR – ML
- Inf – AdvML

- Prace mgr 2018/2019

- Jak korzystać z wiki?
- WiFi AGH
- UCI AGH

- Regresja liniowa
  - Wykład
  - Regresja liniowa jednej zmiennej
  - Regresja liniowa wielu zmiennych
  - Regresja liniowa - Python
    - Zadanie - zbiór danych Boston

## Regresja liniowa

### Wykład




### Regresja liniowa jednej zmiennej

Proszę zaimportować podstawowe biblioteki oraz bazę danych:

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

NumPy jest podstawowym pakietem wykorzystywanym do obliczeń naukowych w języku Python. Pozwala między innymi na wykonywanie wydajnych operacji na macierzach, obliczenia numeryczne, obliczenia z zakresu algebry liniowej, FFT etc. Stanowi darmową alternatywę dla MATLABa.

W pierwszej części ćwiczenia mamy za zadanie zastosowanie regresji liniowej z jedną zmienną do przewidywania zysków przyczepy gastronomicznej -food truck. Rozważamy różne miasta, aby otworzyć nowy punkt sprzedaży. Sieć ma już ciężarówki w różnych miastach i masz dane dotyczące zysków i ludności z miast. Baza danych: Dane (usuń rozszerzenie .pdf)

```
import os
path = os.getcwd() + '/dane1.txt'
data = pd.read_csv(path, header=None, names=['Population', 'Profit'])
```

**Zad 1.** Proszę wywołać metodę `head` oraz `describe` i zapoznać się z danymi.

**Zad 2.** Przedstaw na wykresie dane: oś x - populacja, oś y - zysk.

Celem regresji liniowej jest dopasowanie do danych doświadczalnych/historycznych prostej, która oddaje charakter tych danych. Zadanie polega na znalezieniu funkcji liniowej  $f(x) \rightarrow \mathbf{R}$  postaci:

$$f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots \theta_n x_n \tag{1}$$

gdzie  $x \in \mathbb{R}^N$  a  $\theta \in \mathbb{R}^N$  jest wektorem zawierającym współczynniki prostej. I to właśnie na znalezieniu wektora w będziemy skupiać swoją uwagę.

Naszym zadaniem będzie minimalizacja funkcji kosztu:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m \left( f(x^{(i)}) - y^{(i)} \right)^2 \tag{2}$$

**Zad 3.** Najpierw stwórzmy funkcję do obliczenia funkcji kosztu danego rozwiązania (charakteryzującego się parametrami  $\theta$ ).

```
def computeCost(X, y, theta):
    #rozwiązanie zadania (ok 2-3 Linijki kodu, wersja zwektoryzowana)
```

**Zad 4.** Aby skorzystać z wektoryzowanej wersji rozwiązania - do analizowanych danych konieczne jest dodanie pierwszej kolumny wypełnionej wartościami 1.

**Zad 5.** Podziel zbiór danych na dane wejściowe x (kolumna 0,1) - populacja oraz odpowiedź y - zysk (kolumna 3). Sprawdź przy pomocy `head`, czy dane zostały poprawnie rozdzielone:

```
x =
y =
```

Funkcja kosztu spodziewa się macierzy typu `numpy`, więc musimy przekonwertować  $X$  i  $y$ , zanim będziemy mogli z nich korzystać. Musimy również zainicjować  $\theta$ .

```
X = np.matrix(X.values)
y = np.matrix(y.values)
theta = np.matrix(np.array([0,0]))
```

**Zad 6.** Proszę obliczyć funkcję kosztu naszego początkowego rozwiązania ( $\theta = 0$ ).

Wynik: 32.07

#### Metoda gradientu prostego

W celu dopasowania parametrów regresji liniowej  $\theta$  do naszego zbioru danych zaimplementowana zostanie metoda gradientu prostego.

Celem regresji liniowej jest zminimalizowanie funkcji kosztu:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m \left( f(x^{(i)}) - y^{(i)} \right)^2 \tag{3}$$

gdzie hipoteza  $h_{\theta}(x)$  jest podana przez model liniowy:

$$h_{\theta}(x) = \theta^T x = \theta_0 + \theta_1 x \tag{4}$$

w celu optymalizacji parametrów  $\theta$  zaimplementowany zostanie algorytm gradientowy zbiorczy (ang. batch gradient descent). *Batch* dlatego, że w każdym kroku algorytmu korzystamy z całego zbioru treningowego.

Jak wygląda krok metody gradientu prostego?

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

Każdy krok polega na jednoczesnym poprawianiu parametrów  $\theta_j$  o wartość gradientu funkcji kosztu w stronę parametru, który poprawiamy, stąd w powyższym wzorze pochodna cząstkowa  $\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ .

Parametr  $\alpha$  steruje długością kroku (ang. learning rate).

Jak widać, we wzorze na gradient (1) występuje pochodna:  $\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ . Musimy ją więc wyznaczyć:

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 = \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^m (\theta_0 + \theta_1 x^{(i)} - y^{(i)})^2$$

Potrzebujemy tych pochodnych dla  $j = 0$  i  $j = 1$ :

$$j = 0 : \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \quad j = 1 : \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x^{(i)}$$

Wstawmy otrzymane wartości do wzoru na gradient:

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x^{(i)}$$

Powyższe parametry poprawiamy oczywiście równocześnie.

**Zad.7** Proszę zaimplementować algorytm gradientu prostego:

- funkcja zwraca zoptymalizowane wartości  $\theta$  oraz wektor wartości funkcjo kosztu dla każdej iteracji

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

    return theta, cost
```

inicjalizacja parametrów:

```
alpha = 0.01
it = 1000
```

**Zad 8.** Proszę wyliczyć optymalne parametry dla analizowanego zbioru danych.

**Zad 9.** Przy pomocy wyliczonych parametrów  $\theta$  proszę podać wartość funkcji kosztu.

**Zad 10.** Proszę przedstawić na wykresie wynik regresji liniowej oraz analizowane dane.

**Zad 11.** Proszę narysować wykres przedstawiający zależność funkcji kosztu od iteracji.

### Regresja liniowa wielu zmiennych

Do wykonania kolejnego zadania proszę pobrać bazę danych  housing.

Zestaw danych zawiera ceny mieszkań z 2 zmiennymi - wielkość domu w stopach kwadratowych i liczba sypialni. Celem zdania jest wyznaczenie ceny domu (target) oraz cel (cena domu). Do wykonania zadania wykorzystane zostaną wcześniej zaimplementowane funkcje. Jeżeli zostały poprawnie zaimplementowane - w wersji zwektoryzowanej - zadziałają także dla tego przykładu.

```
path = os.getcwd() + '/dane2.txt'
data2 = pd.read_csv(path, header=None, names=['Size', 'Bedrooms', 'Price'])
data2.head()
```

**Zad 1.** Proszę znormalizować (standaryzacja) dane wykorzystując bibliotekę `panda`. Standaryzacja, to przekształcenie danych, w wyniku którego zmienna uzyskuje średnią równą 0 a odchylenie standardowe równe 1. Standaryzacja polega na wyliczeniu średniej i odchylenia standardowego zmiennej, a następnie odjęciu średniej od wszystkich wartości i podzieleniu otrzymanej różnicy przez odchylenie standardowe.

**Zad X.** Proszę wykonać kroki 3-11 z poprzedniego zadania poprzez wywołanie tych samych funkcji.

### Regresja liniowa - Python

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

Implementacja dowolnego typu regresji jest bardzo prosta, tworzymy obiekt odpowiedniej klasy (`LinearRegression`, `Ridge`, `Lasso`, `ElasticNet`), na którym wywołujemy metodę `fit` podając jej za argumenty zbiór treningowy, docelowe wartości oraz ewentualne parametry ( $\alpha, p$ ). Następnie wywołujemy metodę `predict`, która zwróci przewidywaną wartość.

Szablon rozwiązania można przedstawić w następujący sposób:

```
import numpy as np
from sklearn import datasets, linear_model

# Import danych
# Normalizacja/Standaryzacja
# Podział na zbiór treningowy i testowy (70-30%)

# Stworzenie obiektu
regr = linear_model.LinearRegression()

# Uczenie modelu przy pomocy bazy treningowej
regr.fit(X_train, Y_train)
# Przewidywanie wartości dla danych testowych
Y_predicted = regr.predict(X_test)

# Wyświetlenie parametrów prostej
print('Coefficients: \n', regr.coef_)

# Obliczamy rzeczywisty popetniony błąd średnio-kwadratowy
error = np.mean((regr.predict(X_test) - Y_test) ** 2)
print("Residual sum of squares: {}".format(error))
```

#### Zadanie - zbiór danych Boston

W zadaniu wykorzystamy zbiór Boston House Pricing, który ma 14 atrybutów oraz 506 instancji, zadanie polega na przewidzeniu ostatniego 14 parametru będącego ceną domu.

Wczytanie zbioru danych:

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

# Regression models
# http://scikit-learn.org/stable/modules/linear_model.html

# Load the diabetes dataset
boston = datasets.load_boston()
# print description
print(boston.DESCR)
# get the data
boston_X = boston.data
boston_Y = boston.target
```

**Zad 1.** Proszę podzielić zbiór na część treningową i testową.

**Zad 2.** Zgodnie z powyższym przykładem proszę wykonać uczenia modelu regresji liniowej oraz predykcji wartości.

**Zad 3.** Przedstaw wizualizację prostych regresji obliczonych dla poszczególnych zmiennych w zbiorze danych Boston.

**Zad 4.** Dokonaj porównania modeli regresji (`Linear Regression`, `Lasso`, `Ridge`, `ElasticNet`) pod względem błędu dopasowania(wartości błędu średnio kwadratowego).

```
reg_LinReg =linm.LinearRegression()
reg_Ridge = linm.Ridge(alpha = .5)
reg_Lasso = linm.Lasso(alpha = 5.1)
reg_ElNet =linm.ElasticNet(alpha = .5, l1_ratio=0.5)
```