Wstęp do Sztucznej Inteligencji - rok akademicki 2022/2023

Przed rozpoczęciem pracy z notatnikiem zmień jego nazwę zgodnie z wzorem: `NrAlbumu_Nazwisko_Imie_PoprzedniaNazwa`.

Przed wysłaniem notatnika upewnij się, że rozwiązałeś wszystkie zadania/ćwiczenia.

Temat: Wprowadzenie do języka Python cz. II

Zapoznaj się z treścią niniejszego notatnika czytając i wykonując go komórka po komórce. Wykonaj napotkane zadania/ćwiczenia.

Moduły i skrypty

Z plikami z rozszerzeniem `.py` można pracować na 2 sposoby.

- jak ze skryptem (ala plik wykonywalny),
- jak z modułem (biblioteką) w innych projektach.

Utwórz plik tekstowy `pytest.py` o następującej treści:

```
def suma(a,b):
    return a+b

def roznica(a,b):
    return a-b

PI = 3.1415

print(suma(3,5))
```

W notatniku Jupyter (Colab) można to zrobić wykorzystując tzw. "funkcje magiczne" jak w komórce poniżej:

```
%%writefile pytest.py
def suma(a,b):
    return a+b

def roznica(a,b):
    return a-b

PI = 3.1415

print(suma(3,5))
```

Aby wykonać plik `pytest.py` należy w terminalu wykonać poniższe polecenie:

```
python pytest.py
```

W notatniku Jupyter (Colab) można to zrobić rozpoczynając polecenie znakiem `!` co oznacza że nie są to polecienia języka Python a polecenia w bash'u.

Oczywiście należy być w folderze w którym jest plik `pytest.py` lub podać pełną scieżkę do pliku.

```
!python pytest.py
```

Po wykonaniu polecenia na ekranie powinien pojawić się wynik 8.

Możemy również używać pliku `pytest.py` importując go tak jak wcześniej poznane modułu.

```
import pytest
```

W tym przypadku plik również został wykonany (pojawiło się 8), ale mamy też możliwość wykorzystania obiektów zdefiniowanych w naszym module.

```
pytest.suma(1,4)

pytest.roznica(pytest.PI,1)
```

Aby podczas importowania nie wykonywała się linia `print(suma(3,5))` powinna ona znaleźć się w bloku `if __name__=="__main__":`

Ćwiczenie 1:

Zmodyfikuj plik `pytest.py` w poniższy sposób i sprawdź różnicę w działaniu.

```
def suma(a,b):
    return a+b

def roznica(a,b):
    return a-b

PI = 3.1415

if __name__=="__main__":
    print(suma(3,5))
```

W przypadku wykonywania pliku zmienna `__name__` przyjmuje wartość `"__main__"`, a w przypadku importu nazwę modułu, dzięki temu z tego samego pliku możemy korzystać zarówno jak ze skryptu i jak z modułu.

Numpy (http://www.numpy.org/)

`Numpy` to podstawowa biblioteka do implementacji operacji i algorytmów numerycznych.

Głównym obiektem biblioteki jest `ndarray` czyli tablica wielowymiarowa, za pomocą której można reprezentować wektory, macierze, tensory itp.

Moduł `numpy` najczęściej imprtowany jest z aliasem `np`.

```
import numpy as np
```

Obiekt `ndarray`.

Tworzenie:

```
a = np.array([1,2])
b = np.array([1.3, 2.5],)
c = np.array([1.3, 2.5], dtype=np.int) #rzutowanie na typ int
d = np.array([1+2j, 3+1j])
e = np.array([[1,2,4],[4,5,7]])
print('a:', a)
print('b:', b)
print('c:', c)
print('d:', d)
print('e:', e)
type(a)

np.ones((1,5))+np.ones((5,1))
```

Podstawowe atrybuty obiektu `ndarray`.

```
print(a.shape) #wymiary tablicy
print(e.shape)
print(b.ndim) #liczba wymiarów
print(e.ndim)
print(c.dtype) #typ danych
print(d.dtype)
print(d.size) #liczba elemetów
print(e.size)
```

Do elementów obiektu `ndarray` odwołujemy się w taki sam sposób jak do elementów listy.

```
print(a[0])
print(e[1])
print(e[1,2])
print(a[1:3])
print(e[:,1])
print(e[0,:])
```

Wybieranie elementów spełniających podany warunek.

```
#a>2
a[a>2]
```

Wybieranie elementów podajac ich indeksy.

```
a[[0,1,0,1]]
```

Przydatne wbudowane metody do tworzenia obiektów `ndarray`.

```
np.zeros(10)

np.zeros((2,3))

np.ones(10)

np.ones((3,4))

np.arange(5)

np.arange(3,15)

np.arange(3,15,0.9)

np.linspace(0,1,10) # 10 równorozłożonych punktów od 0 do 1
```

Arytmetyka

Wszystkie operacje arytmetyczne na obiektach `ndarray` wykonywane są na elementach będących na tych samych pozycjach, a wiec wymiary muszą sie zgadzać.

```
a = np.array([1,2,3,4])
b = np.ones(4)
print('a:',a,'b:',b)
print('a+b:',a+b)
print('a-b:',a-b)
print('a*b:',a*b)
print('a/(2*b):',a/(2*b))
print('a**2:',a**2)
print('a**(b/2):',a**(b/2))
```

W module `numpy` wbudowane są również podstawowe funkcje matematyczne, które jako argument przyjmują obiekt `ndarray`, np.

```
print('sin(a):', np.sin(a))
print('log(a):', np.log(a))
```

Inne przydatne metody:

```
a = np.arange(1,5,0.5)
print('a:',a)
print('suma elementów:', np.sum(a))
print('iloczyn elementów:', np.prod(a))
print('cumsum:', np.cumsum(a))
print('max:', np.max(a)) #analogicznie minimum
print('argmax:', np.argmax(a))
```

W przypadku wiekszej liczby wymiarów możemy wykonywać operacje takie jak np. `sum`, `max`, `min` itp. według zadanego wymiaru `axis`.

```
m = np.array([[1,2,3],[4,5,6],[7,8,9]])
print('m:',m)
print('suma:',np.sum(m))
print('suma w wierszach:',np.sum(m, axis=1))
print('suma w kolumach:',np.sum(m, axis=0))
```

Zmiana wymiarów:

```
a = np.ones(15)
print(a)
print(a.reshape(3,5))
```

Algebra liniowa

```
v = np.array([1,1])
m = np.array([[1,2],[3,4]])
print('wektor v:',v)
print('macierz m:',m)
print('transpozycja wektora:',v.T) #nic nie zmienia
print('transpozycja macierzy:',m.T)
print('wektor * wektor:', np.dot(v,v)) #iloczyn skalarny
print('macierz * wektor:', np.dot(m,v))
print('wektor * macierz:', np.dot(v,m))
```

Ponieważ transpozycja tablicy jednowymiarowaj nic nie zmienia, mnożenie macierzy przez wektor nie zawsze zadziała poprawnie. Aby operacje były zgodne z rachunkiem macierzowym wektor powinnien być zdefiniowany jako macierz o jednym wierszu/kolumnie. Można w tym celu zamiast `np.array` użyć `np.matrix`.

```
v = np.array([[1,1]])
m = np.array([[1,2],[3,4]])
print('wektor v:',v)
print('macierz m:',m)
print('transpozycja wektora:',v.T)
print('transpozycja macierzy:',m.T)
print('wektor razy wektor:', np.dot(v,v.T)) #macierz o jednym
elemencie
```

```
print('macierz razy wektor:', np.dot(m,v.T)) #macierz o jednej
kolumnie
print('wektor razy macierz:', np.dot(v,m)) #macierz o jednym wierszu

np.matrix([1,2,3])

np.matrix([[1,2],[3,4]])
```

Wyznacznik i macierz odwrotna (`np.linalg`)

```
m = np.matrix([[1,2],[3,4]])
print('macierz m:',m)
print('det(m):', np.linalg.det(m))
print('m^(-1):', np.linalg.inv(m))
print('m*m^(-1):', np.dot(m,np.linalg.inv(m)))
```

Scipy (http://www.scipy.org)

`Scipy` to zbudowany na bazie `numpy` moduł zawierający podstawowe algorytmy numeryczne służące m.in do całkowania, optymalizacji, interpolacji itp.

Całkowanie numeryczne

```
from scipy.integrate import quad, trapz

def func(x):
    return x**2

# całka z x^2 na przedziale [0,1]
# metoda kwadratury gaussa
c1 = quad(func, 0, 1) # zwraca wartość całki i oszacowany błąd
print('quad:',c1)

# metoda trapezów
x = np.linspace(0, 1,1000)
y = func(x)
c2 = trapz(y, x) # zwaraca wartość całki
print('trapz:',c2)
```

Optymalizacja

```
from scipy.optimize import minimize

def func(x):
    return np.sum(x**2)  #suma kwadratów

# punkt startowy
x0 = [10, 10.]
# minimalizacja metodą nelder-mead, dostępnych wiele innych algorytmów
res = minimize(func, x0, method='nelder-mead', options={'xtol': 1e-8,
```

```
'disp': True})
print('punkt minimum:', res.x)
print('wartośc minimum:', res.fun)
```

Interpolacja

```
from scipy.interpolate import lagrange

# punkty interpolacji
x = np.linspace(0.0,6.0,10) # 10 równorozłożonych punktów od 0 do 6
y = np.sin(x)
print('(x,y):',list(zip(x,y)))  #zip łączy elementy tablic w pary

# wielomian interpolacyjny Lagrange'a, stopień taki jak liczba punktów
w = lagrange(x, y)
print('w_coef:', w.coef)
```

Matplotlib (http://matplotlib.org/)

Moduł `matplotlib` to bardzo rozbudowana biblioteka do tworzenia najróżniejszych wykresów 2D i 3D (warto zaglądnąć do galerii przykładowych wykresów http://matplotlib.org/gallery.html).

```
# import modułu matplotlib
import matplotlib.pyplot as plt

# rysunki domyślnie powinny wyświetlać się w notatniku
# gdyby tak nie było należy odkomentować poniższa linijkę
#%matplotlib inline
```

Tworzenie wykresu

Szybki wykres dla przykładu interpolacji funkcji sinus wielomianem lagrange'a.

```
plt.figure()
plt.plot(x, y, 'o')
x2 = np.linspace(0.0,6.0,1000)
y2 = w(x2)
plt.plot(x2,y2, 'r')
plt.show()
```

Prosty przykład (jeden wykres na rysunku).

```
# przykładowe dane
x = np.linspace(1, 10, 100)
y = np.sin(x)

# tworzenie rysunku
```

```
fig, ax = plt.subplots() #zwraca obiekt rysunku fig, i obiekt wykresu
na rysunku ax
# rysowanie wykresu
ax.plot(x, y)
# wyświetlenie rysunku
plt.show()
```

Modyfikowanie wyglądu

```
# przykładowe dane
x = np.linspace(1, 10, 100)
y = np.sin(x)

# tworzenie rysunku
fig, ax = plt.subplots()
# rysowanie wykresu
ax.plot(x, y, c='r', marker='+', linestyle='-.', linewidth=5) #wiele
innych opcji patrz w dokumentacji
# wyświetlenie rysunku
plt.show()
```

Dodanie tytułu, opisu osi, siatki i legendy.

```
# przykładowe dane
x = np.linspace(1, 10, 100)
y = np.sin(x)

# tworzenie rysunku
fig, ax = plt.subplots()
# rysowanie wykresu
ax.plot(x, y, c='r', linestyle='-.', label='sin(x)') #wiele innych
opcji patrz w dokumentacji
ax.legend() # dla legendy można ustalić pozycje podając argument
0,1,2,3 lub 4
ax.set_title('Sin(x)')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.grid(True)
# wyświetlenie rysunku
plt.show()
```

Wartości na osiach dobierane są automatycznie jednak możemy to zmodyfikować.

```
# przykładowe dane
x = np.linspace(1, 10, 100)
y = np.sin(x)

# tworzenie rysunku
fig, ax = plt.subplots()
```

```
# rysowanie wykresu
ax.plot(x, y, c='r', linestyle='-.', label='sin(x)') #wiele innych
opcji patrz w dokumentacji
ax.legend() # dla legendy mozna ustalic pozycje podając argument
0,1,2,3 lub 4
ax.set_title('Sin(x)')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_xlim(0,11)
ax.set_ylim(-1.1,1.1)
ax.grid(True)
# wyświetlenie rysunku
plt.show()
```

Wiele wykresów na jednym rysunku.

```
# przykładowe dane
x1 = np.linspace(1, 10, 100)
y1 = np.sin(x)
x2 = np.linspace(1, 10, 10)
y2 = np.sin(x2)

# tworzenie rysunku
fig, (ax1, ax2) = plt.subplots(nrows=2, ncols=1) # dwa wykresy jeden
pod drugim
# rysowanie wykresu nr 1
ax1.plot(x1, y1, c='r', linestyle='-.', label='sin(x)') #wiele innych
opcji patrz w dokumentacji
ax1.legend() # dla legendy mozna ustalic pozycje podając argument
0,1,2,3 lub 4
ax1.set_title('Sin(x) nr 1')
ax1.set_xlabel('x')
ax1.set_ylabel('y')
ax1.grid(True)
# rysowanie wykresu nr 2
ax2.scatter(x2, y2, color='r', label='sin(x)') #wiele innych opcji
patrz w dokumentacji
ax2.legend() # dla legendy mozna ustalic pozycje podając argument
0,1,2,3 lub 4
ax2.set_title('Sin(x) nr 2')
ax2.set_xlabel('x')
ax2.set_ylabel('y')
ax2.grid(True)

# wyświetlenie rysunku
plt.show()
```

Histogram.

```
# Aby mieć polskie znaki w tekstach na wykresie należy zmienic
czcionkę
plt.rcParams['font.family'] = 'DejaVu Sans'

# przykładowe dane
x = np.random.randn(100) # sto punktów ze standardowego rozkładu
normalnego

# tworzenie rysunku
fig, (ax1, ax2) = plt.subplots(nrows=2, ncols=1) # dwa wykresy jeden
pod drugim
# rysowanie wykresu nr 1
ax1.scatter(range(len(x)), x, color='r')
ax1.set_title('Random')
ax1.set_xlabel('nr')
ax1.set_ylabel('y')
ax1.grid(True)
# rysowanie wykresu nr 2
ax2.hist(x, 20) #wiele innych opcji patrz w dokumentacji
ax2.set_xlabel('x')
ax2.set_ylabel(u'częstość')
ax2.grid(True)

# wyświetlenie rysunku
plt.show()
```

Wykres 3D.

```
from mpl_toolkits.mplot3d import Axes3D

x = np.random.randn(100)
y = np.random.randn(100)
z = np.random.randn(100)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x, y, z, c='r', marker='o')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

plt.show()
```

Fajny przykład z galerii.

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
from matplotlib import cm

fig = plt.figure()
```

```
ax = fig.gca(projection='3d')
X, Y, Z = axes3d.get_test_data(0.05)
ax.plot_surface(X, Y, Z, rstride=8, cstride=8, alpha=0.3)
cset = ax.contour(X, Y, Z, zdir='z', offset=-100, cmap=cm.coolwarm)
cset = ax.contour(X, Y, Z, zdir='x', offset=-40, cmap=cm.coolwarm)
cset = ax.contour(X, Y, Z, zdir='y', offset=40, cmap=cm.coolwarm)

ax.set_xlabel('X')
ax.set_xlim(-40, 40)
ax.set_ylabel('Y')
ax.set_ylim(-40, 40)
ax.set_zlabel('Z')
ax.set_zlim(-100, 100)

plt.show()
```

Pandas (http://pandas.pydata.org/)

`Pandas` to moduł do szybkiego i wygodnego przetwarzania i analizowania danych w postaci tabel.

```
# import modułu pandas
import pandas as pd
```

DataFrame

Podstawowym obiektem modułu `pandas` jest `DataFrame`.

```
# stworzenie obiektu dataframe
df = pd.DataFrame([[1,2,3],[1,2,3]], columns=['A','B','C'])

df

df.values
```

Obiekt `DataFrame` możemy stworzyć bezpośrednio wczytując dane z pliku np. txt, csv, xls itp.

Wykorzystywany w poniższych komórkach plik `titanic.xls` można wgrać bezpośredniu środowiska (Zakładka `Files`) lub na swój Google Drive i go połączyć ze śodowiskiem, jak poniżej:

```
import sys
from google.colab import drive
drive.mount('/content/drive')
# upewniej się że poniższa ścieżka jest poprawna
path_nb = r'/content/drive/My Drive/Colab
Notebooks/WdSI_2021/T2_Lab_Python/'
sys.path.append(path_nb)
```

```
# wczytanie danych z pliku xls
df_titanic = pd.read_excel(path_nb+'titanic.xls')

df_titanic

# wyświetlenie pierwszych n wierszy
df_titanic.head(6)

# wyświetlenie ostatnich n wierszy
df_titanic.tail(6)

# wyświetlenie podstawowych statystyk
df_titanic.describe().T  #transpozycja dla lepszej czytelności
# wyniki tylko dla kolumn zawierających dane liczbowe

# odwołanie do konkretnej kolumny
df_titanic['age'][:10]  # pierwsze 10

# to samo uzyskamy pisząc
df_titanic.age[:10]

# lista nazw kolumn
df_titanic.columns
```

Brakujące dane w komórkach zastępowane są poprzez obiekt `NaN`.

```
# policzenie liczby brakujących wartości w poszczególnych kolumnach
df_titanic.apply(lambda x: sum(x.isnull()),axis=0)

# brakujące dane możemy zastąpić ustaloną wartością np. -1
df_titanic.fillna(-1, inplace=True)
df_titanic.apply(lambda x: sum(x.isnull()),axis=0)

# wybieranie kilku kolumn
df_titanic2 = df_titanic[['name','age']]
df_titanic2.head()

# usuwanie kolumn
df_titanic = df_titanic.drop(['name','cabin','boat', 'body', 'ticket',
'home.dest'], 1)
df_titanic.head()

# unikalne wartości w kolumnach
for col in df_titanic.columns:
    print(col, df_titanic[col].unique())
```

Rysowanie wykresów bezpośednio z tabel.

```
import matplotlib.pyplot as plt
# rozkład wieku
```

```
df_titanic['age'].hist()  # pik w -1 to nasze brakujące dane
plt.show()

# bar plot z dwóch kolumn
ct = pd.crosstab(df_titanic['pclass'], df_titanic['survived'])
ct.plot(kind='bar')
plt.show()
```

Na danych w postaci obiektów `DataFrame` można wykonywać bardzo dużo przeróżnych operacji jak również pracować z nimi jak z tabelami w bazach danych (zapytania SQL). Więcej w dokumentacji.

Przedstawione wyżej przykłady to tylko bardzo niewielki przykład możliwości modułów numpy, scipy, matplotlib oraz pandas. Zachęcam do poczytania (przeglądnięcia) dokumentacji tych modułów

Zadanie 1:

Napisz funkcję wyliczającą macierz odległości (euklidesowych) dla podanego zbioru wektorów. Funkcja powinna działać dla dowolnej liczby wektorów oraz dowolnego wymiaru tych wektorów.

Macierz odległości $M = \left[m_{ij} \right]$, gdzie m_{ij} to odległość Euklidesa między i-tym a j-tym wektorem.

Funkcję tą napisz w dwóch wersjach. Jedną z użyciem pakietu `numpy`, a drugą bez (czysty Python). Porównaj czasy ich działania dla losowych danych wejściowych. Do pomiaru czasu można wykorzystać moduł `time` lub dostępną w notatniku metodę magiczną `%%timeit` (http://ipython.readthedocs.io/en/stable/interactive/magics.html).

Dla wersji z użyciem pakietu `numpy` spróbuj napisać tą funkcję tak aby uniknąć wykorzystywania pętli `for` (dodatkowe punkty).

```
import numpy as np
import random
import time

# Dane wejściowe:
m = 1000  # liczba wektorów
n = 300  # wymiar wektora
wektory = [[random.random() for _ in range(n)] for _ in range(m)]

### YOUR CODE HERE
```

Zadanie 2:

Korzystając z pakietu `scipy.optimize` znajdź minimum poniższej funkcji

$$F(x, y) = \frac{1}{f}\left(-a \exp\left[-b \sqrt{\frac{1}{n}\left(x^2 + y^2\right)} \right] - \exp\left[\frac{1}{n}\left(\cos(c\,x) + \cos(c\,y)\right) \right] + a + \exp(1) + d \right)$$

dla $a=20, b=0.2, c=2\pi, d=5.7, f=0.8, n=2$ oraz przy ograniczniach $x, y \in [-1.5, 1.5)$.

Wypróbuj różne punkty startowe, czy zawsze dostajesz ten sam wyniki?, dalczego?.

```
import numpy as np
from scipy.optimize import minimize

### YOUR CODE HERE
```

Zadanie 3:

Korzystając z pakietu `matplotlib` narysuj wykres funkcji z zadania 2.

```
import numpy as np
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt

### YOUR CODE HERE
```