Wstęp do Data Mining

http://www.fizyka.umk.pl/~piotra/dydaktyka/WDAM

Piotr Ablewski

Katedra Informatyki Stosowanej
Instytut Nauk Technicznych
Wydział Fizyki Astronomii i Informatyki Stosowanej
Uniwersytet Mikołaja Kopernika w Toruniu

mgr inż. Piotr Ablewski

www.fizyka.umk.pl/~piotra

e-mail: **piotra@fizyka.umk.pl** lub **piotr@ablewski.pl**
Discord: **https://discord.gg/84yCUWz** ← **wstęp_do_data_mining**
Telegram: **https://t.me/PiotrA_KIS_UMK**

Konsultacje: **środa 12:00 - 14:00** lub w **dowolnym umówionym mailowo terminie** - jestem dostępny od poniedziałku do piątku od 12:00 do 20:00, za wyjątkiem czasu, gdy prowadzę zajęcia.

Gdzie mnie szukać: **pokój 571** (naprzeciwko wejścia do PK1 i PK2)

Zasady zaliczenia przedmiotu

- **Obowiązkowa obecność na laboratoriach** - dopuszczalna jedna nieobecność nieusprawiedliwiona
- Na ostateczną ocenę składa się:
  - **Kolokwium - 30%** wkładu do oceny [termin zostanie ustalony]
  - **Praca na zajęciach - 70%** wkładu do oceny [bieżąca praca polegająca na realizacji zadań]
- W ramach zajęć można otrzymać łącznie max. **100 pkt**., z czego **50 pkt. niezbędnych jest do otrzymania zaliczenia**

Jak wyglądać będą zajęcia?

20h/semestr

10 spotkań po 2h (20h)
lub
4 spotkania po 1h (4h) + 8 spotkań po 2h (16h)

Struktura zajęć:

- ok. 30 min - wykład
- ok. 30 min - praca wspólna przed klawiaturą
- ok. 30 min - praca samodzielna nad zadaniami


100100 1
1100100 10

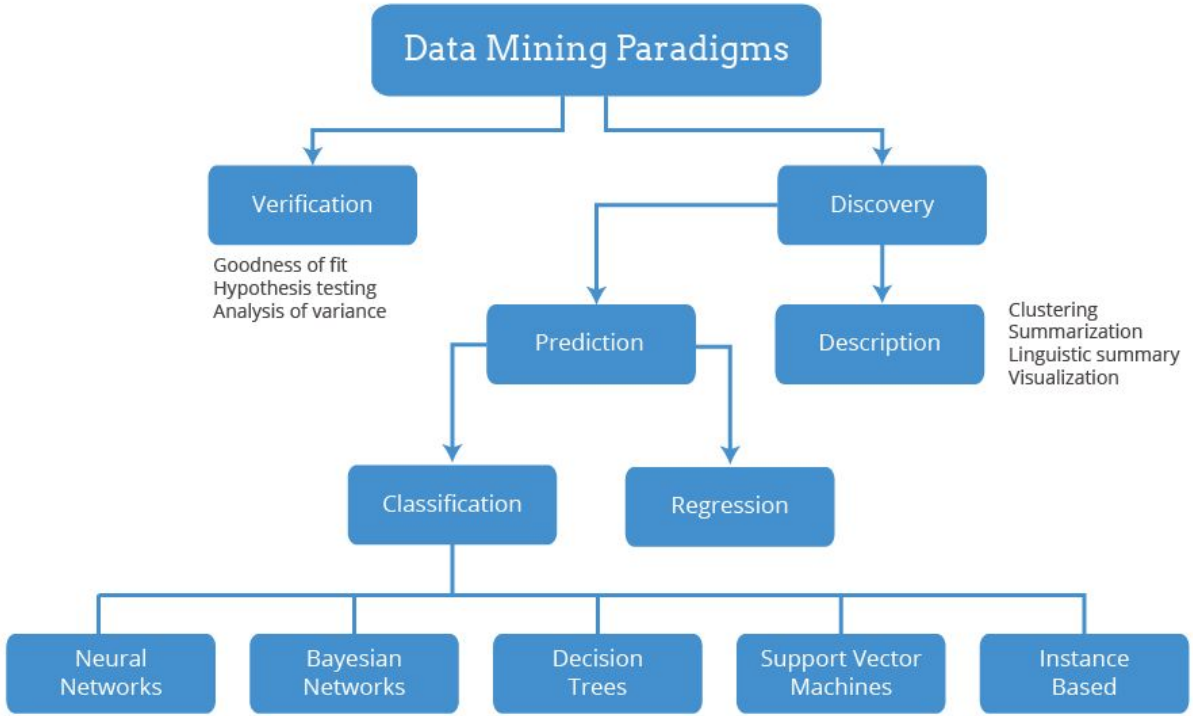
Data Mining

Data mining, czyli eksploracja danych, jest technologią, służącą do zautomatyzowanego odkrywania statystycznych zależności i schematów w bardzo dużych bazach danych. Zależności i schematy te, przedstawiane są najczęściej w formie reguł logicznych, drzew decyzyjnych lub sieci neuronowych.

Metody eksploracji danych

1) klasyfikacja
2) klasteryzacja
3) regresja
4) znajdowanie wzorców
5) predykcja
6) reguły asocjacyjne
7) aproksymacja
8) ...

Metody eksploracji danych - inna klasyfikacja



Wykorzystywane narzędzia

1) **R**
2) **Python**
3) C++
4) Java
5) Matlab
6) SQL
7) SAS / SPSS
8) frameworki do pracy ze strukturami Big Data: **Hadoop**, **Storm**, Samza, Spark, Flink

Wykorzystywana wiedza

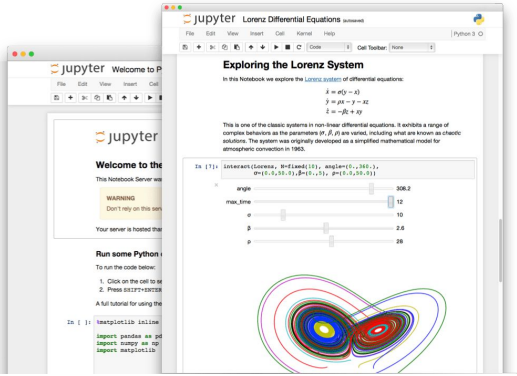
1) Linux
2) bazy danych (relacyjne i nierelacyjne)
3) statystyka, algebra liniowa
4) algorytmy i struktury danych
5) i wiele, wiele więcej...

Z czego korzystać będziemy na zajęciach

- **Python 3.7**
  - numpy
  - pandas
  - matplotlib

Polecam zainstalować **Anaconda Distribution** (https://www.anaconda.com/distribution/)

- **Jupyter Notebook** lub **JupyterLab**



- **GitHub**

Dlaczego DM jest ważny dla biznesu?

- **zbieranie danych** == profilowanie działalności i dostosowywanie się do potrzeb klienta lub dostosowywanie potrzeb klienta do możliwości oferowanych przez firmy
    - ekosytem Google, Microsoft, Apple
    - Yahoo & Google - petabajty danych na temat sieci WWW
    - Facebook - dane o użytkownikach
    - banki - kontrola transakcji

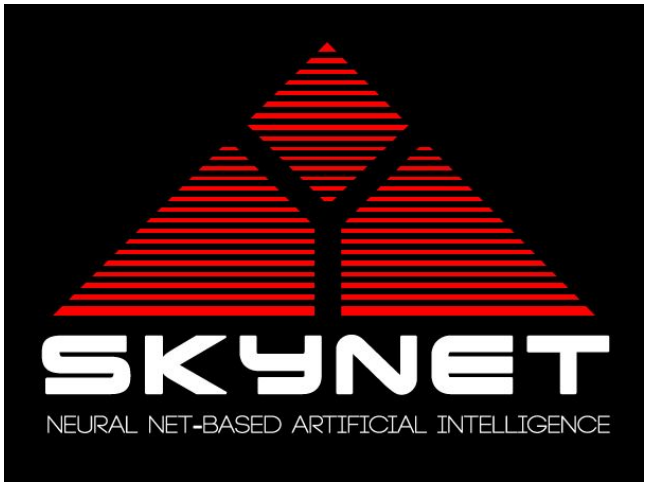
- **komputery są coraz tańsze (moc obliczeniowa)-** dlaczego tego nie wykorzystać i nie zarobić jeszcze więcej

Dlaczego DM jest ważny dla nauki?

- **zbieranie i analiza danych** z eksperymentów
    - NASA
    - CERN
    - dane medyczne
    - symulacje inżynierskie i naukowe
- **automatyzacja** analizy dużych zbiorów danych, które nie zawsze mają w sobie cenne informacje oraz pomoc w **formułowaniu hipotez**

Dlaczego DM jest ważny dla społeczeństwa?

- **zbieranie danych** o obywatelach
    - poprawa bezpieczeństwa
    - Wielki Brat patrzy - utrata prywatności
- **cyberbezpieczeństwo**
- **IoT**
- **E-Commerce**
- **wszechobecna OPTYMALIZACJA**



Zajęcia 2 (1h)

Przygotowanie środowiska pracy

Linux + Python + Jupyter Notebook

- **Linux:**
  - python 3.7 (z repozytorium dystrybucji lub Anaconda)
  - Jupyter Notebook (conda lub pip)
- **Windows:**
  - można zainstalować pythona 3.7 (Anaconda) i Jupyter Notebook natywnie
  - Hyper-V lub VirtualBox - maszyna wirtualna
  - WSL2 (Windows 10 build 18917+)

Instalacja WSL 2

- **Windows Subsystem for Linux**

```
dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart
```

- **Virtual Machine Platform**

```
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart
```

Po wydaniu tych komend w konsoli systemu lub w PowerShell'u i zainstalowaniu komponentów, restartujemy maszynę

Instalacja WSL 2 - wybór dystrybucji

- **WSL 2 jako domyślna architektura**

```
wsl --set-default-version 2
```

- **Wybór dystrybucji**

```
wsl -l
wsl --set-version <Distro> 2
```

Pierwsza komenda wyświetla dostępne dystrybucje, a druga pozwala ustawić dystrybucję, z której będziemy korzystali

Instalacja WSL 2 - wybór dystrybucji

- **WSL 2 jako domyślna architektura**

```
wsl --set-default-version 2
```

- **Wybór dystrybucji**

```
wsl -l
wsl --set-version <Distro> 2
```

- **Uruchomienie dystrybucji**

```
wsl <Distro>
lub poprzez ikonę (np. Ubuntu.exe)
```

W razie problemów

Jeśli w trakcie instalacji pojawią się problemy - proszę kontaktować się ze mną mailowo lub przez komunikator Discord.

Można również samodzielnie spróbować rozwiązać problem, szukając pomocy na stronach Microsoft

https://docs.microsoft.com/en-us/windows/wsl/wsl2-index

Anaconda

Ze strony: https://www.anaconda.com/distribution/#download-section
pobieramy instalator w wersji Python 3.7

Proces instalacji pokazany zostanie po prezentacji

Jupyter Notebook

Instalacja Jupyter Notebook poprzez pip:

```
pip install jupyterlab
```

Uruchomienie zeszytu na domyślnym porcie:

```
jupyter notebook
```

Korzystanie z Jupyter Notebook’a - przez przeglądarkę

GitHub

Każdy z Państwa ma utworzone swoje własne repozytorium (zarządzanie):

IS-UMK/WDDM2020-PiotrA-XXXXXX,

gdzie XXXXXX to numer indeksu.

Repozytorium kodów z zajęć (tylko czytanie):

IS-UMK/WDDM2020-PiotrA

Na potrzeby zajęć - wystarczy nam korzystanie z GitHub'a przez przeglądarkę, ale zainteresowanych pracą z GitHub'em jako ze zdanlym repozytorium przez konsolę odsyłam do tutoriala: https://rogerdudler.github.io/git-guide/index.pl.html

Zajęcia 3 (1h)

Wstęp do języka Python

Dlaczego Python?

- wygodne narzędzie do analizy danych
- łatwe operacje na tablicach
- dynamiczne typowanie
- mnogość bibliotek

Co Cię będzie początkowo irytować? Brak średnika na końcu linii i brak klamer { } - ich funkcję pełnią wcięcia, których poziomy muszą się zgadzać.

Zmienne

```
nazwa = wartość
```

Python wykorzystuje dynamiczne typowanie - typy ustalane są przez interpreter na podstawie wartości zmiennych. Możliwe proste typy danych:

- liczbowe:
  - integer (x = 2)
  - long integer (x = 10L)
  - floating point (x = 3.14) - tak na prawdę to typ double
- logiczny:
  - boolean (x = True lub y = False)
- tekstowe:
  - character (x = ‘c’)
  - string (x = ‘lancuch’ lub x = “lancuch”)

Programowanie obiektowe w Pythonie

```
class Klasa
    pole1 = 3.14
    pole2 = "BlahBlah"

    def __init__(self):
        print("Hello world z konstruktora")
        self.pole3 = -1
        self._pole4 = 0
        self.__pole5 = 1

    def metoda(self):
        print("Hello world")


obiekt = Klasa()
```

listy i słowniki

```
lista = [1, 2, 3, 4]

lista2 = []
lista2[0] = 1
lista2[1] = 2
…
lista2.append(88)


słownik = { klucz1: “wartosc2”, klucz2: “wartosc2” }

slownik2 = {}
slownik2[‘k1’] = “war1”
slownik2[‘k2’] = “war2”
```

Więcej przykładów w części praktycznej - po kod zapraszam na GitHub’a

Instrukcja warunkowa IF

```
if WARUNEK_LOGICZNY :
    # kod, który wykona się, gdy WARUNEK_LOGICZNY będzie spełniony
else:
    # kod, który wykona się, gdu WARUNEK_LOGICZNY nie będzie spełniony
```

```
if WARUNEK_LOGICZNY_1:
    # kod, który wykona się, gdy WARUNEK_LOGICZNY_1 będzie spełniony
elif WARUNEK_LOGICZNY_2:
    # kod, który wykona się, gdy WARUNEK_LOGICZNY_2 będzie spełniony, a
    nie był spełniony WARUNEK_LOGICZNY_1
else:
    # kod, który wykona się, gdy ani WARUNEK_LOGICZNY_1 ani
    WARUNEK_LOGICZNY_2 nie będzie spełniony
```

Warunki logiczne

== - porównanie wartości zmiennych
!= - porównanie wartości zmiennych z zaprzeczeniem
>, <, >=, <= – numeryczne porównanie wartości zmiennych

operator **not** - zmiana wartości logicznej wyrażenia na przeciwną

operator **in** - można sprawdzić, czy konkretny obiekt znajduje się w innym obiekcie (np. liście lub słowniku)

operator **is** - sprawdza, czy zmienne wskazują na ten sam obszar pamięci

Puste zmienne:

- numeryczna: x
- tekstowa: “”
- obiekt: None
- lista: []
- logiczna: False

Pętla FOR

```
for zmienna in parametry:
    # wykonaj operację na każdej zmiennej zmienna, należącej do listy
    parametrów w parametry
```

można również uprościć sobie życie:

```
wynik = [ len(klucz) for klucz in lista ]
```

W taki sposób można wywołać funkcję len() dla każdego elementu na liście `lista`!

Dla słowników (np. wybór wszystkich kluczy ze słownika `slownik`) :

```
wynik = [ klucz for (klucz, wartosc) in slownik.items() ]
```

Pętla while

```
while  WARUNEK_LOGICZNY:
    # kod, który wykonuje się, gdy WARUNEK_LOGICZNY jest spełniony
```

```
while WARUNEK_LOGICZNY:
    #kod, który wykonuje się, gdy WARUNEK_LOGICZNY jest spełniony
else:
    #kod, który wykonuje się, gdy WARUNEK_LOGICZNY nie jest spełniony
```

break i continue

`break` – przerywa wykonanie instrukcji for lub while, nawet gdy warunek jej iteracji jest wciąż spełniony

```
for x in [1, 2, 3, 4, …., 10] :
    print(x)
    if x > 4:
        break
```

`continue` – przerywa wykonanie bieżącej iteracji instrukcji, przechodząc do następnej iteracji tej samej instrukcji

```
for x in [1, 2, 3, 4, …., 10] :
    if x % 2 == 0:
        break
    print(x)
```

Definicja funkcji

```
def nazwa_funckcji( lista_argumentow )
    #kod funkcji
    return wartosc_zwracana
```

Albo poprzez funkcje nienazwane (lambdy, funkcje anonimowe), np.:

```
square = lambda x: x*x
cztery = square(2)


cube = lambda x: x*x*x
osiem = cube(2)
```

Wejście/wyjście z pliku

```
f = open( nazwa_pliku, uprawnienia)
# operacje na danych w pliku
f.close()
```

Uprawnienia:

r - czytanie; **w** - pisanie; **a** - dopisanie na koniec pliku; **x** - utworzenie nowego pliku, jeśli nie istniał; **t** - tryb tekstowy; **b** - tryb binarny; **+** - plik można odczytywać i zapisywać (czyli aktualizować)

Operacje:

```
f.write(dane)              # zapis do pliku
zawartosc = f.read()       # odczyt całego pliku
dane = f.read(n)           # odczyt n znaków z pliku
dane = f.readlines()       # odczyt wszystkich linii z pliku
dane = f.readline()        # odczyt jednej linii z pliku
```

Wejście/wyjście z pliku

Jednorazowy zapis do pliku:

```
with open(nazwa_pliku, 'w') as f:
    # tutaj mamy dostęp do otwartego pliku pod zmienną f
#a tutaj już tego dostępu nie ma
```

Jednorazowy odczyt z pliku:

```
with open(nazwa_pliku, 'r') as f:
    for line in f:
        # w zmiennej line mamy dostęp do pojedynczej linii z pliku
# a tutaj już nie ma dostępu
```

Import biblioteki - na przykładzie Math

```
import math
x = 4
print ( math.sqrt(x) )
```

```
import math as mat
x = 4
print( mat.sqrt(x) )
```

```
from math import *
x = 4
print ( sqrt(x) )
```

Operacje na łańcuchach znaków

Python posiada rozbudowany zestaw funkcji do operowania na łańcuchach znaków, czyli na zmiennych typu string. Ich praktyczne wykorzystanie pokazane zostanie w zeszycie Jupyter Notebook - na konkretnym przykładzie.

Zajęcia 4 (1h)

Podstawy Numpy, Pandas i Matplotlib