Машинно обучение

Лабораторно упражнение №2

Въведение в Python

1. **Символни низове. -** могат да са затворени в единични или двойни кавички

Примери:

```
>>> 'hello'
'hello'
>>> "hello"
'hello'
```

1.1. Вмъкване на нов ред. - \n

```
>>> print("hello\n")
hello
```

1.2. Вмъкване на разстояние между думите. - \ t

```
>>> print("hello\t", "world")
hello     world
```

1.3. Конкатенация. – чрез знака +

```
>>> ' a'+'b'
 ' ab'
>>> "a"+"b"
'ab'
>>> 'a'+"b"
'ab'
```

1.4. Индексиране на символни низове. – първия индекс е 0

```
>>> x='ab'+'cd'
>>> x
'abcd'
>>> x[0]
'a'
>>> x[1:3] # извежда символи от индекс 1 до индекс 2
```

```
'bc'
>>> x[-1] # извежда последния символ
'd'
>>> x[-2] # извежда предпоследния символ
'c'
>>> x[:3] # извежда първите три символа
'abc'
>>> x[3:] # извежда всичко, освен първите три символа
'd'
>>> x[5:] # извежда празен символен низ, защото индекса е твърде голям
' '
```

1.5. Символни низове. Методи.

1.5.1. Дължина на низ

```
>>> len(x) # извежда дължина на низа
4
```

1.5.2. Брой символи

```
>>> x.count('a')
1
```

1.5.3. Преобразуване на символния низ в големи букви

```
>>> print(x.upper())
ABCD
```

1.5.4. Преобразуване на символния низ в малки букви

```
>>> print(x.lower())
abcd
```

1.5.5. Разделяне на символния низ на отделни думи

```
>>> mytext="Hello world!"
>>> mytext.split()
['Hello', 'world!']
```

1.5.6. Замяна на подниз в даден символния низ с друг подниз

```
>>> mytext=mytext.replace("Hello","Hi")
```

```
>>> mytext
'Hi world!'
```

1.5.7. Сортиране на списък от низове

1.5.7.1. Сортиране на списък от низове чрез sort()

```
>>> x=['x','abc','acb']
>>> x.sort()
>>> x
['abc', 'acb', 'x']
```

1.5.7.2. Сортиране на списък от низове чрез sorted в нарастващ ред

```
>>> x=['x','abc','acb']
>>> sorted(x)
['abc', 'acb', 'x']
```

Но:

```
>>> x
['x', 'abc', 'acb']
```

1.5.7.3. Сортиране на списък от низове чрез sorted в намаляващ ред

```
>>> sorted(x,reverse=True)
['x', 'acb', 'abc']
```

1.5.7.4. Сортиране на списък от низове чрез sorted в зависимост от дължината на низа

```
>>> x=['ac','acb','x']
>>> sorted(x,key=len)
['x', 'ac', 'acb']
>>> sorted(x,key=len,reverse=True)
['acb', 'ac', 'x']
```

1.5.7.5. Сортиране на низ чрез sorted

```
>>> sorted('python')
['h', 'n', 'o', 'p', 't', 'y']
```

За повече информация: напишете в командния интерпретатор: help(). След появата на промпта: help> напишете STRINGMETHODS.

2. Библиотека NumPy

2.1. Основни понятия

NumPy е основната библиотека за реализация на изчисления в Python. Предоставя следните най-важни функционалности:

- Обработване на N-мерни масиви.
- Бързо изпълнение на математически операции с многомерни масиви
- Средства за реализация на операции от линейна алгебра, генериране на числени стойности и др.

Основен обект в NumPy са многомерните масиви. Многомерният масив представлява таблица от елементи (най-често числени стойности) от еднакъв тип, индексирани с положителни цели числа. Размерностите на масивите в NumPy се наричат оси (axes).

Например координатите на точка в 3D пространство [1, 2, 1] се представят като масив с една ос. Тази ос има 3 елемента, т.е. дължина 3.

Класът за реализация на масив в NumPy се нарича ndarray. Важно е да се отбележи, че numpy.array се различава от стандартния клас array.array в Pthon, който обработва едномерни масиви и предлага по-малко функционалности. Най-важните атрибути на ndarray обект са:

ndarray.ndim – брой оси (размерности) на масива

ndarray.shape – връща размерностите на масива. Резултатът представлява комплект (tuple) от целочислени стойности, показващи всяка размерност на масива. За матрица с *n* реда и *m* колони shape връща двойка (n,m).

ndarray.size – общ брой на елементите в масива равен на произведението на броя елементи във всяка размерност

ndarray.dtype – този обект описва типовете на елементите в масива. Освен стандартните типове данни в Python NumPy предлага собствени типове numpy.int32, numpy.int16, numpy.float64 и др.

За създаване на последователности от числени стойности NumPy предоставя функцията *arange*.

```
>>> np.arange( 10, 30, 5 )
array([10, 15, 20, 25])
>>> np.arange( 0, 2, 0.3 )                 # it accepts float arguments
array([ 0. ,  0.3,  0.6,  0.9,  1.2,  1.5,  1.8])
```

Пример:

```
>>> import numpy as np
>>> a = np.arange(15).reshape(3, 5)
```

```
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape
(3, 5)
>>> a.ndim
2
>>> a.dtype.name
'int64'
>>> a.itemsize
8
>>> a.size
15
>>> type(a)
<type 'numpy.ndarray'>
>>> b = np.array([6, 7, 8])
>>> b
array([6, 7, 8])
>>> type(b)
<type 'numpy.ndarray'>
```

2.2. Създаване на масиви

Има различни начини за създаване на масиви.

Например масив може да се създаде чрез стандартен списък (list) или комплект (tuple) с използване на функцията array.

Пример:

```
>>> import numpy as np
>>> a = np.array([2,3,4])
>>> a
array([2, 3, 4])
```

```
>>> a.dtype
dtype('int64')
>>> b = np.array([1.2, 3.5, 5.1])
>>> b.dtype
dtype('float64')
```

Методът array преобразува масив с две размерности в двумерен масив, масив с три размерности в тримерен и т.н.

```
>>> b = np.array([(1.5,2,3), (4,5,6)])
>>> b
array([[ 1.5,  2. ,  3. ],
       [ 4. ,  5. ,  6. ]])
```

Типът на елементите на масива може да се зададе при създаването:

```
>>> c = np.array( [ [1,2], [3,4] ], dtype=complex )
>>> c
array([[ 1.+0.j,  2.+0.j],
       [ 3.+0.j,  4.+0.j]])
```

Често се налага създаване на масив с неизвестни елементи, но с известни размерности. NumPy предлага функции за създаване на масиви, чиито елементи са инициализирани автоматично. По този начин се премахва необходимостта от обработване на динамично променящи се размерности, което е бавна операция.

Функцията *zeros* създава масив от нулеви стойности, функцията *ones* създава масив със стойности едно, а функцията *empty* създава масив от случайни числени стойности, чиито тип по подразбиране е float64.

```
>>> np.zeros( (3,4) )
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])
>>> np.ones( (2,3,4), dtype=np.int16 )   # dtype can also be specified
array([[[ 1, 1, 1, 1],
        [ 1, 1, 1, 1],
```

```
        [ 1, 1, 1, 1]],
       [[ 1, 1, 1, 1],
        [ 1, 1, 1, 1],
        [ 1, 1, 1, 1]]], dtype=int16)
```

```
>>> np.empty( (2,3) )            # uninitialized, output may vary
```

При използване на *aranage* с реални числа не е възможно да се определи броят на генерираните елементи, който зависи от точността на представяне на числата. По тази причина за генериране на реални числа се използва функцията *linspace*, приемаща като аргумент броя на елементите:

```
>>> from numpy import pi
>>> np.linspace( 0, 2, 9 )                 # 9 numbers from 0 to 2
array([ 0.  ,  0.25,  0.5 ,  0.75,  1.  ,  1.25,  1.5 ,  1.75,  2.
])
>>> x = np.linspace( 0, 2*pi, 100 )          # useful to evaluate
function at lots of points
>>> f = np.sin(x)
```

2.3. Отпечатване на масиви

NumPy отпечатва масиви с *print*. Едномерен масив се изобразява като ред, двумерен като матрица, а многомерен като списък от матрици.

```
>>> a = np.arange(6)                         # 1d array
>>> print(a)
[0 1 2 3 4 5]
>>>
>>> b = np.arange(12).reshape(4,3)           # 2d array
>>> print(b)
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
>>> c = np.arange(24).reshape(2,3,4)         # 3d array
>>> print(c)
[[[ 0  1  2  3]
```

```
  [ 4  5  6  7]
  [ 8  9 10 11]]
 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

2.3. Основни операции с масиви

Аритметичните операции с масиви се изпълняват поелементно (elementwise).

```
>> a = np.array( [20,30,40,50] )
>>> b = np.arange( 4 )
>>> b
array([0, 1, 2, 3])
>>> c = a-b
>>> c
array([20, 29, 38, 47])
>>> b**2
array([0, 1, 4, 9])
>>> 10*np.sin(a)
array([ 9.12945251, -9.88031624,  7.4511316 , -2.62374854])
>>> a<35
array([ True, True, False, False])
```

За разлика от други езици операторът за умножение * обработва NumPy масиви поелементно. За произведение на матрици се използва оператор @ или функцията *dot*.

```
>>> A = np.array( [[1,1], [0,1]] )
>>> B = np.array( [[2,0],[3,4]] )
>>> A * B                        # elementwise product
array([[2, 0],
```

```
       [0, 4]])
>>> A @ B                          # matrix product - добавен в Python 3.5
array([[5, 4],
       [3, 4]])
>>> A.dot(B)                       # another matrix product
array([[5, 4],
       [3, 4]])
```

Някои операции като += и *= се използват за модифициране на съществуващ масив вместо за създаване на нов.

```
>>> a = np.ones((2,3), dtype=int)
>>> b = np.random.random((2,3))
>>> a *= 3
>>> a
array([[3, 3, 3],
       [3, 3, 3]])
>>> b += a
>>> b
array([[ 3.417022  ,  3.72032449,  3.00011437],
       [ 3.30233257,  3.14675589,  3.09233859]])
>>> a += b                         # b is not automatically converted to integer type
Traceback (most recent call last):
  ...
TypeError: Cannot cast ufunc add output from dtype('float64') to dtype('int64') with casting rule 'same_kind'
```

При обработване на масиви от различен тип типът на резултантния масив съответства на масива с по-голяма точност (upcasting).

```
>>> a = np.ones(3, dtype=np.int32)
```

```
>>> b = np.linspace(0,pi,3)
>>> b.dtype.name
'float64'
>>> c = a+b
>>> c
array([ 1.        ,  2.57079633,  4.14159265])
>>> c.dtype.name
'float64'
>>> d = np.exp(c*1j)
>>> d
array([ 0.54030231+0.84147098j, -0.84147098+0.54030231j,
       -0.54030231-0.84147098j])
>>> d.dtype.name
'complex128'
```

Много унарни операции като изчисляване на сума на елементите са имплементирани като методи на класа *ndarray.*

```
>>> a = np.random.random((2,3))
>>> a
array([[ 0.18626021,  0.34556073,  0.39676747],
       [ 0.53881673,  0.41919451,  0.6852195 ]])
>>> a.sum()
2.5718191614547998
>>> a.min()
0.1862602113776709
>>> a.max()
0.6852195003967595
```

По подразбиране тези операции се изпълняват за всяка ос на масива. Възможно е да се зададе ос, за която да се изпълнява операцията:

```
>>> b = np.arange(12).reshape(3,4)
```

```
>>> b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> b.sum(axis=0)                          # sum of each column
array([12, 15, 18, 21])
>>> b.min(axis=1)                          # min of each row
array([0, 4, 8])
>>> b.cumsum(axis=1)                        # cumulative sum along each row
array([[ 0,  1,  3,  6],
       [ 4,  9, 15, 22],
       [ 8, 17, 27, 38]])
```

2.4. Универсални функции

NumPy предоставя често използвани математически функции като sin, cos, exp. В NumPy тези функции се наричат универсални (ufunc). Те се изпълняват поелементно върху масив и връщат в резултат нов масив.

```
>>> B = np.arange(3)
>>> B
array([0, 1, 2])
>>> np.exp(B)
array([ 1.        ,  2.71828183,  7.3890561 ])
>>> np.sqrt(B)
array([ 0.        ,  1.        ,  1.41421356])
>>> C = np.array([2., -1., 4.])
>>> np.add(B, C)
array([ 2.,  0.,  6.])
```

Повече информация за NumPy:

https://docs.scipy.org/doc/numpy/user/quickstart.html

3. **Стандартен модул NLTK - NLTK (Natural Language Toolkit)**е open source библиотека на Python, предоставящи безплатни онлайн книги. За повече информация въведете в командния интерпретатор >>> help(nltk) и/или посетете сайта http://www.nltk.org/. За да инсталирате данните въведете в командния интерпретатор nltk.download().

```
>>> import nltk
>>> mytext='Hello, world!'
>>> nltk.word_tokenize(mytext) # разделя текста на отделни "токъни"
['Hello', ',', 'world', '!']
```

3.1. Работа със стандартния модул NLTK - nltk.book

```
>>> from nltk.book import *
```

След посланието за добре дошли, зареждат се текстовете на няколко книги, като първия текст е книгата Moby Dick.

```
>>> texts()
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
```

```
>>> text1
<Text: Moby Dick by Herman Melville 1851>
```

3.1.1. Търсене на текст съгласуван с определена дума в текста

```
>>> text1.concordance("love")
```

3.1.2. Друг текст в същия контекст на дума love, от предходния пример

```
>>> text1.similar("love")
```

```
sea man it ship view life queequeg land matter gush me place
ships way him line boats bone by hope
```

3.1.3. Проверка на контекста на дадена дума в различни текстове

```
>>> text2.common_contexts(["monstrous"])
```

```
a_lucky is_pretty a_pretty be_glad am_glad was_happy a_deal
is_fond
```

```
>>> text1.common_contexts(["monstrous"])
```

```
what_cannibal the_pictures this_cabinet of_clubs that_bulk
most_and a_size a_fable more_stories most_size
```

3.1.4. Дължина на текста

```
>>> len(text1)
260819
```

3.1.5. Речник на думите в текста – чрез използване на множество

```
>>> sorted(set(text3))
```

3.1.6. Брой срещания на дадена дума в текста

```
>>> text1.count("love")
```

```
24
```

3.1.7. Използване на индекси за получаване на достъп до дума

```
>>> text1[1]
```

```
'Moby'
```

```
>>> text1[2]
```

```
'Dick'
```

```
>>> text1[1:5]
```

```
['Moby', 'Dick', 'by', 'Herman']
```

```
>>> text1[-2] # предпоследна дума в текста
```

```
'orphan'
```

3.1.8. Честота на срещания на всички думи и символи в текста – чрез функцията FreqDist

```
>>> fd=FreqDist(text1)
>>> fd
FreqDist({',': 18713, 'the': 13721, '.': 6862, 'of': 6536,
'and': 6024, 'a': 4569, 'to': 4542, ';': 4072, 'in': 3916,
'that': 2982, ...})
>>> fd['and']
6024
```

Самостоятелна задача 1:

Пребройте думите в текст1 с дължина по-голяма от 10.

3.2. Работа със стандартния модул nltk. corpus

3.2.1. Колекция от текстове brown

```
>>> from nltk.corpus import brown
```

3.2.2. Категории (жанрове) в корпуса brown

```
>>> brown.categories()
['adventure', 'belles_lettres', 'editorial', 'fiction',
'government', 'hobbies', 'humor', 'learned', 'lore', 'mystery',
'news', 'religion', 'reviews', 'romance', 'science_fiction']
```

3.2.3. Използвани думи в определена категория от корпуса brown

```
>>> brown.words(categories='news')
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
```

Или в повече от една категория(жанр)

```
>>> brown.sents(categories=['news', 'editorial', 'reviews'])
```

Самостоятелна задача 2:

Намерете честотата на срещане на думите в жанра 'news', на текстовете в корпуса brown

За повече информация:

```
>>> import nltk
>>> help(nltk)
```

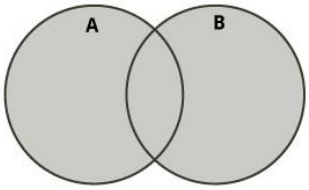
Или: http://www.nltk.org

4. **Множества (set)** – всеки елемент се съдържа **точно** един път. Елементите може да не са подредени. Създават се със служебната дума

set ([списък / комплект]) или {елемент$_1$, елемент$_2$, … , елемент$_n$}.
Множеството трябва да съдържа поне един елемент.

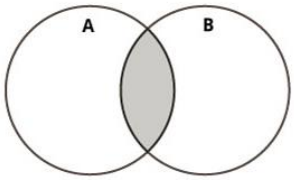
```
>>> x={3,1,2,5,5}
>>> x
{1, 2, 3, 5}
```

4.1. Обединение на множества (дизюнкция).



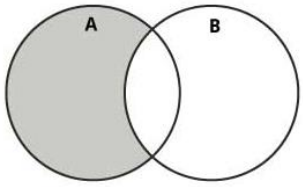

```
>>> A={1,2,3}
>>> B={2,1,4}
>>> A|B
{1, 2, 3, 4}
```

4.2. Сечение на множества (конюнкция).



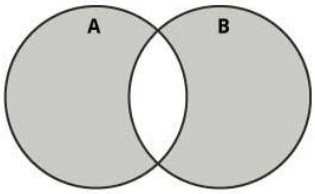

```
>>> A&B
{1, 2}
```

4.3. Разлика на множества.




```
>>> A-B
{3}
```

4.4. Симетрична разлика на множества.




```
>>> A^B
{3, 4}
```

4.5. Подмножество.

```
>>> C={1,2}
>>> A>C
True
```

Множеството C е подмножество на множеството A.

4.6. Сортиране на елементите на множество

```
 >>> sorted(x)
 [1, 2, 3, 5]
```

Самостоятелна задача 3:

Разгледайте самостоятелно методите на множество, като използвате help(set). Тествайте със собствени примери.

5. **Речници. –** представляват неподредено множество от двойка ключ(с уникална стойност) и стойност. Ключовете могат да бъдат число, низ, комплект. Стойностите могат да бъдат всеки тип. Можете да дефинирате, променяте, търсите и изтривате двойки ключ-стойност в речника.

5.1. Създаване на речник, чрез използване на фигурални скоби {}.

```
>>> lang={'USA':'English','Bulgaria':'Bulgarian'}
```

5.2. Проверка дали дадена дума(key) я има в речника

```
>>> lang['USA']
'English'
```

5.3. Проверка дали дадена дума(key) я има в речника чрез оператора in

```
>>> 'USA' in lang
True
>>> 'English' in lang
False
```

5.4. Извеждане на ключовете и стойностите на речника, поотделно и по двойки

```
>>> lang.keys()
dict_keys(['USA', 'Bulgaria'])
>>> lang.values()
dict_values(['English', 'Bulgarian'])
>>> lang.items()
dict_items([('USA', 'English'), ('Bulgaria', 'Bulgarian')])
```

5.5. Добавяне на нова двойка ключ-стойност в речника

```
>>> lang['Russia']='Russian'
>>> lang
{'USA': 'English', 'Bulgaria': 'Bulgarian', 'Russia': 'Russian'}
```

5.6. Изтриване на двойка ключ-стойност в речника

```
>>> del lang['Russia']
>>> lang
{'USA': 'English', 'Bulgaria': 'Bulgarian'}
```

5.7. Сортиране

```
>>> sorted(lang)
['Bulgaria', 'USA']
```

5.8. Създаване на речник, чрез използване на ключова дума dict и именувани параметри.

```
>>> dict(USA='English',Bulgaria='Bulgarian')
{'Bulgaria': 'Bulgarian', 'USA': 'English'}
```

5.9. Създаване на речник, чрез използване на ключова дума dict и списък от двойки.

```
>>> dict([('USA','English'),('Bulgaria','Bulgarian')])
{'Bulgaria': 'Bulgarian', 'USA': 'English'}
```

Самостоятелна задача 4:

Разгледайте самостоятелно методите на речника, като използвате help(dict). Тествайте със собствени примери.

6. Преобразуване между списък, комплект и множество. – чрез използване на ключовите думи: list, tuple и set.

6.1. Проверка на типа на променливата

```
>>> x=[1, 2, 3]
>>> type(x)
<class 'list'>
>>> y=1
>>> type(y)
<class 'int'>
```

6.2. Преобразуване от set в list

```
>>> list({1,2,3})
[1, 2, 3]
```

6.3. Преобразуване от tuple в list

```
>>> list((1,2,3))
[1, 2, 3]
```

6.4. Преобразуване от list в tuple

```
>>> tuple([1,2,3])
(1, 2, 3)
```

6.5. Преобразуване от set в tuple

```
>>> tuple({1,2,3})
(1, 2, 3)
```

6.6. Преобразуване от tuple в set

```
>>> set((1,2,3))
{1, 2, 3}
```

6.7. Преобразуване от list в set

```
>>> set([1,2,3])
{1, 2, 3}
```

7. Модули

Модулът(скриптът) е файл, състоящ се от оператори и дефиниции на Python, с разширение .py. Дефинициите от даден модул могат да бъдат импортирани в други модули. Python съдържа библиотека със стандартни модули. Обикновено всички оператори import(но, не задължително) се разполагат в началото на модула (скрипта).

След създаването на собствен файл, който можете да напишете в обикновен текстов редактор с примерно име name_modul.py, може да импортирате модула в интерпретатора на Python със следната команда:

```
>>> import name_modul
```

С вградената функция dir, можете да видите всички атрибути на модула:

```
>>>dir(name_modul)
```

8. Функции

```
def име на функция(списък от формални параметри):
    "Документация за функцията"
    блок от оператори   # задължително трябва да има отстъп
    [return [аргумент]] # задължително трябва да има отстъп
# return и/или аргумент не са задължителни
```

Дефинирането на функция започва с ключовата дума def, последвана от име на функцията и списък от формални параметри, оградени в скоби. Първият ред завършва задължително с две точки. Операторите в тялото на функцията задължително са въведени с отстъп. Вторият ред, може да бъде стринг, който представлява документация за функцията. Този пояснителен текст, относно функцията се появява при въвеждане в интерпретатора на: help(myfunc), където myfunc е името на вашата функция.

```
>>> def myfunc(x):
     "функция за.."
>>> help(myfunc)
Help on function myfunc in module __main__:
myfunc(x)
    функция за..
```

Функции могат да се дефинират и в командния интерпретатор, което е удобство при първоначалното им тестване.

Самостоятелна задача 5:

Създайте програма на Python, съдържаща функция, която да проверява дали дадена дума е палиндром. Реализирайте функцията по два начина: чрез оператора print и оператора return.

Стартиране на скрипта (вариант с оператора print***:***

Първи начин: чрез командния интерпретатор

```
>>> import func_Palindrom # примерно име на скрипта func_Palindrom
>>> func_Palindrom.Palindrom("aba")
Да! Думата aba е палиндром
```

Втори начин: чрез използване на командата Run (F5) от средата, където записвате скрипта.

```
>>> Palindrom("aba")
Да! Думата aba е палиндром
```

Стартиране на скрипта - вариант с оператора `print`***, чрез използване на командата Run (F5) от средата, където записвате скрипта***

```
>>> Palindrom("aba")
True
```

Самостоятелна задача 6:

Създайте функция на Python, която да изчислява най-големия общ делител на две числа, зададени като параметри на функцията.