Машинно обучение

Лабораторно упражнение №2

Въведение в Python

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

Пример 1:

```
'hello'
```

Резултат: *?*

Пример 2:

```
"hello"
```

Резултат: *?*

1.1. Вмъкване на нов ред. - \n

Пример 3:

```
print("hello\nhello ")
```

Резултат: *?*

1.2. Вмъкване на разстояние между думите. - \ t

Пример 4:

```
print("hello\tworld")
```

Резултат: *?*

1.3. Конкатенация. – чрез знака +

Пример 5:

```
'a'+'b'
```

Резултат: *?*

Пример 6:

```
"a"+"b"
```

Резултат: *?*

Пример 7:

```
'a'+"b"
```

Резултат: *?*

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

Пример 8:

```
x='ab'+'cd'
x
```

Резултат: *?*

Пример 9:

```
x[0]
```

Резултат: *?*

Пример 10:

```
x[1:3] # извежда символи от индекс 1 до индекс 2
```

Резултат: *?*

Пример 11:

```
x[-1] # извежда последния символ
```

Резултат: *?*

Пример 12:

```
x[-2] # извежда предпоследния символ
```

Резултат: *?*

Пример 13:

```
x[:3] # извежда първите три символа
```

Резултат: *?*

Пример 14:

```
x[3:] # извежда всичко, освен първите три символа
```

Резултат: *?*

Пример 15:

```
x[5:] # извежда празен символен низ, защото индекса е твърде голям
```

Резултат: *?*

1.5. Символни низове. Методи.

1.5.1. Дължина на низ

Пример 16:

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

Резултат: *?*

1.5.2. Брой символи

Пример 17:

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

Резултат: *?*

1.5.3. Преобразуване на символния низ в големи букви

Пример 18:

```
print(x.upper())
```

Резултат: *?*

1.5.4. Преобразуване на символния низ в малки букви

Пример 19:

```
print(x.lower())
```

Резултат: *?*

1.5.5. Разделяне на символния низ на отделни думи

Пример 20:

```
mytext="Hello world!"
mytext.split()
```

Резултат: *?*

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

Пример 21:

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

Резултат: *?*

1.5.7. Сортиране на списък от низове

1.5.7.1. Сортиране на списък от низове чрез sort()

Пример 22:

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

Резултат: *?*

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

Пример 23:

```
sorted(x)
```

Резултат:

['abc', 'acb', 'x']

Но:

Пример 24:

```
x
```

Резултат:

['x', 'abc', 'acb']

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

Пример 25:

```
sorted(x,reverse=True)
```

Резултат: *?*

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

Пример 26:

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

Резултат: *?*

Пример 27 :

```
sorted(x,key=len,reverse=True)
```

Резултат : *?*

1.5.7.5. Сортиране на низ чрез sorted

Пример 28 :

```
sorted('python')
```

Резултат : *?*

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.

Пример 29:

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

Резултат:

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

Пример 30:

```
a.shape
```

Резултат: *?*

Пример 31:

```
a.size
```

Резултат: *?*

Пример 32:

```
b = np.array([6, 7, 8])
b
```

Резултат: *?*

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:

Пример 33:

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

```
a
```

Резултат: *?*

***Пример* 34:**

```
a.dtype
```

Резултат: *?*

***Пример* 35:**

```
b = np.array([1.2, 3.5, 5.1])
b.dtype
```

Резултат: *?*

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

***Пример* 36:**

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

Резултат:

```
array([[ 1.5,  2. ,  3. ],
       [ 4. ,  5. ,  6. ]])
```

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

***Пример* 37:**

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

Резултат: *?*

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

Функцията *zeroes* 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.

Пример 38:

```
np.zeros( (3,4) )
```

Резултат: *?*

Пример 39:

```
np.ones( (2,3,4), dtype=np.int16 )    # dtype can also be specified
```

Резултат: *?*

Пример 40:

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

Резултат: *?*

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

Пример 41:

```
np.arange( 10, 30, 5 )
```

Резултат:

```
array([10, 15, 20, 25])
```

Пример 42:

```
np.arange( 0, 2, 0.3 )                 # it accepts float arguments
```

Резултат: *?*

При използване на *arange* 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:

Пример 43:

```
from numpy import pi
np.linspace( 0, 2, 9 )                 # 9 numbers from 0 to 2
```

Резултат:

```
array([ 0.  ,  0.25,  0.5 ,  0.75,  1.  ,  1.25,  1.5 ,  1.75,  2.
])
```

Пример 44 :

```
  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.

Пример 45 :

```
  a = np.arange(6)                         # 1d array
  print(a)
```

Резултат :?

Пример 46 :

```
  b = np.arange(12).reshape(4,3)           # 2d array
  print(b)
```

Резултат :

```
 [[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
```

Пример 47 :

```
  c = np.arange(24).reshape(2,3,4)         # 3d array
  print(c)
```

Резултат :?

```
 [[[ 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).

Пример 48:

```
a = np.array( [20,30,40,50] )
b = np.arange( 4 )
b
```

Резултат:?

```
array([0, 1, 2, 3])
```

Пример 49:

```
c = a-b
c
```

Резултат:?

```
array([20, 29, 38, 47])
```

Пример 50:

```
b**2
```

Резултат:?

```
array([0, 1, 4, 9])
```

Пример 51:

```
10*np.sin(a)
```

Резултат:?

Пример 52:

```
a<35
```

Резултат:

```
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*.

Пример 53:

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

Резултат:

```
array([[2, 0],
       [0, 4]])
```

Пример 54:

```
  A @ B                         # matrix product
```

Резултат:

```
array([[5, 4],
       [3, 4]])
```

Пример 55:

```
  A.dot(B)                      # another matrix product
```

Резултат:

```
array([[5, 4],
       [3, 4]])
```

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

Пример 56:

```
  a = np.ones((2,3), dtype=int)
  b = np.random.random((2,3))
  a *= 3
  a
```

Резултат:

```
array([[3, 3, 3],
       [3, 3, 3]])
```

Пример 57:

```
b += a
b
```

Резултат: *?*

Пример 58:

```
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).

Пример 59:

```
a = np.ones(3, dtype=np.int32)
b = np.linspace(0,pi,3)
b.dtype.name
```

Резултат:

```
'float64'
```

Пример 60:

```
c = a+b
c
```

Резултат:

```
array([ 1.        ,  2.57079633,  4.14159265])
```

Пример 61:

```
c.dtype.name
```

Резултат:

```
'float64'
```

Пример 62:

```
d = np.exp(c*1j)
d
```

Резултат: *?*

Пример 63:

```
d.dtype.name
```

Резултат: *?*

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

Пример 64:

```
a = np.random.random((2,3))
a
```

Резултат: *?*

Пример 65:

```
a.sum(),a.min(),a.max()
```

Резултат: *?*

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

Пример 66:

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

Резултат:

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

Пример 67 :

```
b.sum(axis=0)                                   # sum of each column
```

Резултат :

```
array([12, 15, 18, 21])
```

Пример 68 :

```
b.min(axis=1)                                   # min of each row
```

Резултат :

```
array([0, 4, 8])
```

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.

Пример 69 :

```
B = np.arange(3)
B
```

Резултат : *?*

Пример 70 :

```
np.exp(B)
```

Резултат : *?*

Пример 71 :

```
np.sqrt(B)
```

Резултат : *?*

Пример 72 :

```
C = np.array([2., -1., 4.])
np.add(B, C)
```

Резултат : *?*

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().

Пример 73:

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

Резултат:

```
['Hello', ',', 'world', '!']
```

3.1. Работа със стандартния модул NLTK - nltk.book

Пример 74:

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

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

```
texts()
```

Резултат:

```
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
```

Резултат:

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

3.1.1. Търсене на текст съгласуван с определена дума в текста

Пример 75:

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

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

Пример 76:

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

Резултат:

```
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

Пример 77:

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

Резултат:

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

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

Резултат:

```
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

Пример 78:

```
len(text1)
```

Резултат: *?*

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

Пример 79:

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

3.1.6. Брой срещания на дадена дума в текста

Пример 80:

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

Резултат: *?*

3.1.7. Използване на индекси за получаване на достъп до дума

Пример 81:

```
text1[1]
```

Резултат: *?*

Пример 82:

```
text1[2]
```

Резултат: *?*

Пример 83:

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

Резултат: *?*

Пример 84:

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

Резултат: *?*

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

Пример 85:

```
fd=FreqDist(text1)
fd
```

Резултат: *?*

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

Пример 86:

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

3.2.2. Категории (жанрове) в корпуса brown

```
brown.categories()
```

Резултат: *?*

3.2.3. Използвани думи в определена категория от корпуса brown

Пример 87:

```
brown.words(categories='news')
```

Резултат:

```
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
```

Или в повече от една категория(жанр)

Пример 88:

```
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. 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:

Пример:

```
import name_ modul
```

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

Пример:

```
dir(name_ modul)
```

5. 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 3:

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")
```

Резултат:

Да! Думата aba е палиндром

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

```
Palindrom("aba")
```

Резултат:

Да! Думата aba е палиндром

Самостоятелна задача 4:

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