УНИВЕРСИТЕТ
искусственного
интеллекта

Библиотека Numpy

#2

Библиотека Numpy

Занятие № 2

Библиотека NumPy

Библиотека NumPy – библиотека Python для работы с многомерными массивами, включая матрицы, и включающая большое количество функций для работы с ними.

В Google colab данная библиотека установлена по умолчанию.

Официальную документацию можно почитать https://numpy.org/doc/.

На русском можно посмотреть, например, https://pythonworld.ru/numpy.

Для подключения библиотеки: `import numpy as np`

импортируем модуль numpy как np.

Структура np.ndarray

ndarray – это многомерный массив фиксированного размера, состоящий из данных одного типа. В случае если при создании массива были указаны данные разного типа, то при переводе в numpy-массив данные будут приведены к одному типу.

Структура np.ndarray

Например,

```
my_list1 = [2, 4, 5]
my_numpy1 = np.array(my_list1)
print(my_numpy1)
print(type(my_numpy1))
print(my_numpy1.dtype)
```

вернет

```
[2 4 5]
<class 'numpy.ndarray'>
int64
```

Для

```
my_list2 = [2, 3.14]
my_numpy2 = np.array(my_list2)
print(my_numpy2.dtype)
print(my_numpy2)
```

вернет

```
float64
[2.   3.14]
```

Усложним,

```
my_list3 = [2, 3.14, 'text data']
my_numpy3 = np.array(my_list3)
print(my_numpy3)
print(my_numpy3.dtype)
```

вернет

```
['2' '3.14' 'text data']
<U32
```

Типы данных

В NumPy реализовано больше типов данных, чем в Python.

Подробнее о типах:

https://docs.scipy.org/doc/numpy/user/basics.types.html

https://numpy.org/doc/1.18/user/basics.types.html?highlight=basics%20types#module-numpy.doc.basics

Типы данных

Например,

```
value = -167
numpy_value = np.uint8(value)
print(value)
print(numpy_value)
```

вернет

```
-167
89
```

Посмотрим типы этих переменных:

```
print(type(value))
print(type(numpy_value))
```

вернет

```
<class 'int'>
<class 'numpy.uint8'>
```

Одномерные массивы

Одномерный массив в аналогии таблиц – это таблица, в которой только одна строка.

Одномерные numpy массивы

Для того чтобы создать одномерный numpy-массив, можно в качестве аргумента встроенной функции .array() передать список значений.

Например,

```
numpy_digital = np.array([1, 2, 3, 4, 5])
```

либо можно воспользоваться генератором списка,

```
arr2 = np.array([i for i in range(7)]
```

либо передать в качестве аргумента число,

`numpy_10 = np.arange(5)` – создает массив из пяти элементов от 0 до 4,

Одномерные массивы

либо, если необходимо в заданном интервале,

`numpy_11 = np.arange(3, 6)` - создает массив от 3 до 5, 6 не включается,

либо, если необходимо создать массив в заданном интервале с шагом,

`np.arange(3, 16, 4)` - создает массив от 3 до 15 с шагом 4.

Для того чтобы при создании массива указать тип данных, необходимо в качестве значения переменной dtype передать тип данных,

```
np.array(values, dtype=my_type)
```

В numpy для работы с массивами есть встроенные функции:

.dtype() – вернет тип элементов массива, пример,

```
print(numpy_digital.dtype)
```

вернет

```
int64
```

.shape() – выводит размер массива, например,

```
print(numpy_digital.shape)
```

вернет

```
(5,)
```

.size() – вернет количество элементов массива, например,

```
print(numpy_digital.size)
```

вернет,

```
5
```

.reshape() – функция вернет новый массив с новым размером, в качестве аргумента принимает размер нового массива в виде кортежа. Например,

```
numpy_digital2 = np.array([1, 2, 3, 4, 5, 6])
numpy_new = numpy_digital2.reshape((3, 2))
```

посмотрим, что получилось:

```
print(numpy_new)
print(numpy_new.shape)
```

вернет

```
[[1 2]
 [3 4]
 [5 6]]
(3, 2)
```

Одномерные массивы

Индексация

Обратиться к элементу массива numpy можно так же как и для обычного списка, через индекс элемента.

Рассмотрим примеры.

Для того чтобы получить элемент с индексом 1

```
numpy_digital2[1]
```

также можно использовать и отрицательные индексы

`numpy_digital2[-1]`, вернет последний элемент массива.

Срезы:

вывести все элементы, начиная с заданного и до конца массива,

```
numpy_new5[1:]
```

вывести все элементы, начиная с первого и заканчивая заданным,

```
numpy_new5[:3]
```

вывести все элементы из заданного диапазона,

```
numpy_new5[1:4]
```

вывести все элементы, которые попадают под маску (<, >, ==,!=),

```
numpy_new5 > 55
```

выборка элементов, например, нужны элементы, которые больше 55 и меньше 67,

```
numpy_new5[(numpy_new5 > 25) & (numpy_new5 < 67)]
```

Встроенные методы

Создадим массив с целочисленными данными:

```
numpy_array = np.array([1,2,3,4,5,6,7,8,10])
```

Для обработки массива есть встроенные методы:

```
.sum() - выводит сумму элементов массива,
numpy_array.sum()

.mean() - выводит среднее значение элементов массива,
numpy_array.mean()
```

Одномерные массивы

.max() - выводит максимальное значение элементов массива, numpy_array.max()

.min() - выводит минимальное значение элементов массива, numpy_array.min()

.prod() - выводит произведение всех элементов, numpy_array.prod()

.sort() - сортирует все элементы массива numpy_array.sort()

Векторные операции

Создадим одномерные массивы

```
my_list1 = [0,1,2,3,4,5]
my_list2 = [6,7,8,9,10,11]
my_numpy1 = np.array(my_list1)
my_numpy2 = np.array(my_list2)
```

- Сложение двух массивов `my_numpy1 + my_numpy2`, результат:

```
[ 6  8 10 12 14 16]
```

- Разница двух массивов `my_numpy1 - my_numpy2`, результат:

```
[-6 -6 -6 -6 -6 -6]
```

- Частное двух массивов `my_numpy1 / my_numpy2`, результат:

```
[0.         0.14285714 0.25       0.33333333 0.4        0.45454545]
```

- Поэлементное умножение `my_numpy1 * my_numpy2`, результат:

```
[ 0  7 16 27 40 55]
```

- По элементное возведение в степень `my_numpy1 ** my_numpy2`, результат:

```
[      0       1     256   19683 1048576 48828125]
```

В случае если идет каскад действий, необходимо каждое действие оборачивать в скобки, например, `((my_numpy1 * my_numpy2 - my_numpy1) ** 2).sum()`, результат: `4604`.

Многомерные массивы

Двумерный массив (2D) – матрица

Многомерный массив (3D, ..., nD) – тензоры

Способы создания многомерного массива ничем не отличаются от создания одномерного массива. Например,

двумерный массив:

```
my_2d_array = np.array([[3, 6, 2, 7],
                        [9, 2, 4, 8],
                        [8, 2, 3, 6]])
```

посмотрим размерность:

```
print(my_2d_array.shape)
(3, 4)
```

трехмерный массив:

```
my_3d_array = np.array([[[3, 6, 2, 7],
                        [9, 2, 4, 8],
                        [8, 2, 3, 6]]])
```

посмотрим размерность:

```
print(my_3d_array.shape)
(1, 3, 4)
```

Индексация

Для того чтобы вывести строку, необходимо воспользоваться построчной индексацией, то есть указать индекс строки, например,

```
print(my_2d_array[0])
```

выведет

```
[3 6 2 7]
```

Для получения элемента необходимо указать индекс строки и индекс элемента в этой строке, например,

```
print(my_2d_array[0,0])
```

выведет

```
3
```

Многомерные массивы

Для того чтобы получить конкретный столбец, необходимо указать, что во всех строках нужно вывести элемент, при этом указав индекс элемента (столбца), например,

```
print(my_2d_array[:, 0])
```

выведет

```
[3 9 8]
```

то есть вывели на экран из всех строк первый элемент (столбец).

Операции с двумерными массивами и встроенные методы

Как и в одномерных массивах существуют встроенные методы, такие как

.sum() – сумма элементов, которая может принимать дополнительный необязательный параметр, axis, при axis = 0, сумма в каждом столбце, при axis = 1, сумма в каждой строке. Пример,

```
print(my_2d_array.sum(axis=0))
```

выведет

```
[20 10  9 21]
```

В случае если axis не указан, то метод применяется ко всему массиву.

Данное правило применяется и для остальных методов: .min(), .max(), .mean().

Для создания единичной матрицы с размерностью как у исходной, можно воспользоваться методом np.ones_like(), например,

```
new_2d_array = np.ones_like(my_2d_array)
 [[1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]]
```

Многомерные массивы

Для создания нулевой матрицы существует метод np.zeros(), который на вход получает размерность матрицы, например,

```
new_2d_array2 = np.zeros((5,3))
```

выведет

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

Структура np.matrix

Класс numpy.matrix возвращает матрицу из массива подобного объекта или из строки данных. Он имеет определенные специальные операторы, такие как * (умножение матриц) и **(матричная степень).

Общий вид выглядит так: numpy.matrix(data, dtype=None, copy=True), где

data: массив данных или строка. Если data – это строка, она интерпретируется, как матрица с запятыми или пробелами, разделяющими столбцы, и точками с запятой, разделяющими строки.

dtype: тип данных выходной матрицы;

copy: этот флаг определяет, копируются ли данные (по умолчанию) или создается представление.

Подробнее можно ознакомиться в официальной документации https://numpy.org/doc/stable/reference/generated/numpy.matrix.html?highlight=numpy%20matrix.

Рассмотрим примеры,

создадим двумерный массив:

```
arr1 = np.array([[1,2,3],
                 [4,5,6]])
```

Многомерные массивы

создадим матрицу из `arr1`:

```
my_matrix = np.matrix(arr1)
```

содержимое матрицы:

```
print(my_matrix)
```

выведет

```
[[1 2 3]
 [4 5 6]]
```

транспонированная матрица:

```
print(my_matrix.T)
```

выведет

```
[[1 4]
 [2 5]
 [3 6]]
```

перемножим две матрицы, в случае перемножения матриц * – это матричное умножение, а не поэлементное, поэтому `arr1` необходимо будет еще транспонировать:

```
print(my_matrix * my_matrix.T)
```

результат:

```
[[14 32]
 [32 77]]
```

Тензоры a.k.a. - многомерные массивы

Тензоры – это многомерные массивы. Посмотрим, как с помощью numpy создаются многомерные массивы:

- создадим четырехмерный массив из случайных значений размерности 2 на 3 на 4 на 6 на 5:

```
my_tensor = np.random.random((2, 3, 4, 5))
print(my_tensor)
```

Многомерные массивы

выведет

```
[[[[0.490094   0.49870947 0.74660723 0.07044891 0.35543921]
   [0.6637609  0.75409547 0.80830949 0.61249681 0.98270387]
   [0.39662531 0.2481841  0.2758839  0.25208422 0.18516   ]
   [0.28643845 0.89836457 0.48252541 0.24582907
0.42072561]]

  ..........................................................................

  [[0.60010854 0.90914786 0.09340423 0.13080453 0.30693353]
   [0.17938668 0.31719075 0.71592526 0.97029386 0.62679286]
   [0.87345786 0.21215177 0.90419294 0.58731224 0.56136374]
   [0.73950434 0.4657924  0.45530383 0.04914556
0.99901369]]]]
```

- посмотрим размерность получившегося массива:

```
print(my_tensor.shape)
```

выведет

```
(2, 3, 4, 5)
```

- для того чтобы обратиться к конкретному элементу массива, необходимо указать его индексы, например,

```
my_tensor[0, 0, 0, 0]
```

равен

```
0.49009399824242994
```

- только первые строки:

```
my_tensor[:,  :, :, 0]
```

результат:

```
[[[0.490094  , 0.6637609 , 0.39662531, 0.28643845],
        [0.34161537, 0.10467532, 0.89377351, 0.00218099],
        [0.14864933, 0.69638872, 0.92274297, 0.11965057]],

       [[0.36429309, 0.18257134, 0.95442749, 0.64174413],
        [0.10201686, 0.29380802, 0.38462493, 0.79692998],
        [0.60010854, 0.17938668, 0.87345786, 0.73950434]]]
```

Многомерные массивы

- сумма элементов по второй оси:

```
my_tensor.sum(axis=1)
```

результат:

```
[[[0.9803587 , 1.0906532 , 2.1976006 , 1.69274232, 1.67936266],
  [1.46482494, 1.7081485 , 1.48352424, 2.12398954, 2.26971894],
  [2.2131418 , 1.20362279, 1.18144247, 1.52908601, 1.28289239],
  [0.40827001, 2.11817831, 1.19940149, 1.54632365, 0.69872843]],

 [[1.06641849, 1.33281923, 0.65566497, 0.63855332, 1.27098487],
  [0.65576604, 2.06634724, 1.61393052, 1.19160588, 1.08153308],
  [2.21251028, 1.25356364, 2.14046712, 1.38625198, 0.92587103],
  [2.17817845, 0.74200055, 1.7723153 , 1.26714828, 2.01744559]]]
```

Матричные операции

В numpy помимо векторных операций существуют и матричные. Для матричного умножения в модуле numpy есть метод .dot().

Синтаксис `arr1.dot(arr2)`, в случае матричного умножения надо учитывать, чтобы массивы были совместимы, то есть количество столбцов arr1 должно было равно количеству строк массива arr2.

Для операции транспонирования: замена строк на столбцы (то есть строки становятся столбцами) есть метод .T

Синтаксис arr1.T, например,

для матрицы arr1

```
print(arr1)
[[1 2 3]
 [4 5 6]]
```

Транспонированная матрица

```
arrT = arr1.T
print(arrT)
[[1 4]
 [2 5]
 [3 6]]
```

Семплирование из распределений

Для создания массива из случайных чисел в numpy есть модуль np.random. Описание всех возможностей модуля np.random:

https://docs.scipy.org/doc/numpy-1.15.0/reference/routines.random.html

Для создания массива из случайных чисел с нормальным распределением в np.random есть метод .normal(), который в качестве аргументов принимает на вход начальное значение, конечное значение и размер массива. Например,

```
np.random.normal(0, 1, (5,5))
```

в результате получим массив:

```
[[-1.02502993e-01 -8.66954912e-05 -6.80314253e-01 -7.65429869e-02
  -8.23748683e-01]
 [-8.99243177e-01 -1.33474028e+00 -1.02546839e+00  8.97763756e-02
   5.43277534e-03]
 [-1.20754220e+00 -1.21841014e-01 -1.50036886e+00 -3.09280770e-01
   5.27568987e-01]
 [-1.51109176e+00 -1.62000272e+00 -5.40749918e-01 -1.28826852e+00
   1.11916647e+00]
 [-1.82363482e-01  1.73263674e+00 -5.08088135e-01  7.70289317e-01
  -3.47502891e-01]]
```

Для генерации массива из целочисленных случайных чисел в модуле np.random есть метод .randint(), который на вход принимает начальное значение, конечное значение, размер массива, например,

```
np.random.randint(0, 10, (5,5))
```

в результате получим массив:

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

Чтение файлов

Для загрузки данных с диска есть встроенные методы для чтения файлов,

Чтение файлов

Для загрузки данных с диска есть встроенные методы для чтения файлов,

np.loadtxt(), в качестве аргументов принимает: fname, dtype=<class 'float'>, comments='#', delimiter=None, converters=None, skiprows=0, usecols=None, unpack=False, ndmin=0, encoding='bytes', max_rows=None, где

fname – имя файла,

dtype – тип данных NumPy (необязательный).

Определяет тип данных выходного массива. По умолчанию float. Если в файле находятся структурированные данные, т.е. структурированный тип данных, то результирующий массив будет одномерным, а каждая строка файла станет его отдельным элементом. В этом случае необходимо определить структурированный тип данных, количество полей в котором должно соответствовать количеству колонок в текстовом файле.

comments – строка или последовательность строк (необязательный).

Символы или список символов, используемых для указания начала строк-комментариев. Байтовые строки декодируются как Latin-1. По умолчанию используется символ '#'.

delimiter – строка (необязательный).

Строка-разделитель между значениями в текстовом файле. По умолчанию используется пробел.

converters – словарь (необязательный).

Словарь, в качестве ключей которого используются целые числа, соответствующие столбцам данных в файле. Значение ключа сопоставляет соответствующий ключу столбец с соответствующей ключу функцией конвертирования.

skiprows – целое число (необязательный).

Указывает количество строк, которое необходимо пропустить. По умолчанию равно 0.

Чтение файлов

usecols – целое число или кортеж целых чисел (необязательный).

Указывает, какие столбцы будут считаны. Первый столбец имеет номер 0. По умолчанию usecols=None, что соответствует чтению всех столбцов в файле. Целое число, например, usecols=1 так же как и кортеж с одним значением usecols=(1,) приведет к считыванию единственного из всех столбцов в файле, т.е. будет прочитан только второй столбец.

unpack – True или False (необязательный).

Если этот параметр равен True, то все столбцы текстового файла могут быть разделены и распакованы с помощью a, b, c = loadtxt(...). Использование структурированного типа данных приводит к тому, что для каждого отдельного поля будет возвращен отдельный массив. По умолчанию unpack=False

ndmin – целое число (необязательный).

Определяет минимальную размерность возвращаемого массива. По умолчанию ndmin=0, что соответствует сжатию всех осей длиной 1 до одной оси.

Пример,

```
my_data = np.loadtxt('iris.csv', delimiter=',', skiprows=1)
```

– загрузить файл iris.csv при чтении которого использовать разделитель ',', пропустить первую строчку, так как в ней название колонок базы.

Глоссарий

array[n1:n2] – срез массива с элемента n1 по n2 (не включительно).

*Основные методы для ndarray *

.dtype – узнать тип данных массива

.shape – узнать размер массива

.reshape((новый размер)) – поменять размер массива

.sum() сумма всех элементов

.mean() среднее всех элементов

.min() минимальное значение массива

.max() максимальное значение

.prod() произведение всех элементов

np.arange(n) – генерация массива со значениями от 0 до n (не включительно)

np.loadtxt (адрес расположения файла+параметры загрузки).