Python libraries: NumPy

WELCOME

Insert a
RECENT and
HIGH QUALITY
photo of yourself
here

Trainer Name

Trainer Role, QA

- A few key facts…
- Previous role
- Qualifications

SESSION OVERVIEW

- Introductions and ice-breaker activity
- Engage in practical activities to support module evidence collection
- Provide an overview of the 3-day class-based learning
- Provide support and guidance for the successful completion of Module 4B

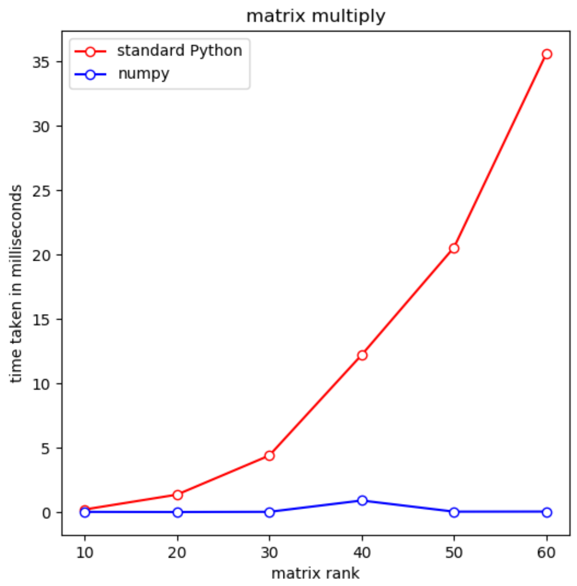
OVERVIEW

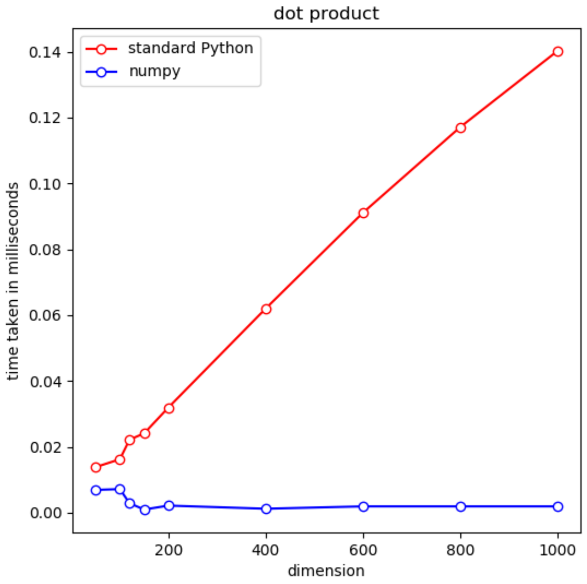
- Why NumPy?
- What is NumPy?
- ND-arrays
- Slice and Dice
- Array operations
- Summary Statistics

PYTHON IS SLOW

Python collections are not designed for computational efficiency.

C and FORTRAN arrays are much more efficient computationally than Python lists for large datasets.





WHAT IS NUMPY?

NumPy was introduced in 2006 to address the inefficiencies of Python in dealing with large amounts of data.

- Written in C and FORTRAN
- Internal data structure uses C arrays
- Python API for seamless integration with Python
- **Provides its own array types (ND-arrays)**
- **Arrays retain most Python collection behaviours, so that it looks and feels ‘native’ to Python language**
- Incorporates fast maths libraries, such as OpenBLAS (default, open source), for efficient linear algebraic operations (dot products, matrix multiply, etc.)

NOTE: To use NumPy it is necessary to import it

```
import numpy as np
```

ND-ARRAYS

ND-arrays stands for **N-dimensional arrays**

- The basic data type in NumPy, intended to replace Python’s list
- Can be created from Python’s list using **numpy.array()**
- nd-arrays are **mutable**
- **numpy.arange()** produces a sequence of numbers contained in an array

ND-ARRAYS

Creating a 1-dimensional array

```
import numpy as np

arr = np.array([1, 3, 5, 7, 9])

print(arr)
```

```
[1 3 5 7 9]
```

Creating a 2-dimensional array

```
arr2 = np.array([[1, 3, 5, 7], [2, 4, 6, 8]])

print(arr2)
```

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

The ndim attribute returns the number of dimensions of the array

```
arr2.ndim
```

```
2
```

ND-ARRAYS

arange() creates an array with evenly spaced values.

numpy.arange([start,]stop, [step,], dtype=None)

- **start -** the first value in the array.
- **stop -** the number that defines the end of the array. It is not included in the array.
- **step -** the spacing (difference) between each two consecutive values in the array. The default step is 1. Step cannot be zero.
- **dtype -** the type of the elements of the output array. Defaults to None. If dtype is omitted, arange() will try to deduce the type of the array elements from the types of start, stop, and step.

```
MyArray = np.arange(start=1, stop=10, step=2)
print(MyArray)
```

```
[1 3 5 7 9]
```

```
MyArray = np.arange(start=1, stop=10, step=3)
print(MyArray)
```

```
[1 4 7]
```

DTYPE

All arrays can only contain elements of the same data type.

→ This is valid for ND-arrays too

The type of the element in an array is recorded as a dtype object

- Standard Python data types can be used as dtypes: e.g., int, float, str
- dtype of an array can be obtained using array.dtype property
- We can perform type conversion using array.astype(new_type)
- The new_type must be compatible with the original type of the elements
- If in doubt, NumPy automatically converts an array to an array of strings

ND-ARRAY SHAPE AND SIZES

The built-in function len() does not work with nd-arrays

- To find out the size of a nd-array, use **array.size** property
- To find out the shape (size of each dimension) of an array, use **array.shape** property
- To change the shape of an array, use **array.reshape()**

NOTE: It is the programmer's responsibility to make sure the new shape is compatible with the total number of elements.

ND-ARRAY SHAPE AND SIZES

The shape attribute returns a tuple with the number of elements in each dimension.

```
arr2 = np.array([[1, 3, 5, 7], [2, 4, 6, 8]])

print(arr2.shape)
```

```
(2, 4)
```

→2 rows, 4 columns

Reshaping an array means change the number of dimensions or change the number of elements in each dimension. This is done using reshape()

```
arr = np.array([[1, 3, 5, 7, 9, 11], [2, 4, 6, 8, 10, 12]])

print(arr)
```

```
[[ 1  3  5  7  9 11]
 [ 2  4  6  8 10 12]]
```

```
arr2 = arr.reshape(3,4)
print(arr2)
```

```
[[ 1  3  5  7]
 [ 9 11  2  4]
 [ 6  8 10 12]]
```

SLICE AND DICE

Standard Python list slices:

→array[i] obtains the i-th element

→array[n:m] obtains the elements array[n], array[n+1], …, array[m-1] in a new array

→array[I,j] obtains the element on row I and column j of a 2-dimensional array

New to ND-arrays:

Cherry-picking

→array[[2, 4, 5, 1]] obtains the elements array[2], array[4], array[5], array[1] in a new array

→Cherry picking list can be any Python iterator with integer elements

Filtering

→array[[True, True, False, … False, True]] obtains the elements from positions marked as True in a new array, omits those marked by False

→Filter list can be any Python iterator with Boolean elements, and its length must be the same as the array

→The filter list is usually computed rather than written by hand

OPERATIONS

- All arithmetic operations are performed element by element
- All comparisons are performed element by element
- Rounding can be performed element-wise using **array.round()**
- **numpy.dot(v, u)** computes the **dot product** of arrays v and u
- **A @ B** computes the **matrix product** of arrays A and B. Alternatively, you can use **numpy.matmul(A, B)**
- **array.T** transposes an array

DOT PRODUCT

The **dot product** of two vectors (one-dimensional arrays) with equal number of elements

$A = [A_1, A_2, \dots, A_n]$ and $B = [B_1, B_2, \dots, B_n]$

Is defined as

$$A \cdot B = \sum_{i=1}^{n} A_i B_i = A_1 B_1 + A_2 B_2 + \cdots + A_n B_n$$

Dot product produces a single value (scalar)

Example:

X = [1, 3, 5]

Y = [2, 4, 6]

$$X \cdot Y = 1 * 2 + 3 * 4 + 5 * 6 = 44$$

MATRIX PRODUCT

The matrix product of two matrices produces a matrix.

- The number of columns in the first matrix must be equal to the number of rows in the second matrix.
- The resulting matrix has the number of rows of the first and the number of columns of the second matrix.

$$A_{m\times n} \quad B_{n\times p} \quad \rightarrow \quad AB_{m\times p}$$

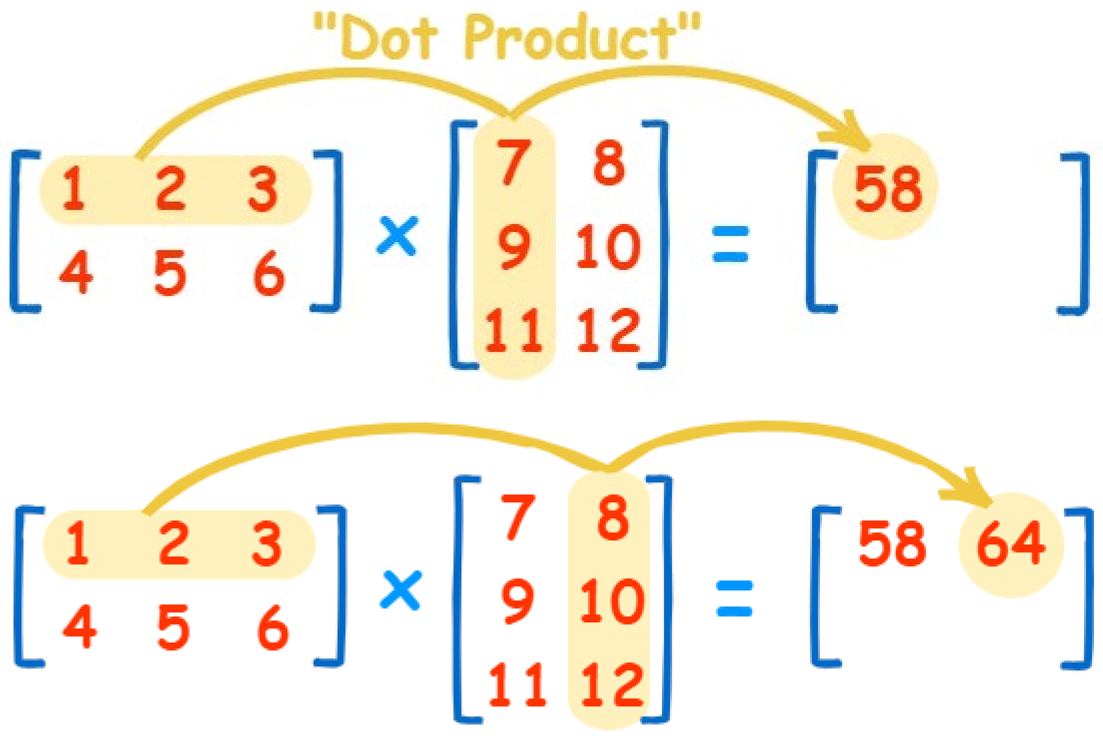
Each element C_{ij} of the matrix product C=AB is calculated as follows

$$C_{ij} = \sum_{k=1}^{n} A_{ik}B_{kj}$$

→ Row by Column

MATRIX PRODUCT - CONTINUED

Difficult? A picture is worth a thousand words!



Images source:

https://mathsisfun.com/

MATRIX PRODUCT – SIMPLE EXAMPLE

EXAMPLE:

A coffee shop sells three kinds of cake: Banana (price £3 per slice)**, Kiwi** (price £4 per slice) **and Black Forest** (price £5 per slice)**. The quantities of each cake they have sold during the week are:**

SALES	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
Banana	5	6	4	7	0	10	9
Kiwi	0	3	5	4	6	8	7
Black Forest	6	7	0	5	8	10	13

→ **Calculate the total revenue (money) for each day of the week – in Excel. There are three ways to do it, and one of them is using Excel's matrix multiplication function.**

→ **You can use file Cakes.xlsx which contains the data.**

LOGICAL OPERATORS AND FUNCTIONS

Elementwise operators

- →~ NOT
- →& AND
- →| OR
- →^ XOR

Functions acting on entire array

- →numpy.all()
- →numpy.any()

DESCRIPTIVE (SUMMARY) STATISTICS

NumPy comes with a full set of statistical functions:

numpy.sum()

numpy.min()

numpy.max()

numpy.mean()

numpy.median()

numpy.var()

numpy.std()

numpy.corrcoef()

Further Reading

https://www.python.org/