Linear Algebra with NumPy

Introduction to AI Programming II Course :: Copylight by Mr.Oh

What's a Matrix?

According to Wikipedia:
A matrix *is a rectangular array of numbers or other mathematical objects for which operations such as addition and multiplication are defined.*
As a data scientist, you are using matrices all the time, but you probably don't know that (just yet). Any dataset you've used in the past can be thought of as a matrix — a rectangular array of numbers — **rows and columns** to be more specific.
You might be wondering how does matrix differ from a vector from a data scientists perspective. Simply put, a vector is a **single column** (attribute) in your dataset, while matrix is a **collection of all columns**.
Matrices are usually denoted with a capital bolded letter, like this for example:

1. Matrix Addition

   Matrix addition (or subtraction) is really similar to the one you did with vectors earlier. The only difference is that there are multiple columns instead of just one. The whole idea remains the same, you only need to **add up the corresponding components**. In the general formula, I've used ***a*** and *b* for placeholders, and you can see how each component is added up:

$$\begin{bmatrix} \boldsymbol{a_1} & \boldsymbol{a_2} \\ \boldsymbol{a_3} & \boldsymbol{a_4} \end{bmatrix} + \begin{bmatrix} \boldsymbol{b_1} & \boldsymbol{b_2} \\ \boldsymbol{b_3} & \boldsymbol{b_4} \end{bmatrix} = \begin{bmatrix} \boldsymbol{a_1 + b_1} & \boldsymbol{a_2 + b_2} \\ \boldsymbol{a_3 + b_3} & \boldsymbol{a_4 + b_4} \end{bmatrix}$$

Although this is fairly simple to grasp, here's a simple example of 2 matrix addition:

$$A = \begin{bmatrix} 3 & 5 \\ 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 2 & -3 \\ 1 & 2 \end{bmatrix}$$

$$A + B = \begin{bmatrix} 3 + 2 & 5 + (-3) \\ 1 + 1 & 0 + 2 \end{bmatrix}$$

$$A + B = \begin{bmatrix} 5 & 2 \\ 2 & 2 \end{bmatrix}$$

Matrix addition is really simple to implement in Numpy.

```
import numpy as np
A = np.array([[3,5],[1,0]])
B = np.array([[2,-3],[1,2]])
C = A+B
print(C, C.shape, C.ndim)
```

2. Scalar Multiplication

The concepts are more or less the same as with vectors. Every number in the matrix will be multiplied with some scalar $\boldsymbol{n}$.
The formula is also very similar:

$$\boldsymbol{n} \begin{bmatrix} \boldsymbol{a_1} & \boldsymbol{a_2} \\ \boldsymbol{a_3} & \boldsymbol{a_4} \end{bmatrix} = \begin{bmatrix} \boldsymbol{na_1} & \boldsymbol{na_2} \\ \boldsymbol{na_3} & \boldsymbol{na_4} \end{bmatrix}$$

For the example, I've chosen to use an arbitrary matrix, and I've set the scalar $\boldsymbol{n}$ to 2:

$$A = \begin{bmatrix} 3 & 5 \\ 1 & 0 \end{bmatrix}$$

$$2A = \begin{bmatrix} 2 \times 3 & 2 \times 5 \\ 2 \times 1 & 2 \times 0 \end{bmatrix}$$

$$2A = \begin{bmatrix} 6 & 10 \\ 2 & 0 \end{bmatrix}$$

```
import numpy as np
A = np.array([[3,5],[1,0]])
B = 2*A
print(B, B.shape, B.ndim)
```

3. Matrix Multiplication

Here comes a topic that I would say is slightly more complex to grasp on then the others encountered so far. It isn't as hard as it might seem at first, but you'll need to solve a couple of examples to get the gistfully.
For the following examples in matrix multiplication section, two matrices are declared:

1. Matrix **A** — has dimensions of *m* by *n* (*m* rows, *n* columns)
2. Matrix **B** — has dimensions of *n* by *p* (*n* rows, *p* columns)

Multiplication of **A** and **B** will yield a new matrix that has dimensions of *m* by *p* (*m* rows by *p* columns). In plain English, the resulting matrix will have the number of rows that matrix **A** has, and a number of columns that matrix **B** has.
You will probably need to read the last paragraph a couple of times before you understand it fully, and that's okay. To help you out, here everything I've said so far presented visually:

$$\underset{m \times n}{\mathbf{A}} \quad \times \quad \underset{n \times p}{\mathbf{B}} \quad = \quad \underset{m \times p}{\mathbf{AB}}$$

As you can see, two *n*'s in the middle need to match. If they are not equal, **matrix multiplication cannot be performed**. Most programming languages will throw you an error on **dimension mismatch**.

Okay, now when you understand the basic rule of matrix multiplication, you are ready for the general formula:

$$\begin{bmatrix} \boldsymbol{a_1} & \boldsymbol{a_2} \\ \boldsymbol{a_3} & \boldsymbol{a_4} \end{bmatrix} + \begin{bmatrix} \boldsymbol{b_1} & \boldsymbol{b_2} \\ \boldsymbol{b_3} & \boldsymbol{b_4} \end{bmatrix} = \begin{bmatrix} \boldsymbol{a_1 b_1 + a_2 b_3} & \boldsymbol{a_1 b_2 + a_2 b_4} \\ \boldsymbol{a_3 b_1 + a_4 b_3} & \boldsymbol{a_3 b_2 + a_4 b_4} \end{bmatrix}$$

Matrix multiplication is performed by calculating the dot product of the corresponding row of matrix A and the corresponding column of matrix B.
If you understand that sentence, you understand matrix multiplication. If not, let's drive the point home with a simple example:

$$A = \begin{bmatrix} 3 & 4 \\ 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 2 & 2 \\ 1 & 2 \end{bmatrix}$$

$$AB = \begin{bmatrix} (3 \times 2) + (4 \times 1) & (3 \times 2) + (4 \times 2) \\ (1 \times 2) + (0 \times 1) & (1 \times 2) + (0 \times 1) \end{bmatrix}$$

$$AB = \begin{bmatrix} 6 + 4 & 6 + 8 \\ 2 + 0 & 2 + 0 \end{bmatrix}$$

$$AB = \begin{bmatrix} 10 & 14 \\ 2 & 2 \end{bmatrix}$$

```
import numpy as np
A = np.array([[3,4],[1,0]])
B = np.array([[2,2],[1,2]])
C = A.dot(B)
print(C, C.shape, C.ndim)
```

4. Matrix Transpose

And now something simple, to rest your brain for a minute. But just for a minute. Matrix transpose is one of those topics that sounds super fancy, particularly if you're not a native English speaker and you don't know what 'Transpose' means. The idea is really simple — you only need to **exchange rows and columns** of the matrix. Transpose operator is in most cases denoted with capital letter **T**, and notation can be put either before the matrix or as an exponent. Either way, here's the general formula:

$$\boldsymbol{T}\left(\begin{bmatrix} \boldsymbol{a_1} & \boldsymbol{a_2} \\ \boldsymbol{a_3} & \boldsymbol{a_4} \end{bmatrix}\right) = \begin{bmatrix} \boldsymbol{a_1} & \boldsymbol{a_3} \\ \boldsymbol{a_2} & \boldsymbol{a_4} \end{bmatrix}$$

As you can see the diagonal elements stayed the same, and those off-diagonal switched their position.
Here's a simple example with a 2x2 matrix:

$$A = \begin{bmatrix} 3 & 4 \\ 1 & 0 \end{bmatrix}$$

$$T(A) = \begin{bmatrix} 3 & 1 \\ 4 & 0 \end{bmatrix}$$

```
import numpy as np
A = np.array([[3,4],[1,0]])
B = np.array([[2,2],[1,2]])
C = A.dot(B)
print(C, C.shape, C.ndim)
```

5. Identity Matrix

Just as with transpose, Identity matrices are also really simple to grasp on. It is a matrix where:

1. Every diagonal element is 1
2. All the other elements are 0

And that's it! It's usually denoted with a capital letter **I**, and the number representing its size in a subscript.

Here's how the identity matrix of size 3 would look like:

$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
A = np.eye(3)
print(A)
```

6. Determinant
   (According to Wikipedia)
   The determinant is a scalar value that can be computed from the elements of a square matrix and encodes certain properties of the linear transformation described by the matrix.
   The determinant of a matrix A *is denoted det(*A*), det* A*, or* |A|. *Geometrically, it can be viewed as the volume scaling factor of the linear transformation described by the matrix*
   To develop a more intuitive sense of what the determinant is, and what it is used for, please refer to the video playlist linked down in the article conclusion section.
   The calculation process is simple for 2x2 matrix, get's a little more difficult for 3x3 matrices, and shouldn't be computed by hand for larger ones. I mean you can if you want to, but why? The goal here is to **develop the intuition**, computers were made to do the calculations.
   Here's the general formula for calculating the determinant of 2x2 matrix:

$$\boldsymbol{det \begin{bmatrix} a & b \\ c & d \end{bmatrix} = ad - bc}$$

   And to drive a point home here's the most basic example of calculation by hand:

$$A = \begin{bmatrix} 3 & 1 \\ 2 & 6 \end{bmatrix}$$

$$|A| = (3 \times 6) - (1 \times 2)$$

$$|A| = 18 - 2 = 16$$

```
import numpy as np

A = np.array([[3,1],[2,6]])

B = np.linalg.det(A)
```

```
print(B, B.shape, B.ndim)
```

7. Matrix Inverse

A square matrix is called invertible (or *nonsingular*) if **multiplication of the original matrix by its inverse results in the identity matrix**.
From that statement, you can conclude that not all matrices have inverses. For a matrix to be invertible, it has to satisfy the following conditions:

- Must be square
- The determinant cannot be 0

A matrix that isn't invertible is called a *singular* matrix. Logically, for square matrix to be singular, its determinant must be equal to 0. Let's see why by exploring the general formula:

$$A^{-1} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

As you can see the matrix inverse is denoted by this **-1** term in the superscript. The formula might already look really familiar to you — there's previously seen ***ad* — *bc*** term (the *determinant*). You can see here why the determinant cannot be 0 — **division by 0 is undefined**.

This term is then multiplied with the slightly rearranged version of the original matrix. The diagonal items are switched, and off-diagonal elements are multiplied by negative one (-1).

Here's a simple example of calculating the inverse of the 2x2 matrix:

$$A = \begin{bmatrix} 4 & 3 \\ 5 & 4 \end{bmatrix}$$

$$A^{-1} = \begin{bmatrix} 4 & 3 \\ 5 & 4 \end{bmatrix}^{-1} = \frac{1}{(4 \times 4) - (3 \times 5)} \begin{bmatrix} 4 & -3 \\ -5 & 4 \end{bmatrix}$$

$$A^{-1} = \frac{1}{16 - 15} \begin{bmatrix} 4 & -3 \\ -5 & 4 \end{bmatrix}$$

$$A^{-1} = \frac{1}{1} \begin{bmatrix} 4 & -3 \\ -5 & 4 \end{bmatrix}$$

$$A^{-1} = \begin{bmatrix} 4 & -3 \\ -5 & 4 \end{bmatrix}$$

```
import numpy as np
A = np.array([[4,3],[5,4]])
B = np.linalg.inv(A)
print(B, B.shape, B.ndim)
```

Now let’s verify the claim stated earlier, and that is that multiplication of the original matrix by its inverse yields the identity matrix:

$$\boldsymbol{A^{-1}A = I_n}$$

Here’s the example calculated by hand, and the statements holds true!

$$A^{-1} = \begin{bmatrix} 4 & -3 \\ -5 & 4 \end{bmatrix}, \quad A = \begin{bmatrix} 4 & 3 \\ 5 & 4 \end{bmatrix}$$

$$A^{-1}A = \begin{bmatrix} (4 \times 4) + (-3 \times 5) & (4 \times 3) + (-3 \times 4) \\ (-5 \times 4) + (4 \times 5) & (-5 \times 3) + (4 \times 4) \end{bmatrix}$$

$$A^{-1}A = \begin{bmatrix} 16 - 15 & 12 - 12 \\ -20 + 20 & -15 + 16 \end{bmatrix}$$

$$A^{-1}A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

```
print(A.dot(np.linalg.inv(A)))
```