

C1_W4_lecture_nb_01_vector_manipulation

December 12, 2023

1 Vector manipulation in Python

In this lab, you will have the opportunity to practice once again with the NumPy library. This time, we will explore some advanced operations with arrays and matrices.

At the end of the previous module, we used PCA to transform a set of many variables into a set of only two uncorrelated variables. This was done by means of a transformation of the data called *rotation*.

In this week's assignment, you will need to find a transformation matrix from English to French vector space embeddings. Such a transformation matrix is nothing else but a matrix that rotates and scales vector spaces.

In this notebook, we will explain in detail the rotation transformation.

1.1 Transforming vectors

There are three main vector transformations: * Scaling * Translation * Rotation

In previous notebooks, we applied the first two kinds of transformations. Now, let us learn how to use a fundamental transformation on vectors called *rotation*.

The rotation operation changes the direction of a vector, leaving unaffected its dimensionality and its norm. Let us explain this with some examples.

In the following cells, we will define a NumPy matrix and a column vector as a NumPy array. Soon we will explain how this is related to matrix rotation.

```
[ ]: import numpy as np                # Import numpy for array manipulation
      import matplotlib.pyplot as plt  # Import matplotlib for charts
      from utils_nb import plot_vectors # Function to plot vectors (arrows)
```

1.1.1 Example 1

```
[ ]: # Create a 2 x 2 matrix
      R = np.array([[-2, 0],
                    [0, 2]])
      R
```

```
[ ]: x = np.array([[1, 1]]) # Create a row vector as a NumPy array with a single row
x
```

The dot product between a square matrix and the transpose of a row vector produces a rotation and scaling of the original vector.

Remember that our recommended way to get the dot product in Python is `np.dot(a, b)`:

```
[ ]: y = np.dot(R, x.T) # Apply the dot product between R and x.T
y          # Column vector as a NumPy array with a single column
```

We are going to use Pyplot to visually inspect the effect of the rotation on 2D vectors. For that, we have created a function `plot_vectors()` that takes care of all the intricate parts of the visual formatting. The code for this function is inside the `utils_nb.py` file.

Now we can plot the vector $\vec{x} = [1, 1]$ in a cartesian plane. The cartesian plane will be centered at $[0, 0]$ and its x and y limits will be between $[-4, +4]$

```
[ ]: plot_vectors([x], axes=[4, 4], fname='transform_x.svg')
```

Now, let's plot in the same system our vector $\vec{x} = [1, 1]$ and the dot product of the matrix with $x.T$.

$$R = \begin{bmatrix} -2 & 0 \\ 0 & 2 \end{bmatrix}$$

$$y = R \cdot x.T$$

```
[ ]: plot_vectors([x, y], axes=[4, 4], fname='transformx_and_y.svg')
```

Note that the vector `x` (black) is transformed into vector `y` (blue).

1.1.2 Example 2

We are going to use Pyplot to visually inspect the effect of the rotation on 2D vectors. For that, we have created a function that takes care of all the intricate parts of the visual formatting. The following procedure plots an arrow within a Pyplot canvas.

Data that is composed of 2 real attributes belongs to a $\mathbb{R} \times \mathbb{R}$ or \mathbb{R}^2 space. Rotation matrices in \mathbb{R}^2 rotate a given vector \vec{x} by a counterclockwise angle θ in a fixed coordinate system. Rotation matrices are of the form:

$$R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

(**Note:** This notebook uses

$$y = R_\theta \cdot x.T$$

But if you use

$$y = x \cdot Ro$$

then the rotation matrices in R^2 rotate a given vector \vec{x} by a clockwise angle θ in a fixed coordinate system).

The trigonometric functions in Numpy require the angle in radians, not in degrees. In the next cell, we define a rotation matrix that rotates vectors counterclockwise by 100° .

```
[ ]: angle = 100 * (np.pi / 180) # Convert degrees to radians

Ro = np.array([[np.cos(angle), -np.sin(angle)],
               [np.sin(angle), np.cos(angle)]])

x2 = np.array([[2, 2]]) # Row vector as a NumPy array
y2 = np.dot(Ro, x2.T)

print('Rotation matrix')
print(Ro)
print('\nRotated vector')
print(y2)

print('\n x2 norm', np.linalg.norm(x2))
print('\n y2 norm', np.linalg.norm(y2))
print('\n Rotation matrix norm', np.linalg.norm(Ro))

[ ]: plot_vectors([x2, y2], fname='transform_02.svg')
```

Some points to note:

- The norm of the input vector is the same as the norm of the output vector. Rotation matrices do not modify the norm of the vector, only its direction.
- The norm of any R^2 rotation matrix is always $\sqrt{2} = 1.414221$

1.2 Frobenius Norm

The Frobenius norm is the generalization to R^2 of the already known norm function for vectors

$$\|\vec{a}\| = \sqrt{\vec{a} \cdot \vec{a}}$$

For a given R^2 matrix A, the frobenius norm is defined as:

$$\|A\|_F \equiv \sqrt{\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2}$$

```
[ ]: A = np.array([[2, 2],  
                  [2, 2]])
```

`np.square()` is a way to square each element of a matrix. Its outcome is equivalent to that of using the `*` operator with numpy arrays.

```
[ ]: A_squared = np.square(A)  
A_squared
```

```
[ ]: A_squared = A * A  
A_squared
```

Now you can sum over the elements of the resulting array, and then get the square root of the sum.

```
[ ]: A_Frobenius = np.sqrt(np.sum(A_squared))  
A_Frobenius
```

That was the extended version of the `np.linalg.norm()` function. You can check that it yields the same result.

```
[ ]: print('Frobenius norm of the Rotation matrix')  
print(np.sqrt(np.sum(Ro * Ro)), '== ', np.linalg.norm(Ro))
```

Congratulations!! We've covered a few more matrix operations in this lab. This will come in handy in this week's programming assignment!

A special thanks to **Reinoud Bosch** for his contributions in this notebook.

```
[ ]:
```