How do I normalize values in NumPy array between 0 and 1?

Authored by

stats writer

November 21, 2025

Normalization, specifically Min-Max scaling, is a fundamental technique in data preprocessing used extensively in data science and machine learning. When working with numerical data contained within a NumPy array, it is often critical to scale these features so they all fall within a standard range, typically 0 to 1. This process transforms the data such that the smallest value maps to 0 and the largest value maps to 1, with all other values scaled proportionally between these boundaries.

The Min-Max Scaling Formula

The robust method for scaling values to the range is Min-Max normalization. It relies on a specific mathematical transformation applied element-wise across the NumPy array, where the normalized value (z_i) is calculated based on the data point (x_i), the minimum ($min(x)$), and the maximum ($max(x)$) values of the entire feature set.

The generic formula for achieving this scaling is:

$z_i = (x_i - min(x)) / (max(x) - min(x))$

This approach ensures that the original minimum value is transformed to 0 and the original maximum value is transformed to 1, guaranteeing all results fall strictly within the desired bounds.

Implementation Method 1: Using Pure NumPy

To normalize the values in a NumPy array to be between 0 and 1, the most efficient method for simple, single-array transformations is using the core NumPy library itself. This method translates the Min-Max formula directly into vectorized array operations.

import numpy as np

x_norm = (x-np.min(x))/(np.max(x)-np.min(x))

This method is highly favored for its speed and minimal dependency footprint, requiring only the standard NumPy library.

Implementation Method 2: Utilizing Scikit-learn

For integration into larger machine learning pipelines, the use of the **MinMaxScaler** class from the Scikit-learn library is recommended. This class abstracts the normalization process, making it easier to manage scaling parameters across training and testing datasets.

from sklearn import preprocessing as pre

```
x = x.reshape(-1, 1)

x_norm = pre.MinMaxScaler().fit_transform(x)
```

Both methods assume **x** is the name of the NumPy array you would like to normalize. The following detailed examples show how to use each technique in practice, providing identical results but utilizing different coding philosophies.

Example 1: Normalize Values Using Pure NumPy

Suppose we have the following one-dimensional NumPy array representing a feature variable:

```
import numpy as np

#create NumPy array
x = np.array()
```

We can use the following code, applying the core Min-Max formula directly, to normalize each value in the array to be between 0 and 1:

```
#normalize all values to be between 0 and 1
x_norm = (x-np.min(x))/(np.max(x)-np.min(x))

#view normalized array
print(x_norm)
```

Each value in the NumPy array has been successfully normalized to the range. The minimum value of 13 maps to 0.0, and the maximum value of 71 maps to 1.0.

Mathematical Verification of Normalization

To understand the mechanism fully, we verify the transformation of individual data points. In this dataset, the minimum value is 13 and the maximum value is 71. Thus, the range ($max(x) - min(x)$) is $71 - 13 = 58$.

The calculations for the first few elements are as follows:

To normalize the first value of **13**:

$z_1 = (13 - 13) / (71 - 13) = 0 / 58 = 0$

To normalize the second value of **16**:

$z_2 = (16 - 13) / (71 - 13) = 3 / 58 approx 0.0517$

To normalize the third value of **19**:

$z_3 = (19 - 13) / (71 - 13) = 6 / 58 approx 0.1034$

This formula is consistently applied across the entire array, resulting in the desired Min-Max normalization.

Example 2: Normalize Values Using Scikit-learn

We will use the same input array to demonstrate the Scikit-learn approach. This method is preferred when integrating scaling into complex machine learning pipelines.

import numpy as np

#create NumPy array
x = np.array()

We must use the **MinMaxScaler()** function from sklearn. Crucially, Scikit-learn preprocessing functions require the input data to be two-dimensional (samples by features), necessitating the use of `reshape(-1, 1)` for our 1D NumPy array:

from sklearn import preprocessing as pre

#reshape array so that it works with sklearn
x = x.reshape(-1, 1)

#normalize all values to be between 0 and 1
x_norm = pre.MinMaxScaler().fit_transform(x)

#view normalized array
print(x_norm)

]

The normalized values produced by the **MinMaxScaler** are identical to those generated by the pure NumPy formula, confirming that the underlying mathematical operation for Min-Max scaling remains consistent across implementation styles.

Comparison: NumPy vs. Scikit-learn

Choosing between the two methods often depends on the project scope:

The **NumPy method** is faster for isolated transformations and avoids external dependencies, making it ideal for prototyping or environments focused purely on numerical processing.

The **Scikit-learn method** is more robust for machine learning workflows, as the scaler object (instance of **MinMaxScaler**) maintains the parameters learned during the `fit` process. This ensures that when new data (such as a test set) undergoes data preprocessing, it is scaled consistently using the original training data's minimum and maximum values.

Further Reading on Data Scaling

While Min-Max normalization is effective for scaling data when the underlying distribution is unknown or when bounds are critical, practitioners must be aware of its sensitivity to outliers. In scenarios where extreme values significantly skew the data range, alternative methods like Z-score standardization (which centers the data around a mean of 0 with a standard deviation of 1) may be more appropriate for improving machine learning model performance.

The following tutorials explain how to perform other common tasks in NumPy: