NumPy

Prof. Pranay Kumar Saha

January 29, 2025

What is NumPy?

NumPy (Numerical Python) is the fundamental library for numerical computing in Python.

- Provides the powerful `ndarray` data structure for **fast**, **vectorized** computations.
- Offers tools for **linear algebra**, **Fourier transforms**, and **random number generation**.
- Forms the core of the scientific Python ecosystem (used by pandas, SciPy, scikit-learn, etc.).

Array Creation

- From Python lists using `np.array()`.
- Built-in functions like `np.zeros()`, `np.ones()`, `np.arange()`, `np.linspace()`.

```
import numpy as np

arr_from_list = np.array([1, 2, 3])
zeros_arr = np.zeros((2, 3))
ones_arr  = np.ones((3, 2))
range_arr = np.arange(0, 10, 2) # [0 2 4 6 8]
linspace_arr = np.linspace(0, 1, 5) # [0. 0.25 0.5 0.75 1.]
```

Array Attributes

- `shape` : tuple of array dimensions
- `dtype` : data type (e.g. `float64`, `int32`)
- `size` : total number of elements
- `nbytes`: total bytes used

```
example = np.random.rand(4, 5)
print("Shape:", example.shape)
print("dtype:", example.dtype)
print("Size:",  example.size)
print("Bytes:", example.nbytes)
```

Basic Slicing

```
arr = np.arange(10)
print(arr[2:6])   # elements from index 2 to 5
print(arr[:5])    # first 5 elements
print(arr[::2])   # every 2nd element
```

Advanced Indexing

Integer (list) indexing:

```
idx = [0, 2, 4]
arr[idx]  # picks out arr[0], arr[2], arr[4]
```

Boolean indexing:

```
mask = (arr % 2 == 0)  # even
even_vals = arr[mask]
```

Vectorized Operations vs. Python Loops

```
import time
large_arr = np.random.rand(10_000_000)

# Python loop
start = time.time()
py_sum = 0.0
for v in large_arr:
    py_sum += v
loop_time = time.time() - start

# NumPy sum
start = time.time()
np_sum = np.sum(large_arr)
numpy_time = time.time() - start
```

Broadcasting

NumPy **broadcasting** allows operations between arrays of different shapes, following specific rules. For example, adding a scalar to a matrix or adding a 1D array to a 2D array, as long as dimensions are compatible.

Broadcasting Rules

1. The two arrays are compatible if they have the same shape or if one of them has a dimension of size 1 (or is missing that dimension).
2. When one dimension is 1 and the other is larger, NumPy will virtually ”stretch” that dimension.

Broadcasting (contd..)

Broadcasting allows operations between arrays of different shapes, if dimensions are compatible.

$$A = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \quad B = \begin{bmatrix} 10 & 20 & 30 \end{bmatrix}$$

```
A = np.array([[1],[2],[3]])  # shape (3,1)
B = np.array([10, 20, 30])   # shape (3,)
C = A + B  # Broadcasting
print(C)
```

Linear Algebra Basics

- Dot product: `np.dot(A,B)` or `A @ B`.
- Transpose: `A.T`.
- Inverse: `np.linalg.inv(A)`.
- Eigenvalues/eigenvectors: `np.linalg.eig(A)`.

Solve $A\mathbf{x} = \mathbf{b} \quad \rightarrow \quad \mathbf{x} = A^{-1}\mathbf{b}$

Solving a System of Equations

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

x = np.linalg.solve(A, b)
print("Solution:", x)
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$$

Statistical Methods

- `np.mean(a)`, `np.std(a)`, `np.var(a)`
- `np.min(a)`, `np.max(a)`, `np.median(a)`
- `np.percentile(a, q)`

```
data = np.random.randn(1000)
print("Mean:", np.mean(data))
print("Std:", np.std(data))
```

Understanding the `axis` Parameter in NumPy

Axis refers to the dimension along which an operation is performed.

- A 2D array has shape (m, n):
  - `axis=0` operates *down* the rows (collapses rows).
  - `axis=1` operates *across* the columns (collapses columns).
- Example: Summation in a 2D array

$$\begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \\ a_{20} & a_{21} \end{bmatrix}$$

  - $\sum(\ldots, \texttt{axis=0})$ → sum each column
  - $\sum(\ldots, \texttt{axis=1})$ → sum each row

axis in Practice (Example)

```
import numpy as np

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

print("Sum over axis=0 (columns):", np.sum(arr, axis=0))
# -> [15 18 21 24]

print("Sum over axis=1 (rows)   :", np.sum(arr, axis=1))
# -> [10 26 42]
```

Random Sampling

```
np.random.seed(42)
r1 = np.random.rand(3,3)      # uniform
r2 = np.random.randn(3,3)     # normal
r3 = np.random.randint(0,10,(3,3))  # integers
```

Physics: Projectile Motion

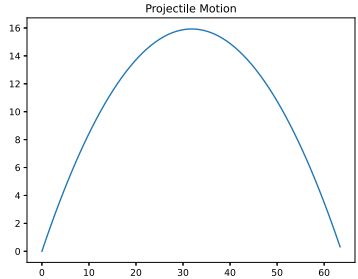
$$x(t) = v_0 \cos(\theta)\, t, \quad y(t) = v_0 \sin(\theta)\, t - \frac{1}{2} g t^2$$

```
import matplotlib.pyplot as plt
g = 9.81
v0 = 25
theta = np.radians(45)
t = np.linspace(0, 5, 100)
x = v0*np.cos(theta)*t
y = v0*np.sin(theta)*t - 0.5*g*t**2
mask = y >= 0
plt.plot(x[mask], y[mask])
plt.title("Projectile Motion")
plt.show()
```

Physics: Projectile Motion

$$x(t) = v_0 \cos(\theta)\, t, \quad y(t) = v_0 \sin(\theta)\, t - \frac{1}{2} g t^2$$

```
import matplotlib.pyplot as plt
g = 9.81
v0 = 25
theta = np.radians(45)
t = np.linspace(0, 5, 100)
x = v0*np.cos(theta)*t
y = v0*np.sin(theta)*t - 0.5*g*t**2
mask = y >= 0
plt.plot(x[mask], y[mask])
plt.title("Projectile Motion")
plt.show()
```



Data Analysis: Histogram

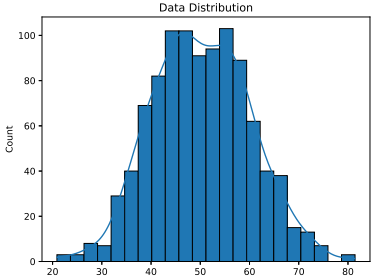
```
import seaborn as sns

data = np.random.normal(loc=50, scale=10,
        size=1000)
sns.histplot(data, kde=True)
plt.title("Data Distribution")
plt.show()
```

Data Analysis: Histogram

```
import seaborn as sns

data = np.random.normal(loc=50, scale=10,
        size=1000)
sns.histplot(data, kde=True)
plt.title("Data Distribution")
plt.show()
```



Performance and Memory

- **Vectorization**: Avoid Python loops for large array operations.
- **Views vs Copies**: Slicing creates **views**, which are more memory-efficient.
- For bigger optimizations:
  - Use `Numba` (JIT compiler) or `Cython`.
  - Consider `Dask` for parallel/distributed computing.

What is PCA?

- **Goal:** Reduce the dimensionality of a dataset while retaining the majority of its variance.
- Finds new axes (principal components) that capture maximum variance in the data.
- Useful for:
  - Data visualization (e.g., projecting to 2D or 3D).
  - Noise reduction and feature extraction.
  - Mitigating overfitting and reducing computation in ML pipelines.

Steps of PCA

1. Center the Data

- Subtract the mean of each feature (column) so each feature has mean zero.

2. Compute Covariance (or use SVD)

- Covariance matrix: $C = \frac{1}{N-1} X^{\square} X$.
- SVD: $X = U \Sigma V^{\square}$.

3. Eigen-decomposition

- Solve $C v_i = \lambda_i v_i$.
- λ_i = variance explained by v_i.

Steps of PCA (continued)

4. Sort and Select Principal Components

- Order eigenvalues λ_i in descending order.
- Choose top k components based on largest λ_i.

5. Project the Data

- $X_{\text{reduced}} = X \cdot [v_1, v_2, \ldots, v_k]$.
- New representation has reduced dimensionality (k).

6. Interpret Results

- Check *explained variance ratio* to see how much information is retained.
- Use lower-dimensional X_{reduced} for:
  - Visualization.
  - Machine learning.
  - Noise reduction.

PCA from Scratch (Brief Overview)

1. Center data by subtracting the mean.
2. Compute covariance matrix.
3. Compute eigenvalues and eigenvectors.
4. Sort eigenvectors by descending eigenvalues.
5. Project data onto top components.

PCA Function in NumPy

```
def pca(X, n_components=2):
    X_mean = np.mean(X, axis=0)
    X_centered = X - X_mean
    cov = np.cov(X_centered, rowvar=False)
    eigvals, eigvecs = np.linalg.eigh(cov)
    idx = np.argsort(eigvals)[::-1]
    eigvals = eigvals[idx]
    eigvecs = eigvecs[:, idx]
    pcs = eigvecs[:, :n_components]
    X_pca = X_centered @ pcs
    return X_pca, pcs, X_mean
```

Key Takeaways

- NumPy is the **backbone** of scientific Python.
- **ndarray** structures: **fast** and **efficient**.
- Master indexing, slicing, and vectorization to write concise, high-performance code.
- Explore further: linear algebra, FFTs, random sampling, advanced broadcasting.

NumPy

Thank You for Listening!