NumPy

1. ***import numpy as np***
   For importing the NumPy library

2. ***arr = np.array([1, 2, 3, 4])***
   For creating a NumPy array from a list

3. ***np.zeros((3, 3))***
   For creating a 3x3 array filled with zeros

4. ***np.ones((2, 2))***
   For creating a 2x2 array filled with ones

5. ***np.empty((3, 3))***
   For creating an empty array (values are random)

6. ***np.arange(0, 10, 2)***
   For creating an array with values from 0 to 9, spaced by 2

7. ***np.linspace(0, 1, 5)***
   For creating an array of 5 equally spaced values between 0 and 1

8. ***arr.shape***
   For getting the shape (dimensions) of the array

9. ***arr.reshape(2, 2)***
   For reshaping the array to a specified shape

10. ***arr.dtype***
    For getting the data type of the array elements

11. ***arr.size***
    For getting the number of elements in the array

12. ***arr1 + arr2***
    For element-wise addition of two arrays

13. ***arr1 * arr2***
    For element-wise multiplication of two arrays

14. ***np.dot(arr1, arr2)***
    For performing a dot product between two arrays

15. ***arr.T***
    For transposing an array

16. ***np.mean(arr)***
    For calculating the mean of the array

17. ***np.median(arr)***
    For calculating the median of the array

18. ***np.std(arr)***
    For calculating the standard deviation of the array

19. ***np.sum(arr)***
    For calculating the sum of the array

20. ***np.min(arr)*** and ***np.max(arr)***
    For finding the minimum and maximum values in the array

21. ***arr[arr > 5]***
    For filtering elements in the array that are greater than 5

22. ***np.concatenate((arr1, arr2))***
    For concatenating two arrays along an axis

23. ***np.split(arr, 2)***
    For splitting the array into sub-arrays

NumPy For Machine Learning

22 most commonly used NumPy functions in machine learning, along with examples and their usage in machine learning:

1. np.array()

- - -: Creates a NumPy array from Python lists or tuples.

- - -: Arrays are the core data structure for storing datasets, features, and labels in machine learning.

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr)
```

Output: [1 2 3 4]

2. np.zeros() & np.ones()

- - -: Creates arrays filled with zeros or ones.

- - -: Used to initialize parameters like weights and biases in neural networks or placeholders for large datasets.

```
zeros_array = np.zeros((2, 3))
ones_array = np.ones((2, 3))
print(zeros_array, ones_array)
```

Output:

```
[[0. 0. 0.]
 [0. 0. 0.]]
[[1. 1. 1.]
 [1. 1. 1.]]
```

3. np.random.rand()

- - -: Generates an array of random numbers from a uniform distribution between 0 and 1.
- - -: Often used to initialize random weights for models.

```
rand_array = np.random.rand(3, 3)
print(rand_array)
```

Output:

```
[[0.845 0.235 0.678]
 [0.123 0.789 0.456]
 [0.111 0.564 0.912]]
```

4. np.random.randn()

- - -: Generates an array of random numbers from a normal (Gaussian) distribution.
- - -: Used to initialize weights when a Gaussian distribution is needed (common in neural networks).

```
randn_array = np.random.randn(3, 3)
print(randn_array)
```

Output:

```
[[ 0.978 -0.574  1.234]
 [-1.456  0.456 -0.678]
 [ 0.345 -0.987  0.789]]
```

5. np.dot()

- - -: Performs the dot product of two arrays.
- - -: Core operation in linear algebra, especially for calculating predictions in linear regression, neural networks, and other models.

```
a = np.array([1, 2])
b = np.array([4, 5])
dot_product = np.dot(a, b)
print(dot_product)
```

Output: 14

6. np.matmul()

- - -: Performs matrix multiplication.

- - -: Used for more complex matrix operations, such as when combining weight matrices in deep learning models.

```
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
matmul_result = np.matmul(A, B)
print(matmul_result)
```

Output:

[[19 22]
 [43 50]]

7. np.sum()

- - -: Sums the elements of an array along a given axis.

- - -: Used to compute the cost function in regression or summing activation outputs in neural networks.

```
arr = np.array([1, 2, 3, 4])
total_sum = np.sum(arr)
print(total_sum)
```

Output: 10

8. np.mean()

- - -: Calculates the mean of elements along a given axis.

- - -: Commonly used in feature scaling and normalization, as well as computing average metrics like accuracy and loss.

```
arr = np.array([1, 2, 3, 4])
mean_value = np.mean(arr)
print(mean_value)
```

Output: 2.5

9. np.var()

- - -: Computes the variance of elements along a given axis.
- - -: Used in feature scaling, normalization, and during statistical analysis of datasets.

```
arr = np.array([1, 2, 3, 4])
variance = np.var(arr)
print(variance)
```

Output: 1.25

10. np.std()

- - -: Calculates the standard deviation of elements along a given axis.
- - -: Useful in feature scaling (standardization) and understanding data spread in statistical modeling.

```
arr = np.array([1, 2, 3, 4])
std_dev = np.std(arr)
print(std_dev)
```

Output: 1.118

11. np.argmax()

- - -: Returns the index of the maximum value along an axis.
- - -: Used to identify the class label with the highest predicted probability in classification tasks.

```
arr = np.array([1, 2, 3, 7, 4])
max_index = np.argmax(arr)
print(max_index)
```

Output: 3

12. np.argmin()

- - -: Returns the index of the minimum value along an axis.
- - -: Can be used to find errors, minimum values in cost functions, etc.

```
arr = np.array([1, 2, 3, 0, 4])
min_index = np.argmin(arr)
print(min_index)
```

Output: 3

13. np.reshape()

- - -: Reshapes an array without changing its data.

- - -: Essential for transforming the data into the correct shape for input into machine learning models (e.g., reshaping flattened images for CNNs).

```
arr = np.array([1, 2, 3, 4, 5, 6])
reshaped_arr = np.reshape(arr, (2, 3))
print(reshaped_arr)
```

Output:

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

14. np.expand_dims()

- - -: Expands the dimensions of an array.

- - -: Often used when preparing data for convolutional neural networks (CNNs).

```
arr = np.array([1, 2, 3, 4])
expanded_arr = np.expand_dims(arr, axis=0)
print(expanded_arr)
```

Output: [[1 2 3 4]]

15. np.concatenate()

- - -: Joins two or more arrays along a specified axis.

- - -: Useful for merging training data or combining predictions from different models.

```
arr1 = np.array([1, 2])
arr2 = np.array([3, 4])
concatenated_arr = np.concatenate((arr1, arr2))
print(concatenated_arr)
```

Output: [1 2 3 4]

16. np.split()

- - -: Splits an array into multiple sub-arrays.
- - -: Used to split datasets into training, validation, and test sets.

```
arr = np.array([1, 2, 3, 4, 5, 6])
split_arr = np.split(arr, 2)
print(split_arr)
```

Output: [array([1, 2, 3]), array([4, 5, 6])]

17. np.linalg.inv()

- - -: Computes the inverse of a matrix.
- - -: Inverse matrices are used in solving linear equations, especially in closed-form solutions for linear regression.

```
matrix = np.array([[1, 2], [3, 4]])
inverse_matrix = np.linalg.inv(matrix)
print(inverse_matrix)
```

Output:

```
[[-2.   1. ]
 [ 1.5 -0.5]]
```

18. np.linalg.eig()

- - -: Computes the eigenvalues and eigenvectors of a matrix.
- - -: Useful in Principal Component Analysis (PCA) for dimensionality reduction.

```
matrix = np.array([[1, 2], [2, 1]])
eigenvalues, eigenvectors = np.linalg.eig(matrix)
print(eigenvalues, eigenvectors)
```

Output:

```
[ 3. -1.]
[[ 0.707  0.707]
 [ 0.707 -0.707]]
```

19. np.linalg.norm()

- - -: Computes the norm (magnitude) of vectors or matrices.
- - -: Used in regularization techniques (like L2 regularization).

```
vector = np.array([3, 4])
norm_value = np.linalg.norm(vector)
print(norm_value)
```

Output: 5.0

20. np.clip()

- - -: Limits the values in an array to within a specified range.
- - -: Often used to limit the output of activation functions or to prevent outliers from skewing results.

```
arr = np.array([1, 5, 10, 15])
clipped_arr = np.clip(arr, 3, 12)
print(clipped_arr)
```

Output: [3 5 10 12]

21. np.unique()

- - -: Finds the unique elements in an array and returns them in sorted order.
- - -: Helpful for analysing categorical data or understanding the distribution of class labels.

```
arr = np.array([1, 2, 2, 3, 4, 4, 5])
unique_elements = np.unique(arr)
print(unique_elements)
```

Output: [1 2 3 4 5]

22. np.where()
- - -: Returns elements from an array based on a condition.
- - -: Useful for replacing values or creating binary masks, often used in decision-based algorithms like decision trees.

```
arr = np.array([1, 2, 3, 4, 5])
result = np.where(arr > 3, arr, 0)
print(result)
```

Output: [0 0 0 4 5]

These examples demonstrate the key NumPy functions used in machine learning. They help with data manipulation, feature scaling, matrix operations, and statistical analysis, all critical in machine learning workflows.