An Essential Guide to Numpy for Machine Learning in Python

The Quintessential Library for ML!

Dec 18, 2018 ·

Why would this be useful to you?
Well since most of us tend to forget(In case of those already who already implemented ML algorithms) the various library functions and end up writing code for pre-existing functions using sheer logic which is a waste of both time and energy, in such times it becomes essential if one understands the nuances of the Library being used efficiently. So Numpy being one of the essential libraries for Machine Learning requires an article of its own.

Since understanding Numpy is the starting point of Data Pre-processing and later on implementing ML Algorithms, So you can be someone who is about to learn Machine Learning in the near future or has just begun and wants to get a more Hands on experience in learning Numpy for ML.

Trending AI Articles:

1. Ten trends of Artificial Intelligence (AI) in 2019

2. Bursting the Jargon bubbles — Deep Learning

3. How Can We Improve the Quality of Our Data?

4. Machine Learning using Logistic Regression in Python with Code

But my main focus while writing this article is for it to serve as a quick refresher to

Numpy for those who have had experience with the library but need a swift recap.

What are we waiting for ?Let's begin!!

Numpy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

Moreover Numpy forms the foundation of the Machine Learning stack. In this article we cover the most frequently used Numpy operartions.

1) Creating a Vector

Here we use Numpy to create a 1-D Array which we then call a vector.

```
#Load Library
import numpy as np

#Create a vector as a Row
vector_row = np.array([1,2,3])

#Create vector as a Column
vector_column = np.array([[1],[2],[3]])
```

2) Creating a Matrix

We Create a 2-D Array in Numpy and call it a Matrix. It contains 2 rows and 3 columns.

```
#Load Library
import numpy as np

#Create a Matrix
matrix = np.array([[1,2,3],[4,5,6]])
print(matrix)
```

3) Creating a Sparse Matrix

Given data with very few non zero values you want to efficiently represent it.

A frequent situation in Machine Learning is having a huge amount of Data; however most of the elements in the data are Zeros. For example, lets imagine a Matrix where the columns are all the products on Amazon and the rows signify whether a given user has bought that item before or not. Now as you might have guessed there would be many products which haven't been bought even a single time till now and thus a vast majority of elements would be Zero.

Sparse Matrices store only non zero elements and assume all other values will be zero, leading to significant computational savings.

```
#Load Library
import numpy as np

#Create a Matrix
matrix = np.array([[0,0],[0,1],[3,0]])
print(matrix)#Create Compressed Sparse Row(CSR) matrix
matrix_sparse = sparse.csr_matrix(matrix)
print(matrix_sparse)
```

4) Selecting Elements

When you need to select one or more element in a vector or matrix

```
#Load Library
import numpy as np

#Create a vector as a Row
vector_row = np.array([ 1,2,3,4,5,6 ])

#Create a Matrix
matrix = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(matrix)

#Select 3rd element of Vector
print(vector_row[2])
```

```
#Select 2nd row 2nd column
print(matrix[1,1])#Select all elements of a vector
print(vector_row[:])#Select everything up to and including the
3rd element
print(vector_row[:3])#Select the everything after the 3rd
element
print(vector_row[3:])#Select the last element
print(vector[-1])#Select the first 2 rows and all the columns
of the matrix
print(matrix[:2,:])#Select all rows and the 2nd column of the
matrix
print(matrix[:,1:2])
```

5) Describing a Matrix

When you want to know about the shape size and dimensions of a Matrix.

```
import numpy as np


#Create a Matrix
matrix = np.array([[1,2,3],[4,5,6],[7,8,9]])#View the Number
of Rows and Columns
print(matrix.shape)#View the number of elements (rows*columns)
print(matrix.size)#View the number of Dimensions(2 in this
case)
print(matrix.ndim)
```

6) Applying operations to elements

You want to apply some function to multiple elements in an array.

Numpy's vectorize class converts a function into a function that can apply to multiple elements in an array or slice of an array.

```
#Load Library
import numpy as np

#Create a Matrix
matrix = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(matrix)

#Create a function that adds 100 to something
add_100 =lambda i: i+100

#Convert it into a vectorized function
vectorized_add_100= np.vectorize(add_100)

#Apply function to all elements in matrix
print(vectorized_add_100(matrix))
```

7) Finding the max and min values

We use Numpy's max and min functions:

```
#Load Library
import numpy as np

#Create a Matrix
matrix = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(matrix)#Return the max element
print(np.max(matrix))#Return the min element
print(np.min(matrix))#To find the max element in each column
print(np.max(matrix,axis=0))#To find the max element in each
row
print(np.max(matrix,axis=1))
```

8) Calculating Average, Variance and Standard deviation

When you want to calculate some descriptive statistics about an array.

```
#Load Library
import numpy as np

#Create a Matrix
matrix = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

```
print(matrix)#Mean
print(np.mean(matrix))#Standard Dev.
print(np.std(matrix))#Variance
print(np.var(matrix))
```

9) Reshaping Arrays

When you want to reshape an array(changing the number of rows and columns) without changing the elements.

```
#Load Library
import numpy as np

#Create a Matrix
matrix = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(matrix)#Reshape
print(matrix.reshape(9,1))#Here -1 says as many columns as
needed and 1 row
print(matrix.reshape(1,-1))#If we provide only 1 value Reshape
would return a 1-d array of that
lengthprint(marix.reshape(9))#We can also use the Flatten
method to convert a matrix to 1-d arrayprint(matrix.flatten())
```

10) Transposing a vector or a Matrix

By transposing you interchange the rows and columns of a Matrix

```
#Load Library
import numpy as np

#Create a Matrix
matrix = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(matrix)#Transpose the matrix
print(matrix.T)
```

11) Finding the Determinant and Rank of a Matrix

The rank of a Matrix is the number of dimensions of the vector space spanned by its rows or columns.

```
#Load Library
import numpy as np
```

```
#Create a Matrix
matrix = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(matrix)#Calculate the Determinant
print(np.linalg.det(matrix))#Calculate the Rank
print(np.linalg.matrix_rank(matrix))
```

12) Getting the Diagonal of a Matrix

When you need to extract only the diagonal elements of a matrix

```
#Load Library
import numpy as np

#Create a Matrix
matrix = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(matrix)#Print the Principal diagonal
print(matrix.diagonal())#Print the diagonal one above the
Principal diagonal
print(matrix.diagonal(offset=1))#Print the diagonal one below
Principal diagonal
print(matrix.diagonal(offset=-1))
```

13) Calculating the trace of a Matrix

Trace of a Matrix is the sum of elements on the Principal Diagonal of the Matrix.

```
#Load Library
import numpy as np

#Create a Matrix
matrix = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(matrix)#Print the Trace
print(matrix.trace())
```

14) Finding Eigenvalues and Eigenvectors

Eigenvectors are widely used in Machine Learning libraries. Intutively given a linear transformation represented by a matrix,A, eigenvectors are vectors that when that transformation is applied, change only in scale(not direction).More formally

Av=Kv

Here A is a square matrix, K contains the eigenvalues and v contains the eigenvectors.

```
#Load Library
import numpy as np

#Create a Matrix
matrix = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(matrix)

# Calculate the Eigenvalues and Eigenvectors of that Matrix
eigenvalues ,eigenvectors=np.linalg.eig(matrix)
print(eigenvalues)
print(eigenvectors)
```

15) Calculating Dot Products

```
#Load Library
import numpy as np

#Create vector-1
vector_1 = np.array([ 1,2,3 ])#Create vector-2
vector_1 = np.array([ 4,5,6 ])#Calculate Dot Product
print(np.dot(vector_1,vector_2))#Alternatively you can use @
to calculate dot products
print(vector_1 @ vector_2)
```

16) Adding, Subtracting and Multiplying Matrices

```
#Load Library
import numpy as np

#Create Matrix-1
matrix_1 = np.array([[1,2,3],[4,5,6],[7,8,9]])#Create Matrix-2
matrix_2 = np.array([[7,8,9],[4,5,6],[1,2,3]])#Add the 2
Matrices
print(np.add(matrix_1,matrix_2))#Subtraction
```

```
print(np.subtract(matrix_1,matrix_2))#Multiplication(Element
wise, not Dot Product)
print(matrix_1*matrix_2)
```

17) Inverting a Matrix

This is used when you want to calculate the inverse of a Square Matrix

```
#Load Library
import numpy as np

#Create a Matrix
matrix = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(matrix)#Calculate its inverse
print(np.linalg.inv(matrix))
```

18) Generating Random values

Numpy offers a wide variety of means to generate Random Numbers.
Moreover, It can sometimes be useful to return the same random numbers to get predictable, repeatable results. We can do so by setting the 'Seed' (An Integer) of the pseudorandom generator. Random processes with the same seed would always produce the same result.

```
#Load Library
import numpy as np#Set seed
np.random.seed(1)#Generate 3 random integers b/w 1 and 10
print(np.random.randint(0,11,3))#Draw 3 numbers from a normal
distribution with mean 1.0 and std 2.0
print(np.random.normal(1.0,2.0,3))
```