GOBIERNO DE
EL SALVADOR

SECRETARÍA DE
INNOVACIÓN

CIENCIA DE DATOS

Agenda
Sesión 9/18

Numpy

- ¿Qué es Numpy?
- ¿Por qué utilizar NumPy?
- Creando matrices con NumPy
- Tipos de datos NumPy
- Modificación y Manipulación de estructura de una matriz
- Filtros para una matriz
- NumPy Random

¿Qué es Numpy?

NumPy es una biblioteca de Python que se utiliza para trabajar con matrices.

También tiene funciones para trabajar en el dominio de álgebra lineal, transformada de Fourier y matrices.

NumPy fue creado en 2005 por Travis Oliphant. Es un proyecto de código abierto y puedes usarlo libremente.

NumPy son las siglas de Numerical Python.

¿Qué es Numpy?

Instalación de NumPy

Si ya tiene Python y PIP instalados en un sistema, la instalación de NumPy es muy fácil.

Instálelo usando este comando:

pip install numpy

¿Qué es Numpy?

Importar NumPy

Una vez que NumPy esté instalado, impórtelo en sus aplicaciones agregando la importpalabra clave:

```
import numpy
```

Al ejecutar el código solo con la librería si no presenta un mensaje de erro significa que la paquetería ha sido reconocida por python

¿Qué es Numpy?

Código en python:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)
```

¿Por qué utilizar NumPy?

En Python tenemos listas que sirven como matrices, pero son lentas de procesar.

NumPy tiene como objetivo proporcionar un objeto de matriz que sea hasta 50 veces más rápido que las listas tradicionales de Python.

El objeto de matriz en NumPy se llama ndarray, proporciona muchas funciones de apoyo que facilitan el trabajo ndarray.

Las matrices se utilizan con mucha frecuencia en la ciencia de datos, donde la velocidad y los recursos son muy importantes.

¿Por qué utilizar NumPy?

¿Por qué NumPy es más rápido que las listas?

Los arreglos NumPy se almacenan en un lugar continuo en la memoria a diferencia de las listas, por lo que los procesos pueden acceder a ellos y manipularlos de manera muy eficiente.

Este comportamiento se denomina localidad de referencia en informática.

Esta es la razón principal por la que NumPy es más rápido que las listas. También está optimizado para trabajar con las últimas arquitecturas de CPU.

¿Por qué utilizar NumPy?

¿En qué idioma está escrito NumPy?

NumPy es una biblioteca de Python y está escrito parcialmente en Python, pero la mayoría de las partes que requieren un cálculo rápido están escritas en C o C ++.

¿Dónde está la base de código de NumPy?

El código fuente de NumPy se encuentra en este repositorio de github **https://github.com/numpy/numpy**

¿Por qué utilizar NumPy?

¿En qué idioma está escrito NumPy?

NumPy es una biblioteca de Python y está escrito parcialmente en Python, pero la mayoría de las partes que requieren un cálculo rápido están escritas en C o C ++.

¿Dónde está la base de código de NumPy?

El código fuente de NumPy se encuentra en este repositorio de github **https://github.com/numpy/numpy**

Creando matrices con NumPy

Crear un objeto ndarray de NumPy

NumPy se utiliza para trabajar con matrices.

Se llama al objeto de matriz en NumPy **ndarray**.

Podemos crear un objeto ndarray NumPy usando la función array().

Creando matrices con NumPy

Dimensiones en matrices

Una dimensión en matrices es un nivel de profundidad de matriz (matrices anidadas).

Matriz anidada: son matrices que tienen matrices como elementos.

Creando matrices con NumPy

Matrices 0-D

Las matrices 0-D, o escalares, son los elementos de una matriz. Cada valor de una matriz es una matriz 0-D.

```
import numpy as np
arr = np.array(42)
print(arr)
```

Creando matrices con NumPy

Matrices 1-D

Una matriz que tiene matrices 0-D como sus elementos se llama matriz unidimensional o 1-D.

Estos son los arreglos más comunes y básicos.

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

Creando matrices con NumPy

Matrices 2-D

Una matriz que tiene matrices 1-D como sus elementos se denomina matriz 2-D.

A menudo se utilizan para representar tensores de matriz o de segundo orden.

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr)
```

Creando matrices con NumPy

Matrices 3-D

Una matriz que tiene matrices 2-D (matrices) como sus elementos se llama matriz 3-D.

A menudo se utilizan para representar un tensor de tercer orden.

```
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(arr)
```

Creando matrices con NumPy

Matrices de dimensiones superiores

Una matriz puede tener cualquier número de dimensiones.

Cuando se crea la matriz, puede definir el número de dimensiones utilizando el argumento **ndmin**.

```
import numpy as np

arr = np.array([1, 2, 3, 4], ndmin=5)

print(arr)
print('number of dimensions :', arr.ndim)
```

Creando matrices con NumPy

¿Verifique el número de dimensiones?

NumPy Arrays proporciona el ndimatributo que devuelve un número entero que nos dice cuántas dimensiones tiene la matriz.

```
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

Creando matrices con NumPy

Acceso a elementos de matriz

La indexación de matrices es lo mismo que acceder a un elemento de matriz.

Puede acceder a un elemento de matriz consultando su número de índice.

Los índices en las matrices NumPy comienzan con 0, lo que significa que el primer elemento tiene un índice 0 y el segundo tiene un índice 1, etc.

Creando matrices con NumPy

Acceso a elementos de matriz

```
import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr[0])
```

Creando matrices con NumPy

Matrices de corte

Cortar en python significa tomar elementos de un índice dado a otro índice dado. Pasamos un corte en lugar de índice de la siguiente manera: .[start:end]

También podemos definir el paso, como esto: .[start:end:step]

Si no pasamos el inicio se considera 0, si no pasamos end su longitud considerada de matriz en esa dimensión, si no pasamos el paso se considera 1

Creando matrices con NumPy

Matrices de corte

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5])

arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[4:])
```

Creando matrices con NumPy

Matrices de corte

```
import numpy as np


arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5])
#Divida los elementos del índice 1 al índice 5 de la siguiente matriz


arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[4:])
#Corta los elementos desde el índice 4 hasta el final de la matriz
```

Creando matrices con NumPy

División negativa de una Matriz

Utilice el operador menos para hacer referencia a un índice desde el final:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[-3:-1])

#Corte desde el índice 3 desde el final hasta el índice 1 desde el final
```

Tipos de datos NumPy

Por defecto, Python tiene estos tipos de datos:

strings- utilizado para representar datos de texto, el texto se da entre comillas. p. ej., "ABCD"

integer- utilizado para representar números enteros. p. ej. -1, -2, -3

float- utilizado para representar números reales. por ejemplo, 1,2, 42,42

boolean - utilizado para representar Verdadero o Falso.

complex- utilizado para representar números complejos. por ejemplo, 1.0 + 2.0j, 1.5 + 2.5j

Tipos de datos NumPy

NumPy tiene algunos tipos de datos adicionales y se refieren a tipos de datos con un carácter, como ienteros, uenteros sin signo, etc.

A continuación se muestra una lista de todos los tipos de datos en NumPy y los caracteres utilizados para representarlos.

i - entero

b - booleano

u - entero sin signo

f - flotar

c - flotador complejo

Tipos de datos NumPy

m - timedelta

M - datetime

O - objeto

S - cuerda

U - cadena unicode

V - fragmento de memoria fijo para otro tipo (vacío)

Tipos de datos NumPy

Comprobación del tipo de datos de una matriz

El objeto de matriz NumPy tiene una propiedad llamada dtype que devuelve el tipo de datos de la matriz:

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

Tipos de datos NumPy

Comprobación del tipo de datos de una matriz

El objeto de matriz NumPy tiene una propiedad llamada dtype que devuelve el tipo de datos de la matriz:

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

Modificación y Manipulación de estructura de una matriz

Forma de una matriz

La forma de una matriz es el número de elementos en cada dimensión.

Obtenga la forma de una matriz

Las matrices NumPy tienen un atributo llamado shape que devuelve una tupla con cada índice que tiene el número de elementos correspondientes.

```
import numpy as np
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
print(arr.shape)            #Resultado (2, 4)
```

Modificación y Manipulación de estructura de una matriz

Forma de una matriz

La forma de una matriz es el número de elementos en cada dimensión.

Obtenga la forma de una matriz

Las matrices NumPy tienen un atributo llamado shape que devuelve una tupla con cada índice que tiene el número de elementos correspondientes.

```
import numpy as np
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
print(arr.shape)            #Resultado (2, 4)
```

Modificación y Manipulación de estructura de una matriz

Remodelando matrices

Reformar significa cambiar la forma de una matriz.

La forma de una matriz es el número de elementos en cada dimensión.

Al remodelar, podemos agregar o eliminar dimensiones o cambiar el número de elementos en cada dimensión.

Modificación y Manipulación de estructura de una matriz

Remodelar de 1-D a 2-D

Convierta la siguiente matriz 1-D con 12 elementos en una matriz 2-D.

La dimensión más externa tendrá 4 matrices, cada una con 3 elementos:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(4, 3)
print(newarr)
```

Modificación y Manipulación de estructura de una matriz

Remodelar de 1-D a 2-D

Convierta la siguiente matriz 1-D con 12 elementos en una matriz 2-D.

La dimensión más externa tendrá 4 matrices, cada una con 3 elementos:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(4, 3)
print(newarr)
```

Modificación y Manipulación de estructura de una matriz

Remodelar de 1-D a 2-D

Convierta la siguiente matriz 1-D con 12 elementos en una matriz 2-D.

La dimensión más externa tendrá 4 matrices, cada una con 3 elementos:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(4, 3)
print(newarr)
```

Iteración de matriz NumPy

Iterando matrices

Iterar significa pasar por los elementos uno por uno.

Como tratamos con matrices multidimensionales en numpy, podemos hacer esto usando el forbucle básico de Python.

Si iteramos en una matriz 1-D, pasará por cada elemento uno por uno.

```
import numpy as np
arr = np.array([1, 2, 3])
for x in arr:
  print(x)
```

Iteración de matriz NumPy

Iterando matrices

Iterar significa pasar por los elementos uno por uno.

Como tratamos con matrices multidimensionales en numpy, podemos hacer esto usando el forbucle básico de Python.

Si iteramos en una matriz 1-D, pasará por cada elemento uno por uno.

```
import numpy as np
arr = np.array([1, 2, 3])
for x in arr:
  print(x)
```

Iteración de matriz NumPy

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
for x in arr:
  print(x)


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


import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
for x in arr:
  for y in x:
    print(y)
```

Iteración de matriz NumPy

Iterando matrices usando nditer ()

La función nditer() es una función de ayuda que se puede utilizar desde iteraciones muy básicas hasta muy avanzadas. Resuelve algunos problemas básicos que enfrentamos en iteración, veamos con ejemplos.

Iterando en cada elemento escalar

En los bucles for básicos , iterando a través de cada escalar de una matriz, necesitamos usar n bucles for que pueden ser difíciles de escribir para matrices con una dimensionalidad muy alta.

```
import numpy as np
arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
for x in np.nditer(arr):
  print(x)
```

Modificación y Manipulación de estructura de una matriz

Unirse a matrices NumPy

Unirse significa poner el contenido de dos o más arreglos en un solo arreglo.

En SQL unimos tablas en función de una clave, mientras que en NumPy unimos matrices por ejes.

Pasamos una secuencia de matrices que queremos unir a la función concatenate(), junto con el eje. Si el eje no se pasa explícitamente, se toma como 0.

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.concatenate((arr1, arr2))
print(arr)
```

Modificación y Manipulación de estructura de una matriz

División de matrices NumPy

La división es una operación inversa de la unión.

La unión fusiona varias matrices en una y la división divide una matriz en múltiples.

Usamos array_split()para dividir matrices, le pasamos la matriz que queremos dividir y el número de divisiones.

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

Modificación y Manipulación de estructura de una matriz

Buscando matrices

Puede buscar en una matriz un valor determinado y devolver los índices que coinciden.

Para buscar una matriz, use el método where().

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 4, 4])
x = np.where(arr == 4)
print(x)

#El ejemplo anterior devolverá una tupla: (array([3, 5, 6],)
#Lo que significa que el valor 4 está presente en los índices 3, 5 y 6.
```

Modificación y Manipulación de estructura de una matriz

Ordenación de matrices

Clasificar significa poner elementos en una secuencia ordenada .

Secuencia ordenada es cualquier secuencia que tiene un orden correspondiente a elementos, como numérico o alfabético, ascendente o descendente.

El objeto ndarray de NumPy tiene una función llamada sort(), que ordenará una matriz especificada.

```
import numpy as np
arr = np.array([3, 2, 0, 1])
print(np.sort(arr))
```

Filtros para una matriz

Arrays de filtrado

Obtener algunos elementos de una matriz existente y crear una nueva matriz a partir de ellos se llama filtrado .

En NumPy, filtra una matriz usando una lista de índice booleano .
Si el valor de un índice es True que el elemento está contenido en la matriz filtrada, si el valor de ese índice es, False ese elemento se excluye de la matriz filtrada.

```
import numpy as np
arr = np.array([41, 42, 43, 44])
x = arr[[True, False, True, False]]
print(x)
```

Random Numbers en NumPy

¿Qué es un número aleatorio?

Número aleatorio NO significa un número diferente cada vez. Aleatorio significa algo que no se puede predecir lógicamente.

Generar número aleatorio

NumPy ofrece el randommódulo para trabajar con números aleatorios.

```
from numpy import random
x = random.randint(100)
print(x)
```

Random Numbers en NumPy

Crear una matriz con datos aleatorios

```
from numpy import random
x=random.randint(100, size=(5))
print(x)
```

Random Numbers en NumPy

Distribución aleatoria de datos

¿Qué es la distribución de datos?

La distribución de datos es una lista de todos los valores posibles y la frecuencia con la que se produce cada valor.

Estas listas son importantes cuando se trabaja con estadísticas y ciencia de datos.

El módulo aleatorio ofrece métodos que devuelven distribuciones de datos generadas aleatoriamente.

Random Numbers en NumPy

Distribución aleatoria

Una distribución aleatoria es un conjunto de números aleatorios que siguen una determinada función de densidad de probabilidad

Tipos de distribuciones que pueden trabajarse con Numpy:

1. Distribución Normal
2. Distribución Binomial
3. Distribución Poisson
4. Distribución Uniforme
5. Distribución Multinomial
6. Distribución Exponencial
7. Distribución Pareto …. Etc.

Random Numbers en NumPy

Ejemplo de Distribución Normal y Poisson

Diferencia entre la distribución normal y de Poisson
La distribución normal es continua, mientras que Poisson es discreta.

Pero podemos ver que similar a binomial para una distribución de Poisson lo suficientemente grande, se volverá similar a la distribución normal con cierto desarrollo estándar y media.

Random Numbers en NumPy

Ejemplo de Distribución Normal y Poisson

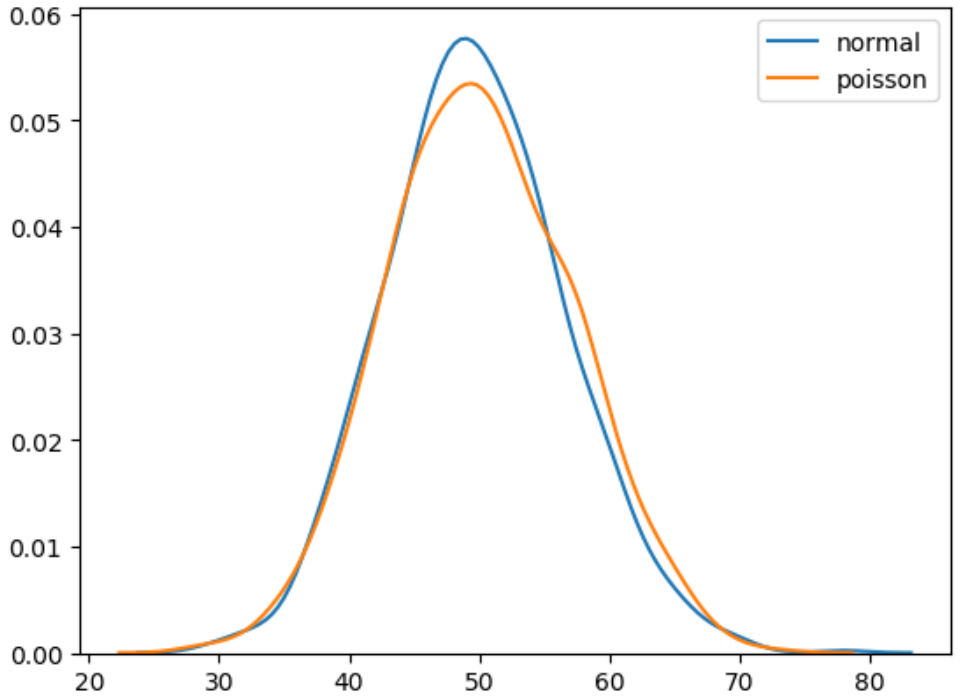
```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot(random.normal(loc=50, scale=7, size=1000), hist=False, label='normal')
sns.distplot(random.poisson(lam=50, size=1000), hist=False, label='poisson')

plt.show()
```

Random Numbers en NumPy

Ejemplo de Distribución Normal y Poisson



RESUMEN DE CLASE

GOBIERNO DE
EL SALVADOR

SECRETARÍA DE
INNOVACIÓN