GOBIERNO DE
EL SALVADOR

SECRETARÍA DE
INNOVACIÓN

CIENCIA DE DATOS

Agenda
Sesión 12/18

SciPy

Parte 1

- ¿Qué es **SciPy**?
- Constantes
- Optimizadores
- Datos escasos o despiersos
- Trabajo con Grafos

¿Qué es SciPy?

SciPy es una biblioteca de cálculo científico que usa NumPy debajo.

SciPy son las siglas de Scientific Python.

Proporciona más funciones de utilidad para la optimización, las estadísticas y el procesamiento de señales.

Al igual que NumPy, SciPy es de código abierto, por lo que podemos usarlo libremente.

SciPy fue creado por el creador de NumPy, Travis Olliphant.

¿Qué es SciPy?

¿Por qué utilizar SciPy?

Si SciPy usa NumPy debajo, ¿por qué no podemos simplemente usar NumPy?

SciPy ha optimizado y agregado funciones que se utilizan con frecuencia en NumPy y Data Science.

¿Qué es SciPy?

¿En qué idioma está escrito SciPy?

SciPy está escrito predominantemente en Python, pero algunos segmentos están escritos en C.

¿Dónde está el código base de SciPy?

El código fuente de SciPy se encuentra en este repositorio de github

https://github.com/scipy/scipy

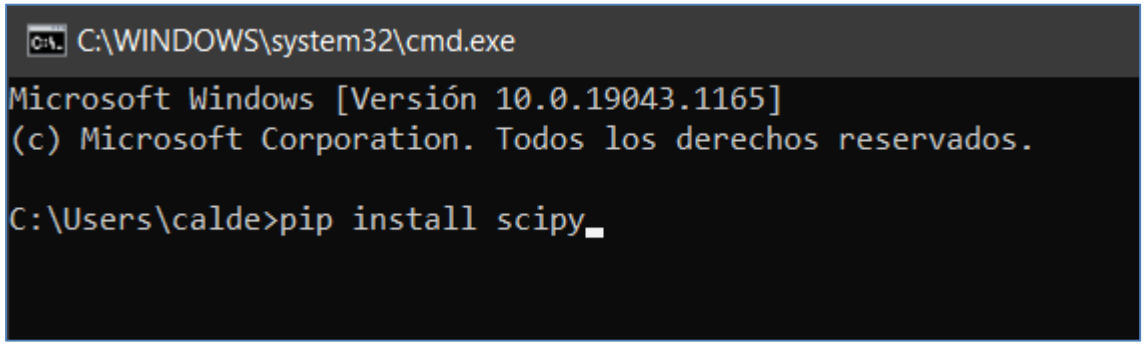
¿Qué es SciPy?

Instalación de SciPy

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

Instálelo usando este comando:

pip install scipy



¿Qué es SciPy?

Importar SciPy

Una vez que SciPy esté instalado, importe los módulos SciPy que desea utilizar en sus aplicaciones agregando la declaración:

from scipy import constants

Ahora hemos importado el módulo de constantes de SciPy y la aplicación está lista para usarlo:

¿Qué es SciPy?

Cuántos metros cúbicos hay en un litro:

```
from scipy import constants

print(constants.liter)
```

constantes

SciPy ofrece un conjunto de constantes matemáticas, una de ellas es la liter que devuelve 1 litro como metros cúbicos.

Constantes

Constantes en SciPy

Como SciPy se centra más en implementaciones científicas, proporciona muchas constantes científicas integradas.

Estas constantes pueden resultar útiles cuando se trabaja con Data Science.

PI es un ejemplo de constante científica.

```
from scipy import constants
print(constants.pi)
```

Constantes

Unidades constantes

Se puede ver una lista de todas las unidades bajo el módulo de constantes usando la función dir().

```
from scipy import constants
print(dir(constants))
```

Constantes

Categorías de unidad

Las unidades se colocan en estas categorías:

Métrico	Binario	Masa	Ángulo
Tiempo	Largo	Presión	Volumen
Velocidad	Temperatura	Energía	Poder
Fuerza			

Constantes

Prefijos métricos (SI)

Devuelve la unidad especificada en metros

(por ejemplo, centi devoluciones 0.01)

Prefijos binarios

Devuelve la unidad especificada en bytes

(por ejemplo, kibi devuelve 1024)

Masa

Devuelve la unidad especificada en kg

(por ejemplo, gram devoluciones 0.001)

Constantes

Ángulo

Devuelve la unidad especificada en radianes

(por ejemplo, degree devuelve 0.0174532925199433295)

Tiempo

Devuelve la unidad especificada en segundos

(por ejemplo, hour devuelve 3600.0)

Largo

Devuelve la unidad especificada en metros

(por ejemplo, nautical_mile devoluciones 1852.0)

Constantes

Presión

Devuelve la unidad especificada en pascales

(por ejemplo, psi devuelve 6894.757293168361)

Zona

Devuelve la unidad especificada en metros cuadrados

(por ejemplo, hectare devoluciones 10000.0)

Volumen

Devuelve la unidad especificada en metros cúbicos

(por ejemplo, liter devuelve 0.001)

Constantes

Velocidad:

Devuelve la unidad especificada en **metros por segundo**

(por ejemplo, speed_of_sound devoluciones 340.5)

Temperatura

Devuelve la unidad especificada en **Kelvin**

(por ejemplo, zero_Celsius devuelve 273.15)

Energía

Devuelve la unidad especificada en **Joules**

(por ejemplo, calorie devuelve 4.184)

Constantes

Poder

Devuelve la unidad especificada en vatios

(por ejemplo, horsepower devoluciones 745.6998715822701)

Fuerza

Devuelve la unidad especificada en newton (por ejemplo, kilogram_force devuelve 9.80665)

Optimizadores

Optimizadores en SciPy

Los optimizadores son un conjunto de procedimientos definidos en SciPy que encuentran el valor mínimo de una función o la raíz de una ecuación.

Optimización de funciones

Esencialmente, todos los algoritmos en Machine Learning no son más que una ecuación compleja que debe minimizarse con la ayuda de datos dados.

Optimizadores

Raíces de una ecuación

NumPy es capaz de encontrar raíces para polinomios y ecuaciones lineales, pero no puede encontrar raíces para ecuaciones no lineales, como esta:

x + cos(x)

Para eso puedes usar la función optimze.root de SciPy .

Optimizadores

Esta función toma dos argumentos obligatorios:

Una Función:

una función que representa una ecuación.

x:

una estimación inicial de la raíz.

La función devuelve un objeto con información sobre la solución.

La solución real se da en el atributo x del objeto devuelto:

Encuentra la raíz de la ecuación x + cos(x):

Optimizadores

```
from scipy.optimize import root
from math import cos

def eqn(x):
  return x + cos(x)

myroot = root(eqn, 0)

print(myroot.x)
```

Optimizadores

Minimizar una función

Una función, en este contexto, representa una curva, las curvas tienen puntos altos y puntos bajos .

Los puntos altos se llaman máximos .

Los puntos bajos se denominan mínimos .

El punto más alto de toda la curva se denomina máximos globales , mientras que el resto de ellos se denominan máximos locales .

El punto más bajo en toda la curva se denomina mínimos globales , mientras que el resto de ellos se denominan mínimos locales

Optimizadores

Encontrar mínimos

Podemos usar la función scipy.optimize.minimize() para minimizar la función.

La función minimize() toma los siguientes argumentos

fun: una función que representa una ecuación.

x0 : una estimación inicial de la raíz.

método : nombre del método a utilizar.

Valores legales: 'CG‘, 'BFGS‘, 'Newton-CG‘, 'L-BFGS-B‘, 'TNC‘, 'COBYLA‘, 'SLSQP‘

#Prácticas de Optimización o Analizadores Numéricos

Optimizadores

devolución de llamada :

función llamada después de cada iteración de optimización.

opciones :

un diccionario que define parámetros adicionales:

{

"disp": boolean - imprimir descripción detallada

"gtol": number - la tolerancia del error

}

Optimizadores

Minimice la función x^2 + x + 2 con BFGS:

```
from scipy.optimize import minimize

def eqn(x):
  return x**2 + x + 2

mymin = minimize(eqn, 0, method='BFGS')

print(mymin)
```

Datos escasos o dispersos

¿Qué son los datos dispersos?

Los datos dispersos son datos que tienen en su mayoría elementos no utilizados (elementos que no contienen información).

Puede ser una matriz como esta:

[1, 0, 2, 0, 0, 3, 0, 0, 0, 0, 0, 0]

Datos dispersos: es un conjunto de datos donde la mayoría de los valores de los elementos son cero.

Matriz densa: es lo opuesto a una matriz dispersa: la mayoría de los valores no son cero.

Datos escasos o dispersos

En computación científica, cuando se trata de derivadas parciales en álgebra lineal, nos encontramos con datos escasos.

Cómo trabajar con datos escasos

SciPy tiene un módulo scipy.sparse que proporciona funciones para tratar con datos escasos.

Existen principalmente dos tipos de matrices dispersas que usamos:

CSC (Columnas Dispersas Comprimidas) y

CSR (Filas dispersas comprimidas)

Datos escasos o dispersos

CSC : columna dispersa comprimida. Para aritmética eficiente, corte rápido de columnas.

CSR : fila dispersa comprimida. Para un corte rápido de filas, productos vectoriales de matriz más rápidos

Matriz de RSE

Podemos crear una matriz CSR pasando una matriz a la función scipy.sparse.csr_matrix().

Datos escasos o dispersos

Cree una matriz de CSR a partir de una matriz:

```
import numpy as np
from scipy.sparse import csr_matrix

arr = np.array([0, 0, 0, 0, 0, 1, 1, 0, 2])

print(csr_matrix(arr))
```

Datos escasos o dispersos

Cree una matriz de CSR a partir de una matriz:

```
import numpy as np
from scipy.sparse import csr_matrix

arr = np.array([0, 0, 0, 0, 0, 1, 1, 0, 2])

print(csr_matrix(arr))
```

Datos escasos o dispersos

Del resultado podemos ver que hay 3 elementos con valor.

El artículo 1.

está en la posición 0 de la fila 5 y tiene el valor 1.

El artículo 2.

está en la posición 0 de la fila 6 y tiene el valor 1.

El artículo 3.

está en la posición 0 de la fila 8 y tiene el valor 2.

Datos escasos o dispersos

Métodos de matriz dispersa

Ver datos almacenados (no los elementos cero) con la propiedad data:

```
import numpy as np
from scipy.sparse import csr_matrix

arr = np.array([[0, 0, 0], [0, 0, 1], [1, 0, 2]])

print(csr_matrix(arr).data)
```

Datos escasos o dispersos

Contando no ceros con el método count_nonzero() :

```
import numpy as np
from scipy.sparse import csr_matrix

arr = np.array([[0, 0, 0], [0, 0, 1], [1, 0, 2]])

print(csr_matrix(arr).count_nonzero())
```

Datos escasos o dispersos

Eliminando entradas cero de la matriz con el método eliminate_zeros() :

```
import numpy as np
from scipy.sparse import csr_matrix

arr = np.array([[0, 0, 0], [0, 0, 1], [1, 0, 2]])

mat = csr_matrix(arr)
mat.eliminate_zeros()

print(mat)
```

Datos escasos o dispersos

Conversión de csr a csc con el método tocsc() :

```
import numpy as np
from scipy.sparse import csr_matrix
arr = np.array([[0, 0, 0], [0, 0, 1], [1, 0, 2]])
newarr = csr_matrix(arr).tocsc()
print(newarr)
```

Nota: Aparte de las operaciones específicas dispersas mencionadas, las matrices dispersas admiten todas las operaciones que admiten las matrices normales, por ejemplo, remodelación, suma, aritmética, transmisión, etc.

Trabajo con Grafos

Los gráficos son una estructura de datos esencial.

SciPy nos proporciona el módulo scipy.sparse.csgraph para trabajar con dichas estructuras de datos.

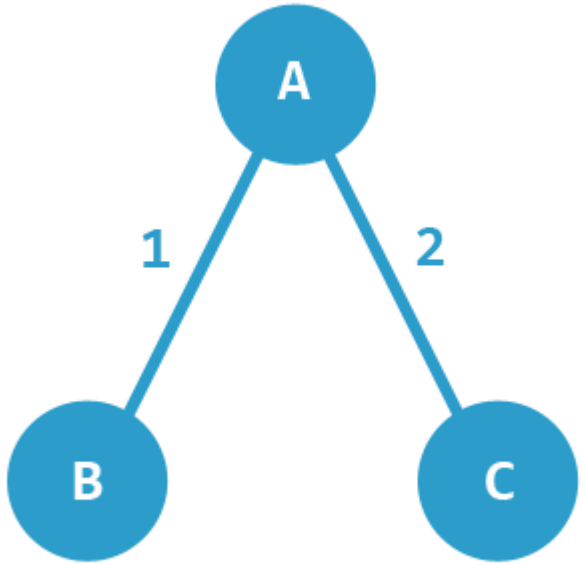
Matriz de adyacencia

La matriz de adyacencia es una matriz nxn donde n es el número de elementos en un gráfico.

Trabajo con Grafos

Y los valores representan la conexión entre los elementos.

Ejemplo:



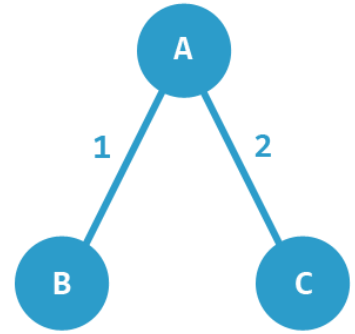
Trabajo con Grafos

Para un gráfico como este, con los elementos A, B y C, las conexiones son:

A y B están conectados con el peso 1.

A y C están conectados con el peso 2.

C & B no está conectado.



La Matriz de Adaptación se vería así:

```
   A B C
A:[0 1 2]
B:[1 0 0]
C:[2 0 0]
```

Trabajo con Grafos

Componentes conectados

Encuentre todos los componentes conectados con el método connected_components().

```
import numpy as np
from scipy.sparse.csgraph import connected_components
from scipy.sparse import csr_matrix
arr = np.array([ [0, 1, 2], [1, 0, 0], [2, 0, 0] ])
newarr = csr_matrix(arr)
print(connected_components(newarr))
```

Trabajo con Grafos

Dijkstra

Utilice el método dijkstra para encontrar la ruta más corta en un gráfico de un elemento a otro.

Toma los siguientes argumentos:

return_predecessors: boolean (Verdadero para devolver la ruta completa de recorrido, de lo contrario Falso).

índices: índice del elemento para devolver todas las rutas de ese elemento únicamente.

límite: peso máximo del camino.

Trabajo con Grafos

```
import numpy as np
from scipy.sparse.csgraph import dijkstra
from scipy.sparse import csr_matrix
arr = np.array([
  [0, 1, 2],
  [1, 0, 0],
  [2, 0, 0]
])
newarr = csr_matrix(arr)
print(dijkstra(newarr, return_predecessors=True, indices=0))
```

Trabajo con Grafos

Floyd Warshall

Utilice el método floyd_warshall() para encontrar la ruta más corta entre todos los pares de elementos.

Encuentre el camino más corto entre todos los pares de elementos:

```
import numpy as np
from scipy.sparse.csgraph import floyd_warshall
from scipy.sparse import csr_matrix

arr = np.array([  [0, 1, 2],  [1, 0, 0],  [2, 0, 0]])

newarr = csr_matrix(arr)
print(floyd_warshall(newarr, return_predecessors=True))
```

Trabajo con Grafos

Bellman Ford

El método bellman_ford() también puede encontrar la ruta más corta entre todos los pares de elementos, pero este método también puede manejar pesos negativos.

```
import numpy as np
from scipy.sparse.csgraph import bellman_ford
from scipy.sparse import csr_matrix

arr = np.array([ [0, -1, 2],  [1, 0, 0],  [2, 0, 0] ])

newarr = csr_matrix(arr)
print(bellman_ford(newarr, return_predecessors=True, indices=0))
```

Trabajo con Grafos

Profundidad de primer orden

El método depth_first_order() devuelve un primer recorrido en profundidad desde un nodo.

Esta función toma los siguientes argumentos: la grafo y el elemento inicial desde el que se recorre el grafo.

```
import numpy as np
from scipy.sparse.csgraph import depth_first_order
from scipy.sparse import csr_matrix

arr = np.array([ [0, 1, 0, 1], [1, 1, 1, 1], [2, 1, 1, 0], [0, 1, 0, 1] ])

newarr = csr_matrix(arr)
print(depth_first_order(newarr, 1))
#Recorra primero la profundidad del gráfico para la matriz de adyacencia dada
```

Trabajo con Grafos

Amplitud de primer orden

El método breadth_first_order() devuelve un primer recorrido en amplitud desde un nodo.

Esta función toma los siguientes argumentos: la grafo y el elemento inicial desde el que se recorre el grafo.

```
import numpy as np
from scipy.sparse.csgraph import breadth_first_order
from scipy.sparse import csr_matrix

arr = np.array([ [0, 1, 0, 1], [1, 1, 1, 1], [2, 1, 1, 0], [0, 1, 0, 1] ])

newarr = csr_matrix(arr)
print(breadth_first_order(newarr, 1))
#Recorra primero la profundidad del gráfico para la matriz de adyacencia dada
```

RESUMEN DE CLASE

GOBIERNO DE
EL SALVADOR

SECRETARÍA DE
INNOVACIÓN