Universidad de Córdoba

Escuela Politécnica Superior de Córdoba

Grado de Ingeniería Informática - Mención en computación

Tercer curso - Segundo cuatrimestre - 2020/2021

Introducción al Aprendizaje Automático

Práctica 1: Ejercicios de introducción a Numpy

Profesor: *Nicolás Emilio García Pedrajas*

Autor: *Ventura Lucena Martínez*

Córdoba, 29 de junio de 2021

Índice

Listings

1 Ejercicio 1

Implemente mediante un programa Python la asignación del reparto de escaños de una circunscripción electoral usando la Ley D'Hondt. Los datos se pueden introducir por teclado o leerse desde un fichero.

Los parámetros a tener en cuenta en dicha asignación del reparto de escaños son los siguientes:

- Número de escaños.
- Número de partidos.
- Votos por partido.

$$votosEscaño = votosTotales/(escañosConseguidos + 1) \tag{1}$$

Listing 1: **Resultados - Ejercicio 1**

```
Select n seats: 350

Data browser
1. File.
2. Terminal.
> 2

Select n parties: 8
Party votes 1: 6792199
Party votes 2: 5047040
Party votes 3: 3119364
Party votes 4: 3656979
Party votes 5: 874859
Party votes 6: 1650318
Party votes 7: 530225
Party votes 8: 379002


Party 1 has obtained 108 seats.
Party 2 has obtained 80 seats.
Party 3 has obtained 50 seats.
Party 4 has obtained 58 seats.
Party 5 has obtained 14 seats.
Party 6 has obtained 26 seats.
Party 7 has obtained 8 seats.
Party 8 has obtained 6 seats.
```

Dichos resultados se han obtenido tras la aplicación del sistema de aplicación de la Ley D'Hont.

2 Ejercicio 2

Implemente un programa Python que genere aleatoriamente una matriz de valores reales de un tamaño indicado por teclado. Sobre la matriz generada realice las siguientes operaciones:

1. **Obtenga los valores máximos y mínimos de la matriz.**
2. **Use el producto escalar para obtener el ángulo formado por dos vectores fila o columna solicitados por teclado.**

Para la realización del ejercicio se han utilizado las siguientes funciones de la librería Numpy de Python:

- **np.random.randint:** para la generación de números enteros aleatorios.
- **np.amax:** para obtener el valor máximo de un array de Numpy.
- **np.amin:** para obtener el valor mínimo de un array de Numpy.
- **np.dot:** para el cálculo del producto escalar de dos arrays de Numpy.

Listing 2: **Resultados - Ejercicio 2**

```
Select matrix rows:
>5

Select matrix cols:
>5

 [[0.2066573  0.98490714 0.80768684 0.28293064 0.95940588]
 [0.02712792 0.87962282 0.38582655 0.23447026 0.50651456]
 [0.54996967 0.73726578 0.51104316 0.22284368 0.86544929]
 [0.61591319 0.92474934 0.94204998 0.62997418 0.83344764]
 [0.86651488 0.08512814 0.77738924 0.48932174 0.97947827]]

Maximum: 0.9849071394728159
Minimum: 0.02712791688954541

Dot product - rows 0 and 1: 1.7358718664878126
Dot product - cols 3 and 4: 2.4693635581088884
```

3 Ejercicio 3

Implemente un programa Python que lea una matriz de número reales desde teclado de una dimensión dada. A partir de la matriz leída debe calcular la siguiente información:

1. **Máximo por filas y por columnas.**
2. **Determinante de la matriz.**
3. **Rango de la matriz.**

Para la realización del ejercicio se han utilizado las siguientes funciones de la librería Numpy de Python:

- **np.empty:** para la generación de un array vacío de **n** dimensiones.
- **np.nditer:** función que permite iterar de manera eficiente sobre un array.

Listing 3: **Resultados - Ejercicio 3**

```
Select matrix size: 3
> 7
> 5
> 3
> 4
> 8
> 9
> 6
> 2
> 1

[[7. 5. 3.]
 [4. 8. 9.]
 [6. 2. 1.]]

[7. 8. 9.]
[7. 9. 6.]

Determinant: 59.999999999999986
Range:  3
```

4 Ejercicio 4

Implemente un programa Python que lea una matriz de número enteros desde teclado de una dimensión dada. A partir de la matriz leída debe calcular la siguiente información:

1. **Moda de la matriz.**
2. **Media de todos los elementos de la matriz.**

Para la realización del ejercicio se han utilizado las siguientes funciones de la librería Numpy de Python:

- **st.mode:** función de la librería Scipy que permite obtener la moda de una matrix de la librería de Numpy.
- **np.mean:** función que permite obtener la media de un array de la librería de Numpy.

Listing 4: **Resultados - Ejercicio 4**

```
Select matrix rows:
>3
Select matrix cols:
>3

> 7
> 5
> 8
> 4
> 2
> 3
> 0
> 8
> 6

[[7 5 8]
 [4 2 3]
 [0 8 6]]

Mode:  [[0 2 3]]
Mean:  4.777777777777778
```

5 Ejercicio 5

Implemente un programa Python que lea una matriz de número reales desde un fichero texto con formato libre. Una vez leído el programa debe obtener la inversa de la matriz y realizar un producto matricial para comprobar que el cálculo de la inversa es correcto.

Para la realización del ejercicio se han utilizado las siguientes funciones de la librería Numpy de Python:

- **np.dot:** para el cálculo del producto escalar de dos arrays de Numpy.
- **np.linalg:** función de Numpy que permite calcular la inversa de un array.

Listing 5: **Resultados - Ejercicio 5**

```
Inverse:
[[-0.842  0.178  0.043  0.258]
 [-8.56   1.511 -0.736  3.358]
```

```
 [10.938 -1.111  0.735 -5.222]
 [-2.701 -0.165 -0.124  1.951]]


Product:
[[ 1.  0.  0.  0.]
 [-0.  1. -0. -0.]
 [ 0.  0.  1.  0.]
 [ 0.  0.  0.  1.]]
```

6 Ejercicio 6

Implemente un programa en Python que lea los coeficientes de un sistema de ecuaciones lineales compatible y determinado y aplique la regla de Cramer para obtener su solución (Regla de Cramer).

En este ejercicio se ha aplicado la regla de Cramer para la resolución de sistemas compatibles determinados:

Listing 6: **Código fuente - Ejercicio 6**

```
"""
References:
        - Cramer's Rule:        https://rosettacode.org/wiki/Cramer%27s_rule#
            Python
"""

import numpy as np


def computeCramerRule(A, B, C):
    #   Result:
    X = []

    for i in range(0, len(B)):
        for j in range(0, len(B)):
            C[j][i] = B[j]
            if i > 0:
                C[j][i - 1] = A[j][i - 1]
        X.append(round(np.linalg.det(C) / np.linalg.det(A), 1))

    return X


def main():
    example = input(
        "\nMenu:\n1. 2x2 System equations.\n2. 3x3 System equations.\n3. 4x4
            System equations.\n> ")
```

```
    if example == "1":
        #       Coefficients matrix:
        A = np.array([[4, -1], [3, 5]])
        #       Independent components:
        B = np.array([-9, -1])

    if example == "2":
        A = np.array([[3, 2, -1], [1, -1, 4], [5, -3, 1]])
        B = np.array([12, 19, 8])

    if example == "3":
        A = np.array([[2, -1, 5, 1], [3, 2, 2, -6],
                      [1, 3, 3, -1], [5, -2, -3, 3]])
        B = np.array([-3, -32, -47, 49])

    C = np.copy(A)

    X = computeCramerRule(A, B, C)
    print("\nSolution:", X)


if __name__ == "__main__":
    main()
```

Listing 7: **Resultados - Ejercicio 6**

```
Menu:
1. 2x2 System equations.
2. 3x3 System equations.
3. 4x4 System equations.
> 3

Solution: [2.0, -12.0, -4.0, 1.0]
```

Referencias

[1] Moodle Universidad de Córdoba - Enunciado práctica 1.

[2] Moodle Universidad de Córdoba - Introducción a Numpy.