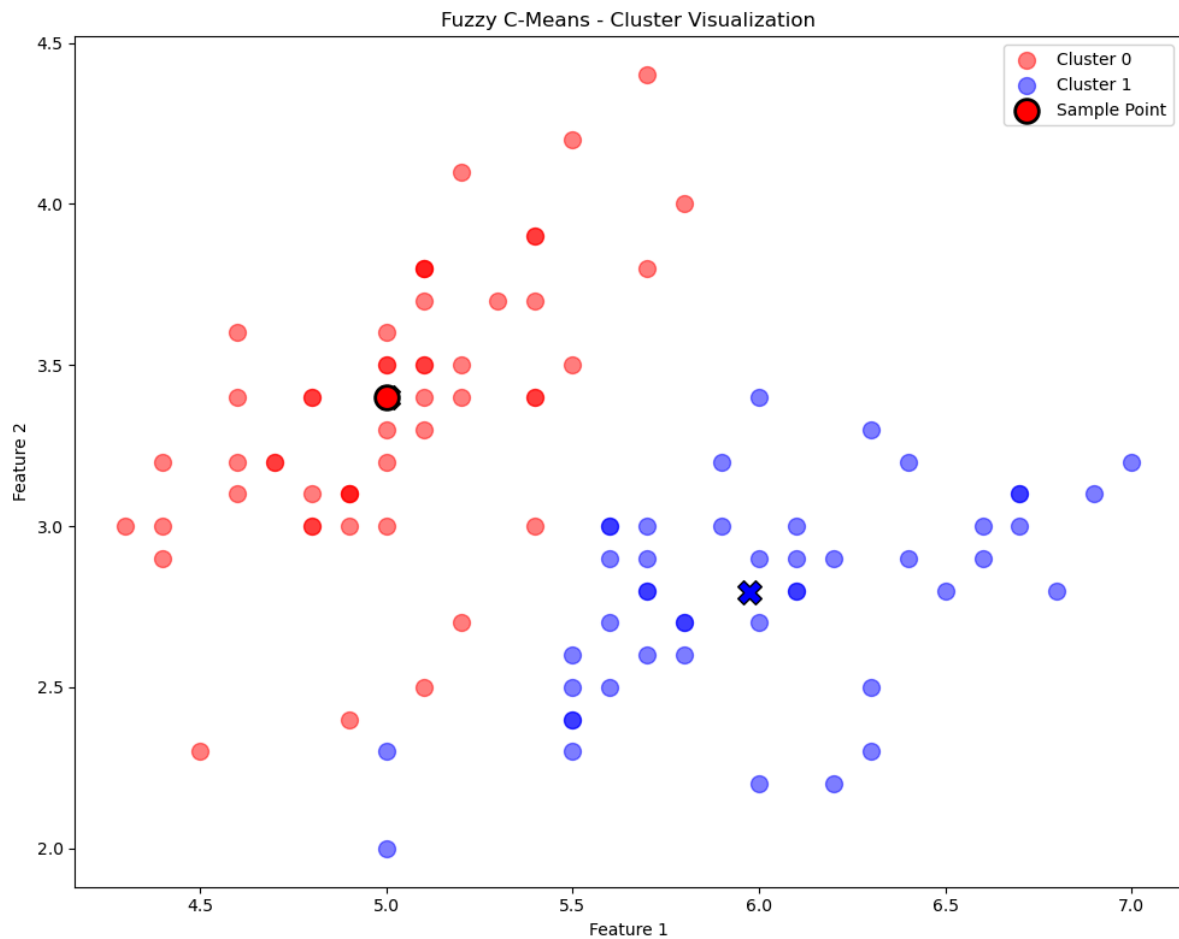


Algoritmos de Clasificación en Python



Jaime Alonso Fernández 2024/2025

Ingeniería del Conocimiento: Grado en Ingeniería Informática
Universidad Complutense de Madrid. Plan académico 2019

1. Descripción del Proyecto:	1
1.1. Implementación del algoritmo base: Medias Borrosas	1
1. Matriz de pertenencia difusa (U)	1
2. Distancia Euclídea	2
3. Actualización de centros (centroides)	2
4. Criterio de convergencia	2
1.2. Expansión del proyecto: Algoritmo de Bayes	3
1. Media de cada clase	3
2. Matriz de Covarianza de cada clase	3
3. Probabilidad de cada clase	3
4. Función Gaussiana	4
1.3. Expansión del proyecto: Algoritmo Lloyd	4
1. Asignación de clases	5
2. Actualización de centros (centroides)	5
3. Criterio de convergencia	5
1.4. Expansión del proyecto: Gráficas	6
2. Manual de uso:	6
Requisitos:	6
Ejecución:	7
3. Ejemplos de uso	7
3.1. Ejemplo con IDE	7
3.2. Ejemplo con cmd	10

1. Descripción del Proyecto:

Este proyecto recoge la metodología empleada para el desarrollo de los algoritmos de Medias Borrosas, Bayes y Lloyd para la categorización de flores de la familia Iris. Cada uno de los algoritmos se ha desarrollado en base a lo visto en las clases de teoría adaptados a las facilidades que ofrece el lenguaje Python para los cálculos numéricos.

Pasemos ahora a analizar las diferentes implementaciones de los algoritmos, en orden de implementación.

1.1. Implementación del algoritmo base: Medias Borrosas

El algoritmo de medias borrosas, se ha implementado según se ha implementado en las clases teóricas. Para ello necesitaremos usar variables que recojan, el número de clústeres (nombre C), el número de muestras de entreno (nombre N), una variable para los centros de los clústeres (nombre V) y finalmente una variable para la matriz de pertenencia (nombre U). Acompañando a estas variables, también tenemos otras que serán estáticas y definirán la tolerancia, ponderación y un límite en cuanto a las iteraciones.

Tras esto indicaremos la variable V a la matriz sugerida en los apuntes (véase en la imagen 1.1.1) e iremos calculando de forma iterativa la pertenencia difusa (método `_calculate_membership`), así como los nuevos centros de los clústeres hasta que o bien se alcance el límite de iteraciones o se alcance el límite de tolerancia.

```
self.V = np.array([[4.6, 3.0, 4.0, 0.0],  
                  [6.8, 3.4, 4.6, 0.7]])
```

Imagen 1.1.1. Valores iniciales para la matriz V

Las fórmulas empleadas para realizar los cálculos son las siguientes:

1. Matriz de pertenencia difusa (U)

Fórmula:

$$u_{ik} = \left(\sum_{j=1}^C \left(\frac{d_{ik}}{d_{ij}} \right)^{\frac{2}{b-1}} \right)^{-1}$$

Significado:

- u_{ik} : grado de pertenencia del dato i al cluster k
- d_{ik} : distancia entre el dato i y el centro del cluster k
- b : parámetro de ponderación
- C : número de clusters

Este valor está **entre 0 y 1** para cada dato-cluster, y la suma de pertenencias de un punto a todos los clusters es 1.

2. Distancia Euclídea

Fórmula:

$$d_{ik} = \sqrt{\sum_{l=1}^p (x_{il} - v_{kl})^2}$$

Significado:

- x_{il} : punto de datos i
- v_{kl} : centro del cluster k
- p : número de características

Es la distancia habitual en espacios vectoriales.

3. Actualización de centros (centroides)

Fórmula:

$$v_k = \frac{\sum_{i=1}^N u_{ik}^b \cdot x_i}{\sum_{i=1}^N u_{ik}^b}$$

Significado:

- v_k : nuevo centro del cluster k
- u_{ik} : grado de pertenencia del dato i al cluster k
- b : parámetro de ponderación (exponente que suaviza las pertenencias)

Este es un promedio ponderado de los puntos, ponderado por la pertenencia elevada al exponente b .

4. Criterio de convergencia

Fórmula:

$$\|V^{\text{nuevo}} - V^{\text{viejo}}\| < \varepsilon$$

- Se compara la norma (distancia euclídea) entre los centros nuevos y los anteriores.
- Si la diferencia es menor que la tolerancia (ε), se detiene el algoritmo.

1.2. Expansión del proyecto: Algoritmo de Bayes

El algoritmo de clasificación de Bayes, se ha implementado según se ha implementado en las clases teóricas. Para ello necesitaremos usar variables que recojan, las clases (nombre *classes*) , las clases anteriores (nombre *classes_prior*), una variable para las medias (nombre *means*) y finalmente una variable las matrices de covarianza (nombre *covariances*).

Tras esto pasaremos a calcular las medias por cada una de las clases, así como sus matrices de covarianza y las probabilidades de cada clase.

Las fórmulas empleadas para realizar los cálculos son las siguientes:

1. Media de cada clase

Fórmula:

$$\mu_c = \frac{1}{N_c} \sum_{i=1}^{N_c} x_i$$

Significado:

- x_i : muestra i de la clase c
- N_c : número de muestras de la clase c

2. Matriz de Covarianza de cada clase

Fórmula:

$$\Sigma_c = \frac{1}{N_c - 1} \sum_{i=1}^{N_c} (x_i - \mu_c)(x_i - \mu_c)^T$$

Significado:

- x_i : muestra i de la clase c
- N_c : número de muestras de la clase c
- μ_c : media de la clase c

3. Probabilidad de cada clase

Fórmula:

$$P(c) = \frac{N_c}{N}$$

Significado:

- N_c : número de muestras de la clase c N_c : número de muestras de la clase c
- N : número de muestras totales N : número de muestras totales

4. Función Gaussiana

Fórmula:

$$p(x | c) = \frac{1}{\sqrt{(2\pi)^n \cdot \det(\Sigma_c)}} \cdot \exp \left(-\frac{1}{2} (x - \mu_c)^T \Sigma_c^{-1} (x - \mu_c) \right)$$

Significado:

- x : Vector de características de la muestra que se quiere clasificar
- μ : Vector de medias (promedios) de las características para una clase dada.
- Σ : Matriz de covarianza de la clase (representa la dispersión y correlación entre características).
- $(x - \mu)^T \Sigma^{-1}$: Distancia de Mahalanobis entre la muestra x y la media, ajustada por la forma de la distribución.
- d : Número de dimensiones o características (longitud del vector x).
- $(2\pi)^d$: Parte del factor de normalización para asegurar que la integral total de la distribución sea 1.
- $\exp \left(-\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right)$: Núcleo exponencial que da la forma de campana a la distribución.

1.3. Expansión del proyecto: Algoritmo Lloyd

El algoritmo de Lloyd, se ha implementado según se ha implementado en las clases teóricas. Para ello necesitaremos usar variables que recojan valores similares a los vistos en la implementación de Medias Borrosas, siendo estas el número de clusters (nombre `n_clusters`), los centros (nombre `centers`) y las clases (nombre `classes`). Acompañando a estas variables, existen unas variables prefijadas, que son la tasa de aprendizaje (nombre `learning_rate`), el máximo de iteraciones (nombre `max_iter`) y la tolerancia (nombre `tolerance`).

Tras esto indicaremos la variable de centros a la matriz sugerida en los apuntes (véase en la imagen 1.3.1). Tras esto asignaremos los datos proporcionados (de entreno) a uno de los clústeres o centros, e iremos actualizando los centros en base a medias y a la tasa de aprendizaje hasta que se alcance el límite de iteraciones o de tolerancia.

```
self.centers = np.array([[4.6, 3.0, 4.0, 0.0],
                        [6.8, 3.4, 4.6, 0.7]])
```

Imagen 1.3.1. Valor inicial para la variable centers

Las fórmulas empleadas para realizar los cálculos son las siguientes:

1. Asignación de clases

Fórmula:

$$d(\mathbf{x}_j, \mathbf{C}_i) = \|\mathbf{x}_j - \mathbf{C}_i\|_2 = \sqrt{\sum_{m=1}^d (x_{j,m} - C_{i,m})^2}$$

Significado:

- x_j : es el vector de características de la muestra j .
- C_i : es el centro del clúster i .
- d : es la cantidad de características (dimensiones).

2. Actualización de centros (centroides)

Fórmula:

$$\mathbf{C}_i^{\text{new}} = \mathbf{C}_i^{\text{old}} + \eta \cdot (\text{mean}(\mathbf{x}_j \mid \text{label}(\mathbf{x}_j) = i) - \mathbf{C}_i^{\text{old}})$$

Significado:

- $\text{mean}(x_j \mid \text{label}(x_j) = i)$: es la media de las muestras asignadas al clúster i .
- η : es la tasa de aprendizaje.

Este es un promedio ponderado de los puntos, ponderado por la pertenencia elevada al exponente b .

3. Criterio de convergencia

Fórmula:

$$\|\mathbf{C}_i^{\text{new}} - \mathbf{C}_i^{\text{old}}\|_2 < \epsilon \quad \forall i$$

- Se compara la norma (distancia euclídea) entre los centros nuevos y los anteriores.
- Si la diferencia es menor que la tolerancia (ϵ), se detiene el algoritmo.

1.4. Expansión del proyecto: Gráficas

Finalmente, para visualizar las ejecuciones del proyecto, he decidido hacer uso de la librería `matplotlib` la cual me permite realizar gráficos.

Con ella, he realizado una muestra de los centros de cada clúster (marcados con una X) y he diferenciado cada cluster y sus miembros con un color diferente (rojo para el clúster 0 y azul para el 1). A partir de ahí paso a realizar un *scatterplot* del resto de puntos de manera

que se coloreen en base al color del clúster al que pertenecen, para acabar mostrando en una variante más intensa del color, el resultado predicho por el modelo y su situación en el plano.

Con esto obtenemos resultados como los vistos en la imagen 1.4.1.

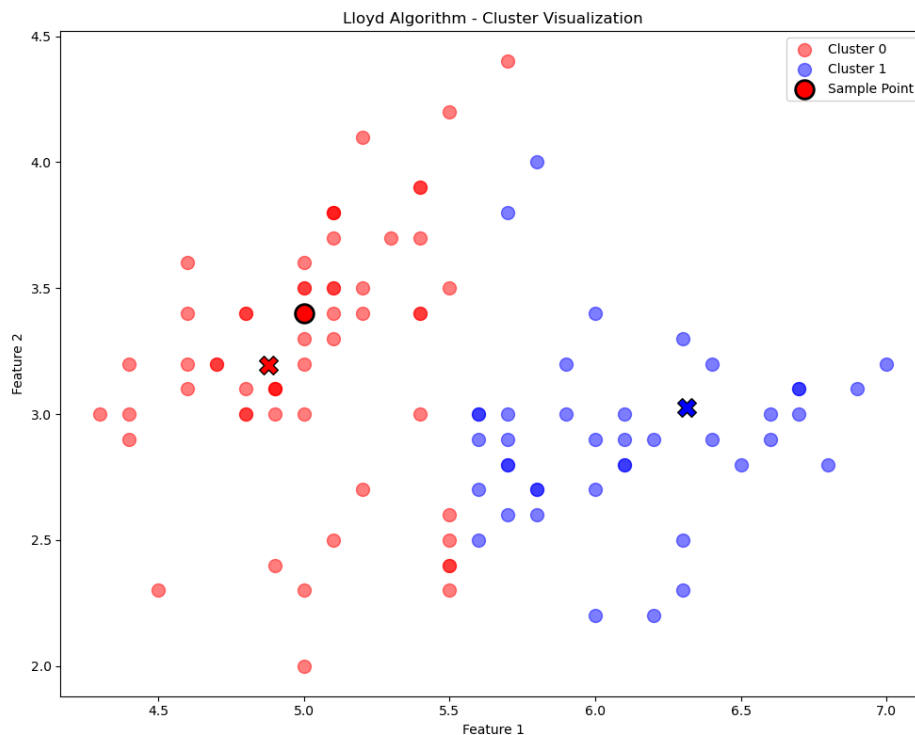


Imagen 1.4.1

2. Manual de uso:

Requisitos:

- Esta entrega no cuenta con un archivo ejecutable por lo que es imprescindible python 3.11 en adelante instalado a través de una fuente oficial (junto a pip) o desde conda.
- Las librerías externas empleadas son las siguientes:

Nombre Librería	Instalación mediante pip	Instalación mediante conda
matplotlib	pip install matplotlib	conda install matplotlib
numpy	pip install numpy	conda install numpy

Ejecución:

La ejecución del proyecto puede llevarse a cabo por diferentes medios, desde IDE 's como [Pycharm](#) o [Visual Studio Code](#), o a través de una terminal.

Para ejecutar la versión de terminal tendremos que ejecutar el archivo **solution.py**, este ejecutará por defecto la lógica de entreno y predicción para todos los casos de test proporcionados, incluyendo la lógica que permite la visualización gráfica de los resultados.

La ejecución sin IDE se realizará abriendo una terminal en la carpeta del proyecto y ejecutando las siguientes líneas de comando:

Versión	Comando para python normal	Comando para python3
No GUI	python solution.py	python3 solution.py

3. Ejemplos de uso

Durante este apartado voy a mostrar la ejecución a través de los dos medios (IDE y cmd) y cómo se visualizará la información en cada una.

3.1. Ejemplo con IDE

Para esta demostración haremos uso del IDE Pycharm, de JetBrains. Para ello comenzaremos abriendo el proyecto a través de la carpeta adjunta a la entrega, de manera que tengamos una vista similar a la de la imagen 3.1.1.

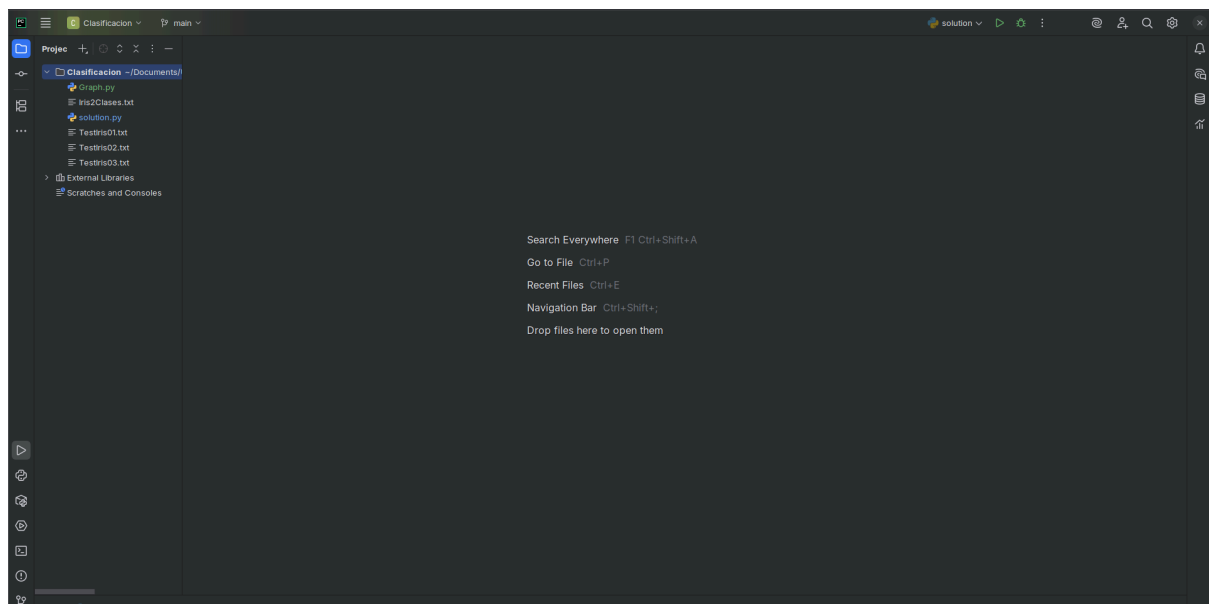
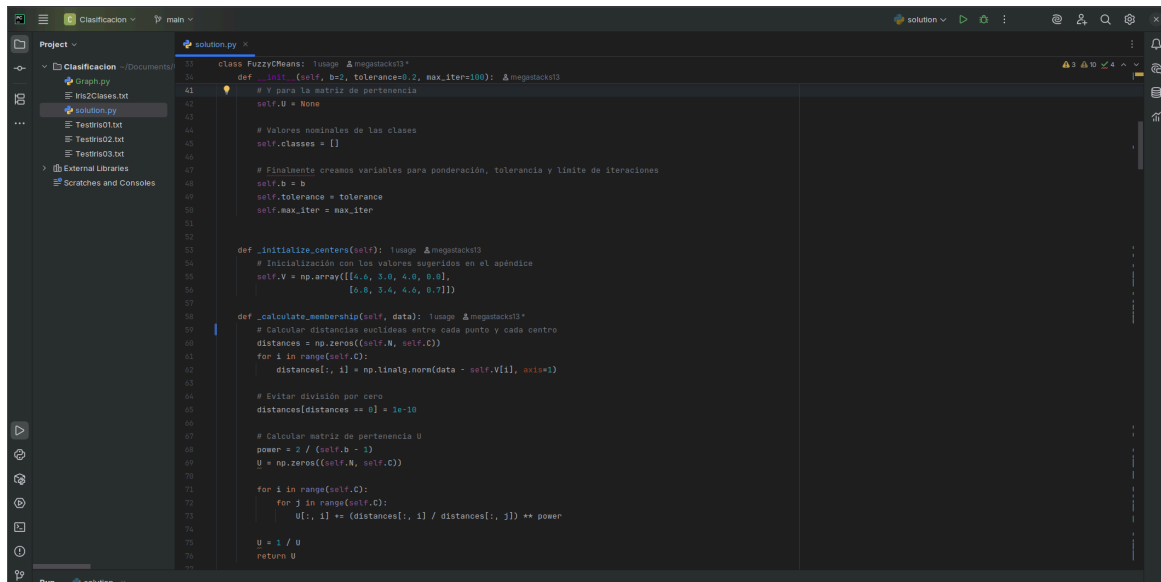


Imagen 3.1.1. Vista del IDE JetBrains nada más abrir el proyecto

Una vez en esta ventana haremos doble click sobre el archivo **solution.py** y este se abrirá (véase la imagen 3.1.2). Sobre esta ventana, haremos click en el símbolo de triángulo verde que se encuentra en la parte superior y pronto veremos como se abre un diálogo a la

derecha mostrando los resultados gráficos (imagen 3.1.3), así como una ventana con una terminal y los resultados detallados en la misma (imagen 3.1.4).

Para ir visualizando los gráficos, tendremos que ir seleccionandolos en el panel de la derecha del todo, donde aparecen numerados.



```
18 class FuzzyCMeans:
19     def __init__(self, b=2, tolerance=0.2, max_iter=100):
20         # Y para la matriz de pertenencia
21         self.U = None
22         # Valores nominales de las clases
23         self.classes = []
24         # Finalmente creamos variables para ponderación, tolerancia y límite de iteraciones
25         self.b = b
26         self.tolerance = tolerance
27         self.max_iter = max_iter
28
29     def _initialize_centers(self):
30         # Inicialización con los valores superiores en el apéndice
31         self.V = np.array([[4.4, 3.0, 4.0, 4.0],
32                           [0.8, 3.4, 4.0, 0.7]])
33
34     def _calculate_membership(self, data):
35         # Calcular distancias euclídeas entre cada punto y cada centro
36         distances = np.zeros((self.N, self.C))
37         for i in range(self.C):
38             distances[:, i] = np.linalg.norm(data - self.V[i], axis=1)
39
40         # Evitar división por cero
41         distances[distances == 0] = 1e-10
42
43         # Calcular matriz de pertenencia U
44         power = 2 / (self.b - 1)
45         U = np.zeros((self.N, self.C))
46
47         for i in range(self.C):
48             for j in range(self.N):
49                 U[j, i] += (distances[j, i] / distances[j, i]) ** power
50
51         U = 1 / U
52         return U
```

Imagen 3.1.2. Vista del IDE con **solutions.py** abierto

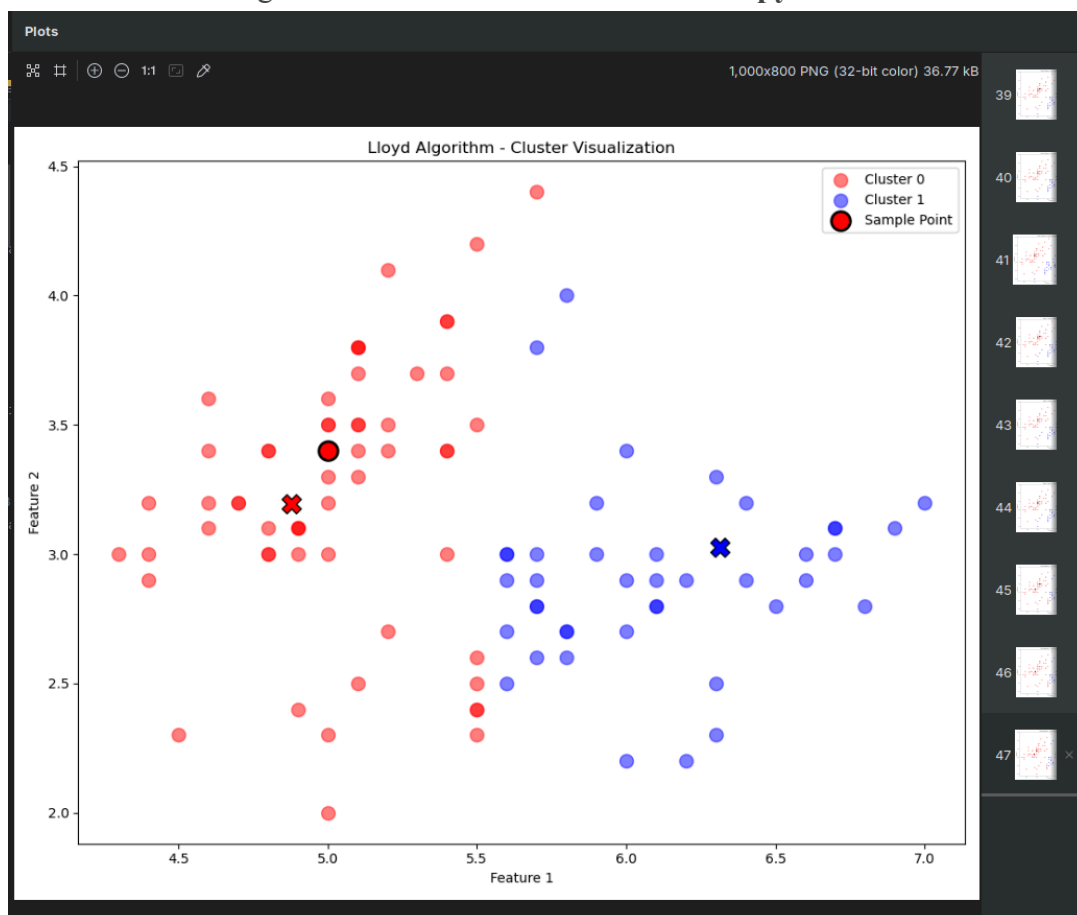
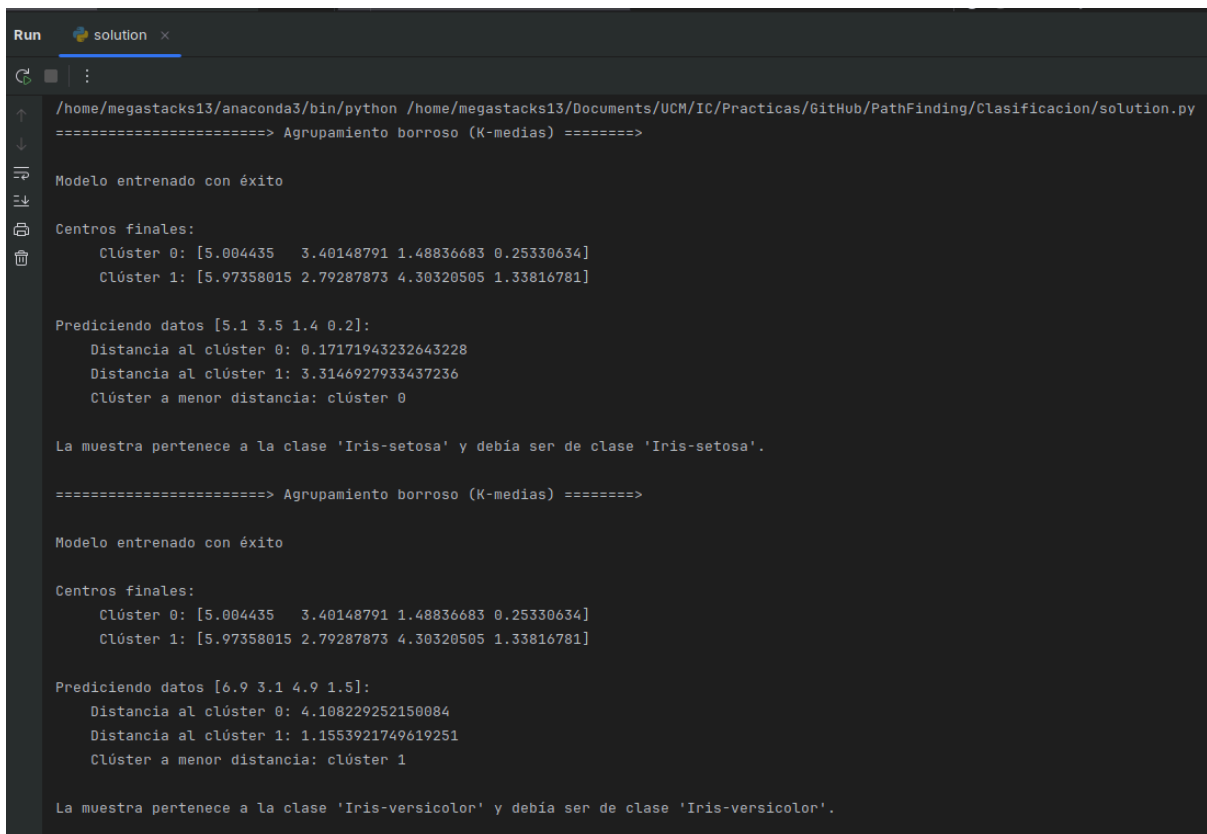


Imagen 3.1.3. Vista de las gráficas



```
Run solution x
/home/megastacks13/anaconda3/bin/python /home/megastacks13/Documents/UCM/IC/Practicas/GitHub/PathFinding/Clasificacion/solution.py
=====> Agrupamiento borroso (K-medias) =====>

Modelo entrenado con éxito

Centros finales:
Clúster 0: [5.004435 3.40148791 1.48836683 0.25330634]
Clúster 1: [5.97358015 2.79287873 4.30320505 1.33816781]

Prediciendo datos [5.1 3.5 1.4 0.2]:
Distancia al clúster 0: 0.17171943232643228
Distancia al clúster 1: 3.3146927933437236
Clúster a menor distancia: clúster 0

La muestra pertenece a la clase 'Iris-setosa' y debía ser de clase 'Iris-setosa'.

=====> Agrupamiento borroso (K-medias) =====>

Modelo entrenado con éxito

Centros finales:
Clúster 0: [5.004435 3.40148791 1.48836683 0.25330634]
Clúster 1: [5.97358015 2.79287873 4.30320505 1.33816781]

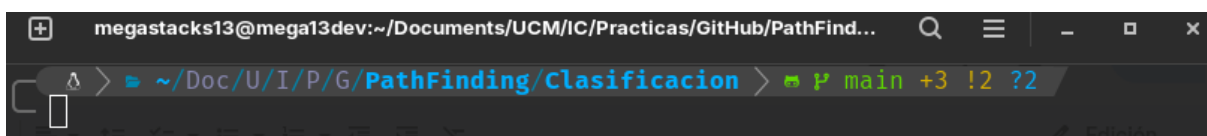
Prediciendo datos [6.9 3.1 4.9 1.5]:
Distancia al clúster 0: 4.108229252150084
Distancia al clúster 1: 1.1553921749619251
Clúster a menor distancia: clúster 1

La muestra pertenece a la clase 'Iris-versicolor' y debía ser de clase 'Iris-versicolor'.
```

Imagen 3.1.4. Vista de fragmento de salida en la cmd

3.2. Ejemplo con cmd

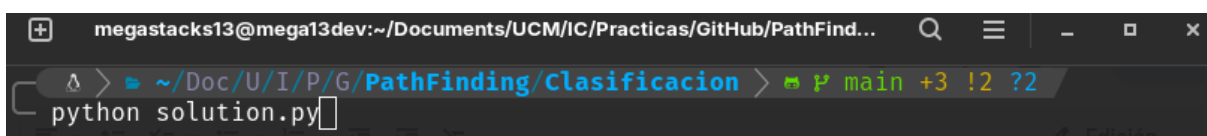
Para esta ejecución usaremos la terminal por defecto de GNOME en *Linux*, de todas formas el resultados sería similar en cualquier terminal de *Windows*. Para ello primero viajamos a la carpeta donde se encuentran el proyecto en la misma (Imagen 3.2.1).



```
megastacks13@mega13dev:~/Documents/UCM/IC/Practicas/GitHub/PathFind...
~/Doc/U/I/P/G/PathFinding/Clasificacion
```

Imagen 3.2.1. CMD en la carpeta del proyecto

Una vez en la ubicación usaremos uno de los comandos proporcionados anteriormente, en este caso usaré el correspondiente a *python*. (Imagen 3.2.2).



```
megastacks13@mega13dev:~/Documents/UCM/IC/Practicas/GitHub/PathFind...
~/Doc/U/I/P/G/PathFinding/Clasificacion
python solution.py
```

Imagen 3.2.2. Comando ejecutado

Tras ejecutar el algoritmo veremos que se nos abrirá una ventana conteniendo el gráfico correspondiente a la primera muestra en el primer algoritmo mientras que en la propia

terminal se muestra la información correspondiente a esa muestra (Imagen 3.2.3). Para avanzar en las muestras tendremos que ir cerrando la ventana de la gráfica, y pasará automáticamente a la siguiente muestra. Sabremos que ya no hay más muestras cuando hayamos cerrado un total de 9 ventanas y el diálogo de pregunta se vuelva a mostrar en la cmd. (Imagen 3.2.4)

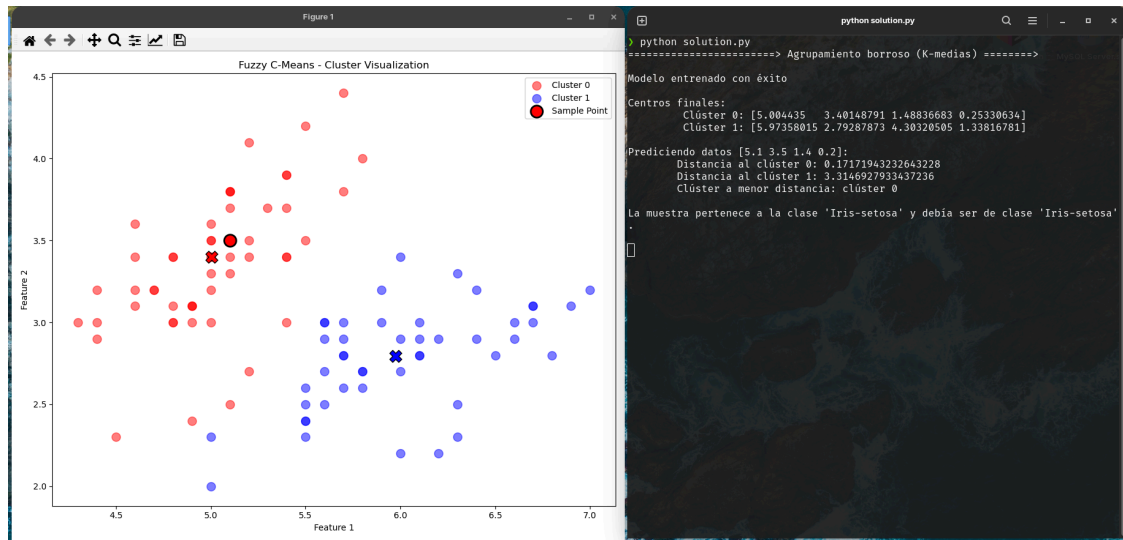


Imagen 3.2.3. Salida con la gráfica a la izquierda y la información a la derecha

```
megastacks13@mega13dev:~/Documents/UCM/IC/Practicas/GitHub/PathFind...
Distancia al clúster 1: 3.4531230843763945
Clúster a menor distancia: clúster 0
La muestra pertenece al cluster 'Iris-setosa' y debía ser de clase 'Iris-setosa'
.

=====> Algoritmo de Lloyd =====>
Modelo entrenado con éxito
Centros finales:
Clúster 0: [4.87828401 3.19589318 2.49772127 0.22129231]
Clúster 1: [6.31286014 3.02766772 4.45935644 1.13288247]
Prediciendo datos [6.9 3.1 4.9 1.5]:
Distancia al clúster 0: 3.3915435243205674
Distancia al clúster 1: 0.8239582519079304
Clúster a menor distancia: clúster 1
La muestra pertenece al cluster 'Iris-versicolor' y debía ser de clase 'Iris-versicolor'.

=====> Algoritmo de Lloyd =====>
Modelo entrenado con éxito
Centros finales:
Clúster 0: [4.87828401 3.19589318 2.49772127 0.22129231]
Clúster 1: [6.31286014 3.02766772 4.45935644 1.13288247]
Prediciendo datos [5.  3.4 1.5 0.2]:
Distancia al clúster 0: 1.0258535322781868
Distancia al clúster 1: 3.389733517542963
Clúster a menor distancia: clúster 0
La muestra pertenece al cluster 'Iris-setosa' y debía ser de clase 'Iris-setosa'
.
```

Imagen 3.2.4. Fin de ejecución