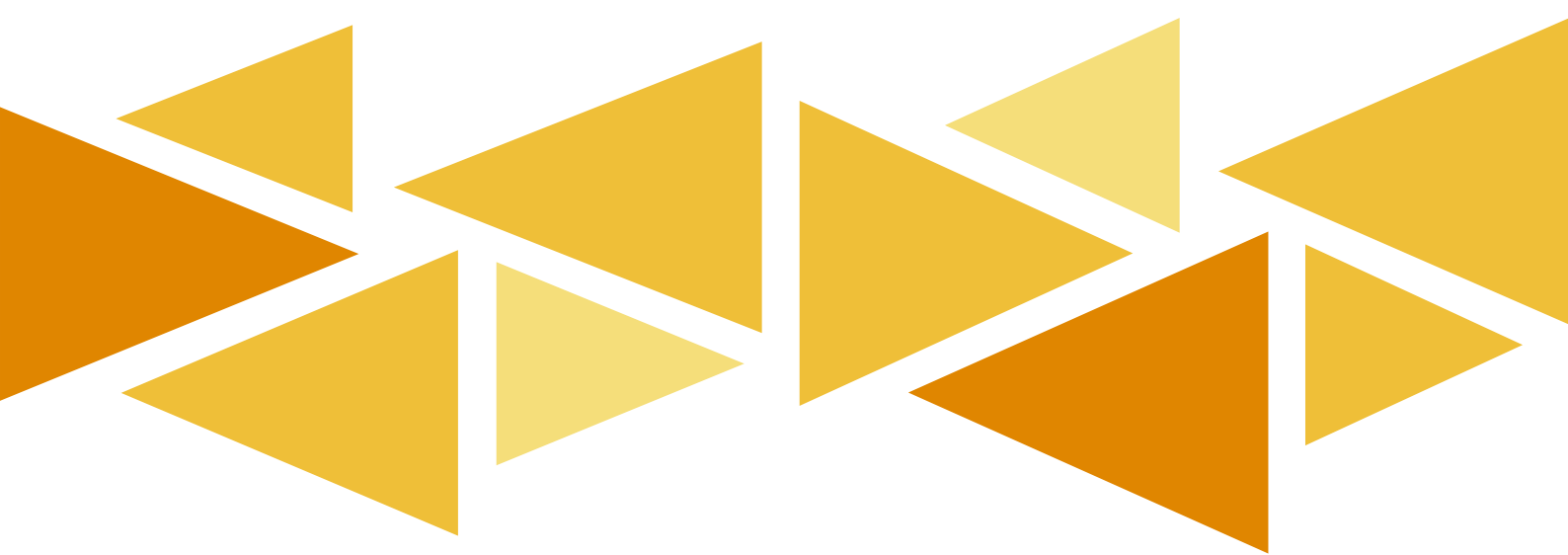


Realizado por:

- Beatriz Espinar Aragón
- Steven Mallqui Aguilar
- Rogger Huayllasco De la Cruz

MÉTODOS DE CLASIFICACIÓN

PRÁCTICA 3



2 de mayo de 2023
Ingeniería del Conocimiento
Grupo E – Ingeniería del Software



CONTENIDO

1. Introducción.....	1
2. Entorno	3
3. Implementación.....	4
3.1. Clase Main	4
3.2. Clase KMeans	5
3.3. Clase Lloyd	7
3.4. Clase Bayes	8
3.5. Clase Table	8
3.6. Clase MainWindow	9
3.7. Clase AlgorithmView	10
3.8. Clase Utilities.....	11
4. Ampliaciones.....	13
4.1. Los tres algoritmos.....	13
4.2. Comprobación clasificación.....	13
5. Ejemplos.....	15
5.1. Ejemplo K-medias borroso	15
5.2. Ejemplo Lloyd	17
5.3. Ejemplo Bayes	20

1. Introducción

La finalidad de este documento es presentar un informe detallado sobre la práctica, consistente en implementar una versión simplificada de uno de los tres métodos de clasificación más aplicados: el agrupamiento borroso (K-medias), Bayes y el algoritmo de Lloyd. Se explica el proceso y las decisiones tomadas para lograr una aplicación que permita visualizar el funcionamiento de los algoritmos.

En primer lugar, el K-medias borroso es un algoritmo que permite asignar un grado de pertenencia de una muestra dada a una de las k clases (o grupos) disponibles. En lugar de asignar un punto a una clase de manera absoluta (o pertenece o no pertenece), se le asigna un valor comprendido entre 0 y 1 que indica en qué medida pertenece a cada uno de los grupos. Por supuesto – y haremos uso de esta propiedad más adelante – la suma de los grados de pertenencia de una muestra ha de ser 1.

El algoritmo consiste en calcular la matriz de grados de pertenencia a partir de unos centros de clase iniciales (se mencionarán después en el algoritmo de Lloyd), para posteriormente ir actualizando los centros. Así pues, este algoritmo requiere de varios parámetros, que se definen a continuación:

- $\varepsilon = 0,01 \rightarrow$ La tolerancia determina el criterio de finalización, que sirve como límite de iteraciones para el algoritmo. El proceso explicado parará cuando la diferencia entre los centros anteriores y los obtenidos sea menor que este valor.
- $b = 2 \rightarrow$ El peso exponencial es un parámetro necesario para calcular tanto el grado de pertenencia de una muestra a una clase, como para recalcular los nuevos centros de las clases. En la implementación del algoritmo se podrá ver su efecto en la fórmula correspondiente.

Además de estos parámetros, es relevante mencionar otro que es utilizado en el cálculo de los grados de pertenencia: d_{ij} , donde i es la clase y j es la muestra. Este valor se define como la distancia euclídea al cuadrado de x_j (muestra j) y v_i (centro de la clase i), es decir:

$$d_{ij} = \|x_j - v_i\|^2 = (x_{j1} - v_{i1})^2 + (x_{j2} - v_{i2})^2 + \dots + (x_{jn} - v_{in})^2 = \sum_{k=1}^n (x_{jk} - v_{ik})^2$$

Por otro lado, Bayes es una técnica que permite calcular la probabilidad de que una muestra pertenezca a una determinada clase. Utiliza información previa (qué muestras hay y a qué clases pertenecen) para inferir la distribución de probabilidad de las variables de cada clase. Para ello, el algoritmo consiste en calcular la media de cada clase y su correspondiente matriz de covarianza.

Este algoritmo no requiere de parámetros iniciales, pues sólo hace uso de las muestras tomadas con las clases a las que pertenecen, y los cálculos que realiza no tienen ninguna complejidad matemática que haya que aclarar, más allá del uso de la trasposición de matrices. Puesto que obtener la matriz de covarianza consiste en multiplicar cada vector ($N \times 1$) por su traspuesto ($1 \times N$), la matriz resultante tendrá dimensiones $N \times N$, siendo N la dimensión de las muestras del problema (en este caso, 4).

Por último, el algoritmo de Lloyd es una versión del K-Medias consistente también en agrupar una serie de datos clasificándolos en los grupos o clases disponibles. Tras definir unos centros iniciales, itera a partir de ellos hasta alcanzar una convergencia en la que los centros no varíen. Para ello, en cada iteración se asigna a cada muestra el centro de la clase más cercana y se recalcula el centro.

Visto esto, es necesario definir una serie de parámetros iniciales que determinarán los cálculos de este algoritmo:

- $\varepsilon = 10^{-10} \rightarrow$ La tolerancia viene definida del mismo modo que en el K-Medias. Esto es, define el criterio de finalización, cuándo se considera que “los centros ya no varían”.
- $k_{max} = 10 \rightarrow$ El número de iteraciones máximo limita las iteraciones que ejecuta Lloyd, de forma que, si se alcanza, el algoritmo finaliza aunque no se cumpla el criterio de finalización.
- $\gamma(k) = 0,1 \rightarrow$ La razón de aprendizaje (constante) determina el ajuste que se aplica al actualizar los centros en cada iteración. Se utiliza, por tanto, en la fórmula que recalcula un centro.

Sumado a esto, hemos visto que tanto el algoritmo de K-Medias borroso como el de Lloyd requieren de unos centros iniciales. Estos pueden inicializarse de manera aleatoria, pero en este caso se cuenta con unos valores apropiados para los datos del problema:

$$v = \begin{bmatrix} 4.6 & 3.0 & 4.0 & 0.0 \\ 6.8 & 3.4 & 4.6 & 0.7 \end{bmatrix}^1$$

Habiendo presentado los tres algoritmos, veamos los datos de entrada de los que disponemos. Contamos con cuatro ficheros, todos almacenados en el directorio `input_files/` del proyecto, aunque el programa podría ser fácilmente refactorizado para permitir al usuario introducir sus propios ficheros. En el fichero `Iris2Clases.txt` se encuentran todas las muestras, formadas por cuatro atributos (dimensiones), y que pueden pertenecer a dos clases distintas (por eso se han definido dos centros iniciales): `Iris-setosa` o `Iris-versicolor`.

Además, existen otros tres ficheros: `TestIris01.txt`, `TestIris02.txt` y `TestIris03.txt`; correspondientes a tres ejemplos que los algoritmos deben clasificar una vez terminado su entrenamiento.

Ahora que ya se ha planteado el objetivo de los algoritmos de clasificación, se presentan las secciones que componen este documento para facilitar su lectura:

- [Entorno](#). Se expone brevemente el entorno utilizado para la implementación de los algoritmos, así como el lenguaje de programación elegido y otras herramientas de interés.
- [Implementación](#). Se explica con mayor detalle las decisiones relativas al diseño e implementación, tales como las estructuras de datos empleadas o el funcionamiento específico de cada algoritmo.
- [Ampliaciones](#). Se presentan las ampliaciones realizadas, incluyendo la lógica implementada y los cambios aplicados sobre el apartado anterior.
- [Ejemplos](#). Se propone al usuario una serie de ejemplos ilustrativos para mostrar el funcionamiento de los algoritmos, incluyendo los cambios con respecto a las ampliaciones.

¹ La primera fila corresponde a v_1 , el centro de la primera clase; y la segunda fila corresponde a v_2 , el centro de la segunda clase.

2. Entorno

Se explica en este primer apartado el entorno de programación empleado para el desarrollo del algoritmo, con las herramientas y librerías utilizadas.

En primer lugar, el lenguaje de programación escogido para implementar el algoritmo ha sido Python. En particular, la versión más reciente de este lenguaje: 3.11.2. Es la misma decisión que se tomó en las prácticas anteriores (Implementación del Algoritmo A* y del ID3), y se han aplicado los mismos criterios: el amplio rango de librerías muy útiles que proporciona Python y el interés personal de los miembros del equipo por familiarizarse con este lenguaje.

Para el desarrollo del código, de igual manera, se ha optado por utilizar el entorno de desarrollo PyCharm, por estar especializado en Python y por las herramientas que ofrece para detectar errores rápidamente, así como para mantener buenas prácticas de programación.

Por otro lado, en lo relativo a las librerías utilizadas, se listan a continuación las más relevantes para este proyecto:

- `Tkinter` → Se trata de una librería preinstalada de Python que proporciona una interfaz gráfica de usuario, y es una de las más populares para el desarrollo de aplicaciones de escritorio por su simplicidad y flexibilidad. En esta práctica ha sido imprescindible para el desarrollo de la GUI (véase [Clase MainWindow](#), [Clase AlgorithmView](#) y [Clase Table](#)), pues todas las ventanas, los botones, la tabla con los datos de entrada, el resultado de los algoritmos, etc, han sido dibujados con herramientas de esta librería.
- `NumPy` → Se trata de una librería que permite trabajar con matrices multidimensionales, proporcionando además una gran cantidad de funciones matemáticas y otras herramientas de trabajo con matrices. En esta práctica se ha utilizado para representar tanto los vectores correspondientes a las muestras o los centros, y poder operar con ellos (restas, multiplicaciones, trasposiciones...), como para calcular la matriz de covarianza en el algoritmo de Bayes.

Además, se ha utilizado `pyinstaller`, una herramienta que permite convertir el programa de Python en un ejecutable independiente empaquetando todas las dependencias necesarias para poder ejecutarse en cualquier máquina sin tener que instalar el lenguaje o las bibliotecas utilizadas.

Por último, el equipo ha hecho uso de otras tecnologías que han permitido desarrollar la práctica, entre las cuales cabe destacar: GitHub y GitHub Desktop, para el control de versiones y facilitar el trabajo en equipo mediante la creación de un repositorio común; Discord o Whatsapp, como plataformas de comunicación para trabajar simultáneamente sobre el código; Word (Office 365) con OneDrive, para la elaboración de la memoria; y Paint, para el diseño de la interfaz gráfica.

3. Implementación

En esta sección se profundiza en la implementación de los algoritmos, explicando los módulos en los que se ha dividido el proyecto y las estructuras de datos utilizadas. Además, se explican los pasos que siguen los algoritmos y cómo han sido plasmados en la interfaz para que el usuario visualice todo el proceso.

El proyecto se ha dividido en 8 módulos principales, que se explicarán a lo largo de este apartado. En «Main» se explicará el funcionamiento inicial de la aplicación, procesando los datos e inicializando la interfaz gráfica con la clase «MainWindow». Esta incluye toda la generación de la GUI, haciendo uso del módulo «AlgorithmView», que completa la interfaz de manera dinámica según el algoritmo seleccionado por el usuario, y «Table», para los datos de entrada. Para la implementación de los algoritmos se ha creado una clase para cada algoritmo: «KMeans», «Lloyd» y «Bayes» (en estas secciones es donde se explicará el flujo principal del algoritmo). Por último, en «Utilities» se incluyen las constantes empleadas por los demás módulos para realizar sus funciones.

Es relevante destacar que, además de las explicaciones más detalladas de este informe, el código de cada clase está documentado, de forma que se incluye al principio de cada clase un breve comentario descriptivo del objetivo de la clase y los atributos que necesita. Por supuesto, el resto del código está igualmente comentado.

NOTA: Tal y como se explica en el apartado de [Ampliaciones](#), se ha decidido implementar todos los algoritmos propuestos, en lugar de sólo uno de ellos. Para facilitar la lectura, las explicaciones de los tres algoritmos se incluirán en este apartado (en lugar de incluir aquí uno de ellos, y los otros dos en el apartado de las ampliaciones), aunque su implementación forme parte de una de las ampliaciones propuestas. La otra ampliación realizada sí se omitirá en esta sección, para detallarse posteriormente en el apartado correspondiente.

3.1. Clase Main

En primer lugar, contamos con el módulo `Main` (realmente no se ha creado una clase, sólo un método `main()` con otras funciones auxiliares, pues no requiere de atributos), que inicia la aplicación desde donde se crea la interfaz gráfica. Para dar comienzo a la aplicación, la función principal debe seguir dos pasos:

- 1) Leer y procesar los datos de entrada
- 2) Iniciar la interfaz gráfica

El segundo paso sólo consiste en crear una instancia de la [Clase MainWindow](#), que se encargará de inicializar la GUI, invocando al constructor con los datos de entrada. Veamos pues cómo se procesan, recordando el formato de los ficheros explicado en el apartado de [Introducción](#):

- Datos → Dado que existen dos clases (`Iris-setosa` e `Iris-versicolor`), se ha optado por almacenar las muestras en dos vectores diferentes, uno para cada clase: `data_set` y `data_ver`. Cada uno de ellos es una lista (array) de muestras, donde cada muestra es un vector de la librería `NumPy` de 4 posiciones, una por cada dimensión. Ahora bien, para crear este vector `NumPy` exige dar unas dimensiones iniciales que, teóricamente, no conocemos de antemano. Por ello, se procesa primero cada muestra guardándola en una lista `[]` y, posteriormente, se genera el vector. Para procesar cada muestra (cada línea del fichero), se

obtiene cada elemento utilizando como carácter separador la ',', y con la última palabra se decide en cuál de las dos listas insertar la muestra (`data_set` o `data_ver`).

- `examples` → Se trata de una lista de los ejemplos que se van a clasificar. Cada ejemplo se lee en una función auxiliar que recibe el fichero donde se encuentra, y se procesa del mismo modo que las muestras anteriores, utilizando la función que convierte la lista `[]` en vector de NumPy.

3.2. Clase KMeans

Empezando con las clases que ejecutan los algoritmos, veamos la clase «KMeans», que implementa el primero de ellos: K-medias borroso. Las clases de los tres algoritmos actúan como clases estáticas (no tienen atributos, y todos sus métodos son estáticos), pues no es necesario almacenar ninguna información. De igual forma, como veremos ahora, las clases de los algoritmos cuentan con dos métodos comunes principales: `execute()` y `classify()`. El primero ejecuta el entrenamiento del algoritmo, mientras que el segundo clasifica un ejemplo dado, una vez calculados los centros (o medias, en el caso de Bayes).

El `execute()` hace uso de una serie de funciones auxiliares, que permiten reducir código repetido:

- `dist()` → Es uno de los métodos básicos más importantes, pues se utiliza en todos los demás. Calcula la distancia euclídea entre dos vectores dados:
→ $\|x - v\|$
- `check_tolerance()` → Este método comprueba si se cumple el criterio de finalización. Dados los centros correspondientes a una iteración t (vector `old`) y los correspondientes a la siguiente iteración $t + 1$ (vector `new`), comprueba si hay alguno que cumpla:
→ $\Delta = \|v_i^{(t+1)} - v_i^{(t)}\| = \|new_i - old_i\| \geq \varepsilon$
Si ninguno lo cumple (si todas las diferencias son menores que la tolerancia), entonces se cumple el criterio de finalización y el algoritmo debe terminar.
- `update_centroid()` → Por último, se utiliza un método auxiliar que actualiza el centro de la clase i (v_i) a partir de la matriz de grados de pertenencia U y las muestras de \vec{x} . Para ello, aplica la fórmula siguiente:

$$\rightarrow v_i = \frac{\sum_{j=1}^n [P(c_i/x_j)]^b * x_j}{\sum_{j=1}^n [P(c_i/x_j)]^b}$$

Visto esto, pasemos a analizar los pasos que sigue el algoritmo (`execute()`). K-medias borroso toma como entrada las muestras y obtiene como resultado los nuevos centros. Se ha decidido también devolver el número de iteraciones que han sido necesarias para llegar a los centros solución. Así pues, los pasos que sigue el algoritmo son:

1. Inicialización (parte 1): El primer paso es fusionar las dos listas de muestras que recibe como entrada. Recordemos que el input venía separado en listas para cada una de las clases, pero K-medias borroso no tiene en cuenta esto, sino que trata todas las muestras a la vez. También se inicializan la $n = n^\circ \text{ de muestras}$ y la $c = n^\circ \text{ de clases}$, y se inicializa el contador de iteraciones a 1.
2. Inicialización (parte 2): Aparte de lo anterior, es necesario inicializar los centros. Para la iteración actual (t), se inicializan a los valores constantes incluidos en la [Clase Utilities](#). En

cambio, para los de la siguiente iteración ($t + 1$), que acabarán siendo los centros solución, sólo creamos una lista vacía de dos posiciones (una por cada clase).

3. **Bucle:** Empieza entonces el bucle. Dada la estructura del código, se ha optado por hacer una adaptación de Python de un `do-while`, de forma que la primera vez siempre se ejecuta el código del bucle. Es al final de la iteración cuando se comprueba la condición de salida que, en este caso, es que los centros cumplan el criterio de finalización. Para comprobarlo basta invocar a la función `check_tolerance()` ya explicada. Veamos entonces los pasos seguidos dentro del bucle:
4. **Matriz de grados de pertenencia:** Primero hay que calcular la matriz de grados de pertenencia, aplicando esta fórmula a cada posición de la matriz:

$$\rightarrow U_{ij} = P(v_i/x_j) = \frac{1/d_{ij}^{1/(b-1)}}{\sum_{r=1}^c (1/d_{rj}^{1/(b-1)})}$$

Necesitamos, por tanto, un primer bucle que recorre las filas de la matriz (correspondientes a las clases i) y uno interno que recorre las columnas de la matriz (correspondientes a las muestras j). Para calcular los d_{ij} se utiliza la función `dist()` presentada anteriormente.

- a. **Última fila:** Para la última fila de la matriz se pueden simplificar los cálculos. Dado que cada columna de la matriz (la suma de grados de pertenencia de una muestra) ha de sumar 1, podemos calcular los valores de la última fila sumando los valores de las filas anteriores y restándoselo a 1, y así evitarnos la fórmula anterior. Para poder hacer esto de manera eficiente, se lleva una variable auxiliar (un array) que acumula la suma de cada columna de la matriz según se van calculando los grados de pertenencia en el bucle.
5. **Actualizar los centros:** Una vez calculada la matriz de grados de pertenencia, habrá que actualizar cada uno de los centros, haciendo uso de la función `update_centroid()`.
6. **Terminar bucle:** Sólo queda comprobar la condición de salida del bucle. Si se cumple, la función termina y devuelve los centros obtenidos (`new[]`). En caso contrario, el bucle se vuelve a ejecutar, se incrementa el contador de iteraciones, y los centros de la iteración $t + 1$ (`new[]`) pasan a ser los de la iteración t (`old[]`).

Recordemos que, aparte del `execute()`, la clase contaba con otro método: `classify()`. Este método recibe los centros obtenidos y el ejemplo a clasificar, y devuelve la clase (en formato string) a la que pertenece el ejemplo. Para saber a qué clase pertenece, calcula la distancia de la muestra a cada uno de los centros. Aquel que esté más cerca (menor distancia) del ejemplo será el centro de la clase a la que pertenece.

Para calcular esta distancia mínima, basta inicializar una variable auxiliar a infinito (para que la primera vez siempre se asigne el valor), que tendrá como invariante “ser la distancia menor encontrada hasta el momento”. Para ir calculando las distancias se utiliza la función `dist()`, cuyo resultado se eleva al cuadrado². Junto a la distancia, se lleva una variable paralela que almacena el índice del centro más cercano encontrado hasta el momento. De esta manera, al terminar el bucle

² Se aplica el cuadrado para mantener los criterios establecidos en los problemas de clase. Nótese que, si la distancia a un centro es menor que a otro centro, entonces su cuadrado también lo es.

que recorre los centros, en función del índice obtenido devolverá “Iris-setosa” o “Iris-versicolor”.

3.3. Clase Lloyd

Pasemos pues a la clase correspondiente al segundo algoritmo: «Lloyd». Antes de nada, esta clase comparte muchas similitudes con la [clase anterior](#), por lo que algunos detalles de implementación se omitirán para evitar redundancias.

Así, la clase actúa como estática, tiene los métodos `execute()` y `classify()`, análogos a los que tenía el algoritmo de K-medias borroso. La clase utiliza además una serie de funciones auxiliares:

- `dist()` → Es el mismo método que en K-medias, consistente en calcular la distancia euclídea entre dos vectores dados: $\|x - c\|$.
- `check_tolerance()` → Es el mismo método que en K-medias, encargado de comprobar si se cumple el criterio de finalización: $\Delta = \|v_i^{(t+1)} - v_i^{(t)}\| = \|new_i - old_i\| \geq \varepsilon$. La única diferencia es la constante que utilizan para la tolerancia ε .
- `closest_centroid()` → Por último, se hace uso de una nueva función, que calcula el índice del centro más próximo a una muestra dada x_j de entre los centros disponibles $c[]$. Es un algoritmo clásico de cálculo lineal del mínimo, y hace uso de `dist()`.

Ahora sí, pasemos al método más importante. En este caso, `execute()` recibe y devuelve los mismos argumentos que en la [Clase KMeans](#), y sigue los siguientes pasos:

1. Inicialización (parte 1): Mismos pasos que en K-medias, se fusionan las dos listas de muestras (Lloyd también las trata de forma conjunta) y se inicializan n , c y el contador de iteraciones.
2. Inicialización (parte 2): Igual que en K-medias, se inicializan los centros. Los de la iteración t con los valores constantes de la [Clase Utilities](#), y los de la iteración $t + 1$ con una lista vacía.
3. Bucle: Empieza entonces el bucle. Igual que antes, se ha implementado un `do-while`, de forma que la primera vez siempre se ejecuta el código del bucle. Es al final de la iteración cuando se comprueba la condición de salida que, en este caso, es que los centros cumplan el criterio de finalización y que no se haya sobrepasado el **límite de iteraciones** (parámetro k_{max}).
4. Calcular centros más próximos: El bucle consiste en recorrer todas las muestras y, para cada una, calcular el centro más próximo utilizando `closest_centroid()`. Una vez calculado, se actualiza el centro aplicando la siguiente fórmula:

$$\rightarrow c_i^{(t+1)} = c_i^{(t)} + \gamma(x_j - c_i^{(t)})$$

NOTA: Recordemos que γ es una constante definida en la [Clase Utilities](#).

5. Terminar bucle: Exactamente la misma funcionalidad que el paso 6 del `execute()` de la [Clase KMeans](#). Sólo cambia la condición de salida, como se ha explicado en el paso 3.

Por último, se implementa la función `classify()`, que es completamente análoga a la que se implementó en K-medias borroso. Recibe y devuelve los mismos argumentos, y calcula la distancia del ejemplo a cada centro, de forma que aquel más próximo indica la clase a la que pertenece.

3.4. Clase Bayes

El último algoritmo implementado es «Bayes», que tiene más diferencias con respecto a los dos anteriores. En cualquier caso, sigue actuando como clase estática, y tiene los métodos `execute()` y `classify()` con el mismo propósito. Veamos los métodos auxiliares que requiere esta clase:

- `calc_mean()` → Este método calcula la media de una clase i , que tiene las muestras $x[]$ pasadas como parámetro de entrada. Recibirá, por tanto, las muestras del array `data_set` o bien del array `data_ver`, y aplica la fórmula clásica de la media:

$$\rightarrow m_i = \frac{1}{n} (\sum_{j=1}^n \vec{x}_j)$$

- `calc_matrix()` → Este método calcula la matriz de covarianza de la clase i , que tiene las muestras $x[]$ y tiene como media m , ambas pasadas como parámetros de entrada. Se calcula aplicando la siguiente fórmula:

$$\rightarrow c = \frac{1}{n} * \sum_{j=1}^n (\vec{x}_j - \vec{m}) * (\vec{x}_j - \vec{m})^t$$

Nótese que las muestras tenían formato array de NumPy, así que para poder calcular su traspuesta y multiplicar las matrices se han formateado a matrices gracias a la función `reshape()` que proporciona la librería. Para trasponer y multiplicar las matrices también se han utilizado operadores y funciones de NumPy ya implementadas.

En este caso, el método `execute()` es mucho más sencillo que los vistos anteriormente, pues sólo consiste en llamar a las funciones anteriores para cada una de las clases. Primero se calculan las medias, y luego las matrices de covarianza. La función devuelve un array de medias y otro de matrices.

Por otro lado, el método `classify()` recibe en este caso las medias y las matrices (en lugar de los centros que recibían los otros dos algoritmos), aparte del ejemplo a clasificar. El problema consiste igualmente en encontrar el centro más próximo a la muestra; pero, en Bayes, para calcular la distancia de la muestra a cada uno de los centros se aplica esta fórmula:

$$\rightarrow d_M(\vec{x}, \vec{m}_i / c_i) = (\vec{x} - \vec{m}_i) * I * (\vec{x} - \vec{m}_i)^t$$

Donde $I = c_i^{-1}$ (la inversa de la matriz de covarianza). Para estos cálculos, como siempre, se utilizan funciones que proporciona NumPy.

3.5. Clase Table

La clase «Table» se ha creado para dibujar las tablas de los datos de entrada iniciales de las dos clases disponibles. Dispone de los siguientes atributos:

- `title` → Es el título que tomará la tabla, correspondiente a la clase «Iris-setosa» o «Iris-versicolor».
- `data` → Son los datos (muestras) de la tabla. El constructor recibirá, por tanto, o bien `data_set` o bien `data_ver`.
- `frame` → Es el frame en el que se dibuja la tabla (frame izquierdo del panel central de contenido, véase [Clase MainWindow](#)).

Para dibujarla, se genera primero el título con una `Label` de `tkinter` y se añade el `scrollbar` para ajustar el contenido, y luego se utilizan las `Entry`'s para representar las celdas de la tabla.

3.6. Clase `MainWindow`

En esta sección presentaremos una de las dos grandes clases de la capa de presentación que se han implementado en esta práctica: la clase «`MainWindow`». Genera la base de la interfaz gráfica, que luego será completada por la [Clase `AlgorithmView`](#), y da funcionalidad a los botones que permiten seleccionar el algoritmo a ejecutar.

Así pues, la clase cuenta con los siguientes atributos:

- `data_set` → Es la lista de muestras leídas correspondientes a la clase “`Iris-setosa`”.
- `data_ver` → Es la lista de muestras leídas correspondientes a la clase “`Iris-versicolor`”.
- `examples` → Es la lista de ejemplos leídos que van a clasificar los algoritmos una vez ejecutados.
- `main_window` → Es la ventana de `tkinter` donde se pinta la interfaz, actúa como objeto raíz de la GUI.
- `main_frame` → Es el frame principal de la aplicación, el que ocupa toda la ventana.
- `content_frame` → Es el frame central donde se dibuja toda la información de la aplicación. Está formado por un `left_frame` y un `right_frame`, como se verá más adelante, y es necesario mantenerlo como atributo de la clase porque es referenciado desde varios métodos.
- `right_frame` → Es el frame derecho, donde se genera la información del algoritmo de manera dinámica dependiendo del algoritmo seleccionado. Dado que se destruirá y construirá cada vez que se seleccione un algoritmo con los botones del panel izquierdo, hace falta guardarlo como atributo de la clase. El contenido de este panel será manejado en su totalidad por la [Clase `AlgorithmView`](#).

Antes de ver los métodos que contiene la clase, veamos cómo quedaría la disposición general de la interfaz según lo ya explicado:

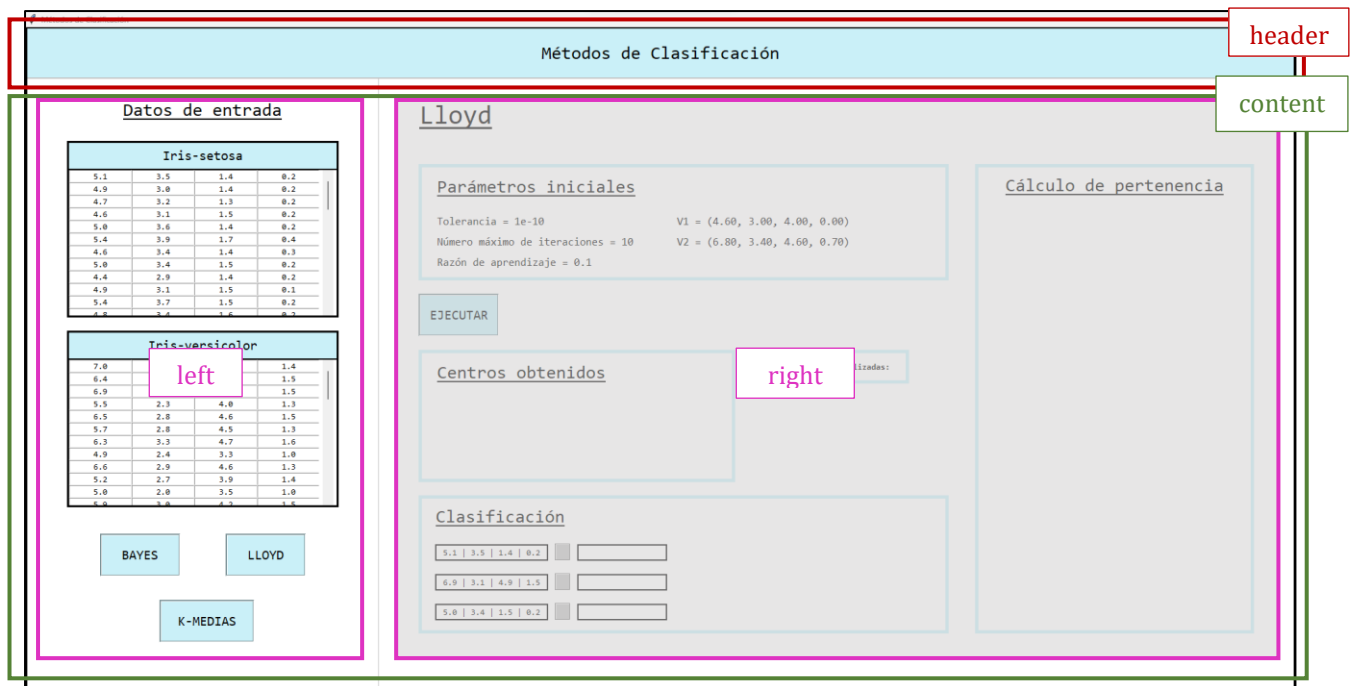


Ilustración 1. Disposición paneles interfaz

En cuanto a los botones, los tres tienen la misma funcionalidad: reiniciar el panel derecho y luego crear una instancia de la [Clase AlgorithmView](#) para que genere su contenido, indicando el algoritmo que ha pulsado el usuario.

Por otro lado, en cuanto a los otros métodos, la clase tiene una función `init_gui()` que inicializa la interfaz gráfica, llamando a su vez a los métodos que crean los frames explicados anteriormente. Nótese que al construir el `left_frame` dentro del `content_frame` es necesario hacer uso de la [Clase Table](#) para mostrar los datos de entrada.

3.7. Clase AlgorithmView

La otra clase imprescindible para implementar la interfaz de usuario es «AlgorithmView». Esta clase genera el contenido del panel derecho dentro del `content_frame` (véase [Clase MainWindow](#)), dependiendo de algoritmo seleccionado por el usuario. Así pues, veamos los atributos que posee:

- `alg` → Es una cadena de caracteres que indica el algoritmo seleccionado. Será útil para distinguir casos en cada funcionalidad, además de al crear los componentes de la GUI.
- `data_set` → Es la lista de muestras leídas correspondientes a la clase “Iris-setosa”.
- `data_ver` → Es la lista de muestras leídas correspondientes a la clase “Iris-versicolor”.
- `examples` → Es la lista de los tres ejemplos a clasificar.
- `centroids` → Son los centros obtenidos por el algoritmo de Lloyd o K-medias (None si el algoritmo seleccionado es Bayes). Es necesario guardarlos como atributo para poder invocar después al `classify()`.
- `means y matrices` → Son las medias y las matrices de covarianza obtenidas por Bayes (None si el algoritmo seleccionado es K-medias o Lloyd). Igual que lo anterior, son necesarios para llamar al método que clasifica.
- `frame` → Es el frame padre donde se dibuja este panel.

- `content_frame`, `calculus_frame`, `centroids_frame`, `examples_frame`, `params_frame`, `means_frame`, `check_frame`, `matrix_frame`, → Son frames que es necesario almacenar como atributos porque su contenido es modificado de forma dinámica en distintos métodos de la clase.
- `execute_button` → Es el botón para ejecutar el algoritmo en cuestión. Necesario como atributo porque se habilita/deshabilita desde otros métodos.
- `iter_label` → Label de tkinter donde se escribe el número de iteraciones realizadas por K-medias/Lloyd. Necesaria como atributo porque su contenido varía de forma dinámica y es modificado desde distintos métodos de la clase.
- `example_button1`, `example_button2` y `example_button3` → Son los botones con los que se clasifica cada uno de los ejemplos. Misma explicación que el `execute_button`.
- `example_class_label1`, `example_class_label2` y `example_class_label3` → Labels de tkinter donde se escribe la clase a la que pertenece cada uno de los ejemplos. Misma explicación que la `iter_label`.

En cuanto a los métodos, contamos con el método principal `init_gui()` que inicializa los paneles principales de este panel, para lo cual hace uso de métodos auxiliares que construyen los subpaneles, teniendo en cuenta el algoritmo. Hay un método para los algoritmos de K-medias y Lloyd, que tienen la misma disposición, y otro para el de Bayes. De igual forma, hay métodos que imprimen los resultados según el algoritmo y otros que imprimen los parámetros iniciales (excepto en Bayes, donde no hay).

Además, los tres algoritmos comparten el panel de los ejemplos, construido por otro método privado que, a su vez, construye cada ejemplo con una función auxiliar. Otro elemento compartido, construido por su correspondiente método, es el botón de ejecutar. A partir de estos botones (ejemplos y ejecutar) surgen los métodos que implementan sus correspondientes funcionalidades, algo más relevante para la práctica.

Por un lado, el botón de ejecutar primero se deshabilita (una vez ejecutado, no puede volver a pulsarse), para después invocar al `execute()` del algoritmo que corresponda, e imprimir los resultados que este devuelva, además de guardarlos en los atributos de la clase. El último paso es habilitar los botones para clasificar los ejemplos, pues ahora ya se cuenta con la información necesaria.

Por otro lado, el botón de clasificar un ejemplo primero invoca al `classify()` del algoritmo seleccionado, habilita/deshabilita los botones de clasificar adecuados (él mismo queda deshabilitado, se habilitan los demás) y escribe el resultado en la Label correspondiente (la de ese ejemplo, y las de los otros ejemplos se resetean).

Por último, es relevante mencionar que existen otros métodos auxiliares muy convenientes para convertir los datos en texto. Por ejemplo, `create_coord()` recibe un vector y lo escribe en formato texto, mostrando sólo los dos primeros decimales. Nótese además que para imprimir la matriz de covarianza se utiliza la propia función de NumPy que transforma la matriz en una cadena de caracteres, indicando además que los valores se redondeen a 3 decimales.

3.8. Clase Utilities

Por último, se presenta la clase «Utilities» (realmente no es una clase, sólo se define en un fichero aparte para mantener la estructura del código), donde se definen las constantes utilizadas por el

resto de las clases. Por un lado, aparecen parámetros referentes a la interfaz gráfica, incluyendo la dimensión de la pantalla, las fuentes de letra y los colores utilizados. Además, se definen los parámetros iniciales expuestos en el apartado de [Introducción](#).

4. Ampliaciones

En este apartado se presentan las ampliaciones realizadas sobre la práctica básica, que incluye únicamente la implementación de uno de los algoritmos, y que sólo ejecuta el entrenamiento del algoritmo y la clasificación de los ejemplos proporcionados.

En este caso, se ha optado por implementar las dos ampliaciones propuestas: implementar los tres algoritmos propuestos (K-medias borroso, Lloyd y Bayes), y mostrar y comprobar el funcionamiento de la clasificación de cada uno de los ejemplos.

Estas ampliaciones sobre el enunciado básico se explican en los siguientes apartados, junto a una explicación de la refactorización del código que han provocado y cómo han sido implementadas.

4.1. Los tres algoritmos

La primera ampliación consiste en implementar los tres algoritmos en lugar de uno sólo. Ahora bien, los apartados anteriores ya han sido explicados teniendo en cuenta esta ampliación, por lo que no es necesario dar ninguna aclaración más. Para comprobar la implementación de cada algoritmo, véanse los apartados de las clases correspondientes: [K-medias borroso](#), [Lloyd](#) y [Bayes](#).

Si se hubiera implementado sólo uno de los tres algoritmos, no habrían hecho falta los botones para seleccionar el algoritmo, las clases correspondientes a los otros dos algoritmos, así como todos los métodos de las vistas relacionadas con ellos.

4.2. Comprobación clasificación

Esta ampliación, tal y como se ha interpretado, consiste en comprobar los cálculos y el resultado que producen los algoritmos al clasificar los ejemplos. Para ello, se han considerado varias opciones o alternativas que ayudan a ilustrar el comportamiento de los algoritmos para los ejemplos dados.

En primer lugar, se comprobarán los cálculos matemáticos que realizan en el apartado de [Ejemplos](#). Por otro lado, se ha optado por permitir al usuario visualizarlos de manera simplificada en una zona extra que se ha añadido al panel de cada algoritmo. En la sección de “Cálculo de pertenencia”, tal y como se mostrará en los ejemplos, se presentan los datos, cálculos y conclusiones que obtiene el algoritmo que está siendo ejecutado.

Para poder implementar esta funcionalidad, se han hecho los cambios correspondientes (y evidentes) en la [Clase AlgorithmView](#) que genera la vista. Ahora bien, lo interesante reside en cómo se rellena el contenido de esta sección:

En dicha clase, se ha creado un método público (`draw_info()`) que será invocado por los algoritmos y que es el encargado de generar el texto de la `Label` de `tkinter` que contendrá la información correspondiente. Este método recibe una serie de parámetros:

- `example` → Es el ejemplo que se está clasificando.
- `i_sol` → Es el índice del centro de la clase a la que pertenece el ejemplo clasificado.
- `dist_sol` → Es la distancia del ejemplo clasificado al centro de la clase a la que pertenece.
- `class_sol` → Es la clase (en formato `string`) a la que pertenece el ejemplo clasificado.
- `data` → Es un vector que contiene el resto de cálculos relevantes, dependiendo del algoritmo que se ha ejecutado para clasificar el ejemplo.

Así pues, para el K-medias borroso se muestra el ejemplo a clasificar, el cálculo de las distancias a cada centro, la conclusión (“La menor distancia es x , por lo tanto pertenece a x clase”), los grados de pertenencia y la conclusión formulada de otro modo (“El mayor grado de pertenencia es a la clase x , por lo que pertenece a esa clase”). Por otro lado, para el algoritmo de Lloyd, se muestra la misma información que en K-medias, omitiendo la parte de los grados de pertenencia. Por último, en Bayes se muestra también el ejemplo y la conclusión y, para cada clase, se imprime la media, la matriz de covarianza, y las fórmulas/cálculos aplicados para obtener la distancia.

Volviendo a cuestiones de implementación, las funciones que imprimen esta información en la vista no tienen aspectos que merezca la pena destacar, más allá del ajuste de decimales en los valores. Sin embargo, veamos qué ocurre en la implementación de los algoritmos:

Los métodos `classify()` han requerido de una cierta refactorización en los tres algoritmos, especialmente para construir el vector `data` que contiene los cálculos. Este vector se va rellenando en cada vuelta de los bucles, con la información correspondiente, y el resto de parámetros adquieren su valor al terminar la función. Para poder enviar esta información a la vista, el `classify()` ahora recibe un argumento extra, que es la instancia de la [Clase AlgorithmView](#), de forma que la última instrucción antes de retornar consiste en invocar a la vista para que ejecute su método `draw_info()` con los parámetros calculados.

Es importante apuntar que, antes de realizar esta refactorización, el `classify()` de K-medias sólo calculaba las distancias, obteniendo la mínima para concluir la clase a la que pertenece la muestra. Ahora, sin embargo, con objeto de ilustrar mejor el funcionamiento del algoritmo, se van calculando también los grados de pertenencia. Así, se puede comprobar que el centro con la mínima distancia coincide, de hecho, con el mayor grado de pertenencia.

Gracias a esta ampliación, el usuario puede visualizar cómo trabajan los algoritmos a bajo nivel y qué cálculos realizan para llegar a la conclusión adecuada. Esta medida, además, facilita la depuración del código al equipo, pudiendo comprobar valores y encontrar errores más fácilmente en los algoritmos.

5. Ejemplos

En esta última sección se proponen tres ejemplos que ilustran el comportamiento de cada algoritmo. En cada ejemplo se analiza primero el entrenamiento del algoritmo y, a continuación, se clasifica uno de los ejemplos dados, con la información correspondiente a la segunda ampliación implementada.

Antes de nada, veamos cómo es la aplicación nada más iniciarla:

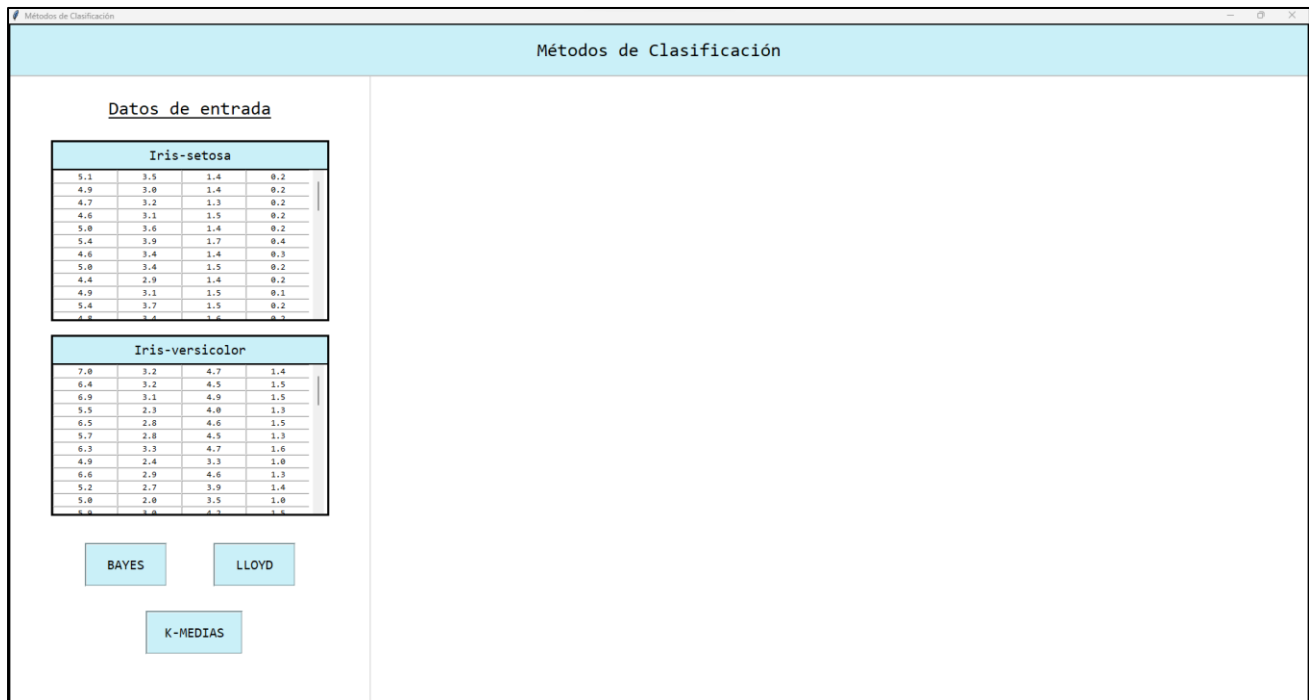


Ilustración 2. Interfaz inicial

A la izquierda podemos ver las muestras iniciales, unas asociadas a Iris-setosa y otras a Iris-versicolor. Además, contamos con un botón para seleccionar cada uno de los algoritmos disponibles.

5.1. Ejemplo K-medias borroso

En este primer ejemplo ejecutaremos el algoritmo K-medias borroso para obtener los centros de las clases Iris-setosa e Iris-versicolor. El primer paso tras iniciar la aplicación es pulsar en el botón correspondiente a este algoritmo, obteniendo la siguiente vista:

Métodos de Clasificación

Datos de entrada

Iris-setosa			
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5.0	3.6	1.4	0.2
5.4	3.9	1.7	0.4
4.6	3.4	1.4	0.3
5.0	3.4	1.5	0.2
4.4	2.9	1.4	0.2
4.9	3.1	1.5	0.1
5.4	3.7	1.5	0.2
4.8	3.4	1.6	0.3

Iris-versicolor			
7.0	3.2	4.7	1.4
6.4	3.2	4.5	1.5
6.9	3.1	4.9	1.5
5.5	2.3	4.0	1.3
6.5	2.8	4.6	1.5
5.7	2.8	4.5	1.3
6.3	3.3	4.7	1.6
4.9	2.4	3.3	1.0
6.6	2.9	4.6	1.3
5.2	2.7	3.9	1.4
5.0	2.0	3.5	1.0
6.8	3.0	4.3	1.6

BAYES LLOYD

K-MEDIAS

K-medias borroso

Parámetros iniciales

Tolerancia = 0.01 V1 = (4.60, 3.00, 4.00, 0.00)
 Peso exponencial = 2 V2 = (6.80, 3.40, 4.60, 0.70)

EJECUTAR

Centros obtenidos

Iteraciones realizadas:

Clasificación

5.1 | 3.5 | 1.4 | 0.2

6.9 | 3.1 | 4.9 | 1.5

5.0 | 3.4 | 1.5 | 0.2

Cálculo de pertenencia

Ilustración 3. Ejemplo K-medias inicio

En la imagen se destacan con un recuadro rosa los parámetros iniciales para este algoritmo, que incluyen la tolerancia, el peso exponencial y los centros iniciales. Si pulsamos el botón de ejecutar, se ejecutará el entrenamiento del algoritmo, y podremos ver los centros obtenidos y el número de iteraciones que han sido necesarias para cumplir el criterio de finalización:

Métodos de Clasificación

Datos de entrada

Iris-setosa			
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5.0	3.6	1.4	0.2
5.4	3.9	1.7	0.4
4.6	3.4	1.4	0.3
5.0	3.4	1.5	0.2
4.4	2.9	1.4	0.2
4.9	3.1	1.5	0.1
5.4	3.7	1.5	0.2
4.8	3.4	1.6	0.3

Iris-versicolor			
7.0	3.2	4.7	1.4
6.4	3.2	4.5	1.5
6.9	3.1	4.9	1.5
5.5	2.3	4.0	1.3
6.5	2.8	4.6	1.5
5.7	2.8	4.5	1.3
6.3	3.3	4.7	1.6
4.9	2.4	3.3	1.0
6.6	2.9	4.6	1.3
5.2	2.7	3.9	1.4
5.0	2.0	3.5	1.0
6.8	3.0	4.3	1.6

BAYES LLOYD

K-MEDIAS

K-medias borroso

Parámetros iniciales

Tolerancia = 0.01 V1 = (4.60, 3.00, 4.00, 0.00)
 Peso exponencial = 2 V2 = (6.80, 3.40, 4.60, 0.70)

EJECUTAR

Centros obtenidos

Iteraciones realizadas: 2

V1 = (5.01, 3.38, 1.54, 0.27)
 V2 = (5.96, 2.81, 4.26, 1.32)

Clasificación

5.1 | 3.5 | 1.4 | 0.2

6.9 | 3.1 | 4.9 | 1.5

5.0 | 3.4 | 1.5 | 0.2

Ilustración 4. Ejemplo K-medias ejecutado

Además, podemos ver que el botón de ejecutar ya no está habilitado, y que ahora podemos clasificar los ejemplos (recuadros en verde en la imagen). Vamos a clasificar entonces, por ejemplo, la primera muestra, que tiene el valor: (5.1, 3.5, 1.4, 0.2). Pulsamos el botón correspondiente y obtenemos:

Datos de entrada

Iris-setosa			
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5.0	3.6	1.4	0.2
5.4	3.9	1.7	0.4
4.6	3.4	1.4	0.3
5.0	3.4	1.5	0.2
4.4	2.9	1.4	0.2
4.9	3.1	1.5	0.1
5.4	3.7	1.5	0.2
4.8	3.4	1.6	0.2

Iris-versicolor			
7.0	3.2	4.7	1.4
6.4	3.2	4.5	1.5
6.9	3.1	4.9	1.5
5.5	2.3	4.0	1.3
6.5	2.8	4.6	1.5
5.7	2.8	4.5	1.3
6.3	3.3	4.7	1.6
4.9	2.4	3.3	1.0
6.6	2.9	4.6	1.3
5.2	2.7	3.9	1.4
5.0	2.0	3.5	1.0
5.8	2.0	4.2	1.5

BAYES

LLOYD

K-MEDIAS

K-medias borroso

Parámetros iniciales

Tolerancia = 0.01 V1 = (4.60, 3.00, 4.40, 0.20)

Peso exponencial = 2 V2 = (6.80, 3.40, 4.90, 0.20)

EJECUTAR

Centros obtenidos

V1 = (5.01, 3.38, 1.54, 0.27)
 V2 = (5.96, 2.81, 4.26, 1.32)

Clasificación

5.1 3.5 1.4 0.2	Iris-setosa
6.9 3.1 4.9 1.5	Iris-versicolor
5.0 3.4 1.5 0.2	Iris-setosa

La muestra ha sido clasificada en Iris-setosa. Si nos fijamos en los datos de entrada, hay una muestra en Iris-setosa que tiene los mismos atributos que el ejemplo que hemos clasificado, por lo que parece que el algoritmo es correcto.

Aun así, para cerciorarnos de la corrección de la clasificación, podemos fijarnos en la zona derecha de la pantalla, donde aparecen los cálculos que se han realizado.

Efectivamente, partimos de la muestra (5.1, 3.5, 1.4, 0.2)

Además, podemos ver cómo calcula la distancia a cada centro sustituyendo por los centros obtenidos (véanse los recuadros verdes de las dos capturas), y comprobar que las cuentas son correctas.

De todas formas, casi a simple vista puede apreciarse que el primer centro es mucho más cercano a la muestra que el segundo.

En cualquier caso, el algoritmo llega a la conclusión de que la muestra pertenece a Iris-setosa, por tener la distancia mínima a su centro.

Cálculo de pertenencia

Xk = (5.10, 3.50, 1.40, 0.20)

Calculamos distancias:

· ||Xk - V1||² =

= ||(5.10, 3.50, 1.40, 0.20) - (5.01, 3.38, 1.54, 0.27)||² =

= 0.047

· ||Xk - V2||² =

= ||(5.10, 3.50, 1.40, 0.20) - (5.96, 2.81, 4.26, 1.32)||² =

= 10.671

-> Menor distancia a V1 (0.047) => Xk pertenece a Iris-setosa

Cálculo grados pertenencia:

- P(V1/Xk) = 0.996

- P(V2/Xk) = 0.004

-> Mayor grado de pertenencia (0.996) a la clase 1

=> Xk pertenece a Iris-setosa

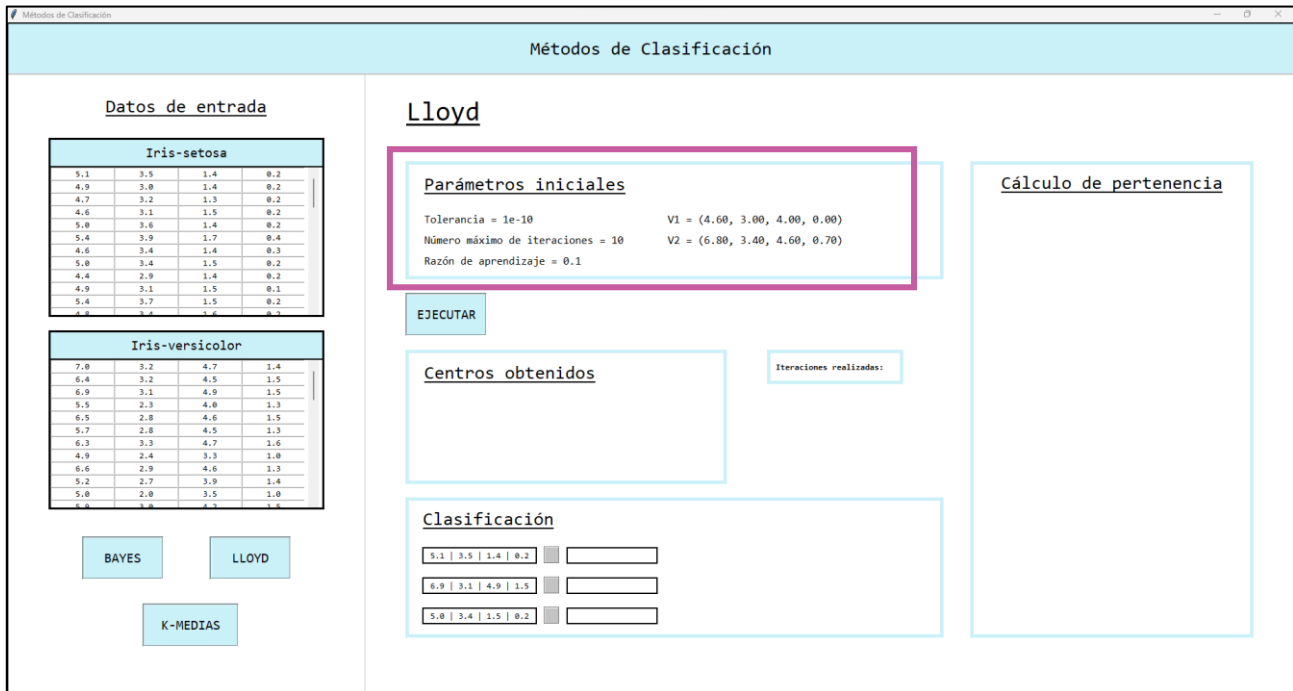
También podemos comprobar el cálculo de los grados de pertenencia, del siguiente modo:

$$\begin{aligned} \Rightarrow P(v_1/x_k) &= \frac{1/d_{1k}^{1/(b-1)}}{\sum_{r=1}^c (1/d_{rk}^{1/(b-1)})} = \frac{1/0.047^{1/(2-1)}}{1/0.047^{1/(2-1)} + 1/10.671^{1/(2-1)}} \approx 0.996 \quad \checkmark \\ \Rightarrow P(v_2/x_k) &= \frac{1/d_{2k}^{1/(b-1)}}{\sum_{r=1}^c (1/d_{rk}^{1/(b-1)})} = \frac{1/10.671^{1/(2-1)}}{1/0.047^{1/(2-1)} + 1/10.671^{1/(2-1)}} \approx 0.004 \quad \checkmark \end{aligned}$$

Efectivamente, los cálculos son correctos, y el grado de pertenencia claramente superior es el de la clase 1.

5.2. Ejemplo Lloyd

El siguiente ejemplo es el análogo con el algoritmo de Lloyd. Si pulsamos el botón correspondiente al algoritmo desde la vista del ejemplo anterior se refrescará la vista y veremos lo siguiente:



Métodos de Clasificación

Datos de entrada

Iris-setosa

5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5.0	3.6	1.4	0.2
5.4	3.9	1.7	0.4
4.6	3.4	1.4	0.3
5.0	3.4	1.5	0.2
4.4	2.9	1.4	0.2
4.9	3.1	1.5	0.1
5.4	3.7	1.5	0.2
4.9	3.4	1.6	0.1

Iris-versicolor

7.0	3.2	4.7	1.4
6.4	3.2	4.5	1.5
6.9	3.1	4.9	1.5
5.5	2.3	4.0	1.3
6.5	2.8	4.6	1.5
5.7	2.8	4.5	1.3
6.3	3.3	4.7	1.6
4.9	2.4	3.3	1.0
6.6	2.9	4.6	1.3
5.2	2.7	3.9	1.4
5.0	2.0	3.5	1.0
6.8	2.8	4.1	1.4

BAYES **LLOYD** **K-MEDIAS**

Lloyd

Parámetros iniciales

Tolerancia = 1e-10 V1 = (4.60, 3.00, 4.00, 0.00)
 Número máximo de iteraciones = 10 V2 = (6.80, 3.40, 4.60, 0.70)
 Razón de aprendizaje = 0.1

EJECUTAR

Centros obtenidos

Iteraciones realizadas:

Clasificación

5.1 | 3.5 | 1.4 | 0.2

6.9 | 3.1 | 4.9 | 1.5

5.0 | 3.4 | 1.5 | 0.2

Cálculo de pertenencia

Ilustración 5. Ejemplo Lloyd inicio

La disposición, como ya sabíamos, es la misma que para el algoritmo de K-medias, exceptuando los parámetros iniciales que toma este algoritmo. Los centros iniciales son los mismos, pero la tolerancia varía, y en lugar del peso exponencial tenemos dos nuevos parámetros: el número máximo de iteraciones y la razón de aprendizaje. Así pues, si ejecutamos el algoritmo, obtenemos:

Métodos de Clasificación

Datos de entrada

Iris-setosa

5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5.0	3.6	1.4	0.2
5.4	3.9	1.7	0.4
4.6	3.4	1.4	0.3
5.0	3.4	1.5	0.2
4.4	2.9	1.4	0.2
4.9	3.1	1.5	0.1
5.4	3.7	1.5	0.2
4.8	3.4	1.6	0.3

Iris-versicolor

7.0	3.2	4.7	1.4
6.4	3.2	4.5	1.5
6.9	3.1	4.9	1.5
5.5	2.3	4.0	1.3
6.5	2.8	4.6	1.5
5.7	2.8	4.5	1.3
6.3	3.3	4.7	1.6
4.9	2.4	3.3	1.0
6.6	2.9	4.6	1.3
5.2	2.7	3.9	1.4
5.0	2.0	3.5	1.0
5.8	2.0	3.2	1.5

BAYES LLOYD K-MEDIAS

Lloyd

Parámetros iniciales

Tolerancia = $1e-10$ V1 = (4.60, 3.00, 4.00, 0.00)
 Número máximo de iteraciones = 10 V2 = (6.80, 3.40, 4.60, 0.70)
 Razón de aprendizaje = 0.1

EJECUTAR

Centros obtenidos

V1 = (4.96, 3.38, 1.48, 0.25)
 V2 = (5.75, 2.75, 4.10, 1.27)

Iteraciones realizadas: 2

Clasificación

5.1 | 3.5 | 1.4 | 0.2
 6.9 | 3.1 | 4.9 | 1.5
 5.0 | 3.4 | 1.5 | 0.2

Cálculo de pertenencia

Ilustración 6. Ejemplo Lloyd ejecutado

Dados esos centros iniciales, el algoritmo de Lloyd sólo ha necesitado dos iteraciones para cumplir el criterio de finalización, y se puede ver que los centros obtenidos son ligeramente diferentes a los que calculó el algoritmo anterior. Clasifiquemos ahora el segundo ejemplar, que tiene el valor (6.9, 3.1, 4.9, 1.5):

Datos de entrada

Iris-setosa

5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5.0	3.6	1.4	0.2
5.4	3.9	1.7	0.4
4.6	3.4	1.4	0.3
5.0	3.4	1.5	0.2
4.4	2.9	1.4	0.2
4.9	3.1	1.5	0.1
5.4	3.7	1.5	0.2
4.8	3.4	1.6	0.3

Iris-versicolor

7.0	3.2	4.7	1.4
6.4	3.2	4.5	1.5
6.9	3.1	4.9	1.5
5.5	2.3	4.0	1.3
6.5	2.8	4.6	1.5
5.7	2.8	4.5	1.3
6.3	3.3	4.7	1.6
4.9	2.4	3.3	1.0
6.6	2.9	4.6	1.3
5.2	2.7	3.9	1.4
5.0	2.0	3.5	1.0
5.8	2.0	3.2	1.5

BAYES LLOYD K-MEDIAS

Lloyd

Parámetros iniciales

Tolerancia = $1e-10$ V1 = (4.60, 3.00, 4.00, 0.00)
 Número máximo de iteraciones = 10 V2 = (6.80, 3.40, 4.60, 0.70)
 Razón de aprendizaje = 0.1

EJECUTAR

Centros obtenidos

V1 = (4.96, 3.38, 1.48, 0.25)
 V2 = (5.75, 2.75, 4.10, 1.27)

Clasificación

5.1 | 3.5 | 1.4 | 0.2
 6.9 | 3.1 | 4.9 | 1.5 Iris-versicolor
 5.0 | 3.4 | 1.5 | 0.2

En este caso, el algoritmo lo clasifica como Iris-versicolor. De hecho, si observamos los datos de entrada, hay una muestra con los mismos parámetros en la tabla de esa clase. Por tanto, de nuevo parece que el algoritmo es correcto.

De todos modos, podemos volver a comprobarlo si miramos en la zona derecha de la pantalla, para ver los cálculos de las distancias que realiza.

Efectivamente, partimos de la muestra (5.1, 3.5, 1.4, 0.2). De nuevo, podemos comprobar que los centros que toma para calcular cada distancia son los que obtuvo el algoritmo (véanse recuadros verdes en las dos capturas).

Y, como los cálculos son correctos, vemos que la muestra está notablemente más cerca de centro de la clase 2, por lo que podemos considerar que pertenece a la clase Iris-versicolor (técnicamente no pertenece, al menos no en su totalidad, pero a efectos prácticos consideramos que una muestra pertenece a la clase de la que tenga el mayor grado de pertenencia).

Cálculo de pertenencia

$X_k = (6.90, 3.10, 4.90, 1.50)$

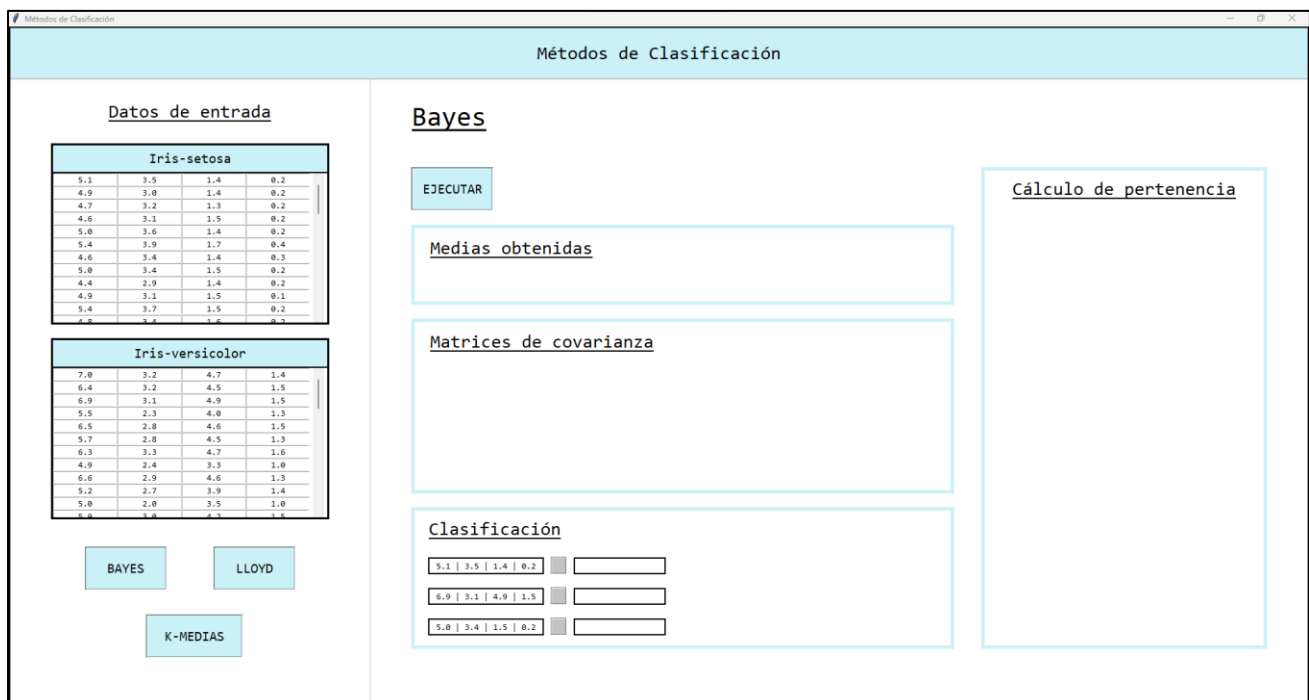
Calculamos distancias:

• $||X_k - V_1||^2 =$
 $= ||(6.90, 3.10, 4.90, 1.50) - (4.96, 3.38, 1.48, 0.25)||^2 =$
 $= 17.104$
 • $||X_k - V_2||^2 =$
 $= ||(6.90, 3.10, 4.90, 1.50) - (5.75, 2.75, 4.10, 1.27)||^2 =$
 $= 2.138$

-> Menor distancia a V2 (2.138) => X_k pertenece a Iris-versicolor

5.3. Ejemplo Bayes

Analicemos entonces al último ejemplo, donde ejecutaremos el algoritmo de Bayes. Pasamos a la vista correspondiente al algoritmo:



The screenshot shows the 'Métodos de Clasificación' window. On the left, under 'Datos de entrada', there are two tables of Iris data. Below them are buttons for 'BAYES', 'LLOYD', and 'K-MEDIAS'. The 'Bayes' section on the right has an 'EJECUTAR' button. Below it are sections for 'Medias obtenidas', 'Matrices de covarianza', and 'Clasificación'. The 'Clasificación' section shows three input rows with their corresponding output boxes. A 'Cálculo de pertenencia' section is also visible on the far right.

	5.1	3.5	1.4	0.2
	4.9	3.0	1.4	0.2
	4.7	3.2	1.3	0.2
	4.6	3.1	1.5	0.2
	5.0	3.6	1.4	0.2
	5.4	3.9	1.7	0.4
	4.6	3.4	1.4	0.3
	5.0	3.4	1.5	0.2
	4.4	2.9	1.4	0.2
	4.9	3.1	1.5	0.1
	5.4	3.7	1.5	0.2
	4.8	3.4	1.6	0.3

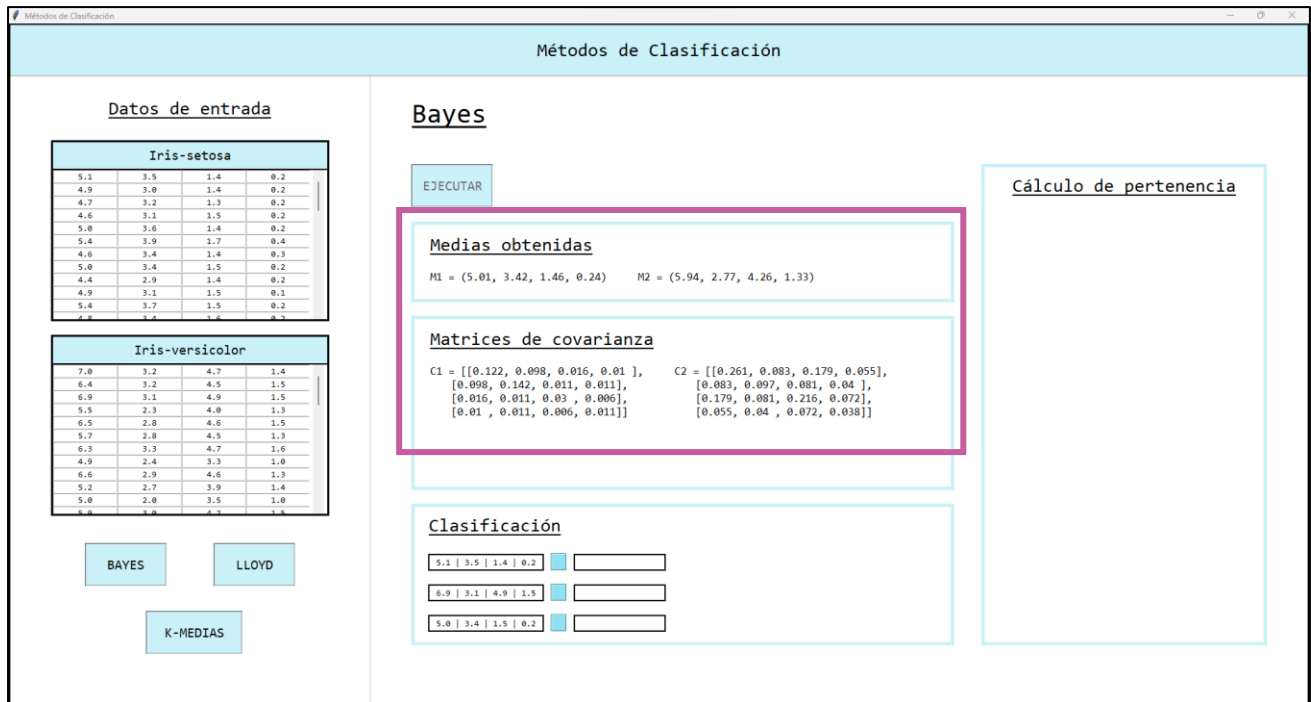
	7.0	3.2	4.7	1.4
	6.4	3.2	4.5	1.5
	6.9	3.1	4.9	1.5
	5.5	2.3	4.0	1.3
	6.5	2.8	4.6	1.5
	5.7	2.8	4.5	1.3
	6.3	3.3	4.7	1.6
	4.9	2.4	3.3	1.0
	6.6	2.9	4.6	1.3
	5.2	2.7	3.9	1.4
	5.0	2.0	3.5	1.0
	6.8	3.0	4.1	1.6

Clasificación

5.1 3.5 1.4 0.2	
6.9 3.1 4.9 1.5	
5.0 3.4 1.5 0.2	

Ilustración 7. Ejemplo Bayes inicio

En este caso, no contamos con parámetros iniciales, por lo que directamente debemos pulsar el botón ejecutar. Con él, obtendremos las medias de cada clase, junto a sus relativas matrices de covarianza. Dado que las muestras están formadas por 4 atributos, la matriz de covarianza tendrá, como ya vimos en su momento, dimensiones 4×4 .



Métodos de Clasificación

Datos de entrada

Iris-setosa

5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5.0	3.6	1.4	0.2
5.4	3.9	1.7	0.4
4.6	3.4	1.4	0.3
5.0	3.4	1.5	0.2
4.4	2.9	1.4	0.2
4.9	3.1	1.5	0.1
5.4	3.7	1.5	0.2
4.8	3.4	1.6	0.3

Iris-versicolor

7.0	3.2	4.7	1.4
6.4	3.2	4.5	1.5
6.9	3.1	4.9	1.5
5.5	2.3	4.0	1.3
6.5	2.8	4.6	1.5
5.7	2.8	4.5	1.3
6.3	3.3	4.7	1.6
4.9	2.4	3.3	1.0
6.6	2.9	4.6	1.3
5.2	2.7	3.9	1.4
5.0	2.0	3.5	1.0
6.8	2.8	4.3	1.5

BAYES **LLOYD**

K-MEDIAS

Bayes

EJECUTAR

Medias obtenidas

M1 = (5.01, 3.42, 1.46, 0.24) M2 = (5.94, 2.77, 4.26, 1.33)

Matrices de covarianza

C1 = [[0.122, 0.098, 0.016, 0.01], [0.098, 0.142, 0.011, 0.011], [0.016, 0.011, 0.03, 0.006], [0.01, 0.011, 0.006, 0.011]] C2 = [[0.261, 0.083, 0.179, 0.055], [0.083, 0.097, 0.081, 0.04], [0.179, 0.081, 0.216, 0.072], [0.055, 0.04, 0.072, 0.038]]

Clasificación

5.1 | 3.5 | 1.4 | 0.2

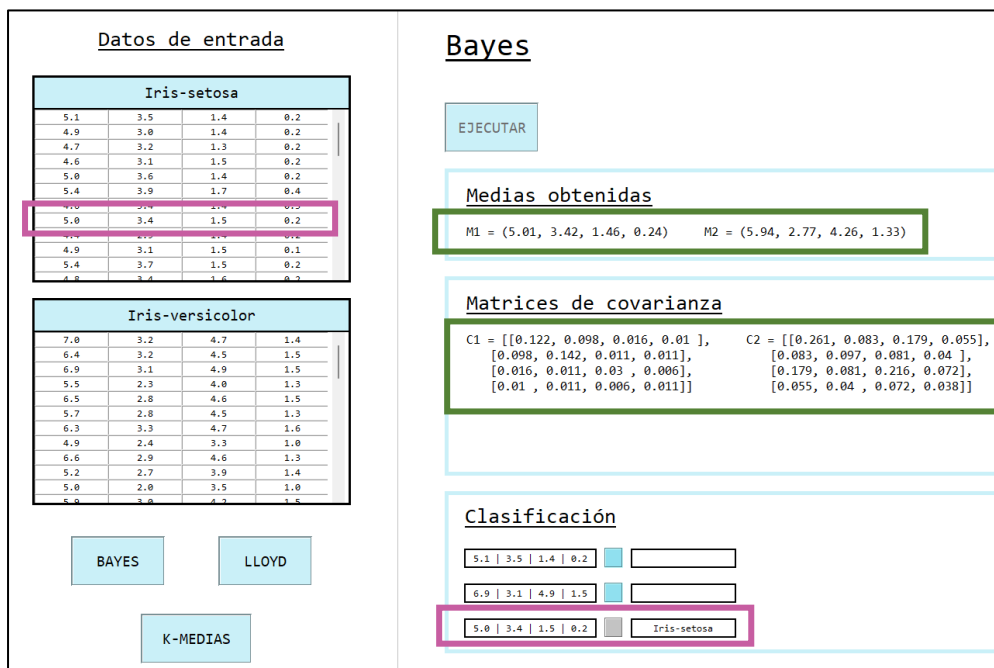
6.9 | 3.1 | 4.9 | 1.5

5.0 | 3.4 | 1.5 | 0.2

Cálculo de pertenencia

Ilustración 8. Ejemplo Bayes ejecutado

De nuevo, las medias calculadas por Bayes varían ligeramente con respecto a los dos anteriores algoritmos. Pasamos entonces a clasificar el tercer ejemplo, con valor (5.0, 3.4, 1.5, 0.2):



Datos de entrada

Iris-setosa

5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5.0	3.6	1.4	0.2
5.4	3.9	1.7	0.4
4.6	3.4	1.4	0.3
5.0	3.4	1.5	0.2
4.4	2.9	1.4	0.2
4.9	3.1	1.5	0.1
5.4	3.7	1.5	0.2
4.8	3.4	1.6	0.3

Iris-versicolor

7.0	3.2	4.7	1.4
6.4	3.2	4.5	1.5
6.9	3.1	4.9	1.5
5.5	2.3	4.0	1.3
6.5	2.8	4.6	1.5
5.7	2.8	4.5	1.3
6.3	3.3	4.7	1.6
4.9	2.4	3.3	1.0
6.6	2.9	4.6	1.3
5.2	2.7	3.9	1.4
5.0	2.0	3.5	1.0
6.8	2.8	4.3	1.5

BAYES **LLOYD**

K-MEDIAS

Bayes

EJECUTAR

Medias obtenidas

M1 = (5.01, 3.42, 1.46, 0.24) M2 = (5.94, 2.77, 4.26, 1.33)

Matrices de covarianza

C1 = [[0.122, 0.098, 0.016, 0.01], [0.098, 0.142, 0.011, 0.011], [0.016, 0.011, 0.03, 0.006], [0.01, 0.011, 0.006, 0.011]] C2 = [[0.261, 0.083, 0.179, 0.055], [0.083, 0.097, 0.081, 0.04], [0.179, 0.081, 0.216, 0.072], [0.055, 0.04, 0.072, 0.038]]

Clasificación

5.1 | 3.5 | 1.4 | 0.2

6.9 | 3.1 | 4.9 | 1.5

5.0 | 3.4 | 1.5 | 0.2

Este último ejemplar ha sido clasificado como Iris-setosa. Una vez más, si observamos las muestras de la tabla de esa clase, encontramos una muestra con los mismos valores que el ejemplar, lo que nos da a entender que el algoritmo es, de hecho, correcto.

Para ver más información, pasamos a la zona de la derecha,

donde vemos los cálculos con mayor profundidad. Podemos ver, en los recuadros verdes, que tanto las medias como las matrices que se están considerando coinciden, así como la muestra que se va a clasificar.

Hagamos entonces los cálculos para comprobar que los resultados cuadran, empezando por las restas ($x_k - m_i$):

Cálculo de pertenencia

$X_k = (5.00, 3.40, 1.50, 0.20)$

Calculamos distancias:

1. Siendo

$M1 = (5.01, 3.42, 1.46, 0.24)$

$C1 = \begin{bmatrix} 0.122, 0.098, 0.016, 0.01 \\ 0.098, 0.142, 0.011, 0.011 \\ 0.016, 0.011, 0.03, 0.006 \\ 0.01, 0.011, 0.006, 0.011 \end{bmatrix}$

$X_k - M1 = (-0.01, -0.02, 0.04, -0.04)$

$Dm(X_k, M1/C1) = (X_k - M1) * I * (X_k - M1)^t = 0.298$

2. Siendo

$M2 = (5.94, 2.77, 4.26, 1.33)$

$C2 = \begin{bmatrix} 0.261, 0.083, 0.179, 0.055 \\ 0.083, 0.097, 0.081, 0.04 \\ 0.179, 0.081, 0.216, 0.072 \\ 0.055, 0.04, 0.072, 0.038 \end{bmatrix}$

$X_k - M2 = (-0.94, 0.63, -2.76, -1.13)$

$Dm(X_k, M2/C2) = (X_k - M2) * I * (X_k - M2)^t = 105.934$

-> Menor distancia a V1 (0.298) => X_k pertenece a Iris-setosa

$$\begin{aligned} \Rightarrow x_k - m_1 &= (5.0, 3.4, 1.5, 0.2) - \\ &= (5.01, 3.42, 1.46, 0.24) = \end{aligned}$$

$$(-0.01, -0.02, 0.04, -0.04) \quad \checkmark$$

$$\begin{aligned} \Rightarrow x_k - m_2 &= (5.0, 3.4, 1.5, 0.2) - \\ &= (5.94, 2.77, 4.26, 1.33) = \end{aligned}$$

$$(-0.94, 0.63, -2.76, -1.13) \quad \checkmark$$

Para los cálculos de las distancias, debemos multiplicar las matrices obtenidas por la inversa de las matrices de covarianza, y por estas matrices traspuestas.

Empezamos calculando la matriz inversa de las c_i :

Para la clase 1:

$$\begin{pmatrix} 0.122 & 0.098 & 0.016 & 0.01 \\ 0.098 & 0.142 & 0.011 & 0.011 \\ 0.016 & 0.011 & 0.03 & 0.006 \\ 0.01 & 0.011 & 0.006 & 0.011 \end{pmatrix}^{(-1)} = \begin{pmatrix} 19,287 & -12,757 & -5,223 & -1,928 \\ -12,757 & 16,126 & 2,017 & -5,629 \\ -5,223 & 2,017 & 39,099 & -18,595 \\ -1,928 & -5,629 & -18,595 & 108,433 \end{pmatrix}$$

Y para la clase 2:

$$\begin{pmatrix} 0.261 & 0.083 & 0.179 & 0.055 \\ 0.083 & 0.097 & 0.081 & 0.04 \\ 0.179 & 0.081 & 0.216 & 0.072 \\ 0.055 & 0.04 & 0.072 & 0.038 \end{pmatrix}^{(-1)} = \begin{pmatrix} 9,651 & -3,575 & -8,836 & 6,536 \\ -3,575 & 19,655 & 2,073 & -19,443 \\ -8,836 & 2,073 & 20,734 & -28,679 \\ 6,536 & -19,443 & -28,679 & 91,661 \end{pmatrix}$$

Ahora, si hacemos las multiplicaciones correspondientes, vemos como obtenemos (aproximadamente, hay desajustes por decimales) los resultados esperados:

$$\begin{aligned} \Rightarrow & \begin{pmatrix} -0.01 & -0.02 & 0.04 & -0.04 \end{pmatrix} \cdot \begin{pmatrix} \frac{19119500}{991313} & \frac{-12646000}{991313} & \frac{-5178000}{991313} & \frac{-1911000}{991313} \\ \frac{-12646000}{991313} & \frac{15986000}{991313} & \frac{1999000}{991313} & \frac{-5580000}{991313} \\ \frac{19119500}{991313} & \frac{15986000}{991313} & \frac{1999000}{991313} & \frac{-5580000}{991313} \\ \frac{-5178000}{991313} & \frac{1999000}{991313} & \frac{38759000}{991313} & \frac{-18433000}{991313} \\ \frac{19119500}{991313} & \frac{15986000}{991313} & \frac{1999000}{991313} & \frac{-5580000}{991313} \\ \frac{-5178000}{991313} & \frac{1999000}{991313} & \frac{38759000}{991313} & \frac{-18433000}{991313} \\ \frac{19119500}{991313} & \frac{15986000}{991313} & \frac{1999000}{991313} & \frac{-5580000}{991313} \\ \frac{-5178000}{991313} & \frac{1999000}{991313} & \frac{38759000}{991313} & \frac{-18433000}{991313} \end{pmatrix} = \begin{pmatrix} -0.070 & 0.111 & 2.320 & -4.949 \end{pmatrix} \\ & \begin{pmatrix} \frac{-68955}{991313} & \frac{109900}{991313} & \frac{2299480}{991313} & \frac{-4906250}{991313} \end{pmatrix} \cdot \begin{pmatrix} \frac{-1}{100} \\ \frac{-1}{50} \\ \frac{1}{25} \\ \frac{-1}{25} \end{pmatrix} = \begin{pmatrix} 0.289 \end{pmatrix} \approx 0.298 \quad \checkmark \end{aligned}$$

$$\begin{pmatrix} -0,94 & 0,63 & -2,76 & -1,13 \end{pmatrix} \cdot \begin{pmatrix} \frac{32994000}{3418861} & \frac{-12222000}{3418861} & \frac{-30208000}{3418861} & \frac{22347000}{3418861} \\ \frac{-12222000}{3418861} & \frac{67197200}{3418861} & \frac{7086600}{3418861} & \frac{-66471400}{3418861} \\ \frac{3418861}{3418861} & \frac{3418861}{3418861} & \frac{3418861}{3418861} & \frac{3418861}{3418861} \\ \frac{-30208000}{3418861} & \frac{7086600}{3418861} & \frac{70887800}{3418861} & \frac{-98051200}{3418861} \\ \frac{22347000}{3418861} & \frac{-66471400}{3418861} & \frac{-98051200}{3418861} & \frac{313376800}{3418861} \end{pmatrix} = \begin{pmatrix} 5,677 & 31,992 & -15,208 & -42,815 \end{pmatrix}$$

$$\begin{pmatrix} \frac{19407750}{3418861} & \frac{109376582}{3418861} & \frac{-51992394}{3418861} & \frac{-146377634}{3418861} \end{pmatrix} \cdot \begin{pmatrix} \frac{-47}{50} \\ \frac{63}{100} \\ \frac{-69}{25} \\ \frac{-113}{100} \end{pmatrix} = \begin{pmatrix} 105,172 \end{pmatrix} \approx 105.934 \checkmark$$

Así pues, vemos que, claramente, la muestra es más cercana a la clase 1, por lo que se clasifica como Iris-setosa. Concluye así la demostración del funcionamiento del algoritmo.