Instituto Tecnológico de Costa Rica

Ingeniería en Computación

IC1803 - Taller programación

Tarea Programada 2: Clasificación No Supervisada

Profesor: Saúl Calderón Ramírez

Estudiantes:

Franco Vinicio Rojas Lagos 2022437823

Dayana Xie Li 2022097967

Abel Gutiérrez Martínez 2022437323

Primer semestre 2022

Contents

1 Introducción

En el presente trabajo práctico se enfrenta el problema de realizar el amontonamiento automático de un conjunto de datos, esto hace referencia a clasificar datos en diferentes clases. En la vida cotidiana es común que las personas se enfrenten a estos problemas, ya sea de manera consciente o inconsciente, la mente relaciona las características similares que tienen diferentes elementos y los clasifica como algo diferente de los otros, que dependiendo de nuestros conocimientos y experiencias, se catalogan bajo un nombre de un conjunto conocido previamente. Por lo que el propósito de este trabajo práctico es implementar un algoritmo de clasificación no supervisado, específicamente el algoritmo popularmente conocido como "KMeans" o su nombre en español "KMedias" . El algoritmo consiste en un procedimiento simple de clasificación de un conjunto de objetos en un determinado número K de clusters, K determinado a priori [1]. Cada cluster por tanto es caracterizado por su centro o centroide que se encuentra en el centro o el medio de los elementos que componen el cluster [1]. Para clasificar la pertenencia de cada dato con su respectivo clase se usa la distancia euclidiana, el cual es la distancia ordinaria entre dos puntos, calculada a través del teorema de Pitágoras [2]. Se realizará el algoritmo por la cantidad de iteraciones especificadas por el usuario y el sistema proporcionará por medio de la interfaz gráfica el proceso de agrupamiento y finalmente, se mostrará el resultado final.

2 Análisis

En la figura 1 se describe el caso de uso que tiene el sistema a la hora de clasificar datos, desde las entradas esperadas de parte del usuario que son: la cantidad de datos por clase, los valores del primer y segundo centroide, los valores del primer y segunda desviación estándar y la cantidad de iteraciones. Y como salida de parte del sistema, debe de mostrar una ventana con el gráfico de la respuesta final de la clasificación. Para cada clase o método específico del programa se pueden observar sus atributos y entradas esperadas en la figura 2.

Subproblemas

1. Verificar que las entradas de usuario sean de tipo int.
2. Generar los datos.
3. Calcular el mínimo y el máximo de los datos.
4. Crear los centroides.
5. Calcular por medio de la distancia euclidiana la distancia de los datos con los centroides.
6. Determinar el dato más cercano al centroide.

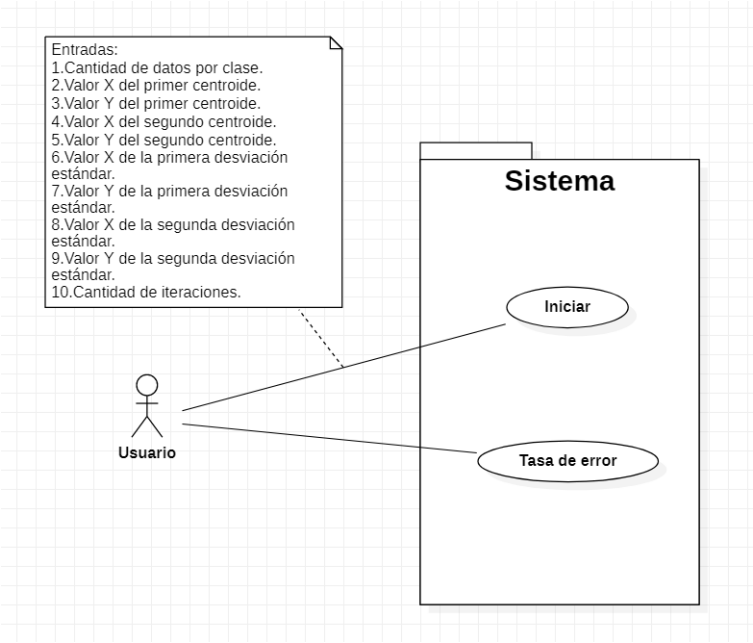

Figure 1: Diagrama de casos de uso

7. Actualizar la matriz de pesos.
8. Actualizar los centroides, con el promedio.
9. Crear la interfaz gráfica.
10. Mostrar al usuario el cambio de los centroides.
11. Mostrar al usuario el resultado final.
12. Colorear los datos pertenecientes a cada clase con un respectivo color en el resultado final.
13. Calcular la tasa de aciertos y errores.

3 Diseño

En esta sección, se detallarán todas las librerías utilizadas en el trabajo práctico y se mostrará el diseño del sistema implementado en el trabajo práctico. La figura 2 se presenta el diseño del sistema por medio del diagrama de clases del sistema de clasificación de datos, en el cual se puede observar que en cada archivo .py está compuesta por diferentes clases que se utiliza en el sistema de filtrado, divididos en tres archivos importantes, se hace mención al Generador_Datos,

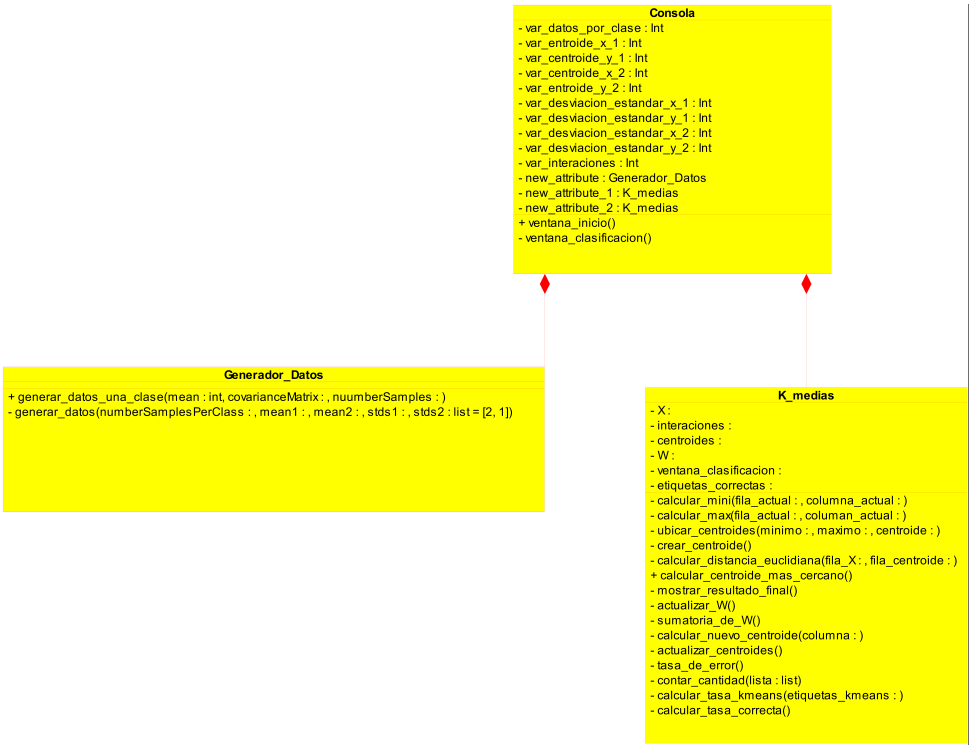

Figure 2: Diagrama de clases

K-Medias, Interfaz_Grafica, cada uno con sus respectivos atributos y entradas esperadas.

Pseudocódigo de los métodos implementados más complejos:

Método: calcular_centroide_mas_cercano(self)

1. Se imprimen los centroides iniciales.
2. Se muestra la gráfica de los datos y centroides iniciales con la librería matplotlib.
3. Se inicializa la variable distancia_euclidiana_min en 9999, una cifra alta de referencia.
4. Se inicializa la variable centroide_mas_cercano en -1, donde se guardará la posición del centroide.
5. Se inicializa la variable iteracion_actual en 1.

6. Mientras la variable iteración actual sea menor o igual que la cantidad de iteraciones especificada por el usuario.

7. Por cada valor en el rango de 0 y las dimensiones de los datos.

   (a) Por cada valor en el rango de 0 y las dimensiones de los centroides.

      i. Se calcula la distancia euclidiana entre el dato actual y el centroide actual, haciendo llamado al método__calcular_distancia_euclidiana.

         A. Si la distancia euclidiana es menor que la distancia_euclidiana_min: se actualiza el valor de la variable distancia_euclidiana_min con la distancia euclidian, se guarda la posición del dato más cercano al centroide, se guarda la posición del centroide más cercano al dato y se guarda el centroide y el dato más cercano.

         B. Luego de revisar cada centroide, se actualiza la matriz W con la posición del dato más cercano al centroide y la posición del centroide más cercano al dato. Esto porque la posición del dato indicará cuál fila de la matriz W se va a modificar y la posición del centroide indicará la columna.

         C. Se reinicia distancia_euclidiana_min con 9999, para usarlo de referencia.

      ii. Luego de haber revisado cada dato se actualizan los centroides.

      iii. Se imprime el número de iteración que se encuentra el programa y se le informa al usuario los valores de los nuevos centroides.

      iv. Se muestra la gráfica de los datos y centroides actualizados con la librería matplotlib.

      v. Se suma uno a la iteración actual, para continuar.

   (b) Luego de que ya termine de realizar todas las iteraciones, se llama al método __mostrar_resultado_final.

Método: __mostrar_resultado_final(self)

1. Se inicializa la variable distancia_inicial con 9999, de referencia.

2. Se inicializa las variables clase_1 y clase_2 con listas vacías. En donde se van a guardar cada dato perteneciente a cada clase.

3. Por cada dato en la lista de datos X.

   (a) Por cada centroide en el rango de 0 y las dimensiones de la lista de centroides.

      i. Se calcula la distancia entre el dato y el centroide.

      ii. Si la distancia es menor que la distancia inicial.

         A. Se guarda el dato y centroide cercano.

B. Se actualiza la distancia_inicial con la distancia recién encontrada.

(b) Se reinicia el valor de la variable distancia_inicial con 9999, para seguir revisando el siguiente dato con los centroides.

(c) Si el centroide cercano es cero, es decir de la primera clase.

i. Se le suma a la lista clase_1 el dato cercano.

(d) Si el centroide más cercano no es cero, es decir es uno.

i. Se le suma a la lista clase_2 el dato cercano.

4. Se convierten las listas de las clases a numpy arrays.
5. Se crea la figura en donde se mostrará el resultado en la ventana.
6. Se grafica cada los datos de cada clase con un color diferente, para marcar la diferencia de clases. Mostrándolo en la ventana de la interfaz.
7. Se crea el botón en la ventana, para realizar el cálculo de la tasa de aciertos.

Método: __calcular_nuevo_centroide(self, columna)

1. Se calcula la cantidad de datos que pertenecen al centroide que se quiere actualizar.
2. Se inicializa la variable resultado en cero.
3. Se inicializa la variable fila_actual en cero.
4. Por cada fila de la matriz W.
   (a) Se multiplica el dato por su indicador de pertenencia indicada en la matriz de pesos W. Guardándolo en la variable resultado.
   (b) Se suma uno a la fila actual.
5. El resultado final se divide por la cantidad de datos que pertenecen al centroide a actualizar.
6. Se retorna el resultado.

Método: __tasa_de_error(self)

1. Se imprime un enunciado con el título "Comparar con datos artificiales", para marcar la separación de este apartado.
2. Se llama al método Kmeans.
3. Se obtienen las etiquetas correctas del kmeans. Se llama al método __calcular_tasa_kmeans con las etiquetas, para que devuelva una lista de las etiquetas de clase 0 y otra lista de las etiquetas de la clase 1.

4. Se llama al método __calcular_tasa_kmeans, para que devuelva una lista de las etiquetas de clase 0 correctas y otra lista de las etiquetas de la clase 1 correctas. Ambas generadas artificialmente.

5. Se calcula la cantidad de etiquetas incorrectas de la clase 0, restando la cantidad de etiquetas de la clase 0 con las correctas artificiales. Y a esto se le saca el valor absoluto en caso de que dé negativo.

6. Se calcula la cantidad de etiquetas correctas de la clase 0, restando la cantidad de etiquetas correctas artificiales con las incorrectas. Sacando valor absoluto del resultado de esta operación, en caso de que dé negativo.

7. Se calcula la cantidad de etiquetas incorrectas de la clase 1, restando la cantidad de etiquetas de la clase 1 con las correctas artificiales. Y a esto se le saca el valor absoluto en caso de que dé negativo.

8. Se calcula la cantidad de etiquetas correctas hay de la clase 1, restando la cantidad de etiquetas correctas artificiales con las incorrectas. Sacando valor absoluto del resultado de esta operación, en caso de que dé negativo.

9. Se calcula la tasa de aciertos, calculando la cantidad de etiquetas correctas de la clase cero con las de la clase uno, el resultado de la suma dividiéndolo por la suma de la cantidad de etiquetas correctas artificiales de la clase 0 y 1. El resultado de esto multiplicado por cien, para que el resultado sea un porcentaje.

10. Se calcula la tasa de aciertos restando 100 con la tasa de aciertos.

11. Se crean los enunciados a mostrar en la ventana de las tasas.

12. Se imprimen los resultados de la tasa de error y aciertos para ser mostrados en el programa.

4 Implementación y pruebas

4.1 Librerías utilizadas

- NumPy: Se utiliza la librería para el manejo de matrices, principalmente a la hora de generar la matriz de pesos W y almacenar los centroides en matrices.

- Random: Se utiliza la función randrange para crear los valores aleatorios de los centroides bajo un rango de los mínimos y máximos de los datos.

- Math: Se utiliza la función “sqrt” de la librería para calcular la distancia euclidiana.

- Matplotlib.pyplot: Se utiliza para graficar los datos y los centroides para poder mostrarlos.

- Sklearn.cluster: Se utiliza la función Kmeans para comparar el resultado de las etiquetas con las correctas, para el cálculo de la tasa de error.
- Torch; Se utiliza para crear los targets de los datos.
- Torch.distributions: Se utiliza la función "multivariate_normal" para generar datos con la distribución gaussiana.
- Tkinter: Librería utilizada para implementar la interfaz gráfica para pedir las entradas y mostrar el resultado final junto a la tasa de aciertos y errores.
- Matplotlib.figure: Librería utilizada para crear el tamaño de la figura del gráfico a implementar en la interfaz.
- Matplotlib.backends.backend_tkagg Librería utilizada para colocar la figura de la gráfica en la ventana de tkinter.

4.2 Prueba 1

1. El sistema abre una ventana, en donde le solicita al usuario ingresar la cantidad de datos por clase, los valores del primer y segundo centroide, los valores del primer y segunda desviación estándar y la cantidad de iteraciones.
   (a) El usuario ingresa, Cantidad de datos por clase: 200, Valor X del primer centroide: 2, Valor Y del primer centroide: 3, Valor X del segundo centroide: 27, Valor Y del segundo centroide: 27, Valor X de la primera desviación estándar: 3, Valor Y de la primera desviación estándar: 3, Valor X de la segunda desviación estándar: 4, Valor Y de la segunda desviación estándar: 3 y Cantidad de iteraciones: 50.
   (b) El usuario da click al botón "Iniciar".
2. El sistema abre una nueva ventana mostrando la gráfica resultado final de la clasificación, a la vez que imprime en el programa la actualización de los centroides por cada iteración. En la ventana se muestra un botón llamado "Tasa de error".
   (a) El usuario da click al botón "Tasa de error".
3. El sistema muestra en la ventana la tasa de aciertos y de errores.

Resultado esperado de la prueba: Se espera ver que la tasa de aciertos resultante de la clasificación sea del 100 % la mayoría de veces, como es mostrado en la figura 3.

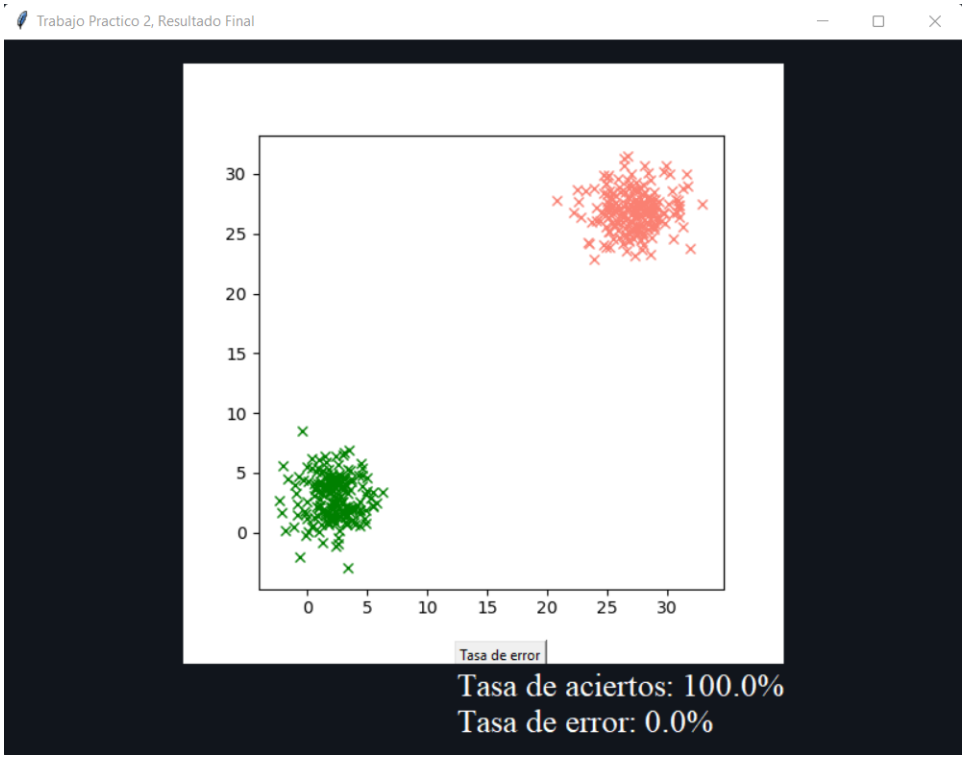

Figura 3: Resultado esperado de la prueba 1

4.3 Prueba 2

1. El sistema abre una ventana, en donde le solicita al usuario ingresar la cantidad de datos por clase, los valores del primer y segundo centroide, los valores del primer y segunda desviación estándar y la cantidad de iteraciones.
   (a) El usuario ingresa, Cantidad de datos por clase: 200, Valor X del primer centroide: 2, Valor Y del primer centroide: 3, Valor X del segundo centroide: 10, Valor Y del segundo centroide: 10, Valor X de la primera desviación estándar: 3, Valor Y de la primera desviación estándar: 3, Valor X de la segunda desviación estándar: 4, Valor Y de la segunda desviación estándar: 3 y Cantidad de iteraciones: 50.
   (b) El usuario da click al botón "Iniciar".
2. El sistema abre una nueva ventana mostrando la gráfica resultado final de la clasificación, a la vez que imprime en el programa la actualización de los centroides por cada iteración. En la ventana se muestra un botón llamado "Tasa de error".
   (a) El usuario da click al botón "Tasa de error".
3. El sistema muestra en la ventana la tasa de aciertos y de errores.

Resultado esperado de la prueba: Se espera ver que la tasa de aciertos resultante de la clasificación sea cerca del 100 % la mayoría de veces, como es mostrado en la figura 4.

4.4 Prueba 3

1. El sistema abre una ventana, en donde le solicita al usuario ingresar la cantidad de datos por clase, los valores del primer y segundo centroide, los valores del primer y segunda desviación estándar y la cantidad de iteraciones.
   (a) El usuario ingresa, Cantidad de datos por clase: 200, Valor X del primer centroide: 2, Valor Y del primer centroide: 3, Valor X del segundo centroide: 6, Valor Y del segundo centroide: 6, Valor X de la primera desviación estándar: 3, Valor Y de la primera desviación estándar: 3, Valor X de la segunda desviación estándar: 4, Valor Y de la segunda desviación estándar: 3 y Cantidad de iteraciones: 50.
   (b) El usuario da click al botón "Iniciar".
2. El sistema abre una nueva ventana mostrando la gráfica resultado final de la clasificación, a la vez que imprime en el programa la actualización de los centroides por cada iteración. En la ventana se muestra un botón llamado "Tasa de error".
   (a) El usuario da click al botón "Tasa de error".

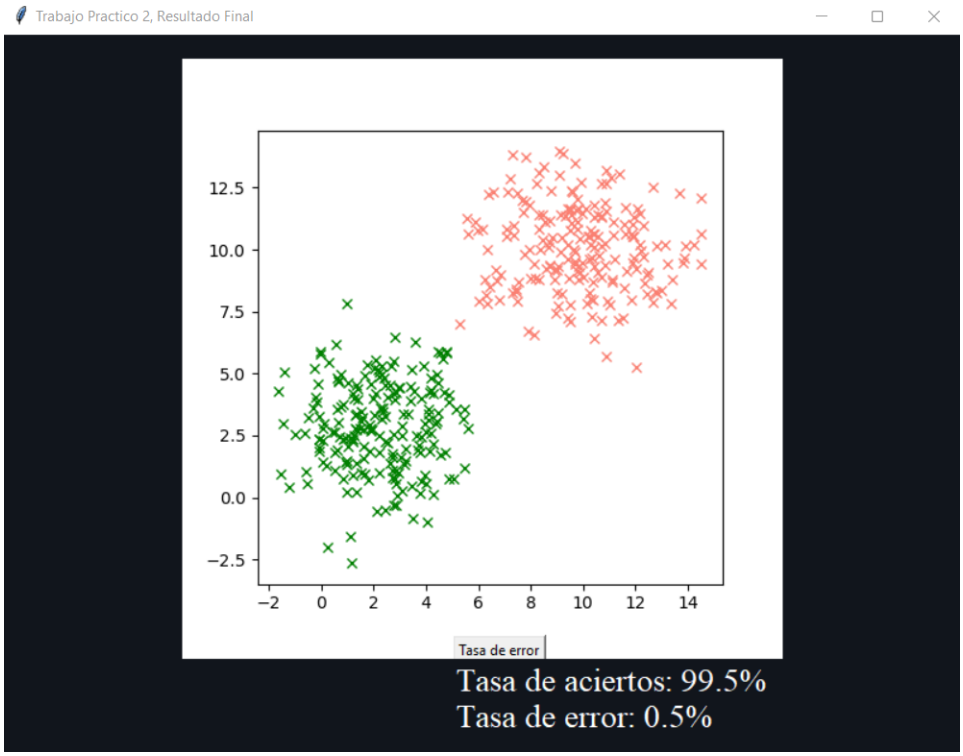

Figura 4: Resultado esperado de la prueba 2

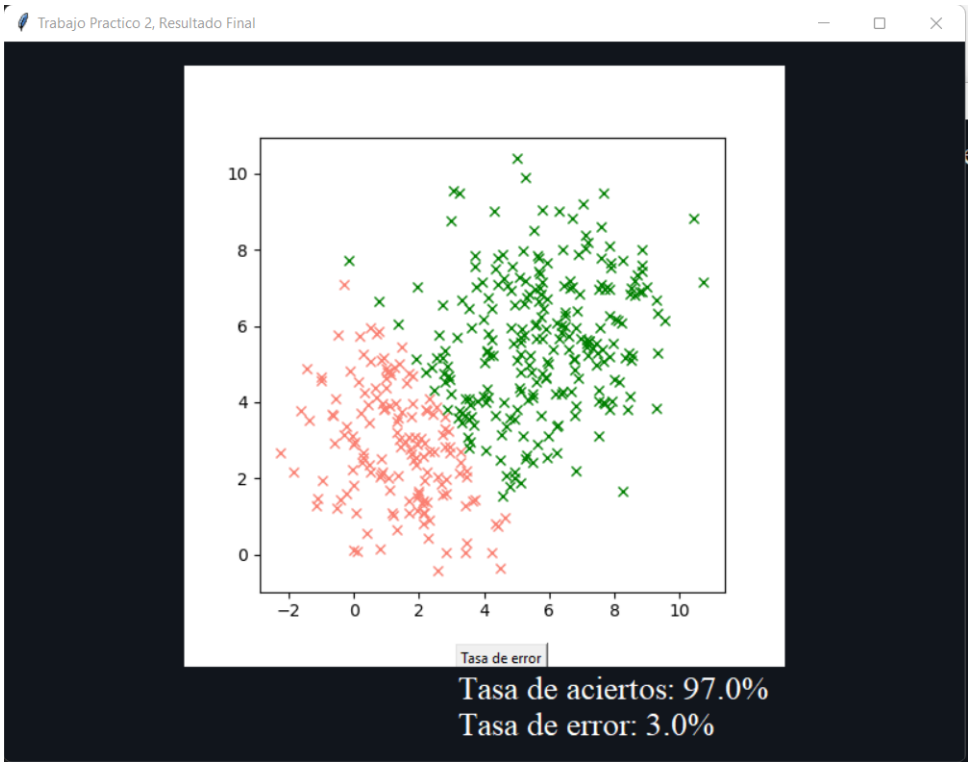

Figura 5: Resultado esperado de la prueba 3

3. El sistema muestra en la ventana la tasa de aciertos y de errores.

Resultado esperado de la prueba: Se espera ver que la tasa de aciertos resultante de la clasificación sea un poco lejana del 100 % pero no tanto, como es mostrado en la figura 5.

4.5 Manual de usuario

En la figura 6 se puede observar que al iniciar el sistema de clasificación, se muestra una pantalla de inicio en donde se le pide al usuario las entradas que debe de especificar para el proceso de clasificación. Luego de ingresarlas como se muestra en la figura 7, al presionar el botón de "Iniciar" , el sistema crea otra ventana en donde muestra la gráfica del resultado final de la clasificación, que se puede observar en la figura 8. A su vez se puede observar en el programa que el sistema muestra la actualización de los centroides por cada iteración en la figura 9 y el resultado del kmeans en la figura 10. Finalmente, se puede observar que en la ventana del resultado existe un botón llamado "Tasa de error", al presionar este la ventana mostrará el porcentaje de aciertos y errores de las etiquetas del kmeans respecto a los correctos, como se puede observar en la figura 11.

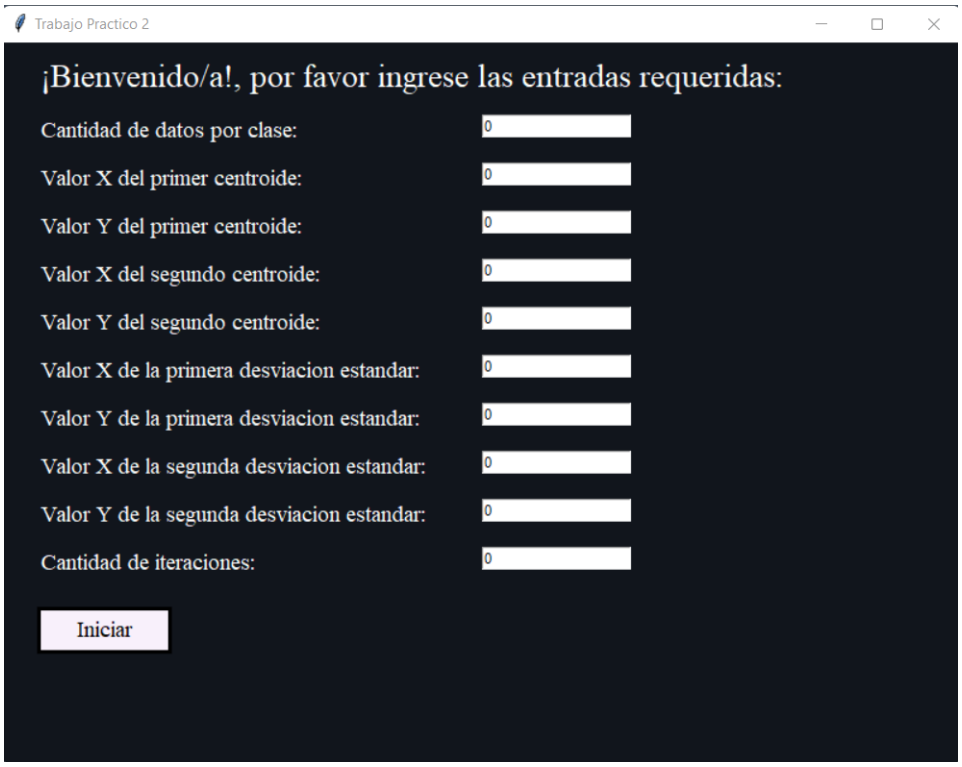

Figure 6: Ventana de inicio

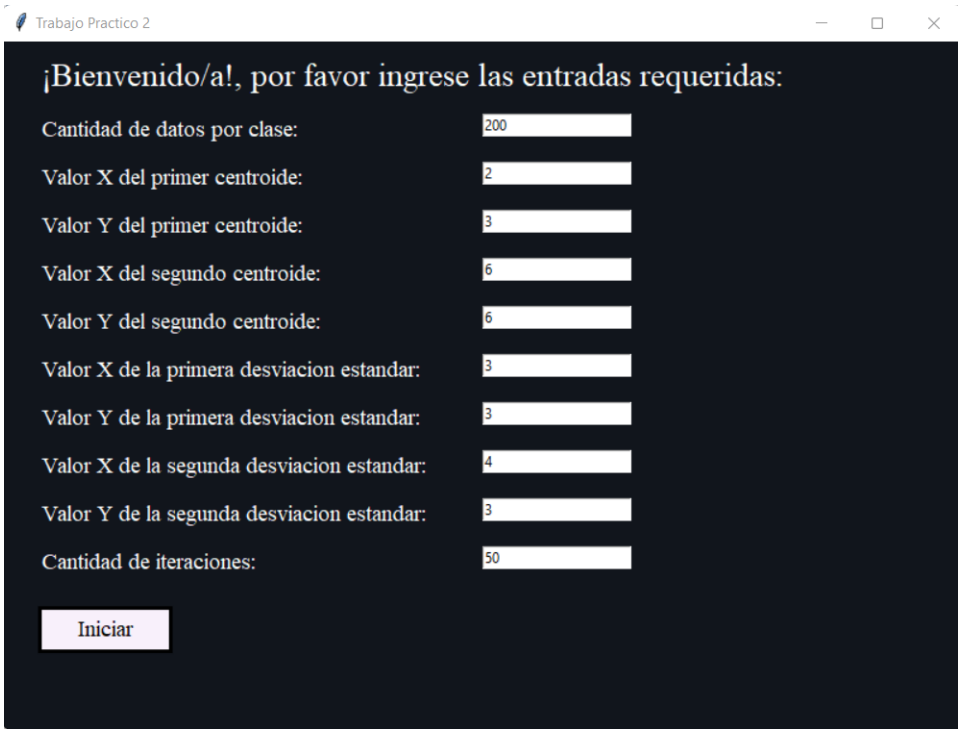

Figure 7: Se ingresan las entradas

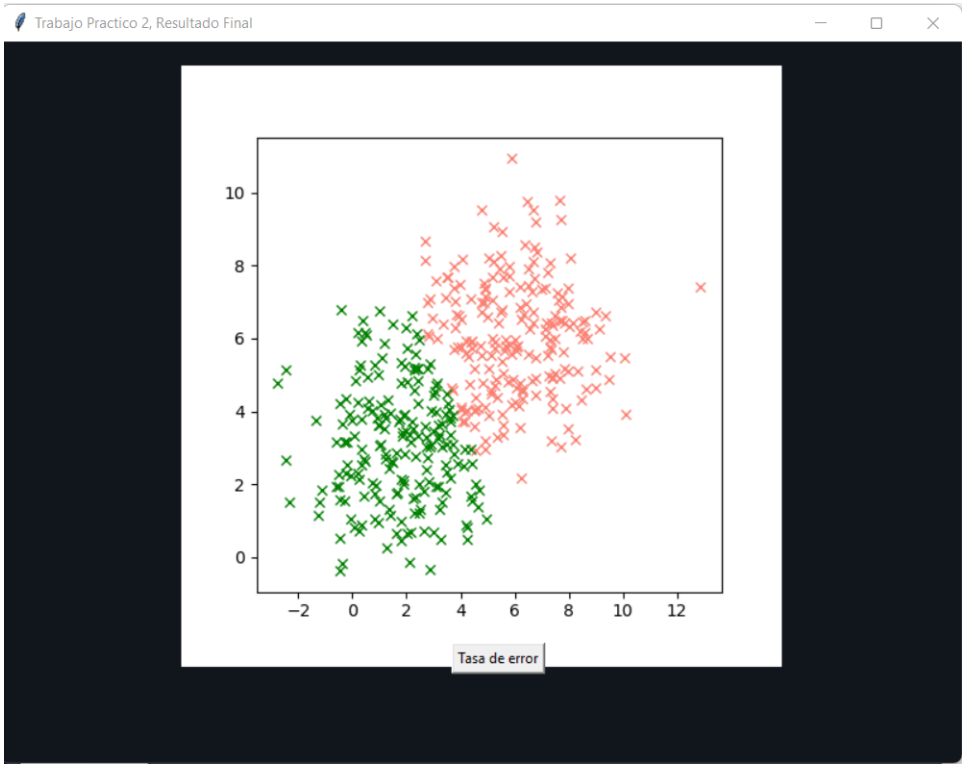

Figure 8: Nueva ventana con el resultado final

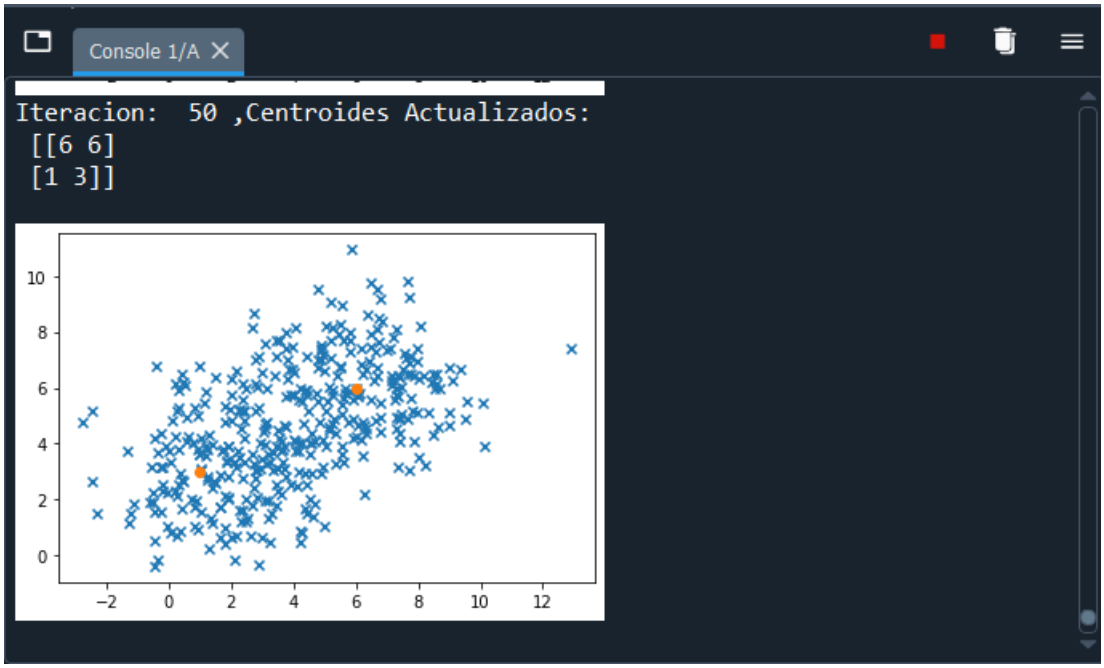

Figure 9: Resultado en el programa

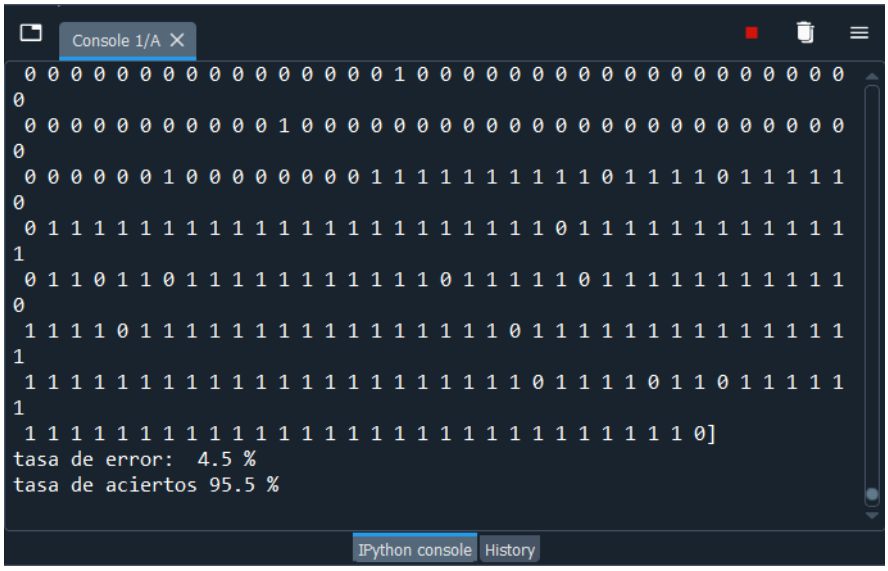

Figure 10: Resultado del programa 2

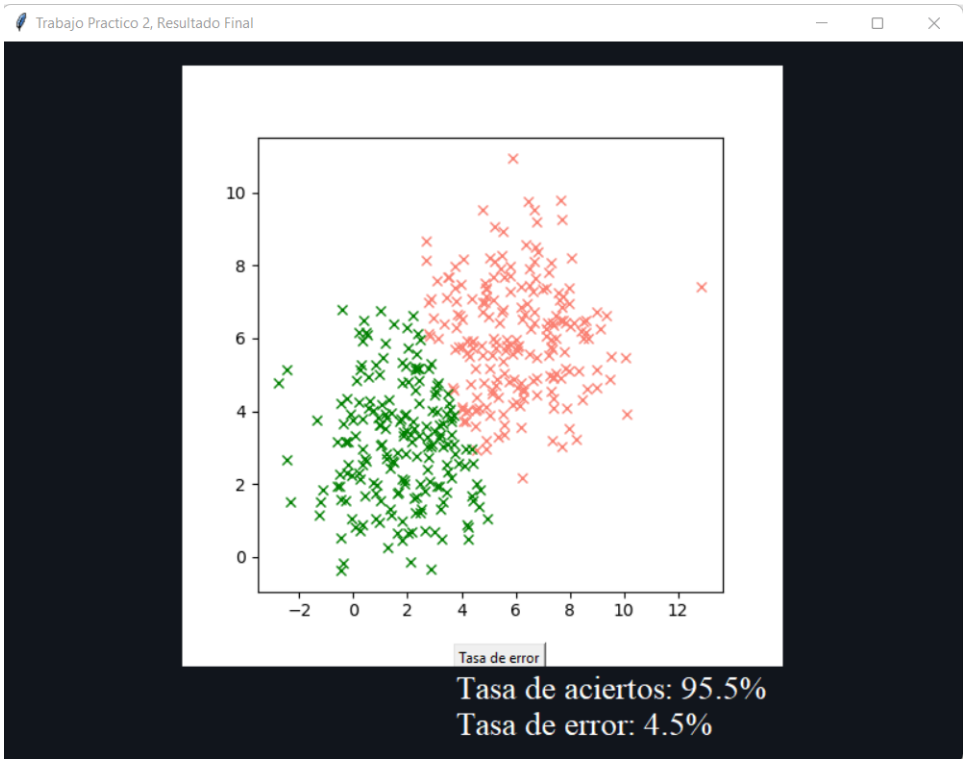

Figure 11: Luego de haber presionado el botón de “Tasa de error”

5 Conclusiones

En conclusión, este trabajo práctico fue muy interesante, se desarrolló un algoritmo de clasificación no supervisada que en este caso fue K Medias. A pesar de existir librerías especializadas para ejecutar este algoritmo, se nos pidió hacerlo de manera manual propiciando el uso de nuestras habilidades y fundamentos de programación con la finalidad de lograr realizar el algoritmo. Por otro lado, se aplicaron las pruebas pedidas por el profesor llegando a tener resultados bastante aceptables. Claramente, se pueden hacer mejoras para optimizar el algoritmo y pueda ser mucho mejor.

6 Bibliografía

References

[1] Cristina García Cambronero and Irene Gómez Moreno. Algoritmos de aprendizaje: knn & kmeans. *Intelgencia en Redes de Comunicación, Universidad Carlos III de Madrid*, 23, 2006.

[2] Marisol Correa Henao. Análisis de clúster automático. 2021.