Programación Técnica y Científica 2023-2024

Grado en Ingeniería Informática

Universidad de Granada

Práctica Tkinter Robótica

Miguel García López

Índice

1. Introducción

Es muy común en el mundo de la Inteligencia Artificial, Ciencia de Datos y Análisis de Datos, que se den procesos en los cuales se deban recoger datos, ya sea a través de un sensor, una cámara o cualquier otro método, y que se procesen esos datos, lo cual implica una organización de los mismos, tratamiento de valores faltantes, normalizaciones o escalados, limpieza de estos, en general.

Todo ello con el propósito de inferir conocimiento a partir de esos datos procesados, de utilizarlos para multitud de aplicaciones en los la solución exacta y perfecta no es conocida o cuya obtención es demasiado costosa.

En la práctica a realizar en la asignatura de **Programación Técnica y Científica** se van a trabajar los dos primeros procesos descritos, obtención de datos y procesamiento de estos, además del posterior entrenamiento de un modelo clasificador con el algoritmo de **Aprendizaje Automático** *SVM* (Support Vector Machine).

1.1. Motivación

El objetivo de esta práctica es pasar por todo los procesos necesarios para la obtención de un modelo de clasificación de piernas. Estos procesos son la obtención de datos, su procesado y su uso para el entrenamiento de un modelo de **Aprendizaje Automático**. Para ello serán utilizadas las herramientas siguientes:

1. **Python**: Se utilizará el lenguaje de programación Python para la programación de una interfaz gráfica para el control del proceso de manera cómoda y para la creación de scripts que automaticen todos los procesos (obtención, limpieza, entrenamiento, predicción y creación de gráficas).

2. **CoppeliaSim (V-Rep)**: Se hará uso del simulador para preparar las escenas de distintos casos positivos y negativos (ya que vamos a tratar de crear un modelo clasificador binario) en las que se recopilarán datos por medio de un escáner láser.

3. **Librerías Python**: Dentro del entorno de Python es necesario destacar algunas librerías como **Tkinter**, para el diseño e implementación de la GUI (interfaz de usuario), **Scikit-Learn**, que permitirá el uso de funciones de aprendizaje automático para entrenar el modelo, **Matplotlib**, que servirá para la creación de gráficas que permitan analizar varias situaciones a lo largo de la práctica. Por supuesto se hacen uso de muchas más librerías, pero se consideran estas tres como las principales dentro del desarrollo.

1.2. Obtención de datos

Para la recolección de los datos se va a utilizar el simulador **CoppeliaSim**, antiguamente conocido como **V-Rep**. Es un entorno de simulación robótica 3D utilizado en la investigación y desarrollo de robótica. Ofrece una interfaz gráfica para diseñar, simular y probar robots virtuales en un entorno tridimensional. V-REP es particularmente útil para ingenieros y científicos que trabajan en robótica y control de sistemas autónomos.

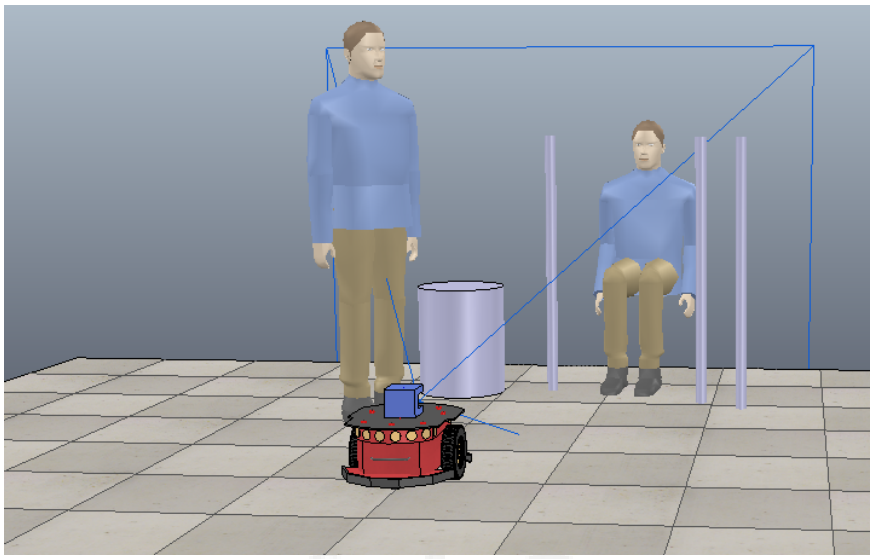

Figura 1: Entorno de trabajo de CoppeliaSim

Se puede utilizar V-REP para modelar robots, entornos y escenarios complejos, y luego simular el comportamiento de los robots en tiempo real. Además de la simulación, V-REP proporciona herramientas para el control y la programación de robots virtuales. Gracias a ello se podrá acceder a través de **Python** a los datos del robot principal, el cual se encargará de escanear el entorno con un sensor láser en buscar de objetos colisionables.

El objetivo principal de la práctica es utilizar los datos recogidos del láser, los cuáles son codificados como puntos x e y en el plano de coordenadas cartesiano. Estos puntos representan los objetos del entorno, en el caso de la práctica serán cilindros metálicos de distinto grosor y piernas.

Estos datos son guardados en archivos con formato **JSON**.

1.3. Procesamiento de datos

Podría decirse que el primer proceso, el de obtención de datos es el más importante, pues de dónde vienen los datos, cómo se han obtenido, es lo que hace que estos sean de calidad o no. Pese a ello, si no se procesan bien, no se obtendrán buenos resultados.

El tratamiento de los datos se divide en varias etapas:

1. Agrupamiento por clusters.
2. Extracción de características geométricas de cada cluster.
3. Construcción del dataset.
4. Escalado de los datos (Se hará posterior comparación con datos sin escalar).

1.4. Entrenamiento del modelo clasificador

Para la última etapa se utilizará el algoritmo **SVM** (Support Vector Machine o Máquinas de Vectores de Soporte).

La idea central de las SVM es encontrar el hiperplano óptimo que mejor separa las clases en un espacio de características. Un hiperplano es una superficie que divide un espacio en dos partes. En el caso de las SVM, se busca el hiperplano que maximiza la distancia entre las instancias más cercanas de las diferentes clases, conocidas como vectores de soporte.

Las SVM son efectivas en espacios de alta dimensión y son especialmente útiles cuando se trabaja con conjuntos de datos complejos que no se pueden separar linealmente fácilmente. Además, las SVM pueden utilizar funciones de kernel para mapear datos en espacios de características de mayor dimensión, permitiendo así la separación de clases no lineales.

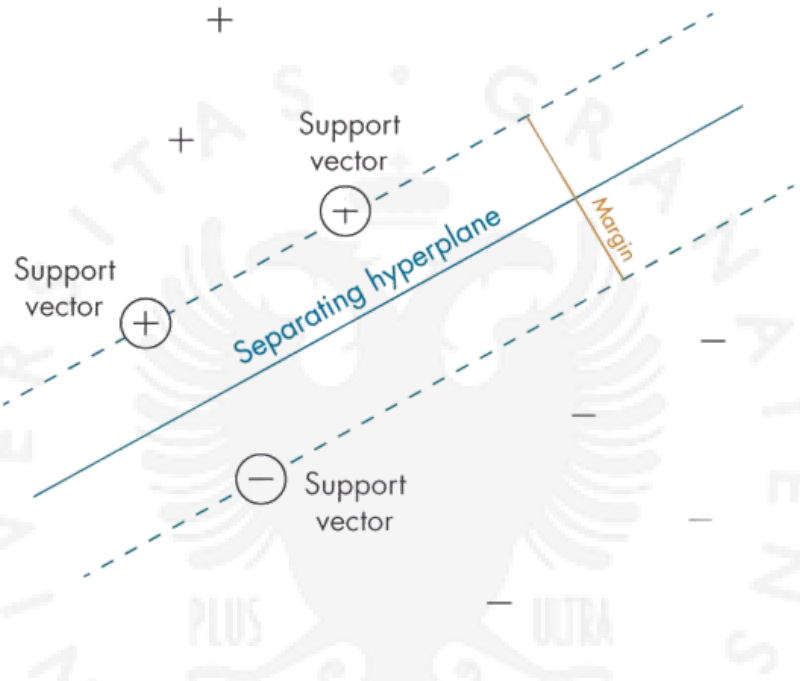

Figura 2: Idea principal de las SVMs

Se realizará posterior análisis de las técnicas utilizadas en el entrenamiento así como el uso de un kernel u otro para la obtención de un modelo final.

2. GUI con Tkinter

Se requiere la construcción de una interfaz gráfica con el objetivo de controlar la ejecución de los distintos scritps de Python de manera controlada y cómoda. La GUI (Graphical User Interface) en distintos botones principales:

1. **Conectar a V-Rep**: Como el nombre indica, se utiliza para establecer conexión con el simulador.

2. **Detener y desconectar de V-Rep**: Para poder desconectar y detener la simulación.

3. **Estado**: Es una etiqueta y no un botón, se actualiza para dar información sobre la conexión con el simulador.

4. **Capturar**: Botón que inicia la captura de datos una de las múltiples escenas. Solo se puede capturar datos si se está conectado al simulador y si se ha seleccionado una escena.
5. **Agrupar**: Su función es la de crear clusters de los datos extraídos y guardarlos en un fichero JSON. Solo deber poder ejecutarse si se han capturado todas las escenas.
6. **Extraer características**: Ejecuta el script que se encarga de construir las características de los datos. Estas se guardan en ficheros que luego son utilizados para crear el dataset final de entrenamiento. Solo podrá ejecutarse si se han agrupado previamente los datos.
7. **Entrenar clasificador**: Este botón inicia el entrenamiento de la SVM y vuelca en un fichero de texto toda la información recabada en el proceso para su posterior análisis. Este botón además guarda el clasificador con mejor fitness para su posterior uso. No se podrá entrenar si no se han extraído antes las características.
8. **Predecir**: Funcionalidad que hace uso del clasificador para predecir sobre la escena de test final en el simulador (obviamente es necesario conectarse al simulador para poder obtener los datos de test). No podrá predecir si no se ha entrado previamente.

Además de los botones principales, en las dos columnas adyacentes a estos se encuentran celdas de texto para los parámetros utilizados en los distintos procesos, los cuáles pueden ser modificados, una lista con los ficheros de texto a seleccionar en el proceso de captura de datos y un botón para salir de la interfaz.

Adicionalmente y con motivos de depuración se añade el botón *Debug*, el cuál activa todos los botones.

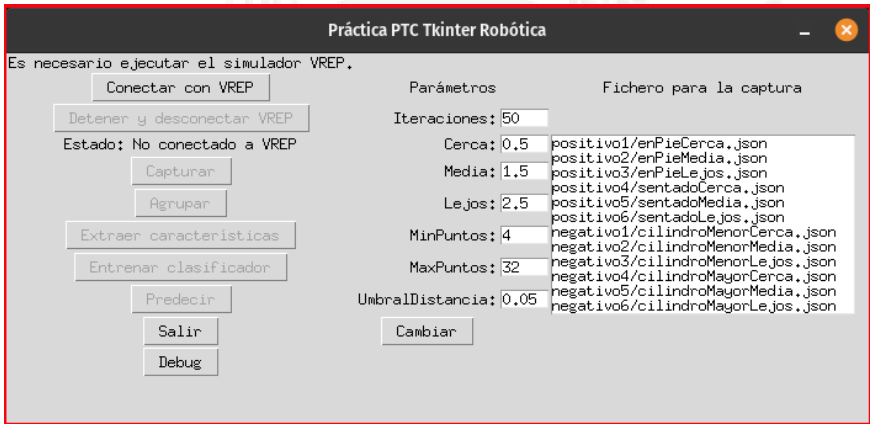

Figura 3: GUI Tkinter

3. Recolección de datos

3.1. Descripción

Para la captura de los datos se han creado 12 escenas distintas, 6 de ellas positivas y 6 negativas. Se entiende por escena positiva aquella de la que se van a extraer datos

sobre lo que se quiere predecir (en este caso se desea crear un clasificador que detecte las piernas). De esta forma las escenas negativas tratarán de piernas, mientras que las negativas serán cilindros.

Se crean tres primeras escenas positivas con una persona de pie. Cada una de ellas en distintas zonas de distancia (entre cerca, media y lejana). Las tres siguientes con una persona sentada y de igual manera, en distintas zonas delimitadas. Con las escenas negativas ocurre lo mismo, las tres primeras con un tipo de cilindro estrecho y las siguientes con uno más ancho.

Los parámetros por defecto para la captura son: **Cerca:** 0,5, **Media:** 1,5, **Lejos:** 2,5, siendo estos valores los límites inferiores. Los límites superiores para cada una respectivamente son **Media**, **Lejos** y **Lejos** +1.

3.2. Movimiento

Para calcular la distancia máxima se debe computar la distancia entre la entidad (persona o cilindros) y el robot recolector. Para ello se debe obtener las posiciones del robot (origen de coordenadas) y la entidad, después, se calcula la distancia euclídea:

$$d(x) = \sqrt{(x_{robot} - x_{entidad})^2 + (y_{robot} - y_{entidad})^2} \tag{1}$$

De esta manera, si esta distancia supera algún límite inferior o superior se resetea. Los operadores de movimiento son los siguientes:

$$x^* = x + \text{step} \tag{2}$$
$$y^* = y + \text{step} \tag{3}$$

Donde $step \in [-0{,}35, 0{,}35]$. Estos valores fueron fijados empíricamente.

En el momento que se superen los límites de distancia se resetean los valores de posición de la entidad:

$$x^* = \text{lower bound} + 0{,}5 \tag{4}$$
$$y^* = \text{rand} \tag{5}$$

Donde **lower bound** es el límite de distancia y $rand \in [-0{,}4, 0{,}4]$. Los valores y constantes fueron fijados de manera empírica. Se añade una constante a la x ya que si no la entidad podía colisionar con el robot y de esta forma desplazarlo.

Son operadores sencillos, pero citando al principio de **La navaja de Ocam**, la solución más sencilla es la más probable (o la mejor). Otros operadores más complicados saturaban el código añadiendo nada positivo a los datos.

Además de cambiar de posición, también se rota en unidades de 0,5 en 0,5 por cada iteración en sentido contrario a las agujas del reloj. Esto se hace para obtener datos de las entidades en distintas orientaciones.

4. Agrupamiento por clusters

4.1. Descripción

Los datos han de ser agrupados en pequeños conjuntos de puntos, estos son llamados clusters. Esto es así ya que un conjunto de puntos forman una entidad, en este caso una pierna o un cilindro, y lo que se requiere es clasificar una agrupación de puntos, no puntos individuales. Para ello, se opta por "*clusterizar*ütilizando el algoritmo de *agrupación por distancia de salto*.

En este algoritmo se tiene en cuenta un umbral, de forma que si de un punto al siguiente se supera en distancia el umbral establecido, se reconoce como acabado el cluster anterior.

4.2. Implementación

Para el agrupamiento en clusters se han tenido en cuenta tres parámetros, **Min Puntos**, **Max Puntos** y **Umbral Distancia**. Los dos primeros se explican por el propio nombre, son los puntos mínimos y máximos que se permiten en un cluster. El umbral es el anteriormente mencionado en la descripción del problema. Se han fijado, tras una investigación empírica, los siguientes valores para cada uno de los parámetros:

1. **Min Puntos**: Se ha fijado con valor 3, debido a que es el mínimo de puntos capturados en las pruebas con los cilíndros pequeños, aunque la mayoría eran de mínimo agrupaciones de tamaño 4.

2. **Max Puntos**: Se ha fijado en 40 puntos. Al igual que con los puntos mínimos, es el máximo observado en las pruebas. Pese a ello, el valor puede variar, pero se considera más representativo por haber sido observado en mayor número de ocaciones y ser capaz de captar bien los cilindros mayores en su mayoría.

3. **Umbral Distancia**: Se han probado valores entre 0,5 a 0,03. Por lo observado, valores superiores a 0,08 y menores a 0,02 empeoran los resultados. Por ello se fija en 0,05.

Con estos valores se implementa el algoritmo de clusterización y se obtienen resultados similares a los siguientes:

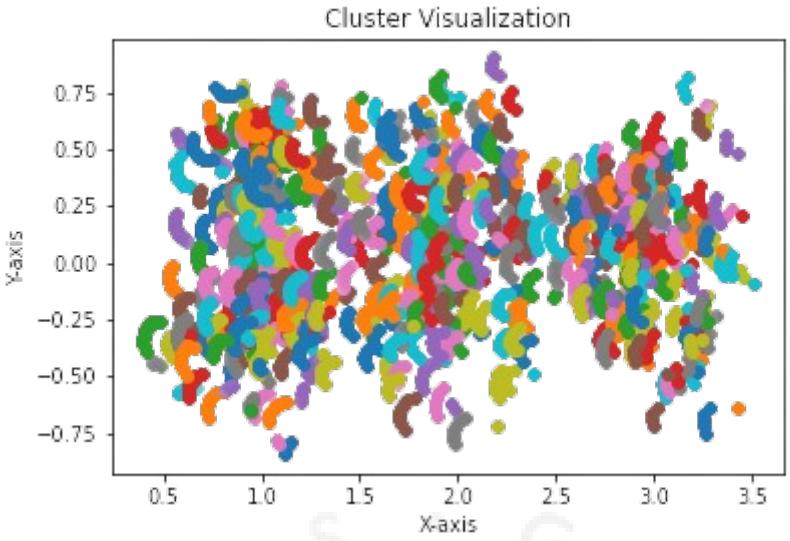

Figura 4: Clusters positivos

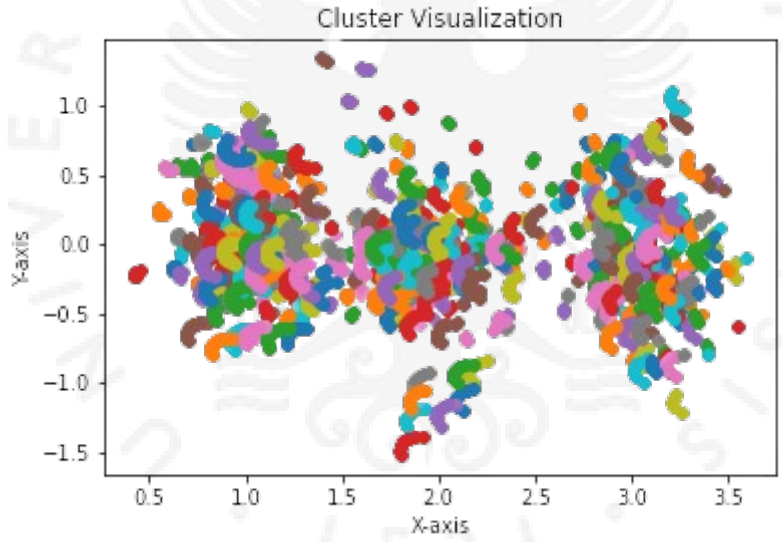

Figura 5: Clusters negativos

Si bien la gráfica es algo difícil de analizar debido a la gran cantidad de clusters obtenidos, pueden intuirse formas muy parecidad en los clusters positivos, además estas son todas de tamaño mediano, lo cual tiene sentido ya que las piernas son el objeto mediano dentro de los que se analizan y son todas iguales y los únicos elementos en los datos positivos.

En cambio en las agrupaciones negativas se pueden apreciar clusters muy grandes (cilindros anchos) y clusters pequeños (cilindros pequeños), todos ellos mezclados.

Obviamente habrá falsos positivos y falsos negativos entre agrupaciones, pero es un error imposible de evitar, lo único posible es minimizarlo.

Estos clusters deben ser guardados en archivos JSON indicando el número de cluster, número de puntos y los puntos x e y que los forman.

5. Construcción de características geométricas

5.1. Descripción

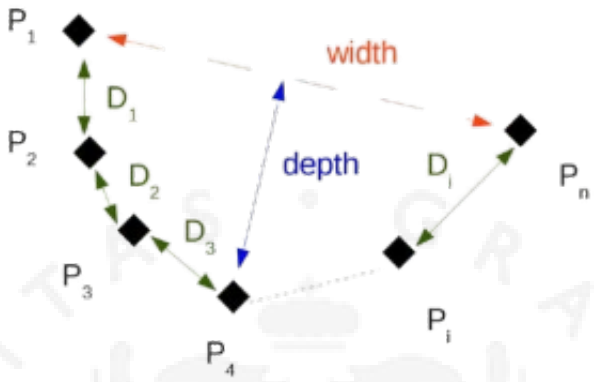

Figura 6: Características geométricas de un cluster

Por cada cluster o agrupación se requiere construir tres tipos de características relacionadas con la geometría del mismo, estas son **perímetro**, **anchura** y **profundidad**. Pueden observarse en la figura 6. Estas deben ser guardadas en archivos JSON junto a la clase de cada cluster y más tarde agrupadas en un dataset final con extensión csv.

5.2. Cálculo de las características

El cálculo de cada una de estas características requiere de fórmulas geométricas básicas:

1. **Perímetro**: $p(x) = \sum_{i=1}^{n} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$
2. **Anchura**: $w(x) = \sqrt{(x_n - x_1)^2 + (y_n - y_1)^2}$
3. **Profundidad**: $d(x) = max(\alpha_i)$

Donde α es la distancia del punto a la recta que forma el primer con el último punto, de forma que se ha de calcular para cada punto i del clúster.

$$\alpha_i = \frac{|Ax_i + By_i + C|}{\sqrt{A^2 + B^2}} \tag{6}$$

Y donde A, B, C:

$$A = y_n + y_0 \tag{7}$$
$$B = x_0 - x_n \tag{8}$$
$$C = y_0 x_n - y_n x_0 \tag{9}$$

6. Entrenamiento del Modelo con SVM

6.1. Descripción

Se utiliza un clasificador de Máquinas de Vectores de Soporte (SVM) para entrenar el modelo. El conjunto de datos se divide en conjuntos de entrenamiento y prueba para evaluar la precisión del modelo.

En cuanto a los kernels en SVM, son funciones matemáticas que transforman los datos de entrada en un espacio de características de mayor dimensión. Estos kernels son fundamentales para permitir a las SVM manejar conjuntos de datos que no son linealmente separables en su forma original. Los kernels utilizados son:

1. **Kernel Lineal:** Utilizado cuando se asume que los datos son linealmente separables en el espacio original. La función de decisión es simplemente un hiperplano en el espacio de características transformado.

2. **Kernel Polinómico:** Introduce la no linealidad mediante la aplicación de transformaciones polinómicas a los datos originales. El parámetro de grado controla la complejidad de la transformación.

3. **Kernel Radial (RBF):** También conocido como kernel gaussiano, es eficaz en la captura de decisiones no lineales. Introduce una función de base radial que mide la similitud entre un punto y los puntos de entrenamiento, asignando un peso basado en la distancia.

6.2. Procedimiento

Los pasos seguidos en la etapa de entrenamiento son los siguientes:

1. Se lee el conjunto de datos desde un archivo CSV.

2. Se escalan los datos para normalizarlos usando la media y desviación estándar.

3. Se dividen los datos en conjuntos de entrenamiento y prueba.

4. Se entrena el clasificador SVM utilizando diferentes kernels (lineal, polinómico, radial).

5. Se realiza una búsqueda de hiperparámetros utilizando GridSearchCV.

6. Se evalúa la precisión del clasificador en el conjunto de prueba.

7. Se usa validación cruzada de 5-folds para evaluar el rendimiento del modelo.

8. Guarda el mejor clasificador y sus resultados en un archivo de resultados.

El escalado de los datos ha sido un paso fundamental a la hora de mejorar el rendimiento del clasificador. Esto se debe a que as SVM calculan distancias entre puntos de datos para determinar la posición del hiperplano de decisión. Si las características tienen magnitudes muy diferentes, la SVM dará más peso a aquellas con magnitudes más grandes, lo que puede sesgar el modelo hacia esas características dominantes.

Además del escalado, la búsqueda de hiperparámetros se vuelve indispensable si se

quiere obtener el modelo más robusto posible. Los hiperparámetros afectan directamente al rendimiento del modelo. Cambios en los valores de los hiperparámetros pueden conducir a mejoras o empeoramientos en la capacidad predictiva del modelo, debido a que obtienen la mejor combinación de estos tras aplicar una búsqueda intensiva.

Los distintos hiperparámetros son **C**, común a todos los kernels, ya que este controla la cantidad de regularización aplicada, **gamma** que es la anchura de la función de base radial y **degree**, que como el nombre indica es un hiperparámetro del kernel polinomial que controla el grado del polinómio.

6.3. Análisis de resultados

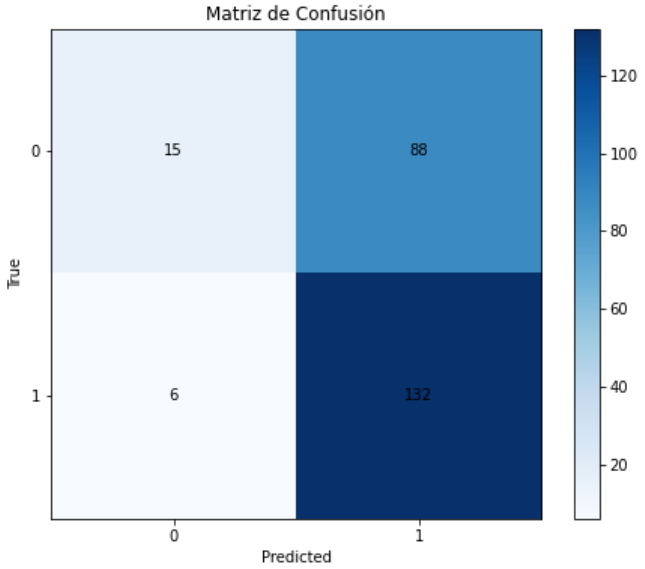

Figura 7: Matriz de confusión con kernel lineal

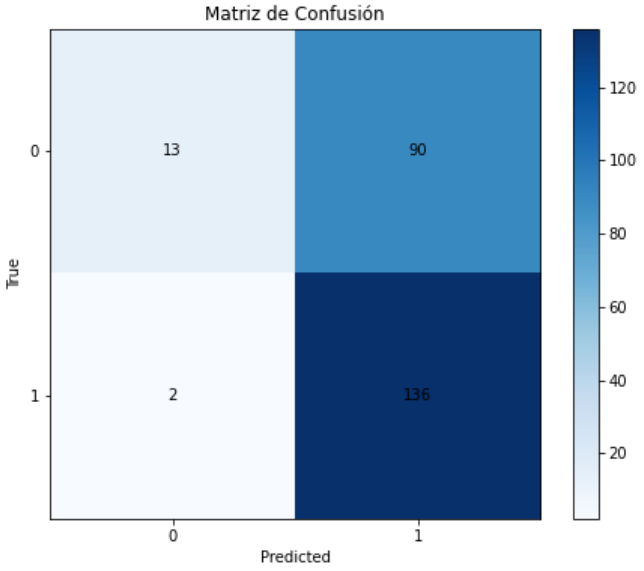

Figura 8: Matriz de confusión con kernel polinómico

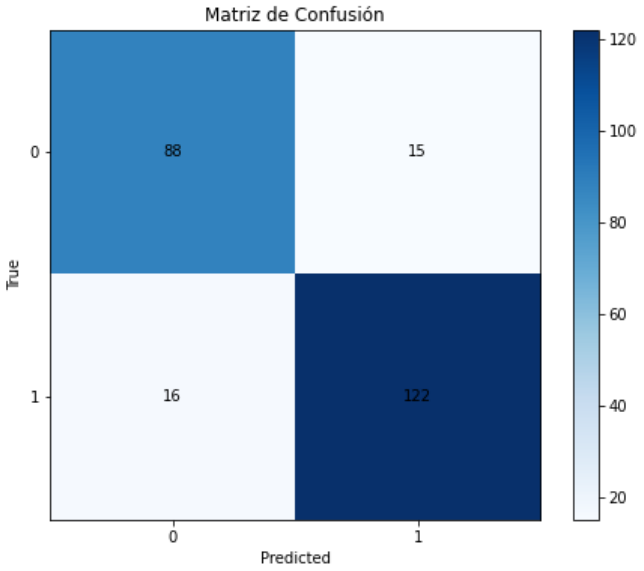

Figura 9: Matriz de confusión con kernel radial

Se puede observar que el SVM radial es muchísimo mejor que los otros dos candidatos, que según la matriz de confusión de ambos, no son capaces de discernir los casos negativos bien. De hecho, se percibe una tendencia a clasificar solo piernas, fallando por completo con los casos negativos. En cambio el modelo basado en el kernel radial es capaz de separar casos negativos de positivos en la mayoría de instancias propuestas.

La razón por la cual el SVM con kernel radial supera al polinómico y al lineal en este

caso tiene que ver con la naturaleza de los datos y cómo estos kernels manejan las relaciones complejas.

Primero, el kernel lineal asume que los datos son separables por una línea recta. Si los datos son bastante complejos y no se pueden dividir fácilmente con una línea recta, pues no va a funcionar tan bien.

Luego está el kernel polinómico, que puede manejar relaciones más complejas. Sin embargo, tiene sus propios problemas. Si se usa un grado polinómico alto, se puede caer en el sobreajuste, donde el modelo se adapta demasiado a los datos de entrenamiento y no generaliza bien a nuevos datos.

El kernel rbf utiliza funciones de base radial, lo que significa que es más flexible y puede capturar relaciones no lineales de manera más eficaz sin llegar a sobreajustar.

6.4. Resultado final

El clasificador final obtenido, es decir, el mejor, ha sido el de base radial. Este modelo ha obtenido los siguientes resultados:

1. **Accuracy (Exactitud) en el conjunto de prueba:**
   - Accuracy: 0.8958, indicando que el 89.58 % de las predicciones son correctas.
2. **Matriz de Confusión:**
   - TP: 111, TN: 104, FP: 14, FN: 11.
3. **Precision, Recall y F1-Score:**
   - Precision (Precisión): 0.89.
   - Recall (Recall): 0.91.
   - F1-Score: 0.90.
4. **Reporte de Clasificación:**
   - Precision y recall altos para ambas clases.
5. **Accuracy en 5-fold Cross Validation:**
   - Accuracy promedio: 0.7927, desviación estándar: ± 0.2356.
6. **Conclusión:**
   - El modelo con kernel RBF muestra un buen rendimiento, con un accuracy sólido y un equilibrio entre precision y recall.

El resultado de predecir con el mejor clasificador encontrado es el siguiente:

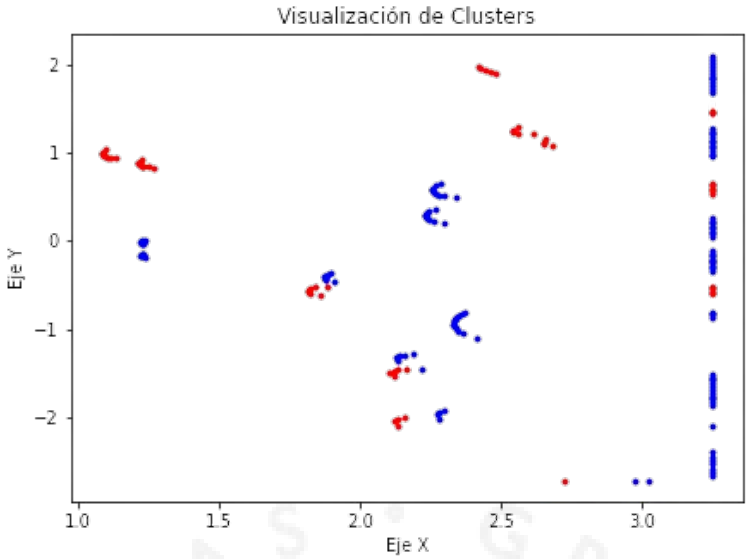

Figura 10: Resultado final

Se ha de decir que el resultado final es algo decepcionante, pues pese a acertar la mayoría, falla en algunos puntos de la pared y en algún cluster completo.