INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

Tarea 04

Elementos básicos de un clasificador inteligente de patrones (cip)

Edgar Fernando Espinosa Torres

Clasificación inteligente de patrones

Dr. Cornelio Yáñez Márquez

CIUDAD DE MÉXICO, 23 de septiembre de 2023.

Índice general

Índice de figuras

Índice de tablas

Introducción

En el ámbito del reconocimiento de patrones, la clasificación es una tarea fundamental. A través de esta, es posible clasificar patrones cuya clase se desconoce, apoyándonos en patrones conocidos. Por su simplicidad, el Clasificador Euclidiano emerge como una herramienta para este fin. En este trabajo, nos centraremos en el desarrollo y la evaluación de un Clasificador Euclidiano basado en un dataset específico derivado de Haberman's Survival, enfocándonos particularmente en datos asociados al año 1965. La implementación se realiza siguiendo los cuatro elementos básicos que cada cip debe tener; desde tener el dataset adecuado, pasando por el método de validación, la implementación del algoritmo y finalizando con el cálculo de la medida para evaluar el desempeño.

Desarrollo y discusión

Propósito de la tarea

Proponer una partición arbitraria y desarrollar los cuatro elementos básicos de un cip sencillo: el Clasificador Euclidiano.

Parte 1: En el dataset D del año 65 ya estudiado, proponer una partición fija de modo que en el conjunto de prueba queden 4 patrones de cada una de las dos clases (8 en total).

Reportar el desarrollo completo y el valor de accuracy.

Parte 1

Recordemos que un clasificador inteligente de patrones (cip) consta de cuatro elementos básicos:

- Un dataset
- Un método de validación
- Un algoritmo
- Una medida para evaluar el desempeño

Primer elemento básico: dataset

A continuación, trabajaremos con un dataset derivado de Haberman's Survival de UCI [1]. Al dataset original se le realizó un conjunto de técnicas conocidas como limpieza de datos para poder aislar al año 1965.

Con ello se obtiene un dataset para trabajar en single-label classification. El dataset generado sólo tiene dos atributos (tipo numérico). Contiene pocos patrones, es decir, exhibe una cardinalidad pequeña y también es biclase.

Segundo elemento básico: método de validación

Un método de validación parte el dataset D en dos conjuntos ajenos o disjuntos: uno de entrenamiento E y uno de prueba P, de modo que los conjuntos E y P formen una partición del dataset D.

En este caso, se propone una partición fija (elegida arbitrariamente) de modo que en el conjunto de prueba queden 4 patrones de cada una de las dos clases (8 en total).

Utilizando la notación de MATLAB para vectores:

$E = Clase1_E \cup Clase2_E$

{[30, 0], [31, 4], [40, 0], [41, 0], [42, 0], [61, 8], [64, 22], [67, 0], [69, 0], [77, 3],

[45, 6], [46, 20], [47, 0], [53, 1], [53, 12], [61, 0], [62, 19], [74, 3], [78, 1]}

$P = Clase1_P \cup Clase2_P$

{[43, 0], [50, 4], [51, 0], [52, 0], [54, 23], [54, 5], [56, 9], [60, 0]}

Claramente se verifica que E y P forman una partición del dataset D.

$E \cap P = \emptyset$

$E \cup P = D$

Tercer elemento básico: algoritmo

El algoritmo fue implementado en Python apoyándose de la biblioteca Numpy, la cual es una biblioteca de Python que proporciona soporte para trabajar con arrays y matrices grandes y multidimensionales, junto con una colección de funciones matemáticas de alto nivel para operar con estos datos [2].

En una primera fase (llamada fase de aprendizaje o fase de entrenamiento), se entrena el cip con todos y cada uno de los patrones del conjunto E.

En nuestro ejemplo, la cardinalidad del conjunto E es 27 (14 patrones de la Clase 1 y 13 de la Clase 2).

El nombre del cip utilizado es Clasificador Euclidiano. La fase de entrenamiento de este cip consiste en calcular los centroides de todos los vectores de entrenamiento para cada una de las clases.

Al concluir la fase de entrenamiento, se tuvieron tantos centroides como clases hubo. En nuestro ejemplo, a la salida de la fase de entrenamiento tendremos $centroide_1$ y $centroide_2$.

Con esto termina la fase de entrenamiento. A la salida de esta fase se tienen los valores de las coordenadas de ambos centroides.

$$centroide_1: [52.2 \;\; 3.7]$$

$$centroide_2: [57.7 \;\; 6.9]$$

En la fase de prueba (o de clasificación) el cip asignó de manera inteligente la clase a cada uno de los patrones de prueba $p_1, p_2, \ldots, p_8$. Se calculó la distancia Euclidiana del patrón de prueba a ambos centroides y el cip asignó la clase del centroide más cercano.

A continuación, se presenta una tabla que incluye las operaciones y resultados para los ocho patrones de prueba:

Patrón	**Distancia al centroide centroide$_1$**	**Distancia al centroide centroide$_2$**	**Clase asignada**	**Resultado**
p_1	9.9	16.2	1	ACIERTO
p_2	2.2	8.2	1	ACIERTO
p_3	3.9	9.6	1	ACIERTO
p_4	3.7	8.9	1	ACIERTO
p_5	19.4	16.5	2	ACIERTO
p_6	2.2	4.2	1	ERROR
p_7	6.5	2.7	2	ACIERTO
p_8	8.6	7.3	2	ACIERTO

Tabla 1: Resultados de la fase de entrenamiento

Dadas las características de nuestro dataset es posible representar los atributos numéricos en un plano cartesiano.

Para ilustrar se muestran algunas gráficas:

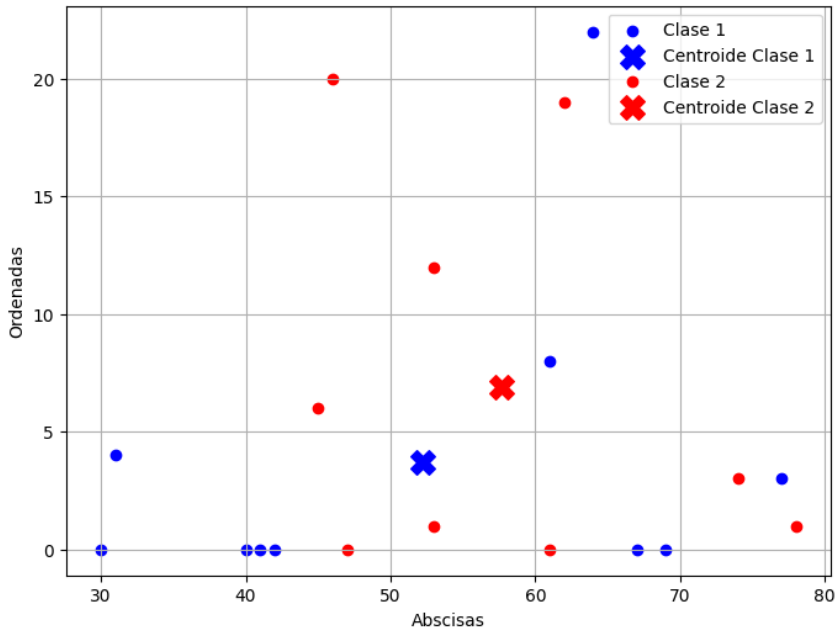

Figura 1: Vectores y sus centroides

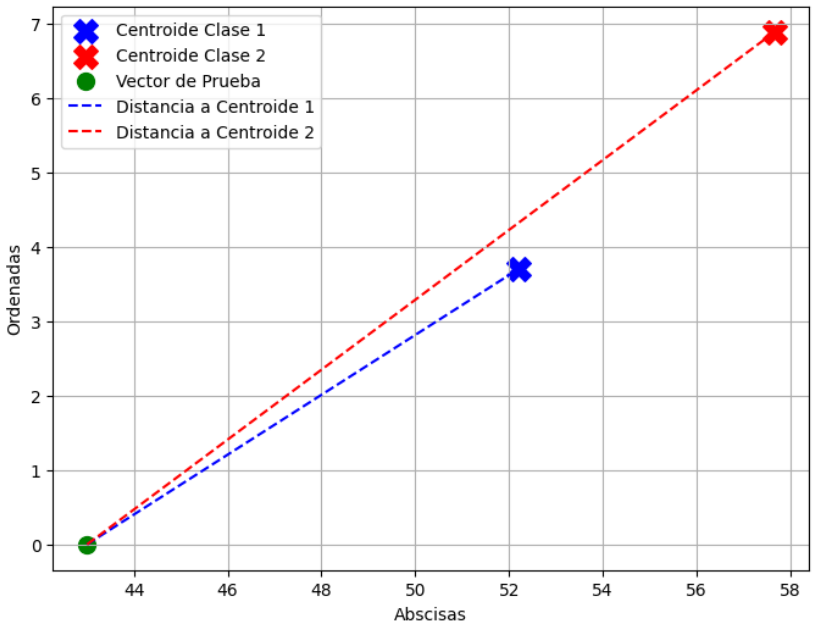

Figura 2: Distancias del vector p_1 a los centroides

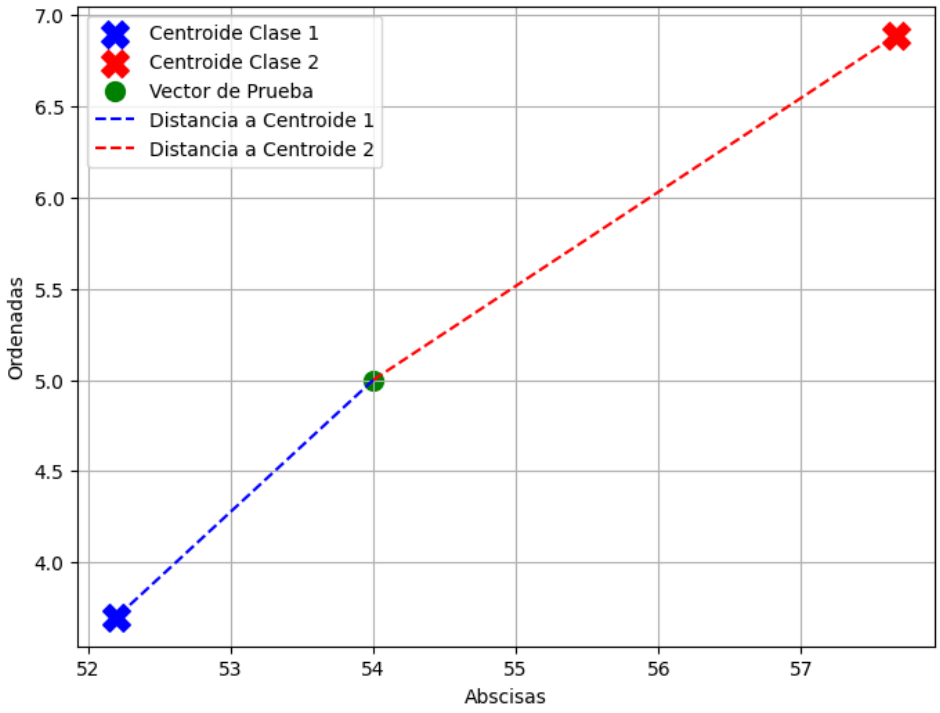

Figura 3: Distancias del vector p_6 a los centroides

Cuarto elemento básico: medida de desempeño

El Imbalance Ratio del dataset es $IR = \frac{14}{13} = 1.07 \leq 1.5$. Esto implica que el dataset está balanceado y por lo tanto es posible utilizar la medida de desempeño llamada accuracy.

En la fase de prueba del cip se obtuvieron 7 aciertos y 1 error.

$$accuracy = \frac{7}{8} = 0.875 = 87.5\%$$

Conclusiones

El Clasificador Euclidiano, si bien es uno de los algoritmos más simples en el ámbito de la clasificación, ha demostrado ser eficaz en este estudio. La estrategia de clasificar los patrones utilizando las distancias euclidianas a centroides calculados previamente ofrece una visión intuitiva y directa sobre cómo se categorizan los datos.

Mediante las particiones elegidas E y P y la implementación en el lenguaje Python con la ayuda de la biblioteca Numpy, hemos logrado un accuracy del 87.5% en nuestro conjunto de prueba. Si bien este resultado es prometedor, es esencial recordar que el rendimiento puede variar según la naturaleza del dataset y las particiones seleccionadas.

No obstante, los resultados reiteran la utilidad del Clasificador Euclidiano, en contextos especiales donde se busca una solución de clasificación sencilla.

Referencias bibliográficas

[1] *UCI Machine Learning Repository*. (s. f.-c).

https://archive.ics.uci.edu/dataset/43/haberman+s+survival

[2] *NUMPY Documentation — NUMPY V1.26 Manual*. (s. f.).

https://numpy.org/doc/stable/

Anexo

```
import numpy as np

def leer_archivo(nombre_archivo):
    with open(nombre_archivo, 'r') as file:
        data = [list(map(int, line.split())) for line in
file.readlines()]
    return np.array(data)

def calcular_centroide(data):
    return np.mean(data, axis=0)

def distancia_euclidiana(p1, p2):
    return np.linalg.norm(p1 - p2)

def clasificar(vector, centroide1, centroide2):
    distancia_c1 = distancia_euclidiana(vector, centroide1)
    distancia_c2 = distancia_euclidiana(vector, centroide2)

    return 'c1' if distancia_c1 < distancia_c2 else 'c2'

def main():
    # Lectura de los archivos
    entrenamiento_c1 = leer_archivo('entrenamiento_c1.txt')
    print("Datos de entrenamiento_c1:\n", entrenamiento_c1)

    entrenamiento_c2 = leer_archivo('entrenamiento_c2.txt')
    print("\nDatos de entrenamiento_c2:\n", entrenamiento_c2)

    prueba_c1 = leer_archivo('prueba_c1.txt')
    print("\nDatos de prueba_c1:\n", prueba_c1)

    prueba_c2 = leer_archivo('prueba_c2.txt')
    print("\nDatos de prueba_c2:\n", prueba_c2)

    # Cálculo de los centroides
    centroide_c1 = calcular_centroide(entrenamiento_c1)
    centroide_c2 = calcular_centroide(entrenamiento_c2)

    print("Centroide de c1:", centroide_c1)
    print("Centroide de c2:", centroide_c2)
    print()
```

```
    # Clasificación de los vectores de prueba
    resultados_c1 = [clasificar(vector, centroide_c1, centroide_c2)
for vector in prueba_c1]
    resultados_c2 = [clasificar(vector, centroide_c1, centroide_c2)
for vector in prueba_c2]

    print("Resultados para prueba_c1:")
    for i, res in enumerate(resultados_c1):
        print(f"Vector {i+1}: {res}")

    print("\nResultados para prueba_c2:")
    for i, res in enumerate(resultados_c2):
        print(f"Vector {i+1}: {res}")

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