UNIVERSITAT JAUME I

Grado en Matemática Computacional

Trabajo Final de Grado

Métodos de clasificación supervisada: Support Vector Machine

Autor:
Rubén Robles Castellar

Tutor académico:
María Victoria Ibáñez Gual

Fecha de lectura: Junio de 2023
Curso académico 2022/2023

Resumen

En este trabajo fin de grado se realiza un estudio sobre los algoritmos de clasificación supervisada y más concretamente sobre el Support Vector Machine (SVM). Para ponernos en contexto, en primer lugar se define qué es un algoritmo de clasificación junto a una breve descripción de los algoritmos más utilizados como son la regresión logística, el análisis discriminante lineal y el método de los k-vecinos más próximos. Seguidamente, nos centraremos en dos métodos de clasificación basados en hiperplanos separadores como son el Perceptrón y el método de los hiperplanos separadores óptimos. Para finalizar con el estudio teórico, se examina el método del Support Vector Machine partiendo de una serie de definiciones básicas para llegar a conceptos más complejos y así poder entender a la perfección el algoritmo.

Finalmente, he realizado un programa en Python empleando el algoritmo de SVM y haciendo uso de una base de datos con información de pacientes médicos, algunos con enfermedades cardiovasculares. He abordado el problema de clasificar a los pacientes en si tienen o no alguna enfermedad cardiaca.

Palabras clave

Clasificación supervisada, Hiperplanos separadores, Espacios de Hilbert, Núcleo reproductor, Support Vector Machine.

Keywords

Supervised classification, separating Hyperplanes, Hilbert Spaces, reproducing Kernels, Support Vector Machine.

Índice general

Capítulo 1

Introducción

El presente trabajo final de grado en Matemática Computacional tiene el objetivo de entender y analizar un método muy utilizado de aprendizaje automático llamado Support Vector Machine (SVM). El aprendizaje automático o machine learning consiste en el desarrollo de técnicas y/o algoritmos que permitan extraer información de datos, observaciones o iteraciones con el mundo y cuya finalidad es que con dicha información de entrada, esa técnica o algoritmo sea capaz de extraer patrones o construir modelos que puedan ser utilizados para generalizar correctamente nuevas circunstancias.

En este trabajo se desarrollan los fundamentos y los conceptos matemáticos para entender el algoritmo del SVM. Para ello, se va a ir incrementando el nivel de complejidad para abarcar los diferentes aspectos teóricos del algoritmo, partiendo de conceptos básicos para, finalmente, llegar a los más complejos. Todos estos fundamentos son necesarios para poder explotarlo al máximo y poder aplicarlo al mundo real.

Además, todo el análisis teórico se ha concluido con la implementación de un código en Python donde se ha programado el algoritmo SVM y se ha aplicado a un problema de clasificación supervisada derivado de una base de datos con información de pacientes los cuales han tenido o no enfermedades cardiacas. Esto destaca la importancia de algunos modelos matemáticos en campos como la medicina.

En cuanto a la estructura del trabajo, se divide en cuatro capítulos y dos anexos. El capítulo 2 es una introducción a los métodos de clasificación supervisada, donde se define qué son los algoritmos de clasificación supervisada y se da una visión de varios métodos importantes. A continuación, en el capítulo 3 se explican cuáles son los dos métodos de clasificación más básicos que están basados en hiperplanos separadores. Estos dos métodos son, primero, el método del Perceptrón desarrollado en la sección 3.2 y el método de los hiperplanos separadores óptimos,

explicado en 3.3. En el capítulo 4 ya se explica el método del Support Vector Machine, primero se introduce el método junto con un ejemplo para poder entender lo que se va a desarrollar en el resto del capítulo. Se definen una serie de términos que serán necesarios y además, en las secciones 4.3 y 4.4 se introducen los espacios de Hilbert con núcleo reproductor y el método de las funciones núcleo. Para finalizar, en la sección 4.5 se explica la aplicación de toda la teoría desarrollada en los Support Vector Machine.

Siguiendo con la estructura del trabajo, en el capítulo 5 se expone la aplicación del modelo estudiado. Se define en una primera sección 5.1 la base de datos utilizada en la implementación del código junto a una breve descripción de cada uno de los atributos de los pacientes analizados y en la sección 5.3 la explicación del propio código. Asimismo, en la sección 5.4 se exponen los resultados obtenidos con el algoritmo SVM y finalmente, en la sección 5.5 se realiza una comparación entre los distintos métodos de clasificación supervisada para comprobar que el método más eficaz es el SVM.

Finalmente, las conclusiones se detallan en el capítulo 6.

En cuanto a los anexos, el primero de los anexos desarrolla el método del gradiente descendiente y en segundo lugar, se explica los Multiplicadores de Lagrange que son utilizados en el algoritmo del SVM. Finalmente, el último de los anexos es el código implementado tanto del algoritmo de SVM como del algoritmo de regresión logística, análisis discriminante lineal y el de los k-vecinos más próximos.

Capítulo 2

Introducción a los métodos de clasificación supervisada

Un algoritmo de clasificación supervisada es una técnica estadística que consiste en aprender de los datos con el objetivo de poder clasificar nuevas observaciones. Es decir, al algoritmo se le proporciona un conjunto de observaciones (inputs), $\{x_i\}_{i=1,\cdots,n}$ con $x_i \in X$ (aunque por simplificar y para que los conceptos queden más claros a menudo cogeremos $X = \mathbb{R}^p$), donde cada observación viene etiquetada con la clase a la que pertenece (output), y_i. El objetivo es que dada una nueva observación, el algoritmo sea capaz de predecir la clase a la que pertenece.

Entre las técnicas de clasificación supervisada más utilizadas destacan la regresión logística, el análisis discriminante lineal y el método de los k-vecinos más próximos.

- La **regresión logística** [2] es una técnica de clasificación supervisada, específica para cuando la variable que nos da la clase a la que pertenece cada observación es una variable dicotómica. Es decir, cada observación, $i \in \{1, \cdots, n\}$, viene caracterizada por p variables, recogidas en un vector $x_i \in \mathbb{R}^p$, y pertenece a una clase (de dos posibles), que se etiqueta como $y_i \in \{0, 1\}$. La finalidad del algoritmo es predecir a cuál de las dos posibles clases pertenece una nueva observación y para ello, dada una nueva observación $\hat{x} = (\hat{x}_1, ..., \hat{x}_p)$, se obtiene la probabilidad, p, que generalmente será la probabilidad de pertenecer a la clase 1. Esta probabilidad se aproximará con la función logística:

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 \hat{x}_1 + ... + \beta_p \hat{x}_p)}} \tag{2.1}$$

  donde los parámetros $\beta_0, ..., \beta_p$ son desconocidos y usualmente se estiman con el método de máxima verosimilitud.

Además, tal y como está definida la función, se puede asegurar que p será un valor entre 0 y 1. Por tanto, si p es la probabilidad de que una observación pertenezca a la clase 1, se definirá un valor umbral de forma que la nueva observación $\hat{x}$ se etiquetará como 1 si la p está por encima del valor umbral o como 0 si está por abajo.

- El **análisis discriminante lineal** [5, 3] es una técnica de clasificación supervisada, donde la variable de clasificación es una variable cualitativa Y que ya no tiene que ser necesariamente dicotómica. De nuevo partimos de un conjunto de observaciones $x_i \in \mathbb{R}^p$, $i = 1, \cdots, n$, etiquetadas como $y_i \in \{1, 2, \cdots, K\}$,$\forall i \in \{1, \cdots, n\}$. El modelo permitirá clasificar, entre las $K \geq 2$ posibles clases, cualquier nueva observaciones $\hat{x}$ obteniendo la probabilidad de pertenecer a cada clase k por separado $P(Y = k|X = \hat{x})$ utilizando el teorema de Bayes.

  Por el teorema de Bayes se tiene que

$$p_k = P(Y = k|X = x) = \frac{P(X=x|Y=k)P(Y=k)}{P(X=x)} = \frac{f_k(x)\pi_k}{\sum_{l=1}^{K} \pi_l f_l(x)}$$

  siendo $f_k()$ la función de densidad de X en cada clase k, y π_k la probabilidad a priori para la clase k.

  Para poder calcular $P(Y = k|X = x)$, por el teorema de Bayes, necesitamos conocer $f_k()$, que como hemos dicho es la distribución de $X \mid Y = k$. Es habitual en este contexto asumir que la distribución de $X \mid Y = k$ es normal multivariante, es decir,

$$X \mid Y = k \sim N(\mu_k, \Sigma_k), \forall k = 1, \cdots, K$$

  y por tanto:

$$f_k(X) = f(x \mid Y = k) = \frac{1}{(2\pi)^{1/2} \mid \Sigma_k \mid^{1/2}} exp(-\frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1}(x - \mu_k))$$

  El análisis discriminante lineal (LDA) surge en el caso particular de considerar que todas las clases tienen la misma matriz de varianzas-covarianzas, i.e. que $\Sigma_k = \Sigma \ \forall k$.

  En la práctica, desconocemos tanto π_k como los valores de los parámetros de las distribuciones gausianas y necesitamos estimarlos a partir de los datos observados:

  - Definimos $N_k = \#\{y_i : y_i = k\}$
  - Estimamos

$$\hat{\pi}_k = \frac{N_k}{n}$$

  - Estimamos las medias:

$$\hat{\mu}_k = \frac{\sum_{i:y_i=k} x_i}{N_k}$$

- Estimamos la matriz de varianzas covarianzas:

$$\hat{\Sigma} = \sum_{k=1}^{K} \sum_{i:y_i=k} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T/(n-K)$$

Una vez estimados los parámetros, dada una nueva observación $\hat{x}$ se asignará a la clase k que tenga mayor p_k, entonces sustituyendo f_k en p_k se asignará la nueva observación a la clase para la cual

$$\delta_k(\hat{x}) = \hat{x}^T \hat{\Sigma}^{-1} \hat{\mu}_k - \frac{1}{2} \hat{\mu}_k^T \Sigma^{-1} \hat{\mu}_k^T + log(\hat{\pi}_k)$$

sea mayor.

- El **método de los k-vecinos más próximos** es un algoritmo de aprendizaje supervisado que comienza seleccionando un valor para la constante k. El algoritmo consiste en dada una observación, seleccionar los k elementos más cercanos y analizar las clases a las que pertenecen para tomar la decisión sobre la muestra a clasificar. Es decir, dado un conjunto de observaciones $x_i \in \mathbb{R}^p$, $i = 1, \cdots, n$ etiquetadas según el grupo al que pertenecen ($y_i \in \{1, 2, \cdots, K\}$) y dada una nueva observación $\hat{x}$:

  - se calcula la distancia entre la nueva observación y todas las disponibles en el conjunto observado: $d(\hat{x}, x_i), \forall i = 1, \cdots, n$
  - y se seleccionan las k observaciones más cercanas a $\hat{x}$ (k-vecinos más próximos).

  La nueva observación $\hat{x}$ se clasifica como la clase que más se repite entre estos k vecinos más próximos.

Además de los métodos anteriores, existen algunos más avanzados como son los hiperplanos separadores y el Support Vector Machine (SVM), que se explicarán a continuación.

Capítulo 3

Primeros métodos de clasificación basados en hiperplanos separadores

3.1. Introducción

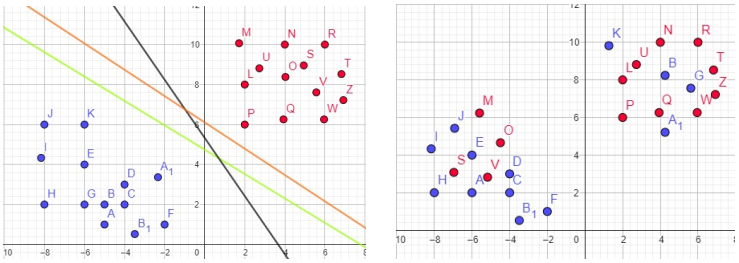
Partiendo de un conjunto de observaciones $\{x_i\}_{i=1,\cdots,n}$, $x_i \in \mathbb{R}^p$, etiquetadas según la clase a la que pertenezcan, en esta sección se van a estudiar dos métodos para separar las distintas clases con hiperplanos. Para empezar, vamos a simplificar y a suponer que las observaciones están etiquetadas en dos grupos, por lo que solo necesitaremos un hiperplano separador.

Si $p = 2$ y dibujamos con colores distintos las observaciones de cada clase, en la figura 3.1 (a) se puede apreciar que existen dos clases separables con infinitos hiperplanos separadores y en la figura 3.1 (b) las clases no son separables y, por tanto, no existe ningún hiperplano que separe ambas clases.

El primero de los métodos que vamos a revisar en la sección 3.2 es el modelo de *Perceptrón* que encuentra un hiperplano separador en el caso de que exista, y el segundo que revisaremos en la sección 3.3 es el método del *hiperplano separador óptimo* que en caso de existir encontrará un hiperplano separador y en el caso de no existir, encontrará un hiperplano que maximice la distancia de cada punto al hiperplano separador.

Antes de continuar, veamos un pequeño repaso sobre los hiperplanos que llamaremos L, con ecuación $f(x) = \beta_0 + \beta^T x = 0$, donde $\beta_0 \in \mathbb{R}$ y tanto x como β^T son vectores de $\mathbb{R}^p$:

1. Para cada dos puntos x_1 y x_2 en L se tiene que $\beta^T(x_1 - x_2) = 0$. Esto es porque para x_1 se



(a) (b)

Figura 3.1: Un ejemplo de clases linealmente separables en (a) y de clases no separables linealmente en (b)

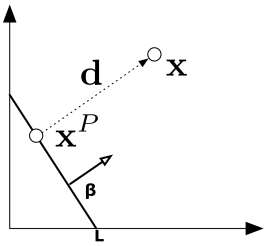


Figura 3.2: Ejemplo distancia de un punto al hiperplano.

tiene que $\beta_0+\beta^T x_1 = 0$ y para x_2, $\beta_0+\beta^T x_2 = 0$ y entonces $\beta^T x_1 - \beta^T x_2 = \beta^T(x_1-x_2) = 0$.

2. Para todo punto x_0 en L, $\beta^T x_0 = -\beta_0$

3. Como se muestra en la figura 3.2, queremos calcular la distancia de un punto cualquiera $x \in \mathbb{R}^p$ al hiperplano, siendo x^P la proyección ortogonal de x en L y d el vector que une ambos puntos: $x^P = x - d$. Este vector d será perpendicular al hiperplano y paralelo a β, por tanto, $d = \alpha\beta$ para algún $\alpha \in \mathbb{R}$.

   Sabemos que x^P está en L, luego $\beta_0 + \beta^T x^P = 0$ y se tiene que

   $$\beta_0 + \beta^T x^P = \beta^T(x - d) + \beta_0 = \beta^T(x - \alpha\beta) + \beta_0 = 0.$$

   Lo que implica que $\alpha = \frac{\beta^T x + \beta_0}{\beta^T \beta}$.
   Ahora se busca $\|d\|$ que es la distancia del punto al hiperplano,

   $$\|d\| = \sqrt{d^T d} = \sqrt{\alpha^2 \beta^T \beta} = \alpha\sqrt{\beta^T\beta} = \frac{\beta^T x + \beta_0}{\sqrt{\beta^T\beta}} = \frac{f(x)}{\|f'(x)\|}. \tag{3.1}$$

   Por tanto la distancia entre un punto x y un hiperplano de ecuación $f(x) = 0$ es $\frac{f(x)}{\|f'(x)\|}$

3.2. Algoritmo de aprendizaje del Perceptrón

El algoritmo del *Perceptrón* [5] tiene como objetivo encontrar un hiperplano separador, minimizando el número de puntos mal clasificados.

Como el resto de algoritmos que hemos visto hasta ahora, es un algoritmo de clasificación supervisada, es decir, existe un conjunto de datos que están etiquetados con etiquetas que representan la clase a la que pertenecen. La variable que representa estas etiquetas se suele denotar con la letra y. En este algoritmo, en caso más sencillo de haber solo dos clases, los valores de las etiquetas suelen ser +1 y -1.

El objetivo es encontrar un hiperplano $\beta_0 + \beta^T x = 0$ que separe las dos clases, y para ello buscaríamos valores de β_0 y de β de forma que $x_i^T\beta + \beta_0 > 0$ para todas las observaciones de un grupo (que etiquetaremos como +1) y $x_i^T\beta + \beta_0 < 0$ para todos los puntos del otro grupo (que etiquetaremos como -1). Una vez conocido el hiperplano separador, para cualquier nueva observación $\hat{x}$, tendríamos una función de clasificación h tal que

$$h(\hat{x}) = \begin{cases} +1 \quad si \quad \beta_0 + \beta^T\hat{x} > 0 \\ -1 \quad si \quad x\beta_0 + \beta^T\hat{x} < 0 \end{cases} \tag{3.2}$$

El objetivo principal del algoritmo es encontrar el hiperplano que minimice el número de puntos mal clasificados y para ello buscamos minimizar el sumatorio de la ecuación 3.3, ya que si la analizamos, se puede observar como tomando x_i mal clasificados, se tiene que $y_i(x_i^T\beta + \beta_0)$ siempre será negativo porque en el caso en el que y_i sea -1, entonces $x_i^T\beta + \beta_0$ será 1 y a la inversa. Es por esto que el sumatorio de la ecuación 3.3 siempre será negativo. Por tanto, se añade el signo negativo delante del sumatorio para obtener el lado derecho de la igualdad positivo. La expresión a minimizar es

$$D(\beta, \beta_0) = -\sum_{i \in M} y_i(x_i^T\beta + \beta_0) \tag{3.3}$$

donde M es el conjunto de puntos mal clasificados.

Para minimizar la expresión 3.3 podemos:

1. Iniciar el proceso con un hiperplano definido por unos valores iniciales β y β_0 aleatorios.
2. Clasificar todas las observaciones acorde a este hiperplano con la función definida $h(x_i)$.

3. Ir visitando las observaciones mal clasificadas, siguiendo algún criterio, de forma que en cada paso i se actualiza el hiperplano siguiendo

$$\begin{pmatrix} \beta \\ \beta_0 \end{pmatrix} \leftarrow \begin{pmatrix} \beta \\ \beta_0 \end{pmatrix} + \rho \begin{pmatrix} y_i x_i \\ y_i \end{pmatrix}$$

siendo ρ la tasa de aprendizaje.

Notar que estamos actualizando en la dirección del gradiente:

$$\frac{\partial D(\beta, \beta_0)}{\partial \beta} = -\sum_{i \in M} y_i x_i \tag{3.4}$$

$$\frac{\partial D(\beta, \beta_0)}{\partial \beta_0} = -\sum_{i \in M} y_i \tag{3.5}$$

4. Repetir los pasos 2 y 3 hasta que todas las observaciones estén bien clasificadas.

Normalmente, para este paso se utiliza el algoritmo del *gradiente descendente* detallado en el Anexo A.1.

Por último, cabe destacar que existen varios problemas al respecto de este algoritmo:

- Cuando los datos son separables, existen infinitas soluciones.
- El número 'finito' de pasos del algoritmo iterativo puede ser muy alto.

3.3. Hiperplanos de separación óptimos

Dadas dos clases distintas, este método tiene el propósito de separarlas y de maximizar la distancia al hiperplano separador del punto más cercano de cada clase. A dicha distancia se le llamará margen y se puede apreciar en el segundo gráfico de la figura 3.3 representado por M.

Tal y como se puede observar en el primer gráfico de la figura 3.3, las dos clases son separables y hay infinitos hiperplanos separadores, en este caso se busca el que maximice la distancia a los puntos más cercanos de las dos clases.

Hemos visto en la ecuación 3.1 que la distancia de un punto x a un hiperplano de ecuación $\beta_0 + \beta^T x = 0$ es

$$d = \frac{\beta^T x + \beta_0}{\sqrt{\beta^T \beta}} = \frac{\beta^T x + \beta_0}{\|\beta\|}$$

Queremos que esa distancia sea mayor que M en todos los casos y que las observaciones estén bien clasificadas. Si imponemos que $\|\beta\| = 1$, el problema a optimizar será

$$\begin{aligned} \underset{\beta,\beta_0,\|\beta\|=1}{\text{máx}} \quad & M \\ \text{sujeto} \quad \text{a} \quad & y_i(x_i^T\beta + \beta_0) \geq M, \quad i = 1, ..., n. \end{aligned} \tag{3.6}$$

siendo M la distancia mínima que hay entre cualquier punto al hiperplano.

La restricción $y_i(x_i^T\beta+\beta_0) \geq M$ significa que el punto está bien clasificado y que la distancia entre el punto y el hiperplano será mayor que M.

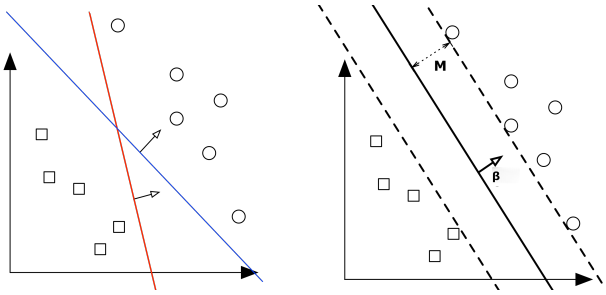


Figura 3.3: Ejemplo de internet que muestra la distancia del punto más cercano de cada clase al hiperplano.

Si queremos relajar la restricción $\|\beta\| = 1$, el problema se formulará como:

$$\begin{aligned} \underset{\beta,\beta_0}{\text{máx}} \quad & M \\ \text{sujeto} \quad \text{a} \quad & \frac{1}{\|\beta\|} y_i(x_i^T\beta + \beta_0) \geq M, \quad i = 1, ..., n. \end{aligned}$$

O lo que es lo mismo

$$\begin{aligned} \underset{\beta,\beta_0}{\text{máx}} \quad & M \\ \text{sujeto} \quad \text{a} \quad & y_i(x_i^T\beta + \beta_0) \geq M\|\beta\|, \quad i = 1, ..., n. \end{aligned}$$

Ahora para toda β y β_0 que satisfagan las desigualdades, se puede tomar $\|\beta\| = 1/M$. Y por tanto, queda el problema de optimización siguiente:

$$\begin{aligned} \underset{\beta,\beta_0}{\text{mín}} \quad & \frac{1}{2}\|\beta\|^2 \\ \textit{sujeto} \quad \textit{a} \quad & y_i(x_i^T\beta + \beta_0) \geq 1, \quad i = 1, ..., n. \end{aligned} \tag{3.7}$$

Lo que queda es un problema de optimización convexo, donde hay una función objetivo cuadrática con n restricciones de desigualdad lineales. Para resolver este nuevo problema de optimización se va a utilizar el método de los multiplicadores de Lagrange, y la función de Lagrange que tiene que ser minimizada con respecto de α_i, β y β_0 es

$$L_P = \frac{1}{2}\|\beta\|^2 - \sum_{i=1}^{n} \alpha_i [y_i(x_i^T \beta + \beta_0) - 1] \tag{3.8}$$

En el Anexo A.2 se desarrolla de forma extendida los multiplicadores de Lagrange y las condiciones de Karush-Kuhn-Tucker.

En este anexo, vemos que dado un problema de minimización con restricciones de igualdad y de desigualdad del tipo

$$\begin{array}{lll} (PI)\ min & f(x_1, ..., x_n) & \\ sujeto \quad a & g_i(x_1, ..., x_n) \leq 0 & \forall i = 1, ..., m \\ & h_j(x_1, ..., x_n) = 0 & \forall j = 1, ..., l \end{array}$$

se define su función lagrangiana como (Eq:A.9):

$$L(x, \alpha_1, ..., \alpha_m, \lambda_1, ..., \lambda_l) = f(x) + \sum_{i=1}^{m} \alpha_i g_i(x) + \sum_{j=1}^{l} \lambda_j h_j(x)$$

La función dual se define como

$$g(\alpha, \lambda) = inf_{x \in D} L(x, \alpha, \lambda).$$

y se define el problema dual como:

$$\begin{array}{ll} (PI)\ max & g(\alpha, \lambda) \\ sujeto \quad a & \alpha \geq 0 \end{array}$$

En este caso se puede demostrar que resolver el problema inicial es equivalente a resolver su problema dual que a menudo es menos exigente del problema original (PI).

En nuestro caso, el problema que queremos minimizar es

$$\begin{aligned} \min_{\beta,\beta_0} \quad & \frac{1}{2}\|\beta\|^2 \\ sujeto \quad a \quad & y_i(x_i^T\beta + \beta_0) \geq 1, \quad i = 1, ..., n. \end{aligned}$$

cuya función lagrangiana es:

$$L_P = \frac{1}{2}\|\beta\|^2 - \sum_{i=1}^{n} \alpha_i[y_i(x_i^T\beta + \beta_0) - 1] \tag{3.9}$$

En el Anexo A.2 por las condiciones de Karush-Kuhn-Tucker (Eq. A.10, Eq.A.11, Eq.A.12 y Eq.A.13) que incluye que el producto de las restricciones de desigualdad en el óptimo por los multiplicadores, son 0 (Eq. A.12). En este caso:

$$\alpha_i[y_i(x_i^T\beta + \beta_0) - 1] = 0 \quad \forall i \tag{3.10}$$

y además $\alpha_i \geq 0 \ \forall i$ (Eq.A.13)

Analizando estas condiciones de K-K-T, se obtiene que

- si $\alpha_i > 0$, entonces $y_i(x_i^T\beta + \beta_0) = 1$, es decir, x_i está en la frontera.
- si $y_i(x_i^T\beta + \beta_0) > 1$, entonces x_i no está en la frontera y $\alpha_i = 0$.

Además, de 3.11 veremos como el vector β se define como una combinación lineal de los puntos soporte x_i, los puntos definidos en la frontera donde $\alpha_i > 0$. Y finalmente, β_0 se puede obtener resolviendo 3.10.

Pero no es fácil encontrar el óptimo de este problema, por lo que vamos a plantear su problema dual.

Para definir la función dual necesitamos el ínfimo respecto a β y a β_0, por lo que:

$$\frac{\partial L_p}{\partial \beta} = 0 \longrightarrow \beta = \sum_{i=1}^{n} \alpha_i y_i x_i \tag{3.11}$$

$$\frac{\partial L_p}{\partial \beta_0} = 0 \longrightarrow \sum_{i=1}^{n} \alpha_i y_i = 0$$

y la función dual quedará

$$g(\alpha) := L_D = \frac{1}{2}(\sum_{i=1}^{n} \alpha_i x_i y_i)^T (\sum_{i=1}^{n} \alpha_i x_i y_i) - \sum_{i=1}^{n} \alpha_i (y_i (x_i^T (\sum_{i=1}^{n} \alpha_i x_i y_i) + \beta_0) - 1) = \tag{3.12}$$
$$= \frac{1}{2}(\sum_{i=1}^{n} \alpha_i x_i y_i)^T (\sum_{i=1}^{n} \alpha_i x_i y_i) + \sum_{i=1}^{n} \alpha_i - (\sum_{i=1}^{n} \alpha_i y_i x_i^T)(\sum_{i=1}^{n} \alpha_i x_i y_i) - \beta_0 \sum_{i=1}^{n} \alpha_i y_i =$$
$$= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{k=1}^{n} \alpha_i \alpha_k y_i y_k x_i^T x_k$$

y el problema dual queda como el siguiente problema de optimización

$$max \qquad L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{k=1}^{n} \alpha_i \alpha_k y_i y_k x_i^T x_k \tag{3.13}$$
$$sujeto \quad a \qquad \alpha_i \geq 0, \tag{3.14}$$

Se ha conseguido obtener una expresión (L_D) que solo involucra los multiplicadores de Lagrange, a diferencia de L_P que involucra además de los multiplicadores de Lagrange, los parámetros del hiperplano, β y β_0.

Ahora la solución se obtiene maximizando L_D en el ortante positivo.

Capítulo 4

Support Vector Machine

4.1. Introducción

Las máquinas de vectores de soporte o máquinas de vector soporte (del inglés support-vector machines, SVM) son un conjunto de algoritmos de aprendizaje supervisado desarrollados por Vladimir Vapnik y su equipo en los laboratorios de AT&T Bell [12].

El Support Vector Machine (SVM) es un algoritmo de clasificación supervisado que consiste en buscar un hiperplano que separe de la mejor forma posible varias clases de datos, que eventualmente han podido ser previamente proyectados a un espacio de dimensionalidad superior. A diferencia de los hiperplanos separadores óptimos vistos en la sección anterior, ahora normalmente no se tienen clases que se pueden separar con un hiperplano lineal, sino que será necesario proyectar los datos a otro espacio donde las clases ya sean separables mediante hiperplanos.

Como ejemplo, en la figura 4.1 existen dos clases que visualmente se puede apreciar como son imposibles de separar por un hiperplano lineal. En estos casos se usará el truco de pasar del espacio vectorial inicial, llamado espacio original, que en este caso es $\mathbb{R}^2$, a uno nuevo de dimensión superior y que llamaremos espacio de características. La finalidad de esta transformación es conseguir que en el espacio de características las clases sean linealmente separables y se pueda encontrar un hiperplano separador.

Si denotamos el espacio original por $X = \mathbb{R}^p$, la transformación que nos permite pasar a un nuevo espacio donde los datos esperamos que sean separables por un hiperplano, es una función $\varphi : X \to H$ donde H es el nuevo espacio de características. En aprendizaje automático la aplicación φ se llama *feature map* (aplicación característica). Esta aplicación nos trasladará los puntos del espacio original al nuevo espacio de características. En este nuevo espacio, H,

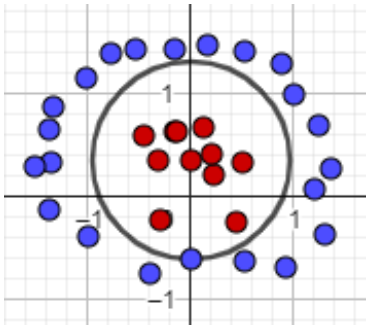


Figura 4.1: Ejemplo de dos clases no linealmente separables.

tenemos que las etiquetas de las observaciones (y_i) no han cambiado, pero cada x_i ahora es $\varphi(x_i)$.

Por tanto, el objetivo actual es encontrar un hiperplano separador que sea capaz de clasificar ambas dos clases en el espacio de características. Este hiperplano será lineal en el espacio de características y no lineal en el espacio de entrada. El problema tiene un planteamiento igual que en hiperplanos separadores óptimos, donde recordamos que llegábamos al problema de optimización definido por la ecuación 3.13, que ahora será:

$$\begin{aligned} max \quad & L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{k=1}^{n} \alpha_i \alpha_k y_i y_k < \varphi(x_i)^T, \varphi(x_k) >_H \\ sujeto \quad a \quad & \alpha_i \geq 0, \\ & \sum \alpha_i y_i = 0; i = 1, ..., n. \end{aligned} \tag{4.1}$$

Para poder definir esta función L_D y encontrar el óptimo al problema de optimización convexa, necesitamos que el espacio de características tenga definido un producto escalar $< \cdot, \cdot >_H$, por lo que recurrimos a los espacios de Hilbert.

Veamos esto con un ejemplo obtenido de [8] con el código correspondiente en Python.

Ejemplo 1 *Consideramos los datos siguientes en forma de dos círculos concéntricos y que se pueden visualizar en la figura 4.2:*

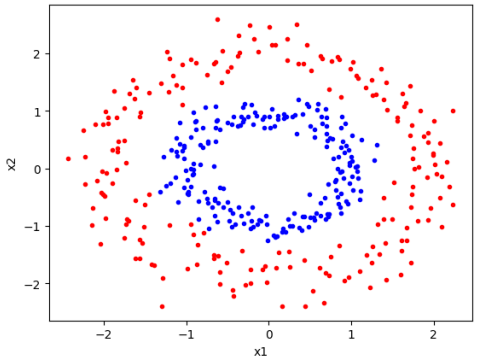


Figura 4.2: Conjunto de observaciones en el espacio original, $\mathbb{R}^2$. Ejecución de la función plot.show().

Código para generar las observaciones en el espacio original

```
N = 200                             # 200 points per class
X = np.zeros(shape=(N*2, 3))        # dummy dataset
Y = np.repeat([0, 1], repeats=N)  # true class labels

radius0 = np.random.normal(loc=1, scale=0.15, size=N)
theta0 = np.linspace(start=0, stop=360, num=N)
radius1 = np.random.normal(loc=2, scale=0.25, size=N)
theta1 = np.linspace(start=0, stop=360, num=N)
radius = np.concatenate([radius0, radius1])
theta = np.concatenate([theta0, theta1])
X[:, 0] = radius * np.cos(theta)   # x1-coordinate
X[:, 1] = radius * np.sin(theta)   # x2-coordinate

plt.plot(X[Y==0, 0], X[Y==0, 1], 'bo', markersize=3)
plt.plot(X[Y==1, 0], X[Y==1, 1], 'ro', markersize=3)
plt.xlabel('x1'); plt.ylabel('x2')
plt.show()
```

Es fácil apreciar en la figura 4.2 como no se puede encontrar un hiperplano separador para las dos clases. Por tanto, consideramos transformar los datos del espacio de dos dimensiones a un espacio de tres dimensiones utilizando la función $\varphi : \mathbb{R}^2 \to \mathbb{R}^3$ *definida como*

$$\varphi(x_i) = [x_{i,1} \quad x_{i,2} \quad x_{i,1}^2 + x_{i,2}^2]^T; \quad \forall x_i = (x_{i1}, x_{i2}) \in \mathbb{R}^2 \tag{4.2}$$

Si representamos ahora las coordenadas de los puntos $\varphi(x_i)$, como se puede observar en la figura 4.3, añadiendo una tercera coordenada se ha conseguido que que las dos clases sean separables en el espacio de características y por tanto, se puede encontrar el hiperplano separador que separa ambas clases.

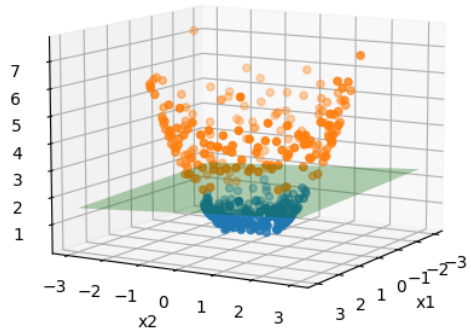


Figura 4.3: Conjunto de observaciones en el espacio de características, $\mathbb{R}^3$. Representación del hiperplano separador.

Código para transformar las observaciones al nuevo espacio de características

```
X[:, 2] = np.square(X[:, 0]) + np.square(X[:, 1])    # create x3-coordinate
clf = svm.SVC(kernel='linear')
clf.fit(X, Y)        # fit SVM and store model weights
coef, intercept = clf.coef_[0], clf.intercept_

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X[Y==0, 0], X[Y==0, 1], X[Y==0, 2], 'b')
ax.scatter(X[Y==1, 0], X[Y==1, 1], X[Y==1, 2], 'r')
xx, yy = np.meshgrid(range(-3, 4), range(-3, 4))
z = (-coef[0]*xx - coef[1]*yy - intercept)/coef[2]
ax.plot_surface(xx, yy, z, color='green', alpha=0.3)

ax.view_init(10, 30)
```

```
ax.set_xlabel('x1'); ax.set_ylabel('x2'); ax.set_zlabel('x3')
plt.show()
```

4.2. Definiciones

En esta sección se van mostrar una serie de definiciones partiendo de aspectos básicos sobre espacios vectoriales para llegar hasta los espacios vectoriales de Hilbert.

Definición 1 *Sea V un espacio vectorial sobre $\mathbb{R}$. Una norma sobre V es una función*

$$\|.\| : V \to \mathbb{R}$$

tal que para cualesquiera dos vectores $v, w \in V$ y escalar $\alpha \in \mathbb{R}$,

- $\|v\| \geq 0$ *de forma que* $\|v\| = 0$ *si y solo si* $v = 0$
- $\|\alpha v\| = |\alpha|\|v\|$
- $\|v + w\| \leq \|v\| + \|w\|$

Definición 2 *Sea V un espacio vectorial equipado con una norma $\|.\|$. Se dice que V es completo con respecto a $\|.\|$ si cada sucesión de Cauchy en V converge a un vector $v \in V$.*

Definición 3 *Sea V un espacio vectorial sobre $\mathbb{R}$. Un producto interior sobre V es una función $< ., . >: V \times V \to \mathbb{R}$ tal que para cualesquiera vectores $u, v, w \in V$ y escalares $\alpha, \beta \in \mathbb{R}$, se tienen las siguientes propiedades:*

- $< u, v >=< v, u >$ *(simetría)*
- $< \alpha u + \beta v, w >= \alpha < u, w > +\beta < v, w >$ *(linealidad en el primer argumento)*
- $< v, v >\geq 0$ *donde* $< v, v >= 0$ *si y solo si* $v = 0$ *(semidefinición positiva)*

Además, cabe destacar que un espacio vectorial equipado con un producto interior es un espacio vectorial interior.

Definición 4 *Sea V un espacio vectorial equipado con un producto interior $< ., . >: V \times V \to \mathbb{R}$. Para $v \in V$, se define la norma inducida por el producto interior como*

$$\|v\| = \sqrt{< v, v >}.$$

Finalmente, tras todas las definiciones anteriores, la última de las definiciones es la del espacio de Hilbert.

Definición 5 *Un espacio de Hilbert es un espacio vectorial interior que es completo con respecto a la norma inducida por el producto interior. El espacio de Hilbert se denota por H.*

4.3. Núcleos reproductores en espacios de Hilbert

Tal y como se ha introducido en la sección anterior, los espacios de Hilbert son un aspecto importante a tratar para poder entender correctamente el algoritmo del Support Vector Machine, es por esto que en esta sección se va a tratar de explicar de forma más detallada un tipo particular de espacios de Hilbert, que son los espacios de Hilbert con núcleo reproductor.

En la siguiente definición 6, se introducen los funcionales de evaluación, pero para poder entenderlo bien, cabe recordar que un funcional lineal en V es una aplicación o transformación lineal del espacio vectorial V en su cuerpo de escalares.

Definición 6 *Sea X un conjunto arbitrario, y H el espacio de Hilbert de todas las funciones $f : X \to \mathbb{R}$. Para cada elemento $x \in X$, el* ***funcional de evaluación*** *es un funcional lineal que evalúa cada $f \in H$ en el punto x, escrito*

$$L_x : H \to \mathbb{R}, \text{ donde } L_x(f) = f(x) \quad \forall f \in H.$$

Se dice que H es ***un espacio de Hilbert con núcleo reproductor (RKHS)*** *si para cualquier $x \in X$, L_x es continuo para todas las $f \in H$.*

En la definición anterior y siguiendo el razonamiento de aprendizaje automático, X es el espacio original, luego es un conjunto que contiene los vectores de atributos que pueden ser números reales, imágenes, cadenas, etc. La función f, asigna a cada $x \in X$ un número real. El funcional de evaluación L_x, es un operador que se utiliza de forma que dado $x \in X$, se puede tomar cualquier función $f \in H$ y evaluar dicha f en x para producir un número real.

El siguiente teorema es el teorema de representación de Riesz que únicamente está enunciado y se puede encontrar su demostración en [6].

Teorema 1 *Dado H un espacio de Hilbert con producto interior y norma, y dado un funcional lineal $L : H \to \mathbb{R}$ continuo en cada $f \in H$, entonces $\exists! k \in H \quad / \quad \forall f \in H$:*

$$L(f) =< f, k >_H$$

En particular:

Corolario 1 *Sea x, X, f, H y L_x definidas como en la definición 6. Si cada L_x es continuo en cada $f \in H$, entonces $\forall L_x, \quad \exists! k_x \in H \quad / \quad \forall f \in H$:*

$$L_x(f) = f(x) =< f, k_x >_H$$

Y esta ecuación se conoce como propiedad reproductiva.

El corolario 1 afirma que dado un funcional de evaluación L_x, se puede encontrar una única función $k_x \in H$ tal que puede representar la acción de L_x sobre cada función f utilizando un producto interior entre f y k_x. Pero en particular, como $k_x \in H$ entonces,

$$L_z(k_x) = k_x(z) =< k_x, k_z >_H \tag{4.3}$$

Definición 7 *Sea X un conjunto arbitrario, y H un espacio de Hilbert de todas las funciones $f : X \to R$. Si $\forall x \in X$, el funcional de evaluación lineal $L_x : H \to R$ es continuo en todo $f \in H$, se puede definir una función bivariante $K : X \times X \to R$ como*

$$K(x, z) =< k_x, k_z >_H$$

*a la que llamaremos **núcleo reproductor**, o función núcleo para abreviar en ocasiones. Al espacio de Hilbert H, se le llama espacio de Hilbert con núcleo reproductor (RKHS).*

4.4. Métodos con núcleos

Como hemos visto en la sección 4.1, para poder realizar la clasificación de datos que inicialmente no son separables por hiperplanos, será necesario transformar el espacio vectorial de entrada en un nuevo espacio de dimensión superior. En esta sección se va a desarrollar el material necesario para entender cómo pueden contribuir a ello las funciones núcleo.

En primer lugar, tal y como se ha definido en la definición 7 los núcleos K son unas funciones bivariantes. Además, como se ha introducido en la sección 4.1, será necesaria la función

$$\varphi : X \to H$$

para realizar la transformación del espacio original al espacio de características. En particular, podemos definir φ como:

$$\varphi(x) = k_x \quad \forall x \in X \tag{4.4}$$

y podemos garantizar que esta función está bien definida gracias al corolario 1.

Así pues, utilizando la definición de φ y la definición 7, reemplazando k_x por $\varphi(x)$ y k_z por $\varphi(z)$ se obtiene la expresión

$$K(x, z) =< k_x, k_z >_H =< \varphi(x), \varphi(z) >_H . \tag{4.5}$$

para el núcleo reproductor.

Definición 8 *Sea X un conjunto arbitrario no vacío. Una función $K : X \times X \to R$ se denomina núcleo definido positivo (d.p.) en X si:*

- *la función K es simétrica*
- *$\sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j K(x_i, x_j) \geq 0$ se cumple $\forall x_1, \ldots, x_n \in \mathcal{X}$, $n \in \mathbb{N}, a_1, \ldots, a_n \in \mathbb{R}$.*

Esto es equivalente a exigir que cualquier matriz finita construida mediante evaluación por pares, $\mathbf{K}_{ij} = K(x_i, x_j)$, tenga autovalores completamente positivos (d.p.) o no negativos (s.d.p.).

Propiedad 1 *Las siguientes funciones bivariantes son núcleos definidos positivos:*

1. *Lineal: $K(x, z) = x^T z$*
2. *Polinómico: $K(x, z) = (x^T z + c)^d$, $c \geq 0$, $d \geq 1$*
3. *Radial o Gaussiano: $K(x, z) = exp(-\gamma \|x - z\|^2)$, $\gamma \in \mathbb{R}^+$*
4. *Sigmoide: $K(x, z) = tanh(\gamma x^T z + c)$, $c \geq 0, \gamma \in \mathbb{R}^+$*

Demostración:

Vamos a demostrar que el núcleo lineal es semidefinido positivo y por tanto, núcleo reproductor.

El núcleo lineal es un núcleo definido positivo:

- *veamos que K lineal es simétrico. Sean* $x, z \in X$

$$K(x,z) = x^T z =< x, z >=< z, x >= z^T x = K(z,x) \tag{4.6}$$

- *veamos que* $\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j K(x_i, x_j) \geq 0$, $\forall x_1, \ldots, x_n \in \mathcal{X}$, $n \in \mathbb{N}, a_1, \ldots, a_n \in \mathbb{R}$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j K(x_i, x_j) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j x_i^T x_j == \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{p} a_i a_j x_{ik}^T x_{jk} = \tag{4.7}$$
$$= \sum_{k=1}^{p}(\sum_{i=1}^{n} a_i x_{ik})(\sum_{j=1}^{n} a_j x_{jk}) = \sum_{k=1}^{p}(a x_k^T)(a x_k^T) = \sum_{k=1}^{p}(a x_k^T)^2 \geq 0$$

donde $a = (a_1, \cdots, a_n)$ *y* $x_i = (x_{i,1}, \cdots, x_{i,p})$

Además del núcleo lineal, el resto de núcleos también cumplen dicha propiedad y por tanto son núcleos reproductores.

El significado de que un núcleo sea semidefinido positivo nos lo da el teorema 2 de Moore-Aronszajn [6] que dice que todo núcleo semidefinido positivo es un núcleo reproductor para algún espacio de Hilbert con núcleo reproductor (RKHS). Este resultado nos será útil a continuación, ya que necesitaremos conocer algún núcleo reproductor para poder resolver el problema de optimización, de forma que conociendo un núcleo semidefinido positivo, tendremos un núcleo reproductor y, por tanto, lo podremos aplicar en el método estudiado.

Teorema 2 *(Moore-Aronszajn theorem). Dado un núcleo* $K(.,.)$ *semidefinido positivo, existe un único espacio de Hilbert* $H(K)$ *de funciones para el que* $K(.,.)$ *es un núcleo reproductor.*

4.5. Support Vector Machine

Cabe recordar que el Support Vector Machine (SVM) es un algoritmo de aprendizaje supervisado que tiene como objetivo clasificar datos mediante hiperplanos separadores, en un espacio que puede no ser el espacio original. Para ello, tal y como se ha adelantado en la sección 4.1, tendríamos que buscar una aplicación $\varphi()$ para transformar el espacio de origen X a un nuevo espacio de características H, que será un espacio de Hilbert dotado de producto interior. Nuestro objetivo entonces sería buscar un hiperplano separador en H que tendrá una estructura no lineal en X.

Una vez tenemos las observaciones en un espacio donde son linealmente separables, el problema tiene un planteamiento igual que en hiperplanos separadores óptimos, donde recordamos que llegábamos a la ecuación de Lagrange Dual siguiente

$$L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{k=1}^{n} \alpha_i \alpha_k y_i y_k < x_i^T, x_k >_H . \tag{4.8}$$

Ahora estamos en el espacio de características donde cada x_i de 4.8, es $\varphi(x_i)$ por lo que tenemos la nueva ecuación de Lagrange Dual del espacio de características con el nuevo producto escalar

$$L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{k=1}^{n} \alpha_i \alpha_k y_i y_k < \varphi(x_i)^T, \varphi(x_k) > \tag{4.9}$$

y por la definición de φ, y la de función núcleo reproductor y la relación entre ellas (Ec. 4.5), la función de Lagrange Dual queda

$$L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{k=1}^{n} \alpha_i \alpha_k y_i y_k K(x_i^T, x_k) \tag{4.10}$$

Entonces, el problema de optimización a resolver es

$$\begin{aligned} max \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{k=1}^{n} \alpha_i \alpha_k y_i y_k K(x_i^T, x_k) \\ sujeto \quad & a \quad \alpha_i \geq 0, \quad \sum \alpha_i y_i = 0 \quad i = 1, ..., n. \end{aligned} \tag{4.11}$$

De esta forma, tenemos el mismo problema de optimización de Lagrange que en hiperplanos separadores óptimos donde solo interesa saber el resultado del producto interior, por lo que únicamente es necesario saber la definición del núcleo reproductor K para poder resolver el problema. Como nos garantiza el teorema de Moore-Aronszajn(teorema 2) tenemos garantizado que hemos pasado las observaciones a un espacio de Hilbert, sin necesidad de definir la aplicación característica $\varphi(.)$.

En la sección 4.4 ya hemos comentado que son núcleos definidos positivos (y por tanto, núcleos reproductores):

- Lineal: $K(x, z) = x^T z$

- Polinómico: $K(x, z) = (x^T z + c)^d$, $c \geq 0$, $d \geq 1$
- Radial o Gaussiano: $K(x, z) = exp(-\gamma \|x - z\|^2)$, $\gamma \in \mathbb{R}^+$
- Sigmoide: $K(x, z) = tanh(\gamma x^T z + c)$, $c \geq 0, \gamma \in \mathbb{R}^+$

Por tanto, estas funciones serán las que se podrán emplear para ejecutar el algoritmo.

Capítulo 5

Aplicación

5.1. Introducción

Los problemas cardiovasculares o ataques cardíacos suceden cuando el flujo de sangre al corazón se detiene. Según algunas estadísticas de la OMS (Organización Mundial de la Salud), [1], mueren alrededor de 17.9 millones de personas de ataques al corazón. La principal causa es el estilo de vida de los pacientes, pero existen algunos factores que pueden llegar a advertir.

En este capítulo se resuelve un problema de clasificación supervisada donde se busca analizar una base de datos sobre enfermedades de corazón de diferentes pacientes de forma que se pueda predecir si tiene o no dicha enfermedad. Esto podrá ser muy útil a la hora de prevenir enfermedades de corazón en muchos tipos de pacientes diferentes y además, se podría llegar a ampliar a otro tipo de enfermedades.

5.2. Definición de la base de datos

El conjunto de datos que se ha utilizado para poder desarrollar el código se ha obtenido de una plataforma web llamada Kaggle [11] donde se pueden obtener conjuntos de datos de diferentes temas. En este caso, el tema es "Heart Attack". La base de datos consiste en diferentes factores más o menos influyentes en enfermedades de corazón de un total de 270 individuos y un atributo final que indicará si tiene o no la enfermedad.

5.2.1. Atributos

Para cada observación o individuo, se tiene un total de 12 atributos distintos. Estos atributos se componen de 11 atributos que aportan información médica y un atributo que aporta la información de si se tiene o no el problema cardiovascular. A continuación, una explicación de cada uno de ellos:

- **age**: edad del paciente en años.
- **sex**: sexo del paciente (0-1) siendo:
  - 0: femenino
  - 1: masculino
- **cp**: tipo de dolor torácico (1-4) siendo:
  - 1: angina típica
  - 2: angina atípica
  - 3: dolor no anginoso
  - 4:asintomático
- **trestbps**: presión arterial en reposo (en mmHg, en el momento del ingreso).
- **restecg**: resultado del electrocardiográfico en reposo (0-2).
  - 0: normal
  - 1: anomalía en la onda ST-T (inversión de la onda T y/o elevación o depresión del ST $>$ 0,05 mV)
  - 2: Muestra hipertrofia ventricular izquierda probable o definitiva según los criterios de Estes
- **thalach**: frecuencia cardíaca máxima alcanzada.
- **exang**: angina inducida por el ejercicio (0-1) siendo:
  - 0: no
  - 1: si
- **oldpeak**: depresión del ST inducida por el ejercicio frente al descanso.
- **slope**: la pendiente del segmento ST de ejercicio máximo (1-3) siendo:
  - 1: ascendiente

- 2:plano
- 3:descendente

- **ca**: número de vasos mayores teñidos por fluroscopia.
- **thal**: frecuencia cardíaca máxima alcanzada siendo:
  - 7: defecto reversible
  - 6: defecto normal
  - 3: normal
- **target**: tener la enfermedad o no, (0-1) siendo:
  - 1: si que se tiene
  - 0: no se tiene

5.3. Explicación código

Como bien se ha expresado en la sección 5.1, estamos ante un problema de clasificación supervisada y el objetivo es, a la vista de los datos, poder predecir si una serie de individuos tienen o no enfermedades de corazón. En esta memoria, hemos revisado muy por encima los métodos de clasificación supervisada más elementales

- Regresión Logística
- Análisis discriminante Lineal
- k-vecinos más próximos

y hemos explicado con más detalle cómo funciona el método de funciones soporte (SVM).

Para ver cómo funcionan todos estos algoritmos de clasificación en el problema planteado, he empezado realizando un programa en lenguaje Python que desarrolla un algoritmo de clasificación supervisada utilizando el método del Support Vector Machine. Se desarrolla ahora una explicación sobre el código que se ha implementado para poder entenderlo mejor. El código del SVM se puede ver en el Anexo B.1.

En primer lugar, se han realizado las importaciones de una serie de librerías de tratamiento de datos y de Machine Learning. Entre las librerías de tratamiento de datos se encuentran NumPy [4], pandas [9], seaborn [13] y matplotlib [7]. **NumPy** es una librería que permite

tratar grandes volúmenes de datos, **pandas** ofrece una estructura de datos llamada Dataframe con la que se suele trabajar en ciencia de datos, **matplotlib y seaborn** son dos librerías que se utilizan para la representación de gráficos en Python.

Respecto a la librería de Machine Learning, se ha importado **sklearn** [10] junto a algunas de sus funciones básicas que permiten implementar el modelo de SVM. Las funciones utilizadas de esta librería son: **svm** que implementa el modelo de Support Vector Machine, **train_test_split** que se utiliza para poder dividir los datos en dos conjuntos de entrenamiento y de test, **StandardScaler** que permite escalar los datos cuantitativos, **accuracy_score** que obtiene la precisión del modelo en cuestión, **confusion_matrix** que pertime representar las matrices de confusión y **GridSearchCV** para encontrar el núcleo e hiperparámetros óptimos.

El siguiente de los pasos, ha sido la lectura de un CSV donde se encuentran todos los datos, para ello se ha utilizado la función **read_csv()** de pandas. Con la finalidad de visualizar los atributos de los 5 primeros individuos, se puede utilizar la función **head()** de pandas y se aprecia en la figura 5.1 su resultado.

	age	sex	cp	trestbps	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	70	1	4	130	2	109	0	2.4	2	3	3	1
1	67	0	3	115	2	160	0	1.6	2	0	7	0
2	57	1	2	124	0	141	0	0.3	1	0	7	1
3	64	1	4	128	0	105	1	0.2	2	1	7	0
4	74	0	2	120	2	121	1	0.2	1	1	3	0

Figura 5.1: Resultado de la función head() con los datos leídos.

Continuando con el código, se calcula la matriz de correlaciones, para ello, primero se eliminan los atributos cualitativos y se dejan solo los atributos cuantitativos con la función **drop()** de pandas. Una vez eliminados, se visualiza la matriz con un total de 4 atributos que son *age, trestbps, thalach y oldpeak*. El resultado obtenido es el que se aprecia en la figura 5.2. La matriz muestra la correlación entre dos atributos, lo que indica el grado de relación lineal que existe entre ambos, es decir, cuando esta es cercana a 0 nos dará más información para el modelo debido a que tienen poca relación lineal entre ellos. Además de la matriz, se van a visualizar las diferentes distribuciones de los 4 atributos. Se pueden ver dichas distribuciones en la figura 5.3.

En cuanto a las 8 variables cualitativas, se han realizado tablas de frecuencias con la finalidad de analizar la cantidad de veces que aparece cada uno de los valores. Estos atributos categóricos son: *sex, cp, restecg, exang, slope, ca, thal y target*. Se pueden apreciar las tablas de frecuencias creadas en la tabla 5.3.

Antes de empezar con el algoritmo de Machine Learning, se necesita convertir los atributos

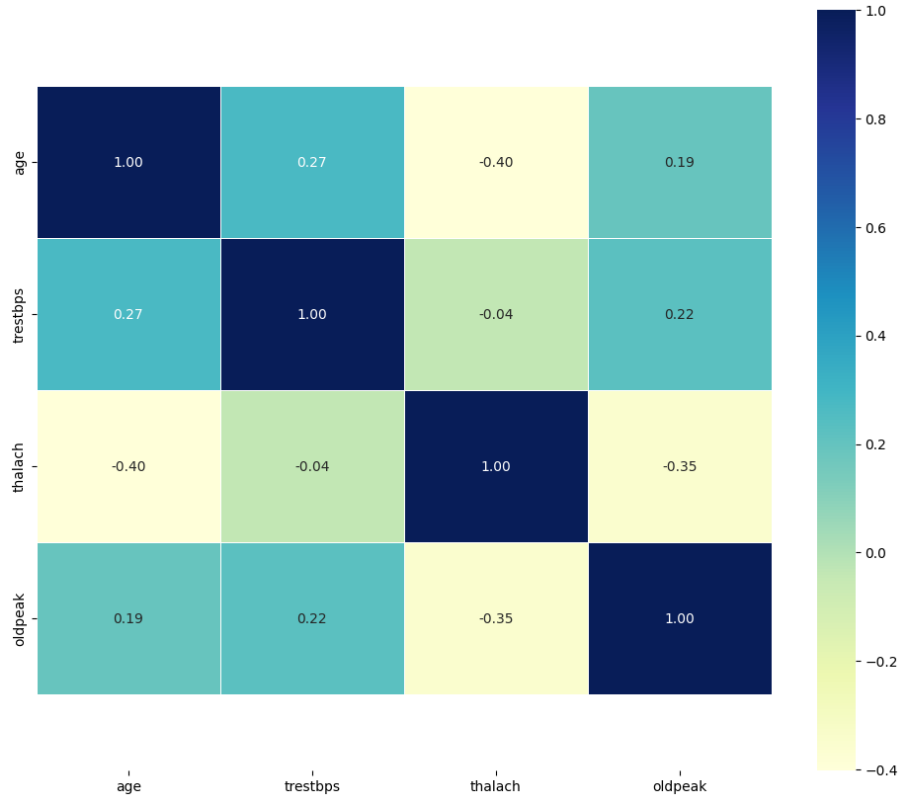


Figura 5.2: Matriz de correlaciones entre atributos cuantitativos.

categóricos en variables dicotómicas para definir etiquetas y escalar las diferentes variables cuantitativas. Para convertir en variables dicotómicas, se descomponen las columnas categóricas en varias columnas de 0 o 1, por ejemplo, el atributo *cp*, que es el tipo de dolor torácico, puede tomar un valor entre 1 y 4 por lo que se divide en 4 columnas distintas poniendo 1 en la columna del tipo que sea el dolor y 0 en las restantes. En cuanto a escalar atributos, se usa la función **StandardScaler** que permite modificar los valores de forma que se elimina la media y toma varianza igual a 1.

Una vez realizados todos estos pasos, ha llegado el momento de identificar las variables. El primer paso es dividir entre input y output, de forma que todos los atributos son inputs, a excepción del *target* que es el output del modelo. El segundo paso es dividir todo el conjunto de datos en dos conjuntos, un primer conjunto de entrenamiento que será del 80 % de los datos y servirá para entrenar el modelo, y un segundo conjunto de test del 20 % con el que se realizarán las pruebas pertinentes para evaluar el modelo.

En cuanto a la implementación del propio modelo, he querido probar con los diferentes

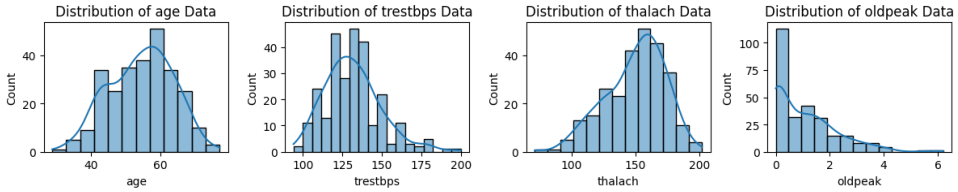


Figura 5.3: Distribuciones de los atributos cuantitativos.

sex	
VALORES	FRECUENCIA
0: mujer	87
1: hombre	183

cp	
VALORES	FRECUENCIA
1: angina típica	20
2: angina atípica	42
3: dolor no anginoso	79
4:asintomático	129

restecg	
VALORES	FRECUENCIA
0: normal	131
1: anomalía en la onda ST-T	2
2: Muestra hipertrofia ventricular izquierda probable o definitiva	137

exang	
VALORES	FRECUENCIA
0: no	181
1: si	89

slope	
VALORES	FRECUENCIA
1: ascendiente	130
2:plano	122
3:descendente	18

ca	
VALORES	FRECUENCIA
0	160
1	58
2	33
3	19

thal	
VALORES	FRECUENCIA
3: normal	152
6: defecto normal	14
7: defecto reversible	104

target	
VALORES	FRECUENCIA
0: no tiene la enfermedad	150
1: si tiene la enfermedad	120

Figura 5.4: Tablas de frecuencias de los atributos categóricos.

núcleos que permite la librería que son el Lineal, Polinómico, Radial o Gaussiano y Sigmoide. Cabe que recordemos las ecuaciones de los núcleos utilizados:

- Lineal: $K(x, z) = x^T z$
- Polinómico: $K(x, z) = (x^T z + c)^d$, $c \geq 0$, $d \geq 1$
- Radial o Gaussiano: $K(x, z) = exp(-\gamma \|x - z\|^2)$, $\gamma \in \mathbb{R}^+$
- Sigmoide: $K(x, z) = tanh(\gamma x^T z + c)$, $c \geq 0, \gamma \in \mathbb{R}^+$

En los núcleos hay parámetros que se deben ajustar para conseguir el mejor resultado en la clasificación. El núcleo polinómico tiene c y d (d será «degree» en el método de Python), el Gaussiano y Sigmoide tienen γ (γ será «gamma» en el método de Python). Estos parámetros se llaman *parámetros de kernel* o *hiperparámetros*.

Se implementa un bucle donde se crea el modelo con la función **svm.SVC(kernel=kernels[i])**, siendo **kernels** un vector con los nombres de los núcleos a utilizar. También se entrena el modelo con la función **fit(X_train, y_train)** y finalmente, se obtiene la precisión del modelo con **accuracy_score(y_test, y_pred_svm[i])*100**. La función **svm.SVC()** asigna valores por defecto a los hiperparámetros siendo $C = 1$, $degree = 3$ y $gamma =' scalar' = 1/(n\_features * X.var())$ donde n_features es el número de atributos de cada observación y X.var() es la varianza.

Tras obtener las precisiones del método con cada núcleo he formado una matriz de confusión para cada núcleo y así poder apreciar de forma visual el número de aciertos y errores. Se pueden ver las matrices de confusión en la figura 5.6.

Gracias a este bucle hemos podido conseguir un valor aproximado de la precisión del modelo, pero como bien he dicho, los núcleos tienen parámetros que se pueden ajustar para obtener un mejor resultado. Es decir, para encontrar el modelo óptimo, debemos ajustar correctamente los hiperparámetros. El propio programador es quien elige estos valores en función de la intuición o de haber realizado distintas pruebas, encontrarlos es una tarea compleja, ya que hay que probar todas las combinaciones posibles para ver cuál funciona mejor. Sin embargo, es posible hacer esto de forma automática utilizando la función **GridsearchCV** de Scikit-learn que toma un diccionario con los posibles valores y realiza todas las combinaciones posibles.

Además, esta función permite la técnica de validación cruzada, que consiste en validar el modelo consiguiendo una estimación más precisa del rendimiento. Las primeras precisiones que hemos obtenido para cada tipo de núcleo dependerán en gran medida de los conjuntos de datos de entrenamiento y test, es decir, cuando cambiemos estos conjuntos, cambiarán las precisiones. Al usar validación cruzada se realizan diferentes pruebas con diferentes conjuntos de entrenamiento y test para, finalmente, mostrar una media de las precisiones obtenidas.

Para la implementación de la función **GridsearchCV**, primero he creado un diccionario con los posibles valores que podrán tomar los hiperparámetros. A continuación, se crea el objeto GridSearchCV pasando como parámetro el clasificador y el diccionario. Posteriormente, se muestra por pantalla los resultados ordenados de mejor a peor en forma de tabla, véase en la figura 5.5.

	C	gamma	degree	kernel	Accuracy
102	1	0.1	2	sigmoid	0.878324
134	1	0.1	4	sigmoid	0.878324
150	1	0.1	5	sigmoid	0.878324
118	1	0.1	3	sigmoid	0.878324
86	1	0.1	1	sigmoid	0.878324
..	...	...	...	...	...
77	0.1	0.001	5	poly	0.576684
76	0.1	0.001	5	rbf	0.576684
74	0.1	0.01	5	sigmoid	0.576684
72	0.1	0.01	5	rbf	0.576684
0	0.1	1	1	rbf	0.576684

Figura 5.5: Resultado ordenado de las pruebas del SVM.

Núcleo Polinómico		
c	grado	Precisión
0.1	1	85,17 %
1	1	84,66 %
0.1	2	84,65 %
0.1	3	84,12 %
1	3	79,01 %

Cuadro 5.1: Núcleo Polinómico

Núcleo Gaussiano (rbf)		
C	gamma	Precisión
100	0.001	86,76 %
1	0.1	86,23 %
1	0.01	85,18 %
100	0.01	84,66 %
1	'sacalar'	82,72 %

Cuadro 5.2: Núcleo Gaussiano (rbf)

Además del código del Support Vector Machine, se han implementado los tres algoritmos de clasificación supervisada introducidos en el capítulo 2. Para poder implementarlos he necesitado importar las funciones **LogisticRegression, LinearDiscriminantAnalysis, KNeighborsClassifier** de sklearn, los códigos de los tres algoritmos se encuentran en los Anexos B.2, B.3 y B.4. Además, también he necesitado la función **cross_val_score** para poder hacer las validaciones cruzadas pertinentes. En cuanto al código completo, coinciden con el código de SVM, a excepción de la parte donde se crea el método que habrá que añadir las funciones específicas de cada método.

Para los tres métodos he creado los clasificadores y he obtenido las precisiones realizando validación cruzada en 4 conjuntos. Esto significa que se han realizado 4 pruebas y el resultado final es la media de las 4 precisiones obtenidas. Además de esto, se han implementado las matrices de confusión de cada uno de los métodos para una de las 4 pruebas realizadas.

Para finalizar con la explicación de los códigos implementados, el último de los pasos ha sido representar mediante un gráfico de barras las precisiones obtenidas con los 4 métodos de clasificación supervisada, figura 5.7, el código se encuentra en el Anexo B.5. En este se representan las precisiones del método SVM utilizando el caso óptimo, de regresión logística, del análisis discriminante lineal y del método de los k-vecinos más próximos.

Núcleo Sigmoide		
C	grado	Precisión
1	0.1	87,83 %
10	0.01	85,17 %
100	0.001	85,17 %
100	0.01	85,17 %
1	'sacalar'	79,01 %

Cuadro 5.3: Núcleo Sigmoide

5.4. Resultado

En esta sección voy a analizar los resultados obtenidos con el método del Support Vector Machine, estudiando las precisiones y las matrices de confusión. La precisión se obtiene comparando las predicciones obtenidas (**y_pred_svm**) a partir del conjunto de test (**X_test**), con las clases reales (**y_test**). Además de la precisión, también se han realizado matrices de confusión donde se puede analizar como se equivoca cada método creado.

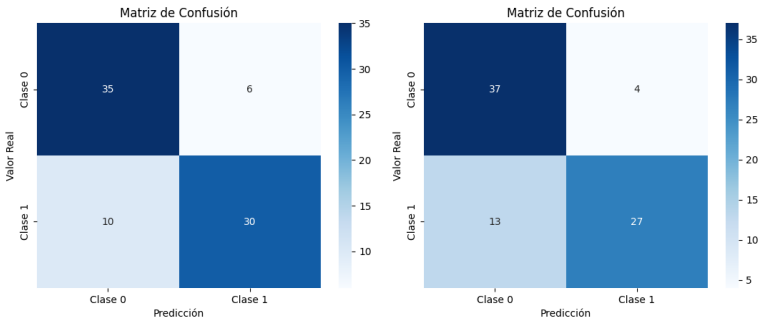
En primer lugar, tal y como se ha detallado en la sección 5.3, se ha creado un modelo por cada uno de los 4 núcleos estudiados, se han entrenado con el conjunto de prueba y finalmente, se han obtenido unas precisiones comparando las predicciones con las clases reales. Estos valores obtenidos son dependientes de la partición realizada entre el conjunto de entrenamiento y el de prueba. Esto se debe a que si cambiamos los datos de entrenamiento, el modelo cambiará y, por tanto, nos dará precisiones diferentes. Además, en esta primera implementación de SVM se está creando y entrenando el modelo con los hiperparámetros de núcleo dados por defecto, los cuales podrían no ser los óptimos.

En esta primera implementación se ha obtenido un 80,25 % de precisión con el Lineal, un 79,01 % con el Polinómico, un 82,72 % con el Radial y un 79,01 % con el Sigmoide. Por tanto, en esta primera prueba, utilizando los hiperparámetros por defecto (gamma= 1 / (n_features * X.var()), C=1 y degree=3), el mejor núcleo ha sido el radial.

Para solucionar el problema anterior y poder tener resultados más precisos, se ha utilizado la validación cruzada probando con diferentes conjuntos y obteniendo una media de varias precisiones. Además, se ha creado un diccionario con los posibles valores de los hiperparámetros C, gamma y degree (grado del núcleo polinómico), de forma que la función prueba todas las combinaciones posibles para, finalmente, obtener el mejor resultado. Tras las pruebas, se ha obtenido el resultado de la figura 5.5. Se puede analizar de forma más visual en las tablas 5.1, 5.2 y 5.3 viendo los resultados ordenados para cada uno de los valores de c, *gamma* y *degree*. En la tabla 5.1 se puede ver el resultado obtenido por defecto con $c = 1$ y $degree = 3$. En la

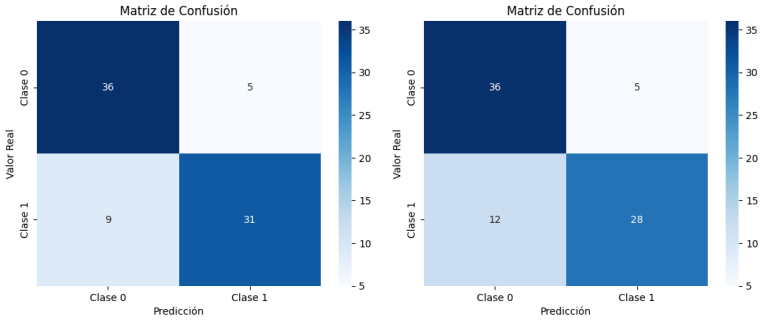
tabla 5.2, se puede apreciar como el mejor resultado para el núcleo Gaussiano es 86,76 % con los parámetros $c = 100$ y $gamma = 0{,}001$ y además, también se puede apreciar el resultado para los valores por defecto. Finalmente, en la tabla 5.3 aparece el resultado óptimo para el núcleo Sigmoide con C=1 y gamma=0.1, lo que ha obtenido un 87,83 % de precisión.

Finalmente, se puede ver en la figura 5.6 las matrices de cada una de las primeras pruebas realizadas, en todas ellas el número de observaciones clasificadas va relacionado con la precisión del núcleo. En las 4 matrices se puede apreciar en la diagonal el número de aciertos siendo en el mejor de los casos 36 verdaderos negativos (Clase 0) y 31 verdaderos positivos (Clase 1).



(a) Núcleo Lineal.

(b) Núcleo Polinómico.



(c) Núcleo Gaussiano.

(d) Núcleo Sigmoide.

Figura 5.6: Matrices de confusión.

5.5. Comparación con otros métodos

Como bien se ha explicado en la sección 5.3, se han implementado los 4 métodos de clasificación supervisada Support Vector Machine, regresión logística, análisis discriminante lineal y k-vecinos más próximos. En esta sección se va a realizar una comparación de los resultados obtenidos con cada uno de los métodos teniendo en cuenta las precisiones y las matrices de

Método k-vecinos más próximos	
k	Precisión
7	84,04 %
3	83,59 %
5	81,50 %

Cuadro 5.4: Método k-vecinos más próximos

confusión.

Para obtener las precisiones de regresión logística, análisis discriminante lineal se ha utilizado validación cruzada con 4 conjuntos por lo que las precisiones obtenidas son una media de las 4 pruebas. Por otro lado, y similar a lo realizado con SVM, para el método de los k-vecinos se ha probado con diferentes valores del parámetro k y a su vez con la validación cruzada obteniendo los resultados mostrados en la tabla 5.4.

El modelo de regresión logística ha obtenido una precisión media del 85,57 %, el análisis discriminante lineal 86,68 %, y el método de los k-vecinos más próximos 84,08 % para el mejor de los casos. Estos resultados se pueden apreciar en forma de gráfico en la figura 5.7.

Además de las precisiones, en la figura 5.8 se puede contemplar las 4 matrices de confusión de cada uno de los métodos obtenidas de una de las 4 pruebas realizadas. Analizando dichas matrices, se puede concluir también que el método que más aciertos tiene es el SVM, ya que ha predicho correctamente 36 negativos (clase 0) y 30 positivos (clase 1). Entre los falsos negativos, SVM solo se ha equivocado en 10 pacientes, a diferencia de regresión logística y ADL que este valor aumenta hasta 11 o en KNN que son 16 los erróneos. Finalmente, los falsos positivos disminuyen en los 4 métodos, siendo 2 para el más óptimo y 5 o 6 para el resto.

Por tanto, comparando los 4 modelos realizados, se puede concluir que SVM con el núcleo Sigmoide y hiperparámetros $C = 1$ y $gamma = 0{,}1$, es el modelo que realiza las mejores predicciones, ya que ha obtenido una mayor precisión y un mayor número de aciertos.

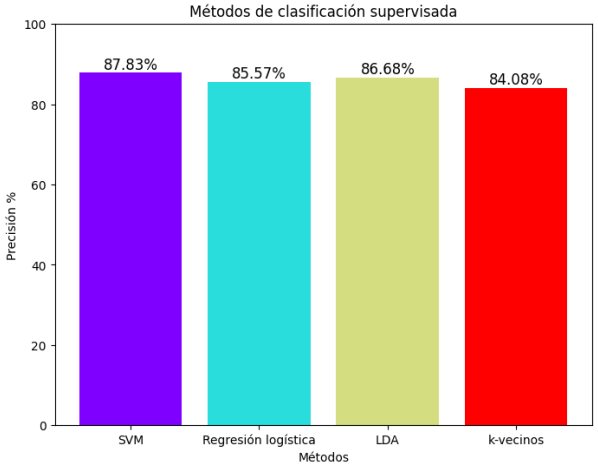
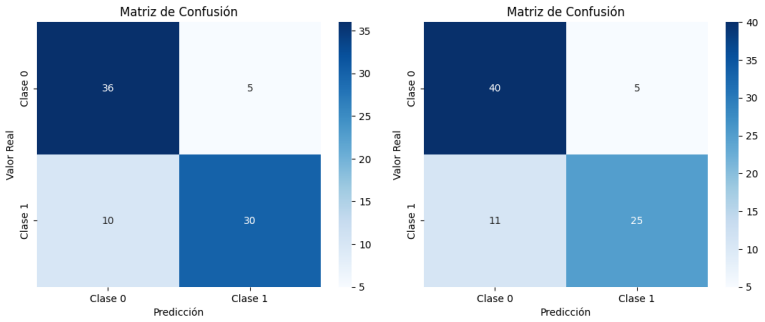
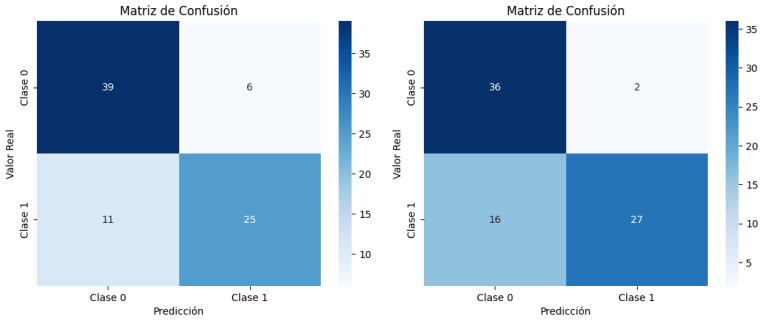


Figura 5.7: Precisión de los diferentes núcleos probados.



(a) Support Vector Machine con núcleo e hiperparámetros óptimos.

(b) Regresión logística.



(c) Análisis discriminante lineal.

(d) Método k-vecinos más próximos.

Figura 5.8: Matrices de confusión.

Capítulo 6

Conclusiones

En conclusión, en este trabajo final de grado he indagado y estudiado el método del Support Vector Machine desde una perspectiva matemática. Además, he implementado un código de Python en la predicción de enfermedades cardiovasculares comparando los diferentes métodos más utilizados de clasificación supervisada.

He analizado los conceptos fundamentales del SVM, partiendo de métodos básicos como son el Perceptrón y los hiperplanos separadores óptimos, hasta conceptos más complejos como son los espacios de Hilbert con núcleo reproductor o el truco del núcleo. Gracias a este análisis, he conseguido un conocimiento sólido sobre los principales fundamentos del SVM.

Otro de los aspectos fundamentales de este trabajo ha sido la importancia en el campo de la medicina, en este caso, en la predicción de enfermedades cardiovasculares. Gracias a esta implementación, se han podido complementar los aspectos puramente teóricos con aspectos más prácticos y reales, utilizando algunas herramientas de programación como Python.

En resumen, gracias a este estudio he adquirido las habilidades necesarias para futuras investigaciones y aplicaciones del aprendizaje automático en campos como el de la medicina, donde se está introduciendo cada vez más, pudiendo tomar en un futuro, un papel fundamental.

Bibliografía

[1] Hipercolesterolemia. https://www.amgen.es/enfermedades-tratadas/hipercolesterolemia/enfermedad-cardiovascular-esp.

[2] Alan Agresti. *Categorical data analysis*, volume 792. John Wiley & Sons, 2012.

[3] James Gareth, Witten Daniela, Hastie Trevor, and Tibshirani Robert. *An introduction to statistical learning: with applications in R*. Spinger, 2013.

[4] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[5] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[6] Tailen Hsing and Randall Eubank. *Theoretical foundations of functional data analysis, with an introduction to linear operators*, volume 997. John Wiley & Sons, 2015.

[7] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[8] Il Shan Ng. Reproducing kernel hilbert spaces & machine learning. https://ngilshie.github.io/jekyll/update/2018/02/01/RKHS.html. [Consulta: 30 de Mayo de 2023].

[9] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[10] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[11] PRIT SHETA. Datos sobre ataque al corazón. `https://www.kaggle.com/datasets/pritsheta/heart-attack?resource=download`.

[12] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 1999.

[13] Michael L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021.

Anexo A

Anexo I

A.1. Algoritmo del gradiente descendiente

El gradiente descendiente es un algoritmo de optimización iterativo de primer orden, que permite encontrar mínimos locales de una función diferenciable. Un concepto importante en este algoritmo es la **tasa de aprendizaje**, es un valor constante que representa el tamaño de los pasos que se van a dar para alcanzar el mínimo. Cuando la tasa es grande, los pasos realizados serán mayores, pero puede tener el riesgo de pasar el mínimo. Cuando la tasa de aprendizaje es menor, se puede conseguir una precisión mayor, pero provoca un mayor número de cálculos y tiempo a la hora de encontrar el mínimo.

Antes de explicar el algoritmo, cabe recordar la definición de gradiente. Dada una función $f : \mathbb{R}^n \to \mathbb{R}^n$ diferenciable, se define su gradiente en un punto $p = (x_1, ..., x_n)$ como

$$\nabla f(p) = (\frac{\partial f(p)}{\partial x_1}, ..., \frac{\partial f(p)}{\partial x_n}) \tag{A.1}$$

Este algoritmo se lleva a cabo con un proceso iterativo que se puede apreciar de manera visual en la figura A.1. El proceso tiene una serie de pasos que son los siguientes:

1. Tomar un punto de partida arbitrario de la función, x_0.
2. Inicializar $i = 0$
3. Calcular el gradiente de la función en ese punto, $\nabla f(x_i)$.

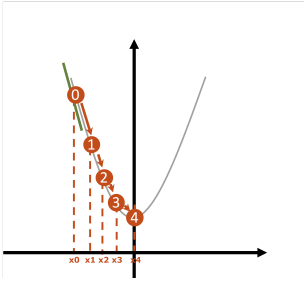


Figura A.1: Ejemplo algoritmo gradiente descenciente.

4. Dar un paso escalado en la dirección contraria al gradiente (minimizar), definiendo el punto siguiente como
$$x_{i+1} = x_i - \alpha \nabla f(x_i), \tag{A.2}$$
donde α es la tasa de aprendizaje.

5. Hacer $i = i+1$ y repetir los dos pasos anteriores hasta que se cumpla uno de los siguientes criterios de parada:

   - Se llega a un número máximo de iteraciones permitidas.
   - Cuando $|x_{i+1} - x_i|$ es menor que un valor de tolerancia establecido.

A.2. Multiplicadores de Lagrange

A.2.1. Problema de programación no lineal con restricciones (conceptos básicos)

Un problema de optimización consiste en buscar el óptimo de una función $f()$, denominada **función objetivo**, que depende de variables $x_1, \cdots, x_n$ denominadas **variables de decisión**. Además, no nos interesará encontrar el máximo o el mínimo de $f()$ en todo $\mathbb{R}^n$, sino restringir la optimización a un dominio menor $F \subseteq \mathbb{R}^n$, denominado **conjunto factible**. Este conjunto F quedará definido por ciertas propiedades o características, que se pueden formular e incorporar al problema de optimización como **restricciones**. En el caso en que tanto la función objetivo como las restricciones sean funciones lineales, diremos que tenemos un problema de optimización lineal con restricciones.

$$\begin{aligned} opt \quad & f : D \subseteq \mathbb{R}^n \to \mathbb{R} \\ sujeto \quad a \quad & x \in F \subseteq \mathbb{R}^n \end{aligned} \tag{A.3}$$

Dependiendo del objetivo de la optimización, existen dos tipos de problemas, los que buscan valores donde la función objetivo alcanza su máximo, problema de maximización, o valores donde la función objetivo alcanza su mínimo, problema de minimización. En ambos casos, los problemas podrán estar sujetos a restricciones o no estarlo.

Las restricciones que se aplican a las variables de la función objetivo pueden ser igualdades o desigualdades:

$$\begin{aligned} & opt && f(x_1, ..., x_n) \\ sujeto \quad & a && g_i(x_1, ..., x_n) = b_i \ \forall i = 1, ..., r \\ & && h_i(x_1, ..., x_n) \leq b_i \ \forall i = r+1, ..., k \\ & && p_i(x_1, ..., x_n) \geq b_i \ \forall i = k+1, ..., m \end{aligned} \tag{A.4}$$

A.2.2. Multiplicadores de Lagrange

El método de los multiplicadores de Lagrange, es un método que tiene la finalidad de encontrar máximos y mínimos relativos de funciones de múltiples variables sujetas a una o varias restricciones de igualdad.

$$\begin{aligned} & opt && f(x_1, ..., x_n) \\ sujeto \quad & a && g_i(x_1, ..., x_n) = b_i \quad \forall i = 1, ..., r \end{aligned}$$

Este método reduce un problema de optimización con n variables y r restricciones a un problema de $n+r$ variables sin restricciones. Para ello, se define la llamada función lagrangiana, considerando de todas las restricciones, los términos $g_i(x_1, ..., x_n) - b_i$ para $i = 1, ..., r$, y además, añadiendo una nueva variable, α_i, llamada multiplicador, por cada una de las restricciones. La función lagrangiana se define en la definición 9.

Definición 9 *Sean $f, g_i : D \subseteq \mathbb{R}^n \to \mathbb{R}$ de clase C^1 en $int(D)$, $b_i \in \mathbb{R}$, $\forall i = 1, \cdots, r$. Dado el problema:*

$$\begin{aligned} & opt && f(x_1, ..., x_n) \\ sujeto \quad & a && g_i(x_1, ..., x_n) = b_i \quad \forall i = 1, ..., r \end{aligned} \tag{A.5}$$

siendo todas las restricciones de igualdad. La función lagrangiana asociada a dicho problema es:

$$L(x_1, ..., x_n, \alpha_1, ..., \alpha_r) = f(x_1, ..., x_n) - \sum_{i=1}^{r} \alpha_i [g_i(x_1, ..., x_n) - b_i] \tag{A.6}$$

Una vez creada la función lagrangiana y ya sin restricciones, el problema consiste en encontrar los puntos óptimos de dicha función (puntos críticos), para lo que es necesario estudiar cuando se anula el gradiente. Si nos fijamos únicamente en las variables originales de la función $(x_1, x_2, ..., x_n)$:

$$\begin{pmatrix} \frac{\partial L}{\partial x_1}(x) \\ . \\ . \\ . \\ \frac{\partial L}{\partial x_n}(x) \end{pmatrix} = \begin{pmatrix} \frac{\partial f}{\partial x_1}(x) - \alpha_1 \frac{\partial g_1(x)}{\partial x_1} - ... - \alpha_r \frac{\partial g_r(x)}{\partial x_1} \\ . \\ . \\ . \\ \frac{\partial f}{\partial x_n}(x) - \alpha_1 \frac{\partial g_1(x)}{\partial x_n} - ... - \alpha_r \frac{\partial g_r(x)}{\partial x_n} \end{pmatrix}$$

Tal y como se puede apreciar, el gradiente se anula cuando el gradiente de la función f es combinación lineal de los gradientes de las diferentes restricciones. Es decir, se cuando

$$\frac{\partial f}{\partial x_i}(x) = \alpha_1 \frac{\partial g_1(x)}{\partial x_i} + ... + \alpha_r \frac{\partial g_r(x)}{\partial x_i} \ \forall i = 1, \cdots, n$$

$$\nabla f(x_0) = \alpha_1 \nabla g_1(x_0) + ... + \alpha_r \nabla g_r(x_0) \tag{A.7}$$

A.2.3. Condiciones de Karush-Kuhn-Tucker

Ahora se busca cambiar el problema de forma que se añaden restricciones donde no solo hay igualdades, sino que aparecen desigualdades (A.9). Se tiene el problema

$$\begin{array}{lll} (PI)\ min & f(x_1, ..., x_n) & \\ sujeto \quad a & g_i(x_1, ..., x_n) \leq 0 & \forall i = 1, ..., m \\ & h_j(x_1, ..., x_n) = 0 & \forall j = 1, ..., l \end{array} \tag{A.8}$$

siendo g_i las restricciones de desigualdad y h_j las restricciones de igualdad, todas diferenciables.

Si llamamos $x = (x_1, ..., x_n)$, la ecuación de Lagrange en este caso se define como:

$$L(x, \alpha_1, ..., \alpha_m, \alpha_{m+1}, ..., \alpha_l) = f(x) + \sum_{i=1}^{m} \alpha_i g_i(x) + \sum_{j=1}^{l} \lambda_j h_j(x) \tag{A.9}$$

Teorema 3 *Sea $\bar{x}$ un mínimo local del problema*

$$\begin{array}{lll} (PI)\ min & & f(x_1, ..., x_n) \\ sujeto \quad a & & h_j(\bar{x}) = 0\ \forall j = 1, \cdots, l \\ & & g_i(\bar{x}) \leq 0\ \forall i = 1, \cdots, m \end{array}$$

siendo $f, g_i, h_j : D \subseteq \mathbb{R}^n \to \mathbb{R}$ de clase C^1 en $int(D)$ con $i = 1, ..., m$ y $j = 1, ..., l$. Entonces existen multiplicadores $\alpha_1, ..., \alpha_m \geq 0$, $\lambda_1, ..., \lambda_l \in \mathbb{R}$ tales que:

- *Las derivadas de la función lagrangiana respecto de las variables originales son 0, $\frac{\partial L}{\partial x_i} = 0$.*

$$\nabla_x f(\bar{x}) + \sum_{i=1}^{m} \alpha_i \nabla_x g_i(\bar{x}) - \sum_{j=1}^{l} \lambda_j \nabla_x h_j(\bar{x}) = 0 \tag{A.10}$$

- *Se cumplen las restricciones:*

$$\begin{array}{l} g_i(x_1, ..., x_n) \leq 0 \quad \forall i = 1, ..., m \\ h_j(x_1, ..., x_n) = 0 \quad \forall j = 1, ..., l \end{array} \tag{A.11}$$

- *El producto de las restricciones de desigualdad en x_0 por los multiplicadores, son 0.*

$$\alpha_i(g_i(\bar{x})) = 0, \forall i = 1, \cdots, m \tag{A.12}$$

-

$$\alpha_i \geq 0\ \forall i = 1, \cdots, m \tag{A.13}$$

Se dice que el punto $\bar{x} \in \mathbb{R}^n$, junto con los vectores de multiplicadores $\alpha \in \mathbb{R}^m$ y $\lambda \in \mathbb{R}^l$, verifica las condiciones de Karush-Kuhn-Tucker (KKT) para (PI) si cumple todas las condiciones del teorema anterior.

El teorema 3 muestra las condiciones de K-K-T, que son las condiciones que hay que añadirle a la función lagrangiana en el momento en el que en las restricciones aparecen desigualdades

para encontrar el óptimo del problema. Además, estas condiciones son necesarias, es decir, dado un punto que no cumple dichas condiciones, el punto no será óptimo.

En resumen de este apartado las restricciones de Karush-Kuhn-Tucker son las siguientes

$$\begin{aligned} h_j(x) &= 0 \\ g_i(x) &\leq 0 \\ \alpha_i g_i(x) &= 0 \\ \alpha_i &\geq 0 \end{aligned} \tag{A.14}$$

Anexo B

Anexo II

B.1. Código Support Vector Machine

```
#Import python packages
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.cm import rainbow

# Machine Learning
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

dataset = pd.read_csv("/content/Heart_Disease_Prediction.csv", delimiter=";")

#Visualizar las primeras 5 filas
dataset.head()

#Se dejan solo los atributos que son cuantitativos para poder ver las correlaciones
aux=dataset.drop(['age', 'sex', 'cp', 'restecg'
            ,'exang', 'slope','ca','thal'] , axis=1)

# Matriz de correlaciones
```

```
corr_matrix = aux.corr()
fig, ax = plt.subplots(figsize=(12, 10))
ax = sns.heatmap(corr_matrix,
                 annot=True,
                 linewidths=0.5,
                 fmt=".2f",
                 cmap="YlGnBu");
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)

#Distrbuciones de los atributos cuantitativos
plt.figure(figsize=(12,13))
for i,col in enumerate(aux.columns,1):
    plt.subplot(6,4,i)
    plt.title(f"Distribution of {col} Data")
    sns.histplot(aux[col],kde=True)
    plt.tight_layout()
    plt.plot()


# dividir cada columna categórica en columnas ficticias con 1 y 0
dataset = pd.get_dummies(dataset, columns = ['cp', 'restecg', 'slope', 'ca', 'thal'])

# "Escalar" el conjunto de datos para el que usaremos el StandardScaler
columnas_escalar = ['age', 'trestbps', 'thalach', 'oldpeak']
StandardScaler = StandardScaler()
dataset[columnas_escalar] = StandardScaler.fit_transform(dataset[columnas_escalar])


#Características de destino separadas
X = dataset.drop('target',axis = 1)
y = dataset['target']


#División de los datos en conjunto de entrenamiento y conjunto de prueba
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.3,
                                                    random_state=121)

#*****************SECCIÓN ESPECÍFICA SVM******************
#Crear 4 clasificadores svm con diferentes kernels 'linear', 'poly', 'rbf', 'sigmoid'
#buscar kernel gausiano
```

```
svc_Accuracy = []
svc_classifiers=[]
y_pred_svm=[]
kernels = ['linear', 'poly', 'rbf', 'sigmoid']

for i in range(len(kernels)):

    #Crear un clasificador svm
    svc_classifier = svm.SVC(kernel = kernels[i])
    svc_classifiers.append(svc_classifier )

    #Entrena el modelo usando conjuntos de entrenamiento
    svc_classifiers[i].fit(X_train, y_train)

    #Predecir la respuesta para el conjunto de prueba
    y_pred_svm.append(svc_classifiers[i].predict(X_test))

    #Precisión de registro
    svc_Accuracy.append(accuracy_score(y_test, y_pred_svm[i])*100)
    print(kernels[i],": ",accuracy_score(y_test, y_pred_svm[i])*100)

#Matriz de confusión
for i in y_pred_svm:
    # Calcular la matriz de confusión
  confusion = confusion_matrix(y_test, i)

    # Visualizar la matriz de confusión
  labels = ['Clase 0', 'Clase 1']  # Etiquetas de las clases
  sns.heatmap(confusion, annot=True, fmt='d',
                        cmap='Blues', xticklabels=labels,
                        yticklabels=labels)
  plt.xlabel('Predicción')
  plt.ylabel('Valor Real')
  plt.title('Matriz de Confusión')
  plt.show()


#Buscar el resultado óptimo  con la función GridSearchCV
# Definir el rango de parámetros
param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],
                                      'kernel': ['rbf', 'poly', 'sigmoid', 'linear'],
                                      'degree':[1,2,3,4,5]}
```

```
grid_svm = GridSearchCV(estimator=svm.SVC(), param_grid=param_grid, cv=4)
grid_svm.fit(X_train,y_train)

results = grid_svm.cv_results_
df_results = pd.DataFrame(results)

# Filtrar los resultados para mostrar solo los valores relevantes
df_results = df_results[['param_C', 'param_gamma', 'param_kernel', 'mean_test_score']]
df_results.columns = ['C', 'gamma', 'kernel', 'Accuracy']

# Ordenar los resultados por Accuracy de forma descendente
df_results = df_results.sort_values('Accuracy', ascending=False)

# Mostrar la tabla de resultados
print(df_results)

#Guardamos el resultado obtenido
precisiones=[]
precisiones.append(grid_svm.best_score_)

#*****************************************************************
```

B.2. Código regresión logística

En cuanto al código de regresión logística, cabe destacar que es el mismo que el de SVM de la sección B.1 pero únicamente cambiando la parte del código donde se crea y entrena el modelo con el comentario de "SECCIÓN ESPECÍFICA SVM", por lo que a continuación se muestra la parte del código que modificaremos para utilizar el método de regresión logística.

```
#*****************SECCIÓN ESPECÍFICA Regresión Logística*****************
# Crear el clasificador LDA
rl = LogisticRegression()

# Calcular el score promedio de la validación cruzada
scores = cross_val_score(rl, X, y, cv=4)
accuracy = scores.mean()
precisiones.append(accuracy)
print("Accuracy:", accuracy)
```

```
# Entrenar el modelo para obtener matriz de confusión
rl.fit(X_train, y_train)

# Obtener las predicciones
y_pred = rl.predict(X_test)

# Calcular la matriz de confusión con los datos de test
confusion_matrix_rl = confusion_matrix(y_test, y_pred)

# Visualizar la matriz de confusión
etiquetas = ['Clase 0', 'Clase 1']  # Etiquetas de las clases
sns.heatmap(confusion_matrix_rl, annot=True, fmt='d',
                            cmap='Blues', xticklabels=etiquetas,
                                yticklabels=etiquetas)
plt.xlabel('Predicción')
plt.ylabel('Valor Real')
plt.title('Matriz de Confusión')
plt.show()
**************************************************************************
```

B.3. Código análisis discriminante lineal

En cuanto al código de análisis discriminante lineal (LDA), cabe destacar que es el mismo que el de SVM de la sección B.1 pero únicamente cambiando la parte del código donde se crea y entrena el modelo con el comentario de "SECCIÓN ESPECÍFICA SVM", por lo que a continuación se muestra la parte del código que modificaremos para utilizar el método de análisis discriminante lineal.

```
#*****************SECCIÓN ESPECÍFICA LDA*****************
# Crear el clasificador LDA
lda = LinearDiscriminantAnalysis()

# Calcular el score promedio de la validación cruzada
scores = cross_val_score(lda, X, y, cv=4)
accuracy = scores.mean()
precisiones.append(accuracy)
print("Accuracy:", accuracy)
```

```
# Entrenar el modelo para obtener matriz de confusión
lda.fit(X_train, y_train)

# Obtener las predicciones
y_pred = lda.predict(X_test)

# Calcular la matriz de confusión con los datos de test
confusion_matrix_lda = confusion_matrix(y_test,y_pred)

# Visualizar la matriz de confusión
etiquetas = ['Clase 0', 'Clase 1']  # Etiquetas de las clases
sns.heatmap(confusion_matrix_rl, annot=True, fmt='d',
                            cmap='Blues', xticklabels=etiquetas,
                                yticklabels=etiquetas)
plt.xlabel('Predicción')
plt.ylabel('Valor Real')
plt.title('Matriz de Confusión')
plt.show()
*******************************************************
```

B.4. Código k-vecinos más próximos

En cuanto al código de los k-vecinos más próximos, cabe destacar que es el mismo que el de SVM de la sección B.1 pero únicamente cambiando la parte del código donde se crea y entrena el modelo con el comentario de "SECCIÓN ESPECÍFICA SVM", por lo que a continuación se muestra la parte del código que modificaremos para utilizar el método de los k-vecinos más próximos. Cabe destacar que se ha seleccionado un valor de 9 para la variable k, pero es un número que puede ser seleccionado por el programador.

```
#*****************SECCIÓN ESPECÍFICA KNN******************
#PROBANDO CON DIFERENTES VALORES DE k
# Definendo el rango de parámetros
parameters = {'n_neighbors': [3, 5, 7]}

grid_knn = GridSearchCV(estimator=KNeighborsClassifier(), param_grid=parameters, cv=4)
grid_knn.fit(X_train,y_train)

results = grid_knn.cv_results_
df_results = pd.DataFrame(results)
```

```
# Filtrar los resultados para mostrar solo los valores relevantes
df_results = df_results[['param_n_neighbors', 'mean_test_score']]
df_results.columns = ['k', 'Accuracy']

# Ordenar los resultados por Accuracy de forma descendente
df_results = df_results.sort_values('Accuracy', ascending=False)

# Mostrar la tabla de resultados
print(df_results)

# Calcular la matriz de confusión con los datos de test
confusion_matrix_knn = confusion_matrix(y_test,y_pred)

# Visualizar la matriz de confusión
etiquetas = ['Clase 0', 'Clase 1']  # Etiquetas de las clases
sns.heatmap(confusion_matrix_rl, annot=True, fmt='d',
                            cmap='Blues', xticklabels=etiquetas,
                                yticklabels=etiquetas)
plt.xlabel('Predicción')
plt.ylabel('Valor Real')
plt.title('Matriz de Confusión')
plt.show()


**********************************************************
```

B.5. Gráfica final de comparación de precisión

```
colores = rainbow(np.linspace(0, 1, 4))
precisiones=[i *100 for i in precisiones]
print(precisiones)
plt.figure(figsize=(8,6))
plt.bar(['SVM', 'Regresión logística', 'LDA', 'k-vecinos'], precisiones, color = colores)
for i in range(4):
    plt.text(i, precisiones[i], "{:.2f}%".format(precisiones[i]),
                                ha='center', va='bottom' ,size=12)
plt.ylim([0,100])
plt.xlabel('Métodos')
plt.ylabel('Precisión %')
```

```
plt.title('Métodos de clasificación supervisada')
```