Universidad de Valladolid

FACULTAD DE CIENCIAS

TRABAJO FIN DE GRADO

Grado en Estadística

Métodos de Clasificación con R

Autor/a: Juan Del Pozo Yuste

Tutor/es/as: Eusebio Arenal Gutiérrez

Año 2025

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABLAS

RESUMEN

La clasificación de individuos en grupos previamente definidos en función de sus características es algo muy común en diferentes disciplinas científicas. Este TFG aborda los principales métodos de clasificación estadística, con especial atención en el análisis discriminante y los enfoques basados en la regresión. Se describen detalladamente los fundamentos teóricos de cada técnica, así como su implementación práctica mediante funciones del lenguaje de programación R.

Con el objetivo de ilustrar su aplicación y comparar su desempeño, se desarrolla un ejemplo aplicado de clasificación de dígitos manuscritos empleando tanto las variables originales como versiones reducidas mediante técnicas de reducción de dimensionalidad. Dichas técnicas incluyen el análisis de componentes principales (ACP) y una propuesta propia fundamentada en la estructura de los datos. Este estudio comparativo permite evaluar la eficacia de los distintos métodos y analizar el efecto de la reducción de variables sobre el rendimiento de la clasificación.

ABSTRACT

The classification of individuals into predefined groups based on their characteristics is a common practice across various scientific disciplines. This TFG explores the main statistical classification methods, with particular focus on discriminant analysis and regression-based methods. The theoretical foundations of each technique are described in detail, as well as their practical implementation using functions from the R programming language.

To illustrate their application and compare their performance, an applied example involving the classification of handwritten digits is developed, using both the original variables and reduced versions obtained through dimensionality reduction techniques. These techniques include principal component analysis (PCA) and a custom proposal based on the data structure. This comparative study enables the evaluation of the effectiveness of the different methods and the analysis of the impact of variable reduction on classification performance.

1. INTRODUCCIÓN

El problema de la clasificación consiste en asignar individuos a grupos ya existentes a partir de una serie de variables medidas en dichos individuos. Dentro de los métodos utilizados por la inteligencia artificial y el aprendizaje automático los métodos de clasificación son considerados como métodos de aprendizaje supervisado. En la actualidad se aplica en muchos ámbitos como la bioinformática, medicina (diagnóstico médico, medicina personalizada, epidemiología, salud pública), psicología, neurociencia, economía (estratificación del riesgo), industria etc.

La variable que indica el grupo al que pertenece cada individuo se denomina variable respuesta y, evidentemente, es una variable categórica cuyos niveles se corresponden con los grupos existentes. Las variables utilizadas para llevar a cabo esta asignación se conocen como variables explicativas, variables predictivas o predictores. Planteado en estos términos, un clasificador no es más que una aplicación que asigna a cada vector de posibles valores de los predictores un nivel de la variable respuesta. El objetivo inicial es encontrar el clasificador que mejor clasifique, esto es, que minimice la tasa de error.

La tasa de error de un clasificador viene dada por la frecuencia relativa del conjunto de los individuos mal clasificados, esto es, por la probabilidad de clasificar mal un individuo cualquiera de la población. Es evidente que como sólo se cuenta con los datos de la variable respuesta y los predictores en una muestra, el clasificador óptimo sólo podrá ser estimado a partir de la muestra.

En la práctica, la tasa de error también deberá ser estimada a partir de la muestra. Si se usa la misma muestra que se empleo para estimar el clasificador (conjunto de entrenamiento), se obtiene la tasa de error de entrenamiento, que tiende a ser una subestimación del error real. Por eso, se suele reserva una parte de la muestra para evaluar el rendimiento del clasificador (conjunto de prueba), así se obtiene la tasa de error de prueba que proporciona una estimación más realista del comportamiento del modelo. Normalmente, se asigna una mayor proporción de la muestra al conjunto de entrenamiento, ya que contar con más información mejora la estimación del clasificador.

Es posible demostrar que la tasa de error se minimiza con el clasificador de Bayes, que asigna a cada individuo el grupo más probable (con mayor frecuencia relativa) dados sus valores de los predictores. Si Y es la variable categórica que indica el grupo al que pertenece el individuo, X es el vector formado por los predictores, y x_0 los valores que toman los predictores en un individuo concreto, el clasificador de Bayes asignará al vector x_0 el grupo k donde:

$$P(Y = k \mid X = x_0)$$

alcance el máximo. En resumen, el clasificador de Bayes (C) se puede definir como:

$$C(x_0) = \underset{k \in \{1,\ldots,K\}}{\operatorname{argmax}} P(Y = k | X = x_0)$$

En el problema de dos grupos (variable respuesta binaria), el clasificador de Bayes asignará un individuo a la clase 2 si

$$P(Y = 2 \mid X = x_0) > 0.5$$

y a la clase 1 en caso contrario.

Dado un individuo cuyos predictores tomen valores del vector x_0 la probabilidad de clasificar mal al individuo vendrá dada por:

$$1 - \max_{k \in \{1,\dots,K\}} P(Y = k \mid X = x_0)$$

y por tanto la tasa de error del clasificador de Bayes es:

$$1 - E\left(\max_{k \in \{1,\dots,K\}} P(Y = k \mid X)\right)$$

Dentro de los métodos de clasificación existen dos grandes grupos: los basados en el análisis discriminante y los basados en la regresión.

Los primeros se tratarán en el capítulo 2. Estos métodos suponen que los predictores son variables aleatorias continuas y se basan en el conocimiento de las funciones de densidad $f_k(x) = f_k(x|Y = k)$ del vector de predictores, condicionadas por que el individuo pertenezca al grupo k $(\{Y = k\})$ y por las probabilidades a priori de que un individuo elegido al azar pertenezca al grupo k $\left(\pi_k = P(Y = k)\right)$, puesto que el teorema de Bayes establece que:

$$P(Y = k \mid X = x_0) = \frac{\pi_k f_k(x_0)}{\sum_{l=1}^{K} \pi_l f_l(x_0)}$$

La estimación de las probabilidades de pertenencia a un grupo (π_k) es sencilla a partir de las frecuencias relativas en la muestra. Diferentes modelos teóricos para las densidades f_k llevarán a diferentes estimaciones del clasificador de Bayes.

En los métodos basados en la regresión (capítulo 3) se utilizan modelos de regresión cuyos regresores son los predictores para estimar las probabilidades de clasificación en un grupo concreto. Dentro de estos métodos se encuentra la regresión logística, utilizada para clasificar individuos en el caso del problema de dos grupos.

Para ilustrar la aplicación de estos métodos se utilizará un ejemplo de clasificación de dígitos en el capítulo 4. Los datos han sido obtenidos de la página web del libro [9].

2. ANÁLISIS DISCRIMINANTE

El análisis discriminante consiste principalmente en la construcción de un clasificador mediante la identificación de combinaciones lineales de los predictores que dan lugar a variables linealmente independientes, las cuales maximizan la separación entre grupos [1] [9]. Esta técnica solo puede aplicarse cuando las variables explicativas son continuas. Si no lo son, deberán ser transformadas o codificadas previamente para cumplir con estos requisitos. Comenzaremos analizando el caso binario por ser conceptualmente más sencillo y representar una base sólida para comprender situaciones más complejas.

2.1. Análisis discriminante para variable respuesta binaria

En este caso particular, la variable respuesta que se desea clasificar presenta únicamente dos categorías. En este escenario, es posible aplicar tanto el análisis discriminante lineal (LDA) como el análisis discriminante cuadrático (QDA). También veremos cómo podemos usar métodos de suavizado.

2.1.1. Análisis discriminante lineal

En el análisis discriminante lineal (LDA), al igual que en el resto de métodos, pueden presentarse dos situaciones distintas, una en la que solo contamos con una variable explicativa y otra en la que contamos con más de una. En ambos casos, se denotará por p el número total de variables explicativas consideradas en el modelo.

Comenzaremos con el primero de los casos en el que p=1. Para obtener el clasificador mediante este método se asume que las funciones de densidad $f_k(x)$ corresponden a distribuciones normales de media μ_k y varianza común σ^2(la suposición de varianza común hará que el clasificador tenga forma de recta). Esto permite calcular $P(Y = k \mid X = x_0)$ mediante la aplicación directa de la fórmula de Bayes y por tanto obtener un clasificador LDA que estime el clasificador de Bayes. Una vez hecho esto clasificaremos la observación en la clase para la cual $P(Y = k \mid X = x_0)$ sea mayor.

Sustituyendo la función de densidad de la normal en la fórmula de $P(Y = k \mid X = x_0)$, y posteriormente aplicando el logaritmo, se obtiene la siguiente expresión para la función discriminante:

$$\delta_k(x) = x \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k) \text{ , k=1,2}$$

Esta función discriminante es lineal respecto a x debido a la suposición clave de que la varianza es común para todas las clases. En efecto, la función de densidad normal incluye un término cuadrático $\frac{(x-\mu_k)^k}{2\sigma_k^2}$, pero al asumir varianza común, el término cuadrático es igual en las dos clases. Cuando se comparan las funciones discriminantes para decidir la clase a la que pertenece una observación, estos términos cuadráticos se cancelan, ya que no afectan a la diferencia entre ellas. Por eso, la function discriminante final no contiene terminos cuadráticos, lo que da lugar a una frontera de decision lineal.

En este caso el clasificador LDA asigna una observación a la clase 1 si $2x(\mu_1 - \mu_2) > \mu_1^2 - \mu_2^2 - 2\sigma^2 \log\left(\frac{\pi_1}{\pi_2}\right)$ y a la clase 2 en caso contrario. La frontera de decisión LDA es el punto para el cual $\delta_1(x) = \delta_2(x)$, que es:

$$x = \frac{\mu_1 + \mu_2}{2} - \frac{\sigma^2}{\mu_1 - \mu_2} * \log\left(\frac{\pi_1}{\pi_2}\right)$$

Para ilustrar con un ejemplo la manera en la que se calcula el clasificador poblacional consideraremos una población compuesta por individuos pertenecientes a dos clases, 1 y 2. Supondremos que las observaciones de cada clase siguen distribuciones normales $f_1(x)$ y $f_2(x)$ con medias $\mu_1 = 3$ y $\mu_2 = 4$, varianza común $\sigma^2 = 2$ y probabilidades de pertenencia a cada clase $\pi_1 = 0.4$ y $\pi_2 = 0.6$.

Con estos datos la frontera de decisión sería $x \approx 2.689$ la cual podemos representar junto con la población:

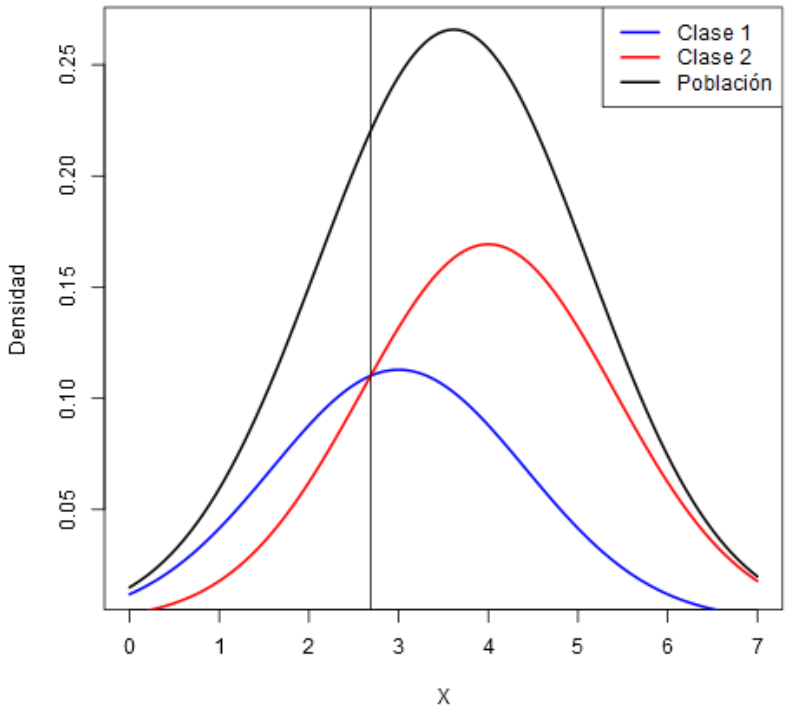

Figura 2.1.1.1: Frontera de decisión del clasificador poblacional LDA para el ejemplo propuesto de análisis discriminante lineal con una variable explicativa

En este ejemplo ideal podemos calcular la tasa de error poblacional del clasificador LDA sumando las probabilidades de cometer un error al clasificar un individuo de cada clase:

- Error al clasificar un individuo de la clase 1 como perteneciente a la clase 2:

$$P(X \geq 2.689|Y = 1) = 1 - \Phi\left(\frac{2.689 - 3}{\sqrt{2}}\right) \approx 0.587$$

- Error al clasificar un individuo de la clase 2 como perteneciente a la clase 1:

$$P(X < 2.689|Y = 2) = \Phi\left(\frac{2.689 - 4}{\sqrt{2}}\right) \approx 0.176$$

Como $\pi_1 = 0.4$ y $\pi_2 = 0.6$, la tasa de error poblacional total es:

$$t_e = 0.4 * 0.587 + 0.6 * 0.176 \approx 0.341$$

En la práctica no conoceremos todos los parámetros de la población y por tanto no podremos calcular el clasificador poblacional teniéndonos que conformar con calcular una estimación de este a través de tomar una muestra de la población. Para calcular este clasificador muestral lo que haremos será sustituir estimaciones para π_k, μ_k y σ^2, realizadas en la muestra, en la ecuación:

$$\delta_k(x) = x \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k) \text{ , k=1,2}$$

y asignar una observación a la clase para la cual esta sea mayor.

Se utilizan las siguientes estimaciones:

$$\hat{\mu}_k = \frac{1}{n_k} \sum_{\{i|y_i=k\}} x_i, \quad \hat{\sigma}^2 = \frac{1}{n-K} \sum_{k=1}^{K} \sum_{\{i|y_i=k\}} (x_i - \hat{\mu}_k)^2 \text{ , } k = 1,2$$

Donde n es el número total de observaciones de la muestra y n_k es el número de observaciones de la muestra pertenecientes a la clase k-ésima. La estimación de $\hat{\mu}_k$ es el promedio de todas las observaciones de la muestra pertenecientes a la clase k-ésima y $\hat{\sigma}^2$ es un promedio ponderado de las varianzas muestrales de cada una de las 2 clases. Acerca de las probabilidades de pertenencia, LDA estima π_k usando la proporción de observaciones de la muestra que pertenecen a la clase k-ésima:

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

Para ilustrar con un ejemplo la manera en la que se calcula el clasificador muestral lo que haremos será tomar una muestra de 800 individuos de la población. En esta muestra los primeros 600 individuos formarán el conjunto de entrenamiento y los 200 restantes el conjunto de test.

Para calcular el clasificador muestral en primer lugar, se calculan las estimaciones de π_k, μ_k y σ^2 que usando las fórmulas que vimos antes para este ejemplo serían las siguientes: $\hat{\mu}_1 \approx 2.807$, $\hat{\mu}_2 \approx 3.960$, $\hat{\sigma}^2 \approx 1.862$, $\hat{\pi}_1 = 0.395$ y $\hat{\pi}_2 = 0.605$

Una vez tenemos calculadas las estimaciones anteriores obtenemos la frontera de decisión LDA como hicimos antes:

$$x = \frac{\hat{\mu}_1 + \hat{\mu}_2}{2} - \frac{\hat{\sigma}^2}{\hat{\mu}_1 - \hat{\mu}_2} * \log\left(\frac{\hat{\pi}_1}{\hat{\pi}_2}\right) \approx 2.695$$

La frontera de decisión LDA la podemos representar junto con las observaciones y además añadir el clasificador poblacional para compararlos visualmente:

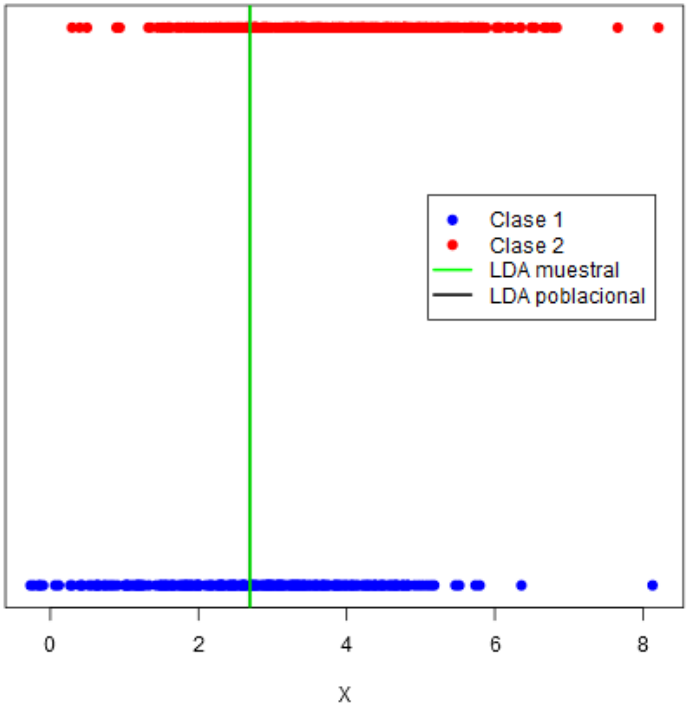

Figura 2.1.1.2: Fronteras de decisión del clasificador muestral y poblacional LDA junto con las observaciones del conjunto de entrenamiento del ejemplo propuesto de análisis discriminante lineal con una variable explicativa

Con todo esto podemos calcular la tasa de error de entrenamiento y de prueba muestrales que proporcionarán diferentes estimaciones de la tasa de error poblacional. La tasa de error de entrenamiento para este ejemplo sería:

$$t_e = \frac{196}{600} \approx 0.327$$

Como podemos ver es más pequeña que la tasa de error poblacional calculada anteriormente, seguramente debido al tamaño de la muestra. Como ya explicamos esta subestimando el error.

Para obtener una estimación de la tasa de error poblacional más creíble obtenemos la tasa de error de prueba usando el conjunto de test:

$$t_p = \frac{73}{200} = 0.365$$

Aprovechando que hemos podido calcular el clasificador poblacional vamos a comparar su funcionamiento con el del clasificador muestral (estimación del poblacional). Para ello calculamos la tasa de error del clasificador poblacional en el conjunto de test de la muestra:

$$t_p = \frac{72}{200} = 0.36$$

Como podemos ver este último ofrece una estimación más cercana de la tasa de error poblacional, como es lógico.

A continuación extenderemos el método al caso de múltiples predictores en el que p>1.Para calcular el clasificador asumiremos que $X = (X_1, X_2, \dots, X_p)$ se obtiene de una distribución normal multivariante con un vector de medias para cada clase y una matriz de covarianzas común. La distribución normal multivariante asume que cada predictor individual sigue una distribución normal unidimensional, con cierta correlación entre cada par de variables explicativas. La forma en la que denotaremos que una variable aleatoria p-dimensional X tiene una distribución normal multivariante será $X \sim N(\mu, \Sigma)$. El vector de medias de X es $E(X) = \mu$ (vector con p componentes) y la matriz de covarianzas $p \times p$ de X es $Cov(X) = \Sigma$. Con todo esto se obtiene la siguiente función discriminante:

$$\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \log \pi_k \ , k = 1,2$$

Como en el caso anterior, la frontera de decisión LDA es el punto para el cual $\delta_1(x) = \delta_2(x)$. Como antes debido a que se asume una matriz de covarianza común para todas las clases, el término cuadrático $x^T\Sigma^{-1}\mu_k$ se cancela al comparar las funciones discriminantes, simplificando la frontera a una forma lineal. Se puede demostrar que la ecuación de la frontera de decisión es:

$$x^T\Sigma^{-1}(\mu_1 - \mu_2) = \frac{1}{2}\left(\mu_1^T\Sigma^{-1}\mu_1 - \mu_2^T\Sigma^{-1}\mu_2\right) - (\log \pi_1 - \log \pi_2)$$

A partir de esta frontera de decisión, un nuevo individuo se clasifica en la clase 1 si:

$$x^T\Sigma^{-1}(\mu_1 - \mu_2) > \frac{1}{2}\left(\mu_1^T\Sigma^{-1}\mu_1 - \mu_2^T\Sigma^{-1}\mu_2\right) - (\log \pi_1 - \log \pi_2)$$

y en la clase 2 en caso contrario.

En este contexto, también es posible calcular la tasa de error poblacional del clasificador LDA. Para ello, se proyecto el vector x sobre una dirección discriminante que maximiza la separación entre clases. Esta dirección está determinada por el vector ortogonal al hiperplano de decisión.

$$\omega = \Sigma^{-1}(\mu_1 - \mu_2)$$

Sea $Z = \omega^T X$ la proyección escalar de X sobre la dirección discriminante. Dado que $X|Y = i \sim N(\mu_i, \Sigma)$, entonces $Z|Y = i \sim N(m_i, \varphi^2)$, donde $m_i = \omega^T\mu_i$ y $\varphi^2 = \omega^T\Sigma\omega$.

Como $\omega = \Sigma^{-1}(\mu_1 - \mu_2)$, se tiene que:

$$\varphi^2 = (\mu_1 - \mu_2)^T \Sigma^{-1}(\mu_1 - \mu_2)$$

es decir, el cuadrado de las distancias de Mahalanobis entre las medias.

Así, las distribuciones proyectadas son:

$$Z|Y = 1 \sim N(m_1, \varphi^2), \qquad Z|Y = 2 \sim N(m_2, \varphi^2)$$

con $m_1 - m_2 = \varphi^2$.

El punto de corte óptimo sobre el eje proyectado es:

$$z_0 = \frac{m_1 + m_2}{2} + \frac{1}{\varphi}\log\left(\frac{\pi_1}{\pi_2}\right)$$

Con esto una observación se clasifica en la clase 1 si $Z < z_0$ y en la clase 2 en caso contrario. Por tanto las probabilidades de error de clasificación son:

- Error al clasificar un individuo de la clase 1 como clase 2:
$$P(Z > z_0|Y = 1) = 1 - \Phi\left(\frac{z_0 - m_1}{\varphi}\right)$$
- Error al clasificar un individuo de la clase 2 como clase 1:
$$P(Z < z_0|Y = 2) = \Phi\left(\frac{z_0 - m_2}{\varphi}\right)$$

La tasa de error poblacional total se calcula como:

$$t_e = \pi_1 * P(Z > z_0|Y = 1) + \pi_2 * P(Z < z_0|Y = 2)$$

En el caso concreto en el que $X = (x_1, x_2)^T$, la ecuación de la frontera de decisión se puede expresar en forma explícita como una recta:

$$x_1 a_1 + x_2 a_2 = b$$

donde:

$$a_1 = \left[\Sigma^{-1}(\mu_1 - \mu_2)\right]_1, a_2 = \left[\Sigma^{-1}(\mu_1 - \mu_2)\right]_2$$

y por tanto

$$x^T \Sigma^{-1}(\mu_1 - \mu_2) = x_1 a_1 + x_2 a_2$$

el término independiente viene dado por:

$$b = \frac{1}{2}\left(\mu_1^T \Sigma^{-1} \mu_1 - \mu_2^T \Sigma^{-1} \mu_2\right) - (\log \pi_1 - \log \pi_2)$$

Con todo esto la ecuación de la frontera de decisión en forma de recta es:

$$x_2 = -\frac{a_1}{a_2} x_1 + \frac{b}{a_2}$$

En este caso bidimensional, un nuevo punto se clasifica en la clase 1 si se encuentra por encima de la recta y en la clase 2 en caso contrario.

Para ilustrar con un ejemplo la manera en la que se calcula el clasificador poblacional consideraremos una población compuesta por individuos pertenecientes a dos clases, 1 y 2. Supondremos que las observaciones de cada clase siguen distribuciones normales multivariantes $f_1(x)$ y $f_2(x)$ con vectores de medias $\mu_1 = (3, 2)$ y $\mu_2 = (4, 5)$, matriz de covarianzas común $\Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$ y probabilidades de pertenencia a cada clase $\pi_1 = 0.4$ y $\pi_2 = 0.6$. Para generar este tipo de distribuciones se necesita el paquete *mvtnorm* de R [2].

Para construir la frontera de decisión aplicamos las fórmulas que obtuvimos antes obteniendo los siguientes resultados: $a_1 \approx 0.667$, $a_2 \approx -3.333$ y $b \approx -8.928$

Con ellos vemos que la frontera de decisión es la siguiente:$x_2 \approx 0.2x_1 + 2.678$

En este ejemplo debido al signo negativo de a_2, al despejar x_2 en la ecuación de la frontera, la dirección de la desigualdad se invierte respecto a la forma general, es decir, un nuevo punto se clasifica en la clase 1 si se encuentra por debajo de la recta y en la clase 2 en caso contrario.

Una vez calculada la frontera podemos calcular la tasa de error poblacional a partir de los siguientes resultados: $m_1 = -4.665$, $m_2 = -13.997$, $\varphi \approx 3.055$ y $z_0 \approx -9.464$. Con ellos tenemos los siguientes errores de clasificación:

$$P(Z > z_0|Y = 1) \approx 0.058, P(Z < z_0|Y = 2) \approx 0.069$$

Como $\pi_1 = 0.4$ y $\pi_2 = 0.6$, la tasa de error poblacional total es:

$$t_e = 0.4 * 0.058 + 0.6 * 0.069 \approx 0.064$$

Como en el caso en el que p=1 en la práctica tendríamos que calcular una estimación de este clasificador poblacional. Para ello se realizan estimaciones en la muestra de π_k, μ_k y Σ y se sustituyen en la ecuación de la frontera de decisión. Las fórmulas de las estimaciones de π_k y μ_k son las mismas que en el caso de p=1 y la fórmula de la estimación de Σ es la siguiente:

$$\widehat{\Sigma} = \frac{1}{n-K}\sum_{k=1}^{K}\sum_{\{i|y_i=k\}}(x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T \;\; k = 1,2$$

Para ilustrar con un ejemplo la manera en la que se calcula el clasificador muestral lo que haremos será tomar una muestra de 800 individuos de la población. En esta muestra los primeros 600 individuos formarán el conjunto de entrenamiento y los 200 restantes el conjunto de test.

En este muestra tenemos $\hat{\mu}_1 = (2.88, 1.89)$,$\hat{\mu}_2 = (4.01, 5.01)$, $\widehat{\Sigma}_1 = \widehat{\Sigma}_2 = \Sigma \approx \begin{bmatrix} 0.968 & 0.494 \\ 0.494 & 1.011 \end{bmatrix}$, $\hat{\pi}_1 = 0.395$ y $\hat{\pi}_2 = 0.605$.

Una vez tenemos calculadas las estimaciones anteriores obtenemos la frontera de decisión LDA como hicimos antes:

$$x_2 = -\frac{a_1}{a_2}x_1 + \frac{b}{a_2}$$

Se sustituyen las estimaciones calculadas con las fórmulas de antes en las fórmulas del clasificador LDA y se obtienen los siguientes resultados: $a_1 = 0.539$, $a_2 = -3.354$, $b = -9.281$.

Con ellos vemos que la frontera de decisión es la siguiente:$x_2 \approx 0.161x_1 + 2.767$

Una vez tenemos la frontera de decisión LDA la podemos representar junto con las observaciones y además añadir el clasificador poblacional para compararlos visualmente. También meto en el gráfico las direcciones discriminantes muestral y poblacional ($\omega = \Sigma^{-1}(\mu_1 - \mu_2)$):

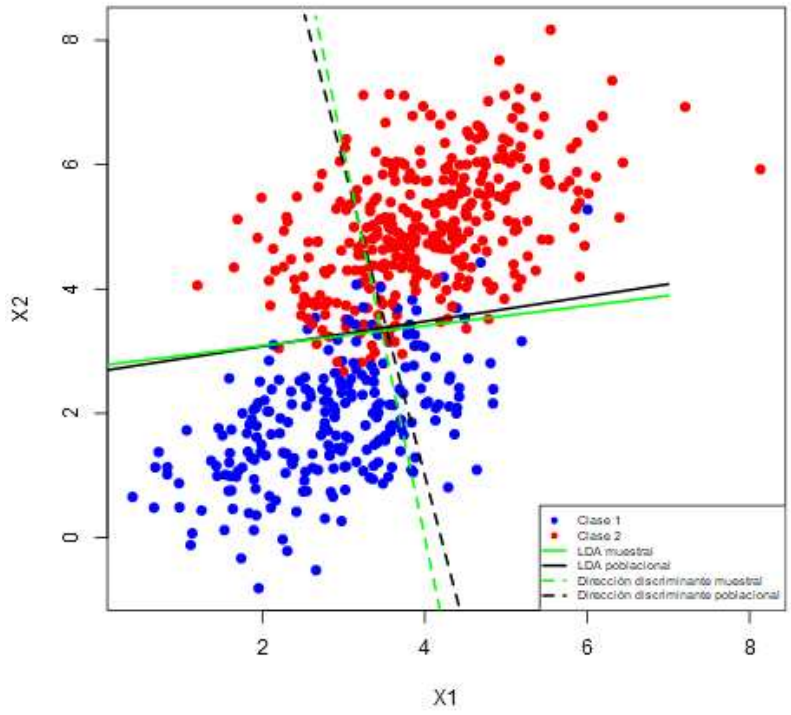

Figura 2.1.1.3: Fronteras de decisión del clasificador LDA muestral y poblacional, junto con las direcciones discriminantes correspondientes y las observaciones del conjunto de entrenamiento, para el ejemplo propuesto de análisis discriminante lineal con múltiples variables explicativas

Con todo esto podemos calcular la tasa de error de entrenamiento y de prueba muestrales que proporcionarán diferentes estimaciones de la tasa de error poblacional. La tasa de error de entrenamiento para este ejemplo sería:

$$t_e = \frac{33}{600} \approx 0.055$$

Como podemos ver es más pequeña que la tasa de error poblacional calculada anteriormente, seguramente por el tamaño de la muestra. Como ya explicamos esta subestimando el error.

Para obtener una estimación de la tasa de error poblacional más creíble obtenemos la tasa de error de prueba usando el conjunto de test:

$$t_p = \frac{13}{200} = 0.065$$

Aprovechando que hemos podido calcular el clasificador poblacional vamos a comparar su funcionamiento con el del clasificador muestral (estimación del poblacional). Para ello calculamos la tasa de error del clasificador poblacional en el conjunto de test de la muestra:

$$t_p = \frac{13}{200} = 0.065$$

Como podemos ver este último ofrece la misma estimación de la tasa de error poblacional. Esto es debido a que se trata de una simulación.

2.1.2. Análisis discriminante cuadrático

En esta sección trataremos directamente el caso en el que p>1. El análisis discriminante cuadrático (QDA, por sus siglas en inglés), al igual que LDA, surge al suponer que las observaciones de cada clase se derivan de una distribución gaussiana pero con la diferencia de asumir que cada clase tiene su propia matriz de covarianza, esto hará que ahora el clasificador tenga forma de curva al obtener términos cuadráticos. Para realizar predicciones usa estimaciones de los parámetros en el teorema de Bayes al igual que LDA. Así una observación de la clase *k*-ésima tiene la forma $X \sim N(\mu_k, \Sigma_k)$, donde $\Sigma_k$es la matriz de covarianza de la clase *k*-ésima. Bajo esta suposición, el clasificador de Bayes asigna una observación $X = x_0$ a la clase para la cual:

$$\delta_k(x) = -\frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) - \frac{1}{2}\log|\Sigma_k| + \log\pi_k\,, k = 1{,}2$$

es máxima. Por tanto la frontera de decisión QDA es el punto para el cual $\delta_1(x) = \delta_2(x)$, es decir:

$$-\frac{1}{2}(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1) - \frac{1}{2}\log|\Sigma_1| + \log\pi_1$$

$$= -\frac{1}{2}(x-\mu_2)^T\Sigma_2^{-1}(x-\mu_2) - \frac{1}{2}\log|\Sigma_2| + \log\pi_2$$

simplificando y reorganizando:

$$(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1) - (x-\mu_2)^T\Sigma_2^{-1}(x-\mu_2) = C$$

donde $C = \log|\Sigma_2| - \log|\Sigma_1| + 2\log\pi_2 - 2\log\pi_1$.

Como cada término del tipo $(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k)$ es una forma cuadrática en x y al restar dos formas cuadráticas el resultado sigue siendo una expresión cuadrática, se puede demostrar que la ecuación de la frontera de decisión es del tipo:

$$x^TAx + b^Tx + c = 0$$

donde:

$$A = \Sigma_1^{-1} - \Sigma_2^{-1}$$

$$b = -2(\Sigma_1^{-1}\mu_1 - \Sigma_2^{-1}\mu_2)$$

$$c = \mu_1^T\Sigma_1^{-1}\mu_1 - \mu_2^T\Sigma_2^{-1}\mu_2 - C$$

Por tanto, la manera de clasificar una nueva observación sería la siguiente:

- Si $x^TAx + b^Tx + c < 0$, se clasifica en la clase 1.
- Si $x^TAx + b^Tx + c > 0$, se clasifica en la clase 2.

Para ilustrar con un ejemplo la manera en la que se calcula el clasificador poblacional consideraremos una población compuesta por individuos pertenecientes a dos clases, 1 y 2. Supondremos que las observaciones de cada clase siguen distribuciones normales multivariantes $f_1(x)$ y $f_2(x)$ con vectores de medias $\mu_1 = (3, 2)$ y $\mu_2 = (4, 5)$, matrices de covarianzas $\Sigma_1 = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$ $y\ \Sigma_2 = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$ y probabilidades de pertenencia a cada clase $\pi_1 = 0.4$ y $\pi_2 = 0.6$.

Para construir la frontera de decisión aplicamos las fórmulas que obtuvimos antes obteniendo los siguientes resultados: $A \approx \begin{bmatrix} 0.667 & -0.333 \\ -0.333 & 0.667 \end{bmatrix}$, $b^T \approx [-3.333, 2.667]$ y $c \approx -6.867$

Con ellos vemos que la frontera de decisión es la siguiente:

$$0.667x_1^2 + 0.667x_2^2 - 0.667x_1x_2 - 3.333x_1 + 2.667x_2 - 6.867 \approx 0$$

A pesar de que estamos en un ejemplo ideal en el que conocemos la población completa junto con todos los parámetros involucrados, para este clasificador QDA no existe una expresión analítica cerrada para su tasa de error poblacional como existía en LDA. Esto es debido a que la frontera de decisión del clasificador QDA es de naturaleza cuadrática y no se puede proyectar sobre una única dirección. Por tanto, en este apartado estimaremos dicha tasa mediante un enfoque de simulación Monte Carlo, generando una muestra suficientemente grande (100000 individuos) a partir de las distribuciones originales de ambas clases. Realizada la simulación tenemos que la tasa de error poblacional estimada es la siguiente:

$$t_e \approx 0.098$$

Como en LDA en la práctica tendríamos que calcular una estimación de este clasificador poblacional. Para ello se realizan estimaciones en la muestra de π_k, μ_k y Σ_k y se sustituyen en la ecuación de la frontera de decisión. Las fórmulas de las estimaciones de π_k y μ_k son las mismas que en el análisis discriminante lineal y la fórmula de la estimación de Σ_k es la siguiente:

$$\hat{\Sigma}_k = \frac{1}{n_k - 1} \sum_{\{i|y_i=k\}} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T \quad k = 1,2$$

Para ilustrar con un ejemplo la manera en la que se calcula el clasificador muestral lo que haremos será tomar una muestra de 800 individuos de la población. En esta muestra los primeros 600 individuos formarán el conjunto de entrenamiento y los 200 restantes el conjunto de test.

En este ejemplo tenemos $\hat{\mu}_1 \approx (2.88, 1.89)$,$\hat{\mu}_2 \approx (4.02, 5.01)$,$\widehat{\Sigma}_1 \approx \begin{bmatrix} 0.941 & 0.515 \\ 0.515 & 0.975 \end{bmatrix}$ $\widehat{\Sigma}_2 \approx \begin{bmatrix} 1.972 & 0.962 \\ 0.962 & 2.068 \end{bmatrix}$, $\hat{\pi}_1 = 0.395$ y $\hat{\pi}_2 = 0.605$.

Una vez tenemos calculadas las estimaciones anteriores obtenemos la frontera de decisión QDA como hicimos antes y se obtienen los siguientes resultados al sustituir las estimaciones calculadas en las fórmulas del clasificador QDA: $A \approx \begin{bmatrix} 0.840 & -0.486 \\ -0.486 & 0.818 \end{bmatrix}$, $b^T \approx [-3.414, 2.928]$ y $c \approx -7.516$

Con ellos vemos que la frontera de decisión es la siguiente:

$$0.840x_1^2 + 0.818x_2^2 - 0.972x_1x_2 - 3.414x_1 + 2.928x_2 - 7.516 = 0$$

Una vez tenemos la frontera de decisión la podemos representar junto con las observaciones y además añadir el clasificador poblacional para compararlos visualmente:

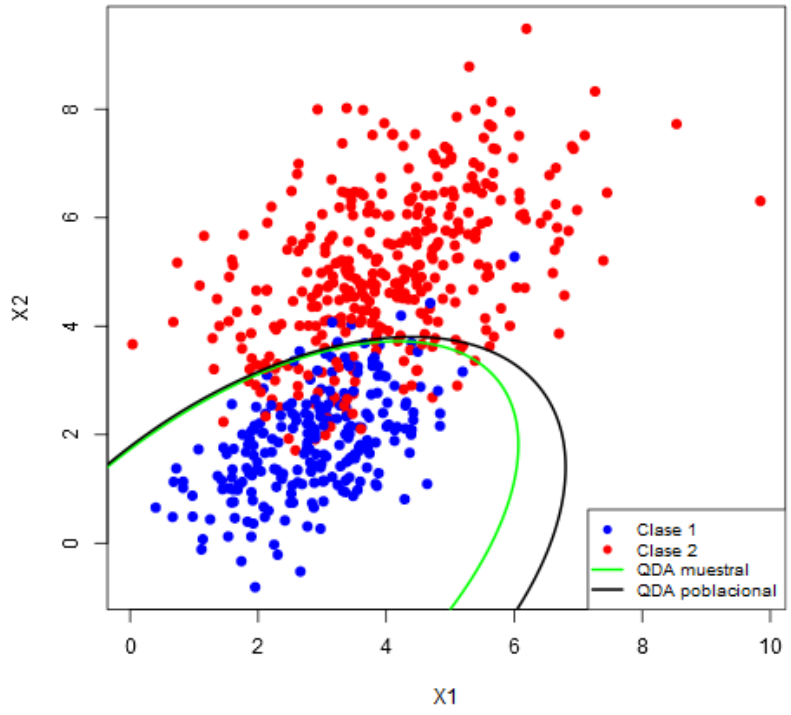

Figura 2.1.2.1: Frontera de decisión del clasificador muestral y poblacional QDA junto con las observaciones del conjunto de entrenamiento para el ejemplo propuesto de análisis discriminante cuadrático

Con todo esto podemos calcular la tasa de error de entrenamiento y de prueba muestrales que proporcionarán diferentes estimaciones de la tasa de error poblacional. La tasa de error de entrenamiento para este ejemplo sería:

$$t_e = \frac{58}{600} \approx 0.097$$

Como podemos ver es más pequeña que la tasa de error poblacional calculada anteriormente, seguramente por el tamaño de la muestra. Como ya explicamos esta subestimando el error.

Para obtener una estimación de la tasa de error poblacional más creíble obtenemos la tasa de error de prueba usando el conjunto de test:

$$t_p = \frac{23}{200} = 0.115$$

Aprovechando que hemos podido calcular el clasificador poblacional vamos a comparar su funcionamiento con el del clasificador muestral (estimación del poblacional). Para ello calculamos la tasa de error del clasificador poblacional en el conjunto de test de la muestra:

$$t_p = \frac{22}{200} = 0.11$$

Como podemos ver este último ofrece una estimación más cercana de la tasa de error poblacional, como es lógico.

La explicación de por qué añadir esta mayor complejidad al método LDA radica en la compensación entre sesgo y varianza del clasificador, entendidos como el error sistemático y la sensibilidad al conjunto de entrenamiento, respectivamente, en la función de decisión estimada. Cuando hay *p* variables explicativas, estimar una matriz de covarianza requiere estimar $\frac{p(p+1)}{2}$ parámetros, mientras que estimar una matriz de covarianzas para cada clase requiere estimar un total de $K\frac{p(p+1)}{2}$ parámetros, lo que es una gran diferencia. Además, al asumir que las *K* clases comparten una matriz de covarianzas común, el modelo LDA se vuelve lineal en x, lo que implica que hay *Kp* coeficientes lineales por estimar. Esto hace que LDA sea un clasificador mucho menos flexible que QDA y con una varianza menor, lo que puede mejorar el desempeño de la predicción. Lo que hace a QDA interesante es en el caso de que la suposición de covarianza común entre clases sea incorrecta, ya que LDA puede sufrir un sesgo elevado en la clasificación. Por tanto, LDA suele ser mejor que QDA si hay relativamente pocas observaciones de entrenamiento y, por lo tanto, reducir la varianza del clasificador es crucial. Por el contrario, se recomienda QDA si el conjunto de entrenamiento es muy grande, de modo que la varianza del clasificador no sea un problema importante, o si la suposición de una matriz de covarianza común para las *K* clases es claramente inaceptable.

2.1.3. Análisis discriminante no paramétrico

Comenzaremos con el caso en el que p=1. Como se ha observado en el enfoque clásico del análisis discriminante, este método se basa en una formulación paramétrica específica para f_k dentro del teorema de Bayes. En el caso del análisis discriminante lineal (LDA), se asume que cada clase sigue una distribución normal con varianzas comúnes. Por el contrario, el análisis discriminante cuadrático (QDA) permite que cada clase tenga su varianza. Sin embargo, una limitación importante de estas reglas de clasificación es que, si las suposiciones paramétricas no se cumplen en la práctica, el rendimiento del clasificador puede verse seriamente comprometido, llegando incluso a ser completamente ineficaz.

Por ello surge la idea de utilizar un análisis discriminante no paramétrico en el que sustituiremos una forma no paramétrica de f_k en el teorema de Bayes [4]. Esta forma es la siguiente:

$$\hat{f}_k(x) = \frac{1}{n_k h^p} \sum_{\{i|y_i=k\}} K\left(\frac{x - x_i}{h}\right)$$

Donde $x \in \mathbb{R}$ es el punto que queremos clasificar, $x_i \in \mathbb{R}$ son las observaciones del conjunto de entrenamiento, n_k es el número de observaciones de cada clase, $K(x)$ es la función núcleo y h es el parámetro de suavizado común.

La función núcleo es una función simétrica y positiva que se utiliza para asignar peso a las observaciones cercanas a x al estimar la densidad de probabilidad. Cuanto más cerca este x_i de x, mayor será el peso de esa observación.

Una parte fundamental en este tipo de estimación es la elección del parámetro h ya que controla el grado de suavidad en la estimación de las densidades. Normalmente se suele utilizar un h común para las dos clases para evitar sesgos artificiales entre las clases al suavizar las densidades estimadas en diferentes escalas, para hacer una comparación directa de las dos densidades al estimarlas con la misma suavidad y para reducir el número de hiperparámetros facilitando el ajuste del modelo y reduciendo el riesgo de sobreajuste.

En términos generales, un valor pequeño de h conduce a una estimación muy ajustada a los datos, con menor suavidad y mayor varianza, lo que puede provocar sobreajuste. Por el contrario, un valor grande de h produce una estimación suave y general, con menor varianza pero mayor sesgo, lo que puede dificultar la detección de estructuras importantes en los datos. Existen distintos métodos para seleccionar su valor, entre ellos la validación cruzada.

Desde otro punto de vista, el parámetro h puede interpretarse como un factor de escala del núcleo utilizado en la estimación. Ajustar h equivale a estirar o comprimir el núcleo alrededor de cada observación, afectando directamente a la región del espacio sobre la que cada dato ejerce influencia. De este modo, el cambio en h no altera la forma del núcleo, sino su alcance efectivo, modificando así la suavidad del modelo.

Esta idea resulta especialmente útil al trabajar con modelos basados en kernels, como las máquinas de vectores soporte (SVM), donde h no aparece de forma explícita pero se representa mediante parámetros equivalentes que actúan reescalando internamente el núcleo. Esta equivalencia será particularmente relevante en el caso del modelo *ksvm*, como se detallará en la sección 2.3.

La siguiente tabla muestra algunas de las funciones núcleo univariantes que se pueden utilizar:

Kernel	Forma	Soporte
Epanechnikov	$\frac{3}{4}(1-u^2)$	$u \in (-1,1)$
Biweight	$\frac{15}{16}(1-u^2)^2$	$u \in (-1,1)$
Triweight	$\frac{35}{32}(1-u^2)^3$	$u \in (-1,1)$
Gaussiano	$(2\pi)^{-1/2}e^{-u^2/2}$	$u \in \mathbb{R}$
Uniforme	$\frac{1}{2}$	$u \in (-1,1)$

Tabla 2.1.3.1: Tabla de algunas funciones núcleo univariantes existentes

Como ya sabemos tenemos:

$$\delta_k(x) = \frac{\pi_k \hat{f}_k(x_0)}{\sum_{l=1}^{K} \pi_l \hat{f}_l(x_0)}, k = 1,2$$

La frontera de decisión no paramétrica entre las dos clases se define entonces como el conjunto de puntos $x \in \mathbb{R}$ que satisfacen $\delta_1(x) = \delta_2(x)$, que es:

$$\frac{\pi_1}{n_1} \sum_{\{i|y_i=1\}} K\left(\frac{x-x_i}{h}\right) = \frac{\pi_2}{n_2} \sum_{\{i|y_i=2\}} K\left(\frac{x-x_i}{h}\right)$$

Como podemos ver no podemos obtener la frontera de decisión de forma explícita como en los anteriores casos. En el clasificador Bayesiano no paramétrico con estimación de densidad mediante núcleos, la frontera se define de manera implícita como el conjunto de puntos donde ambas densidades estimadas, ponderadas por las probabilidades a priori, son iguales. Esta frontera generalmente no está cerrada y puede estar compuesta por varios puntos.

Para poder calcular la frontera tendremos que evaluar siguiente ecuación en una malla de valores de x:

$$x = \frac{\pi_1}{n_1} \sum_{\{i|y_i=1\}} K\left(\frac{x-x_i}{h}\right) - \frac{\pi_2}{n_2} \sum_{\{i|y_i=2\}} K\left(\frac{x-x_i}{h}\right)$$

Estos serán los puntos que definen las fronteras entre ambas clases.

Para el caso de múltiples predictores en el que p>1, las fórmulas son las mismas que antes con la única diferencias de que ahora en vez de obtener uno o varios puntos como frontera de decisión se obtienen regiones curvas que se adaptan a la estructura real de los datos.

Es aquí donde entra el conocido problema de la dimensionalidad que sufren estas estimaciones no paramétricas al aumentar el número de predictores. Conforme crece p la estimación con núcleos pierde precisión y eficacia debido a la escasez de puntos cercanos en espacios de alta dimensión. Por ello, en contextos multivariantes es habitual aplicar técnicas complementarias como la reducción de dimensionalidad, por ejemplo con PCA (Análisis de componentes principales).

2.2 Análisis discriminante para variable respuesta con más de dos grupos

En este caso, la variable respuesta que se desea clasificar presenta más de dos categorías, lo que implica un problema de clasificación multiclase. Para abordarlo, es posible aplicar tanto el análisis discriminante lineal (LDA) como el análisis discriminante cuadrático (QDA), en función de los supuestos que se consideren adecuados. Además, se explorará el uso de métodos de suavizado como una estrategia complementaria para mejorar la precisión del clasificador en este contexto más complejo.

Para ilustrar con un ejemplo el funcionamiento de estos métodos de clasificación en esta situación consideraremos una población compuesta por individuos pertenecientes a tres clases, 1, 2 y 3. Supondremos que las observaciones de cada clase siguen distribuciones normales multivariantes $f_1(x)$, $f_2(x)$ y $f_3(x)$ con vectores de medias $\mu_1 = (1, 2)$, $\mu_2 = (5, 6)$ y $\mu_3 = (9, 2)$, matrices de covarianzas $\Sigma_1 = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$, $\Sigma_2 = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$ y $\Sigma_3 = \begin{bmatrix} 0.5 & 0.25 \\ 0.25 & 0.5 \end{bmatrix}$ y probabilidades de pertenencia a cada clase $\pi_1 = 0.4$, $\pi_2 = 0.3$ y $\pi_3 = 0.3$.

Como en la práctica no dispondremos de la población y los parámetros involucrados tomaremos una muestra de 800 individuos de la población, los primeros 600 individuos los usaremos para construir los clasificadores (conjunto de entrenamiento) y los 200 restantes para evaluarlos y compararlos (conjunto de test).

Con el objetivo de comparar estos métodos de análisis discriminante con los de regresión logística, se utilizarán estos mismos datos como ejemplo para la regresión logística multinomial.

Los cálculos de las fronteras de decisión y las tasas de error de prueba se realizarán usando funciones de R que serán explicadas con detalle en la sección 2.3.

2.2.1. Análisis discriminante lineal

Tal como se analizó previamente en el caso binario, el análisis discriminante lineal (LDA) puede abordarse considerando dos escenarios: cuando se dispone de una única variable explicativa y cuando se cuenta con varias variables explicativas. Esta distinción se mantiene también en el contexto multiclase, es decir, cuando la variable respuesta posee más de dos categorías.

Dado que gran parte del enfoque es análogo al del caso binario, nos centraremos únicamente en señalar y explicar las diferencias clave que surgen en este nuevo contexto.

Comenzaremos con el primero de los casos en el que p=1. En este caso se hacen las mismas suposiciones que en el caso de tener dos categorías y al final se obtiene la siguiente expresión para la función discriminante:

$$\delta_k(x) = x \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k), k = 1, \dots, K$$

A la hora de clasificar una nueva observación x, el clasificador LDA asigna dicha observación a la clase k cuya función discriminante $\delta_k(x)$ sea máxima.

Aunque la clasificación se realiza comparando todas las funciones $\delta_k(x)$ al mismo tiempo, las fronteras de decisión entre clases sí se obtienen comparando las funciones discriminantes por pares. La frontera de decisión LDA entre dos clases k y l es el punto para el cual $\delta_k(x) = \delta_l(x)$, que es:

$$x = \frac{\mu_k + \mu_l}{2} - \frac{\sigma^2}{\mu_k - \mu_l} * \log\left(\frac{\pi_k}{\pi_l}\right)$$

Existen $\binom{K}{2}$ fronteras, una para cada par de clases.

Ahora trataremos el caso de múltiples predictores en el que p>1. En este caso se hacen las mismas suposiciones que en el caso de tener dos categorías y al final se obtiene que el clasificador LDA clasifica una observación en la clase para la cual:

$$\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \log \pi_k \ , k = 1, \dots, K$$

Es mayor. Para las clases k y l, la frontera de decisión LDA es el punto para el cual $\delta_k(x) = \delta_l(x)$, que en términos explícitos es:

$$x_l = -\frac{a_k}{a_l}x_k + \frac{b_{kl}}{a_l}$$

donde

$$a_k = \left[\Sigma^{-1}(\mu_k - \mu_l)\right]_{k,} a_l = \left[\Sigma^{-1}(\mu_k - \mu_l)\right]_l$$

y el término independiente viene dado por:

$$b_{kl} = \frac{1}{2}\left(\mu_k^T\Sigma^{-1}\mu_k - \mu_l^T\Sigma^{-1}\mu_l\right) - (\log \pi_k - \log \pi_l)$$

Existen $\binom{K}{2}$ fronteras, una para cada par de clases.

Las fronteras de decisión obtenidas para el conjunto de entrenamiento del ejemplo propuesto serían:

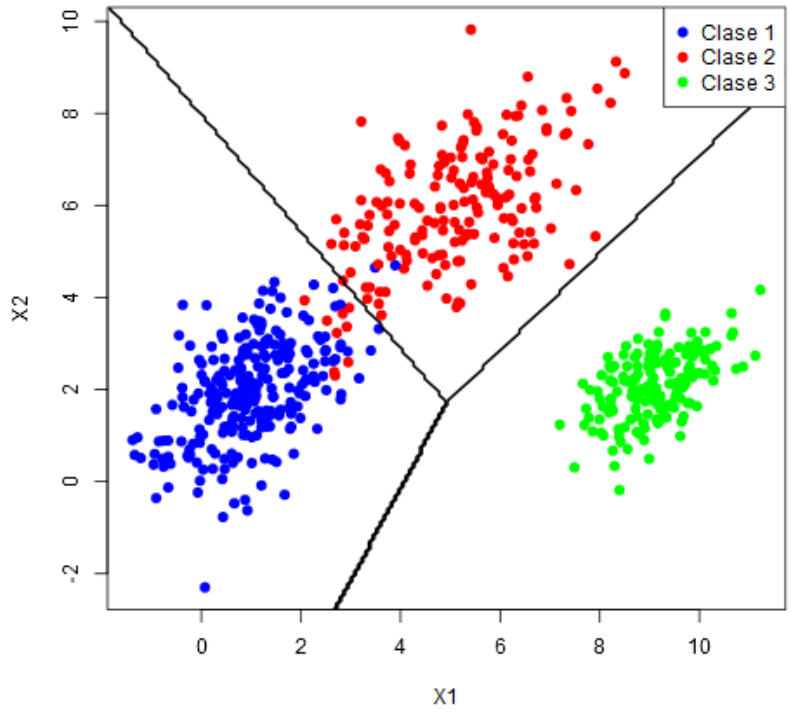

Figura 2.2.1.1: Fronteras de decisión LDA para el conjunto de entrenamiento del ejemplo propuesto para análisis discriminante con variable respuesta con más de dos categorías

Clasificando las distintas observaciones mediante la función discriminante podemos calcular la tasa de error de prueba y obtenemos el siguiente resultado:

$$t_p = \frac{2}{200} = 0.01$$

2.2.2. Análisis discriminante cuadrático

Aquí solo tratamos el caso en el que p>1 como hicimos antes. Al igual que en el análisis discriminante lineal no hay mucha variación al tener una variable respuesta con más de dos categorías. En esta situación, el clasificador QDA asigna una observación $X = x_0$a la clase para la cual:

$$\delta_k(x) = -\frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) - \frac{1}{2}\log|\Sigma_k| + \log\pi_k\,, k = 1, \dots, K$$

Es máxima. Para las clases k y l, la frontera de decisión QDA es el punto para el cual $\delta_k(x) = \delta_l(x)$, expresando cada lado de la ecuación como una forma cuadrática esta frontera se puede escribir como:

$$x^T A_{kl} x + b_{kl}^T x + c_{kl} = 0$$

donde:

$$A_{kl} = \Sigma_k^{-1} - \Sigma_l^{-1}$$

$$b_{kl} = -2(\Sigma_k^{-1}\mu_k - \Sigma_l^{-1}\mu_l)$$

$$c_{kl} = \mu_k^T \Sigma_k^{-1} \mu_k - \mu_l^T \Sigma_l^{-1} \mu_l - C_{k,l}$$

$$C_{k,l} = \log|\Sigma_l| - \log|\Sigma_k| + 2\log \pi_l - 2\log \pi_k$$

Existen $\binom{K}{2}$ fronteras, una para cada par de clases.

Estas fronteras de decisión no son necesariamente planas ni lineales, sino que pueden tener formas curvas complejas y cerradas o abiertas, lo cual le otorga a QDA una alta capacidad de adaptación a estructuras no lineales en los datos. No obstante, esta flexibilidad requiere una mayor cantidad de datos para estimar correctamente todas las matrices de covarianza individuales Σ_k, lo que puede resultar en sobreajuste si el número de muestras por clases es bajo en relación con la dimensión p.Las fronteras de decisión obtenidas para el conjunto de entrenamiento del ejemplo propuesto serían:

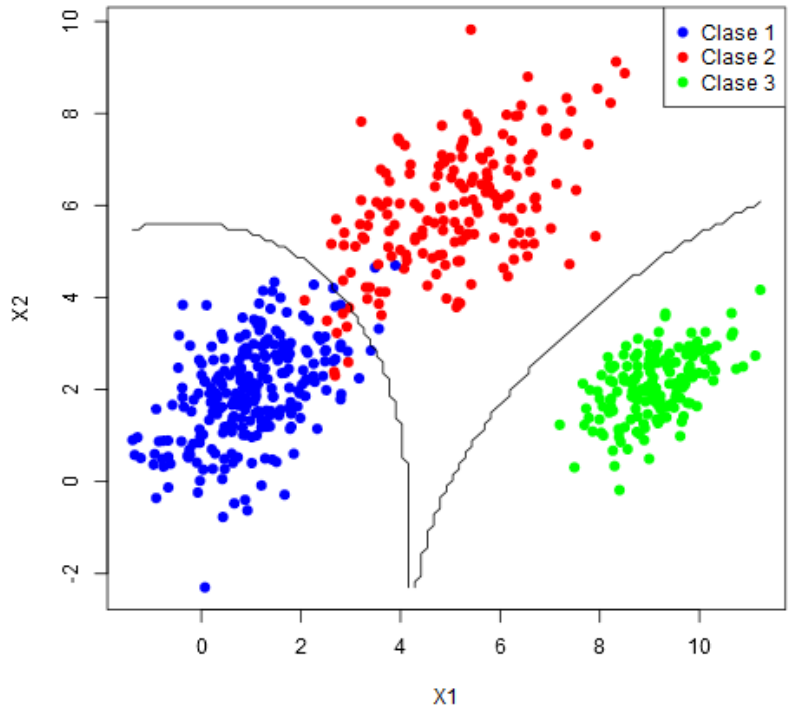

Figura 2.2.2.1: Fronteras de decisión QDA para el conjunto de entrenamiento del ejemplo propuesto para análisis discriminante con variable respuesta con más de dos categorías

Clasificando las distintas observaciones mediante la función discriminante podemos calcular la tasa de error de prueba y obtenemos el siguiente resultado:

$$t_p = \frac{1}{200} = 0.005$$

Como podemos ver funciona mejor que el LDA. Esto puede ser debido a la existencia de matrices de covarianza distintas.

2.2.3. Análisis discriminante no paramétrico

Aquí trataremos un caso general en el que p>0, debido a que los casos p=1 y p>1 son prácticamente similares. Este tipo de análisis tampoco varía a la hora de añadir que la variable respuesta cuente con más de dos categorías.

La regla de clasificación se sigue basando en el teorema de Bayes, seleccionando la clase k que maximiza la probabilidad a posteriori:

$$\delta_k(x) = \frac{\pi_k \hat{f}_k(x_0)}{\sum_{l=1}^{K} \pi_l \hat{f}_l(x_0)}, k = 1, \dots, K$$

Para las clases k y l, la frontera de decisión no paramétrica es el punto para el cual $\delta_k(x) = \delta_l(x)$, es decir:

$$x = \frac{\pi_k}{n_k} \sum_{\{i|y_i=k\}} K\left(\frac{x - x_i}{h}\right) - \frac{\pi_l}{n_l} \sum_{\{i|y_i=l\}} K\left(\frac{x - x_i}{h}\right)$$

Esto conduce a $\binom{K}{2}$ fronteras, una para cada par de clases que, como antes, no puede expresarse de forma cerrada, pero que puede evaluarse en una malla de valores de x para estimar las regiones de decisión.

Las fronteras de decisión obtenidas para el conjunto de entrenamiento del ejemplo propuesto serían:

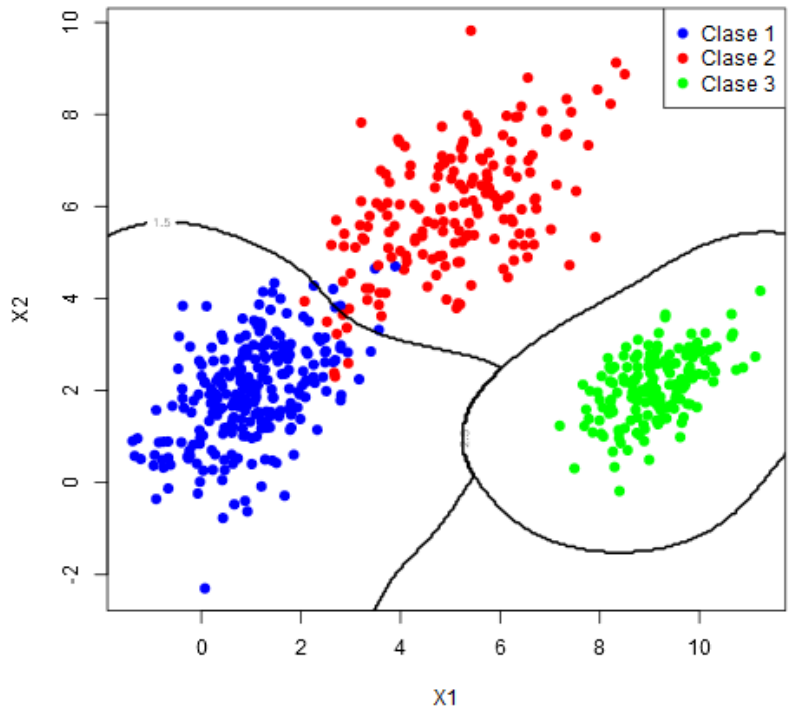

Figura 2.2.3.1: Fronteras de decisión no paramétricas para el conjunto de entrenamiento del ejemplo propuesto para análisis discriminante con variable respuesta con más de dos categorías

Clasificando las distintas observaciones mediante la función discriminante podemos calcular la tasa de error de prueba y obtenemos el siguiente resultado:

$$t_p = \frac{1}{200} = 0.005$$

Como podemos ver funciona igual que el QDA.

2.3 Funciones en R para el análisis discriminante

Las tres principales funciones en R [8] para realizar todos los tipos de análisis discriminante que hemos visto son:

- *lda*: Función del paquete *MASS* [10], útil para realizar análisis discriminante lineal con variable respuesta con dos o más categorías y con una o más variables explicativas. Asume normalidad multivariante en las clases y homogeneidad de varianzas para estimar el clasificador.

- *qda*: Función del paquete *MASS* [10], útil para realizar análisis discriminante cuadrático con variable respuesta con dos o más categorías y con una o más variables explicativas. Asume normalidad multivariante en las clases y varianzas distintas para estimar el clasificador.
- *ksvm*: Función del paquete *kernlab* [5] [6], útil para realizar análisis discriminante no paramétrico con variable respuesta con dos o más categorías y con una o más variables explicativas.

Estas tres funciones son las que se han usado para calcular los clasificadores del análisis discriminante lineal (*lda*), cuadrático (*qda*) y no paramétrico (*ksvm*) en la sección anterior.

Las tres funciones requieren de la introducción de una fórmula en la que se le indique la variable respuesta y los predictores, esta fórmula se indica así:

variable respuesta ~ predictor1 + predictor2 + …

Por defecto las funciones toman esas variables del entorno global (*.GlobalEnv*) pero podemos indicarles el conjunto de datos de donde queremos que las cojan con el argumento *data*.

Además de estos dos argumentos comunes *lda* y *qda* disponen de otro argumento interesante que es *prior* que sirve para indicar las probabilidades previas de pertenencia de las observaciones a cada una de las clases del modelo. Su funcionamiento es simple, si no especificas *prior* R calcula automáticamente las proporción de clases en los datos de entrenamiento y si lo especificas le estás diciendo al modelo que quieres asumir esas probabilidades a priori independientemente de la proporción real de las clases en los datos. Esto puede ser útil para controlar sesgos en los datos, incorporar conocimiento externo o mejorar la clasificación.

Al igual que las funciones anteriores *ksvm* también tiene argumentos propios siendo el más importante *kernel*, este argumento sirve para especificar la función núcleo que se utilizará para crear el clasificador. Por defecto usa el núcleo rbfdot (gaussiano) pero tiene muchos más disponibles como: polydot (núcleo polinómico), vanilladot (núcleo lineal), tanhdot (núcleo tangente hiperbólico), laplacedot (núcleo laplaciano), besseldot (núcleo de Bessel), anovadot (núcleo ANOVA RBF), splinedot (núcleo spline) y stringdot (núcleo de cadena).

Como se mencionó en la sección correspondiente al análisis discriminante no paramétrico, el parámetro *h* desempeña un papel importante en este tipo de modelos. Sin embargo, en la función *ksvm*, no existe un argumento explícito llamado h que permita indicarlo directamente. En su lugar, debe utilizarse el argumento *kpar*, que consiste en una lista de parámetros específicos asociados al núcleo seleccionado que permiten controlar determinadas particularidades de este.

Cada tipo de núcleo admite sus propios parámetros, y algunos de ellos cumplen una función equivalente a la del parámetro *h*. Por ejemplo, en el caso del núcleo rbfdot (gaussiano), el parámetro *sigma* es el que cumple esta función.

La relación matemática entre ambos parámetros es la siguiente:

$$\sigma = \frac{1}{2h^2} \Leftrightarrow h = \sqrt{\frac{1}{2\sigma}}$$

Por lo que para introducir un valor específico de *h* solo habría que poner el *sigma* correspondiente a ese valor de *h*.

Si no se especifica manualmente un valor para estos parámetros de suavizado dentro de *kpar*, el paquete *kernlab* los estima automáticamente mediante métodos heurísticos basados en las distancias entre las observaciones del conjunto de entrenamiento, buscando proporcionar un compromiso razonable entre sobreajuste y subajuste.

Para calcular el clasificador con los distintos métodos visto en las secciones anteriores, apliqué cada una de las funciones correspondientes al conjunto de entrenamiento.

En el caso del análisis discriminante lineal y cuadrático, no especifiqué las probabilidades a priori mediante el parámetro *prior*, ya que nos interesaba que estas se estimasen directamente a partir del conjunto de entrenamiento.

Para el análisis no paramétrico, utilicé un núcleo rbfdot (gaussiano), a pesar de que los datos podrían parecer linealmente separables. Esta decisión se fundamenta en que, como se observó anteriormente, el clasificador cuadrático ofrecía un mejor rendimiento que el lineal. En cuanto al parámetro de suavizado, que en este caso se especificaría con sigma, no lo especifiqué explícitamente, permitiendo que el propio entorno de R lo estimase automáticamente.

Cabe destacar que estas funciones no proporcionan explícitamente las fronteras de decisión, pero si devuelven la información necesaria para poder representarlas gráficamente.

Para calcular la tasa de error de prueba utilicé el método genérico *predict* del paquete *stats* [8] (paquete base de R), que permite obtener predicciones a partir de un modelo previamente ajustado. Este método genérico pertenece al sistema S3, lo que implica que su comportamiento depende de la clase de objeto que se le proporciona como argumento. En nuestro caso, al aplicarlo sobre los objetos de las clases *lda*, *qda* y *ksvm*, *predict* ejecuta internamente las funciones específicas *predict.lda*, *predict.qda* y *predict.ksvm*, respectivamente.

El método *predict* tiene como argumentos principales *object*, que representa el modelo previamente entrenado, y *newdata*, que correspondiente al conjunto de observaciones sobre el que se desea realizar la predicción. Al aplicarlo a los distintos modelos junto con los datos del conjunto de entrenamiento, todas las funciones mencionadas devuelven el vector de clases predichas (*$class*, para *predict.lda* y *predict.qda*). Además de ese vector las funciones *predict.lda* y *predict.qda* devuelven la matriz de probabilidades posteriores por clase (*$posterior*) y, en el caso de *predict.lda*, los valores de las funciones discriminantes lineales (*$x*).

Quedándonos con las predicciones, la tasa de error de prueba se calculó comparando las clases reales con las predichas.

3. MÉTODOS BASADOS EN LA REGRESIÓN

En esta sección se estudiarán diversos métodos basados en la regresión, entre los cuales destaca la regresión logística. Esta técnica se emplea para modelar la probabilidad de pertenencia de un individuo a una clase determinada, siendo especialmente útil cuando la variable respuesta Y es binaria, es decir, cuando solo existen dos categorías posibles.

El modelo de regresión logística se basa en una función con forma de S (o S invertida), que es monótona creciente y acotada entre 0 y 1 [7]. Su objetivo es estimar las probabilidades de que un individuo con valores x_0 en los predictores pertenezca a una de las dos clases. Esta probabilidad se modela mediante la siguiente función:

$$P(Y = 1|X = x_0) = \pi(x_0) = \frac{1}{1 + e^{-x_0'\beta}}$$

Donde β es el vector de parámetros del modelo. Para facilitar la estimación y la interpretación del modelo, se suele utiliza la transformación logit como función de enlace, lo que permite expresar la relación como una función lineal:

$$\ln\left(\frac{\pi(x_0)}{1 - \pi(x_0)}\right) = x_0'\beta$$

Esta transformación, conocida como función logit, convierte la probabilidad en la razón de odds (o razón de probabilidades), facilitando la interpretación de los coeficientes del modelo como cambios log-lineales en dicha razón.

Cabe destacar que la regresión logística es un caso particular de los modelos lineales generalizados (GLM), un marco estadístico que extiende la regresión lineal tradicional para variables respuesta con distribuciones distintas a la normal. En concreto, la regresión logística corresponde a un GLM con distribución binomial y función de enlace logit.

Además, este método de clasificación puede extenderse a situaciones en las que Y puede tomar más de dos valores, lo que da lugar a la regresión logística multinomial. No obstante, como anteriormente, comenzaremos analizando el caso más simple, en el que Y es binaria.

Finalmente, también se analizarán otros métodos de clasificación basados en la regresión, como los árboles de clasificación y las redes neuronales, que permiten abordar tanto problemas lineales como no lineales de forma flexible y potente.

3.1. Regresión logística para variable respuesta binaria

En este caso, la variable respuesta que se desea clasificar presenta únicamente dos categorías, lo que configura un problema de clasificación binaria. En este contexto, es posible considerar el uso tanto de la regresión lineal como de la regresión logística, ya sea en su versión simple (con una sola variable explicativa) o múltiple (con varias variables explicativas), dependiendo de la complejidad del modelo y de las características de los datos disponibles.

Para explicar estos método en el caso de un único predictor usaremos el conjunto de datos *Default* del paquete ISRL [3] el cual tiene como variable respuesta $default$, que denotaremos por Y, y que indica si un cliente ha incurrido en impago [1] [7] [9]:

$$Y = \begin{cases} 0 = no\ default \\ 1 = \quad default \end{cases}$$

Como variable explicativa tenemos *balance*, que denotaremos por X, que indica el saldo promedio de la tarjeta de crédito del cliente.

Para ilustrar con un ejemplo el funcionamiento de estos métodos de clasificación en la situación de más de un predictor y compararlos con el análisis discriminante consideraremos la misma población que la usada para el análisis discriminante lineal en la cual teníamos individuos pertenecientes a dos clases, 1 y 2. Esos individuos de cada clase seguían distribuciones normales multivariantes $f_1(x)$ y $f_2(x)$ con vectores de medias $\mu_1 = (3, 2)$ y $\mu_2 = (4, 5)$, matriz de covarianzas común $\Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$ y probabilidades de pertenencia a cada clase $\pi_1 = 0.4$ y $\pi_2 = 0.6$.

Usaremos la misma muestra de 800 individuos de la población que la usada en el análisis discriminante lineal con el mismo conjunto de prueba y entrenamiento.

Los cálculos de las fronteras de decisión y las tasas de error de prueba se realizarán usando funciones de R que serán explicadas con detalle en la sección 3.3.

3.1.1. Regresión lineal

Para el conjunto de datos *default* introducido anteriormente podríamos intentar ajustar un modelo de regresión lineal para predecir Y a partir de X, interpretando $\pi(x_0) > 0.5$ como predicción de impago y $\pi(x_0) \leq 0.5$ como ausencia de impago.

Estas probabilidades pueden utilizarse para tomar decisiones en función de un umbral. Por defecto, suele fijarse en 0.5, de modo que si $\pi(x_0) > 0.5$, se predice impago. No obstante, este umbral puede ajustarse según el contexto o el coste asociado a los errores de clasificación. Por ejemplo, si una entidad financiera quiere adoptar un criterio más conservador, podría bajar el umbral a 0,1 para detectar más casos potenciales de impago, aunque ello implique un mayor número de falsos positivos.

Al tener la variable respuesta solo dos categorías, se podría demostrar, que incluso si invertimos la codificación anterior, la regresión lineal produciría las mismas predicciones finales.

En este contexto, los coeficientes estimados $\hat{\beta}$ del modelo lineal:

$$\pi(x_0) = \beta_0 + \beta_1 x_0$$

se interpretan como una estimación de la probabilidad de impago para un cliente con saldo x_0. Sin embargo, usando regresión lineal, algunas de las estimaciones podrían estar fuera del intervalo $[0, 1]$ lo que dificulta interpretarlas como probabilidades. Este comportamiento se puede observar en el siguiente gráfico:

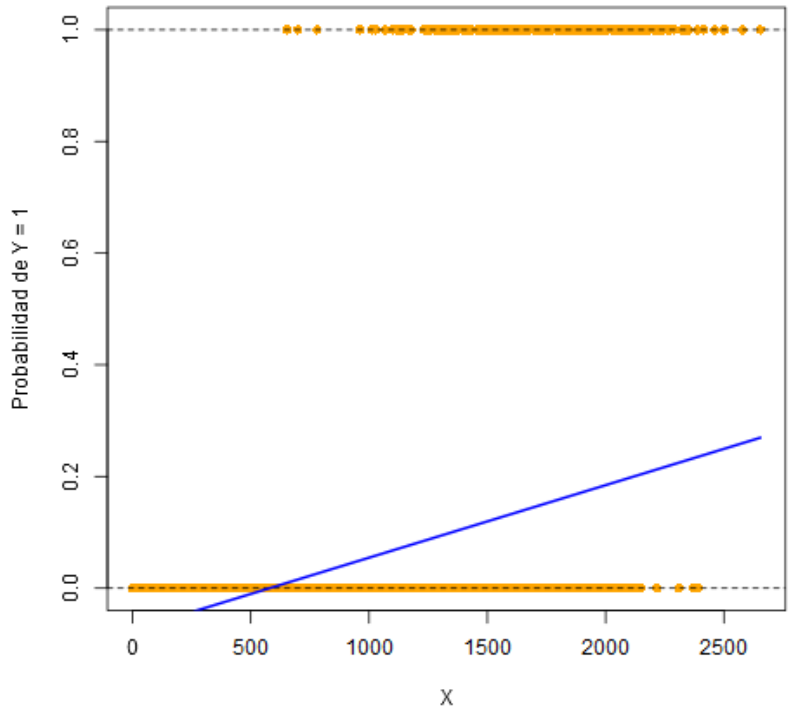

Figura 3.1.1.1: Gráfico de la función de probabilidad de la regresión lineal

A pesar de esta limitación, las predicciones pueden reflejar cierto orden, lo que permite usarlas como una aproximación a la probabilidad. No obstante, la regresión lineal no es un método adecuado para este tipo de problemas, ya que no modela directamente la relación no lineal entre las variables explicativas y la probabilidad de respuesta. Por ello, es preferible utilizar un método específicamente diseñado para variables cualitativas, como es la regresión logística.

Aunque conceptualmente no es una clasificación, se puede considerar una frontera donde, como hemos propuesto, la predicción sea 0.5. Para el caso de tener dos predictores esta frontera se calcularía de la siguiente manera:

$$x_2 = -\frac{\beta_1}{\beta_2} x_1 + \frac{0.5 - \beta_0}{\beta_2}$$

Para ilustrar esta limitación, he ajustado un modelo de regresión lineal para el ejemplo propuesto con más de una variable explicativa y he obtenido la frontera de decisión $x_2 \approx 0.161x_1 - 1.422$, que podemos representar junto con los datos:

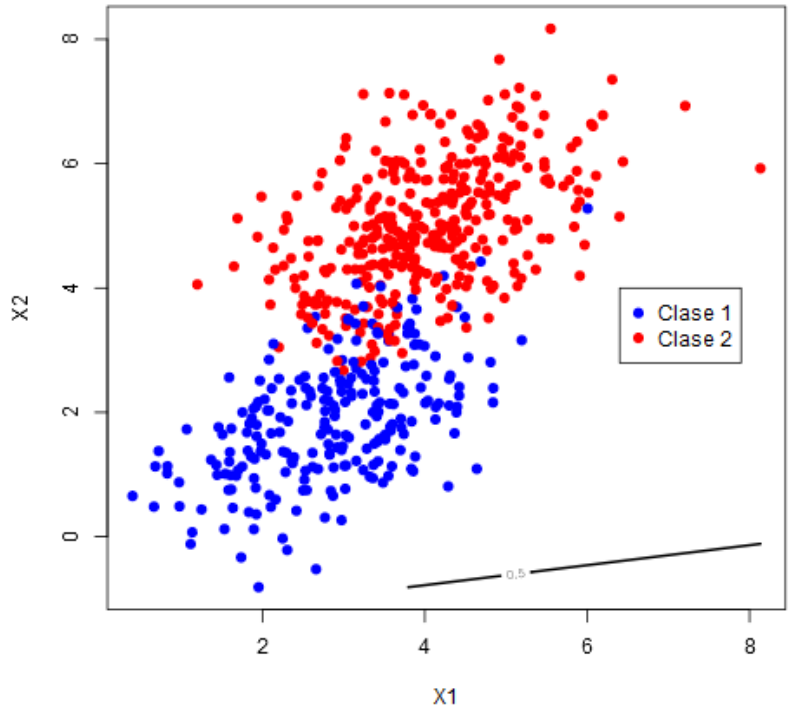

Figura 3.1.1.2: Frontera de decisión de la regresión lineal para el conjunto de entrenamiento de los datos del ejemplo propuesto con más de una variable explicativa

También podemos calcular la tasa de error de prueba y obtenemos el siguiente resultado:

$$t_p = \frac{76}{200} = 0.38$$

Esta coincide con la proporción de individuos pertenecientes a la clase 1 de la muestra de prueba ya que clasifica todos los individuos como clase 2. Por tanto, hemos podido comprobar que la regresión lineal no sirve para este tipo de modelos.

3.1.2. Regresión logística simple

En lugar de ajustar un modelo de regresión lineal a una variable binaria, lo cual conlleva los problemas discutidos anteriormente (como la obtención de valores fuera del intervalo [0,1]), la regresión logística [1] [7] [9] proporciona una alternativa más adecuada al modelar directamente la probabilidad condicional de impago, $\pi(x_0)$, a través de una transformación que garantiza que las predicciones siempre se mantengan en el rango [0,1].

Como ya vimos en la introducción, la regresión logística se basa en la función logística, que relaciona los predictores con el logaritmo de la razón de probabilidades (logit). Esta formulación permite una interpretación probabilística coherente, y evita los inconvenientes de la regresión lineal cuando se trabaja con variables categóricas.

Una vez estimado el modelo, las predicciones se obtienen como probabilidades de pertenencia a la clase $Y = 1$. Como antes usaremos el umbral 0.5, es decir, $\pi(x_0) > 0.5$ como predicción de impago y $\pi(x_0) \leq 0.5$ como ausencia de impago.

A diferencia de la regresión lineal, en la que los parámetros se estiman mediante mínimos cuadrados, la regresión logística emplea el método de máxima verosimilitud, el cual ofrece mejores propiedades estadísticas en contextos de clasificación binaria.

La función de verosimilitud correspondiente, que se maximiza para obtener los estimadores de los parámetros, es:

$$\mathcal{L}(\beta_0, \beta_1) = \prod_{i=1}^{n} (\pi(x_i))^{y_i} (1 - \pi(x_i))^{1-y_i}$$

donde $\pi(x_i)$ es la probabilidad de que el individuo i incurra en impago. Las estimaciones $\hat{\beta}_0$ y $\hat{\beta}_1$ se eligen de forma que esta función se maximice.

El siguiente gráfico ilustra el ajuste del modelo de regresión logística a los datos *Default*:

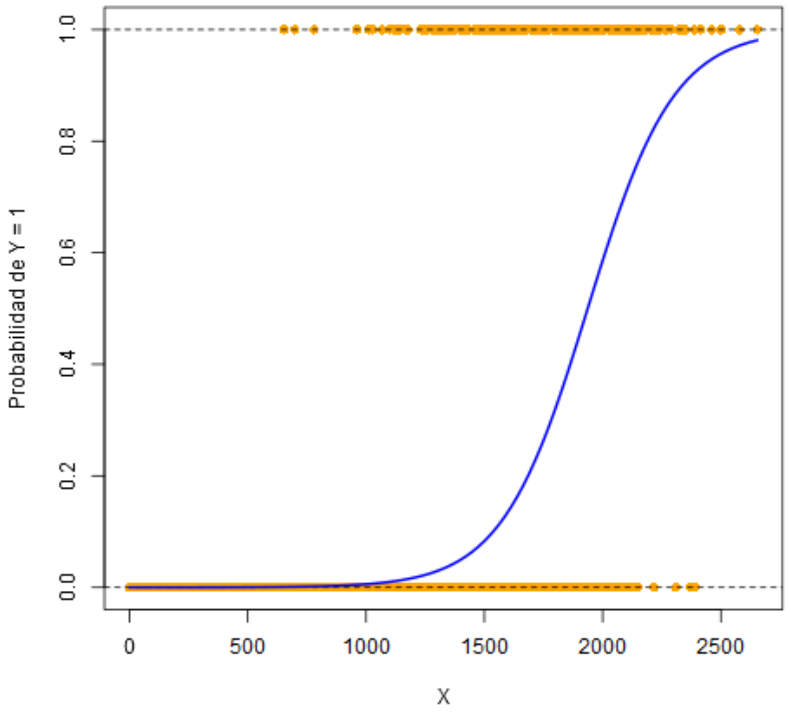

Figura 3.1.2.1: Gráfico de la función de probabilidad de la regresión logística

Aquí podemos ver como las probabilidades son razonables al mantenerse siempre dentro del intervalo [0,1].

3.1.3. Regresión logística múltiple

En la mayoría de situaciones reales, contamos con más de una variable explicativa. Por tanto, resulta necesario extender el modelo de regresión logística simple al caso con múltiples predictores. Esta extensión se conoce como regresión logística múltiple [1] [7] [9]:

$$P(Y = 1|X = x_0) = \pi(x_0) = \frac{1}{1 + e^{-(\beta_0 + \beta^T x_0)}}$$

donde $\beta = (\beta_1, \dots, \beta_p)^T$ es el vector de coeficientes asociados a cada predictor p. Esta expresión generaliza la función logística introducida anteriormente al caso de múltiples variables explicativas.

Como antes, también es habitual expresar el modelo en términos del logaritmo de la razón de probabilidades (logit):

$$\log\left(\frac{\pi(x_0)}{1 - \pi(x_0)}\right) = \beta_0 + \beta^T x_0$$

Al igual que antes se usa el método de máxima verosimilitud para estimar $\beta_0, \beta_1, \dots, \beta_p$.

La regla de clasificación también es la misma que en el caso anterior clasificándose una observación $x_0 \epsilon \mathbb{R}^p$ como clase 1 si $\pi(x_0) > t$ y como clase 0 en caso contrario donde t es un umbral de decisión que se puede ajustar según se desee.

En el caso de tener dos predictores la frontera de decisión con predicción 0.5 sería la siguiente:

$$x_2 = -\frac{\beta_1}{\beta_2} x_1 - \frac{\beta_0}{\beta_2}$$

Para ilustrar lo explicado con un ejemplo, ajuste un modelo de regresión logística múltiple al conjunto de entrenamiento del ejemplo propuesto y obtuve la frontera de decisión $x_2 \approx 0.174x_1 + 2.726$ que podemos representar junto con los datos:

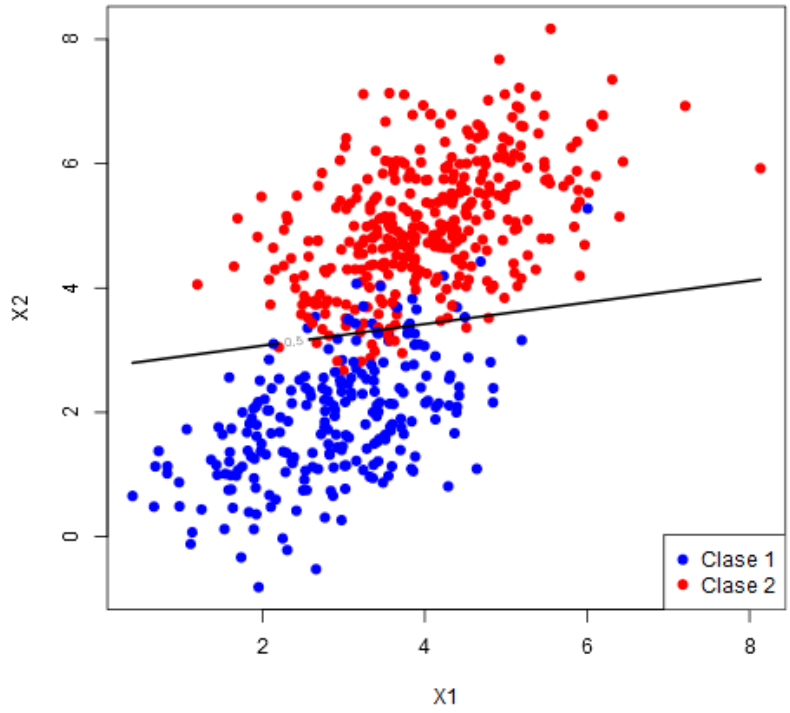

Figura 3.1.3.1: Frontera de decisión de la regresión logística múltiple para el conjunto de entrenamiento de los datos del ejemplo propuesto con más de una variable explicativa

También podemos calcular la tasa de error de prueba y obtenemos el siguiente resultado:

$$t_p = \frac{13}{200} = 0.065$$

Como podemos ver la frontera de decisión es muy similar a la de LDA y la tasa de error idéntica. Por tanto, podríamos decir que funciona prácticamente igual que el LDA.

3.2. Regresión logística para variable respuesta con más de dos grupos

En este caso, la variable respuesta que se desea clasificar presenta más de dos categorías, lo que implica un problema de clasificación multiclase. Para abordarlo, como veremos a continuación, solo es posible usar regresión logística multinomial [1] [7] [9].

Para ilustrar con un ejemplo el funcionamiento de estos métodos de clasificación en esta situación y compararlos con el análisis discriminante consideraremos la misma población que la usada para el análisis discriminante con variable respuesta con más de dos categorías. Además usaremos la misma muestra de 800 individuos de la población con el mismo conjunto de prueba y entrenamiento. Los datos están detallados en la sección 2.2.

Los cálculos de las fronteras de decisión y las tasas de error de prueba se realizarán usando funciones de R que serán explicadas con detalle en la sección 3.3.

3.2.1. Regresión lineal

Supongamos que queremos predecir el nivel de fidelidad de los clientes de una empresa a partir de ciertas características, como su edad, la frecuencia de compra u otros factores. En este caso, la variable respuesta Y puede tomar tres posibles niveles: bajo, medio y alto. Una posible codificación sería:

$$Y = \begin{cases} 1 & si\ el\ nivel\ de\ fidelidad\ es\ bajo \\ 2 & si\ el\ nivel\ de\ fidelidad\ es\ medio \\ 3 & si\ el\ nivel\ de\ fidelidad\ es\ alto \end{cases}$$

A continuación, podríamos intentar ajustar un modelo de regresión lineal mediante mínimos cuadrados, utilizando como variables explicativas un conjunto de predictores $X = (X_1, \dots, X_p)^T$.

Lamentablemente, esta codificación implica un orden en los resultados, colocando el nivel medio entre el bajo y el alto (lo cual en este caso es razonable, pero en otro podría no serlo como por ejemplo en el caso de tener 3 diagnósticos de enfermedad de un paciente), e insistiendo en que la diferencia entre el nivel bajo y el nivel medio es la misma que entre el nivel medio y el alto. Si consideramos que estos valores siguen el orden natural, bajo, medio y alto y consideramos que la diferencia entre bajo y medio y medio y alto es la misma entonces esta codificación 1, 2, 3 sería razonable.

En la práctica no suele ser así y si cambiamos el orden de la codificación como resultado tendríamos una relación totalmente diferente entre las tres condiciones. Por tanto no existe una manera razonable de convertir una variable de respuesta cualitativa con más de dos niveles en una respuesta cuantitativa adecuada para la regresión lineal.

3.2.2. Regresión logística multinomial

En la situación descrita anteriormente, donde la variable respuesta Y representa el nivel de fidelidad de un cliente con categorías bajo, medio y alto, resulta más apropiado utilizar un modelo de regresión logística multinomial. Este método generaliza la regresión logística al caso en que la variable respuesta tiene más de dos categorías ($K > 2$), permitiendo modelar problemas de clasificación multiclase de forma adecuada.

Para construir este modelo, lo primero que se hace es seleccionar una de las K clases como clase de referencia. Esta elección se realiza para evitar la sobreparametrización del modelo y facilitar la interpretación de los coeficientes. Sin pérdida de generalidad, se suele tomar como clase base la última categoría, es decir, $Y = K$.

Dado un individuo con valores de predictores $x_0 = (x_1, \dots, x_p)^T$, las probabilidades condicionales de que pertenezca a una clase $k \in \{1, \dots, K-1\}$ están dadas por:

$$P(Y = k|X = x_0) = \pi_k(x_0) = \frac{e^{\beta_{k0} + \beta_k^T x_0}}{1 + \sum_{l=1}^{K-1} e^{\beta_{l0} + \beta_l^T x_0}} \quad para\ k = 1, \dots, K-1$$

La probabilidad de que el individuo pertenezca a la clase de referencia $Y = K$ se calcula como:

$$P(Y = K|X = x_0) = \pi_K(x_0) = \frac{1}{1 + \sum_{l=1}^{K-1} e^{\beta_{l0} + \beta_l^T x_0}}$$

También se puede expresar el modelo en términos de las log-odds entre cada clase k y la clase de referencia:

$$\log\left(\frac{P(Y = k|X = x_0)}{P(Y = K|X = x_0)}\right) = \beta_{k0} + \beta_k^T x_0$$

Este enfoque permite modelar directamente las razones de probabilidades logarítmicas entre cada clase y la clase base como combinaciones lineales de los predictores.

La elección de la clase de referencia no afecta a las predicciones del modelo, ni a las probabilidades logarítmicas entre pares de clases, pero sí influye en la interpretación de los coeficientes estimados, ya que estos están definidos respecto a dicha clase.

Como antes, los coeficientes del modelo se estiman mediante el método de máxima verosimilitud. Para un conjunto de entrenamiento de n observaciones (x_i, y_i), con $y_i \in \{1, \dots, K\}$, la función de verosimilitud es:

$$\mathcal{L}(\{\beta_k\}) = \prod_{i=1}^{n} \prod_{k=1}^{K} \pi_k(x_i)^{1_{\{y_i = k\}}}$$

donde $\pi_k(x_i)$ es la probabilidad de que la observación i pertenezca a la clase k y $1_{\{y_i = k\}}$ es la función indicadora que vale 1 si y_i=k, y 0 en caso contrario.

Una alternativa es la codificación *softmax*, común en aprendizaje automático. En este enfoque, no se elige una clase de referencia, y se estima un conjunto completo de coeficientes para las K clases:

$$\pi_k(x_0) = \frac{e^{\beta_{k0} + \beta_k^T x_0}}{\sum_{l=1}^{K} e^{\beta_{l0} + \beta_l^T x_0}}$$

Por lo tanto, estamos estimando coeficientes para las *K* clases. No es difícil ver que, la razón de probabilidades logarítmicas entre dos clases cualesquiera k y k' es:

$$\log\left(\frac{P(Y=k|X=x_0)}{P(Y=k'|X=x_0)}\right) = (\beta_{k0} - \beta_{k'0}) + (\beta_k - \beta_{k'})^T x_0$$

Esta formulación también maximiza una log-verosimilitud similar, pero ahora con un parámetro libre para cada clase. Aunque la parametrización es distinta, las predicciones del modelo y las decisiones de clasificación siguen siendo equivalentes a las del modelo con clase de referencia.

La regla de clasificación en este caso es un poco diferente a las anteriores ya que ahora al tener la variable respuesta más de dos categorías tenemos que se clasifica una observación $x_0 \epsilon \mathbb{R}^p$ como clase k si $\pi_k(x_0) = \max_{j\epsilon\{1,\dots,K\}} \pi_j(x_0)$, es decir, se le asigna la clase con mayor probabilidad estimada.

Con respecto a las fronteras de decisión, como pasaba en el análisis discriminante, se obtiene una frontera de decisión para cada par de clases. Cuando tenemos dos predictores la frontera de decisión con predicción 0.5 entre las clases k y l sería la siguiente:

$$x_2 = -\frac{\beta_{k1} - \beta_{l1}}{\beta_{k2} - \beta_{l2}} x_1 - \frac{\beta_{k0} - \beta_{l0}}{\beta_{k2} - \beta_{l2}}$$

Para ilustrar lo explicado con un ejemplo, ajuste un modelo de regresión logística multinomial al conjunto de entrenamiento del ejemplo propuesto y obtuve la siguiente frontera de decisión:

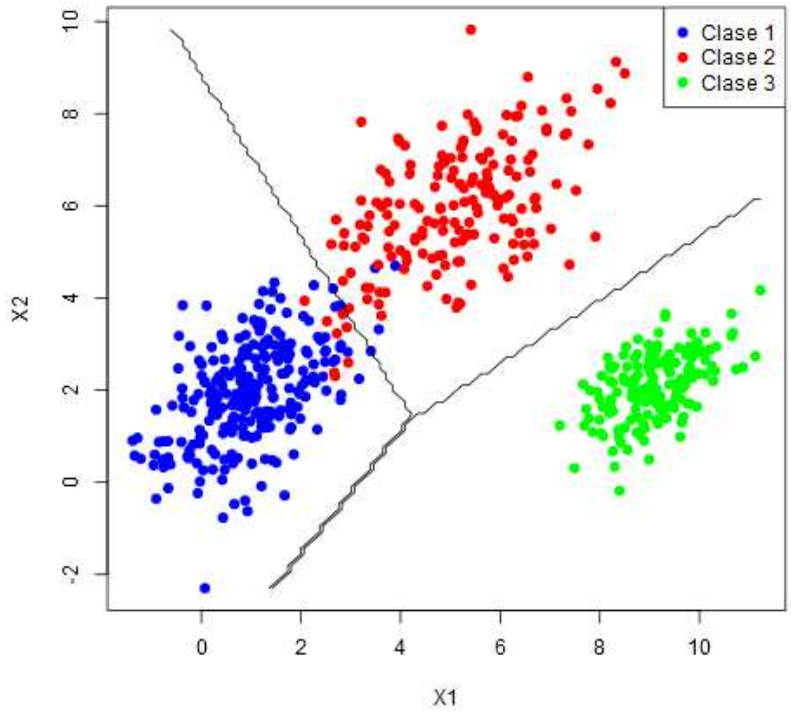

Figura 3.2.2.1: Frontera de decisión de la regresión logística multinomial para el conjunto de entrenamiento de los datos del ejemplo propuesto con variable respuesta con más de dos categorías

También podemos calcular la tasa de error de prueba y obtenemos el siguiente resultado:

$$t_p = \frac{1}{200} = 0.005$$

Como podemos ver funciona igual que el QDA y el no paramétrico y ligeramente mejor que el LDA.

3.3. Funciones en R para la regresión logística

Las tres principales funciones en R [8] para realizar todos los tipos de regresión que hemos visto son:

- lm (No es una función para hacer regresión logística): Función del paquete stats [8] (paquete base del R), útil para ajustar modelos de regresión lineal. En este caso se emplea con una variable respuesta binaria, aunque no es el enfoque más adecuado para este tipo de datos como ya vimos, y con una o más variables explicativas.

- glm: Función del paquete stats [8] (paquete base del R) que permite ajustar modelos lineales generalizados. Por tanto, como se ha explicado en la parte teórica, esta función resulta útil para ajustar modelos de regresión logística con variable respuesta binaria y con una o más variables explicativas, siempre que se especifique adecuadamente la familia y la función de enlace.
- multinom: Función del paquete nnet [10], útil para ajustar modelos de regresión logística multinomial (variable respuesta con dos o más categorías y con una o más variables explicativas).

Estas funciones han sido utilizadas para calcular los clasificadores correspondientes a la regresión lineal (*lm*), regresión logística múltiple (*glm*) y regresión logística multinomial (*multinom*) en la sección anterior.

Al igual que en el caso del análisis discriminante, las funciones *lm*, *glm* y *multinom* requieren como argumento principal una fórmula en la que se indica la variable respuesta y los predictores, con la notación habitual:

variable respuesta ~ predictor1 + predictor2 +…

Las variables mencionadas en la fórmula son tomadas, por defecto, del entorno global (*.GlobalEnv*), aunque se puede indicar explícitamente el conjunto de datos a usar mediante el argumento *data*.

Además de los argumentos comunes, la función *glm* dispone del argumento *family*, que se utiliza para especificar la familia de distribuciones del modelo. Por defect, usa la familia gaussian (gaussiana), lo que equivale a ajustar un modelo de regresión lineal mediante la función *lm*. Sin embargo, como dijimos, también admite otras familias, estas son: binomial, Gamma, inverse.gaussian, poisson, quasi, quasibinomial y quasipoisson. Para regresión logística se debe especificar *family = binomial*. Esta configuración aplica por defecto la función de enlace logit, que corresponde a la formulación teórica vista anteriormente. Si se desea modificar la función de enlace, puede especificarse mediante el argumento *link* dentro de *binomial*, por ejemplo: *binomial(link = "logit").* Además del enlace logit, existen otras funciones de enlace disponibles, entre las que destacan "*probit*" (usa la función inversa de la normal estándar), "*cloglog*" (complemenary log-log) y "*cauchit*" (basada en la función de distribución Cauchy), entre otras.

En el caso de la función *multinom*, no se especifica explícitamente la familia ni la función de enlace, ya que está diseñada exclusivamente para modelos logísticos multinomiales. Internamente utiliza una generalización del enfoque de *glm* para variables con más de dos categorías.

Para calcular el clasificador con los distintos métodos visto en las secciones anteriores, apliqué cada una de las funciones correspondientes al conjunto de entrenamiento.

Estas funciones, al igual que las vistas en el análisis discriminante, no devuelven explícitamente las fronteras de decisión, pero sí proporcionan los coeficientes del modelo y permiten obtener predicciones, lo que es suficiente para representarlas gráficamente.

Para calcular el error de prueba se utilizó el método genérico *predict* del paquete base *stats* [8], que es el que ya vimos en el análisis discriminante lineal. Ahora, al aplicarlo sobre los objetos de las clases *lm*, *glm* y *ksvm*, *multinom* ejecuta internamente las funciones específicas *predict.lm*, *predict.glm* y *predict.multinom*, respectivamente.

Al aplicar el método *predict* a los distintos modelos junto con los datos del conjunto de entrenamiento, se obtienen distintos resultados. La función *predict.lm* devuelve valores que se interpretan como probabilidades de pertenencia a la clase 2. No obstante, estos valores pueden encontrarse fuera del intervalo (0,1), por lo que carecen de interpretación probabilística válida. En el caso de *predict.glm*, es necesario añadir el argumento *type="response"* para que devuelva las probabilidades estimadas de pertenencia a la clase 2. Por su parte, la función *predict.multinom* devuelve directamente las clases predichas de cada observación.

Para determinar las clases predichas a partir de las probabilidades generadas por *predict.lm* y *predict.glm*, se aplicó la regla de clasificación explicada anteriormente: asignar la clase 2 a las observaciones cuya probabilidad de pertenencia a esta categoría es mayor que 0.5, y la clase 1 en caso contrario.

Quedándonos con las predicciones, el error de prueba se calculó comparando las clases verdaderas con las predichas.

3.4. Otros métodos de clasificación basados en la regresión

Además del análisis discriminante y la regresión logística, existen otros métodos ampliamente utilizados para clasificar individuos en grupos previamente definidos. En esta sección, se explorarán algunas de estas técnicas alternativas, centrándonos específicamente en los árboles de clasificación y las redes neuronales, debido a su eficacia y versatilidad en diversos contextos de clasificación.

3.4.1. Árboles de clasificación

Los árboles de decisión son métodos de clasificación que se fundamentan en la estratificación del espacio clasificador en un conjunto de regiones simples y no solapadas [1]. Para clasificar una nueva observación, se considera la región a la que pertenece dentro del espacio estratificado y, en función de ello, se asigna un valor de respuesta basado en la media (en regresión) o la moda (en clasificación) de las observaciones de entrenamiento contenidas en dicha región. Estos métodos reciben el nombre de árboles de decisión porque el conjunto de reglas de partición utilizadas para dividir el espacio clasificador puede representarse de manera natural mediante una estructura jerárquica en forma de árbol.

Estos métodos se pueden aplicar para problemas de regresión y clasificación (árboles de regresión y árboles de clasificación), con la única diferencia de que en los primeros se utilizan para predecir una respuesta cuantitativa y en los segundos una respuesta cualitativa. En esta sección nos centraremos en los segundos, en los que predecimos que cada observación pertenece a la clase que ocurre con mayor frecuencia entre las observaciones de entrenamiento en la región a la que pertenece.

Para construir un árbol de clasificación utilizamos la división binaria recursiva usando como criterio para realizar las divisiones binarias la tasa de error de clasificación que se define como la fracción de observaciones de entrenamiento en esa región que no pertenecen a la clase más común:

$$E = 1 - \max_k(\hat{p}_{mk})$$

Aquí, $\hat{p}_{mk}$ representa la proporción de observaciones de entrenamiento en la región m que pertenecen a la clase k. Como la tasa de error de clasificación resulta no ser suficientemente sensible para el crecimiento del árbol, en la práctica se prefieren otras dos medidas.

El índice de Gini:

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

Es una medida de la varianza total entre las K clases. Este índice se considera una medida de la pureza del nodo: Un valor pequeño indica que un nodo contiene predominantemente observaciones de una sola clase.

La entropía:

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$$

Como podemos ver la entropía tomará un valor pequeño si el nodo m es puro. De hecho, se ha comprobado que el índice de Gini y la entropía con bastante similares numéricamente.

Para la construcción del árbol se suele utilizar generalmente uno de estos dos métodos aunque si el objetivo es la precisión de predicción del árbol podado final, la tasa de error de clasificación es preferible.

3.4.2. Redes neuronales

Las redes neuronales constituyen la base fundamental del denominado *Deep Learning*. Este tipo de modelos toma como entrada un vector de variables explicativas $X = (X_1, X_2, \dots, X_p)$ y construye una función no lineal $f(X)$ con el objetivo de predecir la respuesta Y [1]. En función de su arquitectura, una red neuronal puede clasificarse como monocapa o multicapa, según cuente con una única capa oculta o con múltiples capas ocultas, respectivamente. Cuantas más capas tenga, mayor será su capacidad para capturar relaciones complejas entre las variables.

Las redes neuronales de una sola capa oculta son aquellas que cuentan únicamente con una única capa intermedia entre la capa de entrada y la de salida. Para una mejor comprensión, la siguiente figura ilustra una red neuronal simple de tipo *feed-forward* diseñada para modelar una respuesta cuantitativa a partir de $p = 4$ variables explicativas:

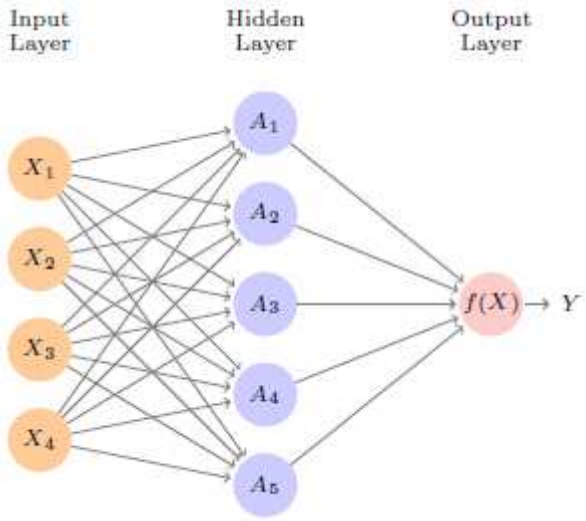

Figura 3.4.2.1: Red neuronal simple de tipo *feed-forward* diseñada para modelar una respuesta cuantitativa a partir de $p = 4$ variables explicativas

Las cuatro características $X_1, \dots, X_4$ forman las unidades en la capa de entrada que se conectan con cada una de las unidades de la capa oculta K como podemos ver con las flechas. El modelo de red neuronal tiene la forma:

$$f(X) = \beta_0 + \sum_{k=1}^{K} \beta_k h_k(X)$$

$$= \beta_0 + \sum_{k=1}^{K} \beta_k \, g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j\right)$$

Se calcula en dos pasos. Primero, se calculan las K activaciones A_k,$k = 1, \dots, K$, en la capa oculta como funciones de las características de entrada $X_1, \dots, X_p$:

$$A_k = h_k(X) = g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j\right)$$

Donde $g(z)$ es una función de activación no lineal especificada de antemano. Estas K activaciones de la capa oculta se transmiten luego a la capa de salida, dando como resultado:

$$f(X) = \beta_0 + \sum_{k=1}^{K} \beta_k A_k$$

que es un modelo de regresión lineal en las $K = 5$ activaciones. Todos los parámetros $\beta_0, \dots, \beta_k$ y $w_{10}, \dots, w_{Kp}$necesitan ser estimados por los datos.

La función de activación que se usa actualmente es la $ReLU\ (rectified\ linear\ unit)$, que toma la forma:

$$g(z) = (z)_+ = \begin{cases} 0 & si\ z < 0 \\ z & en\ otro\ caso \end{cases}$$

La no linealidad en la función de activación $g(\cdot)$ es esencial, ya que sin ella, el modelo $f(X)$ colapsaría en un modelo lineal simple en $X_1, \ldots, X_p$. Tener una función de activación no lineal permite además que el modelo capture no linealidades complejas y efectos de interacción.

Ajustar una red neuronal requiere estimar los parámetros desconocidos. Para una respuesta cuantitativa, típicamente se utiliza la pérdida de error cuadrático, de modo que los parámetros se eligen para minimizar:

$$\sum_{i=1}^{n} \left(y_i - f(x_i)\right)^2$$

Las redes neuronales modernas suelen incorporar múltiples capas ocultas y, con frecuencia, un gran número de unidades por capa. El uso de varias capas, incluso si cada una tiene un tamaño moderado, permite representar funciones complejas y facilita significativamente el proceso de aprendizaje para encontrar una solución eficaz. La siguiente figura muestra una red neuronal multicapa utilizada para modelar una variable respuesta cualitativa con 10 categorías, a partir de p variables explicativas:

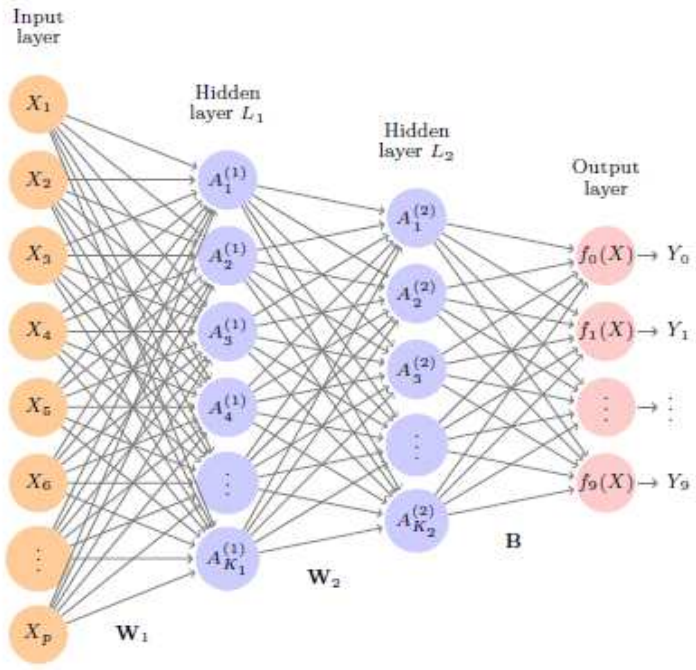

Figura 3.4.2.2: Red neuronal multicapa utilizada para modelar una variable respuesta cualitativa con 10 categorías, a partir de p variables explicativas

El cálculo de las K activaciones en la primera capa oculta se realiza como antes:

$$A_k^{(1)} = h_k^{(1)}(X)$$
$$= g\left(w_{k0}^{(1)} + \sum_{j=1}^{K_1} w_{kj}^{(1)} X_j\right)$$

Para $k = 1, \ldots, K_1$. La segunda capa oculta trata las activaciones $A_k^{(1)}$ de la primera capa oculta como entradas y calcula nuevas activaciones:

$$A_l^{(2)} = h_l^{(2)}(X)$$
$$= g\left(w_{l0}^{(2)} + \sum_{k=1}^{K_1} w_{lk}^{(2)} A_k^{(1)}\right)$$

Para $l = 1, \ldots, K_2$. Los superíndices adicionales de las activaciones y los pesos sirven para indicar a que capa pertenecen (1 o 2). La notación W_1 en la anterior figura representa toda la matriz de pesos que alimenta desde la capa de entrada hasta la primera capa oculta L_1. Cada elemento $A_k^{(1)}$ alimenta a la segunda capa oculta L_2 a través de la matriz de pesos W_2.

Finalmente, llegamos a la capa de salida, donde ahora tenemos diez respuestas en lugar de una. El primer paso es calcular diez modelos lineales diferentes de manera similar a como lo hacíamos en el caso simple:

$$Z_m = \beta_{m0} + \sum_{l=1}^{K_2} \beta_{ml} h_l^{(2)}(X)$$
$$= \beta_{m0} + \sum_{l=1}^{K_2} \beta_{ml} A_l^{(2)}$$

Para $m = 0, 1, \ldots, 9$. La matriz de pesos B almacena los pesos que alimentan la capa de salida. Si estas fueran todas respuestas cuantitativas separadas, simplemente estableceríamos $f_m(X) = Z_m$ y terminaríamos. Sin embargo, queremos que nuestras estimaciones representen probabilidades de clase $f_m(X) = P(Y = m|X)$.

Entonces usamos la función de activación especial *softmax*:

$$f_m(X) = Pr(Y = m|X) = \frac{e^{Z_m}}{\sum_{l=0}^{9} e^{Z_l}}$$

Para $m = 0, 1, \ldots, 9$. Esto asegura que las 10 respuestas se comporten como probabilidades (No negativas y cuya suma es igual a uno). El modelo calculará una probabilidad para cada una de las 10 clases para que luego el clasificador asigne la respuesta a la clase con la probabilidad más alta.

Para entrenar esta red, dado que la respuesta es cualitativa, buscamos la estimación de los coeficientes que minimicen la log-verosimilitud multinomial negativa:

$$-\sum_{i=1}^{n}\sum_{m=0}^{9} y_{im}\log\left(f_m(x_i)\right)$$

También conocida como entropía cruzada. En esta fórmula $y_{im} = 1$ si la clase verdadera de la observación y es m e $y_{im} = 0$, en caso contrario. $f_m(x_i)$ es la probabilidad de pertenencia de la observación i a la clase m.

4. EJEMPLO DE CLASIFICACIÓN DE DÍGITOS MANUSCRITOS

Con el objetivo de evaluar y comparar el rendimiento de los distintos métodos de clasificación presentados en los apartados anteriores, se plantea un ejemplo más amplio y realista: la clasificación de dígitos manuscritos. Para ello, se utilizará el conjunto de datos conocido como ZIP code data [9], que incluye un archivo de entrenamiento (.train) y un archivo de prueba (.test). Ambos archivos han sido convertidos a formato .txt, conteniendo 7291 observaciones en el conjunto de entrenamiento y 2007 en el de prueba.

Cada observación representa un dígito manuscrito en escala de grises, codificado mediante 257 variables: la primera corresponde al número representado (la clase), y las 256 restantes describen el nivel de gris de cada píxel que compone la imagen, en una matriz de 16×16 píxeles. La escala de grises está representada por valores entre -1 y 1 siendo el -1 el color blanco y el 1 el negro. A la hora de hacer la representación gráfica es necesario invertir los colores ya que plot() interpreta los valores numéricos al revés (en concreto, a través del paquete imager y su función plot.cimg), siguiendo la convención común en imágenes en escala de grises en [0,1] en los que 0 es negro y 1 blanco. En resumen siempre considera el número menor negro y el mayor blanco.

La representación gráfica de una de estas observaciones es la siguiente:

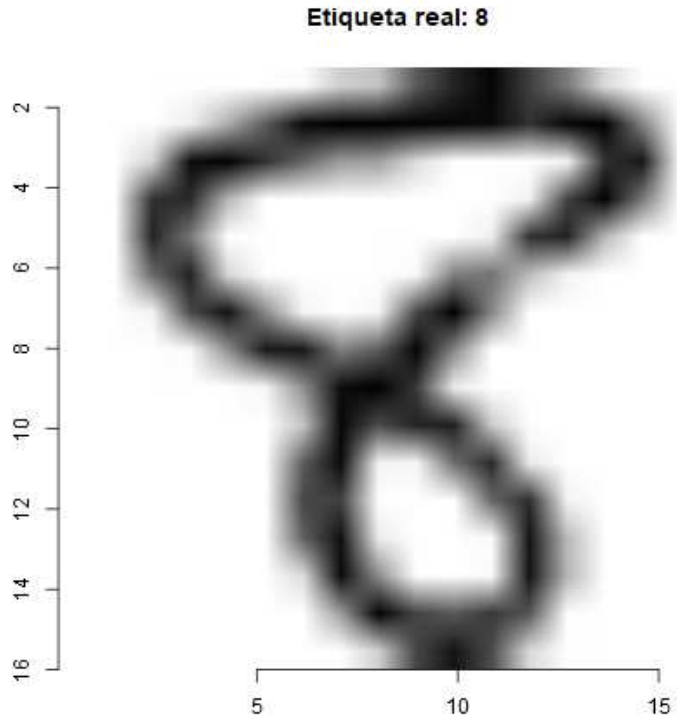

Figura 4.1: Dígito manuscrito en escala de grises

El objetivo principal de este ejemplo es comparar la eficacia de distintos métodos de clasificación aplicados a este problema. Se propondrán tres enfoques:

- Entrenamiento de modelos de clasificación utilizando el conjunto completo de variables originales.
- Entrenamiento tras realizar un análisis de componentes principales (PCA) para reducir la dimensionalidad de los datos.
- Entrenamiento utilizando una versión simplificada de los datos, cuya construcción se explicará más adelante.

4.1. Clasificación utilizando el conjunto completo de variables originales

Una vez presentado el conjunto de datos, se procede al entrenamiento de distintos modelos de clasificación con el objetivo de predecir la clase a la que pertenece cada dígito manuscrito. Para ello, se construirá un modelo utilizando cada uno de los tres enfoques principales estudiados previamente: el análisis discriminante, el análisis discriminante no paramétrico y la regresión logística. Esta comparación permitirá evaluar el desempeño relativo de cada técnica en un problema real de alta dimensión.

4.1.1. Análisis discriminante

Como se explicó anteriormente, existen dos variantes principales del análisis discriminante: el análisis discriminante lineal (LDA) y el cuadrático (QDA). Para este ejemplo, se opta por utilizar LDA, dado que, como se mencionó previamente, este método suele ofrecer mejores resultados que QDA cuando se dispone de un número relativamente reducido de observaciones de entrenamiento, como es el caso. En tales situaciones, reducir la varianza del modelo es fundamental, y la suposición de una matriz de covarianzas común entre las clases resulta razonable.

Para realizar el análisis se entrena un modelo LDA utilizando el conjunto de datos de entrenamiento ya transformado y posteriormente se evalúa el modelo aplicándolo primero al propio conjunto de datos de entrenamiento y posteriormente al conjunto de datos de test. Esto último se hace para ver el grado de sobreajuste del modelo a los datos con los que se construye.

En el conjunto de datos de entrenamiento el resultado obtenido es un error de 0.0620, lo que implica una precisión de clasificación del 93.80%.

En el conjunto de datos de prueba el resultado obtenido es un error de 0.1146, lo que implica una precisión de clasificación del 88.54%, un desempeño bastante satisfactorio considerando la naturaleza y complejidad del problema. Como podemos ver este método sobreajusta pero no mucho.

4.1.2. Análisis discriminante no paramétrico

Como se comentó anteriormente, cuando no se cumplen las suposiciones paramétricas del análisis discriminante tradicional, el rendimiento del clasificador puede verse gravemente afectado. Por este motivo, surge la necesidad de utilizar enfoques no paramétricos, que no requieren asumir una distribución específica para las variables.

En este ejemplo, se implementará un modelo discriminante no paramétrico utilizando un kernel radial (gaussiano), dado que esta opción es una de las más comunes y eficaces para abordar problemas de clasificación no lineal, como es el caso del reconocimiento de dígitos manuscritos.

Para realizar el análisis se entrena un modelo no paramétrico utilizando el conjunto de datos de entrenamiento ya transformado y posteriormente se evalúa el modelo aplicándolo primero al propio conjunto de datos de entrenamiento y posteriormente al conjunto de datos de test. Esto último se hace para ver el grado de sobreajuste del modelo a los datos con los que se construye.

En el conjunto de datos de entrenamiento el resultado obtenido es un error de 0.0089, lo que implica una precisión de clasificación del 99.11%.

El resultado obtenido es un error de prueba de 0.0588, lo que implica una precisión de clasificación del 94.12%, un desempeño bastante mejor que el del análisis discriminante lineal. Como podemos ver este método sobreajusta pero no mucho.

4.1.3. Regresión logística

Como se mencionó anteriormente, la regresión logística clásica resulta especialmente adecuada cuando la variable respuesta es binaria. Sin embargo, en este caso nos enfrentamos a un problema de clasificación con diez categorías distintas, por lo que este enfoque no es aplicable directamente. Ante esta situación, es necesario recurrir a la regresión logística multinomial, que extiende el modelo logístico estándar para abordar problemas con más de dos clases.

Para realizar el análisis se entrena un modelo de regresión logística multinomial utilizando el conjunto de datos de entrenamiento ya transformado y posteriormente se evalúa el modelo aplicándolo primero al propio conjunto de datos de entrenamiento y posteriormente al conjunto de datos de test. Esto último se hace para ver el grado de sobreajuste del modelo a los datos con los que se construye.

En el conjunto de datos de entrenamiento el resultado obtenido es un error de 0.0001, lo que implica una precisión de clasificación del 99.99%.

En el conjunto de datos de prueba el resultado obtenido es un error de 0.1066, lo que implica una precisión de clasificación del 89.34%, un desempeño ligeramente mejor que el del análisis discriminante lineal pero mejor que el del análisis discriminante no paramétrico. Como podemos ver este método sobreajusta mucho los datos.

4.2. Clasificación reduciendo la dimensionalidad de los datos usando PCA

Es razonable suponer que no todos los píxeles que conforman un dígito manuscrito son igualmente relevantes para su clasificación. Por ello, surge la idea de reducir la dimensionalidad de los datos, conservando únicamente aquellas variables que expliquen un porcentaje significativo de la varianza total. El método más comúnmente utilizado para este propósito es el Análisis de Componentes Principales (PCA), que será el enfoque adoptado en este trabajo.

El objetivo principal de este ejercicio es evaluar si la reducción de dimensionalidad mediante PCA compensa el esfuerzo computacional extra, tanto en términos de tiempo de entrenamiento como en la precisión de clasificación obtenida. En concreto, se medirá el tiempo de ejecución necesario para clasificar los dígitos de prueba utilizando los tres métodos estudiados (análisis discriminante, análisis no paramétrico y regresión logística), primero con el conjunto completo de datos y luego tras aplicar PCA. Además del tiempo, se compararán también los errores de entrenamiento y de prueba obtenidos en ambos casos. Nos quedaremos con las componentes principales que expliquen un 95% de la varianza total.

Los tiempos de ejecución son los siguientes:

Modelo	Tiempo de ejecución
LDA	5.79 segundos
LDA con PCA	5.91 segundos
No paramétrico	63.19 segundos
No paramétrico con PCA	70.08 segundos
Regresión multinomial	32.05 segundos
Regresión multinomial con PCA	19.29 segundos

Tabla 4.2.1: Tabla de los tiempos de ejecución para los distintos clasificadores para la versión normal y reducida con PCA

Los errores de prueba son los siguientes:

Modelo	Errores de entrenamiento	Errores de prueba
LDA	0.0620	0.1146
LDA con PCA	0.0738	0.1166
No paramétrico	0.0089	0.0588
No paramétrico con PCA	0.0058	0.0747
Regresión multinomial	0.0001	0.1066
Regresión multinomial con PCA	0.0144	0.0942

Tabla 4.2.2: Tabla de los errores de entrenamiento y prueba para los distintos clasificadores para la versión normal y reducida con PCA

A partir de los resultados obtenidos, se observa que la aplicación de PCA no resulta eficiente en el caso del análisis discriminante, ya que se produce un ligero incremento tanto en el tiempo de ejecución como en el error de prueba. En cuanto al error de entrenamiento vemos que sigue sin sobrestimar los datos.

En el caso de la regresión logística multinomial, la reducción de dimensionalidad mediante PCA sí ofrece beneficios claros, ya que se aprecia una disminución significativa en el tiempo de ejecución y una mejora leve en el error de clasificación. Además vemos como ahora no sobreajusta tanto los datos.

En conclusión, la reducción de dimensionalidad a través de PCA resulta especialmente útil en la regresión multinomial, mientras que su aplicación en los modelos basados en análisis discriminante (paramétrico y no paramétrico) no solo no aporta mejoras, sino que puede perjudicar ligeramente el rendimiento del modelo.

4.3. Clasificación reduciendo la dimensionalidad de los datos usando una técnica propia

Como hemos podido comprobar en el análisis anterior, la reducción de la dimensionalidad mediante PCA resultó especialmente eficaz en el caso de la regresión logística multinomial. Por ello, a continuación, se plantea una reducción aún más drástica de la dimensionalidad con el objetivo de evaluar si esta mejora puede mantenerse e incluso ampliarse, y observar también el comportamiento de los otros dos métodos bajo esta nueva transformación.

Para llevar a cabo esta reducción extrema, se propuso un primer paso de binarización de los píxeles, transformando la imagen de escala de grises a blanco y negro. Específicamente, se asignó el valor 0 (blanco) a los píxeles con intensidad menor o igual a 0, y el valor 1 (negro) a los superiores a dicho umbral. Esto simplifica la representación gráfica de los dígitos y reduce notablemente la complejidad del conjunto de datos:

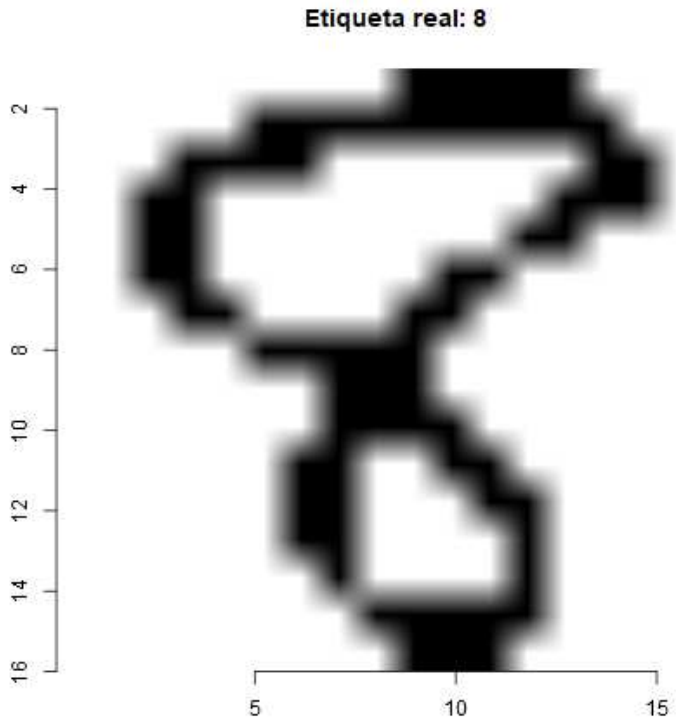

Figura 4.3.1: Dígito manuscrito en escala binaria

A partir de esta nueva representación, se propuso reducir el número de variables explicativas de 256 a solo 32. La idea fue capturar patrones estructurales relevantes mediante una medida simple pero representativa: el número de secuencias de ceros presentes en cada fila y columna de la matriz 16x16 que conforma cada dígito. Por ejemplo, en la siguiente fila o columna el número de secuencias de 1's sería 2:

0,0,1,1,1,1,0,0,0,1,1,1,1,0,0,0

Con estas treinta y dos nuevas variables, se procedió a repetir el experimento anterior aplicando los tres métodos de clasificación.

Los tiempos de ejecución son los siguientes:

Modelo	Tiempo de ejecución
LDA	5.79 segundos
LDA con reducción	14.95 segundos
No paramétrico	63.19 segundos
No paramétrico con reducción	30.86 segundos
Regresión multinomial	32.05 segundos
Regresión multinomial con reducción	19.55 segundos

Tabla 4.3.1: Tabla de los tiempos de ejecución para los distintos clasificadores para la versión normal y reducida

Los errores de prueba son los siguientes:

Modelo	Errores de entrenamiento	Errores de prueba
LDA	0.0620	0.1146
LDA con reducción	0.1701	0.2242
No paramétrico	0.0089	0.0588
No paramétrico con reducción	0.0904	0.1684
Regresión multinomial	0.0001	0.1066
Regresión multinomial con reducción	0.1452	0.1968

Tabla 4.3.2: Tabla de los errores de entrenamiento y prueba para los distintos clasificadores para la versión normal y reducida

A partir de los resultados obtenidos vemos que en el caso del análisis discriminante lineal tanto el tiempo de ejecución como el error de prueba aumentan con respecto a los obtenidos en PCA por lo que este método no resulta para nada eficaz (En cuanto al sobreajuste se mantiene más o menos igual). En el caso del análisis discriminante no paramétrico tenemos un resultado bastante sorprendente y es que el tiempo de reducción disminuye considerablemente aunque a costa de un aumento bastante grande del error de prueba (Sobreajusta más los datos que con el conjunto de datos original). En el caso de la regresión logística, que era el caso que más nos interesaba, ha resultado contrario a lo que esperábamos porque tenemos un aumento ligero del tiempo de ejecución junto con un aumento notable en el error de prueba, con respecto al obtenido usando PCA (El sobreajuste a los datos disminuye).

En general, si bien no alcanzan el rendimiento óptimo obtenido con el conjunto completo de variables o con PCA, los resultados siguen siendo aceptables, y los tiempos de ejecución no se ven incrementados de forma excesiva, disminuyendo en el caso del análisis discriminante no paramétrico. En resumen, la regresión logística multinomial se beneficia significativamente de la reducción de dimensionalidad con PCA, mientras que el análisis discriminante lineal y el no paramétrico muestran un empeoramiento moderado pero manejable bajo simplificaciones más extremas de los datos.

APÉNDICE

A. Código fuente

El código fuente y el conjunto de datos utilizado en el ejemplo de clasificación de dígitos manuscritos están disponibles públicamente en el siguiente repositorio de GitHub: https://github.com/Juandpy10/archivos.tfg

A.1. Análisis discriminante

En esta sección se adjunta el código R empleado para la realización de todos los cálculos y gráficos del capítulo de análisis discriminante.

```r
#Librerias
library(MASS)
library(mvtnorm)
library(kernlab)
library(imager)

#Ejemplo Analisis discriminante lineal con variable respuesta
binaria
#y clasificador LDA poblacional para p=1

# Parámetros
mu1 <- 3
mu2 <- 4

sigma2 <- 2
sigma <- sqrt(sigma2)

pi1 <- 0.4
pi2 <- 0.6

# Rango de valores x
x <- seq(0, 7, length.out = 500)

# Funciones de densidad
f1 <- pi1 * dnorm(x, mean = mu1, sd = sigma)
f2 <- pi2 * dnorm(x, mean = mu2, sd = sigma)

# Población total
f_total <- f1 + f2

# Abrir el dispositivo gráfico PNG
png(filename = "C:/Users/Usuario/Desktop/TFG/Plots/Plot1.png",
    width = 480, height = 480, units = "px", pointsize = 12,
    bg = "white", res = NA, family = "", restoreConsole = TRUE,
    type = "cairo-png", antialias = "default", symbolfamily =
```

```
"default")

#Representación gráfica de la población

plot(x, f_total, type = "l", col = "black", lwd = 2,ylab =
"Densidad", xlab = "X")
lines(x, f1, col = "blue", lwd = 2)
lines(x, f2, col = "red", lwd = 2)
legend("topright", legend = c("Clase 1", "Clase 2",
"Población"), col = c("blue", "red", "black"), lwd = 2)

#Calculo y representación de la frontera de decision
hp <- (mu1 + mu2)/2 - (sigma2 / (mu1 - mu2)) * log(pi1 /
pi2);hp;

## [1] 2.68907

abline(v=hp)
# Cerrar el dispositivo gráfico
dev.off()

#Ejemplo Analisis discriminante lineal con variable respuesta
binaria
#y clasificador LDA muestral para p=1

#Semilla para reproducibilidad
set.seed(129)

# Tomar una muestra aleatoria simple de 800 individuos
n_total <- 800
clases <- sample(c(1, 2), size = n_total, replace = TRUE, prob =
c(0.4, 0.6))

valores <- numeric(n_total)
valores[clases == 1] <- rnorm(sum(clases == 1), mean = mu1, sd =
sigma)
valores[clases == 2] <- rnorm(sum(clases == 2), mean = mu2, sd =
sigma)

# Crear data.frame
muestra <- data.frame(x = valores, clase = factor(clases))

# Tomar 600 para entrenamiento y 200 para test
muestratrain <- muestra[1:600,]
muestratest <- muestra[601:800,]


#Calculo las estimaciones de pi, mu y sigma
# Número total de individuos en la muestra
ntrain <- nrow(muestratrain)
K <- 2  # Número de clases
```

```
# Estimadores de pi (proporciones muestrales)
pi1_hat <- length(muestratrain$x[muestratrain$clase ==
1])/ntrain
pi2_hat <- length(muestratrain$x[muestratrain$clase ==
2])/ntrain

# Estimadores de mu (medias muestrales por clase)
mu1_hat <- mean(muestratrain$x[muestratrain$clase == 1])
mu2_hat <- mean(muestratrain$x[muestratrain$clase == 2])

#Estimador varianza común

# Calculo de la suma de cuadrados dentro de cada clase
ss1 <- sum((muestratrain$x[muestratrain$clase == 1] -
mu1_hat)^2)
ss2 <- sum((muestratrain$x[muestratrain$clase == 2] -
mu2_hat)^2)

# Varianza común
sigma_hat_sq <- (ss1 + ss2) / (ntrain - K)

# Abrir el dispositivo gráfico PNG
png(filename = "C:/Users/Usuario/Desktop/TFG/Plots/Plot2.png",
    width = 480, height = 480, units = "px", pointsize = 12,
    bg = "white", res = NA, family = "", restoreConsole = TRUE,
    type = "cairo-png", antialias = "default", symbolfamily =
"default")

#Representación gráfica de la muestra

plot(muestratrain$x, muestratrain$clase,col =
ifelse(muestratrain$clase == 1, "blue", "red"),pch = 19,
     xlab = "X",ylab = "", yaxt = "n")

#Frontera poblacional
abline(v=hp, col="black",lw=2)

# Calculo y representacion de la frontera de decision
hm <- (mu1_hat + mu2_hat)/2 - (sigma_hat_sq / (mu1_hat -
mu2_hat)) * log(pi1_hat / pi2_hat)
abline(v=hm, col="green",lw=2)

# Agregar la leyenda
legend(5.1, 1.7,legend = c("Clase 1", "Clase 2", "LDA muestral",
"LDA poblacional"), col = c("blue", "red", "green", "black"),pch
= c(19,19,NA,NA), lw=c(NA,NA,2,2))

# Cerrar el dispositivo gráfico
dev.off()
```

```r
#Error clasificador muestral
#Tasa de error de entrenamiento
te <- 0
for (i in 1:ntrain) {
  xi <- muestratrain$x[i]
  clase_real <- muestratrain$clase[i]

  # Clasificacion
  clase_predicha <- ifelse(xi >= hm, 2, 1)

  if (clase_predicha != clase_real) {
    te <- te + 1
  }
}
tentrenamiento <- te / ntrain;tentrenamiento;

#Tasa de error de prueba
ntest <- nrow(muestratest)
tp <- 0
for (i in 1:ntest) {
  xi <- muestratest$x[i]
  clase_real <- muestratest$clase[i]

  # Clasificacion
  clase_predicha <- ifelse(xi >= hm, 2, 1)

  if (clase_predicha != clase_real) {
    tp <- tp + 1
  }
}
tprueba <- tp / ntest;tprueba;

#Error clasificador poblacional

#Tasa de error de prueba
tp <- 0
for (i in 1:ntest) {
  xi <- muestratest$x[i]
  clase_real <- muestratest$clase[i]

  # Clasificacion
  clase_predicha <- ifelse(xi >= hp, 2, 1)

  if (clase_predicha != clase_real) {
    tp <- tp + 1
  }
}
tprueba <- tp / ntest;tprueba;

#Ejemplo Analisis discriminante lineal con variable respuesta
binaria
```

```r
#y clasificador LDA poblacional p>1


# Parámetros
mu1 <- c(3,2)
mu2 <- c(4,5)
Mc <- matrix(c(1, 0.5, 0.5, 1), nrow = 2)

pi1 <- 0.4
pi2 <- 0.6

#Calculo de la frontera de decision
ap <- solve(Mc) %*% (mu1 - mu2)
a1p <- ap[1]
a2p <- ap[2]
bp  <- (1/2)*((t(mu1)%*%solve(Mc)%*%t(t(mu1))) -
(t(mu2)%*%solve(Mc)%*%t(t(mu2))))
bp  <- bp - (log(pi1)-log(pi2))
# Definir la pendiente y la ordenada en el origen
op <- as.numeric(bp/a2p)  # Ordenada en el origen
mp <- as.numeric(-(a1p/a2p))   # Pendiente



#Ejemplo Analisis discriminante lineal con variable respuesta
binaria
#y clasificador LDA muestral p>1

#Semilla para reproducibilidad
set.seed(129)

# Tomar una muestra aleatoria simple de 800 individuos
n_total <- 800
clases <- sample(c(1, 2), size = n_total, replace = TRUE, prob =
c(0.4, 0.6))

valores <- matrix(NA, nrow = n_total, ncol = 2)
valores[clases == 1, ] <- mvrnorm(sum(clases == 1), mu1, Mc)
valores[clases == 2, ] <- mvrnorm(sum(clases == 2), mu2, Mc)

# Crear data.frame
muestra <- data.frame(x1 = valores[,1], x2 = valores[,2], clase
= factor(clases))

# Tomar 600 para entrenamiento y 200 para test
muestratrain <- muestra[1:600,]
muestratest <- muestra[601:800,]

#Calculo las estimaciones de pi mu y matriz de covarianzas comun

# Número total de observaciones y número de clases
```

```r
ntrain <- nrow(muestratrain)
K <- length(unique(muestratrain$clase))


# Separar datos por clase
X1 <- as.matrix(subset(muestratrain, clase == 1)[, c("x1",
"x2")])
X2 <- as.matrix(subset(muestratrain, clase == 2)[, c("x1",
"x2")])

# Estimadores de proporciones muestrales
pi1_hat <- sum(muestratrain$clase == 1) / ntrain
pi2_hat <- sum(muestratrain$clase == 2) / ntrain

# Estimadores de medias
mu1_hat <- colMeans(subset(muestratrain, clase == 1)[, c("x1",
"x2")])
mu2_hat <- colMeans(subset(muestratrain, clase == 2)[, c("x1",
"x2")])

#Estimador de la matriz de covarianzas comun

# Inicializar la suma
S_sum <- matrix(0, nrow = 2, ncol = 2)

# Sumar las matrices (x_i - mu_k)(x_i - mu_k)^T para clase 1
for (i in 1:nrow(X1)) {
  diff <- X1[i, ] - mu1_hat
  S_sum <- S_sum + diff %*% t(diff)
}

# Repetir para clase 2
for (i in 1:nrow(X2)) {
  diff <- X2[i, ] - mu2_hat
  S_sum <- S_sum + diff %*% t(diff)
}

# Dividir por (n - K)
Mc_hat <- S_sum / (ntrain - K)

# Abrir el dispositivo gráfico PNG
png(filename = "C:/Users/Usuario/Desktop/TFG/Plots/Plot3.png",
    width = 480, height = 480, units = "px", pointsize = 12,
    bg = "white", res = NA, type = "cairo-png", antialias =
"subpixel")
#Representación gráfica de la muestra

plot(muestratrain$x1, muestratrain$x2,col =
ifelse(muestratrain$clase == 1, "blue", "red"),pch = 19,
     xlab = "X1",ylab = "X2")
```

```
# Agregar la leyenda
legend("bottomright",legend = c("Clase 1", "Clase 2","LDA
muestral","LDA poblacional", "Dirección discriminante muestral",
"Dirección discriminante poblacional"), col = c("blue", "red",
"green", "black", "green","black"),pch = c(19,19,NA,NA,NA,NA),
lw=c(NA,NA,2,2,2,2),lty = c(1,1,1,1,2,2), cex = 0.6)

#Calculo y representacion de la frontera de decision muestral
am <- solve(Mc_hat) %*% (mu1_hat - mu2_hat)
a1m <- am[1]
a2m <- am[2]
bm  <- (1/2)*((t(mu1_hat)%*%solve(Mc_hat)%*%t(t(mu1_hat))) -
(t(mu2_hat)%*%solve(Mc_hat)%*%t(t(mu2_hat))))
bm  <- bm - (log(pi1_hat)-log(pi2_hat))
# Definir la pendiente y la ordenada en el origen
om <- as.numeric(bm/a2m)  # Ordenada en el origen
mm <- as.numeric(-(a1m/a2m))   # Pendiente

# Crear valores de x para graficar
x <- seq(0, 7, by = 0.1)

#Calculo los valores de y poblacionales
yp <- mp * x + op

#Graficar la recta poblacional
lines(x,yp, col="black", lw = 2)

# Calcular los valores de y muestrales
ym <- mm * x + om

# Graficar la recta muestral
lines(x, ym, col="green", lw = 2)

# Calcular y representación de la dirección discriminante omega
poblacional
omega <- solve(Mc) %*% (mu1 - mu2)

# Normalizar el vector para graficarlo como dirección
omega_unit <- omega / sqrt(sum(omega^2))

# Punto medio entre las medias muestrales
midpoint <- (mu1 + mu2) / 2

# Escalar el vector
scale_factor <- 5

# Punto final de la línea
endpoint1 <- midpoint + scale_factor * omega_unit
endpoint2 <- midpoint - scale_factor * omega_unit
```

```r
# Línea de dirección omega
segments(endpoint1[1], endpoint1[2], endpoint2[1], endpoint2[2],
col = "black", lwd = 2, lty = 2)



# Calcular y representación de la dirección discriminante omega
muestral
omega <- solve(Mc_hat) %*% (mu1_hat - mu2_hat)

# Normalizar el vector para graficarlo como dirección
omega_unit <- omega / sqrt(sum(omega^2))

# Punto medio entre las medias muestrales
midpoint <- (mu1_hat + mu2_hat) / 2

# Escalar el vector
scale_factor <- 5

# Punto final de la línea
endpoint1 <- midpoint + scale_factor * omega_unit
endpoint2 <- midpoint - scale_factor * omega_unit

# Línea de dirección omega
segments(endpoint1[1], endpoint1[2], endpoint2[1], endpoint2[2],
col = "green", lwd = 2, lty = 2)


# Cerrar el dispositivo gráfico
dev.off()

#Error clasificador muestral

# Tasa de error de entrenamiento
ntrain <- nrow(muestratrain)
te <- 0

for (i in 1:ntrain) {
  x1_i <- muestratrain$x1[i]
  x2_i <- muestratrain$x2[i]
  clase_real <- muestratrain$clase[i]

  # Clasificación
  decision_value <- a1m * x1_i + a2m * x2_i
  clase_predicha <- ifelse(decision_value >= bm, 1, 2)

  if (clase_predicha != clase_real) {
    te <- te + 1
  }
}
```

```
tentrenamiento <- te / ntrain; tentrenamiento;

#Tasa de error de prueba
tp <- 0
ntest <- nrow(muestratest)

for (i in 1:ntest) {
  x1_i <- muestratest$x1[i]
  x2_i <- muestratest$x2[i]
  clase_real <- muestratest$clase[i]

  # Clasificación
  decision_value <- a1m * x1_i + a2m * x2_i
  clase_predicha <- ifelse(decision_value >= bm, 1, 2)

  if (clase_predicha != clase_real) {
    tp <- tp + 1
  }
}

tprueba <- tp / ntest;tprueba;

#Error calsificador poblacional

#Tasa de error de prueba
tp <- 0
ntest <- nrow(muestratest)

for (i in 1:ntest) {
  x1_i <- muestratest$x1[i]
  x2_i <- muestratest$x2[i]
  clase_real <- muestratest$clase[i]

  # Clasificación
  decision_value <- a1p * x1_i + a2p * x2_i
  clase_predicha <- ifelse(decision_value >= bp, 1, 2)

  if (clase_predicha != clase_real) {
    tp <- tp + 1
  }
}

tprueba <- tp / ntest;tprueba;

#Ejemplo Analisis discriminante cuadratico con variable
respuesta binaria
#y clasificador QDA poblacional


# Parámetros
```

```r
mu1 <- c(3,2)
mu2 <- c(4,5)
Mc1 <- matrix(c(1, 0.5, 0.5, 1), nrow = 2)
Mc2 <- matrix(c(2, 1, 1, 2), nrow = 2)

pi1 <- 0.4
pi2 <- 0.6

#Calculo de la frontera de decision
# Inversas y determinantes
inv_Mc1 <- solve(Mc1)
inv_Mc2 <- solve(Mc2)
log_det_Mc1 <- log(det(Mc1))
log_det_Mc2 <- log(det(Mc2))


# Constante C
Cp <- log_det_Mc2 - log_det_Mc1 + 2*log(pi2) - 2*log(pi1)

# A, b, c
Ap <- inv_Mc1 - inv_Mc2
bp <- -2 * (inv_Mc1 %*% mu1 - inv_Mc2 %*% mu2)
c1p <- t(mu1) %*% inv_Mc1 %*% mu1
c2p <- t(mu2) %*% inv_Mc2 %*% mu2
cp <- c1p - c2p - Cp


#Simulación Monte Carlo para estimar tasa de error poblacional
en QDA

# Tamaño de la simulación Monte Carlo
N <- 100000

# Simulación de los datos
X1 <- mvrnorm(n = N / 2, mu = mu1, Sigma = Mc1)
X2 <- mvrnorm(n = N / 2, mu = mu2, Sigma = Mc2)

X <- rbind(X1, X2)
clases_verdaderas <- c(rep(1, N / 2), rep(2, N / 2))

# Función de clasificación QDA
qda_clasifica <- function(x, mu1, mu2, Mc1, Mc2, pi1, pi2) {
  inv_Mc1 <- solve(Mc1)
  inv_Mc2 <- solve(Mc2)
  det_Mc1 <- det(Mc1)
  det_Mc2 <- det(Mc2)

  d1 <- -0.5 * t(x - mu1) %*% inv_Mc1 %*% (x - mu1) -
    0.5 * log(det_Mc1) + log(pi1)
  d2 <- -0.5 * t(x - mu2) %*% inv_Mc2 %*% (x - mu2) -
```

```
    0.5 * log(det_Mc2) + log(pi2)

  if (d1 > d2) return(1) else return(2)
}

# Clasificación y cálculo de tasa de error
te <- 0
for (i in 1:N) {
  x_i <- X[i, ]
  clase_predicha <- qda_clasifica(x_i, mu1, mu2, Mc1, Mc2, pi1,
pi2)
  if (clase_predicha != clases_verdaderas[i]) {
    te <- te + 1
  }
}

# Tasa de error estimada
te <- te / N; te;

#Ejemplo Analisis discriminante cuadratico con variable
respuesta binaria
#y clasificador QDA muestral

#Semilla para reproducibilidad
set.seed(129)

# Tomar una muestra aleatoria simple de 800 individuos
n_total <- 800
clases <- sample(c(1, 2), size = n_total, replace = TRUE, prob =
c(0.4, 0.6))

valores <- matrix(NA, nrow = n_total, ncol = 2)
valores[clases == 1, ] <- mvrnorm(sum(clases == 1), mu1, Mc1)
valores[clases == 2, ] <- mvrnorm(sum(clases == 2), mu2, Mc2)

# Crear data.frame
muestra <- data.frame(x1 = valores[,1], x2 = valores[,2], clase
= factor(clases))

# Tomar 600 para entrenamiento y 200 para test
muestratrain <- muestra[1:600,]
muestratest <- muestra[601:800,]

#Calculo las estimaciones de pi mu y matrices de covarianzas

# Número total de observaciones y número de clases
ntrain <- nrow(muestratrain)
K <- length(unique(muestratrain$clase))


# Separar datos por clase
```

```
X1 <- as.matrix(subset(muestratrain, clase == 1)[, c("x1",
"x2")])
X2 <- as.matrix(subset(muestratrain, clase == 2)[, c("x1",
"x2")])

# Estimadores de proporciones muestrales
pi1_hat <- sum(muestratrain$clase == 1) / ntrain
pi2_hat <- sum(muestratrain$clase == 2) / ntrain

# Estimadores de medias
mu1_hat <- colMeans(subset(muestratrain, clase == 1)[, c("x1",
"x2")])
mu2_hat <- colMeans(subset(muestratrain, clase == 2)[, c("x1",
"x2")])

#Estimadores de las matrices de covarianzas

# Centrar los datos restando la media correspondiente
X1_centrado <- sweep(X1, 2, mu1_hat)
X2_centrado <- sweep(X2, 2, mu2_hat)

# Calcular las matrices de covarianza para cada clase
Mc1_hat <- (t(X1_centrado) %*% X1_centrado) / (nrow(X1) - 1)
Mc2_hat <- (t(X2_centrado) %*% X2_centrado) / (nrow(X2) - 1)

# Abrir el dispositivo gráfico PNG
png(filename = "C:/Users/Usuario/Desktop/TFG/Plots/Plot4.png",
    width = 480, height = 480, units = "px", pointsize = 12,
    bg = "white", res = NA, family = "", restoreConsole = TRUE,
    type = "cairo-png", antialias = "default", symbolfamily =
"default")
#Representación gráfica de la muestra

plot(muestratrain$x1, muestratrain$x2,col =
ifelse(muestratrain$clase == 1, "blue", "red"),pch = 19,
     xlab = "X1",ylab = "X2")
# Agregar la leyenda
legend("bottomright",legend = c("Clase 1", "Clase 2", "QDA
muestral", "QDA poblacional"), col = c("blue", "red", "green",
"black"),pch = c(19,19,NA,NA),lw=c(NA,NA,2,2), cex=0.8)

#Calculo y representacion de la frontera de decision muestral

# Inversas y determinantes
inv_Mc1_hat <- solve(Mc1_hat)
inv_Mc2_hat <- solve(Mc2_hat)
log_det_Mc1_hat <- log(det(Mc1_hat))
log_det_Mc2_hat <- log(det(Mc2_hat))

# Constante C
```

```
Cm <- log_det_Mc2_hat - log_det_Mc1_hat + 2*log(pi2_hat) -
2*log(pi1_hat)

# A, b, c
Am <- inv_Mc1_hat - inv_Mc2_hat
bm <- -2 * (inv_Mc1_hat %*% mu1_hat - inv_Mc2_hat %*% mu2_hat)
c1m <- t(mu1_hat) %*% inv_Mc1_hat %*% mu1_hat
c2m <- t(mu2_hat) %*% inv_Mc2_hat %*% mu2_hat
cm <- c1m - c2m - Cm

# Función cuadrática
f_quadratic <- function(x1, x2, A, b, c) {
  A[1,1]*x1^2 + A[2,2]*x2^2 + (A[1,2] + A[2,1])*x1*x2 +
    b[1]*x1 + b[2]*x2 + c
}

# Crear malla muestral
x1_seq <- seq(min(muestratrain$x1) - 1, max(muestratrain$x1) +
1, length = 300)
x2_seq <- seq(min(muestratrain$x2) - 1, max(muestratrain$x2) +
1, length = 300)
zm <- outer(x1_seq, x2_seq,Vectorize(function(x1, x2)
f_quadratic(x1, x2, Am, bm, cm)))

# Crear malla poblacional
zp <- outer(x1_seq, x2_seq, Vectorize(function(x1, x2)
f_quadratic(x1, x2, Ap, bp, cp)))

#Representar la frontera muestral
contour(x1_seq, x2_seq, zm, add= TRUE, levels = 0, drawlabels =
FALSE, col = "green", lw=2)

#Representar la frontera poblacional
contour(x1_seq, x2_seq, zp, add= TRUE, levels = 0, drawlabels =
FALSE, col= "black", lw=2)

# Cerrar el dispositivo gráfico
dev.off()

#Error clasificador muestral

# Tasa de error de entrenamiento
ntrain <- nrow(muestratrain)
te <- 0

for (i in 1:ntrain) {
  x_i <- c(muestratrain$x1[i], muestratrain$x2[i])
  clase_real <- muestratrain$clase[i]

  clase_predicha <- qda_clasifica(x_i, mu1_hat, mu2_hat,
```

```
Mc1_hat, Mc2_hat, pi1_hat, pi2_hat)

  if (clase_predicha != clase_real) {
    te <- te + 1
  }
}

tentrenamiento <- te / ntrain; tentrenamiento;

#Tasa de error de prueba
tp <- 0
ntest <- nrow(muestratest)

for (i in 1:ntest) {
  x_i <- c(muestratest$x1[i], muestratest$x2[i])
  clase_real <- muestratest$clase[i]

  clase_predicha <- qda_clasifica(x_i, mu1_hat, mu2_hat,
Mc1_hat, Mc2_hat, pi1_hat, pi2_hat)

  if (clase_predicha != clase_real) {
    tp <- tp + 1
  }
}

tprueba <- tp / ntest;tprueba;

#Error clasificador poblacional

#Tasa de error de prueba
tp <- 0
ntest <- nrow(muestratest)

for (i in 1:ntest) {
  x_i <- c(muestratest$x1[i], muestratest$x2[i])
  clase_real <- muestratest$clase[i]

  clase_predicha <- qda_clasifica(x_i, mu1, mu2, Mc1, Mc2, pi1,
pi2)

  if (clase_predicha != clase_real) {
    tp <- tp + 1
  }
}

tprueba <- tp / ntest;tprueba;

#Ejemplos funciones de R para analisis discriminante

# Generar datos de ejemplo
set.seed(42)
```

```r
n <- 100
# Vectores de medias
mu1 <- c(1, 2)
mu2 <- c(5, 6)
mu3 <- c(9, 2)
# Matrices de covarianzas
Mc1 <- matrix(c(1, 0.5, 0.5, 1), nrow = 2)
Mc2 <- matrix(c(2, 1, 1, 2), nrow = 2)
Mc3 <- matrix(c(0.5,0.25,0.25,0.5), nrow = 2)

# Datos de ejemplo
# Tomar una muestra aleatoria simple de 800 individuos
n_total <- 800
clases <- sample(c(1, 2, 3), size = n_total, replace = TRUE,
prob = c(0.4, 0.3, 0.3))

valores <- matrix(NA, nrow = n_total, ncol = 2)
valores[clases == 1, ] <- mvrnorm(sum(clases == 1), mu1, Mc1)
valores[clases == 2, ] <- mvrnorm(sum(clases == 2), mu2, Mc2)
valores[clases == 3, ] <- mvrnorm(sum(clases == 3), mu3, Mc3)

# Crear data.frame
muestra <- data.frame(x1 = valores[,1], x2 = valores[,2], clase
= factor(clases))

# Tomar 600 para entrenamiento y 200 para test
muestratrain <- muestra[1:600,]
muestratest <- muestra[601:800,]

#Ajuste de los modelos

# Ajustar el modelo LDA
modelo_lda <- lda(clase ~ x1 + x2, data = muestratrain)

# Ajustar el modelo QDA
modelo_qda <- qda(clase ~ x1 + x2, data = muestratrain)

# Ajustar modelo no parametrico
modelo_NP <- ksvm(clase ~ x1 + x2, data = muestratrain, kernel =
"rbfdot")


#Representacion grafica de las fronteras de decision

# Abrir el dispositivo gráfico PNG
png(filename = "C:/Users/Usuario/Desktop/TFG/Plots/Plot5.png",
    width = 480, height = 480, units = "px", pointsize = 12,
    bg = "white", res = NA, family = "", restoreConsole = TRUE,
    type = "cairo-png", antialias = "default", symbolfamily =
"default")
```

```r
#Modelo lda
# Gráfico con las fronteras de decisión
# Colores para las clases
colores <- c("blue", "red", "green")
plot(muestratrain$x1, muestratrain$x2, col =
colores[muestratrain$clase], pch = 19,
     xlab = "X1", ylab = "X2")


# Crear una malla de puntos (rejilla)
x1_range <- seq(min(muestratrain$x1) - 1, max(muestratrain$x1) +
1, length = 200)
x2_range <- seq(min(muestratrain$x2) - 1, max(muestratrain$x2) +
1, length = 200)
grid <- expand.grid(x1 = x1_range, x2 = x2_range)

# Predecir la clase para cada punto de la rejilla
pred <- predict(modelo_lda, newdata = grid)$class

# Convertir las predicciones en matriz para graficar con contour
z <- matrix(as.numeric(pred), nrow = length(x1_range), byrow =
FALSE)

# Dibujar fronteras con contour (niveles entre clases)
contour(x1_range, x2_range, z, levels = c(1.5, 2.5), add = TRUE,
drawlabels = FALSE, lwd = 2, col = "black")



legend("topright", legend = c("Clase 1", "Clase 2", "Clase 3"),
col = c("blue","red","green"), pch = 19)

# Cerrar el dispositivo gráfico
dev.off()

#Modelo qda

# Abrir el dispositivo gráfico PNG
png(filename = "C:/Users/Usuario/Desktop/TFG/Plots/Plot6.png",
    width = 480, height = 480, units = "px", pointsize = 12,
    bg = "white", res = NA, family = "", restoreConsole = TRUE,
    type = "cairo-png", antialias = "default", symbolfamily =
"default")

# Gráfico con las fronteras de decisión
# Colores para las clases
colores <- c("blue", "red", "green")
plot(muestratrain$x1, muestratrain$x2, col =
colores[muestratrain$clase], pch = 19,
     xlab = "X1", ylab = "X2")
```

```r
# Generar una cuadrícula para el espacio de las variables
grid <- expand.grid(
  x1 = seq(min(muestratrain$x1), max(muestratrain$x1),
length.out = 100),
  x2 = seq(min(muestratrain$x2), max(muestratrain$x2),
length.out = 100)
)

# Predicciones para la cuadrícula
pred_grid <- predict(modelo_qda, grid)

# Transformar las clases predichas a valores numéricos
z <- as.numeric(pred_grid$class)

# Crear una matriz de las clases para la cuadrícula
z_matrix <- matrix(z, nrow = 100, byrow = FALSE)

# Dibujar las fronteras de decisión para todas las clases
contour(
  x = unique(grid$x1), y = unique(grid$x2),
  z = z_matrix,
  levels = c(1.5, 2.5),  # Niveles donde cambian las clases
(entre A-B y B-C)
  add = TRUE, drawlabels = FALSE, col = "black"
)

legend("topright", legend = c("Clase 1", "Clase 2", "Clase 3"),
col = c("blue","red","green"), pch = 19)

# Cerrar el dispositivo gráfico
dev.off()

#Modelo no parametrico

# Abrir el dispositivo gráfico PNG
png(filename = "C:/Users/Usuario/Desktop/TFG/Plots/Plot7.png",
    width = 480, height = 480, units = "px", pointsize = 12,
    bg = "white", res = NA, family = "", restoreConsole = TRUE,
    type = "cairo-png", antialias = "default", symbolfamily =
"default")

# Gráfico con las fronteras de decisión
# Colores para las clases
colores <- c("blue", "red", "green")
plot(muestratrain$x1, muestratrain$x2, col =
colores[muestratrain$clase], pch = 19,
     xlab = "X1", ylab = "X2")

# Crear grid
```

```r
x1_seq <- seq(min(muestratrain$x1) - 1, max(muestratrain$x1) +
1, length.out = 300)
x2_seq <- seq(min(muestratrain$x2) - 1, max(muestratrain$x2) +
1, length.out = 300)
grid <- expand.grid(x1 = x1_seq, x2 = x2_seq)

# Predecir
grid$pred <- predict(modelo_NP, newdata = grid)
grid$pred_num <- as.numeric(grid$pred)

# Convertir a matriz correctamente (ojo: byrow = FALSE)
z_matrix <- matrix(grid$pred_num, nrow = length(x1_seq), byrow =
FALSE)

# Agregar fronteras
contour(x1_seq, x2_seq, z_matrix, levels = c(1.5, 2.5), add =
TRUE, col = "black", lwd = 2)

legend("topright", legend = c("Clase 1", "Clase 2", "Clase 3"),
col = c("blue","red","green"), pch = 19)

# Cerrar el dispositivo gráfico
dev.off()

#Tasas de error de prueba


#Modelo lda
predicciones_lda <- predict(modelo_lda, muestratest)$class
t_prueba <- sum(muestratest$clase !=
predicciones_lda)/length(muestratest$clase)
t_prueba;

#Modelo qda
predicciones_qda <- predict(modelo_qda, muestratest)$class
t_prueba <- sum(muestratest$clase !=
predicciones_qda)/length(muestratest$clase)
t_prueba;

#Modelo no parametrico
predicciones_NP <- predict(modelo_NP, muestratest)
t_prueba <- sum(muestratest$clase !=
predicciones_NP)/length(muestratest$clase)
t_prueba;
```

A.2. Métodos basados en la regresión

En esta sección se adjunta el código R empleado para la realización de todos los cálculos y gráficos del capítulo de métodos basados en la regresión.

```r
#Librerias
library(MASS)
library(mvtnorm)
library(kernlab)
library(ISLR)
library(nnet)
library(imager)

#Regresion lineal
data(Default)
# Transformar usando ifelse
default <- ifelse(Default[,1] == "Yes", 1, 0)

# Agregar al DataFrame si es necesario
Defaultb <- data.frame(default, Default[,2],
Default[,3],Default[,4])
colnames(Defaultb) <- c("default", "student", "balance",
"income")

# Ajustar la regresión lineal
modelo_lineal <- lm(default ~ balance, data = Defaultb)

# Crear una secuencia de valores para X
x_seq <- seq(min(Defaultb$balance), max(Defaultb$balance),
length.out = 100)

# Calcular las probabilidades predichas usando el modelo lineal
predicciones <- predict(modelo_lineal, newdata =
data.frame(balance = x_seq))

# Graficar los puntos originales y la línea de regresión
# Abrir el dispositivo gráfico PNG
png(filename = "C:/Users/Usuario/Desktop/TFG/Plots/Plot8.png",
    width = 480, height = 480, units = "px", pointsize = 12,
    bg = "white", res = NA, family = "", restoreConsole = TRUE,
    type = "cairo-png", antialias = "default", symbolfamily =
"default")

plot(Defaultb$balance, Defaultb$default,col="orange", pch = 16,
xlab = "X",
     ylab = "Probabilidad de Y = 1")
abline(h=1, lty=2)
abline(h=0, lty=2)
lines(x_seq, predicciones, col = "blue", lwd = 2)
# Cerrar el dispositivo gráfico
dev.off()

#Regresion logistica simple
# Ajustar la regresión logística
modelo_logistico <- glm(default ~ balance, data = Defaultb,
```

```
family = binomial)


# Crear una secuencia de valores para X
x_seq <- seq(min(Defaultb$balance), max(Defaultb$balance),
length.out = 100)

# Calcular las probabilidades predichas
predicciones <- predict(modelo_logistico, newdata =
data.frame(balance = x_seq), type = "response")

# Graficar los puntos originales y la curva de predicción
# Abrir el dispositivo gráfico PNG
png(filename = "C:/Users/Usuario/Desktop/TFG/Plots/Plot9.png",
    width = 480, height = 480, units = "px", pointsize = 12,
    bg = "white", res = NA, family = "", restoreConsole = TRUE,
    type = "cairo-png", antialias = "default", symbolfamily =
"default")

plot(Defaultb$balance, Defaultb$default,col="orange", pch = 16,
xlab = "X",
     ylab = "Probabilidad de Y = 1")
abline(h=1, lty=2)
abline(h=0, lty=2)
lines(x_seq, predicciones, col = "blue", lwd = 2)
# Cerrar el dispositivo gráfico
dev.off()

#Ejemplos funciones de R para regresion logistica


#Datos regresion lineal y regresion logistica simple múltiple
set.seed(129)
# Vectores de medias
mu1 <- c(3, 2)
mu2 <- c(4, 5)

# Matriz de covarianza común
Mc <- matrix(c(1, 0.5, 0.5, 1), nrow = 2)

# Tomar una muestra aleatoria simple de 800 individuos
n_total <- 800
clases <- sample(c(1, 2), size = n_total, replace = TRUE, prob =
c(0.4, 0.6))

valores <- matrix(NA, nrow = n_total, ncol = 2)
valores[clases == 1, ] <- mvrnorm(sum(clases == 1), mu1, Mc)
valores[clases == 2, ] <- mvrnorm(sum(clases == 2), mu2, Mc)

# Crear data.frame
muestra <- data.frame(x1 = valores[,1], x2 = valores[,2], clase
```

```
= factor(clases))

# Tomar 600 para entrenamiento y 200 para test
muestratrain1 <- muestra[1:600,]
muestratest1 <- muestra[601:800,]

#Datos regresion logistica multinomial
# Generar datos de ejemplo
set.seed(42)
n <- 100
# Vectores de medias
mu1 <- c(1, 2)
mu2 <- c(5, 6)
mu3 <- c(9, 2)
# Matrices de covarianzas
Mc1 <- matrix(c(1, 0.5, 0.5, 1), nrow = 2)
Mc2 <- matrix(c(2, 1, 1, 2), nrow = 2)
Mc3 <- matrix(c(0.5,0.25,0.25,0.5), nrow = 2)

# Datos de ejemplo
# Tomar una muestra aleatoria simple de 800 individuos
n_total <- 800
clases <- sample(c(1, 2, 3), size = n_total, replace = TRUE,
prob = c(0.4, 0.3, 0.3))

valores <- matrix(NA, nrow = n_total, ncol = 2)
valores[clases == 1, ] <- mvrnorm(sum(clases == 1), mu1, Mc1)
valores[clases == 2, ] <- mvrnorm(sum(clases == 2), mu2, Mc2)
valores[clases == 3, ] <- mvrnorm(sum(clases == 3), mu3, Mc3)

# Crear data.frame
muestra <- data.frame(x1 = valores[,1], x2 = valores[,2], clase
= factor(clases))

# Tomar 600 para entrenamiento y 200 para test
muestratrain2 <- muestra[1:600,]
muestratest2 <- muestra[601:800,]

#Ajuste de los modelos
#Ajuste del modelo de regresion lineal
modelo_lineal <- lm(clase ~ x1 + x2, data = muestratrain1)

#Ajuste del modelo de regresion logistica multiple
modelo_logistico <- glm(clase ~ x1 + x2, data = muestratrain1,
family = binomial)

#Ajuste del modelo de regresion logistica multinomial
modelo_logisticoM <- multinom(clase ~ x1 + x2, data =
muestratrain2)
```

```r
#Representacion grafica de la frontera de decisión lineal
# Abrir el dispositivo gráfico PNG
png(filename = "C:/Users/Usuario/Desktop/TFG/Plots/Plot10.png",
    width = 480, height = 480, units = "px", pointsize = 12,
    bg = "white", res = NA, family = "", restoreConsole = TRUE,
    type = "cairo-png", antialias = "default", symbolfamily =
"default")
#Datos1
plot(muestratrain1$x1, muestratrain1$x2,col =
ifelse(muestratrain1$clase == 1, "blue", "red"),pch = 19,
     xlab = "X1",ylab = "X2")

# Extraer los coeficientes de la regresión lineal
coef_lineal <- coef(modelo_lineal)

# Crear una malla de puntos para predecir la frontera de
decisión
x1_grid <- seq(min(muestratrain1$x1), max(muestratrain1$x1),
length.out = 100)
x2_grid <- seq(min(muestratrain1$x2), max(muestratrain1$x2),
length.out = 100)

# Predecir probabilidades con el modelo lineal para la malla de
puntos
grid <- expand.grid(x1 = x1_grid, x2 = x2_grid)
prob_lineal <- predict(modelo_lineal, newdata = grid, type =
"response")

# Añadir la frontera de decisión de la regresión lineal
contour(x1_grid, x2_grid, matrix(prob_lineal, nrow = 100),
levels = 0.5, add = TRUE, col = "black", lwd = 2)

# Agregar la leyenda
legend(6.4,4,legend = c("Clase 1", "Clase 2"), col = c("blue",
"red"),pch = 19)

# Cerrar el dispositivo gráfico
dev.off()

#Representacion grafica de la frontera de decision logistica
# Abrir el dispositivo gráfico PNG
png(filename = "C:/Users/Usuario/Desktop/TFG/Plots/Plot11.png",
    width = 480, height = 480, units = "px", pointsize = 12,
    bg = "white", res = NA, family = "", restoreConsole = TRUE,
    type = "cairo-png", antialias = "default", symbolfamily =
"default")
#Modelo de regresion logística multiple

plot(muestratrain1$x1, muestratrain1$x2,col =
ifelse(muestratrain1$clase == 1, "blue", "red"),pch = 19,
```

```
     xlab = "X1",ylab = "X2")

# Extraer los coeficientes de la regresión logística
coef_logistico <- coef(modelo_logistico)

# Crear una malla de puntos para predecir la frontera de
decisión
x1_grid <- seq(min(muestratrain1$x1), max(muestratrain1$x1),
length.out = 100)
x2_grid <- seq(min(muestratrain1$x2), max(muestratrain1$x2),
length.out = 100)

# Predecir probabilidades con el modelo logístico para la malla
de puntos
grid <- expand.grid(x1 = x1_grid, x2 = x2_grid)
prob_logistico <- predict(modelo_logistico, newdata = grid, type
= "response")

# Añadir la frontera de decisión de la regresión logística
contour(x1_grid, x2_grid, matrix(prob_logistico, nrow = 100),
levels = 0.5, add = TRUE, col = "black", lwd = 2)

# Agregar la leyenda
legend("bottomright",legend = c("Clase 1", "Clase 2"), col =
c("blue", "red"),pch = 19)


# Cerrar el dispositivo gráfico
dev.off()

#Ecuaciuon de las fronteras de decision lineal y logistica
# Regresión lineal
pendiente_lineal <- -coef_lineal["x1"] / coef_lineal["x2"]
intercepto_lineal <- (0.5 - coef_lineal["(Intercept)"]) /
coef_lineal["x2"]

# Regresión logística
pendiente_logistico <- -coef_logistico["x1"] /
coef_logistico["x2"]
intercepto_logistico <- -coef_logistico["(Intercept)"] /
coef_logistico["x2"]


#Modelo de regresion logistica multinomial
# Abrir el dispositivo gráfico PNG
png(filename = "C:/Users/Usuario/Desktop/TFG/Plots/Plot12.png",
    width = 480, height = 480, units = "px", pointsize = 12,
    bg = "white", res = NA, family = "", restoreConsole = TRUE,
    type = "cairo-png", antialias = "default", symbolfamily =
"default")
```

```r
#Modelo de regresion logistica multinomial
# Colores para las clases
colores <- c("blue", "red", "green")
plot(muestratrain2$x1, muestratrain2$x2, col =
colores[muestratrain2$clase], pch = 19,
     xlab = "X1", ylab = "X2")

# Graficar las fronteras de decisión
x1_seq <- seq(min(muestratrain2$x1), max(muestratrain2$x1),
length = 100)
x2_seq <- seq(min(muestratrain2$x2), max(muestratrain2$x2),
length = 100)
grid <- expand.grid(x1 = x1_seq, x2 = x2_seq)

pred_grid <- predict(modelo_logisticoM, newdata = grid, type =
"class")

# Añadir las líneas de frontera
contour(x1_seq, x2_seq, matrix(as.numeric(pred_grid),
length(x1_seq), length(x2_seq)),
        add = TRUE, levels = 1:3, drawlabels = FALSE, lty = 1,
col = "black")
legend("topright", legend = c("Clase 1", "Clase 2", "Clase 3"),
col = c("blue","red","green"), pch = 19)

# Cerrar el dispositivo gráfico
dev.off()

#Predicciones

#Modelo de regresion lineal, esta no tendria sentido
probabilidades mayores que 1
predicciones_lineal <- predict(modelo_lineal, muestratest1)
# Clasificación: prob > 0.5 ⇒ clase 2, si no ⇒ clase 1
predicciones_lineal_clase <- ifelse(predicciones_lineal > 0.5,
2, 1)
t_prueba <- sum(muestratest1$clase !=
round(predicciones_lineal_clase))/length(muestratest1$clase)
t_prueba;

#Modelo de regresion logistica multiple
predicciones_logistico <- predict(modelo_logistico,
muestratest1, type = "response")
# Clasificación: prob > 0.5 ⇒ clase 2, si no ⇒ clase 1
predicciones_logistico_clase <- ifelse(predicciones_logistico >
0.5, 2, 1)
t_prueba <- sum(muestratest1$clase !=
predicciones_logistico_clase)/length(muestratest1$clase)
t_prueba;
```

```
#Modelo de regresion logistica multinomial
predicciones_logisticoM <- predict(modelo_logisticoM,
muestratest2)
t_prueba <- sum(muestratest2$clase !=
predicciones_logisticoM)/length(muestratest2$clase)
t_prueba;
```

A.3. Ejemplo de clasificación de dígitos manuscritos

En esta sección se adjunta el código R empleado para la realización de todos los cálculos y gráficos del capítulo de clasificación de dígitos manuscritos.

```
#Librerias
library(MASS)
library(kernlab)
library(nnet)
library(imager)

#Representacion de un numero del conjunto de datos de prueba

# Función para cargar imagenes
cargar_datos <- function(ruta) {
  data <- read.table(ruta, sep = "", strip.white = TRUE)
  etiquetas <- data[, 1]
  pixeles <- as.matrix(data[, 2:257])
  data.frame(label = etiquetas, pixeles)
}

# Rutas a tus archivos
train_path <- "C:/Users/Usuario/Desktop/TFG/zip.train.train"
test_path  <- "C:/Users/Usuario/Desktop/TFG/zip.test.test"

# Cargar conjuntos
train <- cargar_datos(train_path)
test  <- cargar_datos(test_path)
# Seleccionar una observación de test (por ejemplo, la 121)
fila <- 121
vector <- as.numeric(test[fila, -1])
vector <- vector*(-1)#Invertir colores para la representación
etiqueta_real <- test[fila, 1]

# Mostrar imagen
# Abrir el dispositivo gráfico PNG
png(filename = "C:/Users/Usuario/Desktop/TFG/Plots/Plot13.png",
    width = 480, height = 480, units = "px", pointsize = 12,
    bg = "white", res = NA, family = "", restoreConsole = TRUE,
    type = "cairo-png", antialias = "default", symbolfamily =
"default")
img <- as.cimg(vector)
```

```r
plot(img, main = paste("Etiqueta real:", etiqueta_real))

# Cerrar el dispositivo gráfico
dev.off()

#Precision de los distintos metodos de clasificacion

#LDA
#Entrenar el modelo lda
lda_model <- lda(label ~ ., data = train)

#Probar el modelo en el conjunto de entrenamiento
lda_pred <- predict(lda_model, train[, -1])$class
1 - mean(lda_pred == train$label);  # Precisión

#Probar el modelo en el conjunto de test
lda_pred <- predict(lda_model, test[, -1])$class
1 - mean(lda_pred == test$label);  # Precisión

#RLM
#Entrenar el modelo RLM
log_model <- multinom(label ~ ., data = train, MaxNWts = 10000,
trace = FALSE)

#Probar el modelo en el conjunto de entrenamiento
log_pred <- predict(log_model, train[, -1])
1 - mean(log_pred == train$label);  # Precisión

#Probar el modelo en el conjunto de test
log_pred <- predict(log_model, test[, -1])
1 - mean(log_pred == test$label);  # Precisión

#KSVM
# Función para cargar los datos especial para KSVM
cargar_datos <- function(ruta) {
  data <- read.table(ruta, sep = "", strip.white = TRUE)
  etiquetas <- data[, 1]
  pixeles <- as.matrix(data[, 2:257])
  data.frame(label = as.factor(etiquetas), pixeles)
}

# Rutas a tus archivos
train_path <- "C:/Users/Usuario/Desktop/TFG/zip.train.train"
test_path  <- "C:/Users/Usuario/Desktop/TFG/zip.test.test"

# Cargar conjuntos
train <- cargar_datos(train_path)
test  <- cargar_datos(test_path)
# Entrenar modelo SVM con kernel radial
svm_model <- ksvm(label ~ ., data = train, kernel = "rbfdot", C
= 1)#kernel radial (gaussiano)
```

```r
#opcion común para problemas de clasificación no lineal como
dígitos escritos a mano

# Probar el modelo en el conjunto de entrenamiento
svm_pred <- predict(svm_model, train[, -1])
# Calcular precisión
1 - mean(svm_pred == train$label);

# Probar el modelo en el conjunto de test
svm_pred <- predict(svm_model, test[, -1])
# Calcular precisión
1 - mean(svm_pred == test$label);

#Medicion de tiempos con datos completos

#LDA
# Medir tiempo total de carga, entrenamiento y predicción
tiempo <- system.time({
  # Función para cargar los datos
  cargar_datos <- function(ruta) {
    data <- read.table(ruta, sep = "", strip.white = TRUE)
    etiquetas <- data[, 1]
    pixeles <- as.matrix(data[, 2:257])
    data.frame(label = etiquetas, pixeles)
  }

  # Rutas a tus archivos
  train_path <- "C:/Users/Usuario/Desktop/TFG/zip.train.train"
  test_path  <- "C:/Users/Usuario/Desktop/TFG/zip.test.test"

  # Cargar conjuntos
  train <- cargar_datos(train_path)
  test  <- cargar_datos(test_path)

  #Entrenar el modelo lda
  lda_model <- lda(label ~ ., data = train)

  #Probar el modelo en el conjunto de entrenamiento
  lda_pred <- predict(lda_model, train[, -1])$class
  1 - mean(lda_pred == train$label);  # Precisión

  #Probar el modelo en el conjunto de test
  lda_pred <- predict(lda_model, test[, -1])$class
  1 - mean(lda_pred == test$label);  # Precisión
})

# Mostrar tiempo
print(tiempo);

#KSVM
# Medir tiempo total de carga, entrenamiento y predicción
```

```
tiempo <- system.time({
  # Función para cargar los datos
  cargar_datos <- function(ruta) {
    data <- read.table(ruta, sep = "", strip.white = TRUE)
    etiquetas <- data[, 1]
    pixeles <- as.matrix(data[, 2:257])
    data.frame(label = as.factor(etiquetas), pixeles)
  }

  # Rutas locales a tus archivos
  train_path <- "C:/Users/Usuario/Desktop/TFG/zip.train.train"
  test_path  <- "C:/Users/Usuario/Desktop/TFG/zip.test.test"

  # Cargar conjuntos de datos
  train <- cargar_datos(train_path)
  test  <- cargar_datos(test_path)

  # Entrenar modelo SVM con kernel radial
  svm_model <- ksvm(label ~ ., data = train, kernel = "rbfdot",
C = 1)#kernel radial (gaussiano)
  #opcion común para problemas de clasificación no lineal como
dígitos escritos a mano

  # Probar el modelo en el conjunto de entrenamiento
  svm_pred <- predict(svm_model, train[, -1])
  # Calcular precisión
  1 - mean(svm_pred == train$label);

  # Probar el modelo en el conjunto de test
  svm_pred <- predict(svm_model, test[, -1])
  # Calcular precisión
  1 - mean(svm_pred == test$label);
})

# Mostrar tiempo
print(tiempo);

#RLM
# Medir tiempo total de carga, entrenamiento y predicción
tiempo <- system.time({
  # Función para cargar los datos
  cargar_datos <- function(ruta) {
    data <- read.table(ruta, sep = "", strip.white = TRUE)
    etiquetas <- data[, 1]
    pixeles <- as.matrix(data[, 2:257])
    data.frame(label = etiquetas, pixeles)
  }

  # Rutas a tus archivos
  train_path <- "C:/Users/Usuario/Desktop/TFG/zip.train.train"
```

```
  test_path  <- "C:/Users/Usuario/Desktop/TFG/zip.test.test"

  # Cargar conjuntos
  train <- cargar_datos(train_path)
  test  <- cargar_datos(test_path)

  #Entrenar el modelo RLM
  log_model <- multinom(label ~ ., data = train, MaxNWts =
10000, trace = FALSE)

  #Probar el modelo en el conjunto de entrenamiento
  log_pred <- predict(log_model, train[, -1])
  1 - mean(log_pred == train$label);  # Precisión

  #Probar el modelo en el conjunto de test
  log_pred <- predict(log_model, test[, -1])
  1 - mean(log_pred == test$label);  # Precisión
})

# Mostrar tiempo
print(tiempo);

#Clasificacion de digitos manuscritos usando PCA

#LDA
# Medir tiempo total de carga, entrenamiento y predicción
tiempo <- system.time({
  # Función para cargar los datos
  cargar_datos <- function(ruta) {
    data <- read.table(ruta, sep = "", strip.white = TRUE)
    etiquetas <- data[, 1]
    pixeles <- as.matrix(data[, 2:257])
    data.frame(label = etiquetas, pixeles)
  }

  # Rutas a los archivos
  train_path <- "C:/Users/Usuario/Desktop/TFG/zip.train.train"
  test_path  <- "C:/Users/Usuario/Desktop/TFG/zip.test.test"

  # Cargar los datos
  train <- cargar_datos(train_path)
  test  <- cargar_datos(test_path)

  # Separar etiquetas y características
  etiquetas_train <- train$label
  etiquetas_test  <- test$label
  x_train <- train[, -1]
  x_test  <- test[, -1]

  # Aplicar PCA sobre los datos de entrenamiento
```

```
  pca <- prcomp(x_train, center = TRUE, scale. = TRUE)

  # Determinar cuántos componentes explican el 95% de la
varianza
  varianza_explicada <- summary(pca)$importance[3, ]
  num_componentes <- which(cumsum(pca$sdev^2 / sum(pca$sdev^2))
>= 0.95)[1]

  # Transformar datos
  x_train_pca <- as.data.frame(pca$x[, 1:num_componentes])
  x_train_pca$label <- etiquetas_train

  # Transformar test con los mismos componentes
  x_test_pca <- as.data.frame(predict(pca, newdata = x_test)[,
1:num_componentes])
  x_test_pca$label <- etiquetas_test

  # Entrenar modelo LDA con datos reducidos
  lda_model <- lda(label ~ ., data = x_train_pca)

  # Predecir y calcular precisión en el conjunto de
entrenamiento
  lda_pred <- predict(lda_model, x_train_pca[, -
ncol(x_train_pca)])$class
  print(1 - mean(lda_pred == etiquetas_train));

  # Predecir y calcular precisión en el conjunto de test
  lda_pred <- predict(lda_model, x_test_pca[, -
ncol(x_test_pca)])$class
  print(1 - mean(lda_pred == etiquetas_test));
})

# Mostrar tiempo
print(tiempo);

#KSVM
# Medir tiempo total de carga, entrenamiento y predicción
tiempo <- system.time({
  # Función para cargar los datos
  cargar_datos <- function(ruta) {
    data <- read.table(ruta, sep = "", strip.white = TRUE)
    etiquetas <- data[, 1]
    pixeles <- as.matrix(data[, 2:257])
    data.frame(label = as.factor(etiquetas), pixeles)
  }

  # Rutas a los archivos
  train_path <- "C:/Users/Usuario/Desktop/TFG/zip.train.train"
  test_path  <- "C:/Users/Usuario/Desktop/TFG/zip.test.test"

  # Cargar los datos
```

```
  train <- cargar_datos(train_path)
  test  <- cargar_datos(test_path)

  # Separar etiquetas y características
  etiquetas_train <- train$label
  etiquetas_test  <- test$label
  x_train <- train[, -1]
  x_test  <- test[, -1]

  # Aplicar PCA sobre los datos de entrenamiento
  pca <- prcomp(x_train, center = TRUE, scale. = TRUE)

  # Determinar cuántos componentes explican el 95% de la
varianza
  varianza_explicada <- summary(pca)$importance[3, ]
  num_componentes <- which(cumsum(pca$sdev^2 / sum(pca$sdev^2))
>= 0.95)[1]

  # Transformar datos
  x_train_pca <- as.data.frame(pca$x[, 1:num_componentes])
  x_train_pca$label <- etiquetas_train

  # Transformar test con los mismos componentes
  x_test_pca <- as.data.frame(predict(pca, newdata = x_test)[,
1:num_componentes])
  x_test_pca$label <- etiquetas_test

  # Entrenar modelo SVM con kernel radial
  svm_model <- ksvm(label ~ ., data = x_train_pca, kernel =
"rbfdot", C = 1)#kernel radial (gaussiano)
  #opcion común para problemas de clasificación no lineal como
dígitos escritos a mano

  # Predecir y calcular precisión en el conjunto de
entrenamiento
  svm_pred <- predict(svm_model, x_train_pca[, -
ncol(x_train_pca)])
  print(1 - mean(svm_pred == etiquetas_train));

  # Predecir y calcular precisión en el conjunto de test
  svm_pred <- predict(svm_model, x_test_pca[, -
ncol(x_test_pca)])
  print(1 - mean(svm_pred == etiquetas_test));
})

# Mostrar tiempo
print(tiempo);

#RLM
# Medir tiempo total de carga, entrenamiento y predicción
tiempo <- system.time({
```

```
  # Función para cargar los datos
  cargar_datos <- function(ruta) {
    data <- read.table(ruta, sep = "", strip.white = TRUE)
    etiquetas <- data[, 1]
    pixeles <- as.matrix(data[, 2:257])
    data.frame(label = etiquetas, pixeles)
  }

  # Rutas a los archivos
  train_path <- "C:/Users/Usuario/Desktop/TFG/zip.train.train"
  test_path  <- "C:/Users/Usuario/Desktop/TFG/zip.test.test"

  # Cargar los datos
  train <- cargar_datos(train_path)
  test  <- cargar_datos(test_path)

  # Separar etiquetas y características
  etiquetas_train <- train$label
  etiquetas_test  <- test$label
  x_train <- train[, -1]
  x_test  <- test[, -1]

  # Aplicar PCA sobre los datos de entrenamiento
  pca <- prcomp(x_train, center = TRUE, scale. = TRUE)

  # Determinar cuántos componentes explican el 95% de la
varianza
  varianza_explicada <- summary(pca)$importance[3, ]
  num_componentes <- which(cumsum(pca$sdev^2 / sum(pca$sdev^2))
>= 0.95)[1]

  # Transformar datos
  x_train_pca <- as.data.frame(pca$x[, 1:num_componentes])
  x_train_pca$label <- etiquetas_train

  # Transformar test con los mismos componentes
  x_test_pca <- as.data.frame(predict(pca, newdata = x_test)[,
1:num_componentes])
  x_test_pca$label <- etiquetas_test

  # Entrenar modelo RLM con datos reducidos
  log_model <- multinom(label ~ ., data = x_train_pca, MaxNWts =
10000, trace = FALSE)

  # Predecir y calcular precisión en el conjunto de
entrenamiento
  log_pred <- predict(log_model, x_train_pca[, -
ncol(x_train_pca)])
  print(1 - mean(log_pred == etiquetas_train));
```

```
  # Predecir y calcular precisión en el conjunto de prueba
  log_pred <- predict(log_model, x_test_pca[, -
ncol(x_test_pca)])
  print(1 - mean(log_pred == etiquetas_test));
})

# Mostrar tiempo
print(tiempo);

#Clasificacion de digitos manuscritos usando una tecnica propia

#Representacion grafica de los digitos binarizados
#Cargamos los datos sin simplificar, solo con el paso a blanco y
negro
# Función para cargar y binamizar píxeles
cargar_datos <- function(ruta) {
  data <- read.table(ruta, sep = "", strip.white = TRUE)
  etiquetas <- data[, 1]
  pixeles <- as.matrix(data[, 2:257])

  # Binarizar: ≤0 -> 0, >0 -> 1
  binarios <- ifelse(pixeles <= 0, 0, 1)

  data.frame(label = etiquetas, binarios)
}

# Rutas a tus archivos
train_path <- "C:/Users/Usuario/Desktop/TFG/zip.train.train"
test_path  <- "C:/Users/Usuario/Desktop/TFG/zip.test.test"

# Cargar conjuntos
train <- cargar_datos(train_path)
test  <- cargar_datos(test_path)

#Representacion grafica
# Seleccionar una observación de test (por ejemplo, la 121)
fila <- 121
vector <- as.numeric(test[fila, -1])
#Invertir pixeles para representacion
vector <- vector*(-1)
etiqueta_real <- test[fila, 1]

# Mostrar imagen
# Abrir el dispositivo gráfico PNG
png(filename = "C:/Users/Usuario/Desktop/TFG/Plots/Plot14.png",
    width = 480, height = 480, units = "px", pointsize = 12,
    bg = "white", res = NA, family = "", restoreConsole = TRUE,
    type = "cairo-png", antialias = "default", symbolfamily =
"default")
img <- as.cimg(vector)
```

```
plot(img, main = paste("Etiqueta real:", etiqueta_real))

# Cerrar el dispositivo gráfico
dev.off()

#LDA
# Medir tiempo total de carga, entrenamiento y predicción
tiempo <- system.time({

  # Función para cargar y simplificar los datos
  simplificar_datos <- function(ruta) {
    # Cargar los datos
    data <- read.table(ruta, sep = "", strip.white = TRUE)
    etiquetas <- data[, 1]
    pixeles <- as.matrix(data[, 2:257])

    # Binarizar: ≤0 -> 0, >0 -> 1
    binarios <- ifelse(pixeles <= 0, 0, 1)

    # Función para contar secuencias de 1's en una fila o
columna
    contar_secuencias <- function(v) {
      rle_vec <- rle(v == 1)
      sum(rle_vec$values)
    }

    # Para cada imagen, contar secuencias de unos por fila y
columna
    extraer_caracteristicas <- function(imagen) {
      matriz <- matrix(imagen, nrow = 16, byrow = TRUE)
      filas <- apply(matriz, 1, contar_secuencias)
      columnas <- apply(matriz, 2, contar_secuencias)
      c(filas, columnas)
    }

    # Aplicar a todas las filas
    caracteristicas <- t(apply(binarios, 1,
extraer_caracteristicas))

    # Crear data frame final
    df_final <- data.frame(label = etiquetas, caracteristicas)
    names(df_final)[-1] <- paste0("seq_", 1:32)
    return(df_final)
  }

  # Rutas a tus archivos
  train_path <- "C:/Users/Usuario/Desktop/TFG/zip.train.train"
  test_path  <- "C:/Users/Usuario/Desktop/TFG/zip.test.test"

  # Procesar los datos
```

```
  train <- simplificar_datos(train_path)
  test  <- simplificar_datos(test_path)

  # Entrenar modelo LDA con los datos simplificados
  lda_model <- lda(label ~ ., data = train)

  # Predecir sobre el conjunto de entrenamiento
  lda_pred <- predict(lda_model, train[, -1])$class

  # Evaluar precisión
  print(1 - mean(lda_pred == train$label));


  # Predecir sobre el conjunto de test
  lda_pred <- predict(lda_model, test[, -1])$class

  # Evaluar precisión
  print(1 - mean(lda_pred == test$label));
})

# Mostrar tiempo
print(tiempo);

#KSVM
# Medir tiempo total de carga, entrenamiento y predicción
tiempo <- system.time({

  # Función para cargar y simplificar los datos
  simplificar_datos <- function(ruta) {
    # Cargar los datos
    data <- read.table(ruta, sep = "", strip.white = TRUE)
    etiquetas <- data[, 1]
    pixeles <- as.matrix(data[, 2:257])

    # Binarizar: ≤0 -> 0, >0 -> 1
    binarios <- ifelse(pixeles <= 0, 0, 1)

    # Función para contar secuencias de 1's en una fila o
columna
    contar_secuencias <- function(v) {
      rle_vec <- rle(v == 1)
      sum(rle_vec$values)
    }

    # Para cada imagen, contar secuencias de unos por fila y
columna
    extraer_caracteristicas <- function(imagen) {
      matriz <- matrix(imagen, nrow = 16, byrow = TRUE)
      filas <- apply(matriz, 1, contar_secuencias)
      columnas <- apply(matriz, 2, contar_secuencias)
      c(filas, columnas)
```

```
    }

    # Aplicar a todas las filas
    caracteristicas <- t(apply(binarios, 1,
extraer_caracteristicas))

    # Crear data frame final
    df_final <- data.frame(label = as.factor(etiquetas),
caracteristicas)
    names(df_final)[-1] <- paste0("seq_", 1:32)
    return(df_final)
  }

  # Rutas a tus archivos
  train_path <- "C:/Users/Usuario/Desktop/TFG/zip.train.train"
  test_path  <- "C:/Users/Usuario/Desktop/TFG/zip.test.test"

  # Procesar los datos
  train <- simplificar_datos(train_path)
  test  <- simplificar_datos(test_path)

  # Entrenar modelo SVM con kernel radial
  svm_model <- ksvm(label ~ ., data = train, kernel = "rbfdot",
C = 1)#kernel radial (gaussiano)
  #opcion común para problemas de clasificación no lineal como
dígitos escritos a mano

  # Predecir sobre conjunto de entrenamiento
  svm_pred <- predict(svm_model, train[, -1])
  # Evaluar precisión
  print(1 - mean(svm_pred == train$label));

  # Predecir sobre conjunto de test
  svm_pred <- predict(svm_model, test[, -1])
  # Evaluar precisión
  print(1 - mean(svm_pred == test$label));
})

# Mostrar tiempo
print(tiempo);

#RLM
# Medir tiempo total de carga, entrenamiento y predicción
tiempo <- system.time({

  # Función para cargar y simplificar los datos
  simplificar_datos <- function(ruta) {
    # Cargar los datos
    data <- read.table(ruta, sep = "", strip.white = TRUE)
    etiquetas <- data[, 1]
    pixeles <- as.matrix(data[, 2:257])
```

```
    # Binarizar: ≤0 -> 0, >0 -> 1
    binarios <- ifelse(pixeles <= 0, 0, 1)

    # Función para contar secuencias de 1's en una fila o
columna
    contar_secuencias <- function(v) {
      rle_vec <- rle(v == 1)
      sum(rle_vec$values)
    }

    # Para cada imagen, contar secuencias de unos por fila y
columna
    extraer_caracteristicas <- function(imagen) {
      matriz <- matrix(imagen, nrow = 16, byrow = TRUE)
      filas <- apply(matriz, 1, contar_secuencias)
      columnas <- apply(matriz, 2, contar_secuencias)
      c(filas, columnas)
    }

    # Aplicar a todas las filas
    caracteristicas <- t(apply(binarios, 1,
extraer_caracteristicas))

    # Crear data frame final
    df_final <- data.frame(label = etiquetas, caracteristicas)
    names(df_final)[-1] <- paste0("seq_", 1:32)
    return(df_final)
  }

  # Rutas a tus archivos
  train_path <- "C:/Users/Usuario/Desktop/TFG/zip.train.train"
  test_path  <- "C:/Users/Usuario/Desktop/TFG/zip.test.test"

  # Procesar los datos
  train <- simplificar_datos(train_path)
  test  <- simplificar_datos(test_path)

  # Entrenar modelo RLM con los datos simplificados
  log_model <- multinom(label ~ ., data = train, MaxNWts =
10000, trace = FALSE)

  # Predecir sobre el conjunto de entrenamiento
  log_pred <- predict(log_model, train[, -1])

  # Evaluar precisión
  print(1 - mean(log_pred == train$label));

  # Predecir sobre el conjunto de test
  log_pred <- predict(log_model, test[, -1])
```

```
  # Evaluar precisión
  print(1 - mean(log_pred == test$label));
})

# Mostrar tiempo
print(tiempo);
```

Bibliografía

[1] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. (2021) An introduction to statistical learning with applications in R. *Springer Text in Statistic.*

[2] Genz A, Bretz F (2009). _Computation of Multivariate Normal and t Probabilities_, series Lecture Notes in Statistics. Springer-Verlag, Heidelberg. ISBN 978-3-642-01688 2.

[3] James G, Witten D, Hastie T, Tibshirani R (2021). _ISLR: Data for an Introduction to Statistical Learning with Applications in R_. R package version 1.4, <https://CRAN.R-project.org/package=ISLR>.

[4] Jeffrey S. Simonoff. (1996) Smoothing Methods in Statistics. *Springer Series in Statistic.*

[5] Karatzoglou A, Smola A, Hornik K (2023). _kernlab: Kernel-Based Machine Learning Lab_. R package version 0.9-32, <https://CRAN.R-project.org/package=kernlab>.

[6] Karatzoglou A, Smola A, Hornik K, Zeileis A (2004). “kernlab - An S4 Package for Kernel Methods in R.” _Journal of Statistical Software_, *11*(9), 1-20. doi:10.18637/jss.v011.i09 <https://doi.org/10.18637/jss.v011.i09>.

[7] Raymond H. Myers, Douglas C. Montgomery, G. Geoffrey Vining, Timothy J. Robinson. (2010) Generalized linear models : with applications in engineering and the sciences. *Wiley Series in Probability and Statistics.*

[8] R Core Team (2024). _R: A Language and Environment for Statistical Computing_. R Foundation for Statistical Computing, Vienna, Austria. <https://www.R-project.org/>.

[9] Trevor Hastie, Robert Tibshirani, Jerome Friedman. (2009) The elements of statistical learning Second Edition. *Springer Series in Statistic.*

[10] Venables, W. N. & Ripley, B. D. (2002) Modern Applied Statistics with S. Fourth Edition. Springer, New York. ISBN 0-387-95457-0.