FACULTAD DE MATEMÁTICAS

DEPARTAMENTO (Estadística e Investigación Operativa)

Trabajo Fin de Grado

Clasificación y su implementación en *R*

Carlos Castro Rey

Dirigido por:
Prof. Fernando López Blázquez

2018

Resumen

La clasificación de datos es un proceso de aprendizaje estadístico que trata de agrupar en categorías conocidas los diferentes casos considerados buscando alguna relación entre las características de individuos de la misma clase. A lo largo de este trabajo serán descritos tanto la construcción como el funcionamiento de algunas de las técnicas más habituales para abordar un problema de clasificación desarrolladas durante el siglo XX. Los clasificadores considerados estarán divididos en lineales, no lineales o árboles de clasificación. Para cada clasificador se realizará un ejemplo utilizando el paquete informático *R* con diferentes conjuntos de datos a fin de mostrar su implementación y ver las ventajas o desventajas del uso de cada uno de ellos. En dichos ejemplos veremos que esta clase de problemas se pueden encontrar en diversos ámbitos de estudio lo cual remarca la utilidad de este proceso.

Abstract

Classification is a statistical learning process which tries to group the different cases in known categories searching for a relationship between the features of the same class. Throughout this paper it will be described how to build a classifier and the performance of several of the most popular techniques developed during the 20^{th} century. These classifiers could be linear, nonlinear or classification trees. For every classifier there will be an example with different data set using the R software in order to show the implementation of these algorithms and the advantages and disadvantages of every one. These examples demonstrate that this kinds of problems could be found in various fields of study which indicates how useful this process is.

Índice general

Capítulo 1

Introducción a los métodos de clasificación

Un problema de clasificación consiste en tratar de predecir una variable categórica a partir de un vector aleatorio. En este capítulo se detalla como funciona la clasificación de datos, los elementos que aparecen en los problemas de clasificación y además sentaremos las bases para evaluar los clasificadores que desarrollaremos en los capítulos posteriores.

1.1. Variables y conjuntos de datos

Partimos de una variable aleatoria discreta Y, no degenerada y con soporte finito y un vector aleatorio p-dimensional $\underline{X} = (X_1, X_2, \ldots, X_p)^t$. Además consideraremos los siguientes conjuntos de posibles valores de las variables:

$$\mathcal{Y} = \{y \in \mathbb{R} : \mathbb{P}(Y = y) \neq 0\},$$

$$\mathcal{X} = \{\underline{x} = (x_1, x_2, \ldots, x_p)^t \in \mathbb{R}^p : x_i \in Im(X_i), \forall i = 1, 2, \ldots, p\}.$$

La variable aleatoria Y es conocida como variable objetivo. Cada elemento $y \in \mathcal{Y}$ corresponde a una clase o categoría diferente. Denotando $|\mathcal{Y}|$ como el cardinal del conjunto, si $|\mathcal{Y}| = 2$, tenemos un problema de *clasificación binario* y si el número de clases $|\mathcal{Y}| > 2$, el problema es de *clasificación múltiple*.

Por otro lado, las variables aleatorias que componen el vector aleatorio $\underline{X}$ son denominadas *variables explicativas*.

En los ejemplos que abordaremos, dispondremos de una muestra aleatoria $\{(\underline{x}_i, y_i)\}_{i=1}^n$ de $(\underline{X}, Y)$. Estas observaciones estarán reflejadas dentro de una matriz $\mathbf{X}$ llamada *matriz de datos*, donde cada fila de la matriz anterior representará una realización muestral de las variables anteriores, es decir:

$$\mathbf{X} = \begin{pmatrix} \underline{x}_{1,1} & \cdots & \underline{x}_{1,p} & y_1 \\ \vdots & \ddots & \vdots & \vdots \\ \underline{x}_{n,1} & \cdots & \underline{x}_{n,p} & y_n \end{pmatrix}.$$

La clasificación de datos se puede entender como un proceso de dos pasos: el primero sería una fase de entrenamiento o aprendizaje, donde se seleccionan $m < n$ realizaciones muestrales contenidas en la matriz de datos anterior y se utilizan para desarrollar el algoritmo de clasificación. A este subconjunto de datos se le llama *datos de entrenamiento*.

La segunda etapa es una fase de clasificación, dónde una vez desarrollado el algoritmo de clasificación se procede a usar el resto de los datos para observar a que clase son asignados los mismos y poder evaluar los errores que se cometen al clasificar, al mismo tiempo se conseguirá tener una referencia del error que se cometerá si se incluyeran otros datos diferentes más adelante. Este segundo conjunto de datos se conoce como *datos de prueba*.

En la práctica el conjunto de datos de prueba puede ser tomado de diversas formas. Aquí se exponen algunas de ellas:

- *Método Hold-out*. Consiste en usar dos tercios de los datos para construir el clasificador y el resto de los datos emplearlos como datos de prueba.
- *Submuestreo aleatorio (Random subsampling)*. Repetir el método Hold-out un número k de veces manteniendo la misma proporción pero cambiando los datos tomados para entrenamiento y prueba.
- *Validación cruzada de k iteraciones*. Se establecen k subconjuntos $T_1, T_2, \ldots, T_k$ de tamaño aproximadamente igual en los datos. En la iteración i se utiliza T_i como conjunto de prueba y el resto para el conjunto de entrenamiento. Con $1 \leq i \leq k$.
- *Validación cruzada dejando uno fuera (Jackknife)*. En este caso se toman todos los datos para entrenar el modelo menos uno, que es el utilizado como único dato de prueba. Este proceso se repite tantas veces como número de datos de entrenamiento dispongamos. En cada iteración se utiliza un dato de prueba diferente a los anteriores.

1.2. Análisis de componentes principales

El análisis de componentes principales (ACP) fue inicialmente desarrollado por Pearson a finales del siglo XIX y profundizado por Hotelling en los años 30 ya en el siglo XX. Aunque el ACP no se utiliza para la clasificación, nos proporciona una reducción de dimensión en los datos facilitando la tarea de clasificar.

El ACP es una técnica no supervisada (no tiene en cuenta la variable objetivo) que está basada en el estudio de la matriz de varianzas y covarianzas o de la matriz de correlaciones de un conjunto de datos para obtener un sistema de variables nuevas denominadas componentes principales.

Estas nuevas variables se definen mediante una combinación lineal de las anteriores y con ello buscamos conseguir la menor pérdida de información posible acorde al criterio de varianza total. De esta forma podemos sintetizar los datos y relacionarlos entre sí, para ello suponemos que las variables son de tipo cuantitativo.

Las componentes principales se construyen para que sean incorreladas entre sí. En el caso de que las variables iniciales se encuentren fuertemente relacionadas, la mayor parte de la variabilidad de las mismas se podrá explicar con un conjunto más reducido de nuevas variables. Si las variables iniciales fueran incorreladas, este proceso no tendría sentido pues todas las variables serían necesarias.

Veamos ahora el proceso de construcción de las componentes principales.

Consideramos un vector aleatorio $\underline{X} = (X_1, \ldots, X_p)'$ y sea Σ la matriz de varianzas y covarianzas que es simétrica y semidefinida positiva. Por ser una combinación lineal, la primera variable queda definida $U_1 = \underline{w}_1^t \underline{X}$. Para determinar $\underline{w}_1 \in \mathbb{R}^n$ tendremos en cuenta que buscamos la máxima varianza, luego:

$$\max_{\underline{w}_1 \in \mathbb{R}^p} Var(\underline{w}_1^t \underline{X}) = \max_{\underline{w}_1 \in \mathbb{R}p} \underline{w}_1^t \Sigma \underline{w}_1.$$

Para determinar la unicidad de solución, ya que solo estamos interesados en la dirección del vector, podemos restringir $\underline{w}_1$ de forma que $\| \underline{w}_1 \| = 1$.

Para encontrar la primera componente principal basta con resolver el problema de optimización no lineal siguiente empleando el método de los multiplicadores de Lagrange:

$$\max_{\underline{w}_1 \in \mathbb{R}^p} \underline{w}_1^t \Sigma \underline{w}_1 - \alpha(\underline{w}_1^t \underline{w}_1 - 1), \qquad \alpha \in \mathbb{R}.$$

Derivando la función objetivo con respecto a $\underline{w}_1$ e igualando a cero se obtiene:

$$\Sigma \underline{w}_1 = \alpha \underline{w}_1,$$

de donde se sigue que α es un autovalor de la matriz Σ y que la primera dirección principal $\underline{w}_1$ es el autovector unitario asociado al autovalor α. Para decidir que autovalor es el adecuado, usando la igualdad anterior se tiene:

$$\max_{\underline{w}_1} \underline{w}_1^t \Sigma \underline{w}_1 = \max_{\underline{w}_1} \underline{w}_1^t \alpha \underline{w}_1,$$

de donde se deduce que para maximizar esa función objetivo hay que tomar el mayor autovalor de la matriz Σ denotado λ_1 y $\underline{w}_1$ corresponderá al autovector unitario asociado.

La segunda componente principal $\underline{w}_2$ se obtiene añadiendo una condición adicional al problema anterior. En este caso impondremos que $\underline{w}_2^t \underline{w}_1 = 0$.

De nuevo hay que resolver el siguiente problema por el mismo método:

$$\max_{\underline{w}_2} \underline{w}_2^t \Sigma \underline{w}_2 - \alpha(\underline{w}_2^t \underline{w}_2 - 1) - \beta \underline{w}_2^t \underline{w}_1, \qquad \alpha, \beta \in \mathbb{R},$$

derivando con respecto a $\underline{w}_2$ e igualando a 0:

$$2\Sigma \underline{w}_2 - 2\alpha \underline{w}_2 - \beta \underline{w}_1 = 0,$$

multiplicando por $\underline{w}_1^t$ a izquierda en ambos lados nos queda:

$$2\underline{w}_1^t \Sigma \underline{w}_2 = \beta.$$

Se observa que $\underline{w}_1^t \Sigma \underline{w}_2 = \underline{w}_2^t \Sigma \underline{w}_1$ por ser un escalar. Además, ya que $\underline{w}_1$ está determinado por ser la primera componente principal cumple que:

$$\underline{w}_2^t \Sigma \underline{w}_1 = \underline{w}_2^t \lambda_1 \underline{w}_1 = 0,$$

de donde se deduce que $\beta = 0$. Por lo tanto, el problema queda reducido al siguiente:

$$\Sigma \underline{w}_2 = \alpha \underline{w}_2$$

que razonando de manera análoga al caso anterior obtenemos que $\alpha = \lambda_2$ siendo este el segundo autovalor más grande y $\underline{w}_2$ el autovector unitario asociado. Nótese que la ortogonalidad de $\underline{w}_2$ con $\underline{w}_1$ se tiene también debido a que la matriz Σ es simétrica.

De manera más general, las componentes principales se obtienen resolviendo el problema de optimización:

$$\max_{\underline{w}_k \in \mathbb{R}^p} \underline{w}_k^t \Sigma \underline{w}_k - \alpha_k(\underline{w}_k^t \underline{w}_k - 1) - \sum_{i=1}^{k-1} \beta_i \underline{w}_k^t \underline{w}_i$$
$$\alpha_k, \beta_i \in \mathbb{R}\ \forall k = 2, \ldots, n,\ \forall i = 1, \ldots, k-1.$$

En este problema estamos imponiendo que todas las componentes principales han de ser ortogonales dos a dos y unitarias. La solución de este problema, razonando de manera similar a los casos anteriores, viene dada por $\alpha_k = \lambda_k$ (k-ésimo autovalor más grande), $\beta_i = 0$, $\forall i = 1, \ldots, k-1$ y $\underline{w}_k$ corresponde al autovector unitario asociado a λ_k.

Sea $\mathbf{W}$ la matriz cuyas filas son los autovectores $\underline{w}_1, \ldots, \underline{w}_p$ ($\mathbf{W}$ es por tanto unitaria y $\mathbf{W}\mathbf{W}^t = I_p$), las nuevas variables son las componentes del siguiente vector:

$$\underline{U} = \mathbf{W}^t \underline{X}.$$

Es posible establecer una relación entre la matriz Σ y la matriz de varianzas y covarianzas de $\underline{U}$. Si denotamos $\mathbf{D}$ como la matriz de varianzas y covarianzas de $\underline{U}$ tenemos:

$$\mathbf{D} = Var(\underline{U}) = Var(\mathbf{W}^t \underline{X}) = \mathbf{W}^t \Sigma \mathbf{W}.$$

Veamos ahora algunas propiedades de las nuevas variables:

- $Var(U_i) = \lambda_i$, $\forall i = 1, \ldots, p$, pues

$$Var(U_i) = Var(\underline{w}_i \underline{X}) = \underline{w}_i^t \Sigma \underline{w}_i = \underline{w}_i^t \lambda_i \underline{w}_i = \lambda_i.$$

- $\sum_{i=1}^n Var(U_i) = \sum_{i=1}^n Var(X_i)$. Se tiene:

$$\sum_{i=1}^p Var(U_i) = \sum_{i=1}^p \lambda_i = traza(\mathbf{D}) = traza(\mathbf{W}^t \Sigma \mathbf{W}) =$$

$$traza(\mathbf{W}\mathbf{W}^t \Sigma) = traza(\Sigma) = \sum_{i=1}^p Var(X_i).$$

- $Cov(U_i, U_j) = 0, \quad \forall 1 \leq i, j \leq p$ con $i \neq j$.

$$Cov(U_i, U_j) = Cov(\underline{w}_i \underline{X}, \underline{w}_j \underline{X}) = \underline{w}_i^t \Sigma \underline{w}_j = \lambda_j \underline{w}_i^t \underline{w}_j = 0.$$

La segunda propiedad nos permite saber qué proporción de la varianza total de $\underline{X}$ es recogida por cada variable U_k:

$$\frac{\lambda_k}{\sum_{i=1}^n \lambda_i}, \quad 1 \leq k \leq p.$$

También es posible expresar la variabilidad de las primeras m variables con respecto al total:

$$\frac{\sum_{j=1}^m \lambda_j}{\sum_{i=1}^p \lambda_i}, \quad 1 \leq m \leq p.$$

En la práctica trabajaremos con una muestra aleatoria $\underline{X}_1, \ldots, \underline{X}_n$ del vector $\underline{X}$ expuesta en una matriz de datos $\mathbf{X}$. Utilizaremos entonces la matriz de cuasivarianzas muestrales denotada $\widehat{\Sigma}$ como estimador de Σ ya que desconocemos la distribución de $\underline{X}$, y ésta será la matriz a la que se le calcularán los autovalores y autovectores para realizar el ACP.

Sean $\hat{\lambda}_1, \ldots, \hat{\lambda}_p$ los autovalores de la matriz $\widehat{\Sigma}$ y $\hat{w}_1, \ldots, \hat{w}_p$ sus autovectores asociados respectivamente. Denotaremos por $\widehat{\mathbf{W}}$ la matriz cuyas columnas son los autovectores de la matriz $\widehat{\Sigma}$ ordenados de mayor a menor autovalor asociado. Las puntuaciones de cada realización muestral puede ser expresada mediante la siguiente matriz:

$$\mathbf{U} = \mathbf{X}\widehat{\mathbf{W}}.$$

Como $\widehat{\Sigma}$ es semidefinida positiva, entonces $\hat{\lambda}_i \geq 0$ con $i = 1, \ldots, n$. Puesto que, el objetivo del ACP es tomar aquellas direcciones que maximicen la varianza, se puede prescindir de aquellas componentes con un autovalor igual o cercano a 0, ya que se entiende que la contribución de dichas componentes a la varianza total es muy pequeña y en consecuencia, esa pérdida no es significativa. Con esto, se conseguiría disminuir la dimensión del problema inicial planteado.

Existen diversos métodos para seleccionar el número de componentes principales, algunos de ellos son:

- Seleccionar componentes hasta que el porcentaje de varianza acumulado sea mayor que un umbral estipulado.
- Descartar aquellas componentes cuyo autovalor sea menor que la media de los autovalores.
- Establecer un gráfico donde a cada entero positivo se le asocia el autovalor correspondiente (gráfico decreciente tendiendo a 0) y escoger aquellos que se encuentren antes de que se estabilice.
- Emplear un contraste de hipótesis paramétrico.

En el caso de que las varianzas muestrales de los datos iniciales varíen demasiado, es frecuente tipificar la matriz de varianzas y covarianzas para tratar de evitar las diferencias creadas por la forma de medir los datos. En este caso los autovalores para buscar las componentes principales se calculan a partir de la matriz de varianzas y covarianzas tipificada o de la matriz de correlaciones muestrales.

Veamos un ejemplo implementado en R. Tenemos una muestra[1] de 8143 mediciones para saber si una habitación se encuentra vacía u ocupada en función de las siguientes variables.

- $X_1 \equiv$ Fecha y hora de la medición
- $X_2 \equiv$ Temperatura en grados Celsius.
- $X_3 \equiv$ Humedad relativa en porcentajes.

[1] Base de datos en https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+

- $X_4 \equiv$ Cantidad de luz.
- $X_5 \equiv$ Cantidad de CO_2.
- $X_6 \equiv$ Índice de humedad.
- $Y \equiv$ Ocupación (1 ocupada, 0 vacía).

Prescindimos de la variable X_1 y buscamos las componentes principales de las otras 5. En primer lugar observamos la matriz de varianzas y covarianzas muestrales.

```
habitaciones = read.table("habitaciones.txt",sep = ",", header = TRUE)
head(habitaciones)
habita = habitaciones[,-1]
colnames(habita) = c(paste("X",2:6,sep =""),"Y")
summary(habita)
sigma = cov(habita[,-6])
round(sigma,4)
print(xtable(sigma, digits = 4))
```

	X_2	X_3	X_4	X_5	X_6
X_2	1.0341	-0.7974	128.7212	178.9634	0.0001
X_3	-0.7974	30.5943	40.7496	763.2740	0.0045
X_4	128.7212	40.7496	37929.8235	40648.6516	0.0382
X_5	178.9634	763.2740	40648.6516	98797.6136	0.1679
X_6	0.0001	0.0045	0.0382	0.1679	0.0000

Cuadro 1.1: Matriz de varianzas y covarianzas muestrales.

La amplia diferencia entre las distintas varianzas muestrales de las variables sugiere que es preferible usar la matriz de correlaciones muestrales para el cálculo de las componentes principales.

```
R = cor(habita[,-6])
print(xtable(R, digits = 4))
```

	X_2	X_3	X_4	X_5	X_6
X_2	1.0000	-0.1418	0.6499	0.5599	0.1518
X_3	-0.1418	1.0000	0.0378	0.4390	0.9552
X_4	0.6499	0.0378	1.0000	0.6640	0.2304
X_5	0.5599	0.4390	0.6640	1.0000	0.6266
X_6	0.1518	0.9552	0.2304	0.6266	1.0000

Cuadro 1.2: Matriz de correlaciones muestrales.

Para calcular las componentes principales en R se utilizará la función *princomp*.

```
comp = princomp(habita[,-6],cor=TRUE)
summary(comp)
x = comp$loadings[1:5,]
print(xtable(x, digits = 4))
```

	Comp.1	Comp.2	Comp.3	Comp.4	Comp.5
X1	0.3439	0.5359	0.7134	0.2254	0.1869
X2	0.3957	-0.5741	-0.0092	0.2270	0.6799
X3	0.4141	0.4446	-0.6654	0.4332	-0.0192
X4	0.5501	0.1201	-0.1109	-0.8173	0.0526
X5	0.5011	-0.4137	0.1895	0.2052	-0.7069

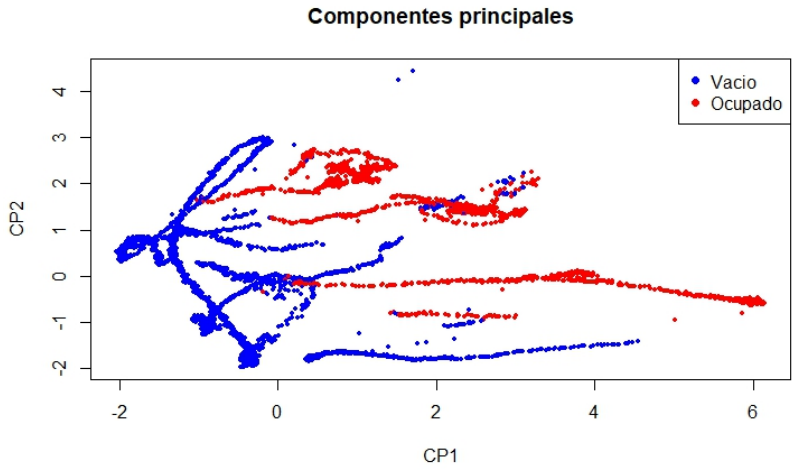
Para conseguir la información que aportan los autovalores para la variabilidad expresada por las componentes principales usamos la función *summary* que nos hace un resumen de lo obtenido.

```
v = (comp$sdev)^2
v1 = v/sum(v)
v2 = cumsum(v)/sum(v)
avalores = rbind(v,v1,v2)
rownames(avalores) = c("Autovalores","Proporcion de Varianza",
"Varianza Acumulada")
print(xtable(avalores,digits = 4))
```

	Comp.1	Comp.2	Comp.3	Comp.4	Comp.5
Autovalores	2.7365	1.6995	0.3488	0.2144	0.0008
Proporcion de Varianza	0.5473	0.3399	0.0698	0.0429	0.0002
Varianza Acumulada	0.5473	0.8872	0.9570	0.9998	1.0000

Si elegimos un umbral en el 80 % de la varianza total, basta con usar las dos primeras componentes ya que éstas expresan el 88,72 % de la varianza total. Pasaríamos por tanto de un problema con 5 variables a otro con sólo 2 consiguiendo así una reducción de tres dimensiones y la posibilidad de representar estos puntos en el plano.

```
puntos = cbind(comp$scores[,1:2],habita[,6])
puntos[,3] = as.factor(puntos[,3])
colores = c("blue", "red")
colores = colores[as.numeric(puntos[,3])]
plot(puntos[,1:2],cex= 0.6, pch = 16, xlab = "CP1", ylab = "CP2",
main = "Componentes principales",col = colores)
legend("topright", legend =c("Vacio" ,"Ocupado"),
col=c(" blue ","red"),pch = 16)
```



Aquí podemos ver una representación de los puntos usando las dos primeras componentes principales teniendo en cuenta si la habitación en cuestión está ocupada o vacía. Esta técnica de reducción de dimensión será muy útil si disponemos de un número amplio de variables.

1.3. Construcción y evaluación de un clasificador

Un algoritmo de clasificación es un procedimiento constructivo que, a partir de una cantidad finita de instrucciones ordenadas, relaciona un valor de las variables explicativas con un valor de la variable de respuesta.

Definición 1.3.1. Llamaremos clasificador a la función que lleve a cabo un algoritmo de clasificación dado, es decir, una función h de la forma:

$$\begin{aligned} h : \mathcal{X} &\longmapsto \mathcal{Y} \\ \underline{x} &\longmapsto h(\underline{x}) = y. \end{aligned}$$

Usando los datos de entrenamiento, un clasificador establecerá los lugares del espacio asociados a las diferentes clases.

Definición 1.3.2. Sea h un clasificador, llamaremos región de decisión asociada a la clase $y \in \mathcal{Y}$ a:

$$R_y = \{\underline{x} \in \mathbb{R}^p : h(\underline{x}) = y\} \quad \forall y \in \mathcal{Y}.$$

Aquellos vectores que separen dos o más regiones de decisión se denominan límite de decisión. El límite de decisión representa aquellos vectores $\underline{x} \in \mathbb{R}^p$ donde $h(\underline{x})$ tiene más de una solución. En ese caso, dicho vector se asignará a cualquiera de las regiones donde exista ese empate.

Definidas las regiones de decisión y el límite de decisión buscamos comprobar la capacidad de predicción que tiene el clasificador que estamos empleando. Para ello utilizamos el conjunto de datos de prueba, ya que son datos que no se emplean para crear las regiones de decisión y además es conocida la variable de respuesta en todos ellos. Permitiendo de esa forma comparar el resultado que devuelve el clasificador con el dato real.

Definición 1.3.3. El error de h se define como:

$$\varepsilon(h) = \mathbb{P}(h(\underline{X}) \neq Y).$$

El cálculo del error anterior no es, en general, posible de realizar. Para calcular la probabilidad anterior debemos conocer las distribuciones de las variables sin embargo, la única información que poseemos, en principio, de Y y $\underline{X}$ es la que recibimos a partir de los datos.

Definición 1.3.4. Dadas m realizaciones muestrales $(\underline{x}_1, y_1), \ldots, (\underline{x}_m, y_m)$ de $(\underline{X}, Y)$ tomadas como conjunto de datos de entrenamiento, llamaremos error empírico o error de entrenamiento a:

$$\varepsilon_T(h) = \frac{1}{m} \sum_{i=1}^{m} I(h(\underline{x}_i) \neq y_i),$$

donde

$$I(h(\underline{x}_i) \neq y_i) = \begin{cases} 1 & si \quad h(\underline{x}_i) \neq y_i \\ 0 & si \quad h(\underline{x}_i) = y_i \end{cases} \qquad \forall i = 1, \ldots, m.$$

Si $I(h(\underline{x}_i) \neq y_i) = 1$ para algún $i = 1, \ldots, m$, entenderemos que el clasificador h está cometiendo un error de clasificación en el dato de entrenamiento $(\underline{x}_i, y_i)$.

Aunque tratamos de encontrar un error empírico cercano a 0, emplear un número excesivo de datos de entrenamiento puede producir un problema de sobreajuste, es decir, el clasificador se ha aproximado demasiado a los datos que disponemos y ésto hace que sea menos probable que realice buenas clasificaciones de nuevos datos.

Tomando aquellas realizaciones muestrales que no hemos considerado para los datos de entrenamiento, construimos el error de prueba.

Definición 1.3.5. Se define el error de prueba como:

$$\varepsilon_P(h) = \frac{1}{n-m} \sum_{i=1}^{n-m} I(h(\underline{x}_i) \neq y_i).$$

Con la función I de la definición anterior.

Se puede probar que tanto el error de prueba como el error de entrenamiento son estimadores del error del clasificador (**Definición 1.3.3**). Además de estudiar el error de prueba, existen otras formas de medir la capacidad de predicción de un clasificador, para ello emplearemos la siguiente matriz.

Definición 1.3.6. Sea $|\mathcal{Y}| = c$ el número total de clases de un problema de clasificación con $c \geq 2$. Una matriz de confusión $M \in \mathbb{R}^{c\times c}$ es una matriz donde las entradas $m_{i,j}$ denotan el número de datos de prueba cuya variable de respuesta es la clase j y que han sido clasificados en la categoría i, $\forall i, j = 1, \ldots, c$.

De esta definición se deduce que todas las entradas que se encuentren en la diagonal de M son aquellos datos que han sido clasificados correctamente y los elementos que se encuentren fuera de ella son los clasificados erróneamente.

Definición 1.3.7. La exactitud *(accuracy)* de un clasificador h se define como:

$$\text{exactitud}(h) = \frac{\sum_{i=1}^{c} m_{i,i}}{\sum_{i,j=1}^{c} m_{i,j}}.$$

La exactitud se interpreta como la capacidad que tiene un clasificador para predecir correctamente los datos. Para obtener un buen clasificador buscamos una exactitud próxima a 1, es decir, que todos los elementos fuera de la diagonal de M sean 0 ó cercanos a 0. Se puede observar que ε_P+ exactitud$(h) = 1$.

Hay que tener en cuenta que una buena exactitud de un clasificador no siempre es un buen indicador. Por ejemplo, consideramos 1000 datos de prueba en un problema de clasificación binario donde 900 pertenecen a la clase A y los 100 restantes a la clase B. El clasificador que asocia todos los datos a la clase A tiene una exactitud de 0,9 pero es incapaz de predecir un dato del tipo B.

Realidad / Clasificado	Positivo	Negativo
Positivo	Verdadero Positivo (VP)	Falso Positivo (FP)
Negativo	Falso Negativo (FN)	Verdadero Negativo (VN)

En el esquema anterior, tenemos descrito un problema de clasificación binario (positivo-negativo), definiremos los siguientes conceptos a través de él:

$$\text{exactitud}(h) = \frac{VP + VN}{VP + VN + FP + FN}.$$

Vamos a tratar de obtener información de las predicciones que se realizan en las diferentes clases estudiándolas de manera individual. Para ello, introducimos la sensitividad y la especificidad.

$$\text{Sensitividad} = \frac{VP}{VP + FN}.$$

La sensitividad mide la proporción de predicciones positivas correctas.

$$\text{Especificidad} = \frac{VN}{VN + FP}.$$

La especificidad mide la proporción de los valores clasificados negativos que son correctos. También resultará de interés estudiar el valor predictivo positivo y negativo que son:

$$\text{Valor Predictivo Positivo} = \frac{VP}{VP + FP}.$$

El valor predictivo positivo calcula, de todos los valores que han sido clasificados en la categoría positivo, cuantos de ellos han sido clasificados correctamente. Análogamente:

$$\text{Valor Predictivo Negativo} = \frac{VN}{VN + FN},$$

que al igual que el valor predictivo positivo, el negativo nos especifica aquellos valores que fueron correctamente clasificados en la categoría negativo. Trataremos de generalizar estos conceptos de evaluación a problemas de clasificación múltiple. El enfoque que seguiremos es el de uno contra todos, que significa que si un problema tiene c clases, para una clase $y_i \in \mathcal{Y}$ con $i = 1, \ldots, c$ definimos:

- Verdaderos positivos $\equiv$ Valores correctamente predichos en la clase y_i.
- Verdaderos negativos $\equiv$ Valores correctamente predichos en cualquier otra clase y_j, $j = 1, \ldots, c,\ j \neq i$.
- Falsos positivos $\equiv$ Valores cuya clase es y_i y han sido clasificados erróneamente en otra clase.
- Falsos negativos $\equiv$ Valores que han sido mal clasificados en la clase y_i.

Con estas condiciones, dada una matriz de confusión podemos generalizar los conceptos anteriores de la manera siguiente:

$$\text{Sensitividad}_{y_i} = \frac{m_{i,i}}{\sum_{j=1} m_{j,i}}; \quad \text{Especificidad}_{y_i} = \frac{\sum_{j=1, j\neq i}^{c} m_{j,j}}{\sum_{j=1, j\neq i}^{c} m_{j,j} + m_{i,j}} \quad \forall i = 1, \ldots, c.$$

De la misma forma podemos generalizar los dos tipos de valores predictivos (VPP y VPN respectivamente):

$$\text{VPP}_{y_i} = \frac{m_{i,i}}{m_{i,i} + \sum_{j=1, j\neq i}^{c} m_{i,j}}; \quad \text{VPN}_{y_i} = \frac{\sum_{j=1, j\neq i}^{c} m_{j,j}}{\sum_{j=1, j\neq i}^{c} m_{j,j} + m_{j,i}} \quad \forall i = 1, \ldots, c.$$

Con los errores y las definiciones anteriores podemos estudiar la eficiencia de un clasificador basándonos en los datos de entrenamiento y prueba.

Otro aspecto importante es la elección del modelo. Cuando se construye un clasificador podemos hablar de dos aspectos importantes como son el sesgo y la varianza.

El sesgo es el error que se comete al aproximar un problema real a partir de un modelo estadístico más simple. Por otro lado, la varianza mide el cambio producido por un modelo si se emplean diferentes conjuntos de entrenamiento de una misma muestra aleatoria.

Los clasificadores lineales tienen en general una varianza muy baja pero un sesgo muy alto, ya que hacer una suposición de linealidad es altamente improbable para un problema real. Los clasificadores no lineales y los árboles de clasificación suelen tener un sesgo bajo pero una varianza muy alta pues son sensibles a los cambios producidos dentro de los datos de entrenamiento.

El objetivo consistirá en elegir los modelos de manera que el sesgo y la varianza estén balanceados dentro de las características de cada uno.

1.4. El clasificador de Bayes

Otra forma de abordar un problema de clasificación es mediante el uso de funciones discriminantes. Denotaremos g_y a la función discriminante asociada a la clase $y \in \mathcal{Y}$. El clasificador asignará al vector $\underline{x}$ la clase y^* si se cumple:

$$g_{y^*}(\underline{x}) \geq g_y(\underline{x}) \quad \forall y \in \mathcal{Y}.$$

Claramente, con esta definición, la elección de funciones discriminantes no es única, pues si a todas ellas les sumamos un número real cualquiera, las multiplicamos por una constante positiva o las componemos con cualquier función f que sea monótona creciente, el resultado de clasificación no varía.

Aunque las funciones discriminantes puedan ser escritas de varias formas, las regiones de decisión son equivalentes, ya que estas regiones son de la forma:

$$R_{y^*} = \{\underline{x} \in \mathbb{R}^n : g_{y^*}(\underline{x}) \geq g_y(\underline{x}),\ \forall y \in \mathcal{Y}\}.$$

Al igual que antes, los límites de decisión serán aquellos vectores $\underline{x} \in \mathbb{R}^n$ donde el valor máximo se alcance en más de una función discriminante. Estos vectores serán clasificados aleatoriamente entre cualquiera de las clases donde exista este empate.

En esta sección vamos a detallar el clasificador de Bayes, que está basado en la probabilidad condicionada y que emplea esta clase de funciones.

Para este caso, definimos la función discriminante asociada a la clase $y \in \mathcal{Y}$ como:

$$g_y(\underline{x}) = \mathbb{P}(Y = y | \underline{X} = \underline{x}).$$

El algoritmo de clasificación consistirá entonces en asignar al vector $\underline{x} \in \mathcal{X}$ aquella clase $y \in \mathcal{Y}$ para la cual la probabilidad $\mathbb{P}(Y = y|\, \underline{X} = \underline{x})$ es máxima.

De esta forma definimos el clasificador de Bayes h^* como sigue:

$$h^*(\underline{x}) = \arg\max_{y \in \mathcal{Y}} g_y(\underline{x}) = \arg\max_{y \in \mathcal{Y}} \mathbb{P}(Y = y|\, \underline{X} = \underline{x}).$$

Recordamos que el teorema de Bayes permite convertir la probabilidad anterior en:

$$\mathbb{P}(Y = y|\, \underline{X} = \underline{x}) = \frac{f_{\underline{X}|Y=y}(\underline{x})\mathbb{P}(Y = y)}{f_{\underline{X}}(\underline{x})} = \frac{f_{\underline{X}|Y=y}(\underline{x})\pi(y)}{f_{\underline{X}}(\underline{x})},$$

donde a la probabilidad $\mathbb{P}(Y = y|\, \underline{X} = \underline{x})$ se conoce como probabilidad *a posteriori* y al miembro $\mathbb{P}(Y = y) = \pi(y)$ se denomina probabilidad *a priori*.

Se pueden dar algunas versiones alternativas de las funciones discriminantes para este clasificador, por ejemplo:

- Eliminando el denominador: $g_y(\underline{x}) = f_{\underline{X}|Y=y}(\underline{x})\mathbb{P}(Y = y)$.
- Aplicando logaritmos: $g_y(\underline{x}) = \log(f_{\underline{X}|Y=y}(\underline{x})) + \log(\mathbb{P}(Y = y))$.

Siendo $f_{\underline{X}|Y=y}(\underline{x})$ la función de densidad o de probabilidad del vector aleatorio $\underline{X}$ condicionada a la clase $y \in \mathcal{Y}$ en el caso que proceda.

Notamos también que maximizar la probabilidad $\mathbb{P}(Y = y|\, \underline{X} = \underline{x})$ equivale a:

$$\max_{y \in \mathcal{Y}} \mathbb{P}(Y = y|\, \underline{X} = \underline{x}) = \max_{y \in \mathcal{Y}} \frac{f_{\underline{X}|Y=y}(\underline{x})\pi(y)}{f_{\underline{X}}(\underline{x})} = \max_{y \in \mathcal{Y}} f_{\underline{X}|Y=y}(\underline{x})\pi(y).$$

Teorema 1.4.1. Sea $\mathcal{Y} = \{y_1, \ldots, y_c\}$ y h un clasificador arbitrario. Entonces:

$$\varepsilon(h^*) \leq \varepsilon(h), \quad \text{para cualquier clasificador } h.$$

Es decir, el error del clasificador de Bayes es mínimo.

Demostración. El objetivo de esta prueba consiste en encontrar unas regiones $\{R_1, \ldots, R_c\}$ de un espacio R tales que $\bigcup_{i=1}^{c} R_i = R$ con $R_i \cap R_j = \emptyset$ si $i \neq j$ e $i, j = 1, \ldots, c$ que minimicen $\mathbb{P}(h(\underline{X}) \neq Y)$ y comprobar que esas regiones son las que se generan al utilizar el clasificador de Bayes.

Un error de clasificación ocurre cuando un vector $\underline{x}$ que pertenece a la clase y_i es asignado a una clase y_j. $i, j = 1, \ldots, c$ con $i \neq j$. Sea $f_{\underline{X}|Y=y_i}(\underline{x})$ la función de densidad del vector aleatorio $\underline{X}$ condicionada a la clase y_i con $i = 1, \ldots, c$, la probabilidad de cometer el error anterior es:

$$\int_{R_j} f_{\underline{X}|Y=y_i}(\underline{x})\, d\underline{x}, \quad \forall i, j = 1, \ldots, c, \quad i \neq j.$$

De manera más general la probabilidad de clasificar erróneamente un vector perteneciente a la clase y_i.

$$\mathbb{P}(\text{no asignar en } y_i) = \sum_{j=1, j\neq i}^{c} \int_{R_j} f_{\underline{X}|Y=y_i}(\underline{x})\, d\underline{x} = 1 - \int_{R_i} f_{\underline{X}|Y=y_i}(\underline{x})\, d\underline{x} \quad \forall i = 1, \ldots, c.$$

En total para todas las clases:

$$\mathbb{P}(h(\underline{x}) \neq y_i) = \sum_{i=1}^{c} \pi(y_i)(1 - \int_{R_i} f_{\underline{X}|Y=y_i}(\underline{x}))\, d\underline{x} = 1 - \sum_{i=1}^{c} \int_{R_i} \pi(y_i) f_{\underline{X}|Y=y_i}(\underline{x})\, d\underline{x}.$$

Definimos la función indicadora $\chi_A(\underline{x}) = 1$ si $\underline{x} \in A \subset \mathbb{R}^p$ y 0 en caso contrario. Dadas las regiones $\{R_1, \ldots, R_c\}$ obtenidas a partir de un clasificador h cualquiera se tiene:

$$\sum_{i=1}^{c} \int_{R_i} \pi(y_i) f_{\underline{X}|Y=y_i}(\underline{x})\, d\underline{x} = \sum_{i=1}^{c} \int_{R} \chi_{R_i}(\underline{x}) \pi(y_i) f_{\underline{X}|Y=y_i}(\underline{x})\, d\underline{x}.$$

Usando el teorema de Bayes dentro de la integral:

$$\sum_{i=1}^{c} \int_{R} \chi_{R_i}(\underline{x}) \pi(y_i) f_{\underline{X}|Y=y_i}(\underline{x})\, d\underline{x} = \int_{R} \sum_{i=1}^{c} \chi_{R_i}(\underline{x}) \mathbb{P}(Y = y_i|\underline{X} = \underline{x}) f_{\underline{X}}(\underline{x})\, d\underline{x},$$

siendo $f_{\underline{X}}(\underline{x})$ la función de densidad del vector aleatorio $\underline{X}$ en el vector $\underline{x}$.

Ahora consideramos las regiones generadas a partir del clasificador de Bayes $\{R_1^*, \ldots, R_c^*\}$. Supongamos que un vector $\underline{x} \in R$ es asignado a la región R_j mediante un clasificador h y que ese mismo vector es clasificado en R_i^* a través del clasificador de Bayes. Se tiene entonces que:

$$\mathbb{P}(Y = y_i|\underline{X} = \underline{x}) \geq \mathbb{P}(Y = y_j|\underline{X} = \underline{x}) \quad \forall j = 1, \ldots, c.$$

Definimos las siguientes funciones dependientes del vector considerado:

$$g(\underline{x}) = \sum_{s=1}^{c} \chi_{R_s}(\underline{x}) \mathbb{P}(Y = y_s|\underline{X} = \underline{x}) f_{\underline{X}}(\underline{x});$$

$$g^*(\underline{x}) = \sum_{s=1}^{c} \chi_{R_s^*}(\underline{x}) \mathbb{P}(Y = y_s|\underline{X} = \underline{x}) f_{\underline{X}}(\underline{x}).$$

Ya que, $\underline{x}$ fue asignado a las regiones R_j y R_i^* mediante h y h^* respectivamente.

$$g^*(\underline{x}) = \sum_{s=1}^{c} \chi_{R_s^*}(\underline{x}) \mathbb{P}(Y = y_s | \underline{X} = \underline{x}) f_{\underline{X}}(\underline{x}) = \mathbb{P}(Y = y_i | \underline{X} = \underline{x}) f_{\underline{X}}(\underline{x}) =$$
$$= \sum_{s=1}^{c} \chi_{R_s^*}(\underline{x}) \mathbb{P}(Y = y_i | \underline{X} = \underline{x}) f_{\underline{X}}(\underline{x}) \geq \sum_{s=1}^{c} \chi_{R_s}(\underline{x}) \mathbb{P}(Y = y_j | \underline{X} = \underline{x}) f_{\underline{X}}(\underline{x}) = g(\underline{x}).$$

Por consiguiente, se obtiene la siguiente desigualdad:

$$\mathbb{P}(h^*(\underline{x}) \neq y_i) = 1 - \int_R \pi(y_i) f_{\underline{X}|Y=y_i}(\underline{x})\, d\underline{x} \leq 1 - \int_R \pi(y_j) f_{\underline{X}|Y=y_j}(\underline{x})\, d\underline{x} = \mathbb{P}(h(\underline{x}) \neq y_j).$$

Esta desigualdad se obtiene para cualquier vector $\underline{x} \in R$ pues lo hemos tomado de manera arbitraria. Concluimos pues:

$$\varepsilon(h^*) = \mathbb{P}(h^*(\underline{X}) \neq Y) \leq \mathbb{P}(h(\underline{X}) \neq Y) = \varepsilon(h),$$

sea cual sea el clasificador h considerado.

□

Con este teorema hemos probado que el clasificador que mejor predice las clases es el clasificador de Bayes. La importancia de esta función reside en que, aunque no es posible aplicarlo sin conocer las distribuciones exactas de las variables, el tener el error mínimo hace que muchos de los clasificadores que expondremos posteriormente estén basados en buscar estimadores de $f_{\underline{X}|Y=y}(\underline{x})$, $f_{\underline{X}}(\underline{x})$ y $\pi(y)$, con el fin de conseguir acercarse a la cota de error óptima.

Capítulo 2

Clasificadores lineales

Los clasificadores lineales son aquellos que buscan mediante una combinación lineal de las variables explicativas contenidas en el vector $\underline{X}$ tomar una decisión de clasificación. Consideraremos de aquí en adelante que los problemas de clasificación tienen $c \geq 2$ clases y los vectores $\underline{x}$ son de dimensión $p \geq 1$.

En estos casos, si un clasificador usa funciones discriminantes, éstas serán lineales. Los límites de decisión de estos clasificadores pueden ser: o bien un hiperplano, o varios trozos de hiperplanos definidos por ecuaciones lineales.

2.1. Regresión logística

Los modelos de regresión lineales están basados en tratar de representar la variable objetivo Y (que puede ser contínua) como una función aproximadamente lineal de las variables predictoras contenidas en el vector $\underline{X}$, es decir:

$$Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p + \varepsilon,$$

siendo ε el error que se produce al aproximar de manera lineal la variable objetivo mediante las variables predictoras.

La principal diferencia entre un problema de clasificación y un modelo de regresión lineal se encuentra en la variable objetivo, ya que en los problemas de clasificación, esta variable es de tipo categórica. Esta diferencia hace que sea necesario reformular la expresión anterior para adaptar el modelo de regresión lineal habitual y poder clasificar.

El modelo de regresión logística fue desarrollado por el estadístico David Cox en 1958. Vamos a considerar un problema binario ($\mathcal{Y} = \{0, 1\}$) y buscamos una relación entre $\mathbb{P}(Y = 1|\underline{X} = \underline{x})$ y $\underline{X}$. Estimaremos la probabilidad anterior sin la fórmula de Bayes, usando una función con valores comprendidos entre 0 y 1 para cualquier valor de $\underline{X}$.

Consideraremos pues la función logística:

$$\mathbb{P}(Y=1|\underline{X}=\underline{x})=\frac{e^{\beta_0+\beta_1X_1+\ldots+\beta_pX_p}}{1+e^{\beta_0+\beta_1X_1+\ldots+\beta_pX_p}},$$

que puede ser reescrita de la forma:

$$\log\left(\frac{\mathbb{P}(Y=1|\underline{X}=\underline{x})}{1-\mathbb{P}(Y=1|\underline{X}=\underline{x})}\right)=\beta_0+\beta_1X_1+\ldots+\beta_pX_p.$$

Se asignará al vector $\underline{x}=(x_1,\ldots,x_p)^t$ la clase 1 si $\mathbb{P}(Y=1|\underline{X}=\underline{x})\geq 0{,}5$ y 0 en caso contrario. Equivalentemente, si $\beta_0+\beta_1x_1+\ldots+\beta_px_p\geq 0$, situaremos $\underline{x}$ en la región perteneciente a la clase 1.

En regresión logística, se emplean los datos de entrenamiento para estimar los coeficientes β_i con $i=1,\ldots,p$. La manera más frecuente de realizar las estimación es mediante el método de máxima verosimilitud, aunque también se puede usar el método de mínimos cuadrados. Dada una muestra de tamaño n de $(\underline{X},Y)$ y siendo un problema binario, la función de verosimilitud es:

$$\mathcal{L}(\beta_0,\underline{\beta})=\prod_{k=1}^{n}\mathbb{P}(Y=1|\underline{X}=\underline{x}_k)^{y_k}(1-\mathbb{P}(Y=1|\underline{X}=\underline{x}_k))^{1-y_k},$$

con $\underline{\beta}=(\beta_1,\ldots,\beta_p)^t$.

Si consideramos la función de log-verosimilitud:

$$\begin{aligned}\log(\mathcal{L}(\beta_0,\underline{\beta}))&=\sum_{k=1}^{n}y_k\log(\mathbb{P}(Y=1|\underline{X}=\underline{x}_k))+\sum_{k=1}^{n}(1-y_k)\log(1-\mathbb{P}(Y=1|\underline{X}=\underline{x}_k))=\\&=\sum_{k=1}^{n}\log(1-\mathbb{P}(Y=1|\underline{X}=\underline{x}_k))+\sum_{k=1}^{n}y_k\log\left(\frac{\mathbb{P}(Y=1|\underline{X}=\underline{x}_k)}{1-\mathbb{P}(Y=1|\underline{X}=\underline{x}_k)}\right)=\\&=\sum_{k=1}^{n}-\log(1+e^{\beta_0+\underline{x}_k^t\underline{\beta}})\sum_{k=1}^{n}y_k(\beta_0+\underline{x}_k^t\underline{\beta}).\end{aligned}$$

El enfoque usual llegado a este punto es calcular las derivadas con respecto a cada uno de los coeficientes e igualarlas a cero para buscar los máximos pero si calculamos una de ellas:

$$\frac{\partial\log(\mathcal{L}(\beta_0,\underline{\beta}))}{\partial\beta_j}=-\sum_{k=1}^{n}\frac{e^{\beta_0+\underline{x}^t\underline{\beta}}}{1+e^{\beta_0+\underline{x}^t\underline{\beta}}}x_{kj}+\sum_{k=1}^{n}y_kx_{kj}=\sum_{k=1}^{n}(y_k-\mathbb{P}(Y=1|\underline{X}=\underline{x}_k))x_{kj},\quad\forall j=0,\ldots,p,$$

observamos que no es posible calcular una solución que no sea numérica pues son ecuaciones trascendentes. En el caso de $j=0$ basta con definir $x_{k,0}=1$ con $k=1,\ldots,n$ y resolver.

Aunque no es habitual emplear la regresión logística en problemas de clasificación múltiple, existe una extensión de este modelo llamado regresión logística multinomial (o multiclase) donde, si ahora tenemos $\mathcal{Y} = \{y_1, \ldots, y_c\}$ clases entonces la función logística para una clase y_i es:

$$\mathbb{P}(Y = y_i | \underline{X} = \underline{x}) = \frac{e^{\beta_0^i + \beta_1^i X_1 + \ldots + \beta_p^i X_p}}{\sum_{k=1}^{c} e^{\beta_0^k + \beta_1^k X_1 + \ldots + \beta_p^k X_p}}, \quad \forall i = 1, \ldots, c.$$

En vista de las probabilidades anteriores necesitaremos estimar $c(p+1)$ coeficientes empleando los datos de entrenamiento para poder ajustar el modelo. De nuevo la regla de clasificación viene dada por: clasificamos $\underline{x} = (x_1, \ldots, x_p)^t$ en una clase y_i si:

$$\mathbb{P}(Y = y_i | \underline{X} = \underline{x}) \geq \mathbb{P}(Y = y_j | \underline{X} = \underline{x}) \quad \forall j = 1, \ldots, c$$

$$\Updownarrow$$

$$e^{\hat{\beta}_0^i + \hat{\beta}_1^i x_1 + \ldots + \hat{\beta}_p^i x_p} \geq e^{\hat{\beta}_0^j + \hat{\beta}_1^j x_1 + \ldots + \hat{\beta}_p^j x_p} \quad \forall j = 1, \ldots, c.$$

Al igual que ocurría en el caso binario, esta estimación de los parámetros se suele realizar por el método de máxima verosimilitud, cuya solución es ahora aún más compleja debido al aumento de parámetros por lo que sólo se podrán considerar soluciones numéricas de la función de log-verosimilitud. Tomada una muestra de tamaño n si n_i es el número de vectores cuya clase es y_i ($\sum_{k=1}^{c} n_k = n$) con $i = 1, \ldots, c$, entonces la función a maximizar es la siguiente:

$$\log(\mathcal{L}(\underline{\beta}_0, \underline{\beta}^1, \ldots, \underline{\beta}^c)) = \sum_{k=1}^{n_1} n_1 \log(\mathbb{P}(Y = y_1 | \underline{X} = \underline{x_k})) + \ldots + \sum_{k=1}^{n_c} n_c \log(\mathbb{P}(Y = y_c | \underline{X} = \underline{x_k})),$$

donde $\underline{\beta}_0 = (\beta_0^1, \ldots, \beta_0^c)^t$ y $\underline{\beta}^i = (\beta_1^i, \ldots, \beta_p^i)$ con $i = 1, \ldots, c$.

Podemos realizar un contraste de hipótesis para los coeficientes definido como sigue:

$$\begin{cases} H_0 : \beta_i = 0 \\ H_1 : \beta_i \neq 0 \quad \forall i = 0, \ldots, p. \end{cases}$$

Sea $\hat{\beta}_i$ el estimador por máxima verosimilitud de β_i, este contraste de hipótesis usa el test de Wald con el siguiente estadístico:

$$z = \frac{\hat{\beta}_i^2}{Var(\hat{\beta}_i)}, \quad \forall i = 0, \ldots, p,$$

cuya distribución asintótica sigue una chi-cuadrado con un grado de libertad.

Vamos a ver un ejemplo de regresión logística en R. Tenemos una muestra de sangre de 270 pacientes para tratar de descubrir si tienen o no alguna enfermedad de corazón [1].

[1] Los datos de este ejemplo en : https://archive.ics.uci.edu/ml/datasets/Statlog+%28Heart%29

Tenemos un total de 13 variables explicativas que son:

- $X_1 \equiv$ Edad.
- $X_2 \equiv$ Sexo.
- $X_3 \equiv$ Tipo de dolor en el pecho (numeradas de $1-4$).
- $X_4 \equiv$ Presión sanguínea en reposo.
- $X_5 \equiv$ Colesterol sérico.
- $X_6 \equiv$ Azúcar en sangre en ayunas (binaria, toma el valor 1 si hay mas de 120 mg/dl).
- $X_7 \equiv$ Resultados electrocardiográficos en reposo (toma valores $0, 1, 2$).
- $X_8 \equiv$ Frecuencia cardíaca máxima alcanzada.
- $X_9 \equiv$ Angina inducida por el ejercicio.
- $X_{10} \equiv$ Depresión del segmento ST inducida por el ejercicio.
- $X_{11} \equiv$ La inclinación del segmento ST durante el ejercicio.
- $X_{12} \equiv$ Número de vasos principales coloreados en una fluoroscopía (de 1 a 3).
- $X_{13} \equiv$ Talasemia.
- $Y \equiv$ Existencia o no de una enfermedad de corazón.

Dividimos los datos en dos conjuntos, entrenamiento y prueba. Algunas de las variables son discretas, por tanto, necesitamos que las imágenes de estas sean tratadas como clases y no como números.

```
library(caret)
library(biotools)
library(MASS)
library(xtable)
corazon = read.table("corazon.txt")
summary(corazon)
colnames(corazon) = c(paste("X",1:13,sep=""),"Y")
v = c(2,3,6,7,9,12,13)
for (i in 1:length(v))
{
corazon[,v[i]] = as.factor(corazon[,v[i]])
}
corazon[,14] = as.factor(corazon[,14]-1)
set.seed(12345)
```

```
muestra = sample(1:nrow(corazon),180)
entrena = corazon[muestra,]
test = corazon[-muestra,]
```

Con la función *glm* construimos el modelo de regresión logística. Sin embargo es posible que existan coeficientes que no sean significativos puesto que R por defecto, efectúa un contraste de hipótesis para la significación de los coeficientes.

```
rl.corazon = glm(formula = Y ~ ., data = entrena,
family = binomial)
summary(rl.corazon)
Coefficients:
Estimate Std. Error z value Pr(>|z|)
(Intercept) -9.387475   4.380757  -2.143 0.032122 *
X1          -0.008485   0.036177  -0.235 0.814560
X21          1.562448   0.749293   2.085 0.037049 *
X32          3.514103   1.432093   2.454 0.014135 *
X33          2.806348   1.352592   2.075 0.038006 *
X34          5.038414   1.418301   3.552 0.000382 ***
X4           0.040867   0.017100   2.390 0.016858 *
X5           0.004198   0.005227   0.803 0.421912
X61         -1.024495   0.852180  -1.202 0.229284
X71          0.201352   3.274845   0.061 0.950973
X72          0.886030   0.576744   1.536 0.124474
X8          -0.032839   0.017436  -1.883 0.059651 .
X91         -0.001306   0.655673  -0.002 0.998411
X10          0.505857   0.360352   1.404 0.160383
X11          0.434659   0.554212   0.784 0.432874
X121         1.696654   0.722609   2.348 0.018877 *
X122         4.162333   1.243407   3.348 0.000815 ***
X123         2.781862   1.314026   2.117 0.034255 *
X136         0.357498   1.221272   0.293 0.769732
X137         2.023371   0.613821   3.296 0.000979 ***
```

Aquí vemos que muchos de ellos no son significativos. La función stepAIC de R nos ayudará a encontrar aquella combinación de variables que haga que no se pierda mucha información y ajuste lo mejor posible la variable objetivo.

```
coraz = glm(formula = Y ~ 1, data = entrena, family = binomial)
stepAIC(coraz, scope = list(lower = coraz, upper = rl.corazon),
        trace=0)
Call:  glm(formula = Y ~ X2 + X3 + X4 + X8 +
X10 + X12 + X13, family = binomial, data = entrena)
```

```
rl.corazon2 = glm(formula = Y ~ X2+X3+X4+X8+X10+X12+X13,
                  data = entrena, family = binomial)
```

```
summary(rl.corazon2)
```

```
Coefficients:
Estimate Std. Error z value Pr(>|z|)
(Intercept) -6.01286    3.12124  -1.926 0.054050 .
X21          1.20029    0.65613   1.829 0.067346 .
X32          3.04844    1.30846   2.330 0.019817 *
X33          2.11436    1.20629   1.753 0.079640 .
X34          4.64036    1.25923   3.685 0.000229 ***
X4           0.03135    0.01414   2.217 0.026605 *
X8          -0.03102    0.01513  -2.050 0.040373 *
X10          0.68260    0.30998   2.202 0.027661 *
X121         1.45447    0.65007   2.237 0.025260 *
X122         3.35370    1.03240   3.248 0.001160 **
X123         2.52048    1.30315   1.934 0.053096 .
X136        -0.14813    1.06309  -0.139 0.889180
X137         1.93023    0.58268   3.313 0.000924 ***
```

Las variables discretas citadas previamente son descompuestas en variables indicadoras (*dummies*), esto son, variables que toman el valor 1 en una de las categorías y 0 en el resto. Si una variable de este tipo tiene c categorías distintas, se tomará una clase como referencia (R por defecto selecciona la primera) y se construirán las $c-1$ variables indicadoras necesarias para el modelo. En el caso aquí considerado, un ejemplo sería la variable X_{13}. Esta variable tiene tres categorías $(3, 6, 7)$ y se ha descompuesto en 2 variables indicadoras ($X_{13,6}$ y $X_{13,7}$) en $X_{13,7}$ toma el valor 1 si $X_{13} = 7$ y 0 en caso contrario

Dado que todos los coeficientes son positivos podemos afirmar que a valores altos de las variables mayor es la posibilidad de padecer una enfermedad en el corazón.

```
entrenaE = predict(rl.corazon2, entrena,type="response")
entrenaE = ifelse(entrenaE > 0.5, 1,0)
print(xtable(confusionmatrix(entrenaE,entrena$Y),digits = 0))
```

	0	1
0	85	12
1	9	74

Tenemos entonces un error de entrenamiento de 0,1166667. Pasamos a evaluar el clasificador usando una función de la librería *caret* llamada *confusionMatrix*, tomando como valor positivo el padecer una enfermedad, es decir, $Y = 1$.

```
testE = predict(rl.corazon2,test,type="response")
testE = ifelse(testE > 0.5,1,0)
confusionMatrix(as.factor(testE),test$Y,positive = "1")
```

```
Confusion Matrix and Statistics

   0  1
0 46  8
1 10 26

Accuracy : 0.8
95% CI : (0.7025, 0.8769)
No Information Rate : 0.6222
P-Value [Acc > NIR] : 0.0002225

Kappa : 0.5794
Mcnemar's Test P-Value : 0.8136637

Sensitivity : 0.7647
Specificity : 0.8214
Pos Pred Value : 0.7222
Neg Pred Value : 0.8519
Prevalence : 0.3778
Detection Rate : 0.2889
Detection Prevalence : 0.4000
Balanced Accuracy : 0.7931

'Positive' Class : 1
```

A la vista de los resultados obtenidos, aunque el acierto llega al 80 %, el modelo encuentra más facilidades para clasificar los pacientes sanos, algo que puede apreciarse viendo la diferencia entre los valores predictores positivo y negativo.

El modelo de regresión logística es empleado sobre todo en problemas de clasificación binaria y pueden obtenerse buenos resultados si el tamaño de la muestra es grande. Gracias a los coeficientes, la regresión logística no se ve altamente afectada por la colinealidad. Si un grupo de variables están altamente correladas es probable que uno o más de un coeficiente de esas variables sea 0 o estén bien distribuidos. Esta técnica tiene también a su favor, al igual que todo modelo lineal, que es fácilmente interpretable y podemos sacar conclusiones a partir de los coeficientes.

Para casos en los que tengamos un problema de clasificación múltiple, existen clasificadores que veremos posteriormente que funcionan mejor que la regresión logística.

2.2. Análisis discriminante lineal de Fisher

El análisis discriminante lineal de Fisher (ADLF) se utiliza para encontrar una combinación lineal de las variables que reduzca la dimensión total del problema y permita de esta manera describir las diferencias entre grupos para proporcionar posteriormente una herramienta para clasificar nuevas observaciones. Fue propuesto por Ronald A. Fisher en 1936 para un problema de dos clases y generalizada a más de dos por C.R.Rao en 1948. Aquí expondremos el caso general para c clases.

Esta técnica de aprendizaje supervisada (tiene en cuenta los datos de la variable de respuesta) tiene como objetivo proyectar los datos en una dimensión más pequeña obteniendo más separación entre los datos de las diferentes clases.

Para conseguir esa separación el ADLF busca realizar una transformación lineal que maximice el cociente entre las "varianzas entre clases"(*between-class variance*) y las "varianzas comunes"(*within-class variance*).

Sea $\mathcal{Y} = \{y_1, \ldots, y_c\}$ el conjunto de clases diferentes. De cada clase tenemos una muestra de tamaño n_k $(k = 1, \ldots, c)$, y en total $\sum_{k=1}^{c} n_k = n$.

Denotamos $\underline{x}_{ik}$ como el í-esimo vector p-dimensional de la clase k,
$i = 1, \ldots, n_k,\ k = 1, \ldots, c$. Definimos el vector de medias muestrales de la clase k:

$$\overline{\underline{x}}_k = \frac{1}{n_k} \sum_{i=1}^{n_k} \underline{x}_{ik}, \quad \forall k = 1, \ldots, c.$$

Ahora definimos las siguientes matrices para cada clase:

$$\mathbf{S}_k = \sum_{i=1}^{n_k} (\underline{x}_{ik} - \overline{\underline{x}}_k)(\underline{x}_{ik} - \overline{\underline{x}}_k)^t, \quad \forall k = 1, \ldots, c.$$

Sea la matriz de varianzas comunes:

$$\mathbf{S_W} = \sum_{k=1}^{c} \mathbf{S}_k.$$

Por último se definen el vector de medias totales y la matriz de dispersión entre clases:

$$\overline{\underline{x}} = \frac{1}{c} \sum_{i=1}^{c} \overline{\underline{x}}_i; \quad \mathbf{S_B} = \sum_{k=1}^{c} n_k (\overline{\underline{x}}_k - \overline{\underline{x}})(\overline{\underline{x}}_k - \overline{\underline{x}})^t.$$

Las componentes discriminantes lineales son aquellos vectores de $\mathbb{R}^n$ que maximicen el siguiente cociente:

$$J(\underline{w}) = \frac{\underline{w}^t \mathbf{S_B} \underline{w}}{\underline{w}^t \mathbf{S_W} \underline{w}}.$$

Una propiedad importante de esta función objetivo J es que, si $\underline{w}$ maximiza la función entonces $J(\underline{w}) = J(\alpha\underline{w})$ por tanto, siempre podemos elegir un vector $\underline{w}$ de forma que el denominador $\underline{w}^t\mathbf{S_W}\underline{w} = 1$.

De esta forma podemos transformar el anterior problema de maximizar J en este problema de optimización:

$$\begin{aligned} &\min_{\underline{w}} -\underline{w}^t\mathbf{S_B}\underline{w} \\ &\text{s.a } \underline{w}^t\mathbf{S_W}\underline{w} = 1 \\ &\underline{w} \in \mathbb{R}^p, \end{aligned}$$

que podemos reescribirlo al siguiente problema de Lagrange:

$$F(\underline{w}, \lambda) = -\underline{w}^t\mathbf{S_B}\underline{w} + \lambda(\underline{w}^t\mathbf{S_W}\underline{w} - 1),$$

derivando con respecto a $\underline{w}$ e igualando a 0 se tiene:

$$\mathbf{S_B}\underline{w} = \lambda\mathbf{S_W}\underline{w} \implies \mathbf{S_W^{-1}S_B}\underline{w} = \lambda\underline{w},$$

de donde vemos que λ se corresponde con un autovalor de la matriz $\mathbf{S_W^{-1}S_B}$ y $\underline{w}$ con su autovector asociado.

Al igual que ocurría en el ACP, queda resolver cual es el autovalor óptimo para este problema. Para ello sustituyendo en la función objetivo:

$$\min_{\underline{w}} -\underline{w}^t\mathbf{S_B}\underline{w} = \min_{\underline{w}} -\underline{w}^t\lambda\mathbf{S_W}\underline{w} = -\lambda,$$

la solución del problema tiene a λ como el mayor autovalor y la primera discriminante lineal es su autovector asociado de la matriz $\mathbf{S_W^{-1}S_B}$.

Se puede realizar una observación, y es que la matriz $\mathbf{S_B}$ a lo sumo tiene rango $(c-1)$, pues es suma de c matrices de rango menor o igual que 1 por proceder del producto de un vector columna por un vector fila, pero además debido a la definición de $\overline{x}$ el rango total no podría ser c. Aquellos autovalores distintos de 0 serán los que se empleen para los discriminantes lineales.

Para la k-ésima discriminante lineal habría que resolver el siguiente problema de optimización:

$$\begin{aligned} &\min_{\underline{w}} -\underline{w}^t\mathbf{S_B}\underline{w} \\ &\text{s.a } \underline{w}^t\mathbf{S_W}\underline{w} = 1 \\ &\quad \underline{w}_i^t\mathbf{S_W}\underline{w} = 0 \quad 1 \leq i \leq k-1 \\ &\quad \underline{w} \in \mathbb{R}^p, \end{aligned}$$

cuyas soluciones son los autovectores de la matriz $\mathbf{S_W^{-1}S_B}$ ordenadas de mayor a menor autovalor asociado.

Entonces, si disponemos de una matriz de datos $\mathbf{X}$ de dimensión $n \times p$ (la variable objetivo no la consideramos) las puntuaciones de cada vector correspondiente a una realización muestral vienen dadas por transformación lineal de las discriminantes lineales, esto es:

$$\mathbf{U} = \mathbf{XW},$$

donde $\mathbf{W}$ es una matriz cuyas columnas son las discriminantes lineales ordenadas. Notemos que $\mathbf{W}$ no es única, pues como vimos previamente, si dos autovectores están asociados a un mismo autovalor no nulo, la elección de cualquiera de ellos no altera el resultado del problema de optimización.

Una vez obtenidas las puntuaciones procederemos a clasificar los puntos acorde a la siguiente regla discriminante. Asigno un vector $\underline{x}$ a la clase y_i si:

$$d(\mathbf{W}^t\underline{x}, \mathbf{W}^t\underline{\overline{x}}_i) = \min_{k=1,\dots,c} d(\mathbf{W}^t\underline{x}, \mathbf{W}^t\underline{\overline{x}}_k),$$

donde $d(\cdot,\cdot)$ denota la distancia euclídea entre los vectores considerados.

2.2.1. Analisis discriminante lineal

Vamos a ver un caso particular importante dentro del analisis discriminante lineal con un enfoque diferente. En este caso usaremos funciones discriminantes lineales y estimadores de los elementos que aparecen en el clasificador de Bayes y de ahí obtener una regla de discriminación lineal. Tenemos las siguientes hipótesis.

1. Las funciones $f_{\underline{X}|Y=y}(\underline{x})$ siguen una distribución normal multivariante $\mathcal{N}_p(\underline{\mu}_y, \Sigma_y)$, es decir:

$$f_{\underline{X}|Y=y}(\underline{x}) = \frac{1}{(2\pi)^{\frac{p}{2}} \mid \Sigma_y \mid^{\frac{1}{2}}} \exp\{-\frac{1}{2}(\underline{x} - \underline{\mu}_y)^t \Sigma_y^{-1} (\underline{x} - \underline{\mu}_y)\}, \quad \forall y \in \mathcal{Y}.$$

2. Todas las clases tienen la misma matriz de varianzas y covarianzas que denotaremos Σ.

Bajo estas condiciones, se asignará al vector $\underline{x}$ aquella clase $y^* \in \mathcal{Y}$ que cumpla:

$$g_{y^*}(\underline{x}) > g_y(\underline{x}) \iff \mathbb{P}(Y = y^* \mid \underline{X} = \underline{x}) > \mathbb{P}(Y = y \mid \underline{X} = \underline{x}), \quad \forall y \in \mathcal{Y}.$$

Aplicando el teorema de Bayes:

$$\frac{f_{\underline{X}|Y=y^*}(\underline{x})\pi(y^*)}{f_{\underline{X}}(\underline{x})} > \frac{f_{\underline{X}|Y=y}(\underline{x})\pi(y)}{f_{\underline{X}}(\underline{x})}, \quad \forall y \in \mathcal{Y},$$

cancelando constantes en ambos lados y usando las hipótesis de normalidad y de igualdad de matrices de varianzas y covarianzas tenemos:

$$\exp\{-\frac{1}{2}(\underline{x}-\underline{\mu}_{y^*})^t\Sigma^{-1}(\underline{x}-\underline{\mu}_{y^*})\}\pi(y^*) > \exp\{-\frac{1}{2}(\underline{x}-\underline{\mu}_{y})^t\Sigma^{-1}(\underline{x}-\underline{\mu}_{y})\}\pi(y), \quad \forall y \in \mathcal{Y},$$

tomando logaritmos en ambos lados y operando nos queda:

$$\underline{x}^t\Sigma\mu_{y^*} - \frac{1}{2}\mu_{y^*}^t\Sigma\mu_{y^*} + \log(\pi(y^*)) > \underline{x}^t\Sigma\mu_y - \frac{1}{2}\mu_y^t\Sigma\mu_y + \log(\pi(y)),$$

por lo tanto las funciones discriminantes para cada clase son las siguientes:

$$g_y(\underline{x}) = \underline{x}^t\Sigma\mu_y - \frac{1}{2}\mu_y^t\Sigma\mu_y + \log(\pi(y)), \quad \forall y \in \mathcal{Y}.$$

Se puede apreciar que g_y como función de $\underline{x}$ es una función lineal. Aquellos vectores donde el máximo de las funciones discriminantes se alcance en más de una clase serán clasificados aleatoriamente entre las mismas, perteneciendo de esta forma al límite de decisión.

En la práctica, si tenemos una matriz de datos $\mathbf{X}$, habrá que buscar estimadores de las medias y de las probabilidades a priori de cada clase, además de un estimador para la matriz de covarianzas. Para las medias poblacionales y la matriz de varianzas basta tomar las medias muestrales de cada clase y la matriz de covarianzas muestrales, ya que son estimadores por máxima verosimilitud por tratarse de distribuciones normales.

Un enfoque para encontrar un estimador de las probabilidades a priori de una clase $\pi(y)$, podría ser dividir el número de datos que tienen asignado dicha clase entre el número de datos totales.

La importancia de este caso particular se encuentra en que, al estar basado en el clasificador de Bayes, el error que comete se acerca al óptimo siendo un clasificador lineal. No obstante, las hipótesis para poder aplicar este clasificador son fuertes, por ello no es fácil encontrar datos que cumplan todas estas condiciones.

Veamos un ejemplo del análisis discriminante lineal de Fisher. Necesitaremos la librería MASS de R para el siguiente ejemplo.

Tenemos un conjunto de 178 observaciones sobre un análisis químico de tres tipos diferentes de vinos [2] que se encuentran en la misma región de Italia. Con este análisis se han determinado las cantidades de 13 componentes que se encuentran en los distintos tipos.

- $Y \equiv$ Tipos de vino.
- $X_1 \equiv$ Alcohol.
- $X_2 \equiv$ Ácido málico.
- $X_3 \equiv$ Ceniza.
- $X_4 \equiv$ Alcalinidad de ceniza.
- $X_5 \equiv$ Magnesio.
- $X_6 \equiv$ Cantidad total de fenoles.
- $X_7 \equiv$ Flavonoides.
- $X_8 \equiv$ Fenoles no flavonoides.
- $X_9 \equiv$ Proantocianidinas.
- $X_{10} \equiv$ Intensidad del color.
- $X_{11} \equiv$ Matiz del vino.
- $X_{12} \equiv$ OD280/OD315 de vinos diluidos.
- $X_{13} \equiv$ Prolina.

De manera aleatoria vamos a dividir las observaciones en dos partes, entrenamiento y prueba. En este caso podemos emplear el método Hold-out tomando aproximadamente dos tercios como datos de entrenamiento y el resto de prueba.

```
library(MASS)
library(xtable)
library(biotools)
vino = read.table("vino.txt",sep=",")
vino[,1] = as.factor(vino[,1])
colnames(vino) = c("Y",paste("X",1:13,sep =""))
set.seed(678)
muestra = sample(1:nrow(vino), 120)
entreno = vino[muestra,]
test = vino[-muestra,]
```

[2] Los datos empleados en el siguiente ejemplo se encuentran en: https://archive.ics.uci.edu/ml/datasets/Wine

Procedemos a construir el modelo usando los datos de entrenamiento usando la función *lda* de la libreria MASS.

```
lda.vino = lda(formula = Y ~X1+X2+X3+X4+X5+X6+X7+X8
+X9+X10+X11+X12+X13, data = entreno)
```

Vamos a observar los errores de entrenamiento con la siguiente tabla:

```
predE = predict(lda.vino,entreno)
entrenoE = confusionmatrix(predE$class,entreno$Y)
tipos = c("T1","T2","T3")
colnames(entrenoE)= tipos
rownames(entrenoE)= tipos
print(xtable(entrenoE))
```

	T1	T2	T3
T1	39	0	0
T2	0	52	0
T3	0	0	29

donde se aprecia que el error de entrenamiento es 0. Veamos ahora el error de prueba con la siguiente matriz de confusión.

```
predT = predict(lda.vino,test)
confusionMatrix(predT$class,test$Y)
Confusion Matrix and Statistics
   1  2  3
1 20  0  0
2  0 19  0
3  0  0 19
Overall Statistics
Accuracy : 1
95% CI : (0.9384, 1)
No Information Rate : 0.3448
P-Value [Acc > NIR] : < 2.2e-16
Kappa : 1
Mcnemar's Test P-Value : NA
Statistics by Class:
                     Class: 1 Class: 2 Class: 3
Sensitivity            1.0000   1.0000   1.0000
Specificity            1.0000   1.0000   1.0000
Pos Pred Value         1.0000   1.0000   1.0000
Neg Pred Value         1.0000   1.0000   1.0000
Prevalence             0.3448   0.3276   0.3276
Detection Rate         0.3448   0.3276   0.3276
```

```
Detection Prevalence    0.3448    0.3276    0.3276
Balanced Accuracy       1.0000    1.0000    1.0000
```

El modelo también clasifica de manera exacta los datos de prueba. Podemos calcular las puntuaciones de los vectores y los medias de cada clase.

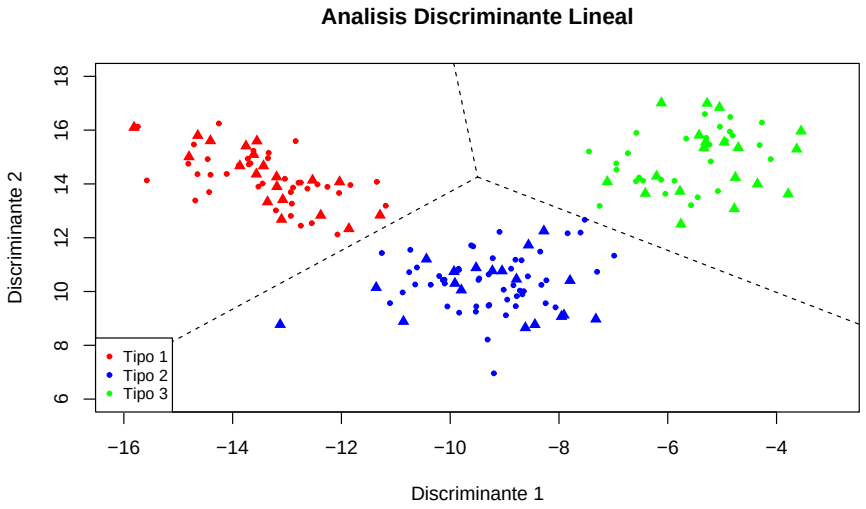
```
puntosE = as.matrix(entreno[,-1]) %*% as.matrix(lda.vino$scaling)
Ent = cbind(puntosE, entreno$Y)
puntosT = as.matrix(test[,-1]) %*% as.matrix(lda.vino$scaling)
prueba = cbind(puntosT, test$Y)
mediasE = as.matrix(lda.vino$means) %*% as.matrix(lda.vino$scaling)
```

Para poder visualizar las regiones de decisión, calculamos los puntos medios de las puntuaciones de las medias, la dirección de las rectas y el punto de corte de ellas.

```
pm1 = (mediasE[1,]+mediasE[2,])/2
pm2 = (mediasE[1,]+mediasE[3,])/2
pm3 = (mediasE[2,]+mediasE[3,])/2
u1 = mediasE[1,]-mediasE[2,]
v1 = c(u1[2], -u1[1])
u2 = mediasE[1,]-mediasE[3,]
v2 = c(u2[2], -u2[1])
u3 = mediasE[2,]-mediasE[3,]
v3 = c(u3[2], -u3[1])
corte=(-v1[2]*pm1[1]/v1[1]+pm1[2]-(-v2[2]*pm2[1]/v2[1]+
pm2[2]))/(v2[2]/v2[1]-v1[2]/v1[1])
```

Por último, representamos gráficamente los datos de entrenamiento y prueba separadas por sus límites de decisión:

```
x1 = seq(-20, corte, by=corte+20)
x2 = seq(corte, -5, by=-5-corte)
y1 = pm1[2]+(v1[2]*(x1-pm1[1])/v1[1])
y2 = pm2[2]+(v2[2]*(x2-pm2[1])/v2[1])
y3 = pm3[2]+(v3[2]*(x2-pm3[1])/v3[1])
colores = c("red","blue","green")
colores1 = colores[as.numeric(Ent[,3])]
colores2 = colores[as.numeric(prueba[,3])]
plot(Ent[,1:2], pch=20, cex = 0.8, col = colores1, xlab = "Discriminante 1",
     ylab = "Discriminante 2", main = "Analisis Discriminante Lineal",
     ylim = c(6,18), xlim = c(-16,-3))
lines(x1, y1, lty = 2)
lines(x1, y2, lty = 2)
lines(x2, y3, lty = 2)
points(prueba[,1:2], pch=17, col=colores2)
legend("bottomleft", legend=c("Tipo 1","Tipo 2","Tipo 3"),
col = colores, pch = 20, cex=0.8)
```



En triángulos están representados los datos de prueba. Como hemos observado previamente, todos los datos han sido correctamente clasificados por el modelo. En vista de la gráfica podemos afirmar que valores más altos de las dos discriminantes tiende a pertenecer al tipo 3, valores medios de la primera discriminante y bajos en la segunda discriminante dan lugar al tipo 2 y valores bajos de la primera discriminante y altos de la segunda pertenecen a la región del tipo 1.

El análisis discriminante lineal es una técnica muy útil si se pretende utilizar un clasificador lineal en un problema con más de una clase. La reducción de dimensión no siempre es una ventaja pues puede acarrear perdida de información en los datos. La dimensión final también depende del número de variables explicativas que se contemplen siendo el número final no mayor que $c-1$.

2.3. Redes neuronales artificiales

Aquí abordaremos otra forma de afrontar un problema de clasificación que es a partir de las redes neuronales. Primero vamos a introducir los elementos básicos de un sistema neuronal artificial. En una neurona la información parte de las dendritas que es el canal por el cual llega la información adquirida, pasa por el soma donde se evalúa la respuesta y sale a través del axón hacia otras neuronas.

En el caso de una neurona artificial el conjunto de datos de entrenamiento $\{(\underline{x}_k, y_k)\}_{k=1}^n$ constituyen la información disponible, además se le añaden unos pesos sinápticos $\underline{\beta}_i = (\beta_{i,1}, \ldots, \beta_{i,p})$ con $i = 1, \ldots, m$.

Para evaluar la información adquirida se utilizan reglas de propagación, esto es, funciones de la forma:

$$h_i(\underline{x}, \underline{\beta}_i, \beta_{0,i}), \quad \forall i = 1, \ldots, m,$$

siendo $\{\beta_{0,i}\}_{i=1}^m$ un conjunto de pesos denominados umbrales.

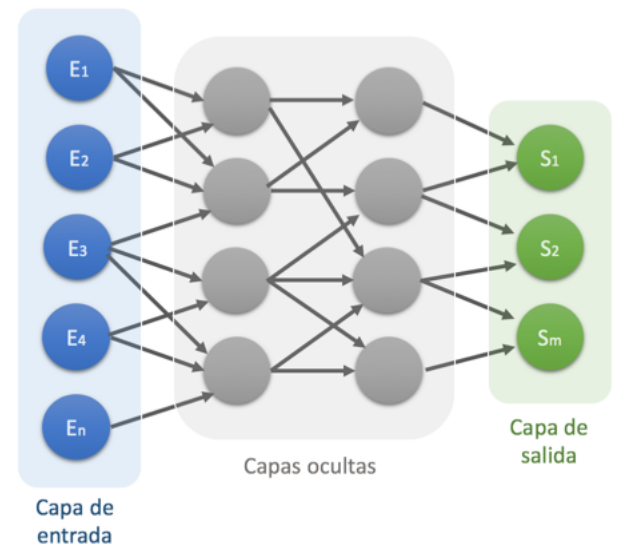
Por último se utilizan funciones de activación para proporcionar una respuesta (estado), estas funciones serán:

$$y_i = f_i(h_i(\underline{x}, \underline{\beta}_i, \beta_{0,i})), \quad \forall i = 1, \ldots, m.$$

Con estos elementos podemos definir una red neuronal como un grafo dirigido donde a cada neurona i se le asocia una variable de estado X_k. Para la conexión entre dos neuronas (i, k) se asocia el peso sináptico $\beta_{i,k}$. Para cada neurona i tenemos un umbral diferente $\beta_{i,0}$. En cada neurona se define una función de activación que depende de los pesos de sus conexiones, los umbrales y los estados de las neuronas conectadas con la que estemos considerando.

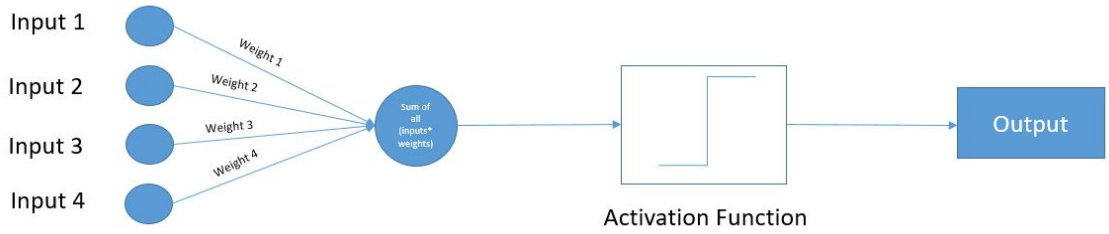
A las conexiones entre las neuronas se le denomina sinapsis. A las neuronas que no tengan sinapsis entrantes serán conocidas como neuronas de entrada. Las neuronas que no tengan sinapsis salientes se definen como neuronas de salida. Aquellas que no sean ni de entrada ni de salida se denominan neuronas ocultas.

Las neuronas de un sistema artificial se organizan en capas. El conjunto de neuronas de entrada constituyen la capa de entrada de la red neuronal artificial. Análogamente definimos la capa de salida y a aquellas capas que no sean ni de entrada ni de salida se denominan capas ocultas. El siguiente gráfico muestra un ejemplo de una red neuronal artificial.



2.3.1. Perceptrón

El algoritmo de clasificación más básico empleando redes neuronales es el Perceptrón. Este algoritmo data del año 1958 por Frank Rosenblatt en los Estados Unidos y aunque su propósito original no era un algoritmo, supuso una revolución en el campo de la inteligencia artificial. Sin embargo, posteriormente se probaron algunas de las carencias que posee el perceptrón que veremos posteriormente. Este modelo de red neuronal es el modelo más simple que existe puesto que solo consiste en una neurona. Aquí vemos un gráfico que representa este modelo, el cual detallaremos a continuación.



Definición 2.3.1. Supongamos que tenemos un conjunto de datos de entrenamiento $\{(\underline{x}_k, y_k)\}_{k=1}^n$ en un problema de clasificación binario ($\mathcal{Y} = \{-1, 1\}$) , diremos que las clases son linealmente separables si existe un hiperplano tal que:

$$\beta_0 + \beta_1 x_{k,1}, \ldots, \beta_p x_{k,p} > 0 \quad \text{si} \quad y_k = 1, \quad \forall k = 1, \ldots, n,$$

$$\beta_0 + \beta_1 x_{k,1}, \ldots, \beta_p x_{k,p} < 0 \quad \text{si} \quad y_k = -1, \quad \beta_0, \ldots, \beta_p \in \mathbb{R}.$$

El hiperplano que cumpla esta propiedad se denomina *hiperplano separador*.

Equivalentemente podemos reescribir un hiperplano separador como sigue:

$$y_k(\beta_0 + \beta_1 x_{k,1}, \ldots, \beta_p x_{k,p}) > 0 \quad \forall k = 1, \ldots, n.$$

Si las clases son linealmente separables podemos fácilmente construir un clasificador. En efecto, dado un vector $\underline{x} = (x_1, \ldots, x_p)^t$ definimos:

$$h(\underline{x}) = \begin{cases} 1 & \text{si} \quad \beta_0 + \beta_1 x_1, \ldots \beta_p x_p > 0, \\ -1 & \text{en caso contrario.} \end{cases} \tag{2.1}$$

El límite de decisión es en consecuencia, el hiperplano dado por los coeficientes β_0 y $\underline{\beta} = (\beta_1, \ldots, \beta_p)^t$ que separa las dos regiones de decisión. Intuitivamente, si el valor de $\beta_0 + \beta_1 x_1, \ldots, \beta_p x_p$ es próximo a 0 podemos pensar que $\underline{x}$ se encuentra a poca distancia del hiperplano. Esto sugiere la posibilidad de que el vector $\underline{x}$ considerado haya sido mal clasificado.

En el caso del Perceptrón vamos a considerar que las clases son linealmente separables y que el problema posee dos clases diferentes $\mathcal{Y} = \{-1, 1\}$, entonces buscamos un hiperplano que separe ambas clases, es decir, $\underline{\beta}$ y β_0 tales que para cada dato de entrenamiento se cumpla que:

$$y_k(\beta_0 + \beta_1 x_{k,1}, \dots \beta_p x_{k,p}), > 0 \quad \forall k = 1, \dots, n,$$

para ello tomamos como regla de propagación a la función:

$$h(\underline{x}_k, \underline{\beta}, \beta_0) = \sum_{i=1}^{p} \beta_i x_{k,i} + \beta_0, \quad \forall k = 1, \dots, n.$$

La función de activación en este caso es la función signo.

$$f(h(\underline{x}_k, \underline{\beta}, \beta_0)) = \begin{cases} 1 & \text{si} \quad \sum_{i=1}^{p} \beta_i x_{k,i} + \beta_0 \geq 0, \\ -1 & \text{si} \quad \sum_{i=1}^{p} \beta_i x_{k,i} + \beta_0 \leq 0. \end{cases}$$

Para encontrar los coeficientes $\underline{\beta}$ y β_0, abordaremos este problema tratando de minimizar la distancia entre los puntos que estén mal clasificados y los límites de decisión. La distancia de uno de los vectores pertenecientes a los datos de entrenamiento al hiperplano separador vendría dada por:

$$d(\underline{x}) = \frac{|\underline{\beta}^t \underline{x} + \beta_0|}{|\underline{\beta}|},$$

si imponemos que $|\underline{\beta}| = 1$ podemos tomar la siguiente expresión equivalente para cada dato de entrenamiento:

$$d(\underline{x}_k) = y_k(\underline{\beta}^t \underline{x}_k + \beta_0), \quad \forall k = 1, \dots, n.$$

En consecuencia, queremos minimizar la siguiente función:

$$\phi(\underline{\beta}, \beta_0) = -\sum_{\underline{x}_k \in M} y_k(\underline{\beta}^t \underline{x}_k + \beta_0),$$

donde M contiene aquellos datos de entrenamiento que han sido mal clasificados.

Para conseguir minimizar la función ϕ emplearemos una variante del método del gradiente o el método del descenso máximo. Consideramos entonces las derivadas siguientes:

$$\frac{\partial \phi}{\partial \underline{\beta}} = -\sum_{\underline{x}_k \in M} y_k \underline{x}_k,$$
$$\frac{\partial \phi}{\partial \beta_0} = -\sum_{\underline{x}_k \in M} y_k.$$

Vamos a iterar tomando los vectores mal clasificados de uno en uno y reevaluando por cada error que se comete. Una vez elegido un vector inicial $\underline{\beta}^1$ y β_0^1, usualmente se considera el vector nulo o alguno sencillo, comienza la iteración:

$$\underline{\beta}^{r+1} = \underline{\beta}^r - \rho y_k \underline{x}_k,$$

$$\beta_0^{r+1} = \beta_0^r - \rho y_k, \quad \forall (\underline{x}_k, y_k) \in M.$$

La constante ρ es un parámetro de escala que determina el tamaño del paso entre dos iteraciones (se podría considerar que ρ sea un valor diferente en cada iteración). Valores altos de ρ puede llevar a oscilaciones muy grandes y valores muy pequeños producen una convergencia lenta. La propiedad más importante es la que enunciamos a continuación.

Teorema 2.3.1. *Si las clases son linealmente separables, el algoritmo Perceptrón converge en un número finito de pasos.*

Demostración. La idea de la demostración se basa en considerar $\underline{\hat{\beta}}$ y $\hat{\beta}_0$ una solución del problema anterior que existe porque las clases son linealmente separables y ver que la distancia de una iteración cualquiera a la solución es siempre más grande que cualquier iteración posterior a la misma.

Como $\underline{\hat{\beta}}$ y $\hat{\beta}_0$ son soluciones entonces:

$$y_k(\underline{\hat{\beta}}^t \underline{x}_k + \hat{\beta}_0) > 0, \quad \forall k = 1, \ldots, n.$$

El proceso de aprendizaje comienza con un vector arbitrario y en este caso, sin perder generalidad, comenzaremos con el vector nulo. Supongamos ahora que, tras realizar r iteraciones, el vector $\underline{x_k}$ ha sido mal clasificado r^k veces con $k = 1, \ldots, n$. Por lo tanto, si partimos del vector nulo, el resultado tras las iteraciones es:

$$\underline{\beta} = \rho \sum_{k=1}^{n} r^k \underline{x}_k y_k,$$

multiplicamos a izquierda en ambos lados por la solución $\underline{\hat{\beta}}^t$ obteniendo así lo siguiente:

$$\underline{\hat{\beta}}^t \underline{\beta} = \rho \sum_{k=1}^{n} r^k \underline{\hat{\beta}}^t \underline{x}_k y_k \geq \rho r \min_k (\underline{\hat{\beta}}^t \underline{x}_k y_k),$$

donde en los pasos sucesivos tomaremos $\rho = 1$ ya que es un valor fijo. De la desigualdad anterior se sigue que $\underline{\hat{\beta}}^t \underline{\beta}$ esta acotado inferiormente por una función que crece linealmente con respecto al parámetro r. Además elevando al cuadrado ambos miembros en el término general de cada iteración se tiene:

$$\|\underline{\beta}^{r+1}\|^2 = \|\underline{\beta}^r\|^2 + \|\underline{x}_k\|^2 - 2(\underline{\beta}^r)^t \underline{x}_k y_k, \quad \forall (\underline{x}_k, y_k) \in M.$$

Como $\underline{x}_k$ está mal clasificado entonces, $(\underline{\beta}^r)^t \underline{x}_k y_k < 0$, por lo tanto podemos reescribir la expresión anterior.

$$\|\underline{\beta}^{r+1}\|^2 \leq \|\underline{\beta}^r\|^2 + \|\underline{x}_k\|^2.$$

Sea $\underline{x}$ el vector en M tal que $\|\underline{x}\|^2 \geq \|\underline{x}_k\|^2$ sea cual sea el vector considerado dentro de M. Tras r iteraciones se observa que la norma del vector de coeficientes $\underline{\beta}$ está acotada por:

$$\|\underline{\beta}\|^2 \leq r\|\underline{x}\|^2,$$

de donde se deduce que $\|\underline{\beta}\|$ no crece más rapido que una función del orden de $\sqrt{r}$. Pero anteriormente vimos que el producto escalar de $\underline{\hat{\beta}}^t \underline{\beta}$ está acotado inferiormente por una función lineal de r. Ya que $\underline{\hat{\beta}}$ está fijado previamente vemos que si r es suficientemente grande estos dos resultados son incompatibles. Luego r no puede crecer indefinidamente y en consecuencia el algoritmo debe converger en un número finito de pasos.

□

Notar que la hipótesis de que las clases han de ser linealmente separables ha sido fundamental en la demostración de la prueba anterior, de no ser así el algoritmo no convergerá y sólo llegaríamos a oscilaciones entre diferentes hiperplanos. Por otra parte solo podemos afirmar la existencia de soluciones pero ésta no será única, dependerá del orden en el que consideremos los vectores mal clasificados y la constante ρ (vease *Perceptrons*(1969) de Minsky y Papert).

Acabamos de exponer el modelo de red neuronal más simple de todos, en el capítulo posterior se detallará una generalización del Perceptrón empleando más capas (ocultas) para obtener mejores resultados para clasificar además de la capacidad de inclusión de un problema con más clases.

2.4. Máquinas de vectores soporte lineales

El algoritmo original de las máquinas de vectores soporte fue inventado por Vladimir Vapnik y Alexei Chervonenkis en 1963 derivado de la teoría del aprendizaje estadístico. Posteriormente en 1992 se expandió este concepto para modelos no lineales. Hoy en día es utilizado debido a los buenos resultados que genera en los problemas de clasificación.

Nos situamos en las condiciones del algoritmo Perceptrón, es decir, queremos encontrar un hiperplano tal que:

$$\beta_0 + \beta_1 x_{k,1}, \ldots \beta_p x_{k,p} > 0 \quad \text{si} \quad y_k = 1, \quad \forall k = 1, \ldots, n,$$

$$\beta_0 + \beta_1 x_{k,1}, \ldots \beta_p x_{k,p} < 0 \quad \text{si} \quad y_k = -1, \quad \beta_0, \ldots, \beta_p \in \mathbb{R}.$$

Sin embargo, ya que gracias al algoritmo Perceptrón sabemos que existe si las clases son linealmente separables buscamos un hiperplano separador óptimo (también hiperplano de margen maximal), que será aquel hiperplano que se encuentra a mayor distancia de todos los datos de entrenamiento. El valor de la distancia del dato de entrenamiento más cercano al hiperplano separador es lo que se conoce como *margen*. Por tanto, el hiperplano cuyo margen sea más grande será el escogido como separador óptimo. Aunque este clasificador suele dar buenos resultados, es posible encontrar un problema de sobreajuste si el número de variables es grande. Aquellos vectores cuya distancia al hiperplano sea igual al margen se le conocen como *vectores soporte*.

Sea V_s el conjunto de vectores soporte, a los coeficientes del hiperplano separador le imponemos la siguiente condición:

$$\underline{\beta}^t \underline{x} + \beta_0 = \begin{cases} 1 & \text{si su clase asociada es 1,} \\ -1 & \text{en caso contrario} \quad \forall \underline{x} \in V_s. \end{cases}$$

A los hiperplanos $H_1 : \underline{\beta}^t \underline{x} + \beta_0 = 1$ y $H_2 : \underline{\beta}^t \underline{x} + \beta_0 = -1$ se denominan les hiperplanos canónicos y tenemos entonces:

$$\begin{aligned} \underline{\beta}^t \underline{x}_k + \beta_0 &\geq 1 \qquad \text{si} \quad y_k = 1, \\ \underline{\beta}^t \underline{x}_k + \beta_0 &\leq -1 \qquad \text{si} \quad y_k = -1. \end{aligned}$$

La distancia entre cualquiera de los dos hiperplanos canónicos y el hiperplano separador es exactamente $\dfrac{1}{|\underline{\beta}|}$ que coincide con el margen anteriormente definido.

En consecuencia, maximizar el margen es equivalente a minimizar $|\underline{\beta}|$. El objetivo se resume en resolver el siguiente problema de optimización:

$$\begin{aligned} &\text{mín} \ \frac{|\underline{\beta}|^2}{2} \\ &\text{s.a.} \\ &\qquad y_k(\underline{\beta}^t \underline{x}_k + \beta_0) \geq 1 \quad \forall k = 1, \ldots, n \\ &\qquad \underline{\beta} \in \mathbb{R}^p, \quad \beta_0 \in \mathbb{R}, \end{aligned}$$

que es posible resolverlo mediante el método de los multiplicadores de Lagrange usando la siguiente función:

$$F(\underline{\beta}, \beta_0, \underline{\alpha}) = \frac{1}{2} \underline{\beta}^t \underline{\beta} - \sum_{k=1}^{n} \alpha_k (y_k (\underline{\beta}^t \underline{x}_k + \beta_0) - 1),$$

donde $\underline{\alpha} = (\alpha_1, \ldots, \alpha_n)$ son los multiplicadores de Lagrange. Debido a que las restricciones son de la forma $c_i \geq 0$ se tiene que los multiplicadores de Lagrange son todos mayores o iguales que 0. Para minimizar derivamos con respecto a $\underline{\beta}$ y β_0 e igualamos a 0 obteniendo lo siguiente:

$$\frac{\partial F}{\partial \underline{\beta}} = \underline{\beta} - \sum_{k=1}^{n} \alpha_k y_k \underline{x}_k = 0 \Longleftrightarrow \underline{\beta} = \sum_{k=1}^{n} \alpha_k y_k \underline{x}_k,$$
$$\frac{\partial F}{\partial \beta_0} = -\sum_{k=1}^{n} \alpha_k y_k = 0,$$

sustituyendo estas condiciones en F obtenemos la forma dual del método de los multiplicadores de Lagrange:

$$F_d(\underline{\alpha}) = \sum_{k=1}^{n} \alpha_k - \frac{1}{2} \sum_{k=1}^{n} \sum_{j=1}^{n} \alpha_k \alpha_j y_k y_j \underline{x}_k^t \underline{x}_j, \tag{2.2}$$

la cual hay que maximizar con respecto a $\underline{\alpha}$ sujeto a las siguientes restricciones:

$$\sum_{k=1}^{n}, \alpha_k y_k = 0, \quad \alpha_k \geq 0 \quad \forall k = 1, \ldots, n,$$

que es un problema de programación cuadrática cuya resolución numérica es más sencilla que la del problema primal. Este problema dual tendrá importancia de cara a estudiar máquinas de vectores soporte no lineales.

Volviendo al problema primal, las condiciones de Karush-Kuhn-Tucker nos proporcionan las condiciones necesarias y suficientes que debe satisfacer un mínimo en la función objetivo sujeto a restricciones de desigualdades e igualdades. En el problema primal tenemos:

$$\frac{\partial F}{\partial \underline{\beta}} = \underline{\beta} - \sum_{k=1}^{n} \alpha_k y_k \underline{x}_k = 0 \Longleftrightarrow \underline{\beta} = \sum_{k=1}^{n} \alpha_k y_k \underline{x}_k,$$
$$\frac{\partial F}{\partial \beta_0} = -\sum_{k=1}^{n} \alpha_k y_k = 0,$$
$$y_k(\underline{\beta}^t \underline{x}_k + \beta_0) - 1 \geq 0, \qquad \forall k = 1, \ldots, n,$$
$$\alpha_k \geq 0, \qquad \forall k = 1, \ldots, n,$$
$$\alpha_k(y_k(\underline{\beta}^t \underline{x}_k + \beta_0) - 1) = 0, \quad \forall k = 1, \ldots, n,$$

En particular de la última condición (conocida como condición complementaria de Karush-Kuhn-Tucker) deducimos que $\alpha_k = 0$ si $\underline{x}_k \notin V_s$ pues $y_k(\underline{\beta}^t \underline{x}_k + \beta_0) - 1 \neq 0$ ya que $\underline{x}_k$ no es un vector soporte.

Una vez obtenidos los valores de la solución $\hat{\underline{\alpha}}$ resolviendo el problema dual, de manera inmediata se puede calcular los coeficientes del hiperplano siendo estos:

$$\hat{\underline{\beta}} = \sum_{\underline{x}_k \in V_s} \hat{\alpha}_k y_k \underline{x}_k.$$

Por último calculamos $\hat{\beta}_0$ usando un vector soporte $\underline{x}_k$ cualquiera una vez obtenida la solución $\hat{\underline{\beta}}$:

$$\hat{\alpha}_k(y_k(\hat{\underline{\beta}}^t \underline{x}_k + \hat{\beta}_0) - 1) = 0.$$

Para un vector $\underline{x}$, la regla de clasificación consistirá en observar el signo de $\hat{\underline{\beta}}^t \underline{x} + \hat{\beta}_0$ y proceder de la forma descrita por el clasificador h en (2.1).

En general, no existe el hiperplano que separa las clases completamente, sin embargo, podemos razonar de manera parecida al caso en el que las categorías sean linealmente separables con el fin de poder trabajar con clases que no lo sean. En este caso se asume que el hiperplano cometerá errores de clasificación.

Se introducirán unas variables de holgura nuevas $\xi_1, \ldots, \xi_n$ para relajar las restricciones relativas a los hiperplanos canónicos que usábamos en el caso anterior. Se tiene entonces:

$$\begin{array}{lll} \underline{\beta}^t \underline{x}_k + \beta_0 \geq 1 - \xi_k & \text{si} & y_k = 1, \\ \underline{\beta}^t \underline{x}_k + \beta_0 \leq -1 + \xi_k & \text{si} & y_k = -1, \\ \xi_k \geq 0, & & \forall k = 1, \ldots, n. \end{array}$$

Si $\xi_k > 1$ para algún k, será un indicador de que el dato $\underline{x}_k$ ha sido mal clasificado a partir del hiperplano. Además introduciremos un parámetro adicional $C > 0$ a la nueva función objetivo. Este parámetro se encargará de realizar una compensación entre maximizar el margen y minimizar el error de entrenamiento. Pequeños valores de C tienden a enfatizar el margen ignorando datos atípicos (outliers) y valores grandes de C puede acarrear un sobreajuste en los datos de entrenamiento.

Si las clases no son linealmente separables hay que resolver el problema de optimización siguiente:

$$\begin{array}{ll} \text{mín } \dfrac{|\underline{\beta}|^2}{2} + \displaystyle\sum_{k=1}^{n} C\xi_k & \\ \text{s.a.} & \\ \quad y_k(\underline{\beta}^t \underline{x}_k + \beta_0) \geq 1 - \xi_k & \forall k = 1, \ldots, n \\ \quad \xi_k \geq 0 & \forall k = 1, \ldots, n \\ \quad \underline{\beta} \in \mathbb{R}^p, \quad \beta_0 \in \mathbb{R}. & \end{array}$$

De nuevo, usando los multiplicadores de Lagrange hay que minimizar para $\beta_0, \underline{\beta}$ y $\underline{\xi} = (\xi_1, \ldots, \xi_n)^t$ la siguiente función:

$$F(\beta_0, \underline{\beta}, \underline{\xi}, C, \underline{\alpha}, \underline{r}) = \frac{1}{2}\underline{\beta}^t \underline{\beta} + C\sum_{k=1}^{n} \xi_k - \sum_{k=1}^{n} \alpha_k(y_k(\underline{\beta}^t \underline{x}_k + \beta_0) - 1 + \xi_k) - \sum_{k=1}^{n} r_k \xi_k,$$

siendo $\underline{\alpha} = (\alpha_1, \ldots, \alpha_n)^t$ y $\underline{r} = (r_1, \ldots, r_n)^t$ los multiplicadores de Lagrange y además $\alpha_k, r_k \geq 0, \ \forall k = 1, \ldots, n$.

Derivando con respecto a β_0 y $\underline{\beta}$:

$$\frac{\partial F}{\partial \underline{\beta}} = \underline{\beta} - \sum_{k=1}^{n} \alpha_k y_k \underline{x}_k = 0 \Longleftrightarrow \underline{\beta} = \sum_{k=1}^{n} \alpha_k y_k \underline{x}_k,$$
$$\frac{\partial F}{\partial \beta_0} = -\sum_{k=1}^{n} \alpha_k y_k = 0,$$

que coincide con el caso anterior. Si derivamos con respecto a una de las variables de holgura ξ_k conseguimos una relacion entre los multiplicadores y C

$$\frac{\partial F}{\partial \xi_k} = C - \alpha_k - r_k = 0, \quad \forall k = 1, \ldots, n.$$

Sustituyendo las condiciones anteriores en la función de Lagrange obtenemos un problema dual exactamente igual al del caso de las clases linealmente separables visto en (2.2). Buscamos de nuevo un máximo para $\underline{\alpha}$ en (2.2) con las siguientes restricciones:

$$\sum_{k=1}^{n} \alpha_k y_k = 0, \quad 0 \leq \alpha_k \leq C, \quad \forall k = 1, \ldots, n.$$

Las condiciones complementarias de Karush-Kuhn-Tucker en este caso son:

$$\alpha_k(y_k(\underline{\beta}^t \underline{x} + \beta_0) - 1 + \xi_k) = 0,$$
$$r_k \xi_k = (C - \alpha_k)\xi_k = 0, \quad \forall k = 1, \ldots, n.$$

Aquellos vectores $\underline{x}_k$ tales que $\alpha_k > 0$ son los denominados vectores soporte. Si para algún k, $0 < \alpha_k < C$, entonces ξ_k ha de ser 0, esto significa que estos vectores soporte se encuentran en los hiperplanos canónicos a una distancia $\frac{1}{|\underline{\beta}|}$ del hiperplano separador. Una variable de holgura ξ_k sólo puede ser distinta de 0 si $C = \alpha_k$. En este caso diremos que $\underline{x}_k$ está mal clasificado si $\xi_k > 1$. Si $\xi_k < 1$ entonces el vector está bien clasificado pero se encuentra entre un hiperplano canónico y el hiperplano separador.

Sea $\hat{\underline{\alpha}}$ la solución del problema dual (2.2) los coeficientes de $\underline{\beta}$ están determinados como en el problema anterior, es decir:

$$\hat{\underline{\beta}} = \sum_{k=1}^{n} \hat{\alpha}_k y_k \underline{x}_k,$$

tomando un vector soporte $\underline{x}_k$ cuya variable de holgura $\xi_k = 0$ podemos determinar fácilmente el coeficiente β_0:

$$\hat{\alpha}_k(y_k(\hat{\underline{\beta}}^t \underline{x}_k + \hat{\beta}_0) - 1) = 0.$$

Las máquinas de vectores soportes también pueden ser aplicadas en un problema de clasificación múltiple (en este caso $\mathcal{Y} = \{y_1, \ldots, y_c\}$) . Existen tres formas de abordarlo:

- *Uno contra todos.* Consiste en tomar una clase y considerar el resto de las clases como una sola. Como ya estaríamos en las condiciones anteriores, buscamos un hiperplano que separe la clase fijada de las demás. Habría que construir un total de c hiperplanos, uno para cada clase.
- *Uno contra uno.* El enfoque sería escoger dos clases, calcular el hiperplano separador y repetir el proceso hasta realizarlo con todas las parejas de clases posibles. En este caso el número de hiperplanos a calcular viene dado por el número combinatorio $\binom{c}{2}$
- *Funciones discriminantes.* Dado un vector $\underline{x}$ construir c funciones lineales discriminantes de la forma:

$$g_{y_i}(\underline{x}) = \underline{\beta}_i^t \underline{x} + \beta_{0,i}, \quad \underline{\beta}_i \in \mathbb{R}^p, \beta_{0,i} \in \mathbb{R} \quad \forall i = 1, \ldots, c.$$

Nos centramos en el último, buscamos una solución $\{(\underline{\beta}_i, \beta_{0,i}),\ i = 1, \ldots, c\}$ tal que la regla de decisión definida por: asignar al vector $\underline{x}$ la clase y_i si se verifica:

$$g_{y_i}(\underline{x}) = \underset{y_j \in \mathcal{Y}}{\text{máx}}\, g_{y_j}(\underline{x}),$$

consiga separar los datos de entrenamiento sin error. Ésto es equivalente a encontrar soluciones para $\{(\underline{\beta}_i, \beta_{0,i}),\ i = 1, \ldots, c\}$ tales que para todo $i = 1, \ldots, c$ se tiene:

$$\underline{\beta}_i^t \underline{x} + \beta_{0,i} - (\underline{\beta}_j^t \underline{x} + \beta_{0,j}) \geq 1,$$

para cualquier $\underline{x}$ cuya clase asignada sea y_i y para cualquier $j \neq i$. Ésto significa que todo par de clases son linealmente separables. En estas condiciones, para encontrar los coeficientes de las funciones discriminantes hay que minimizar la siguiente expresión:

$$\begin{aligned}
&\text{mín} \sum_{i=1}^{c} \underline{\beta}_i^t \underline{\beta}_i \\
&\text{s.a.} \\
&\qquad \underline{\beta}_i^t \underline{x}_k + \beta_{0,i} - (\underline{\beta}_j^t \underline{x}_k + \beta_{0,j}) \geq 1 \quad \forall k = 1, \ldots, n,\ \forall j = 1, \ldots, c,\ j \neq i \\
&\qquad \underline{\beta}_i \in \mathbb{R}^p, \quad \beta_{0,i} \in \mathbb{R} \quad \forall i = 1, \ldots, c.
\end{aligned}$$

En estas restricciones, el vector $\underline{x}_k$ tiene asociada la clase y_i, para todo $k = 1, \ldots, n$ y para todo $i = 1, \ldots, c$. El número total de restricciones del problema es $n(c-1)$.

Si por el contrario las clases no son linealmente separables, entonces introducimos de nuevo las variables de holgura y el parámetro C, de forma que el problema de optimización queda determinado:

$$\text{mín} \sum_{i=1}^{c} \underline{\beta}_i^t \underline{\beta}_i + C \sum_{i=1}^{n} \xi_i$$

s.a.

$$\underline{\beta}_i^t \underline{x}_k + \beta_{0,i} - (\underline{\beta}_j^t \underline{x}_k + \beta_{0,j}) \geq 1 - \xi_i \quad \forall k = 1, \ldots, n, \ \forall j = 1, \ldots, c, \ j \neq i$$

$$\underline{\beta}_i \in \mathbb{R}^p, \quad \beta_{0,i} \in \mathbb{R}, \quad \forall i = 1, \ldots, c.$$

Para este caso, $\underline{x}_k$ debe cumplir lo mismo que en el problema anterior donde las clases son linealmente separables. El número de restricciones es también $n(c-1)$. La resolución de ambos problemas de optimización sigue los mismos pasos que en el caso binario.

En el siguiente ejemplo, tenemos una muestra de 42000 dígitos[3] representados en píxeles. Cada píxel representa una variable, formando un total de 784 variables explicativas más la variable objetivo de la cual sabemos que posee 10 dígitos diferentes (del 0 al 9). Se pretende realizar un reconocimiento de patrones con la información proporcionada por los píxeles, para identificar el número representado usando máquinas de vectores soporte.

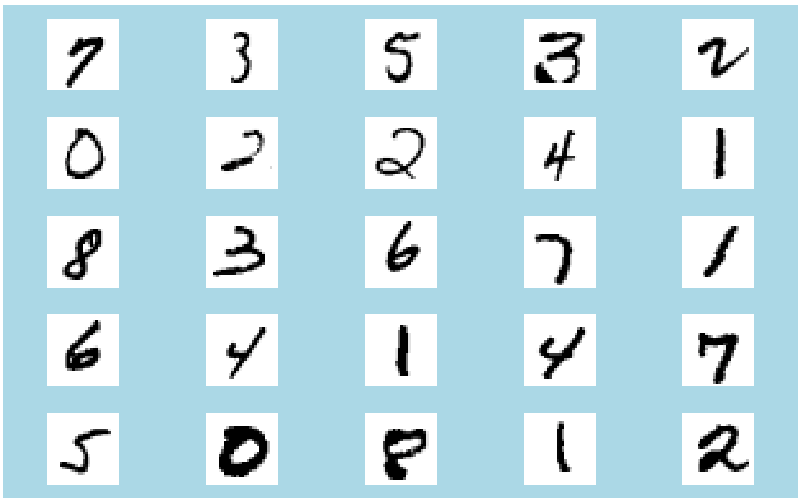
Para este modelo en particular emplearemos las librerias de *R*, *pixmap, e*1071 y *kernlab* además de algunas ya empleadas en casos anteriores.

```
library(pixmap)
library(caret)
library(kernlab)
library(xtable)
library(biotools)

num = read.csv("numeros.csv",sep =",", header = TRUE)
num[,1] = as.factor(num[,1])
set.seed(12345)
muestraE = sample(1:nrow(num),28000)
num1 = num[muestraE,]
num2 = num[-muestraE,]
```

Para ver una representación de alguno de los dígitos usaremos la función *pixmapGrey*. Esta función usa una matriz (en este caso de 28×28) donde cada entrada representa la intensidad de los grises de cada píxel con un número entre 0 y 1, siendo 0 el color negro y 1 el color blanco. Ya que las variables se miden de 0 a 256 hacemos un cambio de escala para reducirla al intervalo deseado y obtenemos los dígitos.

[3]Los datos correspondientes se encuentran en https://www.kaggle.com/c/digit-recognizer/data

```
num1[,2:785] = (-1)*((num1[,2:785]/256)-1)
num2[,2:785] = (-1)*((num2[,2:785]/256)-1)
parantig<-par(mfrow=c(5,5),
mar=c(0,0,0,0)+1,bg="lightblue")
for (i in 1:25)
{matriz= num1[i,2:785]
x=pixmapGrey(matriz,28,28,cellres=1)
x@grey<-t(x@grey)
plot(x)
}
```



Ya que estamos trabajando con 784 variables explicativas, es conveniente reducir la dimensión usando el análisis de componentes principales para poder realizar el modelo de clasificación.

```
componentes = princomp(num1[,-1])
s = summary(componentes)
v = (s$sdev)^2
```

```
> sum(v[1:90])/sum(v)
[1] 0.9043871
```

Como podemos observar, emplear un total de 90 variables conlleva más de un 90 % de varianza total explicada. Calculamos las puntuaciones de cada vector en las componentes principales y añadimos su clase correspondiente tanto en el conjunto de entrenamiento como en el de prueba. Posteriormente creamos el modelo con la función *ksvm*.

```
punt.num = as.data.frame(cbind(num1$label,componentes$scores[,1:90]))
punt.num[,1] = as.factor(punt.num[,1]-1)
test = predict(componentes,num2[,2:785])
punt.test = as.data.frame(cbind(num2$label,test[,1:90]))
```

```
punt.test[,1]= as.factor(punt.test[,1]-1)
svm.num1 = ksvm(punt.num[,1] ~ . ,data = punt.num,
                kernel = "vanilladot")
```

La variable de entrada *kernel* denota el núcleo que estamos utilizando para realizar este modelo, en este caso, el modelo lineal se construye introduciendo a esta variable de entrada la indicación *vanilladot*. En capítulos posteriores veremos ejemplos donde es posible usar otros tipos de núcleos (no lineales) para realizar la clasificación. Pasamos a observar los errores de entrenamiento.

```
predE = predict(svm.num1,punt.num, type= "response")
print(xtable(confusionmatrix(predE,num1$label),digits = 0))
```

	0	1	2	3	4	5	6	7	8	9
0	2742	0	0	0	0	0	0	0	0	0
1	0	3080	0	0	0	0	0	0	0	0
2	0	0	2786	0	0	0	0	0	0	0
3	0	0	0	2886	0	0	0	0	0	0
4	0	0	0	0	2706	0	0	0	0	0
5	0	0	0	0	0	2571	0	0	0	0
6	0	0	0	0	0	0	2734	0	0	0
7	0	0	0	0	0	0	0	2963	0	0
8	0	0	0	0	0	0	0	0	2732	0
9	0	0	0	0	0	0	0	0	0	2800

Se comprueba que las variables son linealmente separables por hiperplanos ya que su error de entrenamiento es 0. Veamos los errores de prueba y la evaluación del clasificador.

```
predT=predict(svm.num1,punt.test,type="response")
confusionMatrix(predT,punt.test[,1])
```

```
Confusion Matrix and Statistics
```

	0	1	2	3	4	5	6	7	8	9
0	1390	0	0	0	0	0	0	0	0	0
1	0	1604	0	0	0	0	0	0	0	0
2	0	0	1391	0	0	0	0	0	0	0
3	0	0	0	1465	0	0	0	0	0	0
4	0	0	0	0	1366	0	0	0	0	0
5	0	0	0	0	0	1224	0	0	0	0
6	0	0	0	0	0	0	1403	0	0	0
7	0	0	0	0	0	0	0	1438	0	0
8	0	0	0	0	0	0	0	0	1331	0
9	0	0	0	0	0	0	0	0	0	1388

```
Overall Statistics

Accuracy : 1
95% CI : (0.9997, 1)
No Information Rate : 0.1146
P-Value [Acc > NIR] : < 2.2e-16
```

Donde comprobamos que también clasifica correctamente el resto de los datos de prueba. El hecho de haber empleado el análisis de componentes principales en este caso ha tenido una ventaja significativa, ya que reducir la dimensión ha ayudado ampliamente a disminuir el coste computacional del modelo debido a que existen menos parámetros a estimar. Además, la perdida de información que existe al usar 90 componentes principales no ha supuesto un problema de cara a clasificar los datos.

Capítulo 3

Clasificadores no lineales

En ocasiones el problema de clasificación se hace más complicado si se restringe a la búsqueda de límites de decisión lineales. Por ello, recurrimos a emplear otro tipo de clasificadores donde los límites de decisión sean no lineales. Una de las ventajas de los clasificadores lineales es que son fácilmente interpretables. Los siguientes clasificadores que se describirán a continuación tratarán de mejorar los resultados a cambio de perder facilidad de interpretación. Al igual que en el capítulo anterior supondremos $c \geq 2$ clases y vectores de dimensión p.

3.1. Clasificador Naive Bayes

El siguiente clasificador que vamos a detallar es conocido como Naive Bayes. El enfoque que emplearemos será similar al del análisis discriminante lineal, es decir, partiendo del teorema de Bayes y usando funciones discriminantes.

Este clasificador requiere que las distribuciones condicionadas a cada clase sean independientes entre ellas. Esta hipótesis es difícil de cumplir y por ello se denomina a este clasificador *naive* (ingenuo).

No obstante, este algoritmo de clasificación se utiliza hoy en día consiguiendo buenos resultados en varios ámbitos, por ejemplo, en la detección de correo *spam* o en la clasificación de textos en función de su temática entre otros usos.

Con estas condiciones veamos la construcción del algoritmo. Al igual que en el ADL, el clasificador naive Bayes busca la clase $y^* \in \mathcal{Y}$ tal que fijado un vector $\underline{x} = (x_1, \ldots, x_n)^t$ cumpla que:

$$g_{y^*}(\underline{x}) \geq g_y(\underline{x}) \iff \mathbb{P}(Y = y^* | \underline{X} = \underline{x}) \geq \mathbb{P}(Y = y | \underline{X} = \underline{x}), \quad \forall y \in \mathcal{Y}.$$

Aplicando el teorema de Bayes, quitando el denominador (no depende de la clase que

sea asignada) y usando la hipótesis de independencia entre clases, las funciones discriminantes en este clasificador son de la forma:

$$g_y(\underline{x}) = \mathbb{P}(Y = y) \prod_{i=1}^{n} f_{X_i|Y=y}(x_i), \quad \text{con } y \in \mathcal{Y}.$$

Un inconveniente importante es el número de parámetros a estimar para decidir a que clase asignar el vector considerado. Existen enfoques diferentes dependiendo del tipo de valores que expresen los datos. Haremos una distinción entre los datos que procedan de variables discretas y aquellos que sigan una distribución continua.

Si los datos son valores discretos podemos realizar la estimación de las funciones de probabilidad $f_{X_i|Y=y}(x_i)$ tomando el número de datos cuyo resultado sea la clase y como variable de respuesta y dividiendo el número total de aquellos datos donde la variable X_i toma el valor x_i entre el número de datos totales de cuya clase sea y, con $1 \leq i \leq p$.

Por el contrario, si las variables son continuas supondremos que dichos datos siguen una distribución normal univariante y trataremos de estimar los parámetros necesarios para las funciones de densidad. Por ser distribuciones normales tenemos que:

$$f_{X_i|Y=y}(x_i) = \frac{1}{\sqrt{2\pi}\sigma_y} \exp\{-\frac{1}{2}(\frac{x_i - \mu_y}{\sigma_y})^2\}, \quad \forall i = 1, \ldots, p, \quad \text{con } y \in \mathcal{Y}.$$

Necesitamos encontrar una estimación de μ_y y σ_y, para ello podemos considerar la media y la cuasivarianza muestral respectivamente de los datos de la variable X_i con variable de respuesta y.

Debido a la independencia de las variables condicionadas es posible realizar este algoritmo si nos encontramos con algún caso donde las variables sean de distinto tipo, basta con realizar las estimaciones de las variables distinguiendo si son continuas o discretas siguiendo los pasos anteriores.

Por otra parte, para estimar la probabilidad *a priori* de una clase fijada, solo tenemos que dividir el número de datos que tienen esa clase entre el total de datos que disponemos.

Para ver un ejemplo del modelo de Naive Bayes vamos a trabajar con una muestra de 1956 comentarios extraídos de 5 videos diferentes de YouTube[1]. El objetivo va a ser encontrar cuales de estos comentarios se consideran basura o *spam*. Estos comentarios estarán en inglés.

Para este problema necesitaremos las siguientes librerías de *R*:

[1]Los datos se encuentran en https://archive.ics.uci.edu/ml/datasets/YouTube+Spam+Collection

```
library(e1071)
library(SparseM)
library(tm)
library(xtable)
library(caret)
library(wordcloud)
```

Procedemos a leer los datos y a unificarlos en un solo archivo, prescindiendo de todas las variables salvo el texto y el tipo de comentario.

```
spam1 = read.csv("youtube1.csv",sep = ";", stringsAsFactors=FALSE)
spam2 = read.csv("youtube2.csv",sep = ";", stringsAsFactors=FALSE)
spam3 = read.csv("youtube3.csv",sep = ";", stringsAsFactors=FALSE)
spam4 = read.csv("youtube4.csv",sep = ";", stringsAsFactors=FALSE)
spam5 = read.csv("youtube5.csv",sep = ";", stringsAsFactors=FALSE)
spam = rbind(spam1,spam2,spam3,spam4,spam5)
spam = spam[,4:5]
colnames(spam) = c("texto","clase")
spam$clase = as.factor(spam$clase)
```

La clase del texto se considerará una categoría o factor (*spam* o no) y el texto será considerada como una cadena de caracteres.

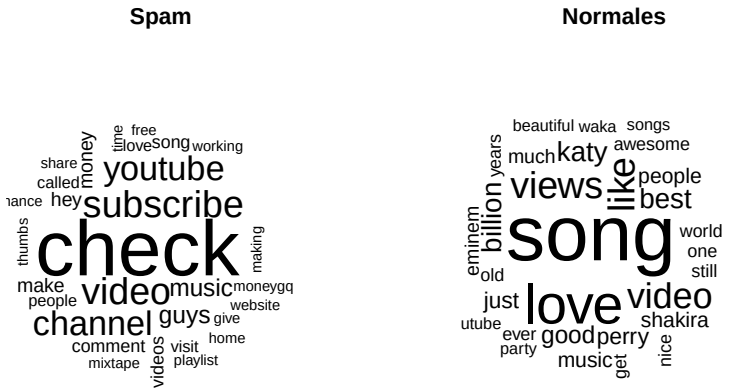
Introducimos ahora las funciones *VectorSource* y *VCorpus*. La primera de ellas habilita a *R* a que interprete cada componente del vector como un texto y la segunda se emplea para crear una estructura donde poder trabajar con cada palabra que aparece en el vector anterior. Para facilitar la lectura de datos cuando realicemos el modelo, pondremos todas las letras en minúscula, se suprimirán los números y aquellas palabras con poco significado o poco relevantes (conjunciones, pronombres, etc), también se eliminarán los símbolos de puntuación. Por último se suprimen los espacios en blanco sobrantes.

```
estructura = VCorpus(VectorSource(spam$texto))
corpspam = tm_map(estructura,content_transformer(tolower))
corpspam = tm_map(corpspam,removeNumbers)
corpspam = tm_map(corpspam,removeWords,stopwords())
corpspam = tm_map(corpspam,removePunctuation)
corpspam = tm_map(corpspam,stripWhitespace)
```

Aquí podemos ver como queda un comentario después de ser filtrado.

```
> as.character(estructura[[6]])
[1] "Hey, check out my new website!! This site is about kids stuff.
 kidsmediausa . com"
> as.character(corpspam[[6]])
[1] "hey check new website site kids stuff kidsmediausa com"
```

Podemos visualizar con un gráfico cuales son las palabras que más se repiten en los comentarios de cada clase. El tamaño indicará mayor repetición.



Con la función *DocumentTermMatrix* se crea una matriz que emplearemos para poder introducir los datos de entrenamiento y prueba dentro del modelo. Por consiguiente, podemos separar los datos una vez filtrados.

```
muestra = sample(1:nrow(spam),1400)
entreno = spam[muestra,]
test = spam[-muestra,]
salidaE = entreno$clase
salidaT = test$clase
dtmspam = DocumentTermMatrix(corpspam)
dtmspamE = dtmspam[muestra,]
dtmspamT = dtmspam[-muestra,]
```

Otra forma de reducir los parámetros a estimar del modelo es simplificar el número de palabras consideradas. Creamos un vector con todas aquellas palabras que se repiten más de 5 veces en todo el texto de entrenamiento y desechamos aquellas que no aparezcan en el mismo ya que no tendrán mucha relevancia a la hora de estimar parámetros y sólo aumentarían el coste computacional del modelo.

```
palabras = findFreqTerms(dtmspamE,5)
frecE = dtmspamE[,palabras]
frecT = dtmspamT[,palabras]
```

Definimos la función palabra para que al introducir los datos de entrenamiento en el clasificador, se sepa si cada palabra en cuestión aparece en el texto que se esté leyendo. Le aplicamos dicha función a los datos para que el modelo pueda interpretar con facilidad si una palabra está dentro del texto o no.

```
palabra = function (x){
x = ifelse(x>0, "Yes","No")
}
```

```
spamE = apply(frecE,MARGIN = 2,palabra)
spamT = apply(frecT,MARGIN = 2,palabra)
```

Creamos el clasificador usando la función contenida en el paquete *e1071* denominada *naiveBayes*. Ademas podemos observar ya el error de entrenamiento.

```
naiveB = naiveBayes(spamE,salidaE)
predE = predict(naiveB,spamE)
print(xtable(confusionmatrix(predE,entreno$clase),digits = 0))
```

	Normal	*Spam*
Normal	665	133
Spam	17	585

A la vista de la tabla el error de entrenamiento del modelo es de 0,1071. Estudiamos ahora el error de prueba y vemos la matriz de confusión del modelo.

```
predT = predict(naiveB,spamT)
confusionMatrix(predT,test$clase,positive = "1")
Confusion Matrix and Statistics
    0   1
0 264  52
1  10 235

Accuracy : 0.8895
95% CI : (0.8606, 0.9142)
No Information Rate : 0.5116
P-Value [Acc > NIR] : < 2.2e-16
Kappa : 0.7796
Mcnemar's Test P-Value : 1.919e-07

Sensitivity : 0.8188
Specificity : 0.9635
Pos Pred Value : 0.9592
Neg Pred Value : 0.8354
Prevalence : 0.5116
Detection Rate : 0.4189
Detection Prevalence : 0.4367
Balanced Accuracy : 0.8912

'Positive' Class : "Spam" = 1
```

Observamos que el error es similar al de entrenamiento lo que implica un buen resultado a la hora de clasificar dado el tamaño de la muestra, por otra parte, el modelo es más permisivo con los comentarios pues tiene más tendencia a cometer errores clasificando en la categoría normal siendo *spam*, algo que también se aprecia en el error de entrenamiento.

3.2. Análisis discriminante cuadrático

El siguiente clasificador es muy similar al análisis discriminante lineal del capítulo anterior. Cuando abordábamos el problema del ADL se suponía normalidad en los datos e igualdad de matriz de varianzas y covarianzas de cada clase.

Para el análisis discriminante cuadrático (ADC) sólo supondremos que los datos se distribuyen a partir de una distribución normal p-variante, es decir:

$$f_{\underline{X}|Y=y}(\underline{x}) = \frac{1}{(2\pi)^{\frac{p}{2}} \mid \Sigma_y|^{\frac{1}{2}}} \exp\{-\frac{1}{2}(\underline{x} - \underline{\mu}_y)^t \Sigma_y^{-1}(\underline{x} - \underline{\mu}_y)\}, \quad \forall y \in \mathcal{Y},$$

como el ADC está basado en el clasificador de Bayes, la regla de clasificación consiste en asignar al vector $\underline{x}$ la clase y^* si:

$$g_{y^*}(\underline{x}) \geq g_y(\underline{x}) \iff \mathbb{P}(Y = y^* | \underline{X} = \underline{x}) \geq \mathbb{P}(Y = y | \underline{X} = \underline{x}), \quad \forall y \in \mathcal{Y},$$

como siempre, en caso de empate entre dos o más clases, se asignará aleatoriamente a una de las clases anteriores. Al igual que en el caso lineal se puede aplicar el teorema de Bayes y utilizar la suposición de normalidad obteniendo:

$$\frac{1}{|\Sigma_{y^*}|^{\frac{1}{2}}} \exp\{-\frac{1}{2}(\underline{x}-\underline{\mu}_{y^*})^t \Sigma_{y^*}^{-1}(\underline{x}-\underline{\mu}_{y^*})\}\pi(y^*) \geq \frac{1}{|\Sigma_y|^{\frac{1}{2}}} \exp\{-\frac{1}{2}(\underline{x}-\underline{\mu}_y)^t \Sigma_y^{-1}(\underline{x}-\underline{\mu}_y)\}\pi(y), \quad \forall y \in \mathcal{Y}.$$

Ahora procedemos de la misma forma y aplicamos logaritmos en ambos lados resultando:

$$-\frac{1}{2}\log(|\Sigma_{y^*}|) - \frac{1}{2}(\underline{x}-\underline{\mu}_{y^*})^t \Sigma_{y^*}^{-1}(\underline{x}-\underline{\mu}_{y^*}) + \log(\pi(y^*)) \geq -\frac{1}{2}\log(|\Sigma_y|) - \frac{1}{2}(\underline{x}-\underline{\mu}_y)^t \Sigma_y^{-1}(\underline{x}-\underline{\mu}_y) + \log(\pi(y)),$$

donde al no tener la suposición de igualdad de matrices de varianzas y covarianzas no es posible reducir más en la desigualdad anterior. Por tanto, las funciones discriminantes del clasificador vienen dadas por la siguiente expresión:

$$g_y(\underline{x}) = -\frac{1}{2}\log(|\Sigma_y|) - \frac{1}{2}(\underline{x} - \underline{\mu}_y)^t \Sigma_y^{-1}(\underline{x} - \underline{\mu}_y) + \log(\pi(y)), \quad \forall y \in \mathcal{Y}.$$

El segundo término en la expresión anterior es de orden cuadrático (de ahí el nombre de este clasificador). Para poder realizar este algoritmo se necesita calcular los estimadores de las matrices de varianzas y covarianzas para cada clase, así como su media y su probabilidad *a priori*. Al ser distribuciones normales los estimadores que se emplearán serán la media muestral y la matriz de varianzas y cuasivarianzas muestrales. Para la probabilidad *a priori* como siempre se tomará el número de datos de entrenamiento que presenten la categoría y dividido entre el número total de datos de entrenamiento.

Para el siguiente ejemplo se toma una muestra de tamaño 150 de tres tipos diferentes de iris[2] (setosa, versicolor y virginica) a las que se le realizan diferentes medidas del pétalo y el sépalo recogidas en las 4 siguientes variables:

- $X_1 \equiv$ Largo del sépalo.
- $X_2 \equiv$ Ancho del sépalo.
- $X_3 \equiv$ Largo del pétalo.
- $X_4 \equiv$ Ancho del pétalo.
- $Y \equiv$ Tipo de iris.

Procedemos a leer los datos y a realizar la prueba de normalidad de Shapiro-Wilk para probar las hipótesis. Ya que los datos se encuentran ordenados por clase tendremos será mas fácil aplicar la prueba de normalidad.

```
iris = read.table("Iris.txt",sep =",")
colnames(iris) = c(paste("X",1:4,sep=""),"Y")
apply(iris[1:50,1:4],2,shapiro.test)
apply(iris[51:100,1:4],2,shapiro.test)
apply(iris[101:150,1:4],2,shapiro.test)
```

Los p-valores obtenidos se encuentran en la siguiente tabla:

	Setosa	Versicolor	Virginica
X_1	0.4595	0.4647	0.2583
X_2	0.2715	0.338	0.1809
X_3	0.0548	0.1585	0.1098
X_4	8.659e-07	0.02728	0.08695

A nivel de significación 0,05 las tres primeras variables podemos suponer que siguen una distribución normal para poder ver como resulta un límite de decisión no lineal, primero efectuaremos un ejemplo usando solamente las dos primeras variables. *R* tiene implementado este clasificador con la función *qda* dentro del paquete *MASS* con el que ya trabajamos en el capítulo anterior.

```
set.seed(264)
muestra = sample(1:nrow(iris),100)
entreno = iris[muestra,]
test = iris[-muestra,]
irisqda = qda(Y ~ X1 + X2 ,entreno[,c(1,2,5)])
```

[2]Los datos del ejemplo están recogidos en https://archive.ics.uci.edu/ml/machine-learning-databases/iris/

Como en los casos anteriores buscamos los errores de entrenamiento y prueba comenzando por el primero de los mencionados.

```
predE = predict(irisqda,entreno[,c(1,2,5)])
print(xtable(confusionmatrix(entreno$Y,predE$class)))
```

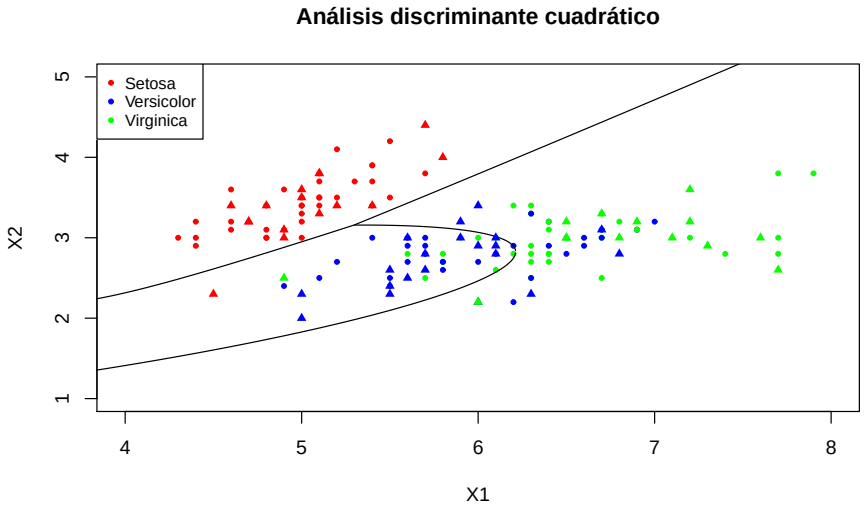
	Iris-setosa	Iris-versicolor	Iris-virginica
Iris-setosa	34	0	0
Iris-versicolor	0	18	12
Iris-virginica	0	12	24

El error de entrenamiento es de 0,24 que suele ser mucho más alto del que se podría obtener con otros clasificadores en este conjunto de datos [3]. También es cierto que solo estamos empleando la mitad de la información disponible usando dos variables. Veamos el error de prueba:

```
predT = predict(irisqda,test[,c(1,2,5)])
confusionMatrix(predT$class,test$Y)
Confusion Matrix and Statistics
                 Iris-setosa Iris-versicolor Iris-virginica
Iris-setosa               15               0              0
Iris-versicolor            1              14              1
Iris-virginica             0               6             13

Overall Statistics
Accuracy : 0.84
95% CI : (0.7089, 0.9283)
No Information Rate : 0.4
P-Value [Acc > NIR] : 1.986e-10
Kappa : 0.7611
Mcnemar's Test P-Value : NA

Statistics by Class:
                     Iris-setosa Iris-versicolor Iris-virginica
Sensitivity                 1.00          0.9500          1.0000
Specificity                 1.00          1.0000          0.9722
Pos Pred Value              1.00          1.0000          0.9333
Neg Pred Value              1.00          0.9677          1.0000
Prevalence                  0.32          0.4000          0.2800
Detection Rate              0.32          0.3800          0.2800
Detection Prevalence        0.32          0.3800          0.3000
Balanced Accuracy           1.00          0.9750          0.9861
```

[3]En este enlace aparecen los resultados usando redes neuronales con este conjunto de datos http://lab.fs.uni-lj.si/lasin/wp/IMIT_files/neural/doc/seminar8.pdf

El error de prueba es algo mejor aunque sigue lejos de lo que otros algoritmos pudieran conseguir. Por último, podemos ver una gráfica con los puntos representados en el plano.



En triángulos aquellos elementos del conjunto de prueba. Notar que la clase setosa es linealmente separable de las otras dos[4] por ello el límite de decisión para esa clase se asemeja a una recta.

Para tratar de mejorar los resultados vamos a realizar otro modelo usando la tercera variable que también podíamos suponer normal.

```
irisqda3 = qda(Y ~ X1 + X2 + X3, entreno[,-4])
predE3 = predict(irisqda3,entreno[,-4])
print(xtable(confusionmatrix(predE3$class,entreno$Y),digits = 0))
```

	Iris-setosa	Iris-versicolor	Iris-virginica
Iris-setosa	34	0	0
Iris-versicolor	0	27	2
Iris-virginica	0	3	34

Vemos como la introducción de la tercera variable mejora los resultados pasando a tener un error de entrenamiento de 0.05.

```
predT3 = predict(irisqda3,test[,-4])
confusionMatrix(predT3$class,test$Y)
Confusion Matrix and Statistics
```

[4]Vease https://pbil.univ-lyon1.fr/R/pdf/course5.pdf

	Iris−setosa	Iris−versicolor	Iris−virginica
Iris−setosa	16	0	0
Iris−versicolor	0	19	0
Iris−virginica	0	1	14

Overall Statistics

Accuracy : 0.98
95% CI : (0.8935, 0.9995)
No Information Rate : 0.4
P−Value [Acc > NIR] : < 2.2e−16
Kappa : 0.9698

Statistics **by** Class:

	Iris−setosa	Iris−versicolor	Iris−virginica
Sensitivity	1.00	1.0000	0.9333
Specificity	1.00	0.9677	1.0000
Pos Pred Value	1.00	0.9500	1.0000
Neg Pred Value	1.00	1.0000	0.9722
Prevalence	0.32	0.3800	0.3000
Detection Rate	0.32	0.3800	0.2800
Detection Prevalence	0.32	0.4000	0.2800
Balanced Accuracy	1.00	0.9839	0.9667

Se observa también como sólo ha existido un único error de clasificación en los datos de prueba. El inconveniente más importante que presenta este clasificador es la suposición de normalidad que posee pues impone una condición fuerte para poder usarse en los datos.

3.3. K vecinos más cercanos

El método de los k vecinos más cercanos (k-nn) es un algorítmo no lineal cuya primera aparición data de 1951 por Fix y Hodges. Se necesita para empezar, fijar un número k entero y positivo. Para cada vector $\underline{x}$ perteneciente al conjunto de entrenamiento hay que buscar aquellos k vectores dentro del conjunto que se encuentren a menor distancia (en este caso se considerará siempre la norma euclídea para medir la distancia entre dos vectores aunque el argumento que se expondrá es válido para cualquier otra distancia) de $\underline{x}$.

Una vez identificados los k vectores que se encuentran a distancia mínima denotamos ese conjunto de vectores por $K_{\underline{x}}$, observamos su clase asociada. Fijado un vector $\underline{x}$ vamos a considerar ahora un estimador de la probabilidad condicionada $\mathbb{P}(Y = y|\underline{X} = \underline{x})$ que viene dado por:

$$\frac{1}{k}\sum_{(\underline{x}_i,y_i)\in K_{\underline{x}}} I(y=y_i), \quad \forall y\in\mathcal{Y},$$

siendo $I(y=y_i)$ la función indicadora que toma el valor 1 si $y=y_i$ y 0 en caso contrario, con $1\leq i\leq k$. La regla de asignación será en este caso la misma que en el clasificador de Bayes, clasificar en la categoría más probable, es decir:

$$h(\underline{x}) = \arg\max_{y\in\mathcal{Y}} \mathbb{P}(Y=y|\underline{X}=\underline{x}) = \arg\max_{y\in\mathcal{Y}} \frac{1}{k}\sum_{(\underline{x}_i,y_i)\in K_{\underline{x}}} I(y=y_i).$$

Diremos también que un vector pertenece al límite de decisión del clasificador si entre los vectores más cercanos predominan dos o más clases.

La elección del valor de k es fundamental para este proceso de clasificación, entendiendo que valores muy grandes de k conllevan la elección más probable pero también podría ser un inconveniente si el vector no pertenece a la clase que más se repite. Por otro lado, una elección pequeña de k trae consigo una alta variabilidad y unos límites de decisión inestables, esto es, un ligero cambio en los datos de entrenamiento implica un posible cambio drástico en los límites de decisión.

Por estas razones no podremos escoger ni un número muy grande ni muy pequeño para k. Para elegir dicho número consideraremos varios valores de k y observaremos los errores obtenidos.

Para ver su implementación en R tenemos un estudio sobre unas medidas que se han realizado en la espalda[5] de 310 pacientes y queremos ver si el paciente presenta alguno de los dos tipos de lesión (hernia discal y espondilolistesis) o no. Por lo tanto, tenemos un problema con tres clases y las siguientes variables.

- $X_1 \equiv$ Incidencia pélvica.
- $X_2 \equiv$ Inclinación pélvica.
- $X_3 \equiv$ Lordosis lumbar.
- $X_4 \equiv$ Pendiente del sacro.
- $X_5 \equiv$ Radio pélvico.
- $X_6 \equiv$ Grado de espondilolistesis.
- $Y \equiv$ Presenta o no alguna de las lesiones anteriores.

[5]Los datos para este ejemplo están en https://archive.ics.uci.edu/ml/datasets/Vertebral+Column

Comenzamos abriendo el fichero de datos y cargando las librerías necesarias para el modelo. En *R* existen varias funciones que implementan este clasificador, la que se empleará en el ejemplo será la contenida en la librería *caret* con la que ya se ha trabajado previamente y la función del clasificador es *knn3*.

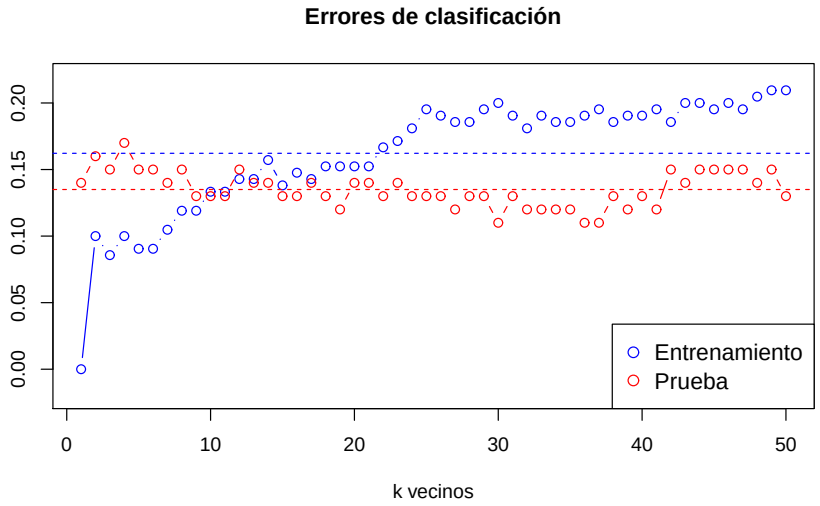
```
library(caret)
library(biotools)
library(xtable)
espalda = read.table("Espalda1.txt", sep="")
colnames(espalda) = c(paste("X",1:6,sep=""),"Y")
set.seed(1241243)
mezcla2 = sample(1:310,310)
espalda = espalda[mezcla2,]
ent = espalda[1:210,]
test = espalda[211:310,]
```

Antes de elegir el valor de k vamos ver en un gráfico los errores de entrenamiento y prueba en función del aumento de k creado mediante el siguiente bucle:

```
erroresE = NULL
erroresT = NULL
for (i in 1:50){
k = knn3(Y~.,ent,k = i)
entrenamiento = predict(k,ent,type = "class")
ec = confusionmatrix(entrenamiento,ent$Y)
e = 1-(sum(diag(ec))/sum(ec))
erroresE = cbind(erroresE,e)
prueba = predict(k,test,type = "class")
et = confusionmatrix(prueba,test$Y)
t = 1- (sum(diag(et))/sum(et))
erroresT = cbind(erroresT,t)
}
```

Representamos pues la linea de puntos para visualizar los errores.

```
plot(1:50,erroresE,type = "b",col = "blue",xlab = "k vecinos", ylab = "",
     ylim = c(-0.02,0.22))
lines(1:50,erroresT,type = "b",col = "red")
abline(h = mean(erroresE),col = "blue", lty = 2)
abline(h = mean(erroresT),col = "red",lty = 2)
legend("bottomright", legend= c("Entrenamiento","Prueba"), pch = 1,
       cex = 1.2, col =c("blue","red"))
title(main = "Errores de clasificación")
```



Las lineas discontinuas se muestran las medias de ambos errores. A la vista de la gráfica las conclusiones que podrían sacarse son: que el modelo está algo sobreajustado para bajos valores de k y que para el resto de modelos, la tendencia del error de entrenamiento es aumentar mientras que el de prueba se estabiliza. Para seguir detallando este ejemplo, se considerará el valor $k = 10$ pues generaliza correctamente los datos que se le han incluido. Pasamos entonces a detallar los errores cometidos por este clasificador.

```
k10 = knn3(Y ~ ., ent, k = 10)
predE10 = predict(k10, ent, type = "class")
print(xtable(confusionmatrix(predE10, ent$Y), digits = 0))
```

	Hernia	Normal	SL
Hernia	28	9	1
Normal	12	61	3
SL	1	1	94

El error de entrenamiento obtenido es de 0,1285714. Por último veamos el error de prueba.

```
pred10 = predict(k10, test, type = "class")
confusionMatrix(pred10, test$Y)
Confusion Matrix and Statistics

    DH NO SL
DH  13  5  0
NO   6 22  0
SL   0  2 52
```

```
Overall Statistics


Accuracy : 0.87
95% CI : (0.788, 0.9289)
No Information Rate : 0.52
P-Value [Acc > NIR] : 1.188e-13

Kappa : 0.7847
Mcnemar's Test P-Value : NA


Statistics by Class:
```

	Hernia	Normal	SL
Sensitivity	0.6842	0.7586	1.0000
Specificity	0.9383	0.9155	0.9583
Pos Pred Value	0.7222	0.7857	0.9630
Neg Pred Value	0.9268	0.9028	1.0000
Prevalence	0.1900	0.2900	0.5200
Detection Rate	0.1300	0.2200	0.5200
Detection Prevalence	0.1800	0.2800	0.5400
Balanced Accuracy	0.8112	0.8371	0.9792

Donde vemos que el error de prueba es de 0,13. Este clasificador detecta mejor los casos de espondilolitesis que los de hernia. Para obtener buenos resultados este algoritmo necesita un tamaño de muestra grande, sin embargo el coste computacional aumenta debido al número de cálculos a realizar para calcular las distancias entre vectores. Este problema también aparece si el número de variables en los datos es grande.

3.4. Perceptrón multicapa

En el capítulo anterior se hacía referencia a las redes neuronales artificiales y en particular a la forma más simple de ellas que es el Perceptrón. También se mencionaba las limitaciones que este algoritmo posee tales como que el problema ha de ser binario y las dos clases linealmente separables para que el algoritmo pueda aplicarse. Con este método buscamos generalizar ese concepto detallado previamente empleando un modelo con más neuronas distribuidas en un número indefinido de capas que sea capaz de solventar las limitaciones anteriores.

El modelo del Perceptrón multicapa aunque fue propuesto por primera vez por Paul Werbos en 1974 no se dio a conocer hasta 1986 por el grupo liderado por Rumelhart gracias a la introducción de un algoritmo llamado retropropagación de errores (*Backpropagation*). Hoy en día el Perceptrón multicapa sigue obteniendo buenos resultados y es empleado con frecuencia si se pretende utilizar algún sistema de redes neuronales.

Esta red es de tipo *feedforward* pues las neuronas procesan la información recibida y la propagan hacia la capa siguiente hasta crear una respuesta en la capa de salida.

Partimos por tanto de un sistema con m capas de las cuales $m-2$ son ocultas con $m \geq 3$. En cada capa se encuentra un número n_m de neuronas salvo en la capa de entrada en la cual disponemos de exactamente p neuronas (una para cada variable del conjunto de datos).

El peso sináptico entre la neurona j de la capa k que llega a partir de la neurona i de la capa anterior vendrá definido por $w_{j,i}^k$, con $i = 1, \ldots, n_{k-1}$, $j = 1, \ldots, n_k$, $k = 2, \ldots, m$. En el Perceptrón multicapa la función de activación para pasar la información de una neurona a otra será no lineal y se asemeja a la función signo usada en el caso del Perceptrón con la diferencia de que en este caso se le impondrá una mayor regularidad a la función. Las funciones a considerar serán de tipo sigmoide y aquí se dejan un par de ejemplos de estas:

- Función logística:

$$y = \frac{1}{1 - e^{-x}}, \quad y \in [0, 1].$$

- Función tangente hiperbólica:

$$y = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad y \in [-1, 1].$$

Para el caso del Perceptrón multicapa, es usual elegir la función logística aunque es posible emplear otro tipo de función sigmoide. Para lo sucesivo definimos $\phi(x)$ como la función logística anterior que emplearemos como función de activación.

El funcionamiento de una neurona se puede entender como un proceso de dos partes. La primera sería la recopilación de la información recibida por los datos o mediante las neuronas a de la capa anterior y la segunda, la propagación de dicha información a las neuronas posteriores o la respuesta final generada por la última capa. Para la primera parte, dado un vector $\underline{x}$ se definen:

$$\begin{aligned} a_j^1 &= x_j, && \forall j = 1, \ldots, p, \\ a_j^2 &= \sum_{i=1}^{p} w_{j,i}^2 x_i + w_{j,0}^2, && \forall j = 1, \ldots, n_2, \\ a_j^k &= \sum_{i=1}^{n_{k-1}} w_{j,i}^k z_i^{k-1} + w_{j,0}^k, && \forall j = 1, \ldots, n_k, \quad \forall k = 3, \ldots, m, \end{aligned}$$

siendo $w_{j,0}^k$ el umbral de cada neurona y $z_j^k = \phi(a_j^k)$ para $j = 1, \ldots, n_k$ y $k = 2, \ldots, m-1$. Los valores a_j^k representan la información recibida en la neurona j de la capa k que ha sido propagada, mientras que los valores z_j^k se entienden como la información que va a ser transmitida desde la neurona j de la capa k hacia la siguiente capa de neuronas.

Con estos valores podemos comenzar a describir el algoritmo de retropropagación de errores. El objetivo de este algoritmo consiste en minimizar el error obtenido a la hora de clasificar datos, por lo tanto si disponemos de n datos de entrenamiento la función objetivo será:

$$E = \sum_{i=1}^{n} E^s = \sum_{s=1}^{n} |\underline{y}_s - \underline{\hat{y}}_s|^2,$$

donde E^s es el posible error del dato s, $\underline{y}_s$ es el vector de respuestas reales frente a $\underline{\hat{y}}_s$ respuestas originadas por la red neuronal artificial ambos con dimensión n_m, con $s = 1, \ldots, n$. El valor de la respuesta (y en consecuencia del error anterior) depende de los parámetros anteriores, por lo que es de interés conocer la derivada de uno de los sumandos con respecto a uno de los pesos sinápticos para tratar de encontrar el mínimo, considerando uno cualquiera de la última capa se tiene:

$$\frac{\partial E^s}{\partial w_{j,i}^m} = \frac{\partial E^s}{\partial a_j^m} \frac{\partial a_j^m}{\partial w_{j,i}^m} = \delta_j^m z_i^{m-1}, \quad \forall j = 1, \ldots, n_m, \quad \forall i = 1, \ldots, n_{m-1},$$

definiendo $\delta_j^m = \frac{\partial E^s}{\partial a_j^m}$. En general para $k \geq 2$ obtenemos:

$$\frac{\partial E^s}{\partial w_{j,i}^k} = \frac{\partial E^s}{\partial a_j^k} \frac{\partial a_j^k}{\partial w_{j,i}^k} = \begin{cases} \delta_j^2 x_j, & \text{si} \quad k = 2, \\ \delta_j^k z_i^{k-1}, & \text{si} \quad k \geq 3, \end{cases} \qquad \forall j = 1, \ldots, n_m, \quad \forall i = 1, \ldots, n_{m-1},$$

tomando de nuevo el valor $\delta_j^k = \frac{\partial E^s}{\partial a_j^k}$. El objetivo es ahora tratar de dar una definición para los valores δ_j^k que aparecen. Para ello empleamos otra vez la regla de la cadena resultando:

$$\delta_j^k = \frac{\partial E^s}{\partial a_j^k} = \sum_{l=1}^{n_{k+1}} \frac{\partial E^s}{\partial a_l^{k+1}} \frac{\partial a_l^{k+1}}{\partial a_j^k} = \sum_{l=1}^{n_{k+1}} \delta_l^{k+1} \frac{\partial a_l^{k+1}}{\partial a_j^k}, \quad \forall j = 1, \ldots, n_k, \quad \forall k = 2, \ldots, m-1,$$

que usando las definiciones anteriores y la regla de la cadena llegamos a:

$$\frac{\partial a_l^{k+1}}{\partial a_j^k} = \frac{\partial a_l^{k+1}}{\partial z_j^k}\frac{\partial z_j^k}{\partial a_j^k} = w_{l,j}^{k+1}\phi'(a_j^k), \quad \forall l = 1,\dots,n_{k+1}, \quad \forall j = 1,\dots n_k, \quad \forall k = 2,\dots,m-1,$$

sustituyendo se llega a la fórmula general de este algoritmo, que es:

$$\delta_j^k = \phi'(a_j^k)\sum_{l=1}^{n_{k+1}} \delta_l^{k+1} w_{l,j}^{k+1}, \quad \forall j = 1,\dots,n_k, \quad \forall k = 2,\dots,m-1.$$

Puede hacerse una observación y es que para calcular δ_j^k son necesarios los valores obtenidos en la capa posterior. Se deduce pues que las variaciones de los pesos sinápticos de capas anteriores dan lugar a variaciones en los pesos sinápticos de las capas posteriores. Falta por tanto calcular los valores de δ_j^m con $j = 1,\dots,n_m$. Para ello podemos entender que el valor de $a_j^m \equiv \hat{y}_{s,j}$ puesto que se trata de un valor en la neurona de la capa de salida y entendemos que es una respuesta de la red. Bajo esa suposición tenemos que:

$$\frac{\partial E^s}{\partial a_j^m} = \frac{\partial E^s}{\partial \hat{y}_{s,j}} = -2(y_{s,j} - \hat{y}_{s,j}), \quad \forall s = 1,\dots,n, \quad j = 1,\dots,n_m.$$

E^s denota el error en el vector s del conjunto de entrenamiento y está definido por dos vectores $\underline{y}_s$ e $\underline{\hat{y}}_s$, el calculo anterior denota que el error se está estudiando componente a componente para este vector de salida.

Para poder optimizar el valor de los umbrales basta con estudiar la siguiente derivada:

$$\frac{\partial E^s}{\partial w_{j,0}^k} = \frac{\partial E^s}{\partial a_j^k}\frac{\partial a_j^k}{\partial w_{j,0}^k} = \delta_j^k, \quad \forall j = 1,\dots,n_k, \quad \forall k = 2,\dots,m,$$

cuyo valor ya ha sido analizado previamente. Una vez obtenidas todas las expresiones de las derivadas, notamos que los valores δ_j^m con $j = 1,\dots,n_m$ son calculables dados unos pesos sinápticos y unos umbrales. En consecuencia, es posible ir propagando hacia atrás el error cometido en la capa de salida e ir calculando las demás derivadas. Con esta información realizamos un proceso iterativo para calcular las derivadas anteriores a fin de encontrar un mínimo local. El método usualmente utilizado para este fin es el método del gradiente. Fijado s, las iteraciones para un peso sináptico son de la forma:

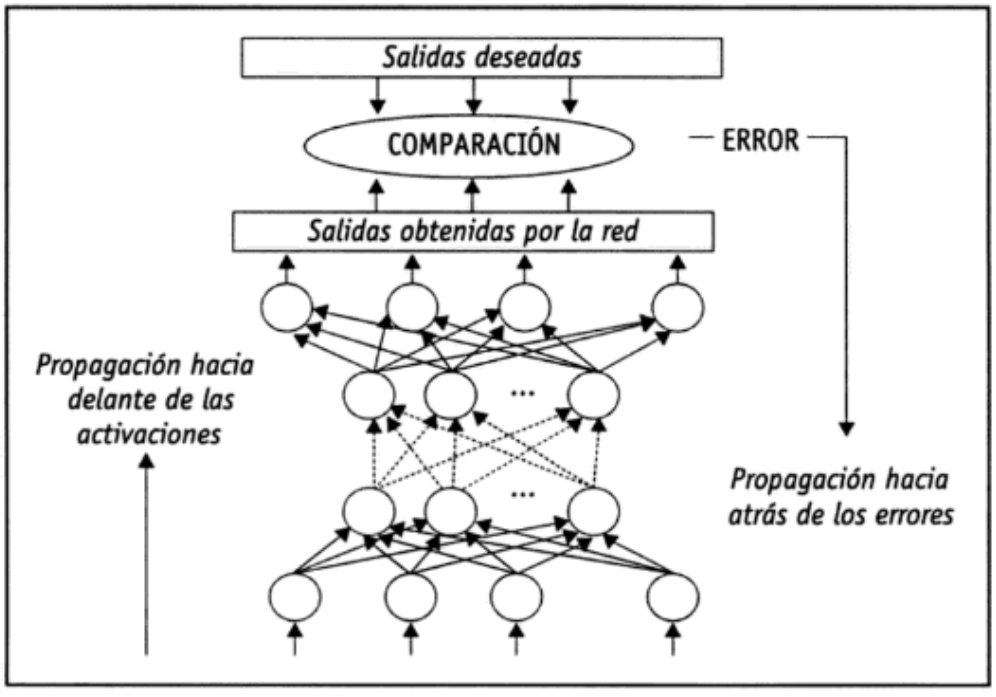
$$(w_{j,i}^k)^{r+1} = (w_{j,i}^k)^r - \rho\frac{\partial E^s}{\partial (w_{j,i}^k)^r}, \quad \forall j = 1,\dots,n_k, \quad \forall i = 1,\dots,n_{k-1}, \quad \forall k = 2,\dots,m,$$

con $\rho > 0$ un parámetro de escala.

En resumen, el algoritmo de la retropropagación de errores sigue los siguientes pasos:

1. Establecer aleatoriamente los pesos sinápticos y umbrales.
2. Obtener una respuesta de cada dato de entrenamiento y evaluar los errores cometidos.
3. Calcular los incrementos infinitesimales de cada peso.
4. Actualizar los pesos y los umbrales.
5. Calcular las respuestas y los errores.
6. Si se considera que el error obtenido es válido, parar. En caso contrario volver al paso 3.

En este algoritmo, se propaga hacia delante la información y hacia atrás los errores. En la siguiente imagen podemos observar como funciona la retropropagación de errores mediante un grafo dirigido.



Al igual que ocurre con la función de activación, existen otros enfoques para minimizar el error basados en otras funciones (diferenciables) diferentes, no obstante el algoritmo y la forma de proceder no varían.

Para ver la implementación en R, trataremos de usar un Perceptrón multicapa para clasificar diferentes tipos de imagenes[6] a color. De todo el conjunto de datos que disponemos tomaremos las imágenes en las que aparezcan aviones, coches, motos o personas.

Al igual que en el ejemplo de las máquinas de vectores soporte lineales, cada variable representará un píxel pero a diferencia del ejemplo de la sección anterior, al tener color la fotografía en cada uno de los píxeles se representan 3 colores (en consecuencia 3 variables): una representa el grado de rojo, otra el grado de verde y la última un grado de azul. Todas están medidas con un número entre 0 y 1 siendo 1 el grado máximo de saturación del color.

Necesitaremos las siguientes librerías de R para procesar las imágenes y poder construir la red neuronal artificial.

```
source("http://bioconductor.org/biocLite.R"); biocLite("EBImage")
library(EBImage)
library(keras)
library(biotools)
library(caret)
library(xtable)
library(RSNNS)
```

Creamos un vector con el nombre de las imágenes que vamos a tratar comenzando por las que contienen un avión.

```
fotos = rep(0,722)

for (i in 0:9){
fotos[i+1] = paste('airplane_000',i,'.jpg',sep='')
}
for (i in 10:99){
fotos[i+1] = paste('airplane_00',i,'.jpg',sep='')
}
for (i in 100:721){
fotos[i+1] = paste('airplane_0',i,'.jpg',sep='')
}
```

Con la función *readImage* obtenemos una matriz que nos da los grados de cada color en cada píxel por toda la imagen. Tras aplicarle esta función a todas las fotografías reduciremos el tamaño de la imagen con la función *resize* a 28×28 píxeles a fin de reducir la dimensión total del ejemplo. Por último transformamos esas matrices en vectores de dimensión 2352 con la función *array_reshape*.

[6] https://www.kaggle.com/prasunroy/natural-images

```
misfotos = list()
for (i in 1:722){
misfotos[[i]] = readImage(fotos[i])
}
for (i in 1:722){
misfotos[[i]] = resize(misfotos[[i]],28,28)
}
for (i in 1:722){
misfotos[[i]] = array_reshape(misfotos[[i]],c(28,28,3))
}
```

Y por último añadimos todos esos vectores a una matriz que emplearemos como matriz de datos una vez hayamos obtenido todas las clases.

```
datos = NULL
for (i in 1:722){
datos = rbind(datos,misfotos[[i]])
}
```

Dado que el proceso para obtener los datos de las demás clases es análogo, se omiten los detalles de su obtención. Al final trabajaremos con una matriz de dimensiones 3464×2352. Como las clases se han añadido individualmente, es posible crear un vector de respuestas para la variable objetivo. Para obtener los conjuntos de entrenamiento y prueba trataremos de mezclar los datos y extraer los conjuntos posteriormente.

```
clases = as.factor(c(rep("Avion",722),rep("Coche",968),rep("Moto",788),
                     rep("Persona",986)))
set.seed(24564)
muestra = sample(1:3464,2500)
entreno = datos[muestra,]
test = datos[-muestra,]
```

Debido al gran número de variables, es recomendable realizar una reducción de dimensión antes de elaborar la red. Empleamos pues el análisis de componentes principales sobre el conjunto de entrenamiento y tomamos un número de componentes que no nos haga perder demasiada información.

```
componentes = princomp(entreno)
> sum((componentes$sdev[1:150])^2)/sum((componentes$sdev)^2)
[1] 0.8514709
```

Con 150 componentes se consigue explicar algo más de un 85 % de la variabilidad total de los datos, por lo tanto asumimos esa pérdida del 15 % y continuamos con 150 variables. Calculamos ahora las puntuaciones de ambos conjuntos.

```
compE = componentes$scores[,1:150]
compT = predict(componentes,test)
compT = compT[,1:150]
```

El siguiente paso es separar las salidas en entrenamiento y prueba y crear una matriz con las salidas deseadas. Para ello usamos la función *decodeClassLabels* que descompone las 4 categorías en 4 variables indicadoras que servirán para señalar a R el número de neuronas en la capa de salida que vamos a utilizar.

```
clasesE = clases[muestra]
clasesT = clases[-muestra]
targetE = decodeClassLabels(clasesE)
targetT = decodeClassLabels(clasesT)
```

Con todos los datos ya disponibles, pasamos a crear el Perceptrón multicapa con la función *mlp*. Dado que la variable objetivo es expresada en una matriz con 4 columnas (una para cada variable indicadora) obtendremos una capa de salida con 4 neuronas, cada una asociada a una clase diferente y la regla de clasificación será asignar $\underline{x}$ a la clase cuya neurona produzca un valor de salida más grande. Puesto que estamos usando funciones logísticas las salidas son valores entre 0 y 1 que pueden ser interpretados como la probabilidad de asignar a esa clase. Por otra parte, especificamos también el número de capas ocultas con sus respectivas neuronas, el valor del parámetro de escala ρ y el número de iteraciones a realizar.

```
perceptron1 = mlp(compE, targetE, size = c(20,5), maxit = 500,
                  learnFuncParams = c(0.1))
```

En este caso, el modelo está formado por dos capas ocultas con 20 y 5 neuronas respectivamente, se han realizado 500 iteraciones y el parámetro de escala es 0,1. Veamos el error de entrenamiento:

```
predE = predict(perceptron1,compE)
print(xtable(confusionMatrix(predE, targetE), digits = 0))
```

	Avión	Coche	Moto	Persona
Avión	508	1	0	0
Coche	0	707	1	0
Moto	0	1	572	0
Persona	0	2	0	708

Puede apreciarse que solo ha errado en 5 casos de 2500 disponibles. Pasamos a estudiar el error de prueba para terminar de evaluar el clasificador.

```
predT = predict(perceptron1,compT)
confusionMatrix(predT, clasesT)
```

```
Confusion Matrix and Statistics
    1   2   3   4
1 202  19   9   2
2   4 231   8   2
3   6   4 196   4
4   2   3   2 270

Overall Statistics

Accuracy : 0.9326
95% CI : (0.9149, 0.9476)
No Information Rate : 0.2884
P-Value [Acc > NIR] : < 2e-16

Kappa : 0.9098
Mcnemar's Test P-Value : 0.05016

Statistics by Class:
```

	Avion: 1	Coche: 2	Moto: 3	Persona: 4
Sensitivity	0.9439	0.8988	0.9116	0.9712
Specificity	0.9600	0.9802	0.9813	0.9898
Pos Pred Value	0.8707	0.9429	0.9333	0.9747
Neg Pred Value	0.9836	0.9638	0.9748	0.9884
Prevalence	0.2220	0.2666	0.2230	0.2884
Detection Rate	0.2095	0.2396	0.2033	0.2801
Detection Prevalence	0.2407	0.2541	0.2178	0.2873
Balanced Accuracy	0.9520	0.9395	0.9465	0.9805

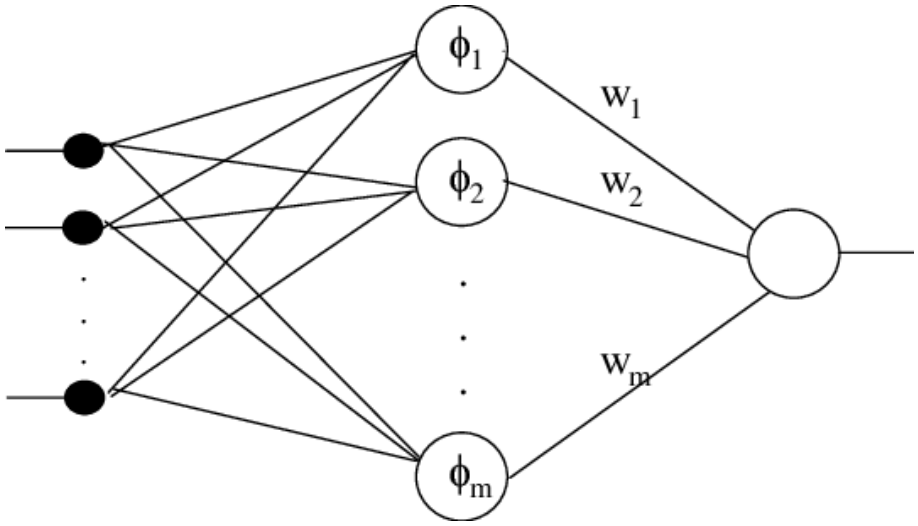
Cuyo error, aunque es ligeramente más elevado, sigue siendo bastante bajo. Se concluye con que esta red generalmente, acierta en las clasificaciones de estos datos. Aunque es habitual emplear redes neuronales para la clasificación de imágenes, la elección de la arquitectura de la red con la que se pretenda trabajar es muy importante para evitar problemas de sobreajustes y malas clasificaciones.

Un resultado importante acerca de las capas ocultas es que, las funciones de tipo sigmoide son capaces de aproximar cualquier función si la red neuronal artificial posee al menos 1 capa oculta (vease Funakashi 1989).

3.5. Redes de base radial

Las redes de base radial fueron propuestas como un método discriminante por Broomhead y Lowe en 1988. Este sistema está basado en redes neuronales artificiales con conexiones hacia delante y una única capa oculta. A diferencia del Perceptrón multicapa, solo existen pesos sinápticos desde la capa oculta hacia la capa de salida.

En esta clase de redes la información producida por los datos es propagada a las neuronas de la capa oculta donde es evaluada mediante una clase específica de funciones de activación denominadas funciones de base radial. Una vez evaluadas son de nuevo propagadas hacia la capa de salida con unos pesos sinápticos a determinar.



Aquí tenemos un ejemplo de una red de base radial con m neuronas en la capa oculta y una única neurona en la capa de salida. Además de las funciones de base radial, introducimos dos tipos de parámetros necesarios para la elaboración de esta red. Por un lado tenemos un parámetro h que servirá para regularizar la función que consideremos y por otro tenemos los parámetros $\underline{\mu}_i$ que utilizaremos para centralizar los datos en torno a un valor que nos interese, con $i = 1, \ldots, m$. Para construir un modelo de red de base radial tenemos que realizar 3 pasos:

1. Especificar las funciones de base radial. Estas pueden ser independientes del conjunto de datos de entrenamiento aunque los parámetros incluidos en ellas no lo sean.

2. Determinar el número de centros y la posición de los mismos.

3. Determinar los pesos sinápticos que dependen de los datos.

Aunque la elección del tipo de funciones puede ser complicada, no es tan importante como la elección de los centros y su posición de cara a elaborar la red. Estas funciones serán de tipo no lineal y a continuación se exponen algunas de estas funciones dados un vector $\underline{x}$, un centro $\underline{\mu}$, y un parámetro de regularización h.

Funciones	$\phi(z), \quad z = \frac{\lvert\underline{x} - \underline{\mu}\rvert}{h}$.
Normal	e^{-z^2}.
Exponencial	e^{-z}.
Cuadrática	$z^2 + \alpha z + \beta, \quad \alpha, \beta \in \mathbb{R}$.
Cuadrática inversa	$\frac{1}{1+z^2}$.
Thin-plate spline	$z^2 \log z$.
Trigonométricas	$\sin(z)$.

De todas las anteriores, las más utilizadas son las de tipo normal y las de tipo *thin-plate spline*. El ajuste del parámetro h se calculará una vez decididas las funciones de base radial y los centros para el problema observando la distribución de los datos creada tras aplicar las funciones.

Para obtener los centros el enfoque más habitual es el de las k-medias. Este algoritmo, aunque es utilizado en análisis de conglomerados (*clusters*), también resulta de interés para encontrar los centros de las redes de base radial. Para explicar este algoritmo, tomaremos un conjunto de datos de entrenamiento $\{\underline{x}_i\}_{i=1}^n$ sin considerar el valor de la variable de salida. El objetivo de las k-medias es dividir el conjunto en k conglomerados diferentes de centros $\{\underline{\mu}_j : j = 1, \ldots, k\}$. Para realizar dicha división el algoritmo resuelve el siguiente problema de optimización para encontrar los valores de los centros:

$$\text{mín} \sum_{j=1}^{k} S_j = \min_{\underline{\mu}} \sum_{j=1}^{k} \sum_{i=1}^{n} z_{j,i} |\underline{x}_i - \underline{\mu}_j|^2,$$

siendo $z_{j,i} = 1$ si $\underline{x}_i$ pertenece al conglomerado j y 0 en caso contrario. Bajo estas condiciones definimos:

$$\underline{\mu}_j = \frac{1}{n_j} \sum_{i=1}^{n} z_{j,i} x_i, \quad \forall j = 1, \ldots, k,$$

entendiendo n_j como el número de elementos contenidos en el conglomerado j, con $j = 1, \ldots, k$.

Falta considerar cuantos centros vamos a tomar, este número vendrá determinado en función al tipo de datos con el que estemos trabajando, al tamaño de la muestra, a la dimensión de los datos o la forma de distribuirse al ser proyectados por las funciones no lineales que tomemos. En general, es conveniente tener muchos centros en un modelo con baja complejidad que tener pocos centros y haber desarrollado una red compleja. Una posible solución es utilizar validación cruzada sobre varios números de centros, estudiar los errores considerando diversos parámetros de regularización y elegir aquel número de centros que no modifique demasiado el error generado en la validación cruzada.

Fijado m como el número de neuronas en la capa oculta, para poder calcular los pesos sinápticos vamos a tener en cuenta tres matrices diferentes, una matriz de datos $\mathbf{X}$ donde las columnas representan las realizaciones muestrales sin considerar la variable objetivo ($\dim(\mathbf{X}) = p \times n$), una segunda matriz denominada $\mathbf{P}$ donde cada fila i es el resultado de evaluar la función radial base ϕ_i a cada vector $\underline{x}$ del conjunto de entrenamiento, con $i = 1, \ldots, m$ ($\dim(\mathbf{P}) = m \times n$), la última matriz denotada $\mathbf{Y}$ es aquella que tiene por columnas los valores de la variable de salida ($\dim(\mathbf{Y}) = r \times n$). El objetivo es encontrar una matriz $\mathbf{W}$ cuyas columnas sean los pesos sinápticos de cada una de las neuronas de la capa oculta, es decir:

$$\mathbf{W} = \begin{pmatrix} w_{1,1} & \ldots & w_{1,r} \\ \vdots & \ddots & \vdots \\ w_{m,1} & \ldots & w_{m,r} \end{pmatrix}.$$

Ya que la propagación se realiza hacia delante, los valores de respuesta obtenidos por la red serán de la forma:

$$\widehat{\mathbf{Y}} = \mathbf{W}^t\mathbf{P},$$

en consecuencia, trataremos de minimizar los errores comparándolos con los originales de forma similar a la realizada en el Perceptrón multicapa, esto es:

$$E = \text{mín} \parallel \mathbf{Y} - \widehat{\mathbf{Y}} \parallel^2 = \min_{\mathbf{W}} \parallel \mathbf{Y} - \mathbf{W}^t\mathbf{P} \parallel^2 .$$

En este caso es posible realizar el método de mínimos cuadrados basándonos en la norma matricial de Frobenius, es decir:

$$\parallel \mathbf{Y}-\mathbf{W}^t\mathbf{P} \parallel^2 = \text{Traza}((\mathbf{Y}-\mathbf{W}^t\mathbf{P})^t(\mathbf{Y}-\mathbf{W}^t\mathbf{P})) = \text{Traza}(\mathbf{Y}^t\mathbf{Y}-\mathbf{Y}^t\mathbf{W}^t\mathbf{P}-\mathbf{P}^t\mathbf{W}\mathbf{Y}+\mathbf{P}^t\mathbf{W}\mathbf{W}^t\mathbf{P}),$$

que una vez desarrollado el producto vemos que en el problema de mínimos, el primer término $\mathbf{Y}^t\mathbf{Y}$ no depende de $\mathbf{W}$ y además la traza del segundo y el tercer termino son iguales ya que son matrices cuadradas y la traza de una matriz cuadrada es igual a la de su traspuesta. Por tanto el problema de optimización nos queda:

$$\min_{\mathbf{W}} \parallel \mathbf{Y} - \mathbf{W}^t\mathbf{P} \parallel^2 = \min_{\mathbf{W}} \text{Traza}(\mathbf{P}^t\mathbf{W}\mathbf{W}^t\mathbf{P} - 2\mathbf{P}^t\mathbf{W}\mathbf{Y}),$$

para encontrar ese mínimo vamos a derivar la expresión anterior obteniendo:

$$\frac{\partial(\text{Traza}(\mathbf{P}^t\mathbf{W}\mathbf{W}^t\mathbf{P}))}{\partial\mathbf{W}} - 2\frac{\partial(\text{Traza}(\mathbf{P}^t\mathbf{W}\mathbf{Y}))}{\partial\mathbf{W}} = 2\mathbf{P}\mathbf{P}^t\mathbf{W} - 2\mathbf{P}\mathbf{Y}^t,$$

que posteriormente si igualamos a 0 se llega a una solución para $\mathbf{W}$ de la forma:

$$\mathbf{W} = (\mathbf{P}\mathbf{P}^t)^{-1}\mathbf{P}\mathbf{Y}^t,$$

donde todos los datos son ahora conocidos y se puede dar un valor para cada uno de los pesos sinápticos una vez escogidas las funciones de base radial. En el caso en que las neuronas de la capa oculta tengan un umbral, basta con añadir una fila de unos al final de la matriz **P** y realizar el mismo proceso, en consecuencia **W** tendrá una fila más que indicará los umbrales. Con todos estos parámetros ya podemos crear un modelo de redes de base radial para clasificar datos.

Para ver un ejemplo de redes de base radial tomaremos una muestra de tamaño 2000 con 8 variables. Las 7 primeras representan la intensidad de una señal Wi-Fi[7] sobre un teléfono móvil. El objetivo está en localizar la posición de la persona que lleva el teléfono móvil con un total de 4 posibilidades. Para usar las redes de base radial necesitaremos las librerías de *R* que se exponen a continuación:

```
library(caret)
library(xtable)
library(biotools)
library(RSNNS)
wifi = read.table("wifi.txt")
colnames(wifi) = c(paste("X",1:7,sep=""),"Y")
set.seed(6567)
```

Una vez leídos los datos, se procede a separar en entrenamiento y prueba de forma aleatoria, además se separarán los valores de la variable de salida en otro vector aparte.

```
muestra = sample(1:2000,1500)
wifiE = wifi[muestra,1:7]
wifiT = wifi[-muestra,1:7]
clasesE = as.factor(wifi[muestra,8])
clasesT = as.factor(wifi[-muestra,8])
clase = decodeClassLabels(clasesE)
```

Al igual que en el ejemplo del Perceptrón multicapa, es necesario usar la función *decodeClassLabels* para crear una matriz con los datos de la variable objetivo, que se empleará para establecer el número de neuronas en la capa de salida (en este caso 4). El modelo viene dado por la función *rbf* de la librería RSNNS de *R*.

```
redbr25 = rbf(wifiE,clase,size=25,maxit = 1000)
```

A la hora de estimar los centros y los pesos sinápticos *R* utiliza un proceso iterativo, por lo que se especifica un número de iteraciones máximas además del número deseado de neuronas en la capa oculta. Por defecto, *R* selecciona funciones de tipo normal para la capa oculta. Veamos ahora el error de entrenamiento:

[7]Los datos del ejemplo están en https://archive.ics.uci.edu/ml/datasets/Wireless+Indoor+Localization

```
predE = predict(redbr25, wifiE)
print(xtable(confusionMatrix(clasesE, predE), digits = 0))
```

	1	2	3	4
1	352	0	1	1
2	35	368	33	19
3	1	6	332	0
4	1	0	7	344

Siendo el error obtenido de 0,069. Dado que el número de neuronas en la capa oculta es alto, es posible haber creado un problema de sobreajuste, por ello es importante observar el error de prueba.

```
predT = predict(redbr25, wifiT)
confusionMatrix(clasesT, predT)

Confusion Matrix and Statistics

    1   2   3   4
1 104   0   1   1
2   7 121  19   4
3   0   5 107   0
4   0   0   0 131

Overall Statistics

Accuracy : 0.926
95% CI : (0.8994, 0.9474)
No Information Rate : 0.272
P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.9012
Mcnemar's Test P-Value : NA

Statistics by Class:
                     Class: 1 Class: 2 Class: 3 Class: 4
Sensitivity            0.9369   0.9603   0.8425   0.9632
Specificity            0.9949   0.9198   0.9866   1.0000
Pos Pred Value         0.9811   0.8013   0.9554   1.0000
Neg Pred Value         0.9822   0.9857   0.9485   0.9864
Prevalence             0.2220   0.2520   0.2540   0.2720
Detection Rate         0.2080   0.2420   0.2140   0.2620
Detection Prevalence   0.2120   0.3020   0.2240   0.2620
Balanced Accuracy      0.9659   0.9401   0.9146   0.9816
```

El error es similar al de entrenamiento, por lo tanto, el modelo generaliza bien los datos que han sido introducidos. Las redes de base radial son modelos de redes neuronales artificiales que tienen relativa facilidad para estimar los parámetros necesarios sin que se convierta en algo demasiado complejo. Existen muchas variantes de este modelo debido al proceso de elección de centros, las funciones base deseadas o el procedimiento para estimar los parámetros.

3.6. Máquinas de vectores soporte no lineales

En el capítulo anterior, se hacía referencia al año 1992 cuando Vladimir Vapnik sugirió una forma para crear modelos con máquinas de vectores soporte cuyos límites de decisión fueran ahora no lineales. La idea que sostiene este clasificador es intentar proyectar los datos en una dimensión mayor donde poder clasificarlos usando un hiperplano de margen maximal como se realiza en el caso lineal. Esta proyección de los datos está basada en el uso de funciones *kernel* que se definen así:

$$\begin{aligned} K : \mathbb{R}^p \times \mathbb{R}^p &\longrightarrow \mathbb{R} \\ (\underline{x}, \underline{y}) &\longrightarrow K(\underline{x}, \underline{y}) = \langle \phi(\underline{x}), \phi(\underline{y}) \rangle_{\mathcal{V}} = \phi^t(\underline{x})\phi(\underline{y}), \end{aligned}$$

es decir, la función K debe definir un producto escalar en un espacio de Hilbert $\mathcal{V}$ mientras que la función ϕ es la encargada de proyectar los datos en $\mathbb{R}^m$. Como ejemplo, en el caso lineal $\phi(\underline{x}) = \underline{x}$, sin embargo en los casos no lineales no siempre será posible dar una expresión explícita de ϕ. Las funciones *kernel* además han de cumplir la condición de Mercer que es:

$$\int_{\mathbb{R}^p \times \mathbb{R}^p} K(\underline{x}, \underline{y}) f(\underline{x}) f(\underline{y}) d\underline{x} d\underline{y} \geq 0 \quad \forall f \in L^2(\mathbb{R}^p),$$

donde $L^2(\mathbb{R}^p)$ denota las funciones de cuadrado integrables en $\mathbb{R}^p$. La elección de estas funciones no es única, al igual que ocurría con las funciones de base radial, existen varios ejemplos de funciones *kernel*, a continuación se dan algunos de ellos:

Funciones	$K(\underline{x}, \underline{y})$.
Polinomial	$(1 + \underline{x}^t \underline{y})^d, \quad d \geq 2, \quad d \in \mathbb{Z}.$
Normal	$e^{-\frac{\lvert \underline{x} - \underline{y} \rvert^2}{\sigma^2}}, \quad \sigma \in \mathbb{R}.$
Sigmoide	$\tanh(\underline{x}^t \underline{y} - c), \quad c > 0.$

El proceso para clasificar datos veremos que no tiene diferencias significativas con el caso lineal detallado en el capítulo anterior. Consideramos un conjunto de datos $\{\underline{x}_i, y_i\}_{i=1}^n$ en un caso binario ($\mathcal{Y} = \{-1, 1\}$) y una función *kernel* K, la proyección a $\mathbb{R}^m$ de los datos se realiza a partir de la función ϕ. Suponemos ahora que una vez proyectados por ϕ en $\mathbb{R}^m$ buscamos un hiperplano de coeficientes $\underline{\beta} \in \mathbb{R}^m, \beta_0 \in \mathbb{R}$ tal que:

$$y_i(\underline{\beta}^t \phi(\underline{x}_i) + \beta_0) > 0, \quad \forall i = 1, \ldots, n,$$

y la regla de clasificación viene dada por:

$$h(\underline{x}) = \begin{cases} 1 & \text{si} \quad \beta_0 + \underline{\beta}^t \phi(\underline{x}) > 0, \\ -1 & \text{si} \quad \beta_0 + \underline{\beta}^t \phi(\underline{x}) < 0, \end{cases}$$

Para encontrar el hiperplano de margen maximal, concluíamos que, la mejor forma era resolver la forma dual del problema de Lagrange donde, tras haber proyectado los datos, tiene ahora la forma:

$$F_d(\underline{\alpha}) = \sum_{k=1}^{n} \alpha_k - \frac{1}{2} \sum_{k=1}^{n} \sum_{j=1}^{n} \alpha_k \alpha_j y_k y_j \phi^t(\underline{x}_k) \phi(\underline{x}_j), \quad 0 \leq \alpha_i \leq C, \quad \forall i = 1, \ldots, n,$$

donde C es la constante que indicaba el coste de errar una clasificación en el caso en que las clases no sean linealmente separables en $\mathbb{R}^m$. La resolución del problema para maximizar $\underline{\alpha}$ sigue exactamente los mismos pasos pues sigue tratándose de un problema de programación cuadrática para $\underline{\alpha}$. Para llegar a esta expresión, se procedía a partir de las derivadas parciales del problema primal donde se daba una expresión de la solución para $\underline{\beta}$ una vez obtenido el valor de $\underline{\alpha}$ que tras aplicar ϕ queda:

$$\underline{\beta} = \sum_{\underline{x}_i \in V_s} \alpha_i y_i \phi(\underline{x}_i),$$

V_s denotaba el conjunto de vectores soporte. Se recuerda que $\alpha_i = 0$ si $\underline{x}_i$ no es un vector soporte, con $i = 1, \ldots, n$. Aunque esta expresión no da un valor para $\underline{\beta}$ si no se conoce ϕ, la regla de clasificación no se ve afectada pues la función discriminante g puede definirse:

$$g(\underline{x}_j) = \beta_0 + \underline{\beta}^t \phi(\underline{x}_j) = \sum_{\underline{x}_i \in V_s} \alpha_i y_i \phi(\underline{x}_i)^t \phi(\underline{x}_j) + \beta_0 = \sum_{\underline{x}_i \in V_s} \alpha_i y_i K(\underline{x}_i, \underline{x}_j) + \beta_0, \quad \forall j = 1, \ldots, n,$$

donde β_0 resulta ahora:

$$\beta_0 = \frac{1}{n_{V_s}} \sum_{\underline{x}_i \in V_s} y_i - \sum_{\underline{x}_i \in V_s} \sum_{\underline{x}_j \in V_s} \alpha_i y_i K(\underline{x}_i, \underline{x}_j))$$

Para resolver los problemas de clasificación múltiple, si se determina la opción de clasificar mediante una clase contra todas, el procedimiento es análogo a lo expresado anteriormente repitiendo el proceso para las c clases diferentes. De igual forma, si se decide afrontar el problema mediante el enfoque de uno contra uno, la única diferencia reside en construir $\binom{c}{2}$ hiperplanos en $\mathbb{R}^m$ para clasificar los datos. Aunque se trabaje con hiperplanos hay que tener en cuenta que los límites de decisión en el espacio original son no lineales en general.

Para el siguiente ejemplo en R vamos a tomar una muestra de tamaño 51490 que corresponden a partidas realizadas en un videojuego[8]. Durante una partida dos equipos de 5 jugadores escogen sus respectivos personajes sin posibilidad de repetir ninguno y con restricciones impuestas previamente. En este conjunto de datos nos encontramos 61 variables de las cuales trabajaremos con 58 (variable objetivo incluida), algunas de ellas pertenecen a situaciones que se dan antes de empezar la partida y otras referidas a objetivos conseguidos durante la misma. El objetivo consistirá en clasificar que equipo ganó la partida (equipo 1 ó equipo 2) en base a la información de las variables anteriores.

Necesitaremos la librería *kernlab* de R ya que vamos a usar la misma función que en el ejemplo del capítulo anterior. Procedemos a la lectura de datos y la eliminación de variables que no interesan.

```
library(caret)
library(xtable)
library(biotools)
library(kernlab)

loldatos = read.csv("games.csv",header = TRUE,sep=",")
loldatos = loldatos[,-4]
loldatos = loldatos[,-2]
loldatos = loldatos[,-1]
```

Dividimos los datos en entrenamiento y prueba:

```
set.seed(75675)
muestra = sample(1:51490,40000)
lolE = loldatos[muestra,]
lolclasesE = as.factor(lolE[,2])
lolT = loldatos[-muestra,]

lolclasesT = as.factor(lolT[,2])
lolE[,2] = as.factor(lolE[,2])
```

Creamos el modelo indicando la función *kernel*, en este caso tomaremos un modelo basado en funciones polinómicas.

```
lolsvm = ksvm(lolE$winner ~ .,lolE,kernel = "polydot")
```

Por defecto, R toma $C = 1$ como coste de clasificación, este valor se puede modificar en los argumentos de entrada de la función *ksvm*. El error de entrenamiento generado es:

```
predE = predict(lolsvm,lolE[,-2])
print(xtable(confusionmatrix(predE,lolE[,2]),digits = 0))
```

[8]Datos en: https://www.kaggle.com/datasnaek/league-of-legends

	new 1	new 2
1	19642	855
2	623	18880

El error en los datos de entrenamiento es 0,03695, para terminar de evaluar observamos el error en los datos de prueba.

```
predT = predict(lolsvm, lolT[,-2])
confusionMatrix(predT, lolclasesT)

Confusion Matrix and Statistics

     1     2
1 5637   269
2  175  5409

Accuracy : 0.9614
95% CI : (0.9577, 0.9648)
No Information Rate : 0.5058
P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.9227
Mcnemar's Test P-Value : 1.017e-05

Sensitivity : 0.9699
Specificity : 0.9526
Pos Pred Value : 0.9545
Neg Pred Value : 0.9687
Prevalence : 0.5058
Detection Rate : 0.4906
Detection Prevalence : 0.5140
Balanced Accuracy : 0.9613

'Positive' Class : 1
```

El recurso de poder emplear funciones *kernel* es otra de las razones por las que este clasificador es empleado hoy en día. Estas funciones dotan a las máquinas de vectores soporte de más versatilidad para elegir modelos y evaluar cual es más adecuado.

Capítulo 4

Árboles de clasificación

Los árboles de clasificación suponen un enfoque diferente dentro de todas las técnicas vistas hasta ahora. Esta nueva manera de abordar un problema de clasificación se basa en encontrar una serie de reglas para dividir el conjunto de datos y poder clasificar pudiendo ser estas reglas representadas en un diagrama de árbol. En lo sucesivo detallaremos el proceso de construcción de árboles binarios.

Esta forma de clasificar datos es fácilmente interpretable al poder ser representado en un gráfico observando las variables que más importancia tienen y además puede emplearse en una gran variedad de problemas. Por otro lado existen técnicas ya expuestas en este trabajo que pueden obtener mejores resultados clasificando datos. Sin embargo esta diferencia podría no ser significativa si tenemos en cuenta las otras cualidades.

Vamos a introducir la notación que usaremos para los árboles. Definimos el árbol T como un conjunto de nodos representados por números naturales. De cada nodo $t \in T$ se pueden generar o bien dos nodos (derecho e izquierdo) o bien ninguno y en cuyo caso t sería una hoja. Será también de interés el conjunto T^* que representará el número de hojas de T. Dado que T genera $|T^*|$ regiones, estas serán representadas por $R_1, \ldots, R_{|T^*|}$ con $R_i \cap R_j = \emptyset$, $\forall i, j = 1, \ldots, |T^*|, i \neq j$. Para realizar la clasificación usaremos los siguientes valores:

$$\hat{p}_{t,y} = \frac{n_{t,y}}{n_t}, \quad \forall t = 1, \ldots, |T^*|, \quad \forall y \in \mathcal{Y},$$

siendo n_t el número de vectores contenidos en R_t y $n_{t,y}$ son aquellos vectores de la clase y en la región R_t. La regla de clasificación que siguen estos árboles consiste en asignar un vector $\underline{x} \in R_t$ a la clase y^* si:

$$\hat{p}_{t,y^*} = \max_{y \in \mathcal{Y}} \hat{p}_{t,y}, \quad \forall t = 1, \ldots, |T^*|,$$

con esta regla deducimos el error de entrenamiento para cada nodo:

$$E_t(T) = 1 - \max_{y \in \mathcal{Y}} \hat{p}_{t,y} \quad \forall t = 1, \ldots, |T^*|.$$

En ocasiones, el error de entrenamiento no es suficiente para determinar si el árbol realiza correctamente el proceso de clasificación, para ello se introduce el índice de Gini que se define:

$$G_t(T) = \sum_{y \in \mathcal{Y}} \hat{p}_{t,y}(1 - \hat{p}_{t,y}), \quad \forall t = 1, \ldots, |T^*|.$$

El índice de Gini especifica si existe una clase predominante en la región creada por la hoja t. Buscamos por tanto, valores pequeños del índice de Gini en todas las regiones en las que se ha dividido el espacio. Otro enfoque similar con este fin es la entropía cruzada dada por:

$$EC_t(T) = -\sum_{y \in \mathcal{Y}} \hat{p}_{t,y} \log(\hat{p}_{t,y}), \quad \forall t = 1, \ldots, |T^*|.$$

A la hora de crear un árbol de clasificación hay que tener en cuenta que no debe ser ni muy profundo, pues puede acarrear sobreajuste en el modelo, ni muy básico, porque el resultado podría dar lugar a malas clasificaciones en los datos de prueba.

Una regla de división consiste en una condición para todos los vectores $\underline{x}$ de forma que el espacio quede dividido en dos regiones que llamaremos R_1 y R_2 definidas:

$$R_1 = \{\underline{x} \in \mathbb{R}^p : f(\underline{x}) \leq s,\ s \in \mathbb{R}\}, \quad R_2 = \{\underline{x} \in \mathbb{R}^p : f(\underline{x}) \geq s,\ s \in \mathbb{R}\},$$

con $f(\underline{x})$ una función que puede ser lineal o no. La definición anterior tiene un punto de vista general, habitualmente y en lo que sigue se utilizarán funciones lineales que dependan de una sola variable para construir las regiones anteriores. Es decir, buscamos una variable X_j con $j = 1, \ldots, p$ y una constante s que consigan:

$$\min_{j,s}(G_1(T) + G_2(T)),$$

con G_1 y G_2 los índices de Gini para R_1 y R_2 respectivamente. De igual forma se puede utilizar cualquiera de las tres medidas de error que se han mencionado anteriormente y el problema de optimización seguiría con la misma estructura.

Existen diversas formas de resolver el problema anterior, usualmente se eligen unos conjuntos finitos I_j de valores para s dentro del rango de posibles valores de cada variable X_j. Estos conjuntos no deben ser excesivamente grande para no aumentar el coste computacional del modelo. Una vez obtenidos los conjuntos se procede a buscar la variable X_j realizando una búsqueda exhaustiva y probando una a una las variables con cada s, finalizando con el mínimo para cada $s \in I_j$ y para cada $j = 1, \ldots, p$. Para elegir el corte, se toman la variable X_j y el valor s que resulte el mínimo de todos los mínimos anteriores. En caso de empate elegir cualquiera de los dos o más cortes.

Con este criterio ya podemos hacer crecer un árbol de manera indefinida usando este argumento recursivamente, ahora tenemos que establecer cuando parar. Uno de los argumentos más comunes para parar de ramificar es la siguiente:

$$|\sum_{t_1 \in |T_1^*|} G_{t_1} - \sum_{t_0 \in |T_0^*|} G_{t_0}| < \epsilon, \quad \text{para algún} \quad \epsilon > 0,$$

es decir, establecer un umbral para el cual la diferencia de las sumas de los índices de Gini entre un árbol T_0 y un subárbol T_1 que tenga una división menos que T_0 sea menor que dicho umbral. Otra opción es definir un número mínimo de vectores por región construida y parar cuando la próxima división implique que alguna de las nuevas regiones tenga menos vectores de los estipulados.

El árbol generado por el proceso anterior puede sobreajustar el modelo, por ello se usará el algoritmo *pruning*. Para este algoritmo se necesita construir un árbol con mucha profundidad, a partir de ahí se seleccionan aquellos nodos terminales (hojas) que aporten poca mejora al error de clasificación y se van eliminando recursivamente con el objetivo de hacer mínimo el coste de complejidad de un árbol T:

$$\min_{|T^*|} C_\alpha(T) = \min_{|T^*|} \sum_{t \in T^*} C_\alpha(t) = \min_{|T^*|} \sum_{t \in T^*} n_t Q_t(T) + \alpha |T^*|,$$

donde $\alpha \geq 0$ es una constante que puede ser interpretada como el coste de complejidad del árbol. Si el valor de α es pequeño significara que no hay una penalización fuerte para un árbol con muchos nodos, por el contrario, si el valor de α es grande significará que el árbol que minimice la expresión anterior tendrá pocos nodos. Por otro lado, $Q_t(T)$ puede estar representado por el error de clasificación, por el índice de Gini o por la entropía cruzada.

Notar que si el número de hojas disminuye entonces el valor de la suma $\sum_{t \in T^*} n_t Q_t(T)$ aumenta, en consecuencia buscaremos un nodo t no terminal cuya contribución a la suma anterior sea mínima.

Sea T_t un subárbol de T cuya raíz es el nodo $t \in T$, definimos la siguiente función que depende del nodo escogido:

$$f(t) = \frac{C(t) - C(T_t)}{|T_t^*| - 1},$$

con $C(t) = n_t Q_t(T)$ y $C(T_t) = \sum_{r \in T_t^*} n_r Q_r(T)$, $\forall t \in T$. Esta función servirá para cuantificar la aportación del nodo t al descenso del error de clasificación. Esta medida será calculada a todos los nodos no terminales. Una vez construido el árbol se observará cual es el nodo con menor valor en f y se procederá a eliminar los nodos que derivan de ahí. Este proceso puede realizarse iterativamente hasta llegar a un árbol trivial, posteriormente se evalúa el coste de complejidad en cada iteración y se elige el árbol que alcance el mínimo.

Para ver su implementación en R tomamos una muestra de 1728 coches[1] que se pretenden evaluar en base a los siguientes atributos:

- $X_1 \equiv$ Precio de Compra.
- $X_2 \equiv$ Coste de mantenimiento.
- $X_3 \equiv$ Número de puertas.
- $X_4 \equiv$ Número de personas.
- $X_5 \equiv$ Tamaño del maletero.
- $X_6 \equiv$ Seguridad.
- $Y \equiv$ Decisión (inaceptable, aceptable, bueno o muy bueno).

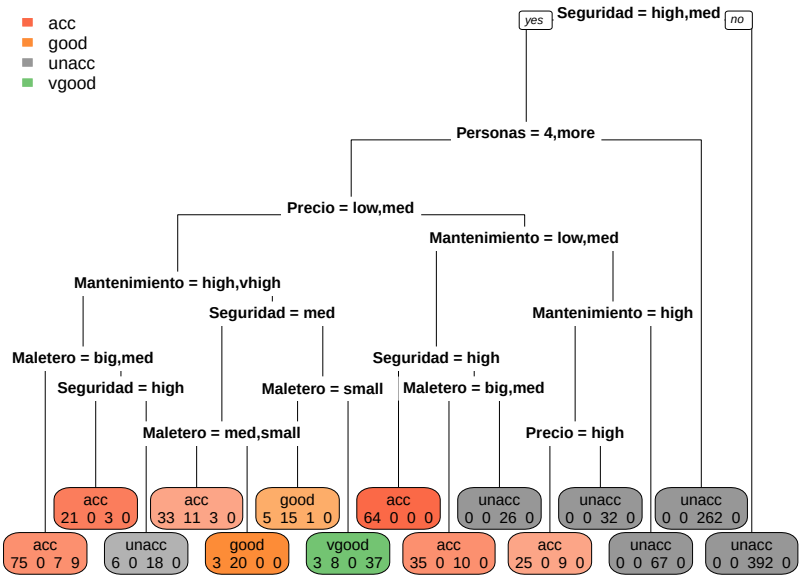
Serán necesarias las librerías de R *rpart* y *rpart.plot* para poder usar este clasificador y realizar una representación gráfica. Procedemos a la lectura de los datos y a dividir en los conjuntos de entrenamiento y prueba.

```
coche = read.table("coche.txt",sep = ",")
colnames(coche) = c("Precio","Mantenimiento","Puertas","Personas",
                    "Maletero","Seguridad","Decision")
set.seed (3245)
muestra2 = sample(1:1728,1200)
entreno2 = coche[muestra2,]
test2 = coche[-muestra2,]
```

Para crear el árbol de clasificación usamos la función *rpart* y le damos un valor a α de 0,01 para la realización del clasificador. A continuación, representamos gráficamente el árbol obtenido.

```
arbolcoche = rpart(entreno2$Decision ~ . ,entreno2 , method = "class",
control = rpart.control(cp = 0.01))
rpart.plot(arbolcoche,tweak = 1,extra = 1,type = 0,legend.y = 1)
```

[1]Los datos se encuentran en https://archive.ics.uci.edu/ml/datasets/car+evaluation



A vista del gráfico se pueden sacar conclusiones como: si un coche no tiene una seguridad alta o media o no tiene capacidad para más de 4 personas es inaceptable. También se puede observar que la variable que equivale al número de puertas no es decisiva para clasificar. Pasamos a estudiar el error de entrenamiento.

```
predE2 = predict(arbolcoche, entreno2, type= "class")
print(xtable(confusionmatrix(predE2, entreno2$Decision), digits=0))
```

	Aceptable	Bueno	Inaceptable	Muy bueno
Aceptable	253	11	32	9
Bueno	8	35	1	0
Inaceptable	6	0	797	0
Muy Bueno	3	8	0	37

El error es de 0.065. Veamos ahora el error de prueba para terminar de evaluar este ejemplo.

```
predT2 = predict(arbolcoche, test2, type ="class")
confusionMatrix(predT2, test2$Decision)
Confusion Matrix and Statistics
```

	acc	good	unacc	vgood
acc	106	4	13	4
good	7	10	3	0
unacc	1	0	364	0
vgood	0	1	0	15

```
Overall Statistics

Accuracy : 0.9375
95% CI : (0.9133, 0.9566)
No Information Rate : 0.7197
P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.8606
Mcnemar's Test P-Value : NA

Statistics by Class:
```

	acc	good	unacc	vgood
Sensitivity	0.9298	0.66667	0.9579	0.78947
Specificity	0.9493	0.98051	0.9932	0.99804
Pos Pred Value	0.8346	0.50000	0.9973	0.93750
Neg Pred Value	0.9800	0.99016	0.9018	0.99219
Prevalence	0.2159	0.02841	0.7197	0.03598
Detection Rate	0.2008	0.01894	0.6894	0.02841
Detection Prevalence	0.2405	0.03788	0.6913	0.03030
Balanced Accuracy	0.9395	0.82359	0.9756	0.89375

Aunque en general este clasificador detecta correctamente las clases, no realiza buenas clasificaciones en la clase "Bueno". Esto puede deberse a que se dispone de pocas muestras con esa clase.

Uno de los principales problemas de los árboles de clasificación es su alta variabilidad con los datos, esto es, si se añaden nuevos datos (o se emplean datos diferentes) de entrenamiento el resultado del árbol puede ser completamente diferente. Los próximos algoritmos que vamos a detallar tratarán de reducir esta variabilidad y consistirán en el uso de más de un árbol para clasificar los datos.

4.1. Bagging

El algoritmo Bagging fue propuesto por Leo Breiman en 1994 para mejorar la clasificación de datos usando técnicas de remuestreo y combinando varios clasificadores en el proceso. Aunque estos clasificadores no tienen que ser necesariamente árboles (pueden ser de cualquier tipo), es aplicado comúnmente usando varios árboles de clasificación.

El nombre de Bagging viene del acrónimo bootstrap aggregating. Bootstrap es una técnica que selecciona a partir de n datos de entrenamiento, un número m de estos datos tomándolos aleatoriamente y con reemplazamiento para crear otro conjunto diferente.

El proceso que sigue el Bagging consiste en utilizar la técnica de remuestreo bootstrap k veces sobre un conjunto de datos de entrenamiento y con cada una de las muestras se genera una matriz de datos $\mathbf{X}_i$ con $i = 1, \ldots, k$, que será la empleada para elaborar un clasificador diferente usando los datos obtenidos del remuestreo. Una vez creados los k clasificadores diferentes, la regla de clasificación del Bagging consiste en asignar un vector $\underline{x}$ a la clase más votada, es decir a la clase que haya salido más veces como respuesta de los k clasificadores anteriores. En caso de empate elegir aleatoriamente entre cualquiera de las clases empatadas.

Para el caso particular en el que se empleen árboles de clasificación para crear el Bagging, estos árboles serán construidos mediante alguna regla de división y una vez que se decide cuando parar de dividir no se aplicará el algoritmo *prunning* para eliminar nodos, si no que se considerará el árbol resultante. Gracias a la reducción de variabilidad, incluso si se emplea un gran número de árboles, el Bagging no crea sobreajuste en el modelo.

Para ver si se mejora con respecto al uso de un solo árbol, el ejemplo de R estará basado en el mismo conjunto de datos que en el ejemplo anterior. Para la implementación de este algoritmo usamos la librería de R *ipred*. A fin de comparar los resultados vamos a dividir los conjuntos de datos de la misma manera.

```
library(caret)
library(biotools)
library(xtable)
library(ipred)

coche = read.table("coche.txt",sep = ",")
colnames(coche) = c("Precio","Mantenimiento","Puertas","Personas",
                    "Maletero","Seguridad","Decision")
set.seed (3245)
muestra2 = sample(1:1728,1200)
entreno2 = coche[muestra2,]
test2 = coche[-muestra2,]
```

Vamos a crear un modelo compuesto por 40 árboles de clasificación con la función *bagging* y vamos a evaluar los errores, comenzando con el de entrenamiento.

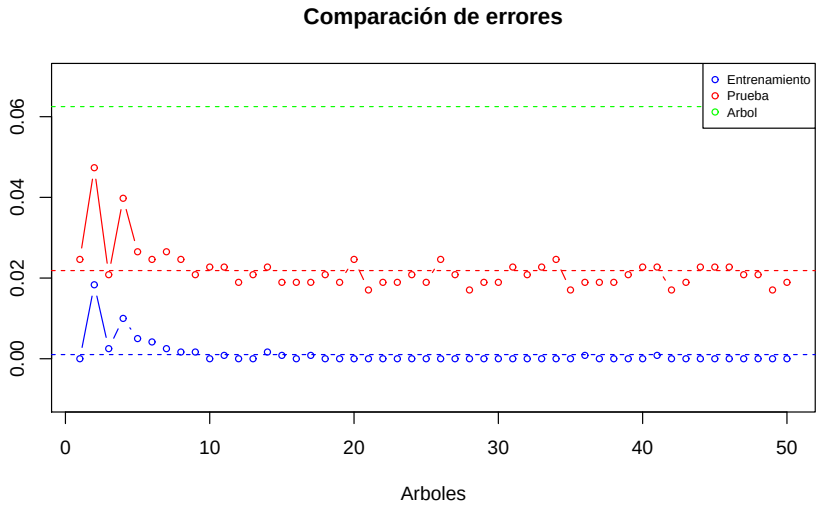
```
bagcoche40 = bagging(entreno2$Decision ~ ., entreno2,nbagg=40)
predE40 = predict(bagcoche40,entreno2[,-7])
print(xtable(confusionmatrix(predE40,entreno2$Decision),digits = 0))
```

	Aceptable	Bueno	Inaceptable	Muy Bueno
Aceptable	270	0	0	0
Bueno	0	54	0	0
Inaceptable	0	0	830	0
Muy Bueno	0	0	0	46

Vemos que ahora con la misma muestra no comete ningún error en el conjunto entrenamiento usando 40 árboles de clasificación. Con respecto al error de prueba tenemos:

```
predT40 = predict(bagcoche40,test2[,-7])
confusionMatrix(predT40,test2$Decision)
Confusion Matrix and Statistics

          acc good unacc vgood
acc       112    1     5     0
good        0   14     2     0
unacc       1    0   373     0
vgood       1    0     0    19

Overall Statistics

Accuracy : 0.9811
95% CI : (0.9654, 0.9909)
No Information Rate : 0.7197
P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.9569
Mcnemar's Test P-Value : NA

Statistics by Class:

                     acc      good     unacc     vgood
Sensitivity       0.9825   0.93333    0.9816   1.00000
Specificity       0.9855   0.99610    0.9932   0.99804
Pos Pred Value    0.9492   0.87500    0.9973   0.95000
Neg Pred Value    0.9951   0.99805    0.9545   1.00000
Prevalence        0.2159   0.02841    0.7197   0.03598
Detection Rate    0.2121   0.02652    0.7064   0.03598
Detection Prevalence 0.2235 0.03030  0.7083   0.03788
Balanced Accuracy 0.9840   0.96472    0.9874   0.99902
```

Donde también vemos que mejora los resultados con respecto al ejemplo con un único árbol. A continuación podemos ver un gráfico donde se ven los errores usando cantidades diferentes de árboles de clasificación.



Vemos que el número de árboles va reduciendo el error de entrenamiento hasta 0 mientras que el error de prueba se estabiliza en torno a la media, más arriba en verde se observa el error de prueba empleando un solo árbol del ejemplo anterior.

El uso de más de un árbol hace que ya no sea tan fácil sacar conclusiones con respecto a las variables, además, también lleva a que sea un algoritmo más costoso computacionalmente.

4.2. Random Forest

Los Random Forest datan del año 1995 y fue propuesto en primera instancia por Ho. Este algoritmo, al igual que el Bagging, también consiste en una colección de árboles de clasificación donde cada uno de ellos emite un voto para cada vector $\underline{x}$ del conjunto de entrenamiento y la clase más votada es la asignada.

Para construir cada árbol se usará el bootstrap para remuestrear los datos y realizar así el clasificador en cada caso. La principal diferencia entre el Bagging y el Random Forest reside en que para cada árbol se elige aleatoriamente un subvector de $\underline{X}$ que denotaremos por Θ y la construcción de dicho árbol se realiza en base únicamente a las variables contenidas en Θ.

Para cada vector $\underline{x}$ del conjunto de entrenamiento tenemos k respuestas diferentes que provienen de los árboles de clasificación $\{h_i(\underline{x}, \Theta_i)\}_{i=1}^k$. Los elementos contenidos en $\{\Theta_i\}_{i=1}^k$ corresponden a vectores aleatorios independientes que forman una muestra aleatoria diferente. La composición de las variables contenidas en Θ_i es elegida aleatoriamente para cada $i = 1, \ldots, k$. Todos los árboles se construyen con mucha profundidad y sin *pruning*.

Con este conjunto de clasificadores podemos definir la función margen según la distribución del vector aleatorio $(\underline{X}, Y)$ como sigue:

$$mg(\underline{X}, Y) = \frac{1}{k}\sum_{i=1}^{k} I(h_i(\underline{X}, \Theta_i) = Y) - \max_{y \in \mathcal{Y}, y \neq Y} \frac{1}{k}\sum_{i=1}^{k} I(h_i(\underline{X}, \Theta_i) = y),$$

esta función nos mide la diferencia en media que excede la clase real con respecto a la segunda más votada. Cuanto mayor sea esta diferencia, más seguridad tendremos al clasificar el dato en cuestión. Usando esta función definimos el error generalizado:

$$EG = \mathbb{P}_{(\underline{X},Y)}(mg(\underline{X}, Y) < 0),$$

el subíndice $(\underline{X}, Y)$ denota que la probabilidad está tomada sobre el espacio de $(\underline{X}, Y)$. Para un número grande de árboles, gracias a la ley fuerte de los grandes números y las propiedades de los árboles de clasificación es posible probar que el error generalizado converge de manera casi segura a una probabilidad que depende de los datos considerados (vease el teorema 1.2 de [5]). Este resultado explica por qué los Random Forest no se sobreajustan si se añaden más árboles de clasificación.

Tanto en Bagging como en Random Forest existe una estimación del error conocida como *out-of-bag* (oob). Cada árbol $h_1, \ldots, h_k$, es construido por los conjuntos de entrenamiento $T_1, \ldots, T_k$ respectivamente formados a partir de aplicar bootstrap a un conjunto T. Si un vector $(\underline{x}, y) \in T$ no apareciera en alguno de los conjuntos $T_1, \ldots, T_k$ se emite el voto que resultaría en su respectivo clasificador y se evalúan los fallos cometidos, el error oob es la media de todas esas predicciones que han sido mal clasificadas.

Este error es usado en los Random Forests para construir una medida de importancia en las variables. Cuando el i-ésimo árbol es construido las muestras oob son transmitidas al árbol y se guardan las predicciones obtenidas. El siguiente paso consiste en permutar aleatoriamente los valores de la muestra oob de una variable X_j cualquiera y comprobar los resultados obtenidos. Cuanto mayor sea la media de los cambios producidos en todos los árboles mayor será la importancia de la variable X_j dentro del modelo, con $j = 1, \ldots, p$.

Para el siguiente ejemplo, vamos a crear un modelo que prediga el rango del precio de un teléfono móvil[2] en función a las prestaciones que ofrece. En total se tiene una muestra de tamaño 2000 con 20 variables diferentes más una variable objetivo con cuatro categorías numeradas de $0 - 3$ que indican el rango de precio de manera ascendente.

[2]Los datos se encuentran en https://www.kaggle.com/iabhishekofficial/mobile-price-classification

Para usar un este algoritmo necesitaremos la librería de *R randomForest*. Procedemos a leer los datos.

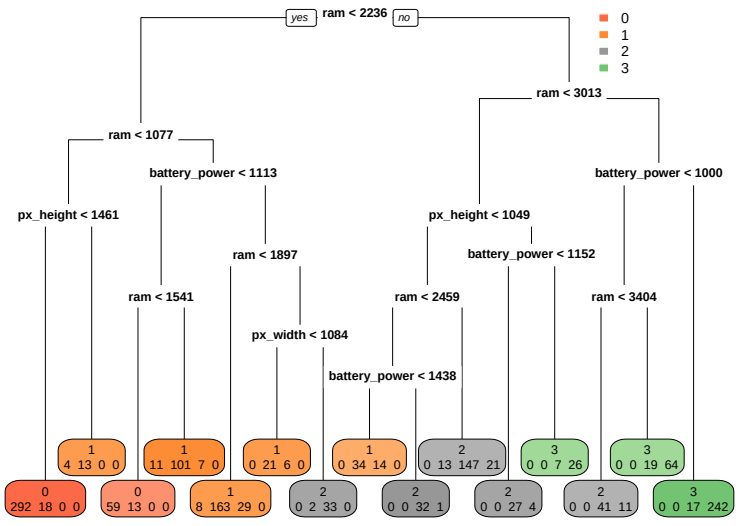
```
library(caret)
library(biotools)
library(xtable)
library(rpart)
library(rpart.plot)
library(randomForest)
movil = read.csv("movilE.csv",header = TRUE, sep=",")
```

Antes de seguir, los datos contienen siete variables (contando la variable objetivo) que son categóricas y *R* no las interpreta como tal, por lo tanto antes de dividir en conjuntos de entrenamiento y prueba, cambiamos el tipo de dichas variables.

```
cat = c(2,4,6,18,19,20,21)
for (i in cat){
movil[,i] = as.factor(movil[,i])
}
set.seed(575658)
muestra = sample(1:2000,1500)
movilE = movil[muestra,]
movilT = movil[-muestra,]
```

Para poder comparar la mejora de los resultados, realizamos un modelo con un sólo árbol.

```
arbolmovil = rpart(movilE$price_range ~ .,movilE,method="class",
control = rpart.control(cp = 0.008))
rpart.plot(arbolmovil,tweak = 1,extra = 1,type = 0,legend.y = 1)
```

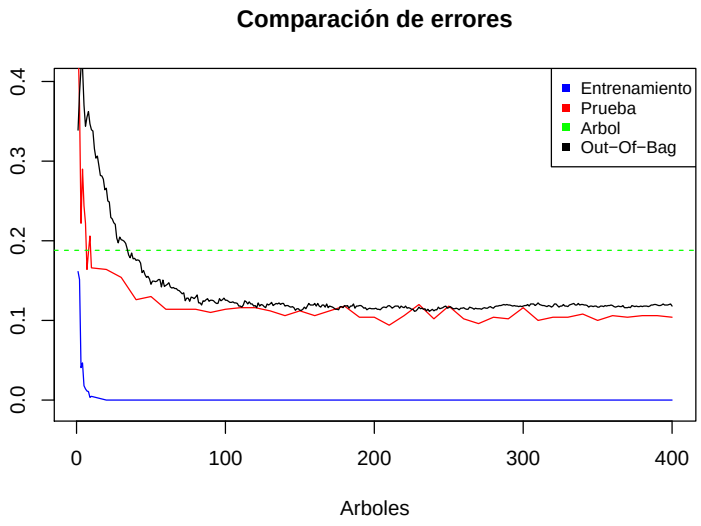


Aquí podemos ver que las variables más decisivas a la hora de tomar una decisión son: la memoria RAM que posee el teléfono y la batería. Además solo aparecen en el árbol un total de 4 variables distintas para la toma de decisión. El error de prueba de este modelo es de 0,188.

Construimos ahora el modelo con la función *randomForest*. Se realizará un bucle para poder observar los errores que se producen si el número de árboles aumenta en el modelo.

```
v = 1:40
v = c(1:9,10*v)
errorE = NULL
errorT = NULL
for (i in v) {
rfmovil = randomForest(movilE$price_range ~ .,movilE,ntree = i)
predRF = predict(rfmovil,movilE[,-21],type = "class")
em = confusionmatrix(predRF,movilE[,21]
e = sum(diag(em))/sum(em)
predRFT = predict(rfmovil,movilT[,-21],type = "class")
tm = confusionmatrix(predRFT,movilT[,21]
t = sum(diag(tm))/sum(tm)
errorE = cbind(errorE,e)
errorT = cbind(errorT,t)
}
errorE = 1-errorE
errorT = 1-errorT
```

Aquí podemos ver el gráfico correspondiente a los errores producidos en función al número de árboles contenidos en el Random Forest. También se detallan los errores oob para la última iteración.



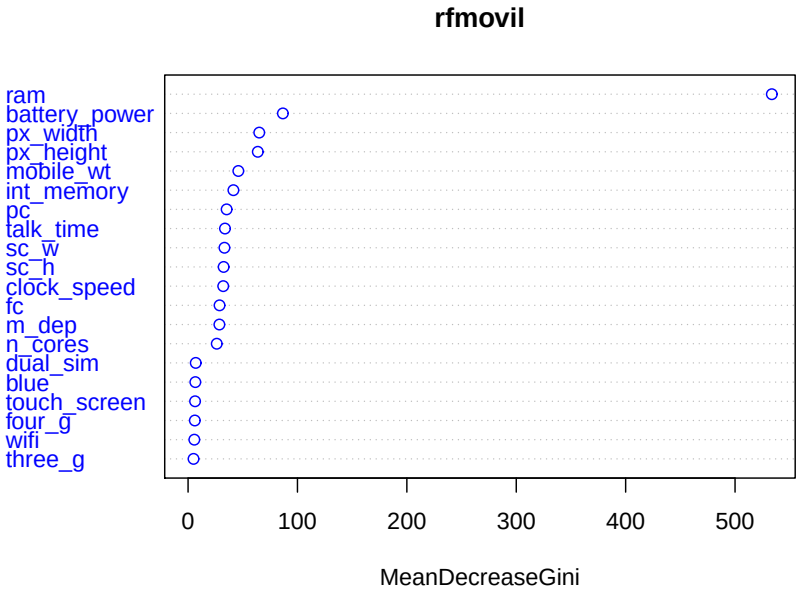
El modelo termina de aprender todos los datos de entrenamiento usando pocos árboles, puede apreciarse que el error de prueba es similar al error oob y que ambos se estabilizan a partir de cierto punto sin que el modelo se sobreajuste. Exponemos ahora con más detalle el error de prueba para en el modelo formado por 400 árboles.

```
Confusion Matrix and Statistics

    0   1   2   3
0 120   8   0   0
1   6 107  18   0
2   0   7  97   4
3   0   0   6 127

Overall Statistics

Accuracy : 0.902
95% CI : (0.8725, 0.9266)
No Information Rate : 0.262
P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.8693
Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2 Class: 3
Sensitivity            0.9524   0.8770   0.8017   0.9695
Specificity            0.9786   0.9365   0.9710   0.9837
Pos Pred Value         0.9375   0.8168   0.8981   0.9549
Neg Pred Value         0.9839   0.9593   0.9388   0.9891
Prevalence             0.2520   0.2440   0.2420   0.2620
Detection Rate         0.2400   0.2140   0.1940   0.2540
Detection Prevalence   0.2560   0.2620   0.2160   0.2660
Balanced Accuracy      0.9655   0.9068   0.8863   0.9766
```

Usando más árboles se ha conseguido una mejora en la clasificación de los datos de un 9 % aproximadamente.

Con la función *varImpPlot* obtenemos un gráfico con la importancia de las variables para este último modelo. Para medir la importancia se ha usado el decrecimiento medio del índice de Gini en los árboles considerados, a mayor decrecimiento se obtiene mayor importancia.

```
varImpPlot(rfmovil, col = "blue")
```



La variable referida a la memoria RAM del teléfono es claramente la más importante de todas seguida por la batería y la anchura y altura de la resolución en píxeles, que eran las cuatro que aparecían en el árbol inicial.

Para ver que variables aparecen seleccionadas en cada modelo se puede usar la función *varUsed*. Por defecto R selecciona siempre un número próximo a $\sqrt{p}$ en cada árbol con p el total de variables explicativas del conjunto de entrenamiento.

4.3. Boosting

La idea de este algoritmo surge alrededor de 1996 a través de Freund y Schapire. Al igual que el Bagging, este modelo es aplicable a cualquier tipo de clasificador aunque también es frecuente su uso con árboles de clasificación.

Tenemos un conjunto de entrenamiento $\{\underline{x}_i, y_i\}_{i=1}^n$ en un problema de clasificación binario con $\mathcal{Y} = \{-1, 1\}$, para estos datos se construye un clasificador h_1 y se evalúan los fallos cometidos, para la construcción del siguiente clasificador se introducen unos pesos a aquellos datos en los que el clasificador anterior cometió errores y se vuelve a evaluar, realizando este proceso tantas veces como clasificadores se deseen construir para aplicar Boosting. Llamaremos h al clasificador final basado en los clasificadores $h_1, \ldots, h_k$ construidos para aplicar Boosting y la regla de clasificación consistirá en:

$$h(\underline{x}) = \begin{cases} 1 & \text{si} \quad \sum_{j=1}^{k} \alpha_j h_j(\underline{x}) \geq 0, \\ -1 & \text{si} \quad \sum_{j=1}^{k} \alpha_j h_j(\underline{x}) \leq 0, \end{cases}$$

siendo α_j unos pesos asociados en este caso a los clasificadores que servirán para dar mayor importancia a aquellos que realicen mejores clasificaciones, con $j = 1, \ldots, k$.

Se define el peso w_i^j como el valor de importancia que se le da al dato i en el clasificador j, con $i = 1, \ldots, n$ y $j = 1, \ldots, k$. Para el primer clasificador establecemos:

$$w_i^1 = \frac{1}{n}, \quad \forall i = 1, \ldots, n,$$

es decir, para el primer clasificador todos los datos tienen la misma importancia. El Boosting se construye como un modelo aditivo por etapas (*stagewise additive model*) que significa que en la etapa j se ajustan los parámetros relacionados con esta etapa asumiendo que los anteriores están bien optimizados. Con todo esto, en cada etapa hay que minimizar:

$$\min_{\alpha_j, h_j} \sum_{i=1}^{n} e^{-y_i \sum_{r=1}^{j} \alpha_r h_r(\underline{x}_i)}, \quad \forall j = 1, \ldots, k,$$

en el exponente podemos reescribir los términos para separar aquellos que han de ser minimizados de los otros, agrupando:

$$-y_i \sum_{r=1}^{j} \alpha_r h_r(\underline{x}_i) = -y_i \sum_{r=1}^{j-1} \alpha_r h_r(\underline{x}_i) - y_i \alpha_j h_j(\underline{x}_i) = f_{j-1}(\underline{x}_i, y_i) - y_i \alpha_j h_j(\underline{x}_i), \ \forall i = 1, \ldots, n, \ \forall j = 1, \ldots, k,$$

sustituyendo en la función objetivo nos queda:

$$\min_{\alpha_j, h_j} \sum_{i=1}^{n} e^{f_{j-1}(\underline{x}_i, y_i)} e^{-y_i \alpha_j h_j(\underline{x}_i)} = \min_{\alpha_j, h_j} \sum_{i=1}^{n} w_i^j e^{-y_i \alpha_j h_j(\underline{x}_i)}, \quad \forall j = 1, \ldots, k,$$

ya que la suma implica todos los datos, podemos separar la suma en función al acierto o error que pueda cometer el clasificador h_j de forma que:

$$\min_{\alpha_j, h_j} \sum_{i=1}^{n} w_i^j e^{-y_i \alpha_j h_j(\underline{x}_i)} = \min_{\alpha_j, h_j} e^{-\alpha_j} \sum_{h_j(\underline{x}_i) = y_i} w_i^j + e^{\alpha_j} \sum_{h_j(\underline{x}_i) \neq y_i} w_i^j, \quad \forall j = 1, \ldots, k,$$

tomando funciones indicadoras en ambos sumandos:

$$\min_{\alpha_j, h_j} e^{-\alpha_j} \sum_{i=1}^{n} w_i^j I(h_j(\underline{x}_i) = y_i) + e^{\alpha_j} \sum_{i=1}^{n} w_i^j I(h_j(\underline{x}_i) \neq y_i), \quad \forall j = 1, \ldots, k,$$

por último, escribimos la expresión anterior en torno al indicador $I(h_j(\underline{x}_i) \neq y_i)$ para obtener la función objetivo a minimizar:

$$\min_{\alpha_j, h_j} e^{-\alpha_j} \sum_{i=1}^{n} w_i^j - e^{-\alpha_j} \sum_{i=1}^{n} w_i^j I(h_j(\underline{x}_i) \neq y_i) + e^{\alpha_j} \sum_{i=1}^{n} w_i^j I(h_j(\underline{x}_i) \neq y_i), \quad \forall j = 1, \ldots, k.$$

Para encontrar un mínimo de la función anterior para α_j derivamos e igualamos a cero obteniendo:

$$\frac{e^{-\alpha_j} \sum_{i=1}^{n} w_i^j I(h_j(\underline{x}_i) \neq y_i)}{\sum_{i=1}^{n} w_i^j} + \frac{e^{\alpha_j} \sum_{i=1}^{n} w_i^j I(h_j(\underline{x}_i) \neq y_i)}{\sum_{i=1}^{n} w_i^j} = e^{-\alpha_j}, \forall j = 1, \ldots, k,$$

donde se han dividido todos los términos por $\sum_{i=1}^{n} w_i^j$. Haciendo $L_j = \dfrac{\sum_{i=1}^{n} w_i^j I(h_j(\underline{x}_i) \neq y_i)}{\sum_{i=1}^{n} w_i^j}$, podemos encontrar una solución para α_j de la forma:

$$\alpha_j = \frac{1}{2} \log(\frac{1 - L_j}{L_j}), \quad \forall j = 1, \ldots, k.$$

Con respecto a los parámetros anteriores, L_j es la función encargada de penalizar los errores de h_j en base a los valores de los pesos w_i^j. Si el valor de L_j es bajo implica que h_j realiza una buena labor de clasificación y en consecuencia el valor de α_j aumenta para darle más importancia a h_j dentro de la regla de clasificación anterior.

Por otra parte, es posible dar una expresión iterativa para los pesos teniendo en cuenta que cuando se cambian estos de una iteración a otra se guarda toda la información obtenida hasta ese momento, es decir:

$$w_i^{j+1} = w_i^j e^{-y_i \alpha_j h_j(\underline{x}_i)}, \quad \forall i = 1, \ldots, n, \quad \forall j = 1, \ldots, k-1,$$

no obstante, ya que $-y_i h_j(\underline{x}_i) = 2I(h_j(\underline{x}_i) \neq y_i) - 1$, sustituyendo tenemos:

$$w_i^{j+1} = w_i^j e^{2\alpha_j I(h_j(\underline{x}_i) \neq y_i)} e^{-\alpha_j}, \quad \forall i = 1, \ldots, n, \quad \forall j = 1, \ldots, k-1,$$

pero a efectos del problema, multiplicar todos los pesos de la etapa j por $e^{-\alpha_j}$ no constituyen ninguna mejora para la clasificación (basta observar la expresión de L_j) por lo tanto:

$$w_i^{j+1} = w_i^j e^{2\alpha_j I(h_j(\underline{x}_i) \neq y_i)}, \quad \forall i = 1, \ldots, n, \quad \forall j = 1, \ldots, k-1,$$

Todo este proceso es conocido por AdaBoost (*Adaptative Boosting*) y podemos resumirlo en las siguientes etapas:

1. Establecer el tipo de clasificadores y el número total que se van a utilizar.
2. Hacer $w_i^1 = \frac{1}{n}$ con $i = 1, \ldots, n$ y construir el primer clasificador h_1 que minimice L_1, obtener α_1.
3. En general, recalcular los pesos w_i^j mediante el proceso iterativo, encontrar el clasificador h_j que minimice L_j y calcular α_j.
4. Una vez obtenidos todos los clasificadores aplicar la regla de clasificación a cada uno de los datos de entrenamiento.

Para extender este concepto a problemas con más clases, se puede recurrir a los enfoques de uno contra uno o uno contra todos que se usaban en las máquinas de vectores soporte, sin embargo, existe un algoritmo con ligeras diferencias con respecto al caso binario que hacen que se pueda extender este algoritmo a problemas con más de dos clases. Este algoritmo es conocido como AdaBoost.M1 y se diferencia en el caso binario en estos detalles, el primero con respecto a la actualización de los pesos que ahora vendría dada por:

$$w_i^{j+1} = \begin{cases} w_i^j e^{-\alpha_j} & \text{si} \quad h_j(\underline{x}_i) = y_i \\ w_i^j e^{\alpha_j} & \text{si} \quad h_j(\underline{x}_i) \neq y_i \end{cases}, \quad \forall i = 1, \ldots, n, \quad \forall j = 1, \ldots, k-1,$$

el segundo con respecto a la regla de clasificación que ahora queda definida como sigue:

$$h(\underline{x}) = \arg\max_{y \in \mathcal{Y}} \sum_{j=1}^{k} \alpha_j I(h_j(\underline{x}) = y).$$

Por lo tanto, la diferencia entre el AdaBoost y el AdaBoost.M1 se encuentra en los pasos 3 y 4 del resumen anterior, siendo los dos primeros exactamente iguales.

Es posible que al tratar de encontrar un clasificador h_j con los pesos correspondientes, el resultado de L_j sea mayor o igual que $\frac{1}{2}$, en ese caso es conveniente parar la iteración y realizar el Boosting con $k = j - 1$ clasificadores. Este hecho también puede verse reflejado en la expresión de α_j, pues para dichos valores de L_j se tiene que α_j es 0 (el clasificador no cuenta en la regla de clasificación) o menor que 0 que puede interpretarse como una medida de poca fiabilidad del clasificador h_j, con $j = 1, \ldots, k$.

Para este último ejemplo, vamos a realizar una comparación entre el Random Forest y el Boosting, para ello vamos a tomar el mismo conjunto de datos que en el ejemplo anterior y vamos a realizar iteraciones del modelo con el fin de comparar los errores cometidos por ambos clasificadores.

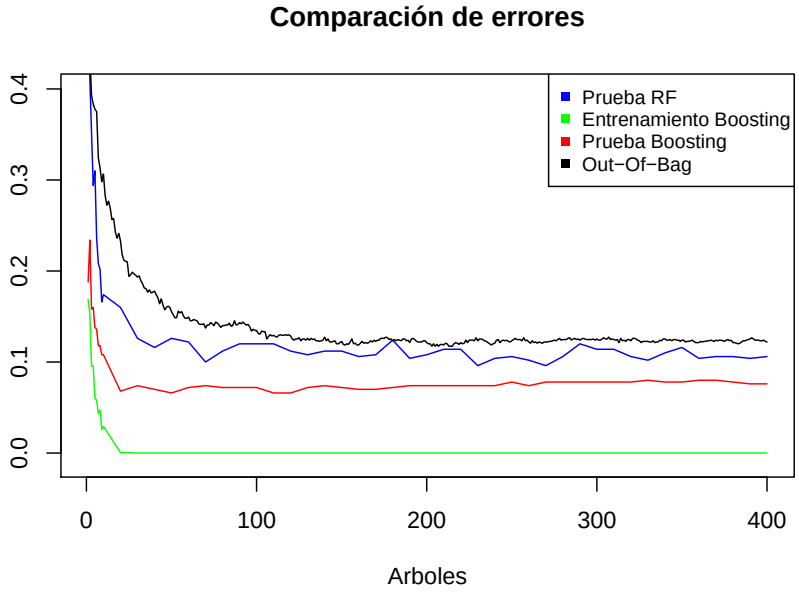
La implementación de este algoritmo se encuentra en la librería de *R adabag*. Leemos los datos dividiendo los conjuntos de la misma forma que en el ejemplo anterior.

```
library(caret)
library(biotools)
library(xtable)
library(rpart)
library(randomForest)
library(adabag)
movil = read.csv("movilE.csv",header = TRUE, sep=",")
cat = c(2,4,6,18,19,20,21)
for (i in cat){
movil[,i] = as.factor(movil[,i])
}
set.seed(575658)
muestra = sample(1:2000,1500)
movilE = movil[muestra,]
movilT = movil[-muestra,]
```

Creamos el modelo a partir de la función *boosting* que aplica el algoritmo AdaBoost.M1 usando árboles de clasificación con la posibilidad de aplicar bootstrap en la muestra. En este caso se hará igual que se ha detallado anteriormente, es decir, sin bootstrap. Con el siguiente bucle *for* calculamos los errores cometidos por los diferentes modelos de Boosting.

```
v = 1:40
v = c(1:9,10*v)
errorBE = NULL
errorBT = NULL
for (i in v){
boostmovil = boosting(price_range ~ .,movilE,mfinal = i,boos = FALSE,
                      coeflearn = "Breiman")
predBME = predict(boostmovil,movilE[,-21])
be = confusionmatrix(predBME$class,movilE[,21])
bem = sum(diag(be))/sum(be)
predBMT = predict(boostmovil,movilT[,-21])
bt = confusionmatrix(predBMT$class,movilT[,21])
btm = sum(diag(bt))/sum(bt)
errorBE = cbind(errorBE,bem)
errorBT = cbind(errorBT,btm)
}
errorBE = 1-errorBE
errorBT = 1-errorBT
```

Los datos referentes al modelo Random Forest se consiguen a partir del anterior ejemplo. Aquí podemos ver una gráfica comparando los diferentes modelos según el número de árboles utilizados.



Los errores cometidos por el Boosting son ligeramente inferiores a los cometidos por el Random Forest. Para el último caso detallamos el error de prueba del clasificador.

```
respuesta = as.factor(predBMT$class)
confusionMatrix(respuesta,movilT[,21])
```

```
Confusion Matrix and Statistics

          0   1   2   3
0  124   4   0   0
1    2 109   9   0
2    0   9 104   6
3    0   0   8 125

Overall Statistics

Accuracy : 0.924
95% CI : (0.8972, 0.9457)
No Information Rate : 0.262
P−Value [Acc > NIR] : < 2.2e−16

Kappa : 0.8986
Mcnemar's Test P−Value : NA
```

```
Statistics by Class:

                     Class: 0 Class: 1 Class: 2 Class: 3
Sensitivity            0.9841   0.8934   0.8595   0.9542
Specificity            0.9893   0.9709   0.9604   0.9783
Pos Pred Value         0.9688   0.9083   0.8739   0.9398
Neg Pred Value         0.9946   0.9658   0.9554   0.9837
Prevalence             0.2520   0.2440   0.2420   0.2620
Detection Rate         0.2480   0.2180   0.2080   0.2500
Detection Prevalence   0.2560   0.2400   0.2380   0.2660
Balanced Accuracy      0.9867   0.9322   0.9100   0.9663
```

Este algoritmo es mucho más costoso computacionalmente que el Random Forest ya que emplea todas las variables y no es aconsejable emplearlo en problemas con un gran número de clases diferentes.

Algunos autores señalan que el algoritmo Boosting consigue en general mejores resultados que el Random Forest, este último mejora al Bagging y el Bagging clasifica mejor que un solo árbol.

Bibliografía

[1] Alpayadin.E. (2004). *Introduction to Machine Learning*. MIT Press.

[2] Bishop.C.N. (1995). *Neural Network for Pattern Recognition*. Oxford.

[3] Breiman.L, Friedman.J, Olshen.R.A, Stone.C.J. (1984). *Classification And Regression Trees*. Chapman and Hall/CRC.

[4] Breiman.L. (1994). *Bagging Predictors*.

[5] Breiman.L. (2001). *Random Forests*.

[6] Devroye.L, László.G, Gábor.L. (1996). *A probabilistic theory of pattern recognition*. Springer.

[7] Duda.R, Hart.P, Stork.D. (1973). *Pattern Classification* .Wiley.

[8] Fernández.J.M, Flórez.R. (2008). *Las Redes Neuronales Artificiales. Fundamentos teóricos y aplicaciones prácticas*. Netbiblo.

[9] Freund.Y, Schapire.R.E. (2012). *Boosting. Foundations and Algorithms*. MIT Press.

[10] Friedman.J, Hastie.T, Tibshirani.R. (2001). *The Elements of Statistical Learning*. Springer.

[11] Funahashi.K.I. (1989). *On the Approximate Realization of Continuous Mappings by Neural Network*.

[12] Ghosdi.A. (2015). uwaterloo.ca/data-analytics/statistical-learning-classification.

[13] Han.J, Kamber.M, Pei.J. (2000). *Data Mining: Concepts and Techniques*. Morgan Kaufmann.

[14] Hastie.T, James.G, Tibshirani.R, Witten.D. (2013). *An introduction to Statistical Learning: With Applications in R*. Springer.

[15] Inza.I, Larrañaga.P, Moujahid.A. (2018). *Tema 8. Redes Neuronales*.

[16] Izenman.A. (2006). *Modern Multivariate Statistical Techniques*. Springer.

[17] Michie.D, Spiegelhalter.D.J, Taylor.C.C. (1994). *Machine Learning, Neural and Statistical Classification.*

[18] Minsky.M, Papert.S. (1969). *Perceptrons: an introduction to computational geometry.* MIT Press.

[19] Mitchell.T. (1997). *Machine Learning.* McGraw-Hill.

[20] Murphy.K. (2012). *Machine Learning: A Probabilistic Perspective.* MIT Press.

[21] Seber.G.A.F. (1984). *Multivariate Observations.* Wiley.

[22] Webb.A. (2002). *Statistical Pattern Recognition.* Wiley.