Máquinas de Vector Soporte (Support Vector Machines, SVMs)

Joaquín Amat Rodrigo j.amatrodrigo@gmail.com

Abril, 2017

Índice

Formato PDF: https://github.com/JoaquinAmatRodrigo/Estadistica-con-R

Introducción

El método de clasificación-regresión Maquinas de Vector Soporte (*Vector Support Machines, SVMs*) fue desarrollado en la década de los 90, dentro de campo de la ciencia computacional, como una generalización del *Maximal Margin Classifier*. Si bien originariamente se desarrollaron como un método de clasificación binaria, su aplicación se ha extendido a problemas de clasificación múltiple y a regresión. *SVMs* ha resultado ser uno de los mejores clasificadores para un amplio abanico de situaciones. En este capítulo se aborda únicamente su aplicación a problemas de clasificación.

Los *SVMs* se engloban dentro de los clasificadores lineales puesto que generan límites de decisión lineales ya sea en el espacio original de las observaciones (casos separables o cuasi-separables linealmente) o en un espacio transformado (casos no separables linealmente).

Maximal Margin Classifier

Las Maquinas de Vector Soporte se fundamentan en el *Maximal Margin Classifier*, que a su vez se basa en el concepto de hiperplano. En un espacio p-dimensional, un hiperplano se define como un subespacio plano y afín de dimensiones $p - 1$. El término afín significa que el subespacio no tiene por qué pasar por el origen. En un espacio de dos dimensiones, el hiperplano es un subespacio plano de 1 dimensión, es decir, una recta. En un espacio tridimensional, un hiperplano es un subespacio plano de dos dimensiones, un plano convencional. Para dimensiones de $p > 3$ no es intuitivo visualizar un hiperplano, pero el concepto de subespacio plano con $p - 1$ dimensiones se mantiene.

La definición matemática de un hiperplano es bastante simple. En el caso de dos dimensiones, el hiperplano se describe acorde a la ecuación de una recta:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$$

Dados los parámetros β_0, β_1 y β_2, todos los pares de valores $X = X_1, X_2$ para los que se cumple la igualdad son puntos del hiperplano. Esta ecuación puede generalizarse para p-dimensiones:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p = 0$$

y de igual manera, todos los puntos definidos por el vector $X = X_1, X_2, \ldots, X_p$ que cumplen la ecuación pertenecen al hiperplano.

Cuando X no satisface la ecuación:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p < 0$$

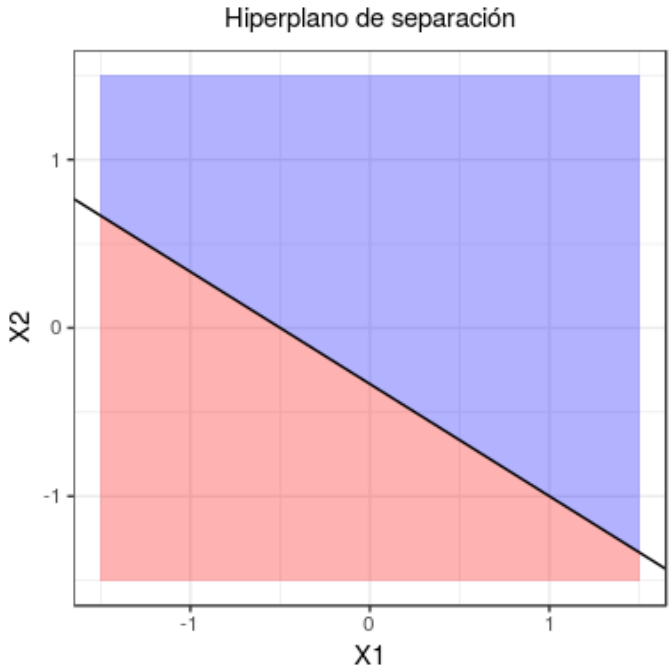
o bien

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p > 0$$

el punto X cae a un lado o al otro del hiperplano. Así pues, se puede entender que un hiperplano divide un espacio p-dimensional en dos mitades. Para saber a qué lado del hiperplano se encuentra un determinado punto X, solo es necesario calcular el signo de la ecuación.

La siguiente imagen muestra el hiperplano de un espacio bidimensional. La ecuación que describe el hiperplano (una recta) es $1 + 2X_1 + 3X_2 = 0$. La región azul representa el espacio en el que se encuentran todos los puntos para los que $1 + 2X_1 + 3X_2 > 0$ y la región roja el de los puntos para los que $1 + 2X_1 + 3X_2 < 0$.

```
library(ggplot2)
library(gridExtra)

# A partir de la ecuación de la recta se pueden obtener las coordenadas para X2
# en función de X1.
# X2 = -(1/3) -(2/3)X1
coordenadas  <- data.frame(X1 = c(-1.5, 1.5, 1.5, -1.5, -1.5, 1.5, 1.5, -1.5),
                           X2 = c(-1.5, -1.5, -4/3, 2/3 , 2/3, -4/3, 1.5, 1.5),
                           poligono = c('a', 'a', 'a', 'a', 'b', 'b', 'b', 'b'))

ggplot() +
geom_polygon(data = coordenadas , aes(x = X1, y = X2, fill = poligono),
             alpha = 0.3) +
scale_fill_manual(values = c(a = "red", b = "blue")) +
geom_abline(intercept = -1/3, slope = -2/3) +
theme_bw() +
labs(title = "Hiperplano de separación") +
theme(legend.position = "none", plot.title = element_text(hjust = 0.5, size = 11))
```



Clasificación binaria (2 clases) empleando un hiperplano

Cuando se dispone de n observaciones, cada una con p predictores y cuya variable respuesta tiene dos niveles (de aquí en adelante identificados como $+1$ y -1), se pueden emplear hiperplanos para construir un clasificador que permita predecir a que grupo pertenece una observación en función de sus predictores. Este mismo problema puede abordarse también con otros métodos (regresión logística, LDA, árboles de clasificación...) cada uno con ventajas y desventajas.

Para facilitar la comprensión, las siguientes explicaciones se basan en un espacio de dos dimensiones, donde un hiperplano es una recta. Sin embargo, los mismos conceptos son aplicables a dimensiones superiores.

CASOS PERFECTAMENTE SEPARABLES LINEALMENTE

Si la distribución de las observaciones es tal que se pueden separar linealmente de forma perfecta en las dos clases ($+1$ y -1), entonces, un hiperplano de separación cumple que:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p > 0, \; si \; y_i = 1$$

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p < 0, \; si \; y_i = -1$$

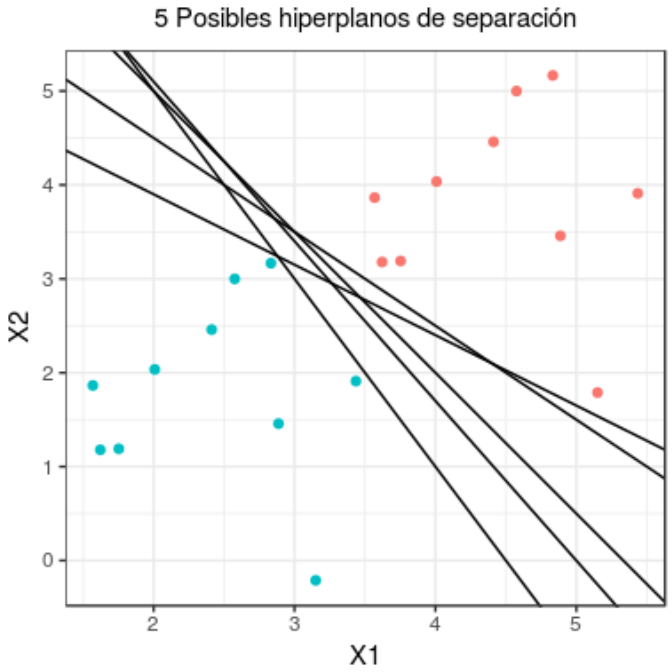
Al identificar cada clase como $+1$ o -1, y dado que multiplicar dos valores negativos resultan en un valor positivo, las dos condiciones anteriores pueden simplificarse en una única:

$$y_i(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p) > 0, \;\; para\; i = 1 \ldots n$$

Bajo este escenario, el clasificador más sencillo consiste en asignar cada observación a una clase dependiendo del lado del hiperplano en el que se encuentre. Es decir, la observación x^* se clasifica acorde al signo de la función $f(x^*) = \beta_0 + \beta_1 x_1^* + \beta_2 x_2^* + \ldots + \beta_p x_p^*$. Si $f(x^*)$ es positiva, la observación se asigna a la clase $+1$, si es negativa, a la clase -1. Además, la magnitud de $f(x^*)$ permite saber cómo de lejos está la observación del hiperplano y con ello la confianza de la clasificación.

La definición de hiperplano para casos perfectamente separables linealmente resulta en un número infinito de posibles hiperplanos, lo que hace necesario un método que permita seleccionar uno de ellos como clasificador óptimo.

```
set.seed(68)
X1 <- rnorm(n = 10, mean = 2, sd = 1)
X2 <- rnorm(n = 10, mean = 2, sd = 1)

observaciones <- data.frame(X1 = c(X1, X1 + 2), X2 = c(X2, X2 + 2) ,
                            clase = rep(c(1, -1), each = 10))
observaciones$clase <- as.factor(observaciones$clase)

ggplot() +
  geom_point(data = observaciones, aes(x = X1, y = X2, color = clase)) +
  geom_abline(intercept = 9, slope = -2) +
  geom_abline(intercept = 8.5, slope = -1.7) +
  geom_abline(intercept = 8, slope = -1.5) +
  geom_abline(intercept = 6.5, slope = -1) +
  geom_abline(intercept = 5.4, slope = -0.75) +
  theme_bw() +
  labs(title = "5 Posibles hiperplanos de separación") +
  theme( legend.position = "none",
  plot.title = element_text(hjust = 0.5, size = 11))
```



Una solución a este problema consiste en seleccionar como clasificador al que se conoce como *maximal margin hyperplane* o *hiperplano óptimo de separación*, que es el hiperplano que se encuentra más alejado de todas las observaciones de entrenamiento. Para obtenerlo, se calcula la distancia perpendicular de cada observación a un determinado hiperplano. La menor de estas distancias (conocida como margen) determina como de alejando está el hiperplano de las observaciones de entrenamiento. El *maximal margin hyperplane* es el hiperplano que consigue un mayor margen, es decir, que la distancia mínima entre el hiperplano y las observaciones es lo más grande posible. Es intuitivo llegar a la conclusión de que el *maximal margin hyperplane* está situado en el punto medio entre las observaciones de clases distintas más cercanas.

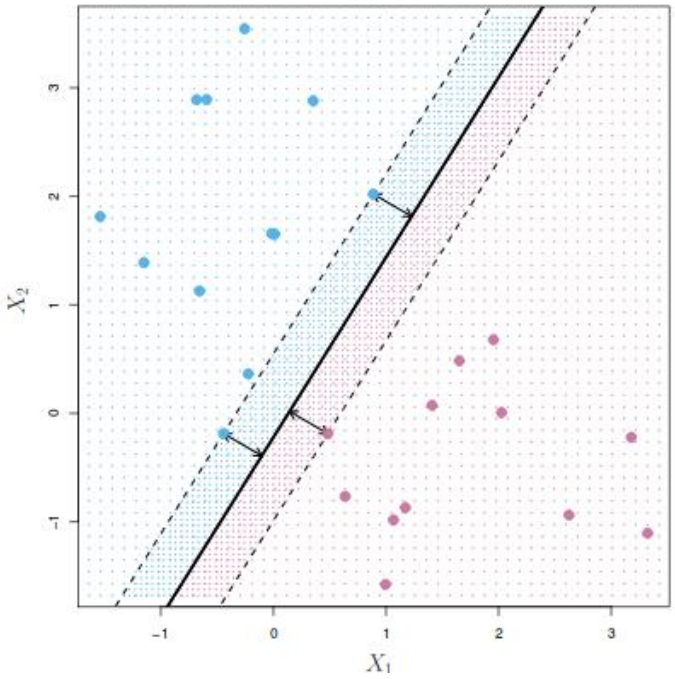


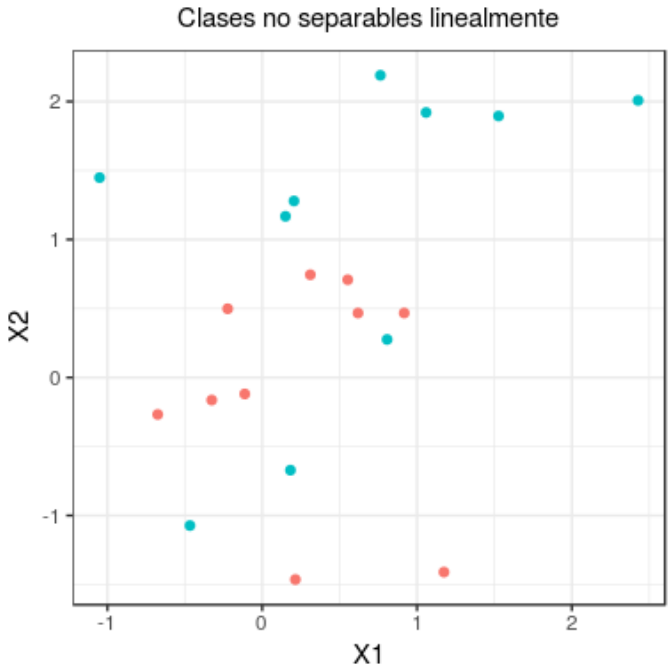
Imagen maximal margin hyperplane obtenida del libro ISLR

La imagen anterior muestra el *maximal margin hyperplane* para un conjunto de datos de entrenamiento. Las tres observaciones equidistantes respecto al *maximal margin hyperplane* se encuentran a lo largo de las líneas discontinuas que indican la anchura del margen. A estas observaciones se les conoce como vectores soporte, ya que son vectores en un espacio *p*-dimensional y soportan (definen) el *maximal margin hyperplane*. Cualquier modificación en estas observaciones (vectores soporte) conlleva cambios en el *maximal margin hyperplane*. Sin embargo, modificaciones en observaciones que no son vector soporte no tienen impacto alguno en el hiperplano.

CASOS CUASI-SEPARABLES LINEALMENTE

El *maximal margin hyperplane* descrito en el apartado anterior es una solución muy simple y natural de clasificación siempre y cuando exista un hiperplano de separación. En la gran mayoría de casos reales, los datos no se pueden separar linealmente de forma perfecta, por lo que no existe un hiperplano de separación y no puede obtenerse un *maximal margin hyperplane*.

```
set.seed(101)
coordenadas  <- matrix(rnorm(40), 20, 2)
colnames(coordenadas ) <- c("X1","X2")
y <- c(rep(-1,10), rep(1,10))
y <- as.factor(y)
coordenadas [y == 1, ] <- coordenadas [y == 1, ] + 1

datos <- data.frame(coordenadas , y)
ggplot(data = datos, aes(x = X1, y = X2, color = y)) +
  geom_point() +
  theme_bw() +
  labs(title = "Clases no separables linealmente") +
  theme( legend.position = "none",
  plot.title = element_text(hjust = 0.5, size = 11))
```



En estas situaciones, se puede extender el concepto de hiperplano de separación para obtener un hiperplano que casi separe las clases permitiendo que cometa unos pocos errores, ha este último se le conoce como *Support Vector Classifier* o *soft margin*.

Support Vector Classifier o hiperplano de separación de margen blando

El *Maximal Margin Classifier* descrito en la sección anterior tiene poca aplicación práctica ya que raramente se encuentran casos en los que las clases sean perfecta y linealmente separables. De hecho, incluso cumpliéndose estas condiciones ideales en las que exista un hiperplano capaz de separar perfectamente las observaciones en dos clases, esta aproximación sigue presentando dos inconvenientes:

- Dado que el hiperplano tiene que separar perfectamente las observaciones, es muy sensible a variaciones en los datos. Incluir una nueva observación puede suponer cambios muy grandes en el hiperplano de separación (poca robustez).
- Que el *maximal margin hyperplane* se ajuste perfectamente a las observaciones de entrenamiento para separarlas todas correctamente suele conllevar problemas de *overfitting*.

Por estas razones, es preferible crear un clasificador basado en un hiperplano que, aunque no separe perfectamente las dos clases, sea más robusto y tenga mayor capacidad predictiva al aplicarlo a nuevas observaciones (menos problemas de *overfitting*). Esto es exactamente lo que consiguen los clasificadores de vector soporte, también conocidos como *soft margin classifiers* o *Support Vector Classifiers*. Para lograrlo, en lugar de buscar el margen de clasificación más ancho posible que consigue que las observaciones estén en el lado correcto del margen; se permite que ciertas observaciones estén en el lado incorrecto del margen o incluso del hiperplano. El termino *soft margin* hace referencia a que ciertas observaciones pueden violar el margen de clasificación.

La siguiente imagen muestra un clasificador de vector soporte ajustado a un pequeño set de observaciones. La línea continua representa el hiperplano y las líneas discontinuas el margen a cada lado. Las observaciones 2, 3, 4, 5, 6, 7 y 10 se encuentran en el lado correcto del margen (también del hiperplano) por lo que están bien clasificadas. Las observaciones 1 y 8, a pesar de que se encuentran dentro del margen, están en el lado correcto del hiperplano, por lo que también están bien clasificadas. Las observaciones 11 y 12 se encuentran en el lado erróneo del hiperplano, su clasificación es incorrecta. Todas aquellas observaciones que, estando dentro o fuera del margen, se encuentren en el lado incorrecto del hiperplano, se corresponden con observaciones de entrenamiento mal clasificadas.

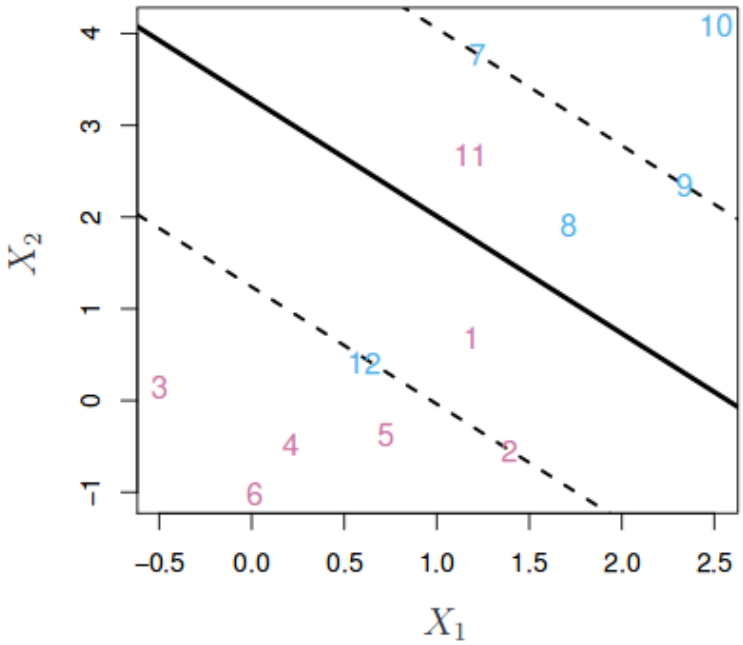


Imagen clasificador vector soporte obtenida del libro ISLR

La obtención del hiperplano de un clasificador de vector soporte, que clasifique correctamente la mayoría de las observaciones a excepción de unas pocas, es un problema de optimización. Si bien la demostración matemática queda fuera del objetivo de esta introducción (ver bibliografía), es importante mencionar que el proceso incluye un parámetro de *tuning* C. C controla el número y severidad de las violaciones del margen (y del hiperplano) que se toleran en el proceso de ajuste. Si $C = 0$, no se permite ninguna violación del margen y por lo tanto, el resultado es equivalente al *maximal margin hyperplane* (teniendo en cuenta que esta solución solo es posible si las clases son perfectamente separables). Cuando $C > 0$, no se permite que más de C observaciones estén en el lado incorrecto del hiperplano. Cuanto mayor sea C mayor es la tolerancia de los errores y por lo tanto los margenes resultantes son más anchos. C es a fin de cuentas el parámetro encargado de controlar el balance entre *bias* y varianza del modelo. En la práctica, su valor óptimo se identifica mediante *cross-validation*.

EL proceso de optimización tiene la peculiaridad de que solo las observaciones que se encuentran justo en el margen o que lo violan influyen sobre el hiperplano. Ha estas observaciones se les conoce como vectores soporte y son las que definen el clasificador obtenido. Este hecho es la razón por la que el parámetro C controla el balance entre *bias* y varianza. Cuando el valor de C es grande, el margen es más ancho, y más observaciones violan el margen convirtiéndose en vectores soporte. El hiperplano está por lo tanto sustentado por más observaciones, lo que aumenta el *bias* pero reduce la varianza. Cuando menor es el valor

de C, menos observaciones serán vectores soporte y el clasificador resultante tendrá menor *bias* pero mayor varianza.

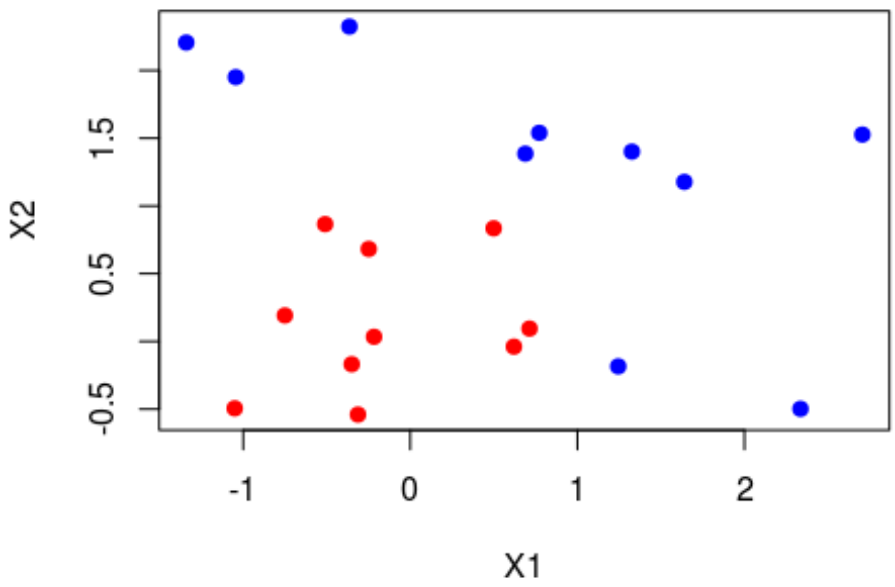
Otra propiedad importante que deriva de que la regla de clasificación dependa únicamente de una pequeña proporción de observaciones (vectores soporte), es su robustez frente a observaciones muy alejadas del hiperplano. Esto hace al método de *clasificación vector soporte* distinto a otros métodos tales como *Linear Discrimiant Analysis LDA*, donde la regla de clasificación depende de la media de todas las observaciones. La regresión logística también es poco sensible a observaciones alejadas del límite de decisión, de hecho, más adelante se puntualizará la semejanza entre esta y los clasificadores de vector soporte.

Ejemplo

`e1071` y `LiblineaR` son dos paquetes que contienen los algoritmos necesarios para obtener clasificadores basados en *Support Vector Classifier* y *Support Vector Machines*. En los siguientes ejemplos se emplea la función `svm()` contenida en `e1071`. Esta función ajusta *Support Vector Classifier* si se le indica el argumento `kernel="linear"` (el método de *Support Vector Machines* es equivalente al *Support Vector Classifier* cuando el *kernel* utilizado es lineal). El argumento `cost` determina la penalización aplicada por violar el margen. Cuanto menor es el valor de `cost` más ancho es el margen y con ello más observaciones pasan a ser vectores soporte. Si el valor de `cost` es alto, la penalización por cada violación del margen es mayor, lo que resulta en márgenes más estrechos y por lo tanto menos observaciones actuando como vectores soporte.

Para ilustrar el funcionamiento de `svm()` se simulan observaciones en un espacio bidimensional que pertenecen a dos clases. *Este ejemplo se ha obtenido de los videos asociados al libro ISLR, no es igual al presentado en el libro.*

```
set.seed(10111)
coordenadas  <- matrix(rnorm(40), 20, 2)
colnames(coordenadas ) <- c("X1", "X2")
y <- c(rep(-1, 10), rep(1, 10))
coordenadas [y == 1, ] <- coordenadas [y == 1, ] + 1
datos <- data.frame(coordenadas , y)
plot(x = datos$X1, y = datos$X2, col = rep(c("red", "blue"), each = 10), pch = 19,
     ylab = "X2", xlab = "X1")
```



La representación gráfica de los datos muestra que los grupos no son linealmente separables.

Para que la función `svm()` realice clasificación, la variable respuesta tiene que ser de tipo factor (si es numérica realiza regresión).

```
library(e1071)
datos$y <- as.factor(datos$y)

# Para que la función svm() calcule el Support Vector Classifier, se tiene
# que indicar que la función kernel es lineal.
modelo_svm <- svm(formula = y ~ X1 + X2, data = datos, kernel = "linear",
                  cost = 10, scale = FALSE)

# IMPORTANTE: En este caso, ambos predictores (X1, X2) tienen la misma
# escala por lo que no es necesario estandarizarlos. En aquellas situaciones
# en las que las escalas son distintas, sí hay que estandarizarlos. Si no se
# hace, los predictores de mayor magnitud eclipsarán a los de menor
# magnitud.
summary(modelo_svm)
```

```
## Call:
## svm(formula = y ~ X1 + X2, data = datos, kernel = "linear", cost = 10,
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##       gamma:  0.5
##
## Number of Support Vectors:  6
##
##  ( 3 3 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

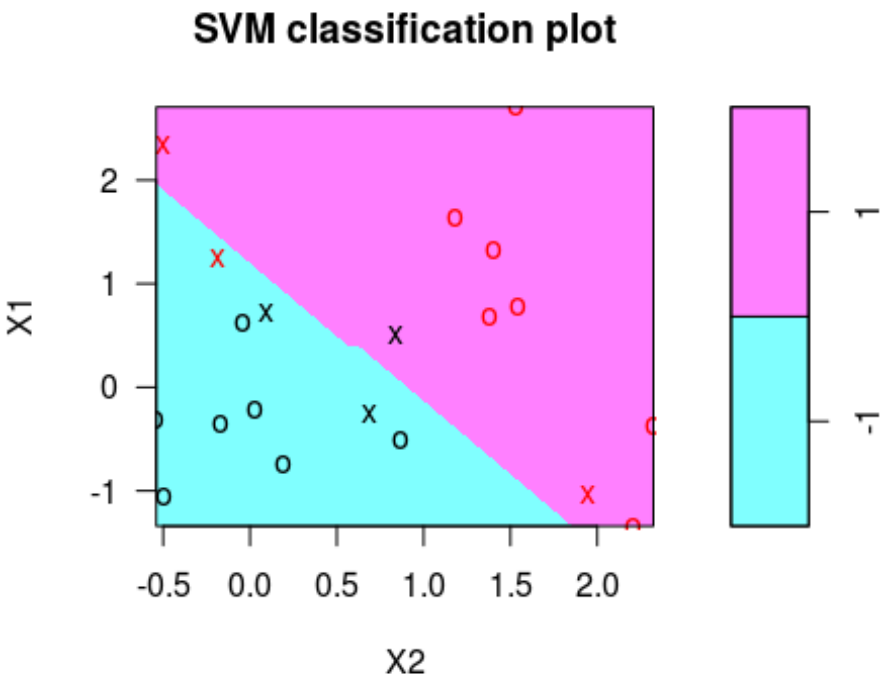
El `summary` del modelo muestra que hay un total de 6 vectores soporte (observaciones que violan el margen), 3 pertenecen a una clase y 3 a la otra. Se pueden extraer los índices de las observaciones que actúan como vectores soporte de la siguiente forma:

```
modelo_svm$index
```

```
## [1]  1  4 10 14 16 20
```

Empleando la función `plot()` con un objeto de tipo `svm` junto con las observaciones de entrenamiento se obtiene una representación del límite de decisión y las dos regiones en las que queda dividido el espacio muestral.

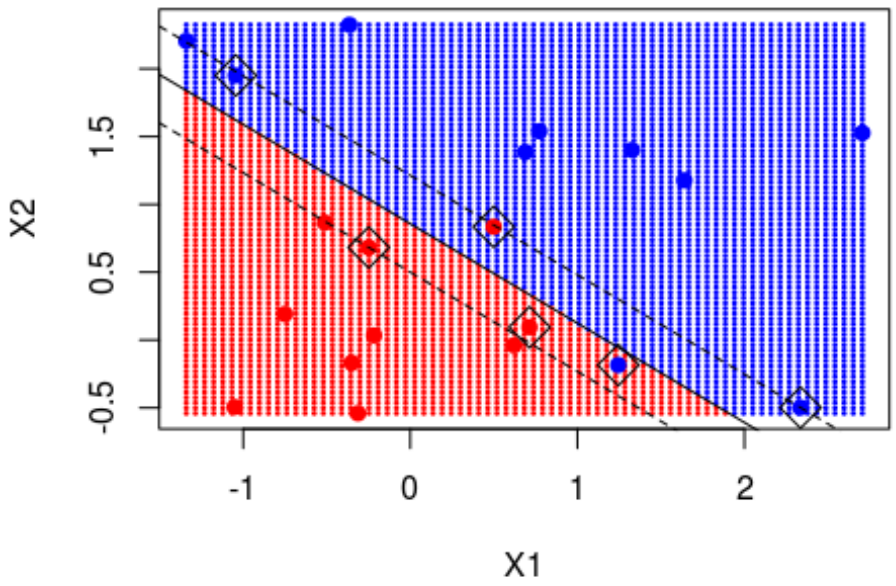
```
plot(modelo_svm, datos)
```



A pesar de que el clasificador es de tipo lineal (`kernel = "linear"`), la baja resolución del gráfico hace parecer que no lo es. Además, esta representación no muestra el margen ni el hiperplano. La siguiente es una alternativa para obtener una representación de un *Support Vector Machine* en espacios bidimensionales.

```
# SI AL AJUSTAR EL MODELO SE INDICA scale = true, SE TIENEN QUE ESTANDARIZAR
# TAMBIÉN LAS OBSERVACIONES PARA QUE COINCIDAN LAS COORDENADAS.

# El primer paso es interpolar puntos dentro del rango de los dos
# predictores X1 y X2. Estos nuevos puntos se emplean para predecir la
# variable respuesta acorde al modelo y así colorear las regiones que separa
# el hiperplano.

# Rango de los predictores
rango_X1 <- range(datos$X1)
rango_X2 <- range(datos$X2)

# Interpolación de puntos
new_x1 <- seq(from = rango_X1[1], to = rango_X1[2], length = 75)
new_x2 <- seq(from = rango_X2[1], to = rango_X2[2], length = 75)
nuevos_puntos <- expand.grid(X1 = new_x1, X2 = new_x2)

# Predicción segun el modelo de los nuevos puntos
predicciones <- predict(object = modelo_svm, newdata = nuevos_puntos)
```

```
# Representación de las 2 regiones empleando los puntos y coloreándolos
# según la clase predicha por el modelo
plot(nuevos_puntos, col = c("red", "blue")[predicciones], pch = 20, cex = 0.35)

# Se añaden las observaciones
points(x = datos$X1, y = datos$X2, col = c("red", "blue")[datos$y], pch = 19)

# Se identifican aquellas observaciones que son vectores soporte del modelo
points(x = datos$X1[modelo_svm$index], y = datos$X2[modelo_svm$index], pch = 5,
    cex = 2)

# Para extraer la ecuación del hiperplano y del margen es necesario aplicar
# algebra lineal.
beta <- drop(t(modelo_svm$coefs) %*% as.matrix(datos[, c("X1",
"X2")])[modelo_svm$index,
    ])
beta0 <- modelo_svm$rho

abline(beta0/beta[2], -beta[1]/beta[2])
abline((beta0 - 1)/beta[2], -beta[1]/beta[2], lty = 2)
abline((beta0 + 1)/beta[2], -beta[1]/beta[2], lty = 2)
```



En el ajuste anterior se ha empleado un valor de penalización `cost = 10`. Este parámetro determina el balance bias-varianza y por lo tanto, es crítico para la capacidad predictiva del modelo. El paquete `e1071` incluye la función `tune()` que realiza *10-cross-validation* para

identificar el valor óptimo de penalización. Entre sus argumentos están: el modelo `smv` y un vector con los valores de penalización que se quieren evaluar.

```
set.seed(1)
svm_cv <- tune("svm", y ~ X1 + X2, data = datos, kernel = "linear", ranges =
list(cost = c(0.001,
    0.01, 0.1, 1, 5, 10, 100)))
summary(svm_cv)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   100
##
## - best performance: 0.1
##
## - Detailed performance results:
##    cost error dispersion
## 1 1e-03  0.70  0.4216370
## 2 1e-02  0.70  0.4216370
## 3 1e-01  0.20  0.2581989
## 4 1e+00  0.25  0.2635231
## 5 5e+00  0.15  0.2415229
## 6 1e+01  0.15  0.2415229
## 7 1e+02  0.10  0.2108185
```

El proceso de *cross-validation* muestra que el valor de penalización con el que se consigue menor *error rate* es `cost = 100`. La función `tune()` almacena el mejor modelo de entre todos los que se han comparado:

```
mejor_modelo <- svm_cv$best.model
summary(mejor_modelo)
```

```
##
## Call:
## best.tune(method = "svm", train.x = y ~ X1 + X2, data = datos,
##     ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)),
##     kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  100
```

```
##       gamma:  0.5
##
## Number of Support Vectors:  5
##
##  ( 2 3 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

Una vez obtenido el modelo final, se puede predecir la clase a la que pertenecen nuevas observaciones empleando la función `predict()`.

```
# Datos de test simulados
set.seed(19)
coordenadas  <- matrix(rnorm(40), 20, 2)
colnames(coordenadas ) <- c("X1", "X2")
y <- sample(c(-1, 1), 20, rep = TRUE)
coordenadas [y == 1, ] <- coordenadas [y == 1, ] + 1
test <- data.frame(coordenadas , y)
head(test)
```

```
##           X1          X2  y
## 1 -0.1894537  2.42342348  1
## 2  1.3885812 -0.04265812  1
## 3  0.6556667  1.00116919  1
## 4  0.4521039  2.09045517  1
## 5  0.9806622 -0.99871525 -1
## 6  0.7633540  1.53482999  1
```

```
# Predicciones
predicciones <- predict(object = mejor_modelo, test)
table(predicción = predicciones, valor_real = test$y)
```

```
##           valor_real
## predicción -1  1
##         -1  7  1
##         1   1 11
```

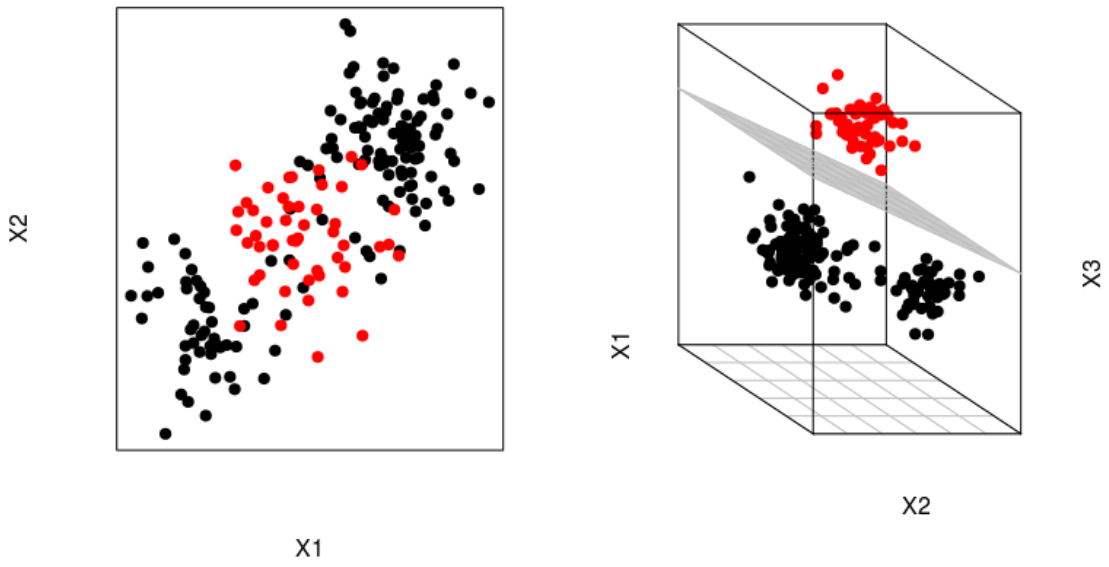
18 de las 20 observaciones han sido clasificadas correctamente por el modelo. El test error es del 10 %.

Support Vector Machines

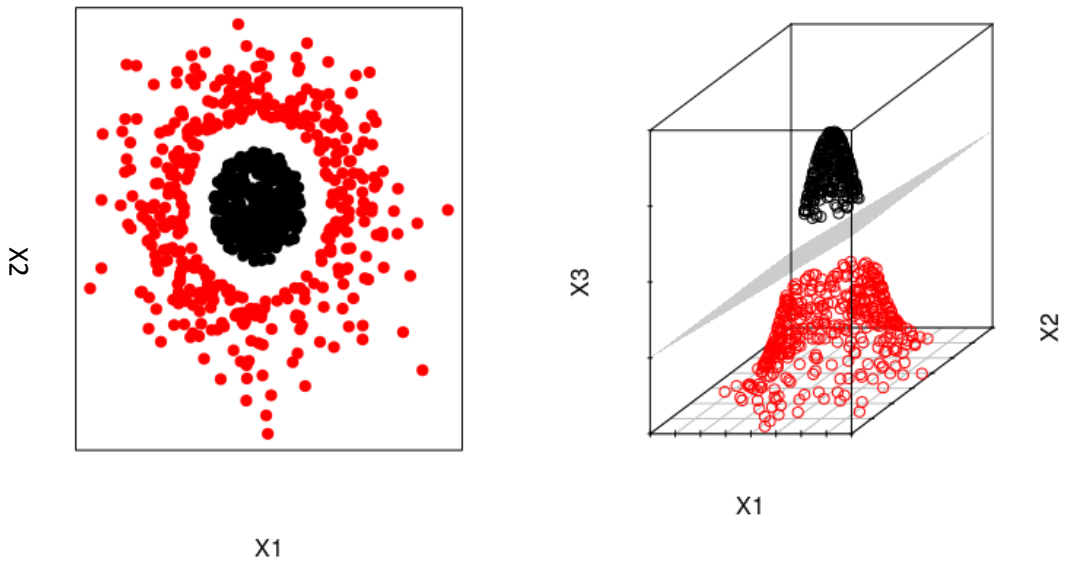
El *Support Vector Classifier* descrito en los apartados anteriores consigue buenos resultados cuando el límite de separación entre clases es aproximadamente lineal. Si no lo es, su capacidad decae drásticamente. Una estrategia para enfrentarse a escenarios en los que la separación de los grupos es de tipo no lineal consiste en expandir las dimensiones del espacio original.

El hecho de que los grupos no sean linealmente separables en el espacio original no significa que no lo sean en un espacio de mayores dimensiones. Las imágenes siguientes muestran como dos grupos, cuya separación en dos dimensiones no es lineal, sí lo es al añadir una tercera dimensión.

```
# EJEMPLO 1
# Datos en 2 dimensiones (X1, X2).
library(scatterplot3d)
par(mfrow = c(1,2))
set.seed(1)
datos <- matrix(rnorm(200 * 2), ncol = 2)
colnames(datos) <- c("X1", "X2")
datos[1:100, ] <- datos[1:100, ] + 2
datos[101:150, ] <- datos[101:150, ] - 2
clase <- c(rep(1, 150) ,rep(2, 50))
datos <- data.frame(datos,clase)
with(datos, plot(x = X1, y = X2, col = clase, pch = 19, labels = FALSE,
                 tick = FALSE))

# Datos en 3 dimensiones (se añade dimensión X3).
datos$X3 <- c(rnorm(n = 150, mean = 3, sd = 1), rnorm(n = 50, mean = 12, sd = 1))
with(datos, {
  p <- scatterplot3d(x = X1, y = X2, z = X3,
                     color = clase, pch = 19,
                     angle = -20, tick.marks = FALSE)
  p$plane3d(8,0.5,0, "solid", col = "grey", draw_polygon = TRUE)
})
```




```r
# EJEMPLO 2
# Distribución normal bivariante.
library(mvtnorm)
sigma_0 <- matrix(c(1, 0, 0, 1), ncol = 2)
datos <- rmvnorm(n = 1000, mean = c(0, 0), sigma = sigma_0)
colnames(datos) <- c("X1", "X2")

# Se añade una tercera dimensión.
X3 <- dmvnorm(datos, mean = c(0,0), sigma = sigma_0)
datos <- cbind(datos, X3)

# Se eliminan observaciones para crear una región vacía en el eje Z.
datos <- datos[!(datos[, "X3"] > 0.06 & datos[, "X3"] < 0.11), ]

#Se asignan las clases
clase <- ifelse(datos[, "X3"] > 0.1, "a", "b")
clase <- as.factor(clase)

# Representación dimensiones X,Y.
plot(x = datos[, "X1"], y =  datos[, "X2"],col = c("black", "red")[clase],
     pch = 19, xlab = "X1", ylab = "X2", labels = FALSE, tick = FALSE)

# Representación dimensiones X1, X2, X3
grafico3d <- scatterplot3d(x = datos[, "X1"], y = datos[, "X2"], z = datos[, "X3"],
                           color=c("black", "red")[clase], xlab ="X1",ylab = "X2",
                           zlab = "X3", angle = 18, label.tick.marks = FALSE)
grafico3d$plane3d(0.09, 0.01, 0, draw_polygon = TRUE, "solid", draw_lines = FALSE)
```



El método de *Support Vector Machines* se puede considerar como una extensión del *Support Vector Classifier* obtenida al aumentar de la dimensionalidad mediante el uso de *kernels*. Los límites de separación lineales generados en el espacio aumentado se convierten en límites de separación no lineales en el espacio original. Un *kernel (K)* es una función que cuantifica la similitud entre dos observaciones en un nuevo espacio dimensional. Existen multitud de *kernels* distintas, algunas de las más usadas son:

Kernel lineal

$$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij}\, , x_{i'j}$$

Si se emplea un Kernel lineal, el clasificador *Support Vector Machine* obtenido es equivalente al *Support Vector Classifier*.

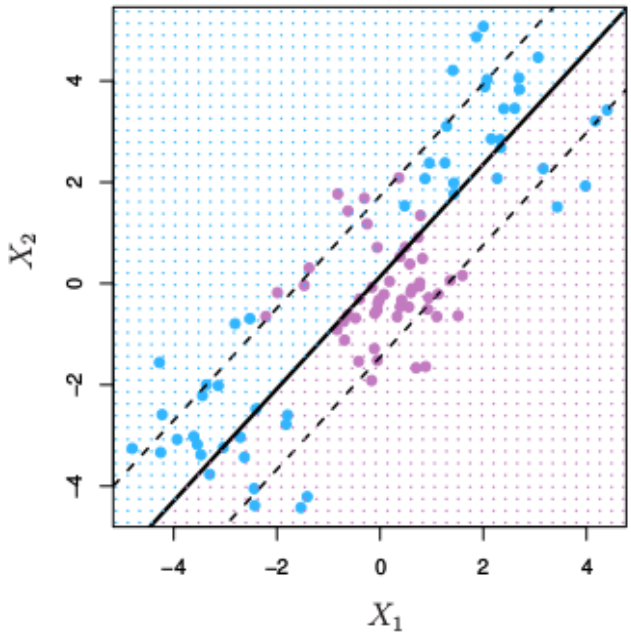


Imagen SVM con una kernel lineal obtenida del libro ISLR

Kernel polinómica

$$K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij}\, x_{i'j})^d$$

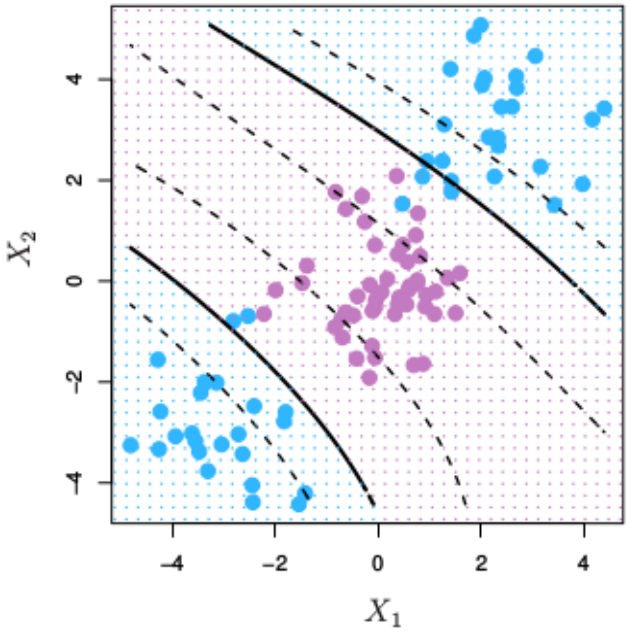
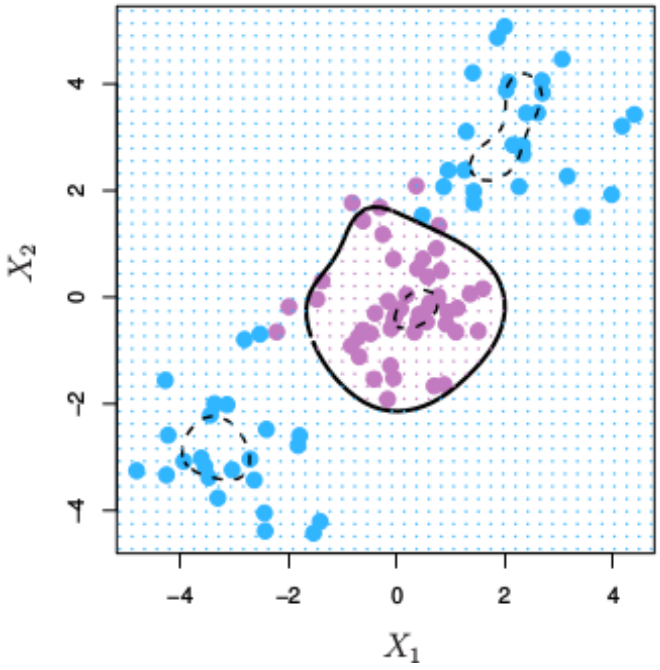


Imagen SVM con una kernel polinómico de grado 3 obtenida del libro ISLR

Si $d > 1$, se generan límites de decisión no lineales. Es al combinar el *Support Vector Classifier* con un *kernel* no lineal cuando se obtiene un clasificador *Support Vector Machine*.

Kernel radial

$$K(x_i, x_{i'}) = exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2))$$



En este tipo de kernel, es necesario indicar el valor del *tunning parameter* γ. Cuanto mayor es su valor, mayor la flexibilidad (no linealidad) del clasificador *SVM* obtenido.

Ejemplo

Tal y como se ha descrito anteriormente, la diferencia entre un *Support Vector Classifier* y un *Support Vector Machine* reside en que estos últimos emplean un *kernel* no lineal. Para obtener este tipo de modelos en `R` se puede emplear la función `svm()` del paquete `e1071`. De entre los *kernel* no lineales que acepta esta función destacan `kernel = "polinomial"`, en cuyo caso hay que indicar el grado d del polinomio, y `kernel = "radial"`, en cuyo caso hay que indicar el parámetro γ.

Para el siguiente ejemplo se va a emplear un set de datos publicado en el libro *Elements of Statistical Learning* que contiene observaciones simuladas con una función no lineal en un espacio de dos dimensiones (2 predictores).

```
# Se eliminan todas las variables para evitar conflictos
rm(list = ls())
# Descargan los datos. Requiere conexión a internet
load(url("https://statweb.stanford.edu/~tibs/ElemStatLearn/datasets/ESL.mixture.rda
"))
class(ESL.mixture)
```

```
## [1] "list"
```

```
names(ESL.mixture)
```

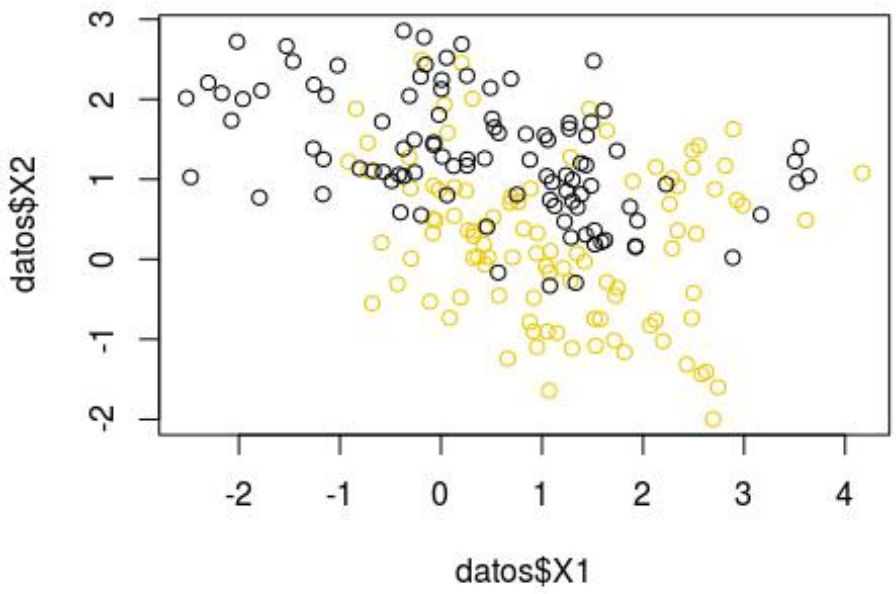
```
## [1] "x"        "y"        "xnew"     "prob"     "marginal" "px1"
## [7] "px2"      "means"
```

El objeto `ESL.mixture` cargado es una lista que contiene almacenados el valor de los dos predictores en el elemento x y el valor de la clase a la que pertenece cada observación en el elemento y.

```
attach(ESL.mixture)
datos <- data.frame(x, y)
datos$y <- as.factor(datos$y)
head(datos)
```

```
##            X1         X2 y
## 1  2.52609297  0.3210504 0
## 2  0.36695447  0.0314621 0
## 3  0.76821908  0.7174862 0
## 4  0.69343568  0.7771940 0
## 5 -0.01983662  0.8672537 0
## 6  2.19654493 -1.0230141 0
```

```
plot(x = datos$X1, y = datos$X2, col = c("gold2", "black")[datos$y])
```

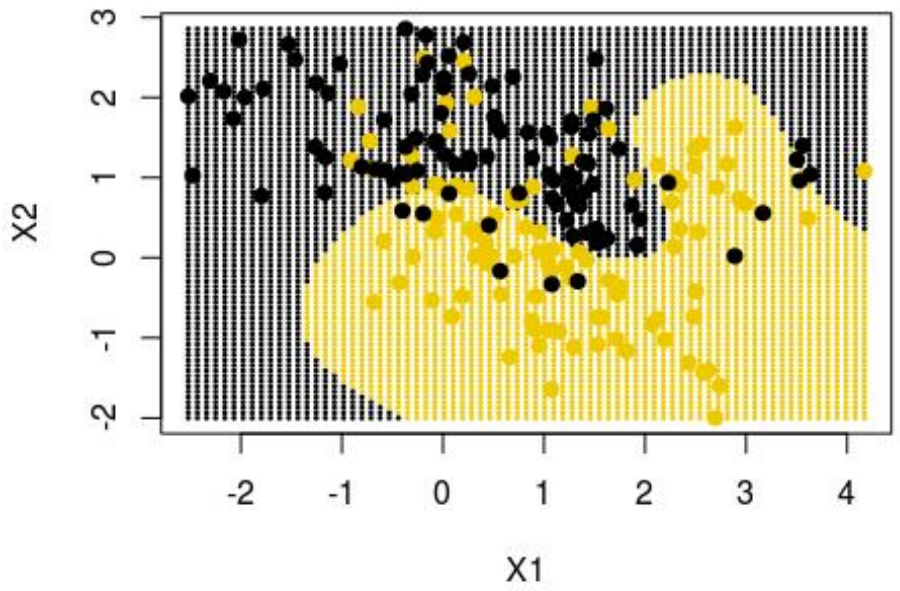



```
# Para obtener el clasificador smv se juntan las predictores y la variable
# respuesta en un único data frame y se convierte la variable respuesta en
# factor.


library(e1071)
modelo_svm <- svm(y ~ ., data = datos, sacle = FALSE, kernel = "radial", cost = 5,
                  gamma = 0.5)
summary(modelo_svm)
```

```
##
## Call:
## svm(formula = y ~ ., data = datos, sacle = FALSE, kernel = "radial",
##     cost = 5, gamma = 0.5)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  5
##       gamma:  0.5
##
## Number of Support Vectors:  110
##
##  ( 55 55 )
```

```
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```r
# Se obtiene un 'grid' de puntos para representar las regiones en las que se
# ha dividido el espacio acorde al modelo

# Rango de los predictores
rango_X1 <- range(datos$X1)
rango_X2 <- range(datos$X2)

# Interpolación de puntos
new_x1 <- seq(from = rango_X1[1], to = rango_X1[2], length = 75)
new_x2 <- seq(from = rango_X2[1], to = rango_X2[2], length = 75)
nuevos_puntos <- expand.grid(X1 = new_x1, X2 = new_x2)

# Predicción de los nuevos puntos según el modelo
predicciones <- predict(object = modelo_svm, newdata = nuevos_puntos)

# Representación de las 2 regiones empleando los puntos y coloreándolos
# según la clase predicha por el modelo
plot(nuevos_puntos, col = c("gold2", "black")[predicciones], pch = 20, cex = 0.35)

# Se añaden las observaciones
points(x = datos$X1, y = datos$X2, col = c("gold2", "black")[datos$y], pch = 19)
```



Al emplear un clasificador *Vector Support Machine* que emplea un función *kernel* radial, no solo hay que encontrar el valor óptimo de penalización, sino también el de γ. Ambos valores pueden encontrarse mediante *cross-validation* utilizando la función `tune()`.

```
set.seed(1)
svm_cv <- tune("svm", y ~ X1 + X2, data = datos, kernel = "radial", ranges =
list(cost = c(0.001,
    0.01, 0.1, 1, 5, 10, 100), gamma = c(0.5, 1, 2, 3, 4)))
summary(svm_cv)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1     4
##
## - best performance: 0.15
##
## - Detailed performance results:
##     cost gamma error dispersion
## 1  1e-03   0.5 0.600 0.06236096
## 2  1e-02   0.5 0.600 0.06236096
## 3  1e-01   0.5 0.285 0.08514693
## 4  1e+00   0.5 0.215 0.08181958
## 5  5e+00   0.5 0.200 0.07817360
## 6  1e+01   0.5 0.210 0.09067647
## 7  1e+02   0.5 0.195 0.08959787
## 8  1e-03   1.0 0.585 0.09732534
## 9  1e-02   1.0 0.585 0.09732534
## 10 1e-01   1.0 0.255 0.09264628
## 11 1e+00   1.0 0.190 0.08755950
## 12 5e+00   1.0 0.180 0.07527727
## 13 1e+01   1.0 0.180 0.08563488
## 14 1e+02   1.0 0.165 0.09732534
## 15 1e-03   2.0 0.585 0.09732534
## 16 1e-02   2.0 0.585 0.09732534
## 17 1e-01   2.0 0.220 0.08881942
## 18 1e+00   2.0 0.165 0.07472171
## 19 5e+00   2.0 0.160 0.08096639
## 20 1e+01   2.0 0.165 0.09442810
## 21 1e+02   2.0 0.185 0.09442810
## 22 1e-03   3.0 0.585 0.09732534
## 23 1e-02   3.0 0.585 0.09732534
## 24 1e-01   3.0 0.210 0.10488088
## 25 1e+00   3.0 0.155 0.07619420
## 26 5e+00   3.0 0.170 0.08881942
```

```
## 27 1e+01   3.0 0.160 0.09368980
## 28 1e+02   3.0 0.205 0.07245688
## 29 1e-03   4.0 0.580 0.11105554
## 30 1e-02   4.0 0.580 0.11105554
## 31 1e-01   4.0 0.210 0.11005049
## 32 1e+00   4.0 0.150 0.07071068
## 33 5e+00   4.0 0.165 0.08834906
## 34 1e+01   4.0 0.175 0.08249579
## 35 1e+02   4.0 0.220 0.10327956
```

De entre todos los modelos estudiados, empleando `cost = 1` y `gamma = 4` se logra el menor *cv-test-error*.

Support Vector Machines para más de dos clases

El concepto de hiperplano de separación en el que se basan los *SVMs* no se generaliza de forma natural para más de dos clases. Se han desarrollado numerosas estrategias con el fin de aplicar este método de clasificación a situaciones con $k>2$-clases, de entre ellos, los más empleados son *one-versus-one* y *one-versus-all*.

One-versus-one

Supóngase un escenario en el que hay $K > 2$ clases y que se quiere aplicar el método de clasificación basado en *SVMs*. La estrategia de *one-versus-one* consiste en generar un total de $\binom{K}{2}$ *SVMs*, comparando todos los posibles pares de clases. Para cada nueva observación se emplea cada uno de los $\binom{K}{2}$ clasificadores, registrando el número de veces que la observación es asignada a cada una de las clases. Finalmente, se considera que la observación pertenece a la clase a la que ha sido asignada con más frecuencia.

Si a la función `svm()` recibe como variable respuesta un factor con más de dos niveles, realiza automáticamente una clasificación multi-clase empleando el método *one-versus-one*.

One-versus-all

Esta estrategia consiste en ajustar K *SVMs* distintos, cada uno comparando una de las K clases frente a las restantes $K-1$ clases. Siendo (x^*) una nueva observación y $f(x^*) = \beta_0 + \beta_1 x_1^* + \beta_2 x_2^* + \ldots + \beta_p x_p^*$ la función que determina como de lejos está (x^*) del hiperplano de separación, la observación se asigna aquella clase para la que $f(x^*)$ es mayor. Como se mencionó en anterioridad, la magnitud de $f(x^*)$ es proporcional a la confianza de que la observación pertenezca a la clase en cuestión.

Ejemplo

El set de datos `khan` contiene información sobre 83 muestras pertenecientes a 4 tipos distintos de tumores. Para cada una de las muestras se dispone del perfil de expresión de 2308 genes. Se pretende crear un clasificador basado en *SVMs* que permita predecir el tipo de tumor en función de la expresión de los genes. El set de datos está dividido en *training set* (`xtrain`, `ytrain`) y *test set* (`xtest`, `ytest`), 63 observaciones destinadas a entrenamiento y 20 a la evaluación del modelo.

```
library(ISLR)

data("Khan")
names(Khan)
```

```
## [1] "xtrain" "xtest"  "ytrain" "ytest"
```

```
dim(Khan$xtrain)
```

```
## [1]   63 2308
```

```
dim(Khan$xtest)
```

```
## [1]   20 2308
```

```
length(Khan$ytrain)
```

```
## [1] 63
```

```
length(Khan$ytest)
```

```
## [1] 20
```

En este tipo de escenario, en el que el número de predictores es varios órdenes de magnitud mayor que el de observaciones, los modelos son proclives a sufrir *overfitting*. Esto sugiere que, de entre los diferentes tipos de *kernels*, sea adecuado emplear el de menor flexibilidad, el *kernel lineal*.

```
library(e1071)

# Como la variable respuesta está separa de los predictores, se unen en un
# único data frame. La variable respuesta tiene que ser de tipo factor.
datos <- data.frame(y = as.factor(Khan$ytrain), Khan$xtrain)
modelo_svm <- svm(formula = y ~ ., data = datos, kernel = "linear", cost = 10)
summary(modelo_svm)
```

```
##
## Call:
## svm(formula = y ~ ., data = datos, kernel = "linear", cost = 10)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##       gamma:  0.0004332756
##
## Number of Support Vectors:  58
##
##  ( 20 20 11 7 )
##
##
## Number of Classes:  4
##
## Levels:
##  1 2 3 4
```

```
# Aciertos del modelo con los datos de entrenamiento
table(prediccion = modelo_svm$fitted, clase_real = datos$y)
```

```
##           clase_real
## prediccion  1  2  3  4
##          1  8  0  0  0
##          2  0 23  0  0
##          3  0  0 12  0
##          4  0  0  0 20
```

El modelo obtenido tiene un *training error* del 0%, es capaz de clasificar correctamente todas las observaciones empleadas para crearlo. Un *training error* muy bajo puede ser un indicativo de *overfitting*, lo que haría que el modelo no fuese capaz de predecir correctamente nuevas observaciones. Para evaluar si es este el caso, se emplea el modelo para predecir las 20 observaciones del *test set*.

```
nuevos_datos <- data.frame(y = as.factor(Khan$ytest), Khan$xtest)
predicciones <- predict(object = modelo_svm, newdata = nuevos_datos)
table(prediccion = predicciones, clase_real = nuevos_datos$y)
```

```
##           clase_real
## prediccion 1 2 3 4
##          1 3 0 0 0
##          2 0 6 2 0
##          3 0 0 4 0
##          4 0 0 0 5
```

De las 20 nuevas observaciones, el modelo *SVM* con `kernel = "linear"` y `cost = 10` predice correctamente 18 y falla en 2, su *test error* es solo del 10%.

Relación entre Support Vector Machines y Regresión Logística

Cuando el método de *SVMs* se introdujo por primera vez a mediados de 1990, se consideró totalmente novedoso. La idea de encontrar un hiperplano que separase los datos de la forma más precisa posible, permitiendo ciertas violaciones en la clasificación final, parecía una aproximación totalmente distinta a los métodos clásicos. Sin embargo, con el tiempo, se han ido encontrando conexiones que demuestran su relación con otros métodos tales como la regresión logística y el análisis lineal discriminante. Sin entrar en demostraciones matemáticas (ver libro *ISLR*), el hecho de que un clasificador basado en *SVMs* solo dependa de los vectores soporte (observaciones en el lado incorrecto del margen) y de que un clasificador basado en regresión logística se vea muy poco influenciado por observaciones alejadas del límite de decisión, hace que ambos métodos consigan resultados muy parecidos. Cuando las clases están bien separadas, *SVMs* suele alcanzar mejor precisión; cuando hay solapamiento, la regresión logística suele ser superior.

Bibliografía

Introduction to Statistical Learning Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani

Tutorial sobre Máquinas de Vector Soporte (SVM) Enrique J. Carmona Suárez

A Gentle Introduction to Support Vector Machines in Biomedicine Alexander Statnikov, Constantin F Aliferis