Máquinas de Vector Soporte (Support Vector Machines, SVMs)

Joaquín Amat Rodrigo j.amatrodrigo@gmail.com

Abril, 2017

Tabla de contenidos

Introducción 2
Hiperplano y Maximal Margin Classifier 2
Clasificación binaria empleando un hiperplano 3
Support Vector Classifier o Soft Margin SVM 7
Ejemplo 9
Máquinas de Vector Soporte 16
Aumento de la dimensión, kernels 17
Ejemplo 20
Support Vector Machines para más de dos clases 24
One-versus-one 24
One-versus-all 24
DAGSVM 25
Ejemplo 25
Apuntes varios (miscellaneous) 28
Algoritmo Perceptron 28
Bibliografía 35

Formato PDF: Github

Introducción

El método de clasificación-regresión Máquinas de Vector Soporte (*Vector Support Machines, SVMs*) fue desarrollado en la década de los 90, dentro de campo de la ciencia computacional. Si bien originariamente se desarrolló como un método de clasificación binaria, su aplicación se ha extendido a problemas de clasificación múltiple y regresión. *SVMs* ha resultado ser uno de los mejores clasificadores para un amplio abanico de situaciones, por lo que se considera uno de los referentes dentro del ámbito de aprendizaje estadístico y *machine learning*.

Las Máquinas de Vector Soporte se fundamentan en el *Maximal Margin Classifier*, que a su vez, se basa en el concepto de hiperplano. A lo largo de este ensayo se introducen por orden cada uno de estos conceptos. Comprender los fundamentos de las *SVMs* requiere de conocimientos sólidos en álgebra lineal. En este ensayo no se profundiza en el aspecto matemático, pero puede encontrarse una descripción detallada en el libro *Support Vector Machines Succinctly by Alexandre Kowalczyk*

En `R`, las librerías `e1071` y `LiblineaR` contienen los algoritmos necesarios para obtener modelos de clasificación simple, múltiple y regresión, basados en *Support Vector Machines*.

Hiperplano y Maximal Margin Classifier

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 de 1 dimensión, es decir, una recta. En un espacio tridimensional, un hiperplano es un subespacio de dos dimensiones, un plano convencional. Para dimensiones $p > 3$ no es intuitivo visualizar un hiperplano, pero el concepto de subespacio 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 $\mathbf{x} = (x_1, x_2)$ 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 ($\mathbf{x} = x_1, x_2, \ldots, x_p$) 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 en qué lado del hiperplano se encuentra un determinado punto **x**, solo hay que 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 $1 + 2x_1 + 3x_2 < 0$.

Clasificación binaria 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:

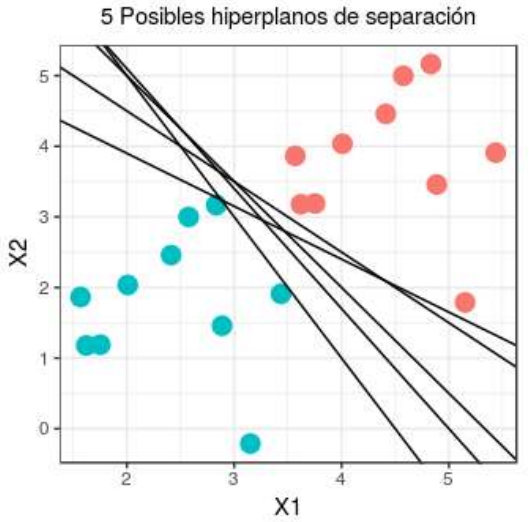
$$y_i(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p) > 0, \;\; para \; i = 1 \ldots n$$

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 $\mathbf{x}^{\wedge} *$ se clasifica acorde al signo de la función $f(\mathbf{x}^{\wedge} *) = \beta_0 + \beta_1 x_1^* + \beta_2 x_2^* + \ldots + \beta_p x_p^*$. 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), size = 4) +
  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))
```



La solución a este problema consiste en seleccionar como clasificador óptimo al que se conoce como *maximal margin hyperplane* o *hiperplano óptimo de separación*, que se corresponde con el hiperplano que se encuentra más alejado de todas las observaciones de entrenamiento. Para obtenerlo, se tiene que calcular la distancia perpendicular de cada observación a un determinado hiperplano. La menor de estas distancias (conocida como margen) determina como de alejado está el hiperplano de las observaciones de entrenamiento. El *maximal margin hyperplane* se define como el hiperplano que consigue un mayor margen, es decir, que la distancia mínima entre el hiperplano y las observaciones es lo más grande posible. Aunque esta idea suena razonable, no es posible aplicarla, ya que habría infinitos hiperplanos contra los que medir las distancias. En su lugar, se recurre a métodos de optimización. Para encontrar una descripción más detallada de la solución por optimización consultar (*Support Vector Machines Succinctly by Alexandre Kowalczyk*).

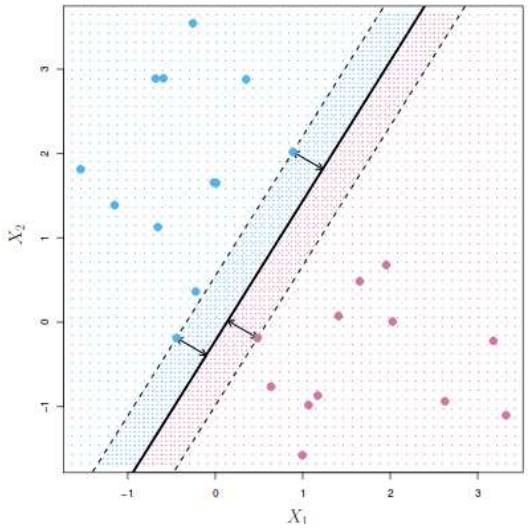


Imagen maximal margin hyperplane obtenida del libro ISLR

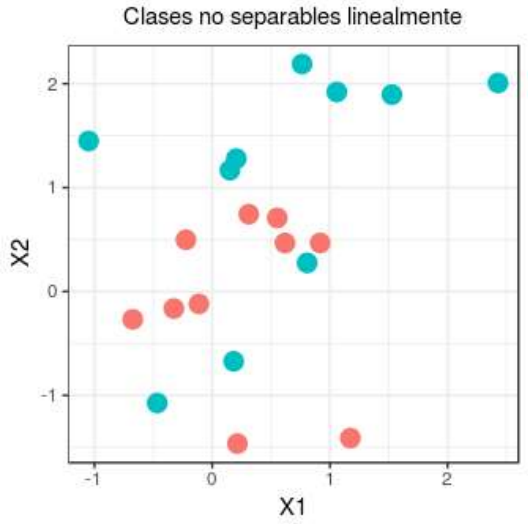
La imagen anterior muestra el *maximal margin hyperplane* para un conjunto de datos de entrenamiento. Las tres observaciones equidistantes respecto al *maximal margin hyperplane* se encuentran a lo largo de las líneas discontinuas que indican la anchura del margen. A estas observaciones se les conoce 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 forma 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 = as.factor(y))) +
  geom_point(size = 4) +
  theme_bw() +
  labs(title = "Clases no separables linealmente") +
  theme(legend.position = "none",
        plot.title = element_text(hjust = 0.5, size = 11))
```



Para solucionar estas situaciones, se puede extender el concepto de *maximal margin hyperplane* para obtener un hiperplano que casi separe las clases, pero permitiendo que cometa unos pocos errores. A este tipo de hiperplano se le conoce como *Support Vector Classifier* o *Soft Margin*.

Support Vector Classifier o Soft Margin SVM

El *Maximal Margin Classifier* descrito en la sección anterior tiene poca aplicación práctica, ya que rara vez 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.

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.

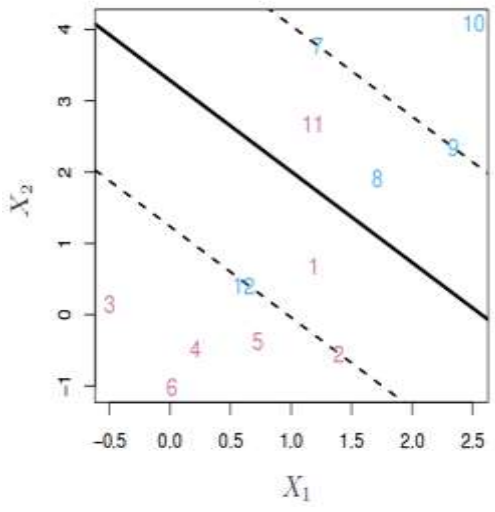


Imagen clasificador vector soporte obtenida del libro ISLR

La identificación 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 convexa. Si bien la demostración matemática queda fuera del objetivo de esta introducción, es importante mencionar que el proceso incluye un hiperparámetro 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 $C = \infty$, no se permite ninguna violación del margen y por lo tanto, el resultado es equivalente al *Maximal Margin Classifier* (teniendo en cuenta que esta solución solo es posible si las clases son perfectamente separables). Cuando más se aproxima C a cero, menos se penalizan los errores y más observaciones pueden estar en el lado incorrecto del margen o incluso del hiperplano. C es a fin de cuentas el hiperparámetro 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. A estas observaciones se les conoce como vectores soporte y son las que definen el clasificador obtenido. Esta es la razón por la que el parámetro C controla el balance entre *bias* y varianza. Cuando el valor de C es pequeño, 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 mayor es el valor de C, menor el margen, menos observaciones serán vectores soporte y el clasificador resultante tendrá menor *bias* pero mayor varianza.

Otra propiedad importante que deriva de que el hiperplano 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.

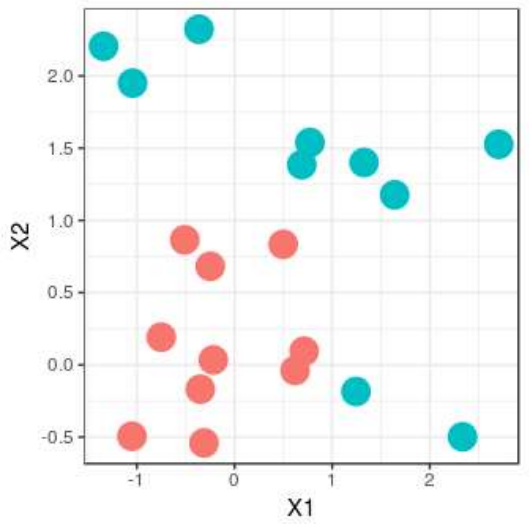
Nota: en el libro Introduction to Statistical Learning se emplea un término C que equivale a la inversa del descrito en este ensayo.

Ejemplo

Para mostrar el uso de un Support Vector Classifier como clasificador binario, se simulan observaciones en un espacio bidimensional que pertenecen a dos clases. Este ejemplo se ha obtenido de los videos asociados al libro Introduction to Statistical Learning, que no es igual al presentado en el libro.

En los siguientes ejemplos, se emplea la función `svm()` contenida en el paquete `e1071`. Esta función ajusta *Support Vector Classifier* si se le especifica el argumento `kernel="linear"` (como se describe más adelante, 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, es el nombre que emplea esta función para el hiperparámetro C.

```
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)
ggplot(data = datos, aes(x = X1, y = X2, color = as.factor(y))) +
 geom_point(size = 6) +
 theme_bw() + theme(legend.position = "none")
```



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

La función `svm()` identifica automáticamente si se trata de un problema de clasificación, la variable respuesta es de tipo factor, o de regresión, la variable respuesta es tipo numérico.

```
library(e1071)
# Se convierte la variable respuesta a factor
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, de lo contrario, los predictores de mayor magnitud eclipsarán a los de menor magnitud.

El `summary` del modelo muestra que hay un total de 6 vectores soporte, 3 pertenecen a una clase y 3 a la otra.

```
summary(modelo_svm)
```

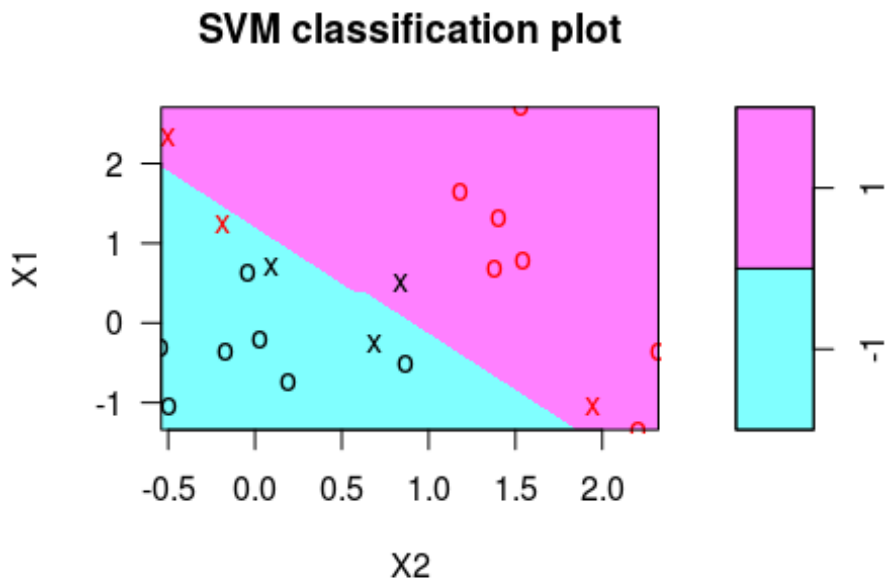
```
##
## Call:
## svm(formula = y ~ X1 + X2, data = datos, kernel = "linear", cost = 10,
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##       gamma:  0.5
##
## Number of Support Vectors:  6
##
##  ( 3 3 )
##
##
## Number of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

```
# Índice de las observaciones que actúan com vector soporte
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.

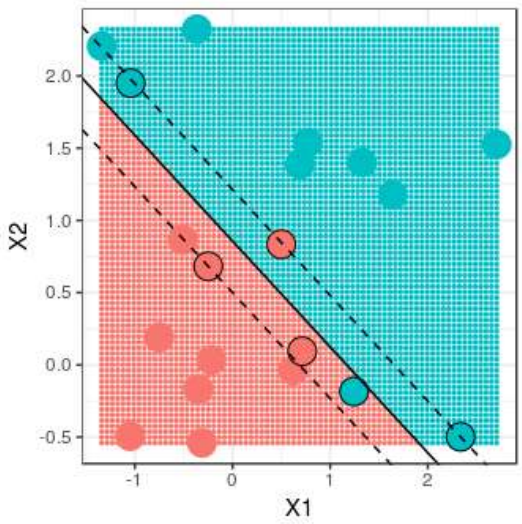
```
plot(modelo_svm, datos)
```



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. A continuación, se muestra una alternativa basada en `ggplot2` para obtener una representación de los resultados de `svm()` cuando el espacio es bidimensionales.

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

# Se 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 según el modelo
predicciones <- predict(object = modelo_svm, newdata = nuevos_puntos)

# Se almacenan los puntos predichos para dar color a las regiones
color_regiones <- data.frame(nuevos_puntos, y = predicciones)

# 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

ggplot() +
 # Representación de las 2 regiones empleando los puntos y coloreándolos
 # según la clase predicha por el modelo
geom_point(data = color_regiones, aes(x = X1, y = X2, color = as.factor(y)),
    size = 0.5) +
 # Se añaden las observaciones
geom_point(data = datos, aes(x = X1, y = X2, color = as.factor(y)), size = 6) +
 # Se identifican aquellas observaciones que son vectores soporte del modelo
geom_point(data = datos[modelo_svm$index, ], aes(x = X1, y = X2,
           color = as.factor(y)), shape = 21, colour = "black", size = 6) +
 # Se añaden las rectas del hiperplano y los márgenes
geom_abline(intercept = beta0/beta[2], slope = -beta[1]/beta[2]) +
geom_abline(intercept = (beta0 - 1)/beta[2], slope = -beta[1]/beta[2],
            linetype = "dashed") +
geom_abline(intercept = (beta0 + 1)/beta[2], slope = -beta[1]/beta[2],
            linetype = "dashed") +
theme_bw() +
theme(legend.position = "none")
```

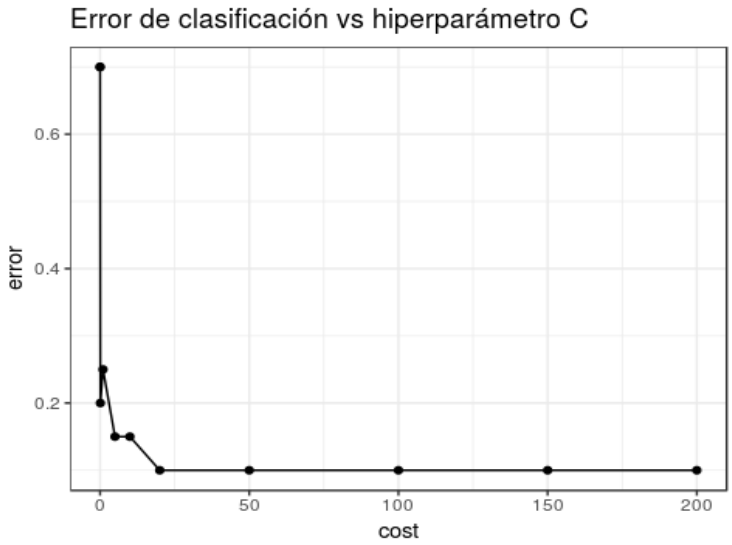


En el ajuste anterior se ha empleado un valor del hiperparámetro de penalización `cost = 10`. Este hiperparámetro determina el balance bias-varianza y por lo tanto, es crítico para la capacidad predictiva del modelo. El paquete e1071 incluye la función `tune()` que realiza *10-cross-validation* para identificar el valor óptimo de penalización. Entre sus argumentos están: el modelo `svm` y un vector `ranges` con los valores de los hiperparámetros 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, 20, 50, 100, 150,
                                      200)))
summary(svm_cv)
```

```
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    20
##
## - best performance: 0.1
##
## - Detailed performance results:
##       cost error dispersion
## 1    0.001  0.70  0.4216370
## 2    0.010  0.70  0.4216370
## 3    0.100  0.20  0.2581989
## 4    1.000  0.25  0.2635231
## 5    5.000  0.15  0.2415229
## 6   10.000  0.15  0.2415229
## 7   20.000  0.10  0.2108185
## 8   50.000  0.10  0.2108185
## 9  100.000  0.10  0.2108185
## 10 150.000  0.10  0.2108185
## 11 200.000  0.10  0.2108185
```

```
ggplot(data = svm_cv$performances, aes(x = cost, y = error)) + geom_line() +
    geom_point() + labs(title = "Error de clasificación vs hiperparámetro C") +
    theme_bw()
```



El proceso de *cross-validation* muestra que el valor de penalización con el que se consigue menor *error rate* es 20 o superior. La función `tune()` almacena el mejor modelo de entre todos los que se han comparado:

```
mejor_modelo <- svm_cv$best.model
```

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)

# Predicciones
predicciones <- predict(object = mejor_modelo, test)
paste("Error de test:", 100 * mean(test$y != predicciones), "%")
```

```
## [1] "Error de test: 10 %"
```

```
table(predicción = predicciones, valor_real = test$y)
```

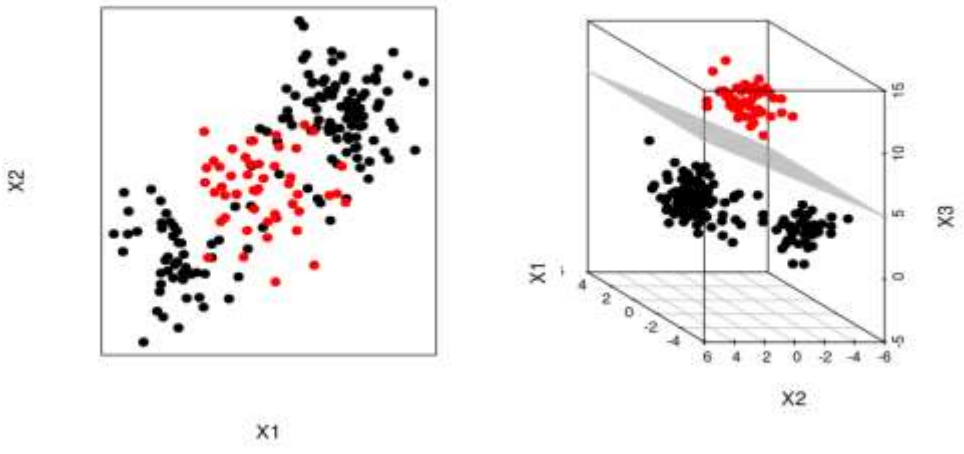
```
##           valor_real
## predicción -1  1
##         -1  7  1
##         1   1 11
```

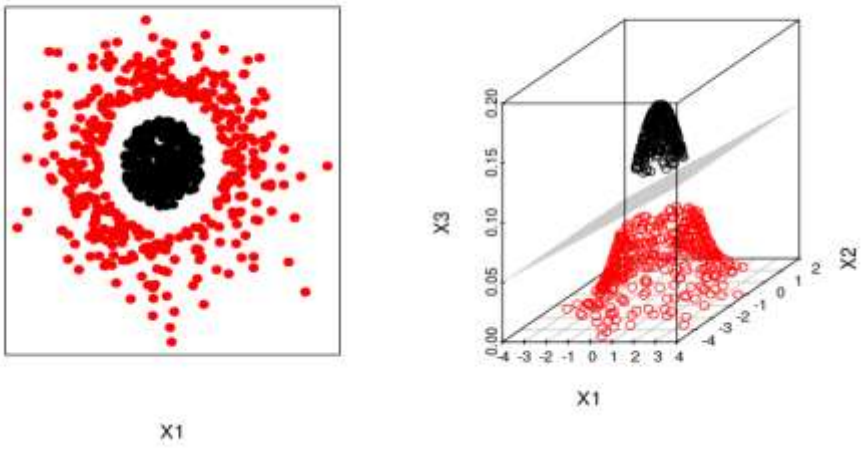
18 de las 20 observaciones han sido clasificadas correctamente por el modelo. El test error es del 10 %.

Máquinas de Vector Soporte

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.





El método de Máquinas Vector Soporte (*SVM*) se puede considerar como una extensión del *Support Vector Classifier* obtenida al aumentar la dimensión de los datos. Los límites de separación lineales generados en el espacio aumentado se convierten en límites de separación no lineales al proyectarlos en el espacio original.

Aumento de la dimensión, kernels

Una vez definido que las Máquinas de Vector Soporte siguen la misma estratégica que el *Support Vector Classifier*, pero aumentando la dimensión de los datos antes de aplicar el algoritmo, la pregunta inmediata es ¿Cómo se aumenta la dimensión y qué dimensión es la correcta?

La dimensión de un conjunto de datos puede transformarse combinando o modificando cualquiera de sus dimensiones. Por ejemplo, se puede transformar un espacio de dos dimensiones en uno de tres aplicando la siguiente función:

$$f(x_1, x_2) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

Esta es solo una de las infinitas trasformaciones posibles, ¿Cómo saber cuál es la adecuada? Es aquí donde los *kernel* entran en juego. Un *kernel (K)* es una función que devuelve el resultado del *dot product* entre dos vectores realizado en un nuevo espacio dimensional distinto al espacio original en el que se encuentran los vectores. Aunque no se ha entrado en detalle en las fórmulas matemáticas empleadas para resolver el problema de optimización, esta contiene un

dot product. Si se sustituye este *dot product* por un *kernel*, se obtienen directamente los vectores soporte (y el hiperplano) en la dimensión correspondiente al *kernel*. Ha esto se le suele conocer como *kernel trick*, porque con solo una ligera modificación del problema original, gracias a los *kernels* se puede obtener el resultado para cualquier dimensión. Existen multitud de *kernels* distintos, algunos de los más utilizados son:

Kernel lineal

$$K(\mathbf{x}, \mathbf{x}') = \mathbf{x} \cdot \mathbf{x}'$$

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

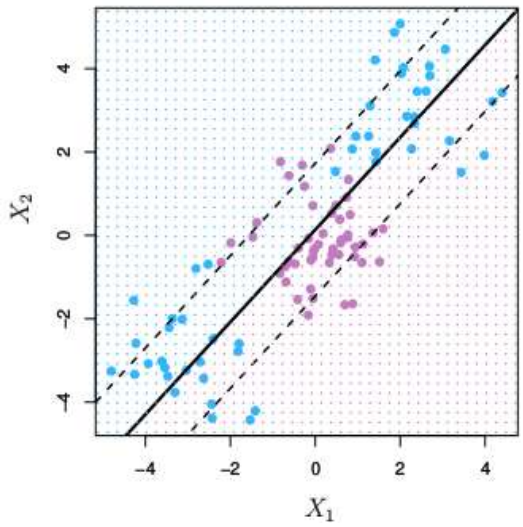


Imagen SVM con kernel lineal obtenida del libro ISLR

Kernel polinómico

$$K(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}' + c)^d$$

Cuando se emplea $d = 1$ y $c = 0$, el resultado es el mismo que el de un *kernel* lineal. Si $d > 1$, se generan límites de decisión no lineales, aumentando la no linealidad a medida que aumenta d. No suele ser recomendable emplear valores de d mayores 5 por problemas de *overfitting*.

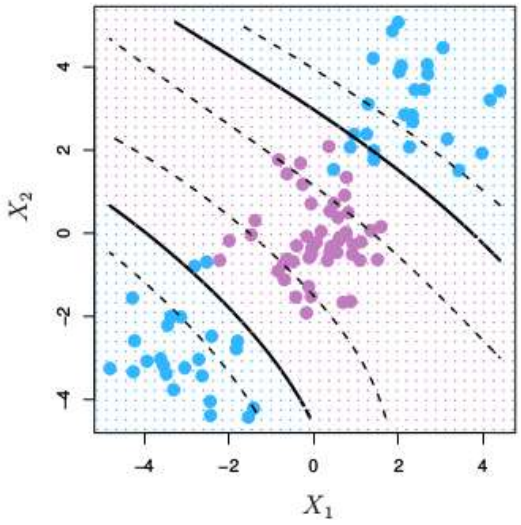


Imagen SVM con una kernel polinómico de grado 3 obtenida del libro ISLR

Gaussian Kernel (RBF)

$$K(\mathbf{x}, \mathbf{x}') = exp(-\gamma||\mathbf{x} - \mathbf{x}'||^2)$$

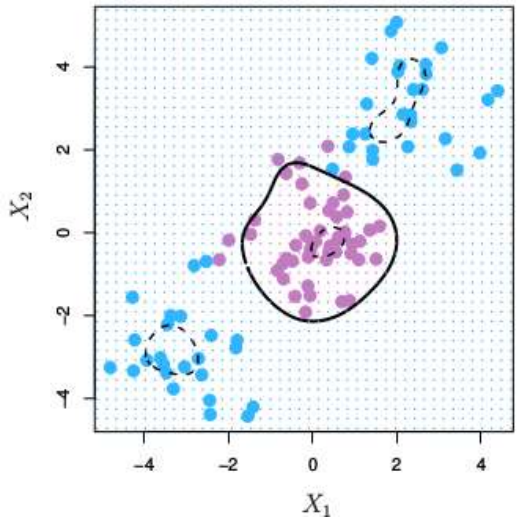


Imagen SVM con una kernel radial obtenida del libro ISLR

El valor de γ controla el comportamiento del *kernel*, cuando es muy pequeño, el modelo final es equivalente al obtenido con un *kernel* lineal, a medida que aumenta su valor, también lo hace la flexibilidad del modelo.

Los *kernels* descritos son solo unos pocos de los muchos que existen. Cada uno tiene una serie de hiperparámetros cuyo valor óptimo puede encontrarse mediante validación cruzada. No puede decirse que haya un *kernel* que supere al resto, depende en gran medida de la naturaleza del problema que se esté tratando. Ahora bien, tal como indican los autores de *A Practical Guide to Support Vector Classification*, es muy recomendable probar el *kernel RBF*. Este *kernel* tiene dos ventajas: que solo tiene dos hiperparámetros que optimizar (γ y la penalización C común a todos los *SVM*) y que su flexibilidad puede ir desde un clasificador lineal a uno muy complejo.

Ejemplo

Para el siguiente ejemplo se emplea 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).

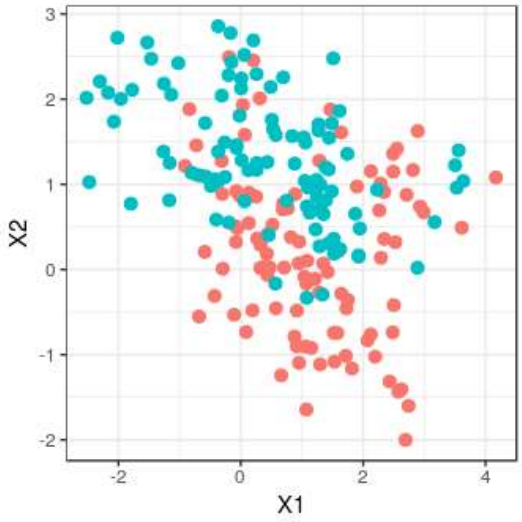
```
# Descargan los datos. Requiere conexión a internet
load(url("https://web.stanford.edu/~hastie/ElemStatLearn/datasets/ESL.mixture.rda"))
```

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.

```
datos <- data.frame(ESL.mixture$x, y = ESL.mixture$y)
# Al tratarse de un problema de clasificación se convierte la variable
# respuesta en factor
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
```

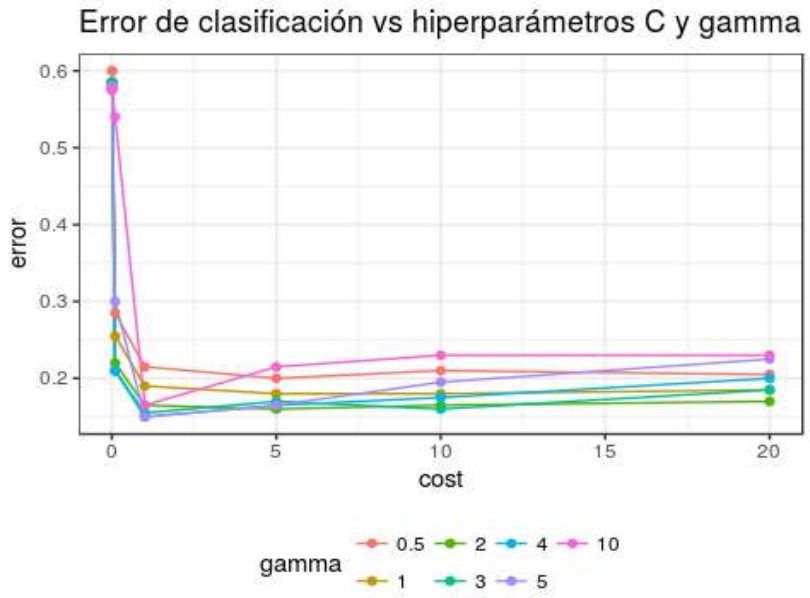
```
ggplot(data = datos, aes(x = X1, y = X2, color = y)) + geom_point(size = 2.5) +
    theme_bw() + theme(legend.position = "none")
```



Entre los *kernel* no lineales que acepta la función `svm()` destacan `kernel = "polinomial"`, en cuyo caso hay que indicar el grado del polinomio d, y `kernel = "radial"`, en cuyo caso hay que indicar el hiperparámetro parámetro γ. Además de los hiperparámetros propios de cada *kernel*, todo *SVM* tiene también el hiperparámetro de penalización C.

Identificación de hiperparámetros óptimos y ajuste del modelo.

```
library(e1071)
# Como los datos se han simulado en una misma escala, no es necesario
# estandarizarlos si no fuese así, es muy importante hacerlo.
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, 20),
                             gamma = c(0.5, 1, 2, 3, 4, 5, 10)))

ggplot(data = svm_cv$performances, aes(x = cost, y = error, color=factor(gamma))) +
  geom_line() +
  geom_point() +
  labs(title = "Error de clasificación vs hiperparámetros C y gamma",
       color = "gamma") +
  theme_bw() + theme(legend.position = "bottom")
```




```
svm_cv$best.parameters
```

```
##    cost gamma
## 32    1     4
```

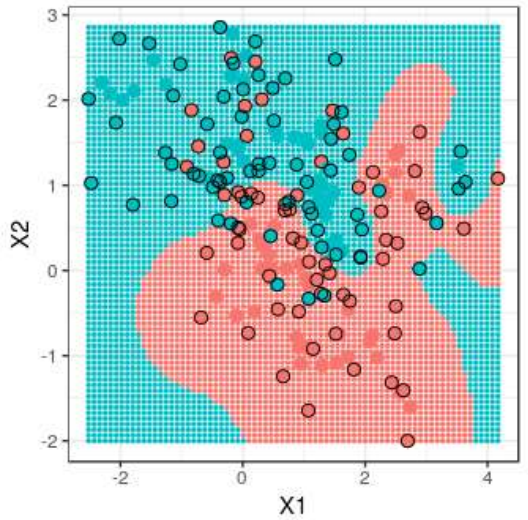
```
modelo_svm_rbf <- svm_cv$best.model
```

De entre todos los modelos estudiados, empleando `cost = 1` y `gamma = 5` se logra el menor *cv-test-error*.

Representación gráfica del clasificador *SVM*.

```
# Se 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 según el modelo de los nuevos puntos
predicciones <- predict(object = modelo_svm_rbf, newdata = nuevos_puntos)
```

```
# Se almacenan los puntos predichos para el color de las regiones en un
# dataframe
color_regiones <- data.frame(nuevos_puntos, y = predicciones)

ggplot() +
 # Representación de las 2 regiones empleando los puntos y coloreándolos
 # según la clase predicha por el modelo
geom_point(data = color_regiones, aes(x = X1, y = X2, color = as.factor(y)),
           size = 0.5) +
 # Se añaden las observaciones
geom_point(data = datos, aes(x = X1, y = X2, color = as.factor(y)), size = 2.5) +
 # Se identifican aquellas observaciones que son vectores soporte
geom_point(data = datos[modelo_svm_rbf$index, ], aes(x = X1, y = X2,
           color = as.factor(y)), shape = 21, colour = "black", size = 2.5) +
theme_bw() + theme(legend.position = "none")
```



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*, *one-versus-all* y *DAGSVM*.

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 $K(K-1)/2$ *SVMs*, comparando todos los posibles pares de clases. Para generar una predicción se emplean cada uno de los $K(K-1)/2$ 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. La principal desventaja de esta estrategia es que el número de modelos necesarios se dispara a medida que aumenta el número de clases, por lo que no es aplicable en todos los escenarios.

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. Como resultado, se obtiene un hiperplano de clasificación para cada clase. Para obtener una predicción, se emplean cada uno de los K clasificadores y se asigna la observación a la clase para la que la predicción resulte positiva. Esta aproximación, aunque sencilla, puede causar inconsistencias, ya que puede ocurrir que más de un clasificador resulte positivo, asignando así una misma observación a diferentes clases. Otro inconveniente adicional es que cada clasificador se entrena de forma no balanceada. Por ejemplo, si el set de datos contiene 100 clases con 10 observaciones por clase, cada clasificador se ajusta con 10 observaciones positivas y 990 negativas.

DAGSVM

DAGSVM (*Directed Acyclic Graph SVM*) es una mejora del método *one-versus-one*. La estrategia seguida es la misma, pero consiguen reducir su tiempo de ejecución eliminando comparaciones innecesarias gracias al empelo de una *directed acyclic graph (DAG)*. Supóngase un set de datos con cuatro clases (A, B, C, D) y 6 clasificadores entrenados con cada posible par de clases (A-B, A-C, A-D, B-C B-D, C-D). Se inician las comparaciones con el clasificador (A-D) y se obtiene como resultado que la observación pertenece a la clase A, o lo que es equivalente, que no pertenece a la clase D. Con esta información se pueden excluir todas las comparaciones que contengan la clase D, puesto que se sabe que no pertenece a este grupo. En la siguiente comparación se emplea el clasificador (A-C) y se predice que es A. Con esta nueva información se excluyen todas las comparaciones que contengan C. Finalmente solo queda emplear el clasificador (A-B) y asignar la observación al resultado devuelto. Siguiendo esta estrategia, en lugar de emplear los 6 clasificadores, solo ha sido necesario emplear 3. *DAGSVM* tiene las mismas ventajas que el método *one-versus-one* pero mejorando mucho el rendimiento.

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 multiclase 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)
```

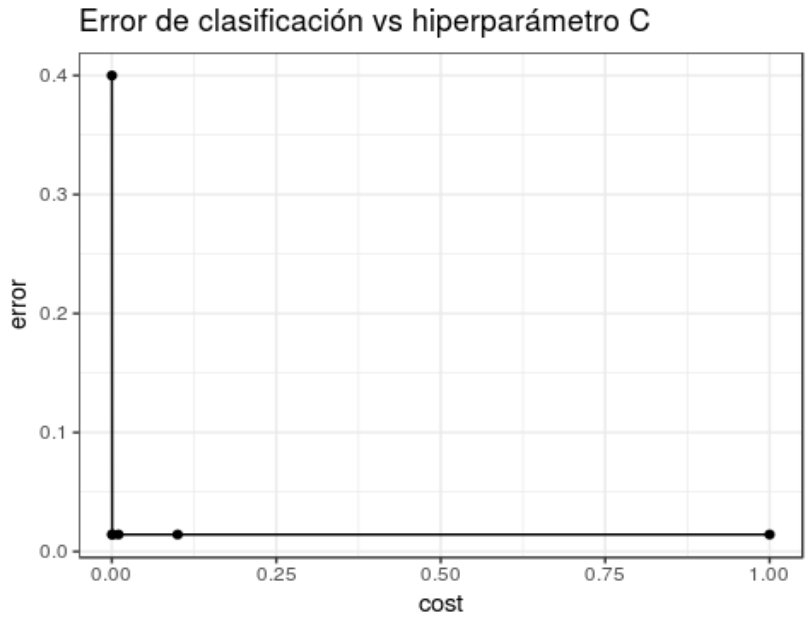
```
## [1] 63
```

```
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 *overffiting*. Esto sugiere que, de entre los diferentes tipos de *kernels*, sea adecuado emplear el de menor flexibilidad, el *kernel* lineal. El único hiperparámetro de un *SVM* con *kernel* lineal es el valor de penalización C.

```
library(e1071)

# Como la variable respuesta está separa de los predictores, se unen en un
# único dataframe. La variable respuesta tiene que ser de tipo factor.
datos_train <- data.frame(y = as.factor(Khan$ytrain), Khan$xtrain)

svm_cv <- tune("svm", y ~ ., data = datos_train, kernel = "linear",
               ranges = list(cost = c(1e-04, 5e-04, 0.001, 0.01, 0.1, 1)))

ggplot(data = svm_cv$performances, aes(x = cost, y = error)) + geom_line() +
    geom_point() + labs(title = "Error de clasificación vs hiperparámetro C") +
    theme_bw()
```



```
svm_cv$best.parameters
```

```
##    cost
## 2 5e-04
```

```
modelo_svm <- svm_cv$best.model

# Aciertos del modelo con los datos de entrenamiento
paste("Error de entrenamiento:", 100*mean(datos_train$y != modelo_svm$fitted), "%")
```

```
## [1] "Error de entrenamiento: 0 %"
```

```
table(prediccion = modelo_svm$fitted, clase_real = datos_train$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*.

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

paste("Error de test:", 100 * mean(datos_test$y != predicciones), "%")
```

```
## [1] "Error de test: 10 %"
```

```
table(prediccion = predicciones, clase_real = datos_test$y)
```

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

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

Apuntes varios (miscellaneous)

En este apartado recojo comentarios, definiciones y puntualizaciones que he ido encontrando en diferentes fuentes y que, o bien no he tenido tiempo de introducir en el cuerpo principal del documento, o que he considerado mejor mantenerlos al margen como información complementaria.

Algoritmo Perceptron

El algoritmo *Perceptron* fue publicado en 1957 por Frank Rosenblatt. El objetivo del *Perceptron* es encontrar un hiperplano capaz de separar correctamente un conjunto de datos que sean linealmente separables, una vez obtenido el hiperplano, este puede utilizarse para clasificaciones binarias. Aunque el *Perceptron* es un algoritmo de aprendizaje muy simple, entender su funcionamiento es clave para aprender otros métodos más complejos como las *máquinas de vector soporte* o las *redes neuronales artificiales*.

Antes de describir en detalle el algoritmo, conviene conocer una serie de términos matemáticos: el producto escalar o *dot product*, el concepto de hiperplano y el concepto de linealmente separable.

PRODUCTO ESCALAR O *DOT PRODUCT*

El *dot product* es una operación entre dos vectores (de la misma dimensión) que devuelve un único valor (escalar) con información sobre la relación entre ambos vectores. Existen dos formas de interpretar el *dot product*: geométrica y algebraica.

Interpretación geométrica

Geométricamente, el *dot product* se define como el producto entre la magnitud euclídea (módulo o segunda norma) de dos vectores y el coseno del ángulo que forman. Supóngase dos vectores **x** e **y** que forman un ángulo α entre ellos. Su *dot product* es:

$$\mathbf{x} \cdot \mathbf{y} = ||\mathbf{x}||\ ||\mathbf{y}||\ cos(\alpha)$$

El *dot product* está por lo tanto influenciado por el ángulo que forman los dos vectores:

- Si $\alpha = 0$, $cos(\alpha) = 1$ y $\mathbf{x} \cdot \mathbf{y} = ||\mathbf{x}||\ ||\mathbf{y}||$. Esto significa que, si los dos vectores tienen exactamente la misma dirección, *el dot product* equivale a la multiplicación de sus magnitudes.
- Si $\alpha = 90$, $cos(\alpha) = 0$ y $\mathbf{x} \cdot \mathbf{y} = 0$. Esto significa que, si los dos vectores son perpendiculares, *el dot product* es cero. Los dos vectores tienen cero relación.
- Si $\alpha = 180$, $cos(\alpha) = -1$ y $\mathbf{x} \cdot \mathbf{y} = -||\mathbf{x}||\ ||\mathbf{y}||$. Esto significa que, si los dos vectores tienen direcciones opuestas, *el dot product* equivale al valor negativo de la multiplicación de sus magnitudes.

```
# Implementación del dot product (geométrico)
dot_product_gemometrico <- function(x, y, alpha){
  modulo_x <- sqrt(sum(x^2))
  modulo_y <- sqrt(sum(y^2))
  aplha_radianes <- (alpha * pi) / 180
  return(modulo_x * modulo_y * cos(aplha_radianes))
}

dot_product_gemometrico(x = c(3, 5), y = c(8, 2), alpha = 45)
```

```
## [1] 34
```

Interpretación algebraica

El *dot product* se define, desde el punto de vista del álgebra, como el sumatorio del producto de cada una de sus dimensiones.

$$\mathbf{x} \cdot \mathbf{y} = \mathbf{x}_1\mathbf{y}_1 + \mathbf{x}_2\mathbf{y}_2$$

o de forma genérica:

$$\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{n} \mathbf{x}_i\, \mathbf{y}_i$$

Con esta definición, no es necesario conocer el ángulo que forman los dos vectores.

```r
# Implementación del dot product (algebra)
dot_product_algebra <- function(x, y) {
  resultado <- 0
  if (length(x) != length(y)) {
    stop("La longitud de los dos vectores debe ser la misma")
  }

  for (i in seq_along(x)) {
    resultado <- resultado + x[i] * y[i]
  }
  return(resultado)
}

dot_product_algebra(x = c(3, 5), y = c(8, 2))
```

```
## [1] 34
```

LINEALMENTE SEPARABLE

El concepto de separación lineal se entiende fácilmente cuando se estudia en espacios de 2 o 3 dimensiones. En 2 dimensiones, que dos grupos de observaciones sean linealmente separables significa que existe al menos una recta que permite separar perfectamente los dos grupos. En el caso de 3 dimensiones, los datos son linealmente separables si existe un plano tal, que es capaz de separar perfectamente los grupos. Este mismo concepto puede generalizarse para cualquier número de dimensiones, la condición sigue siendo la misma, que exista un elemento de una dimensión menor que separe los grupos. Ha este elemento se le conoce como hiperplano.

HIPERPLANO

En geometría, un hiperplano se define como un subespacio con una dimensión menos que el espacio que lo rodea. Esta definición es poco intuitiva, pero se entiende bien si se analiza un ejemplo en dos dimensiones.

En un espacio de 2 dimensiones, un hiperplano tiene *2-1* dimensiones, es decir, una recta. La ecuación que se emplea con más frecuencia en cálculo para definir una recta es:

$$y = ax + b$$

o de forma equivalente:

$$y - ax - b = 0$$

Otra forma de definir una recta es mediante el *dot product* de dos vectores. Supóngase los vectores $\mathbf{x} = (x, y)$ y $\mathbf{w} = (w_1, w_2)$, y la constante b, con ellos puede definirse una recta de la siguiente forma:

$$\mathbf{x} \cdot \mathbf{w} + b = 0$$

$$(x, y) \cdot (w_1, w_2) + b = 0$$

$$(xw_1 + yw_2) + b = 0$$

$$y = -\frac{w_1}{w_2}x - \frac{b}{w_2}$$

Si se define a y b como:

$$a = -\frac{w_1}{w_2} \quad y \quad c = -\frac{b}{w_2}$$

Se obtiene la ecuación de una recta:

$$y = ax + c$$

La ventaja de definir una recta empleando vectores es que puede generalizarse para cualquier número de dimensiones. De hecho, esta es la forma con la que se obtiene la ecuación de un hiperplano en cualquier dimensión.

CLASIFICACIÓN BINARIA MEDIANTE UN HIPERPLANO

Tal y como se ha definido previamente, un hiperplano de dimensión n divide un espacio de dimensión $n + 1$ en dos partes. Esto significa que puede emplearse a modo de clasificador binario. Las observaciones que queden por encima del hiperplano pertenecen a una clase y las que quedan por debajo a la otra. En un espacio de dos dimensiones, cada observación i esta definida por un vector $\mathbf{x}_i$ y una variable respuesta y que puede tomar dos valores (*+1, -1*). Dado un hiperplano definido por el vector $\mathbf{w}$ y la constante b, para clasificar las observaciones, se busca una función h tal que:

$$h(\mathbf{x}_i) = \begin{cases} +1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b \geq 0 \\ -1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b < 0 \end{cases}$$

lo que es equivalente a:

$$h(\mathbf{x}_i) = sign(\mathbf{w} \cdot \mathbf{x} + b)$$

La ecuación de la función h puede simplificarse excluyendo el término b. Para ello, se añade, a todos los vectores $\mathbf{x}_i$, el valor 1 en la primera posición y, al vector del hiperplano $\mathbf{w}$ el valor de b. Aunque el resultado final es el mismo, suele ser más sencillo trabajar de esta forma a la hora de implementar los algoritmos. A los vectores modificados con este fin se les conoce como vectores aumentados.

PERCEPTRON

Tal como se ha descrito hasta ahora, independientemente de la dimensión, se puede crear un clasificador binario (siempre que los datos sean linealmente separables) mediante un hiperplano, pero, ¿Cómo se encuentra dicho hiperplano?. Es aquí donde entra en juego el algoritmo del *Perceptron*. Dado un set de datos con m observaciones n-dimensionales $(\mathbf{x}_i, y_i)$, el *Perceptron* trata de encontrar la función h capaz de predecir correctamente la clase y_i para cada $\mathbf{x}_i$. La función h empleada por el *Perceptron* es $h(\mathbf{x}_i) = sign(\mathbf{w} \cdot \mathbf{x})$, donde $\mathbf{w} \cdot \mathbf{x}$ no es más que la ecuación de un hiperplano definida por dos vectores aumentados, de los cuales, el único desconocido es $\mathbf{w}$. Por lo tanto, el objetivo del *Perceptron* es encontrar el vector $\mathbf{w}$ que permite separar perfectamente las observaciones.

La forma en que el algoritmo *Perceptron* encuentra el hiperplano óptimo es la siguiente:

---

1. Iniciar el proceso con un hiperplano aleatorio definido por el vector aleatorio $\mathbf{w}$.
2. Clasificar todas las observaciones acorde al hiperplano $\mathbf{w}$.
3. De entre las observaciones mal clasificadas, seleccionar una de forma aleatoria (i) y con ella actualizar el hiperplano:

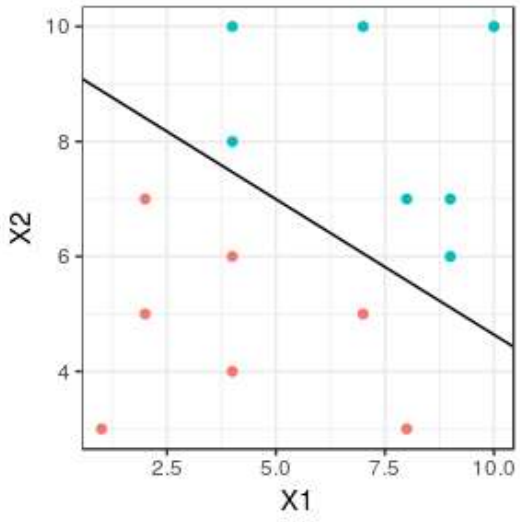
$$\mathbf{w} = \mathbf{w} + \mathbf{x}_i * y_i$$

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

```r
perceptron <-  function(X, y, random_seed = 553){
  # Esta función implementa el algoritmo del perceptron para encontrar un
  # hiperplano que separe correctamente las dos clases.

  # Transformar X en vectores aumentados
  X <- cbind(1, X)
  # Inicialización aleatoria del hiperplano
  set.seed(random_seed)
  w <- c(runif(n = 3, min = 0, max = 1))
  # Clasificación
  clasificaciones <- predict_clase(X = X, w = w)
  # Índice de las observaciones mal clasificadas
  errores_clasificacion <- which((clasificaciones != y) == FALSE)

    while (length(errores_clasificacion) > 0) {
    # Se selecciona aleatoriamente una observación errónea
    i <- sample(x = errores_clasificacion, size = 1)
    # Actualización del hiperplano
    w <- w + X[i,] * y[i]
    clasificaciones <- predict_clase(X = X, w = w)
    errores_clasificacion <- which((clasificaciones == y) == FALSE)
  }
  return(w)
}

predict_clase <- function(X, w){
  # Esta función devuelve la clasificación de las observaciones
  # acorde al valor de sus predictores X y al hiperplano w
  clase_predicha <- apply(X = X, MARGIN = 1, FUN = function(x){crossprod(x,w)})
  clase_predicha <- sign(clase_predicha)
  return(clase_predicha)
}

# Ejemplo observaciones linealmente separables en 2 dimensiones
X <- matrix(c(8, 4, 9, 7, 9, 4, 10, 2, 8, 7, 4, 4, 1, 2, 7, 10, 7, 10, 6, 8, 10,
              7, 3, 5, 4, 6, 3, 5), ncol = 2, byrow = FALSE)
y <- c(1, 1, 1, 1, 1, 1, 1 , -1 , -1 , -1 , -1 , -1 , -1 , -1 )

hiperplano <- perceptron(X = X, y = y)
hiperplano
```

```
## [1] -54.452542   2.744047   5.822881
```

```
library(ggplot2)
datos <- data.frame(X, y)
ggplot(data = datos, aes(x = X1, y = X2, color = as.factor(y))) +
  geom_point() +
  # La pendiente e intersección de la recta se obtienen siguiendo los pasos
  # descritos anteriormente para obtener una recta a partir dos vectores
  geom_abline(intercept = -(hiperplano[1]/hiperplano[3]),
              slope =     -(hiperplano[2]/hiperplano[3])) +
  theme_bw() + theme(legend.position = "none")
```



Es interesante tener en cuenta dos propiedades de este algoritmo:

- En cada actualización del hiperplano, se modifica el vector **w** intentando que clasifique bien una de las observaciones mal clasificadas (seleccionada aleatoriamente) pero sin tener en cuenta el resto. Esto significa que, una determinada actualización, puede hacer que se consiga clasificar bien la observación en cuestión pero que otra u otras que estaban bien clasificadas pasen a estar mal. Afortunadamente, los matemáticos han demostrado que el algoritmo del *Perceptron* siempre acaba encontrando un hiperplano de separación (siempre y cuando los datos sean linealmente separables).

- Como puede intuirse observando la imagen anterior, existen infinitos hiperplanos que consiguen separar las clases, cuando estas son linealmente separables. Dada la selección aleatoria de observaciones en el paso de actualización, el algoritmo genera diferentes hiperplanos cada vez que se ejecuta.

- Si bien el *Perceptron* siempre encuentra un hiperplano que separa perfectamente las clases (siempre y cuando sean linealmente separables), no tiene por qué ser el hiperplano óptimo, entendiendo por hiperplano óptimo aquel que separa correctamente las observaciones y que además se encuentra en el punto medio entre ambas clases.

Bibliografía

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

Support Vector Machines Succinctly by Alexandre Kowalczyk

A Practical Guide to Support Vector Classification by Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin

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

A Gentle Introduction to Support Vector Machines in Biomedicine. Alexander Statnikov, Douglas Hardin, Isabelle Guyon, Constantin F. Aliferis

This work by Joaquín Amat Rodrigo is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.