Junio 2018

Cristina Gil Martínez

MÁQUINAS DE VECTOR SOPORTE

Apuntes personales sobre SVM lineal, polinómico y radial para problemas de clasificación

CONTENIDO

INTRODUCCIÓN

Las máquinas de vector soporte o *Support Vector Machines* (SVM) son otro tipo de algoritmo de *machine learning* **supervisado** aplicable a problemas de **regresión** y **clasificación**, aunque se usa más comúnmente como modelo de clasificación. Las máquinas de vector soporte suponen una generalización de un clasificador simple denominado *maximal margin classifier*. Sin embargo, este clasificador no puede aplicarse a sets de datos donde las clases de la variable respuesta no son separables mediante un **límite lineal**. Una extensión del mismo, el *support vector classifier* es aplicable en un mayor rango de casos. El *support vector machine* supone una extensión más del *support vector classifier* para casos con **límites no lineales** entre clases (clasificación binaria o de más clases).

MAXIMAL MARGIN CLASSIFIER

Concepto de hiperplano

En un espacio euclídeo *p*-dimensional, un hiperplano es un subespacio plano y afín (no tiene por qué pasar por el origen) de dimensión $p - 1$, que divide el espacio en dos mitades. Por ejemplo, en un espacio de 2 dimensiones un hiperplano es un subespacio plano de una sola dimensión, o lo que es lo mismo, una línea, definida por la ecuación lineal

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

donde cualquier $X = (X_1, X_2)$ para los que se cumple la ecuación es un punto en el hiperplano.

En un espacio de 3 dimensiones el hiperplano se corresponde con un sub-espacio de dos dimensiones, es decir, un plano. Si $p > 3$ puede resultar difícil visualizar el hiperplano. Para escenarios *p*-dimensionales, la ecuación anterior puede ser extendida a

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

donde, igualmente, los puntos $X = (X_1, X_2, \ldots X_p)$ que cumplen la ecuación pertenecen al hiperplano.

En el supuesto que *X* no satisfazca la ecuación, dándose uno de estos dos casos

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

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

el punto *X* estará situado a uno u otro lado del hiperplano, no sobre él.

Uso del hiperplano para clasificación binaria

Suponiendo que contamos con una matriz de datos *n x p* con *n* observaciones y *p* predictores (donde en la variable respuesta son distinguibles dos clases distintas $y_1, \ldots, y_n \in \{-1, 1\}$), el objetivo será el de desarrollar un clasificador en base al subgrupo de datos de entrenamiento que clasifique correctamente nuevas observaciones en base a los valores de los predictores en función de un hiperplano de separación con la propiedad

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} > 0 \text{ if } y_i = 1$$

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} < 0 \text{ if } y_i = -1$$

o equivalentemente

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) > 0$$

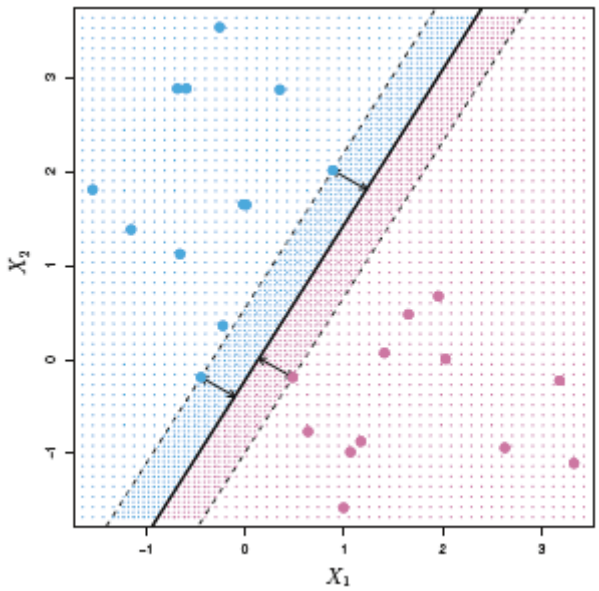
para cada $i = 1, \ldots, n$.

Siendo $\beta_0, \beta_1, \ldots, \beta_p$ los coeficientes del hiperplano, una nueva observación x^* se asignará a un grupo u otro dependiendo de en qué lado del hiperplano se localice (en función del signo de $f(x^*) = \beta_0 + \beta_1 x^*_1 + \ldots + \beta_p x^*_p$). Si $f(x^*)$ es positiva, se asigna la nueva observación a la clase 1, y si es negativa, a la clase -1. La magnitud de $f(x^*)$ también es informativa: si $f(x^*)$ tiene un valor muy lejano a 0, significa que x^* se encuentra muy lejos del hiperplano, lo cual aporta más seguridad a la clasificación de dicha observación. Por el contrario, un valor de $f(x^*)$ próximo a 0 significa que la observación está cerca del hiperplano, con lo que estaremos menos seguros acerca de la clase asignada a esta observación.

Nota: Otros métodos aplicables a este problema son LDA, regresión logística y árboles de clasificación.

Clasificación binaria para casos separables linealmente

Si nuestros datos son perfectamente separables mediante un hiperplano, entonces existirá un número infinito de tales hiperplanos (debido a la posibilidad de moverlos o rotarlos sin entrar en contacto con las observaciones). Esto hace que necesitemos un modo de decidir cuál de todos los hiperplanos posibles utilizar. La solución es escoger el hiperplano de separación que se encuentre más alejado de las observaciones de entrenamiento, al que se conoce como *maximal margin hyperplane* o hiperplano óptimo de separación. Este se obtiene calculando las distancias perpendiculares de cada observación a un hiperplano dado, donde la distancia más pequeña se corresponde con la distancia mínima de las observaciones al hiperplano, espacio conocido como **margen**. Con esto, el **hiperplano óptimo de separación** será aquel que tenga la mayor distancia mínima de las observaciones al hiperplano, o lo que es lo mismo, el **mayor margen** (*M*). Por tanto, los parámetros del hiperplano β_0, β_1, ..., β_p se optimizan para maximizar *M*.



(Imagen obtenida del libro ISLR)

En la imagen, la línea negra continua representa el hiperplano óptimo de separación y el margen en líneas discontinuas. En ella, hay tres observaciones (dos de la clase azul y una de la morada) que son equidistantes al hiperplano. A estas observaciones que son las que se encuentran sobre el margen, y por tanto, más próximas al hiperplano, se conocen como *support vectors* o **vectores soporte**, pues son vectores en el espacio *p*-dimensional (en la imagen, $p = 2$) que "soportan" al hiperplano óptimo de separación en el sentido de que si estas observaciones cambiaran ligeramente, el hiperplano lo haría también. El movimiento del

resto de observaciones no tendría impacto en el hiperplano, siempre y cuando no cruzaran el límite establecido por el margen.

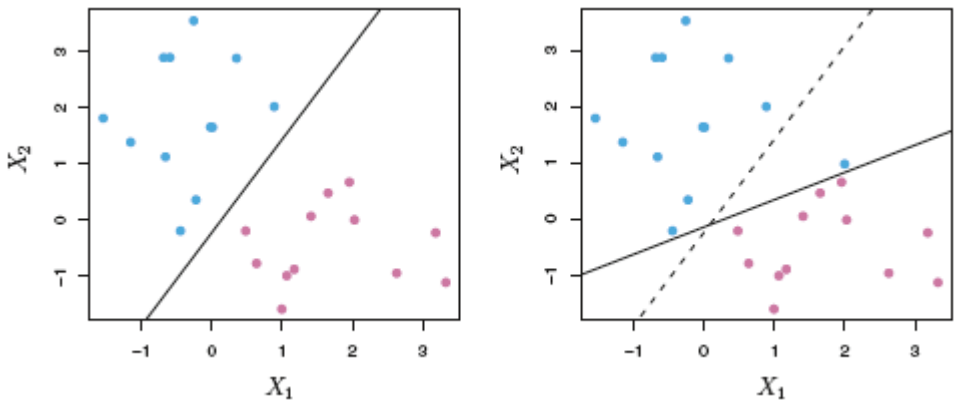
Se espera que un clasificador con un margen alto para las observaciones de entrenamiento lo tenga también para las observaciones de test, y pueda por tanto clasificarlas correctamente.

Un problema que puede surgir es que este clasificador sufra de *overfitting* cuando el número de dimensiones (*p*) es alto.

SUPPORT VECTOR CLASSIFIER

Clasificación binaria para casos cuasi-separables linealmente

Observaciones pertenecientes a dos clases no tienen por qué ser necesariamente separables mediante el uso de un hiperplano, con lo que el uso de este tipo de clasificador puede no ser deseable. Un clasificador basado en un hiperplano clasificará todas las observaciones de entrenamiento perfectamente, lo cual conlleva a una sensibilidad a nuevas observaciones (signo de *overfitting*): la adición de una sola observación tiene la capacidad de cambiar drásticamente el hiperplano óptimo de separación, tal y como se muestra en la siguiente imagen:



(Imagen obtenida del libro ISLR)

El cambio en el hiperplano causado por una sola observación en la imagen superior, hace que éste deje de ser óptimo, ya que el margen se ha reducido drásticamente, y esta distancia supone una medida de la confianza con la que una observación es correctamente clasificada.

En estos casos, puede ser útil considerar el clasificador denominado ***soft margin classifier*** *o support vector classifier*, que, aún basado en un hiperplano, no separe perfectamente las dos clases, con el interés de obtener:

- Mayor robustez a observaciones individuales
- Mejor clasificación de la mayoría de las observaciones de entrenamiento y test.

Con este clasificador, se permite que algunas observaciones se encuentren en el lado incorrecto del margen (de ahí el término *soft* o blando), o incluso en el lado incorrecto del hiperplano. Estas serán las observaciones de entrenamiento mal clasificadas por el modelo.

De nuevo, el *support vector classifier* clasifica cada nueva observación en función de a qué lado de hiperplano pertenezca, es decir, en función del signo de $f(x^*) = \beta_0 + \beta_1 x^*_1 + ... + \beta_p x^*_p$), y al igual que con el *maximal margin classifier*, solo las observaciones que se encuentran sobre el margen o que lo violan (vectores soporte) afectarán al hiperplano y, por lo tanto, al clasificador obtenido.

Nota: en un support vector classifier los ***vectores soporte*** *se corresponden con las observaciones que se encuentran sobre el margen y también las que lo violan.*

El proceso de optimización del hiperplano en este caso incorpora un **parámetro de regularización** o *tuning parameter* ***C***, el cual controla la severidad permitida de las violaciones de las *n* observaciones sobre el margen e hiperplano, y a la vez, el equilibrio *bias*-varianza. Sin entrar en detalles matemáticos, si $\boldsymbol{C > 0}$, no más de *C* observaciones pueden encontrarse en el lado incorrecto del hiperplano. Si *C* es pequeño, los márgenes serán estrechos pues muy pocas observaciones podrán estar en el lado incorrecto del mismo (esto equivale a un modelo bastante bien ajustado a los datos, el cual puede tener poco *bias* pero mucha varianza). Conforme incrementamos *C*, mayor es la toleración a las violaciones sobre el margen, con lo que el margen será más ancho y habrá más vectores soporte (esto equivale a un modelo más flexible y con mayor *bias* pero menor varianza). Si $\boldsymbol{C = 0}$, el clasificador es equivalente al *maximal margin classifier*, pues no están permitidas violaciones sobre el margen (todas las observaciones deben estar correctamente clasificadas). En la práctica el parámetro C se escoge u optimiza por **validación cruzada**.

En resumen:

- *Poca varianza y alto bias: márgenes anchos y mayor número de vectores soporte*
- *Mucha varianza y bajo bias: márgenes estrechos y menor número de vectores soporte*

SUPPORT VECTOR MACHINES

Clasificación binaria para casos no separables linealmente

Anteriormente se ha explicado como los hiperplanos de separación son buenos clasificadores cuando las clases son perfectamente separables o cuasi-perfectamente separables. Sin embargo, la aplicación de un *support vector classifier* para casos claramente no separables linealmente (la mayoría de problemas reales) carece de interés práctico.

Una posibilidad para tratar con límites no lineales entre clases consiste en aumentar manualmente el espacio de los predictores mediante funciones polinómicas o con términos de interacción. Aun así, corremos el riesgo de acabar con demasiados predictores. Como alternativa está el uso de los *support vector machines* o máquinas de vector soporte (*SVM*), que suponen una **extensión de los *support vector classifiers*** que aumenta la dimensionalidad de una manera específica, mediante el uso de ***kernels***, un enfoque computacionalmente más eficiente. Los kernels son funciones que transforman un espacio de pocas dimensiones en un espacio de dimensiones mayores mediante transformaciones complejas de los datos. También puede definirse como una función que cuantifica la similitud entre dos observaciones en un nuevo espacio dimensional.

Entre los kernels más populares para usar con *SVM*s se encuentran:

Kernel lineal

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

El kernel lineal cuantifica la similitud de un par de observaciones usando la correlación de *Pearson*. Con un kernel lineal, el clasificador obtenido es equivalente a un *support vector classifier*.

Kernel polinómico

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

Un kernel polinómico de grado *d* (siendo *d* > 1) permite un límite de decisión mucho más flexible. Cuando un *support vector classifier* se combina con un kernel no lineal, se obtiene un *support vector machine*.

Kernel radial

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

donde γ es una constante positiva que cuanto mayor sea, mayor la flexibilidad del *SVM*. Suponiendo que una observación de test $x^* = (x_1^* \dots x_p^*)$ se encuentra alejada de una observación de entrenamiento xi en términos de distancia Euclídea, entonces $K(x^*, x_i)$ será muy pequeño, lo que significa que xi no influirá en $f(x^*)$. El kernel radial tiene un comportamiento muy local, en el sentido de que solo las observaciones de entrenamiento cercanas a una observación de test tendrán efecto sobre su clasificación.

Es importante tener en cuenta que una mayor flexibilidad no tiene por qué mejorar las predicciones, ya que un modelo muy flexible puede ajustarse demasiado a los datos de entrenamiento.

Clasificación con más de dos clases

Existen varias extensiones de los *SVM*s para problemas de clasificación con más de dos clases ($K > 2$), siendo dos de las más populares:

- *One-versus-one*
- *One-versus-all*

CLASIFICACIÓN *ONE-VS-ONE*

Este método construye $\binom{K}{2}$ *SVM*s, correspondiente a $K(K-1)/2$, cada uno comparando un par de clases. Una observación de test se clasifica usando cada uno de los *SVM*s, contando el número de veces que esta observación es asignada a cada una de las K clases. La clase final predicha será aquella a la que la observación ha sido asignada en la mayoría de los *SVM*s.

CLASIFICACIÓN *ONE-VS-ALL*

Se ajustan *K SVM*s, cada vez comparándose una de las K clases (codificada como +1) con el resto de *K* -1 clases (codificadas como -1). Siendo β_{0k}, β_{1k}, ..., β_{pk} los parámetros resultantes del ajuste de un *SVM* y *x** una observación de test, la observación será asignada a la clase para la que $\beta_{0k} + \beta_{1k}x^*_1 + ... + \beta_{pk}x^*_p$ sea mayor. Es decir, la magnitud de *f(x*)* indica como de lejos está *x** del hiperplano de separación, y cuanto más lejos esté, mayor será el nivel de confianza de que la observación *x** ha sido correctamente clasificada.

SUPPORT VECTOR REGRESSION

SVM también puede utilizarse como un método de regresión (*support vector regression* o *SVR*). *SVR* sigue los mismos principios que el *SVM* para clasificación, con alguna diferencia en cuanto al algoritmo (se establece un margen de tolerancia para las predicciones, *épsilon*).

SVM VS REGRESIÓN LOGÍSTICA

Una característica interesante de los *support vector classifiers* es que solo los vectores soporte juegan un papel importante en la clasificación final obtenida: observaciones en el lado correcto del margen no afectan a las predicciones. En contraposición, el término de penalización en regresión logística es muy pequeño para observaciones lejanas al límite de decisión, pero nunca es exactamente 0. Aun así, dadas las similitudes en cuanto al término de penalización, ambos métodos estadísticos pueden dar con frecuencia resultados similares.

Cuando las clases son fácilmente separables, los *SVM*s suelen superar a la regresión logística. En situaciones donde las clases son más solapantes, la regresión logística cuele ser la opción escogida.

El uso de kernels para aumentar la dimensionalidad no está limitado sólo al uso de *SVM*s (podrían usarse también para regresión logística), pero su uso está más extendido en estos casos.

EJEMPLOS EN R

Support vector classifier/machine: library(e1071)

- **svm()** -> Ajuste de modelo support vector classifier (kernel = "linear") y support vector machine (kernel = "polinomial", "radial"...). Si la variable respuesta contiene más de dos niveles, la función lleva a cabo la clasificación usando el método *one-vs-one*.
- **tune()** -> Función genérica para optimización de hiperparámetros de métodos estadísticos, a partir de valores proporcionados.
- **plot.svm()**

Para este ejemplo de clasificación utilizaremos el set de datos `OJ`, del paquete `ISLR`. Contiene información sobre compra de dos tipos de bebida (*Citrus Hill* y *Minute Maid Orange Juice*) por parte de 1070 clientes (las variables registran distintas características del cliente y el producto). Generaremos modelos basados en *SVM* con tres tipos de kernel: lineal, polinómico y radial, que predigan qué tipo de bebida (*Purchase*) compra el consumidor, en función del conjunto de predictores.

NOTA: La variable respuesta ha de estar codificada como `factor`.

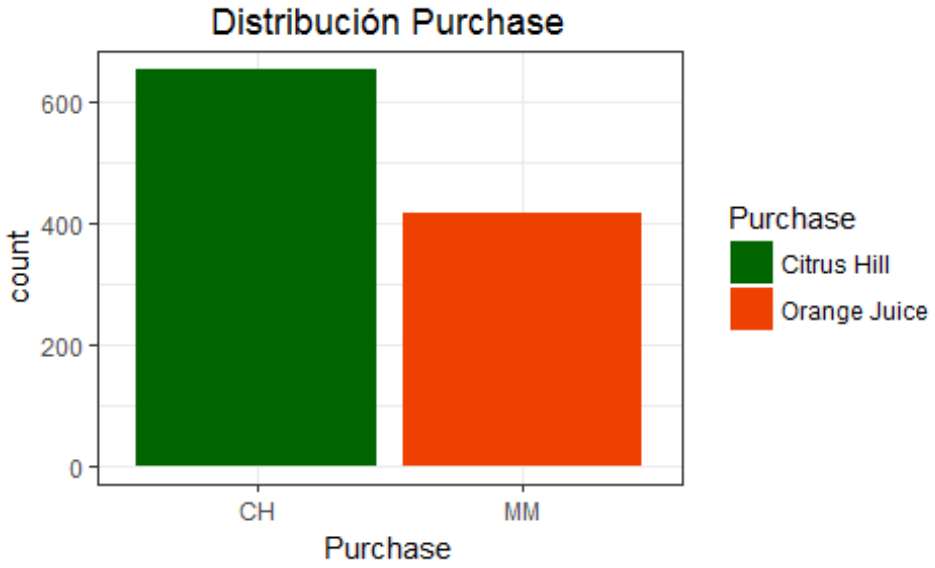
NOTA: Un número alto de variables en relación al número de observaciones implicaría que es fácil encontrar un hiperplano que separe completamente las clases.

(Ver ejemplo de árbol de clasificación aplicado a este set de datos en Árboles de decisión. Como en este caso contamos con un predictor con $K = 2$, la regresión logística también sería una opción).

```
library(ISLR)

str(OJ)
```

```
## 'data.frame':    1070 obs. of  18 variables:
##  $ Purchase      : Factor w/ 2 levels "CH","MM": 1 1 1 2 1 1 1 1 1 1 ...
##  $ WeekofPurchase: num  237 239 245 227 228 230 232 234 235 238 ...
##  $ StoreID       : num  1 1 1 1 7 7 7 7 7 7 ...
##  $ PriceCH       : num  1.75 1.75 1.86 1.69 1.69 1.69 1.69 1.75 1.75 1.75 ...
##  $ PriceMM       : num  1.99 1.99 2.09 1.69 1.69 1.99 1.99 1.99 1.99 1.99 ...
##  $ DiscCH        : num  0 0 0.17 0 0 0 0 0 0 0 ...
##  $ DiscMM        : num  0 0.3 0 0 0 0 0.4 0.4 0.4 0.4 ...
##  $ SpecialCH     : num  0 0 0 0 0 0 1 1 0 0 ...
##  $ SpecialMM     : num  0 1 0 0 0 1 1 0 0 0 ...
##  $ LoyalCH       : num  0.5 0.6 0.68 0.4 0.957 ...
##  $ SalePriceMM   : num  1.99 1.69 2.09 1.69 1.69 1.99 1.59 1.59 1.59 1.59 ...
##  $ SalePriceCH   : num  1.75 1.75 1.69 1.69 1.69 1.69 1.69 1.75 1.75 1.75 ...
##  $ PriceDiff     : num  0.24 -0.06 0.4 0 0 0.3 -0.1 -0.16 -0.16 -0.16 ...
##  $ Store7        : Factor w/ 2 levels "No","Yes": 1 1 1 1 2 2 2 2 2 2 ...
##  $ PctDiscMM     : num  0 0.151 0 0 0 ...
```

```
##  $ PctDiscCH     : num  0 0 0.0914 0 0 ...
##  $ ListPriceDiff : num  0.24 0.24 0.23 0 0 0.3 0.3 0.24 0.24 0.24 ...
##  $ STORE         : num  1 1 1 1 0 0 0 0 0 0 ...

sum(is.na(OJ$Purchase))

## [1] 0

# Distribución variable respuesta
library(ggplot2)

ggplot(data = OJ, aes(x = Purchase, y = ..count.., fill = Purchase)) +
geom_bar() +
labs(title = "Distribución Purchase") +
scale_fill_manual(values = c("darkgreen", "orangered2"),
                  labels = c("Citrus Hill", "Orange Juice")) +
theme_bw() + theme(plot.title = element_text(hjust = 0.5))
```




```
# Tabla frecuencias variable respuesta
table(OJ$Purchase)

##
##  CH  MM
## 653 417

# Tabla proporciones variable respuesta
library(dplyr)
prop.table(table(OJ$Purchase)) %>% round(digits = 2)

##
##   CH   MM
## 0.61 0.39
```

Para que los modelos generados sean útiles, el porcentaje de aciertos en cuanto a la clasificación de las observaciones ha de superar un nivel mínimo, en este caso, el que se obtendría si la predicción de todas las observaciones se correspondiera con la clase mayoritaria. La clase mayoritaria (moda) en este caso es la bebida *CH* con el 61% de las compras. Este será el nivel basal a superar por el modelo (este es el porcentaje mínimo de aciertos si siempre se predijera *CH*). (Recalcular este valor con los datos de entrenamiento).

Antes de proceder a generar los modelos, dividimos el set de datos en un grupo de entrenamiento (para el ajuste de los modelos) y otro de test (para la evaluación de los mismos). Esta división dependerá de la cantidad de observaciones con las que contemos y la seguridad con la que queramos obtener la estimación del *test error*. En este ejemplo se opta por una división 80%-20%.

```
library(caret)

# Índices observaciones de entrenamiento
set.seed(123)
train <- createDataPartition(y = OJ$Purchase, p = 0.8, list = FALSE, times = 1)


# Datos entrenamiento
datos.OJ.train <- OJ[train, ]
dim(datos.OJ.train)

## [1] 857  18

# Datos test
datos.OJ.test <- OJ[-train, ]
dim(datos.OJ.test)

## [1] 213  18
```

Es importante comprobar que la variable respuesta se distribuye de equitativamente en ambos grupos y con respecto al set de datos completo, aunque la función del paquete *caret* `createDataPartition()` asegura esta distribución:

```
prop.table(table(datos.OJ.train$Purchase))

##
##        CH        MM
## 0.6102684 0.3897316

prop.table(table(datos.OJ.test$Purchase))

##
##        CH        MM
## 0.6103286 0.3896714
```

Support vector classifier

AJUSTE DEL MODELO

PAQUETE E1071

A la hora de ajustar un *support vector classifier*, es importante tener en cuenta que el hiperparámetro C (`cost`) controla el equilibrio *bias*-varianza y la capacidad predictiva del modelo, ya que determina la severidad que se permite respecto a las violaciones sobre el margen. En otras palabras, necesitamos fijar un margen de separación entre observaciones a *priori.* Por ello es recomendable evaluar distintos valores del mismo mediante validación cruzada y escoger el valor óptimo.

IMPORTANTE: **Estandarizar** los predictores cuando no estén medidos en la misma escala, para que los de mayor magnitud no tengan mayor influencia que el resto. Un argumento disponible en la función `svm()` para ello es `scale = TRUE`).

Para ajustar un *support vector classifier*, el `kernel` indicado en la función `svm()` ha de ser `lineal`.

```
library(e1071)

set.seed(32)
tuning <- tune(svm, Purchase ~ ., data = datos.OJ.train,
               kernel = "linear",
               ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 50)),
               scale = TRUE)
```

Podemos acceder a los errores de validación de los modelos con cada valor de *cost* con la función `summary()`:

```
summary(tuning)
```

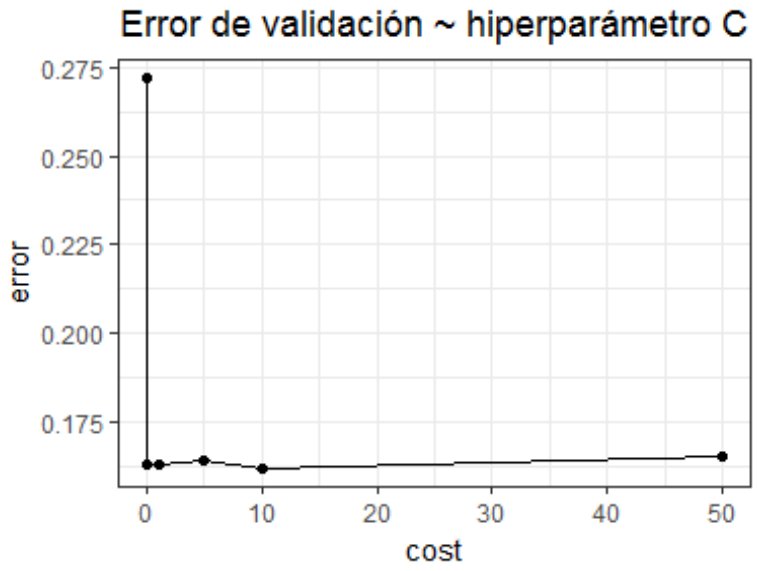
```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.1620383
##
## - Detailed performance results:
```

```
##    cost     error dispersion
## 1 1e-03 0.2718605 0.03663692
## 2 1e-02 0.1632148 0.04200860
## 3 1e-01 0.1631464 0.05356194
## 4 1e+00 0.1631601 0.05038597
## 5 5e+00 0.1643639 0.04686947
## 6 1e+01 0.1620383 0.04624690
## 7 5e+01 0.1655267 0.04402181
```

```
names(tuning)
```

```
## [1] "best.parameters"  "best.performance" "method"           "nparcomb"
## [5] "train.ind"        "sampling"         "performances"     "best.model"
```

```
ggplot(data = tuning$performances, aes(x = cost, y = error)) +
geom_line() +
geom_point() +
labs(title = "Error de validación ~ hiperparámetro C") +
theme_bw() +
theme(plot.title = element_text(hjust = 0.5))
```



El valor de coste que resulta en el menor error de validación (0,162) es *cost* = 10.

```
mejor.modelo.svc <- tuning$best.model
summary(mejor.modelo.svc)
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = datos.OJ.train,
##     ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 50)), kernel = "linear",

##     scale = TRUE)
##
##
```

```
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##       gamma:  0.05555556
##
## Number of Support Vectors:  341
##
##  ( 170 171 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

El número de vectores soporte es de 341, 170 de la clase *CH* y 171 de la clase *MM*. Podemos obtener los índices de las observaciones que se corresponden con los vectores soporte:

```
head(mejor.modelo.svc$index, 100)
```

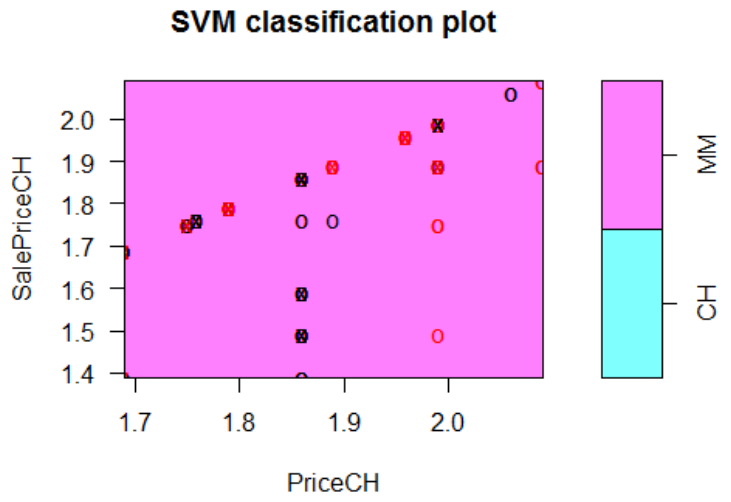
```
##   [1]   1  13  23  30  45  56  57  58  72  73  74  75  76  77  78  79  84  92
##  [19] 111 112 115 116 117 120 167 169 182 189 190 192 208 209 210 214 215 216
##  [37] 217 241 244 247 251 252 253 254 259 260 263 265 267 284 288 289 297 298
##  [55] 301 303 307 308 310 316 322 324 332 344 345 346 347 348 352 356 358 362
##  [73] 367 368 373 374 382 389 391 399 403 421 423 426 430 441 442 444 447 448
##  [91] 449 453 458 460 461 463 468 470 487 488
```

El mejor modelo obtenido sería equivalente a ajustar:

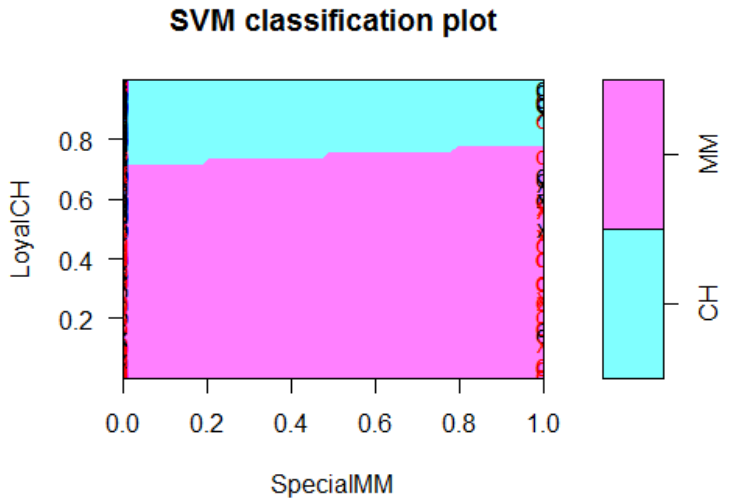
```
modelo.svc <- svm(Purchase ~ ., data = datos.OJ.train,
                  kernel = "linear",
                  cost = 10,
                  scale = TRUE)
```

Al tratarse de un problema con más de dos predictores, podemos representar el modelo usando la función *plot()*, pero creando representaciones entre pares de predictores (teniendo en cuenta que *plot.svm* solo representa predictores continuos). Ejemplos:

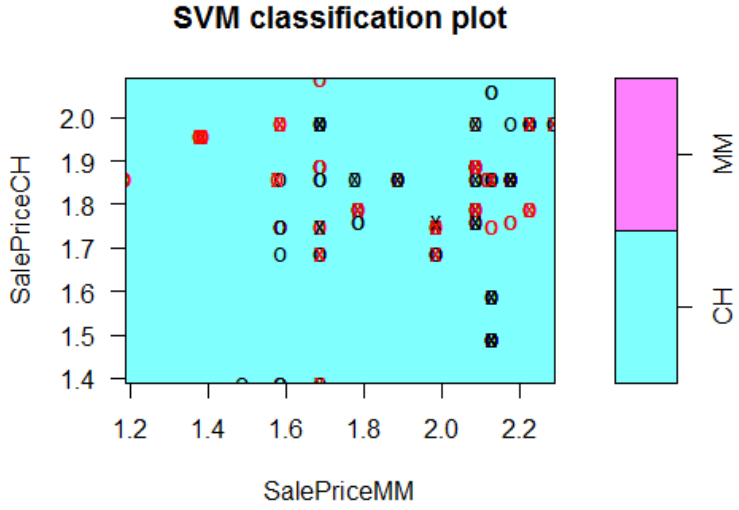
```
plot(modelo.svc, datos.OJ.test, SalePriceCH ~ PriceCH)
```




```
plot(modelo.svc, datos.OJ.test, LoyalCH ~ SpecialMM)
```




```
plot(modelo.svc, datos.OJ.test, SalePriceCH ~ SalePriceMM)
```



EVALUACIÓN DEL MODELO

```
# Error de entrenamiento
pred <- predict(modelo.svc, datos.OJ.train)
table(predicción = pred, real = datos.OJ.train$Purchase)

##           real
## predicción  CH  MM
##         CH 468  77
##         MM  55 257

paste("Error de entrenamiento:",
      100 * mean(datos.OJ.train$Purchase != pred) %>% round(digits = 4), "%")

## [1] "Error de entrenamiento: 15.4 %"



# Error de test
pred <- predict(modelo.svc, datos.OJ.test)
table(predicción = pred, real = datos.OJ.test$Purchase)

##           real
## predicción  CH  MM
##         CH 107  18
##         MM  23  65

paste("Error de test:",
      100 * mean(datos.OJ.test$Purchase != pred) %>% round(digits = 4), "%")

## [1] "Error de test: 19.25 %"
```

Con un valor de *cost* = 10, el 19,25% de las observaciones son incorrectamente clasificadas. El error de entrenamiento subestima el error de test, ya que el modelo es mejor prediciendo las observaciones con las que ha sido entrenado.

PAQUETE CARET

SVM lineal: `method = "svmLinear"`

SVM polinómico: `method = "svmPoly"`

SVM radial: `method = "svmRadial"`

Si no se especifica, la métrica para la evaluación es el Accuracy. Podría emplearse otra como por ejemplo "ROC".

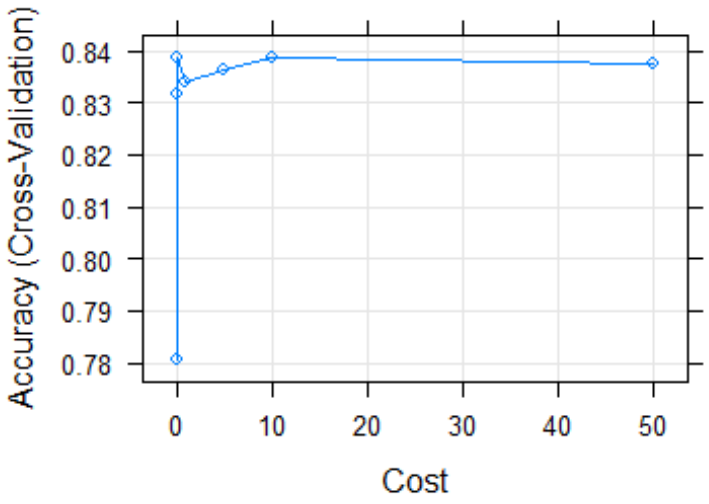
```r
# AJUSTE DEL MODELO
# ------------------------------------------------------------------------------


# Configuración del proceso de selección del modelo
validacion <- trainControl(method = "cv",
                           number = 10,
                           classProbs = TRUE,
                           search = "grid")

getModelInfo(model = "svmLinear")[[2]]$parameters
```

```
##   parameter   class label
## 1         C numeric  Cost
```

```r


# Valores del hiperparámetro C a evaluar
hiperC <- data.frame(C = c(0.001, 0.01, 0.1, 1, 5, 10, 50))



# Entrenamiento del SVM con un kernel lineal y optimización del hiperparámetro C
set.seed(32) # misma semilla que en el ejemplo con el paquete e1071
modelo.svc <- train(Purchase ~ ., data = datos.OJ.train,
                    method = "svmLinear",
                    trControl = validacion,
                    preProc = c("center", "scale"),
                    tuneGrid = hiperC)

# Resultado del entrenamiento
modelo.svc
```

```
## Support Vector Machines with Linear Kernel
##
## 857 samples
##  17 predictor
##   2 classes: 'CH', 'MM'
##
## Pre-processing: centered (17), scaled (17)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 771, 771, 771, 771, 771, 772, ...
## Resampling results across tuning parameters:
##
##   C      Accuracy   Kappa
##   1e-03  0.7804514  0.5601829
##   1e-02  0.8318194  0.6438262
##   1e-01  0.8388372  0.6579138
##   1e+00  0.8341860  0.6474393
##   5e+00  0.8364979  0.6529778
##   1e+01  0.8388509  0.6570923
##   5e+01  0.8376881  0.6552768
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 10.
```

```r


plot(modelo.svc)
```




```
# EVALUACIÓN DEL MODELO
# ------------------------------------------------------------------------------

confusionMatrix(pred, datos.OJ.test$Purchase)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 107  18
##         MM  23  65
##
##                Accuracy : 0.8075
##                  95% CI : (0.7481, 0.8582)
##     No Information Rate : 0.6103
##     P-Value [Acc > NIR] : 4.816e-10
##
##                   Kappa : 0.5997
##  Mcnemar's Test P-Value : 0.5322
##
##             Sensitivity : 0.8231
##             Specificity : 0.7831
##          Pos Pred Value : 0.8560
##          Neg Pred Value : 0.7386
##              Prevalence : 0.6103
##          Detection Rate : 0.5023
##    Detection Prevalence : 0.5869
##       Balanced Accuracy : 0.8031
##
##        'Positive' Class : CH
##
##       'Positive' Class : CH
```

Support vector machine

Además del hiperparámetro de penalización *C*, en los modelos *SVM* es necesario especificar el hiperparámetro gamma (para kernel radial) o el grado de polinomio (para kernel polinómico). Es también importante optimizar estos dos últimos, previo ajuste del modelo.

KERNEL POLINÓMICO

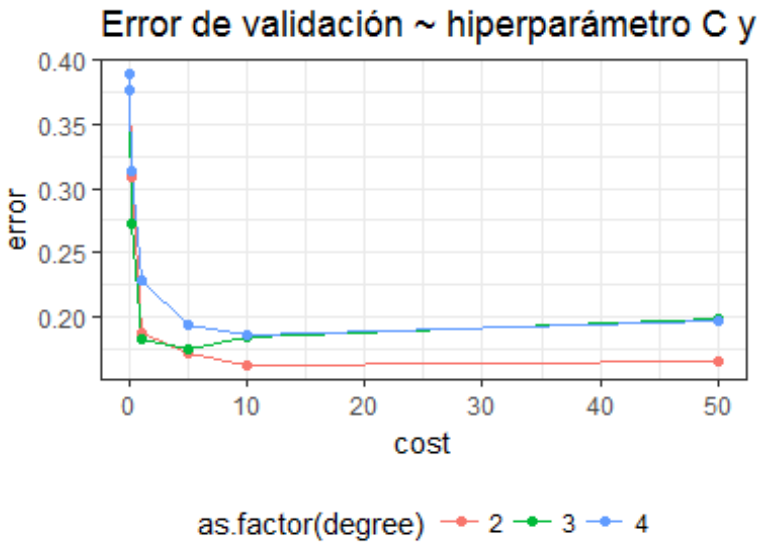
AJUSTE DEL MODELO

```
set.seed(32)

tuning <- tune(svm, Purchase ~ ., data = datos.OJ.train,
               kernel = "polynomial",
               ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 50),
                             degree = c(2, 3, 4)),
               scale = TRUE)

summary(tuning)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##    10      2
##
## - best performance: 0.1620657
##
## - Detailed performance results:
##     cost degree     error dispersion
## 1  1e-03      2 0.3897674 0.02576164
## 2  1e-02      2 0.3897674 0.02576164
## 3  1e-01      2 0.3092202 0.04817787
## 4  1e+00      2 0.1865800 0.03783712
## 5  5e+00      2 0.1713953 0.03666984
## 6  1e+01      2 0.1620657 0.03768158
## 7  5e+01      2 0.1655404 0.04664490
## 8  1e-03      3 0.3897674 0.02576164
## 9  1e-02      3 0.3768810 0.03312511
## 10 1e-01      3 0.2730643 0.02875124
## 11 1e+00      3 0.1830643 0.04126108
## 12 5e+00      3 0.1736936 0.04135789
## 13 1e+01      3 0.1841997 0.04622837
## 14 5e+01      3 0.1981532 0.05421692
## 15 1e-03      4 0.3897674 0.02576164
## 16 1e-02      4 0.3768947 0.03095133
## 17 1e-01      4 0.3139945 0.03556702
```

```
## 18 1e+00      4 0.2275103 0.02674522
## 19 5e+00      4 0.1935978 0.02626949
## 20 1e+01      4 0.1853899 0.03239505
## 21 5e+01      4 0.1970588 0.04414272
```

Con un kernel polinómico, los hiperparámetros óptimos que reducen el error de validación son *cost = 10*, *degree = 2*.

```
ggplot(data = tuning$performances,
       aes(x = cost, y = error, col = as.factor(degree))) +
geom_line() +
geom_point() +
labs(title = "Error de validación ~ hiperparámetro C y polinomio") +
theme(plot.title = element_text(hjust = 0.5)) +
theme_bw() + theme(legend.position = "bottom")
```




```
# Modelo SVM kernel polinómico
modelo.svm.poli <- svm(Purchase ~ ., data = datos.OJ.train,
                       kernel = "polynomial",
                       cost = 10,
                       degree = 2,
                       scale = TRUE)

summary(modelo.svm.poli)

##
## Call:
## svm(formula = Purchase ~ ., data = datos.OJ.train, kernel = "polynomial",
##     cost = 10, degree = 2, scale = TRUE)
##
```

```
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  2
##       gamma:  0.05555556
##      coef.0:  0
##
## Number of Support Vectors:  352
##
##  ( 179 173 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

El modelo ajustado con un kernel polinómico de grado 2 ha aumentado los vectores soporte de 341 (kernel lineal) a 352.

EVALUACIÓN DEL MODELO

```
pred <- predict(modelo.svm.poli, datos.OJ.test)


table(predicción = pred, real = datos.OJ.test$Purchase)
```

```
##           real
## predicción  CH  MM
##         CH 113  24
##         MM  17  59
```

```
paste("Error de test:",
      100 * mean(datos.OJ.test$Purchase != pred) %>% round(digits = 4), "%")
```

```
## [1] "Error de test: 19.25 %"
```

Con un valor de *cost = 10* y kernel *poli = 2*, el error de test iguala al obtenido con un kernel lineal.

KERNEL RADIAL

AJUSTE DEL MODELO

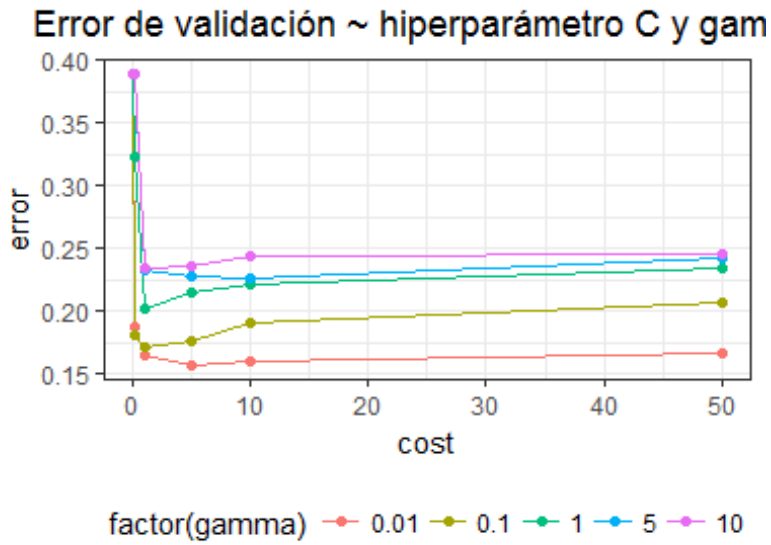
```
set.seed(32)
tuning <- tune(svm, Purchase ~ ., data = datos.OJ.train,
               kernel = "radial",
               ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 50),
                             gamma = c(0.01, 0.1, 1, 5, 10)),
               scale = TRUE)

summary(tuning)
```

```
 ##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     5  0.01
##
## - best performance: 0.1573735
##
## - Detailed performance results:
##     cost gamma     error dispersion
## 1  1e-03  0.01 0.3897674 0.02576164
## 2  1e-02  0.01 0.3897674 0.02576164
## 3  1e-01  0.01 0.1877291 0.03914293
## 4  1e+00  0.01 0.1655130 0.04943103
## 5  5e+00  0.01 0.1573735 0.04064681
## 6  1e+01  0.01 0.1608755 0.04764269
## 7  5e+01  0.01 0.1667168 0.04164827
## 8  1e-03  0.10 0.3897674 0.02576164
## 9  1e-02  0.10 0.3897674 0.02576164
## 10 1e-01  0.10 0.1819152 0.03988427
## 11 1e+00  0.10 0.1713953 0.03277592
## 12 5e+00  0.10 0.1772093 0.04077226
## 13 1e+01  0.10 0.1912038 0.04918271
## 14 5e+01  0.10 0.2075650 0.03919575
## 15 1e-03  1.00 0.3897674 0.02576164
## 16 1e-02  1.00 0.3897674 0.02576164
## 17 1e-01  1.00 0.3232421 0.02176631
## 18 1e+00  1.00 0.2028728 0.04437367
## 19 5e+00  1.00 0.2157319 0.04226528
## 20 1e+01  1.00 0.2215732 0.04036374
## 21 5e+01  1.00 0.2344186 0.03451818
## 22 1e-03  5.00 0.3897674 0.02576164
## 23 1e-02  5.00 0.3897674 0.02576164
## 24 1e-01  5.00 0.3886047 0.02454930
## 25 1e+00  5.00 0.2332421 0.03643499
## 26 5e+00  5.00 0.2274555 0.04141828
## 27 1e+01  5.00 0.2262791 0.04173383
## 28 5e+01  5.00 0.2426539 0.04209658
## 29 1e-03 10.00 0.3897674 0.02576164
```

```
## 30 1e-02 10.00 0.3897674 0.02576164
## 31 1e-01 10.00 0.3897674 0.02576164
## 32 1e+00 10.00 0.2344186 0.03977570
## 33 5e+00 10.00 0.2368263 0.04973500
## 34 1e+01 10.00 0.2438167 0.05099157
## 35 5e+01 10.00 0.2461970 0.04563073
```

Con un kernel radial, los hiperparámetros que reducen el error de validación son *cost* = 5, *gamma* = 0,01.

```
ggplot(data = tuning$performances,
       aes(x = cost, y = error, color = factor(gamma))) +
geom_line() +
geom_point() +
labs(title = "Error de validación ~ hiperparámetro C y gamma") +
theme_bw() +
theme(plot.title = element_text(hjust = 0.5)) +
theme(legend.position = "bottom")
```




```
# Modelo SVM con kernel radial
modelo.svm.rad <- svm(Purchase ~ ., data = datos.OJ.train,
                      kernel = "radial",
                      cost = 5,
                      gamma = 0.01,
                      scale = TRUE)
```

EVALUACIÓN DEL MODELO

```
pred <- predict(modelo.svm.rad, datos.OJ.test)
table(predicción = pred, real = datos.OJ.test$Purchase)

##           real
## predicción  CH  MM
##         CH 107  20
##         MM  23  63

paste("Error de test:",
      100 * mean(datos.OJ.test$Purchase != pred) %>% round(digits = 4), "%")

## [1] "Error de test: 20.19 %"
```

Con el uso de un kernel radial (*cost = 5* y *gamma = 0,01)*, el error de test aumenta al 20,19%.

BIBLIOGRAFÍA

An Introduction to Statistical Learning: with Applications in R (Springer Texts in Statistics)

https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/

https://topepo.github.io/caret/model-training-and-tuning.html