Máquinas de Soporte Vectorial

César Gómez

28/10/2020

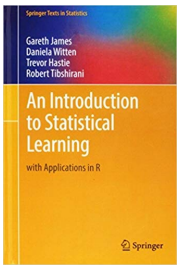


Figure 1: A nice image.

(Tomado textualmente de Introduction to Statistical Learning with Applications in R, by G. James,D. Witten, T.Hastie, R. Tibshirani)

Clasificador de Soporte Vectorial

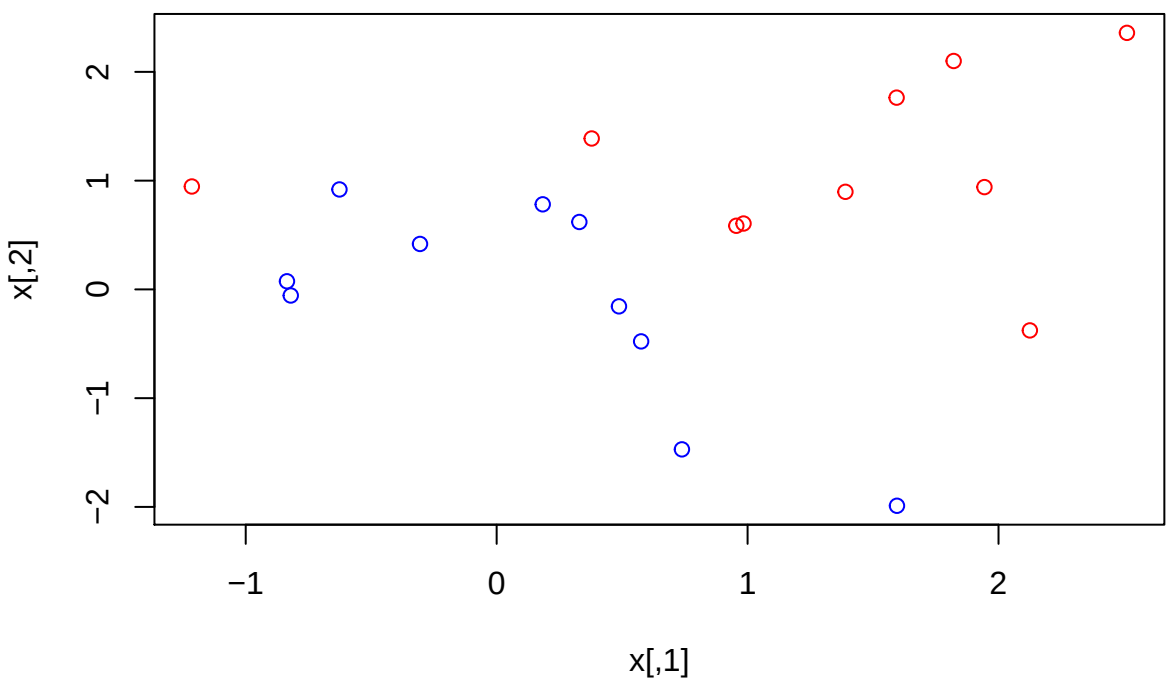
La libreria **e1071** de R, contiene implementaciones para una serie de métodos de aprendizaje estadístico. En particular, la función `svm()` se puede utilizar para ajustar un clasificador de soporte vectorial cuando se usa el argumento 'kernel = "linear"" . Esta función utiliza una formulación ligeramente diferente de (9.14) y (9.25) para el clasificador de soporte vectorial. El argumento **cost** nos permite especificar el costo de una violación de la margen. Cuando el argumento de costo es pequeño, entonces la márgen es ancha y muchos vectores de soporte estarán sobre la margen o invadiran el margen. Cuando el argumento de costo es grande, entonces la márgen es más estrecha y habrá pocos vectores de soporte en el margen o que invadan el margen.

Ahora se utilizará la función `svm()` para ajustar el clasificador de soporte vectorial para un valor dado del parámetro de **cost**. Acá se demostrará el uso de esta función en un ejemplo bidimensional, donde es posible gráficar la frontera de decisión resultante. Se Comenzará generando las observaciones que son de dos clases y se verificará si son linealmente separables.

```
set.seed (1)
x=matrix (rnorm (20*2) , ncol =2)
y=c(rep (-1,10) , rep (1 ,10) )
x[y==1 ,]= x[y==1,] + 1
```

Se verifica si las dos clases son linealmente separables

```
plot(x, col =(3-y))
```

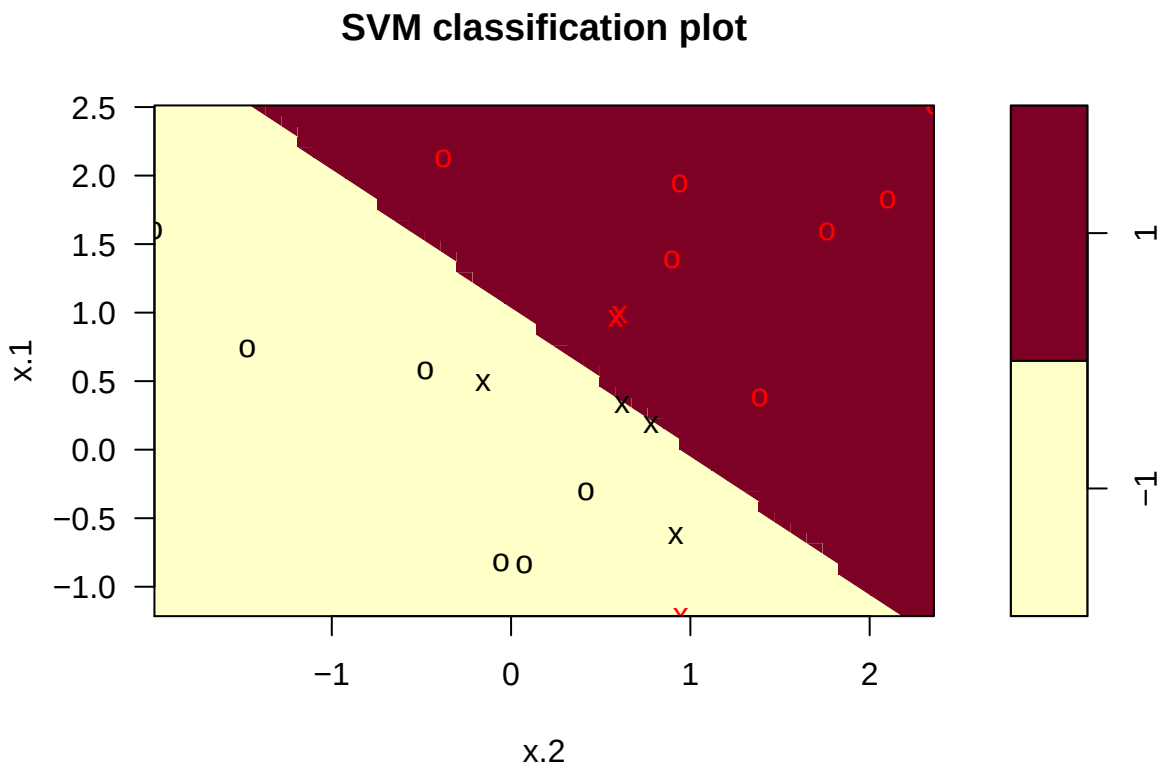


Las clases no son separables. A continuación, se ajustará el clasificador de soporte vectorial. Tenga en cuenta que para que la función **svm()** realice la clasificación (en oposición a la basada en SVM regresión), se debe codificar la respuesta como una variable de factor. Ahora se creará un dataframe con la respuesta codificada como factor.

```
dat=data.frame(x=x, y=as.factor(y))
library(e1071)
svmfit =svm(y~., data=dat , kernel ="linear", cost =10,scale =FALSE )
```

El argumento **scale = FALSE** le dice a la función **svm()** que no escale cada característica tener media cero o desviación estándar uno; dependiendo de la aplicación,uno podría preferir usar scale = TRUE. Ahora se puede trazar el clasificador de soporte vectorial obtenido :

```
plot(svmfit , dat)
```



Tengase en cuenta que los dos argumentos para la función **plot.svm()** son la salida de la llamada a **svm()**, así como los datos utilizados en la llamada a **svm()**. La región del espacio de atributos que se asignará a la clase -1 se muestra en amarillo claro, y la región que se asignará a la clase +1 se muestra en púrpura. La frontera de decisión entre las dos clases es lineal (porque se utilizó el argumento `kernel = "linear"`), aunque debido a la forma en que la función de trazado se implementa en esta libreria, la frontera de decisión se ve algo irregular en la trama. Vemos que en este caso solo una observación está mal clasificada. (Tengase en cuenta que acá el segundo atributo se traza en el eje x y el primero se traza en el eje y, en contraste con el comportamiento usual de la función `plot()` en R.) Los vectores de soporte se trazan como cruces y las observaciones restantes se trazan como círculos; Se ve acá que son siete vectores de soporte.

Y se Puede determinar las identidades de estos vectores de soporte de la siguiente manera:

```
svmfit$index
```

```
## [1]  1  2  5  7 14 16 17
```

Se puede obtener alguna información básica sobre el clasificador de soporte vectorial utilizando el comando `summary()`

```
summary(svmfit)
```

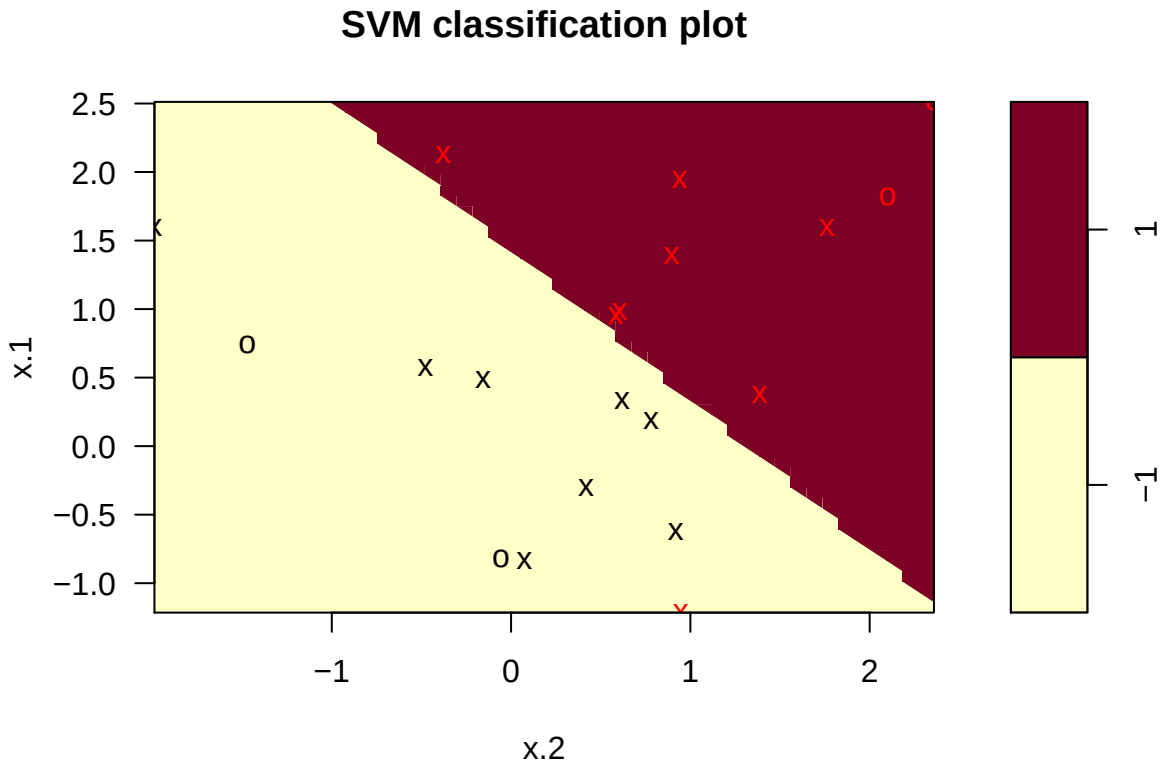
```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
```

```
##        cost:  10
##
## Number of Support Vectors:  7
##
##  ( 4 3 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

Esto informa por ejemplo, que el kernel lineal ha sido utilizado con **cost=10**, y que se obtuvieron 7 vectores de soporte, 3 en una clase y 4 en la otra.

Que sucede si se utiliza un valor más pequeño del parámetro **cost**

```
svmfit = svm(y~., data=dat , kernel ="linear", cost =0.1,scale =FALSE )
plot(svmfit,dat)
```




```
svmfit$index
```

```
##  [1]  1  2  3  4  5  7  9 10 12 13 14 15 16 17 18 20
```

- Ahora que se está utilizando un valor menor del parámetro **cost**, se obtiene un mayor número de vectores de soporte, porque el margen ahora es más amplio. Desafortunadamente, la función `svm()` no genera explícitamente los coeficientes de la frontera de decisión lineal obtenida cuanda el clasificador de soporte vectorial es ajustado, ni da salida de la magnitud del ancho del margen.

- La librería **e1071** incluye una función incorporada, `tune()`, para realizar validación cruzada. Por defecto, `tune()` realiza una validación cruzada de diez folds en un conjunto de modelos de interés. Para utilizar esta función, se pasa información relevante sobre el conjunto de modelos que se están considerando.
- El siguiente comando indica que se quiere comparar SVM con un lineal kernel, utilizando un rango de valores del parámetro **cost**.

```
set.seed(1)
tune.out=tune(svm ,y~.,data=dat ,kernel ="linear",
ranges =list(cost=c(0.001 , 0.01, 0.1, 1,5,10,100) ))
```

Se puede acceder a los errores de validación cruzada para cada uno de estos modelos, utilizando la función **summary()**

```
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.05
##
## - Detailed performance results:
##    cost error dispersion
## 1 1e-03  0.55  0.4377975
## 2 1e-02  0.55  0.4377975
## 3 1e-01  0.05  0.1581139
## 4 1e+00  0.15  0.2415229
## 5 5e+00  0.15  0.2415229
## 6 1e+01  0.15  0.2415229
## 7 1e+02  0.15  0.2415229
```

Se ve que **cost=0.1** proporciona el menor error de validación cruzada. La función **tune()** guarda el mejor modelo obtenido, que puede ser recuperado del siguiente modo

```
bestmod = tune.out$best.model
summary(bestmod)
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = dat, 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:  0.1
##
## Number of Support Vectors:  16
##
```

```
##  ( 8 8 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

La función **predict()** se puede utilizar para predecir la etiqueta de una clase en un conjunto de observaciones de prueba, para cualquier valor dado del parámetro de costo. Comenzamos por generar un conjunto de datos de prueba

```
xtest= matrix(rnorm (20*2) , ncol =2)
ytest= sample(c(-1,1) , 20, rep=TRUE)
xtest[ytest ==1 ,]= xtest[ytest ==1,] + 1
testdat =data.frame(x=xtest , y=as.factor(ytest))
```

Ahora se predicen las etiquetas de clase de estas observaciones de prueba. Acá se utilizará el mejor modelo obtenido mediante validación cruzada para hacer predicciones.

```
ypred = predict(bestmod ,testdat)
table(predict =ypred , truth= testdat$y )
```

```
##        truth
## predict -1 1
##      -1  9 1
##      1   2 8
```

Asi con el valor asignado a **cost** 17 de las observaciones se han clasificado de forma correcta, Que sucede si se toma **cost=0.01**

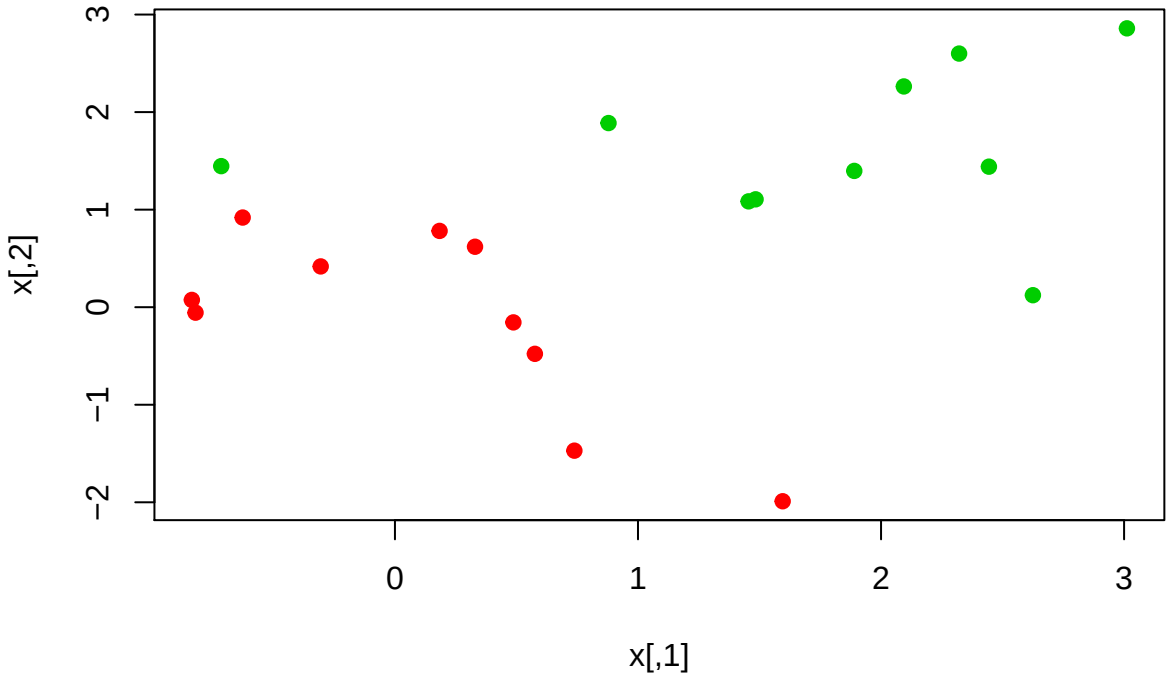
```
svmfit = svm(y~., data=dat , kernel ="linear", cost =.01,
scale =FALSE )
ypred = predict(svmfit ,testdat)
table(predict =ypred , truth= testdat$y )
```

```
##        truth
## predict -1  1
##      -1 11  6
##      1   0  3
```

En esta caso 3 más de las observaciones no se han clasificado correctamente

Ahora considere una situación en la que las dos clases son linealmente separables. Entonces podemos encontrar un hiperplano de separación usando la función **svm()**.Primero se separan las dos clases de los datos simulados para que sean linealmente separable:

```
x[y==1 ,] = x[y==1 ,]+0.5
plot(x, col =(y+5) /2, pch =19)
```

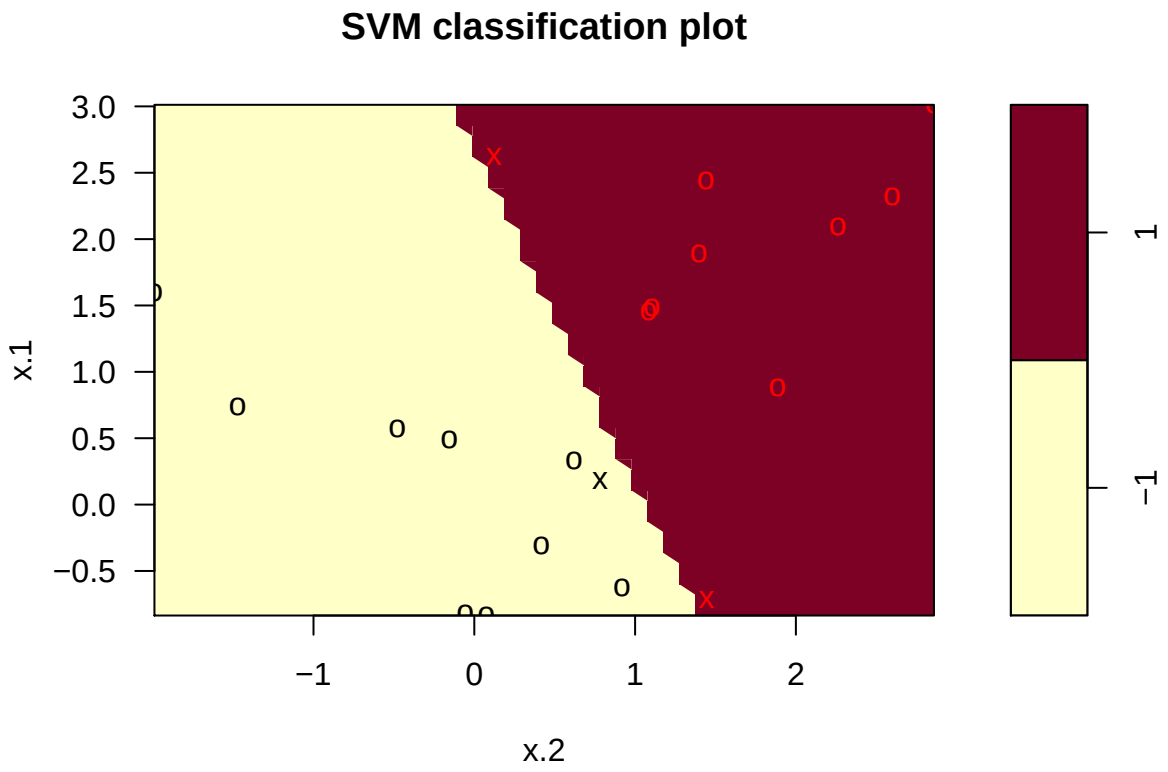


Ahora las observaciones son casi linealmente separables. Se ajusta el clasificador de soporte vectorial y se traza el hiperplano resultante, utilizando un valor muy grande de **cost** para que no hayan observaciones mal clasificadas.

```
dat = data.frame(x=x,y=as.factor(y))
svmfit = svm(y~., data=dat , kernel ="linear", cost =1e5)
summary(svmfit )
```

```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 1e+05)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1e+05
##
## Number of Support Vectors:  3
##
##  ( 1 2 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

```
plot(svmfit,dat)
```



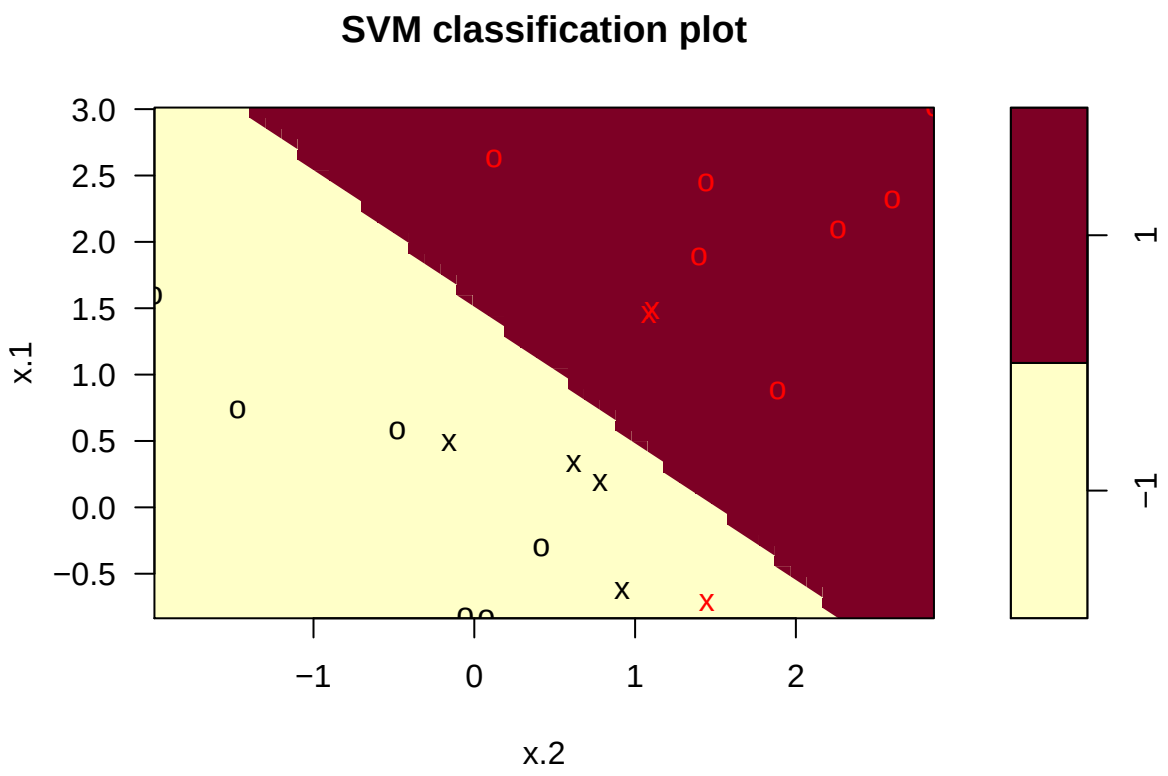
No se cometieron errores en el conjunto de entrenamiento y solo se utilizaron tres vectores de soporte. Sin embargo, se puede ver en la figura que el margen es muy estrecho (porque las observaciones que no son vectores de soporte, indicadas como circulos, estón muy cerca de la frontera de decisión). **Es probable que este modelo funcione mal** en los datos de prueba. Ahora se intenta con un menor valor de **cost**:

```
svmfit =svm(y~., data=dat , kernel ="linear", cost =1)
summary(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  7
##
##  ( 4 3 )
##
##
## Number of Classes:  2
##
```

```
## Levels:
##  -1 1
```

```
plot(svmfit,dat)
```



Utilizando **cost = 1**, se clasificó erróneamente una observación de entrenamiento, pero también se obtiene un margen mucho más amplia y se hace uso de siete vectores de soporte. Parece probable que este modelo tenga un mejor rendimiento en los datos de prueba que el modelo con **cost = 1e5**.

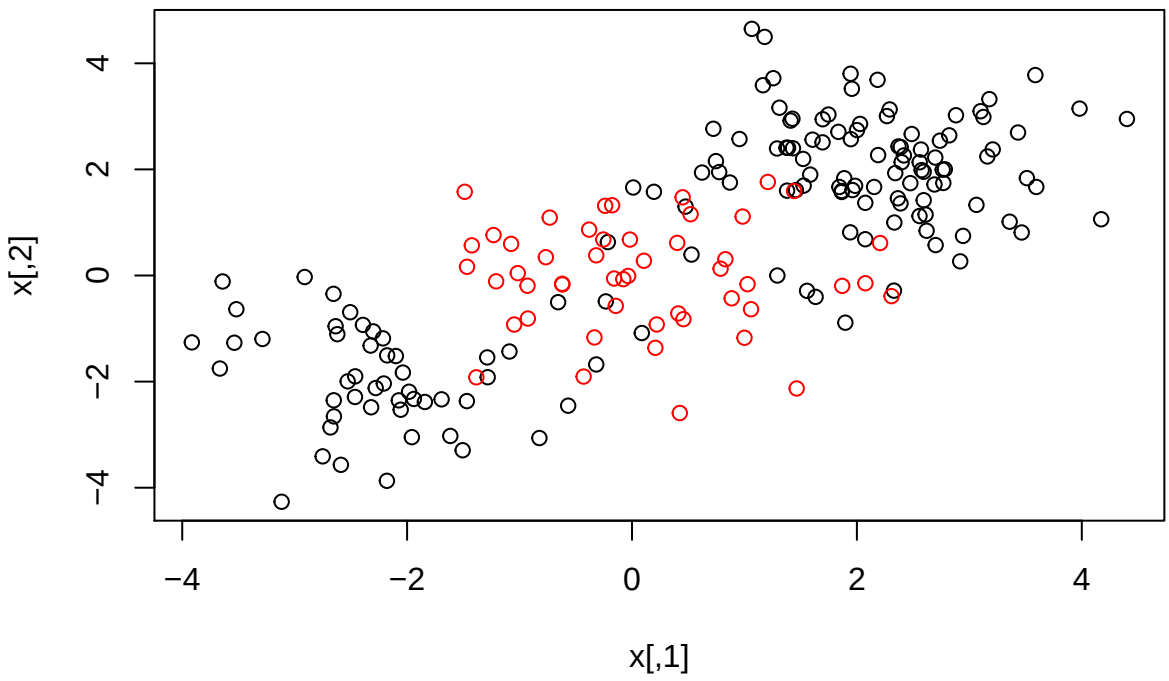
Máquina de Soporte Vectorial

Para ajustar una SVM que utilece un núcleo no lineal, una vez más se utiliza la función **svm()**. Sin embargo, ahora se utilizará un valor diferente para el parámetro **kernel**. Para ajustar un SVM con un núcleo polinomial se usa **kernel = "polynomial"**, y para ajustar un SVM con un núcleo radial usamos **kernel = "radial"**. En el primer caso también se utiliza el argumento **degree** para especificar un grado para el *kernel polinomial* (este es d en la ecuación (9.22)), y en este último caso se utiliza el parámetro **gamma** para especificar un valor de γ para el núcleo de base radial en la ecuación (9.24). Primero se generan algunos datos con una frontera de clase no lineal:

```
set.seed (1)
x = matrix(rnorm(200*2) , ncol =2)
x[1:100 ,]= x[1:100 ,]+2
x[101:150 ,]= x[101:150 ,] -2
y= c(rep(1 ,150) ,rep(2 ,50) )
dat= data.frame(x=x,y=as.factor(y))
```

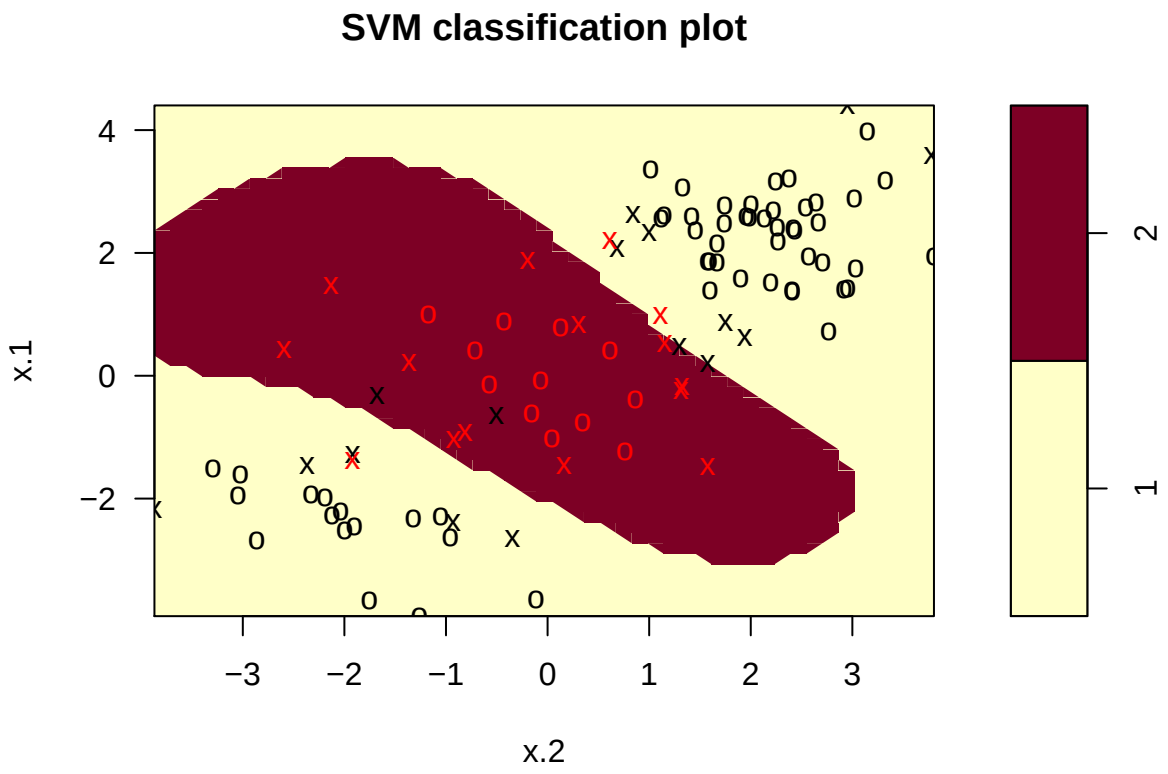
Graficar los datos, para apreciar que si se trata de una frontera no lineal

```
plot(x,col = y)
```



Los datos son divididos de forma aleatoria en un conjunto de entranamiento y un conjunto de prueba . Entonces se ajusta una MSV con los datos de entrenamiento, utilizando la función **svm()** y un **kernel radial** con $\gamma = 1$:

```
train = sample(200,100)
svmfit = svm(y~., data = dat[train ,], kernel = "radial", gamma =1,cost =1)
plot(svmfit , dat[train ,])
```



El gráfico muestra que la SVM resultante posee una frontera de carácter decididamente no lineal. La función **summary()** se puede utilizar para obtener alguna información sobre el ajuste SVM:

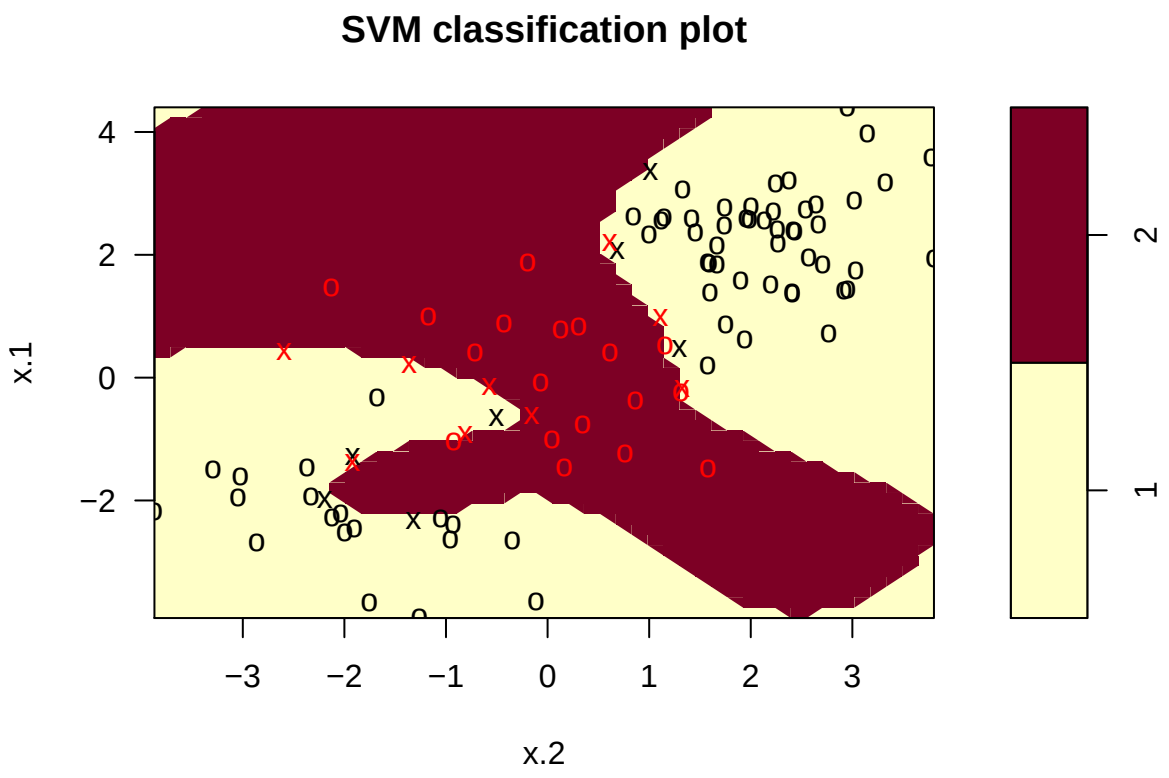
```
summary(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat[train, ], kernel = "radial", gamma = 1,
##     cost = 1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  31
##
##  ( 16 15 )
##
##
## Number of Classes:  2
##
## Levels:
##  1 2
```

Se puede ver en la figura que hay un buen número de errores de entrenamiento en este ajuste SVM. Si

aumentamos el valor del parámetro **cost**, podemos reducir el número de errores de entrenamiento. Sin embargo, esto tiene el precio de una frontera de desición más irregular y parece haber riesgo de estar sobreajustando los datos.

```
svmfit =svm(y~., data=dat [train ,], kernel = "radial",gamma =1,cost=1e5)
plot(svmfit ,dat [train ,])
```



Se puede llevar a cabo una validación cruzada utilizando la función **tune()** para seleccionar el valor más adecuado del parámetro γ y **cost** para una SVM con kernel radial.

```
tune.out=tune(svm , y~., data=dat[train ,], kernel = "radial",
ranges =list(cost=c(0.1 ,1 ,10 ,100 ,1000),gamma=c(0.5,1,2,3,4) ))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1   0.5
##
## - best performance: 0.06
##
## - Detailed performance results:
##     cost gamma error dispersion
## 1  1e-01   0.5  0.27 0.14944341
```

```
## 2  1e+00   0.5  0.06 0.08432740
## 3  1e+01   0.5  0.07 0.08232726
## 4  1e+02   0.5  0.10 0.09428090
## 5  1e+03   0.5  0.12 0.10327956
## 6  1e-01   1.0  0.18 0.16865481
## 7  1e+00   1.0  0.07 0.09486833
## 8  1e+01   1.0  0.09 0.09944289
## 9  1e+02   1.0  0.10 0.08164966
## 10 1e+03   1.0  0.09 0.09944289
## 11 1e-01   2.0  0.26 0.15776213
## 12 1e+00   2.0  0.08 0.11352924
## 13 1e+01   2.0  0.12 0.12292726
## 14 1e+02   2.0  0.12 0.13165612
## 15 1e+03   2.0  0.13 0.10593499
## 16 1e-01   3.0  0.27 0.13374935
## 17 1e+00   3.0  0.09 0.11005049
## 18 1e+01   3.0  0.10 0.13333333
## 19 1e+02   3.0  0.13 0.10593499
## 20 1e+03   3.0  0.15 0.10801234
## 21 1e-01   4.0  0.27 0.13374935
## 22 1e+00   4.0  0.09 0.12866839
## 23 1e+01   4.0  0.10 0.13333333
## 24 1e+02   4.0  0.15 0.13540064
## 25 1e+03   4.0  0.17 0.13374935
```

Por lo tanto, la mejor elección de parámetros parece ser **cost = 1** y $\gamma = 0.5$!!!. Se pueden ver las predicciones del conjunto de pruebas para este modelo aplicando la función **predict()** a los datos. Tengase en cuenta que para hacer esto se toma una muestra del dataframe **dat** utilizando **-train** como un conjunto de índices.

```
table(true=dat[-train ,"y"], pred=predict(tune.out$best.model,
newdata=dat[-train ,]))
```

```
##     pred
## true  1  2
##    1 67 10
##    2  2 21
```

```
print((67+21)/100)
```

```
## [1] 0.88
```

el 12% de las observaciones han sido erroneamente clasificadas por esta SVM.

Cuvas ROC

El paquete **ROCR** se puede utilizar para producir curvas ROC como las de las Figuras 9.10 y 9.11. Primero se escribe una función corta para trazar una curva ROC dado un vector **pred** que contiene una puntuación (score) numérica para cada observación, y Un vector **truth** que contiene la etiqueta de clase para cada observación.

```
###   ----------------------------  instalar paquete ROCR   ------------------------
library(ROCR)
rocplot = function(pred,truth, ...){
  predob = prediction(pred , truth )
  perf = performance(predob,"tpr","fpr")
  plot(perf, ...)}
```

Las SVM y los clasificadores de soporte vectorial generan etiquetas de clase para cada observación. Ahora bien, también es posible obtener valores *ajustados* para cada observación, los cuáles son los puntajes numéricos utilizados para obtener las etiquetas de clase. Por ejemplo, en el caso de un clasificador de soporte vectorial, el valor ajustado para una observación $X = (X_1, X_2, ..., X_p)^T$ toma la forma

$$\hat{f}(X) = \hat{\beta}_0 + \hat{\beta}_1 X_1 + \hat{\beta}_2 X_2 + \cdots + \hat{\beta}_p X_p. \tag{1}$$

Para una SVM con un núcleo (kernel) no lineal, la ecuación que proporciona el valor ajustado corresponde a lal ecuación (9.23) del libro.

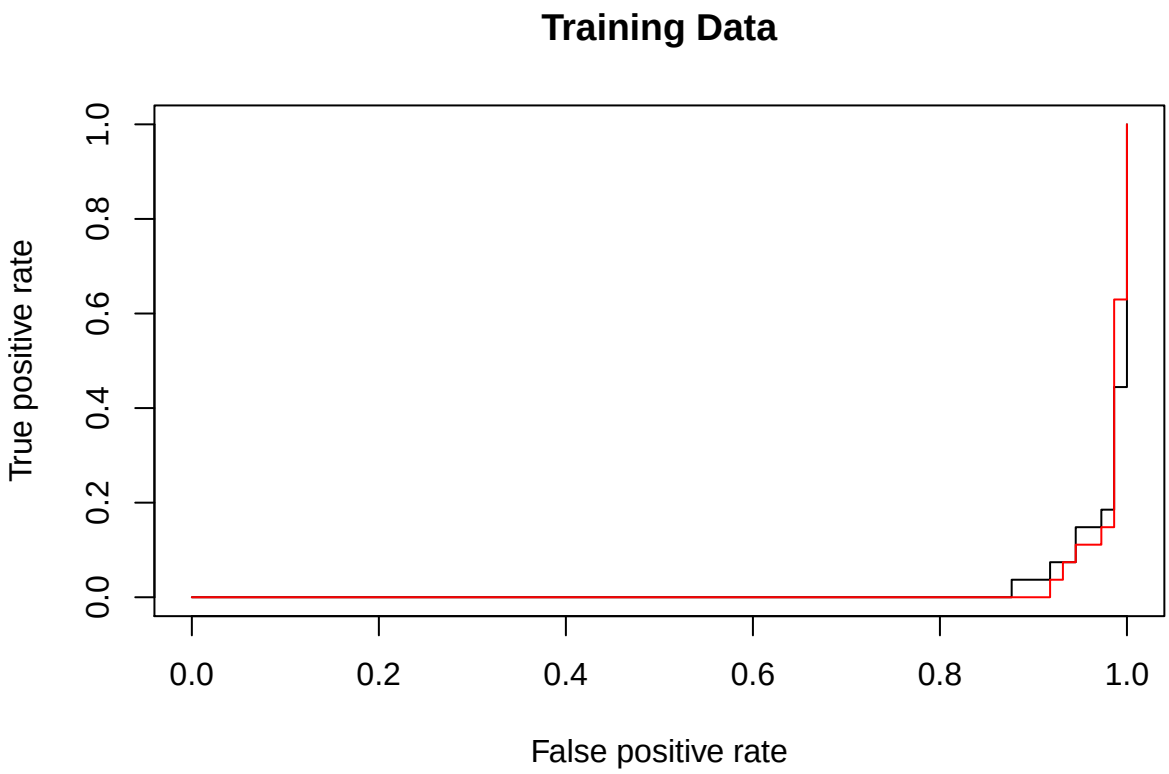
$$\hat{f}(x) = \hat{\beta}_0 + \sum_{i \in \mathcal{S}} \hat{\alpha}_i K(x, x_i)$$

En esencia, el signo del valor ajustado determina de qué lado de la frontera de decisión se encuentra la observación. Por lo tanto, la relación entre el valor ajustado y la predicción de clase para una determinada observación es simple: si el valor ajustado excede cero, entonces la observación se asigna a una clase, y si es menor que cero, se asigna a la otra clase. Para obtener los valores ajustados de un modelo SVM dado, se utiliza **decision.values= TRUE** cuando se ajusta con la fucnción **svm()**. Entonces la función **predic()** entregará los valores ajustados.

```
svmfit.opt=svm(y~.,data=dat[train ,], kernel ="radial",
gamma =0.5, cost=1, decision.values =T)
fitted = attributes(predict (svmfit.opt ,dat[train ,], decision.values =TRUE))$decision.values
```

Ahora se puede graficar la curva ROC

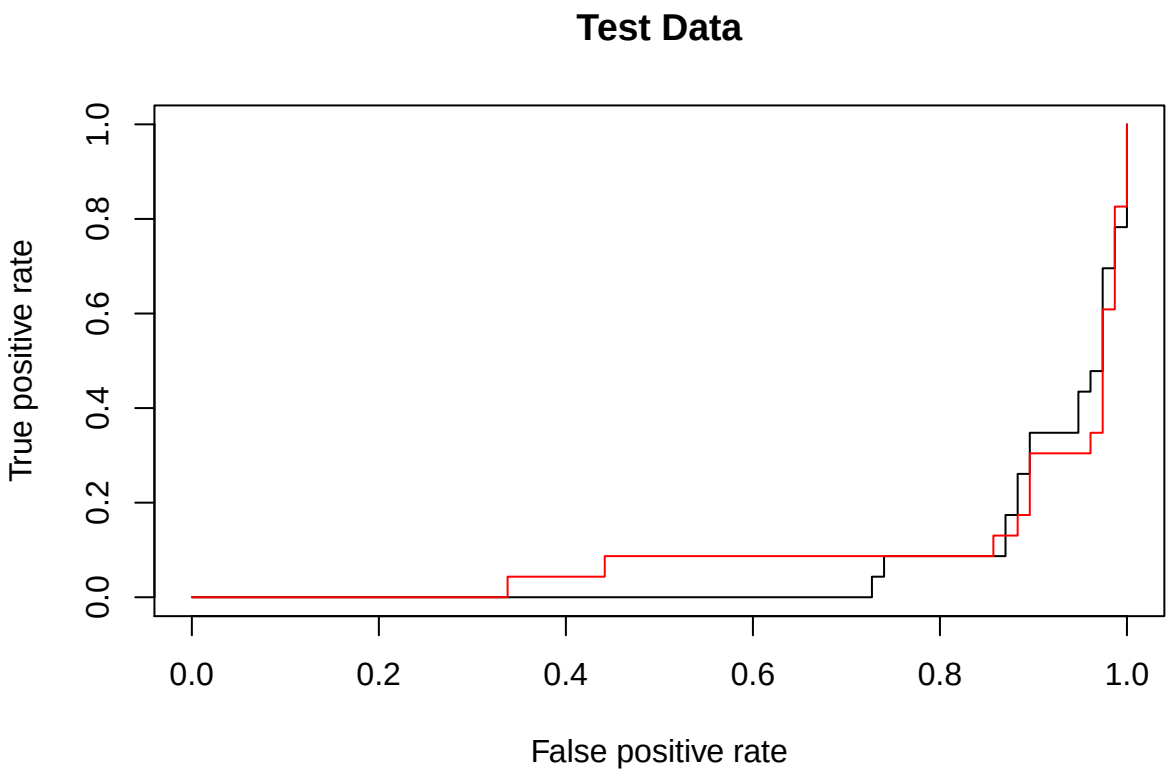
```
rocplot(fitted ,dat[train ,"y"], main="Training Data")

svmfit.flex = svm(y~., data=dat[train ,], kernel ="radial",
gamma = 1.5, cost=1, decision.values =T)
fitted = attributes(predict(svmfit.flex, dat[train, ], decision.values =T))$decision.values

rocplot(fitted, dat[train,"y"], add= T, col = "red")
```



La SVM parece estar produciendo predicciones precisas. Al aumentar γ se puede conseguir un ajuste más flexible y obtener alguna mejora en la precisión.

Sin embargo, estas curvas ROC están realizadas sobre los datos de entrenamiento. El verdadero interés se centra el nivel de precisión de las predicciones en los datos de prueba. Cuando se calculan las curvas ROC en los datos de prueba, el modelo con $\gamma = 0.5$ parece proporcionar los resultados más precisos.

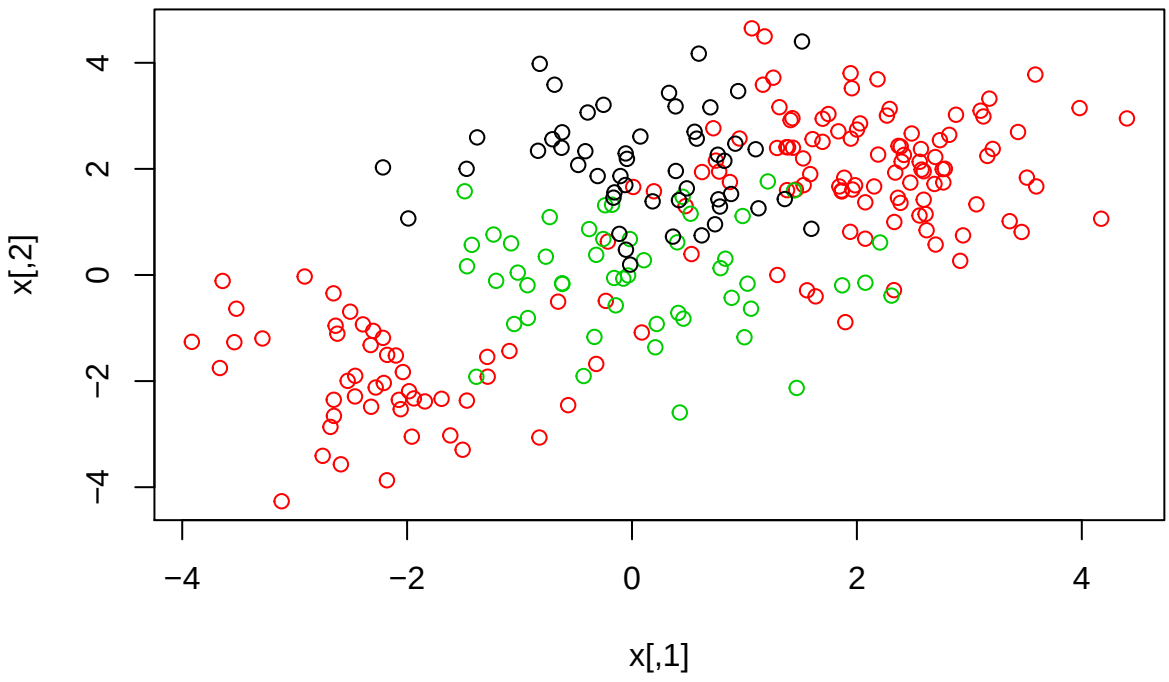
```
fitted = attributes(predict(svmfit.opt ,dat[-train ,], decision.values =T))$decision.values
rocplot(fitted ,dat[-train ,"y"], main ="Test Data")

fitted = attributes(predict(svmfit.flex,dat[-train ,], decision.values =T))$decision.values
rocplot(fitted,dat[-train ,"y"], add=T,col ="red")
```



SVM para multiples clases

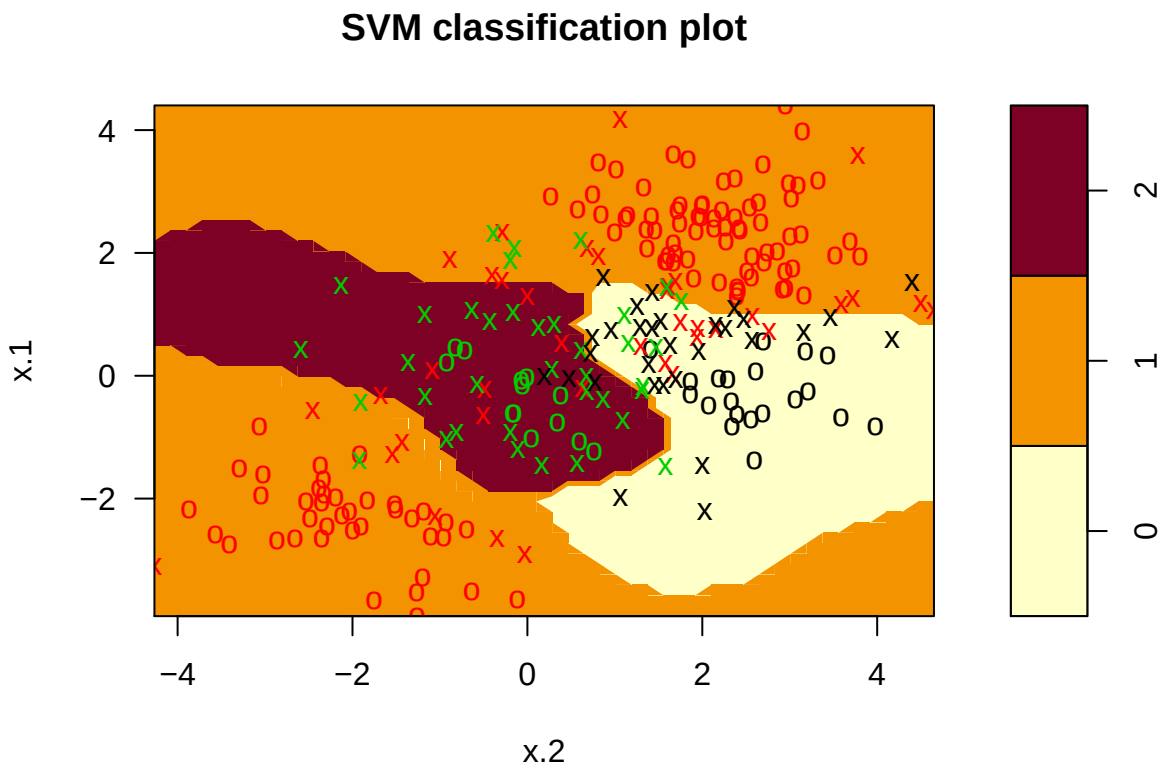
Si la respuesta es un factor que contiene más de dos niveles, entonces la función `svm()` realizará una clasificación de varias clases utilizando el enfoque uno vs uno. se explorará esa configuración aquí generando una tercera clase de observaciones.

```
set.seed (1)
x=rbind(x, matrix (rnorm (50*2) , ncol =2))
y=c(y, rep (0 ,50) )
x[y==0 ,2]= x[y==0 ,2]+2
dat=data.frame(x=x, y=as.factor (y))
par(mfrow =c(1,1))
plot(x,col =(y+1))
```



Ahora ajustamos una máquina de soporte vectorial a estops datos con `svm()`

```
svmfit =svm(y~., data=dat , kernel ="radial", cost =10, gamma =1)
plot(svmfit , dat)
```



La librería `e1071` también se puede utilizar para realizar regresión con máquinas de soporte vectorial, si el vector de respuesta que se pasa a `svm()` es numérico en lugar de un factor.

Aplicación a datos de expresión génica

Ahora examinamos el conjunto de datos de `Khan`, que consiste en una serie de muestras de tejidos correspondientes a cuatro tipos distintos de tumores pequeños de células azules redondas. Para cada muestra de tejido, las medidas de expresión génica están disponibles. El conjunto de datos consta de datos de entrenamiento, `xtrain` e `ytrain`, y datos de prueba, `xtest` y `ytest`. Antes se examina la dimensión de los datos

```
library (ISLR)
 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
```

Este conjunto de datos consiste en medidas de expresión para 2.308 genes. Los conjuntos de entrenamiento y prueba consisten en 63 y 20 observaciones respectivamente.

```
table(Khan$ytrain)
```

```
##
##  1  2  3  4
##  8 23 12 20
```

```
table(Khan$ytest )
```

```
##
## 1 2 3 4
## 3 6 6 5
```

Se utilizará una máquina de soporte vectorial para predecir el subtipo de cáncer basandose en las mediciones de expresión génica. En este conjunto de datos, hay un gran número de atributos en relación al número de observaciones. Esto sugiere utilizar un núcleo lineal, porque la flexibilidad adicional que resultará de usar un núcleo polinomial o radial es innecesario.

```
dat=data.frame(x=Khan$xtrain , y=as.factor ( Khan$ytrain ))
out=svm(y~., data=dat , kernel ="linear",cost =10)
summary(out)
```

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

Vemos que no hay errores de entrenamiento. De hecho, esto no es sorprendente, porque la gran cantidad de variables en relación con la cantidad de observaciones implica que es fácil encontrar hiperplanos que separen completamente las clases. El interés se centra no tanto en el rendimiento del clasificador en las observaciones de entrenamiento, sino más bien su desempeño en las observaciones de prueba.

```
dat.te=data.frame(x=Khan$xtest , y=as.factor (Khan$ytest ))
pred.te=predict (out , newdata =dat.te)
table(pred.te , dat.te$y)
```

```
##
## pred.te 1 2 3 4
##       1 3 0 0 0
##       2 0 6 2 0
##       3 0 0 4 0
```

```
##        4 0 0 0 5
```

se puede ver que utilizando `cost = 10` produce dos errores en el conjunto de prueba en estos datos.