SVM - Parte III

Fernando Anorve

4/2/2020

Máquinas de soporte vectorial

¿Qué hemos visto hasta ahora?

Abordamos el problema de clasificación: tenemos parejas de observaciones

$$(X_1, y_1), (X_2, y_2), \ldots, (X_n, y_n),$$

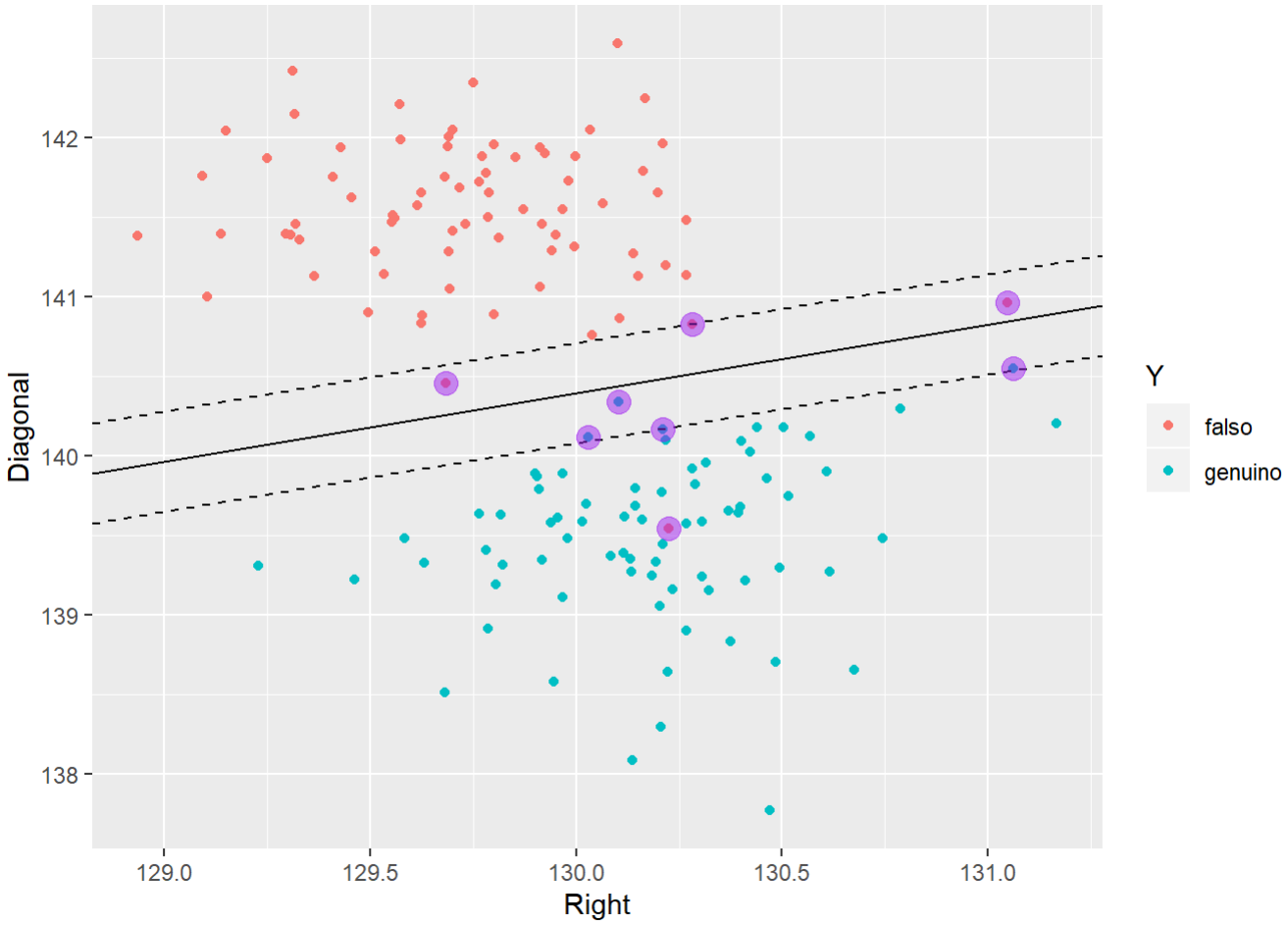
donde $X_i = (x_{1i}, x_{2i}, \ldots, x_{pi})^T \in \mathbb{R}^p$ y $y_i \in \{-1, 1\}$

- **Clasificación por hiperplano:**

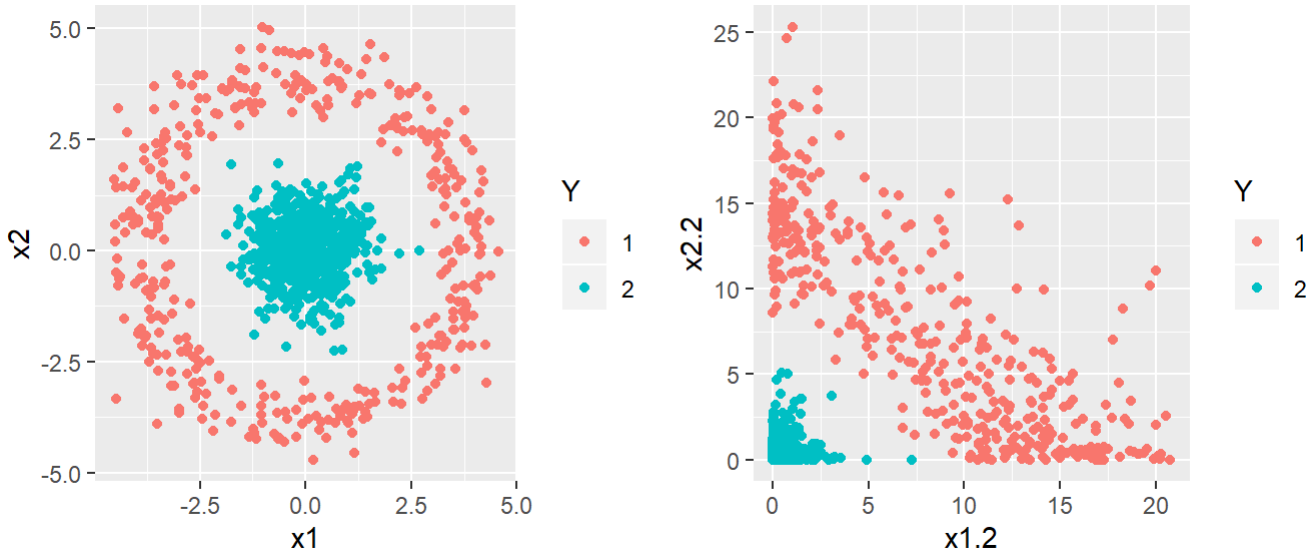
  $$f(x) = x^T\beta + \beta_0,$$

  cuya regla de clasificación es egla de clasificación $G(x) = \text{sign}(f(x))$.

- **Cómo hallar parámetros:** Habíamos usado la función tune() para realizar selección de parámetros basados en los errores de validación cruzada
  - Es importante notar que no conviene utilizar LOOCV (validación cruzada que deja una observación fuera)



- **Expansión del espacio de variables:** No siempre se puede separar el espacio linealmente, por lo que incorporamos transformaciones de las variables que amplíen el espacio. Por ejemplo, habíamos incorporado x_1^2 y x_2^2.



- En el espacio extendido la idea es hallar una separación lineal, que trasladado al espacio original es una separación no lineal

- Lo que antes teníamos era $f(x) = x^T\beta + \beta_0$, y ahora tendríamos

$$f(x) = h(x)^T\beta + \beta_0$$

en el caso de términos de segundo orden se tiene

$$h(x) = (h_1(x), h_2(x), \ldots, h_m(x)) = (x_1, x_2, x_1^2, x_2^2, x_1x_2)$$

- Sin embargo, extender el espacio puede resultar costoso. Por ejemplo, para extender el espacio a términos de grado 3:

$$h(x) = (x_1, x_2, x_1^2, x_2^2, x_1x_2, x_1^3, x_2^3, x_1^2x_2, x_1x_2^2)$$

Hay una forma de generalizar esto:

- Recordemos que hay que hallar el valor de β
- Utilizando multiplicadores de Lagrange, el valor $\hat{\beta}$ tendría la forma

$$\hat{\beta} = \sum_{i=1}^{N} \hat{\alpha}_i y_i h(x_i).$$

Curiosamente, $\hat{\alpha}_i$ es distinto de cero sólo para los i que corresponden a vectores de soporte

¿Cómo quedaría entonces la función $f(x)$ para clasificar?

$$f(x) = \sum_{i=1}^{N} \alpha_i y_i h(x)^T h(x_i) + \beta_0 = \sum_{i=1}^{N} \alpha_i y_i \langle h(x), h(x_i) \rangle + \beta_0$$

* Es importante notar que sólo depende del producto interior $\langle h(x), h(x_i) \rangle$

- Podemos centrar toda la atención a este producto anterior con el fin de no extender el espacio innecesariamente

A este producto interior le llamamos Kernel. Trabajar con el Kernel resulta más sencillo porque sólo nos interesa el resultado de $\langle h(x), h(x_i) \rangle$,

- Para trabajar con un kernel polinomial,

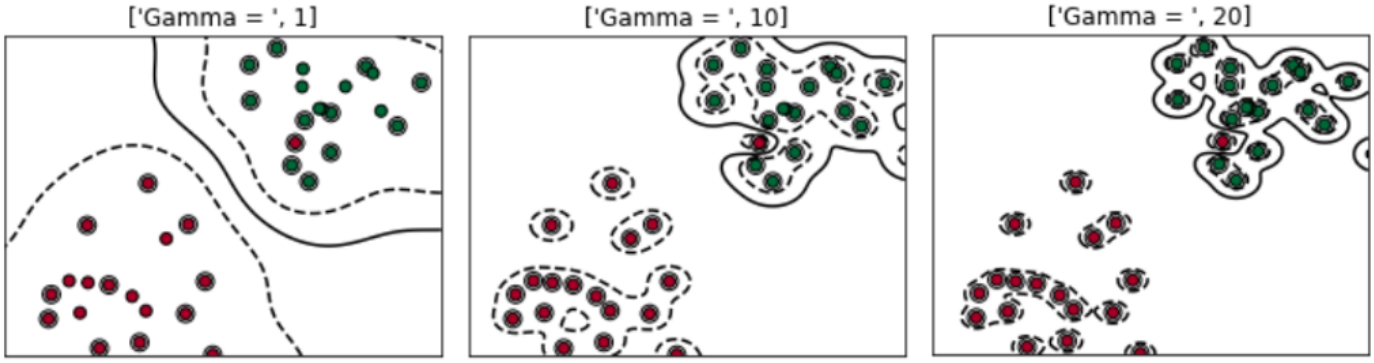
$$K(x, x') = (1 + \langle x, x' \rangle)^d$$

- Para trabajar con un kernel radial

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

Cuando una observación de prueba x^* está muy lejos de una observación de entrenamiento x_i, ocurre que $\|x - x'\|^2$ tiene un valor alto, lo que implica que $-\gamma\|x - x'\|^2$ tiene un valor negativo muy bajo, por lo que $\exp(-\gamma\|x - x'\|^2)$ tiene un valor cercano a cero. Es decir, el kernel aplicado en x_i aporta muy poco

$$f(x) = \sum_{i=1}^{N} \alpha_i y_i K(x, x_i) + \beta_0$$



- Kernel de red neural

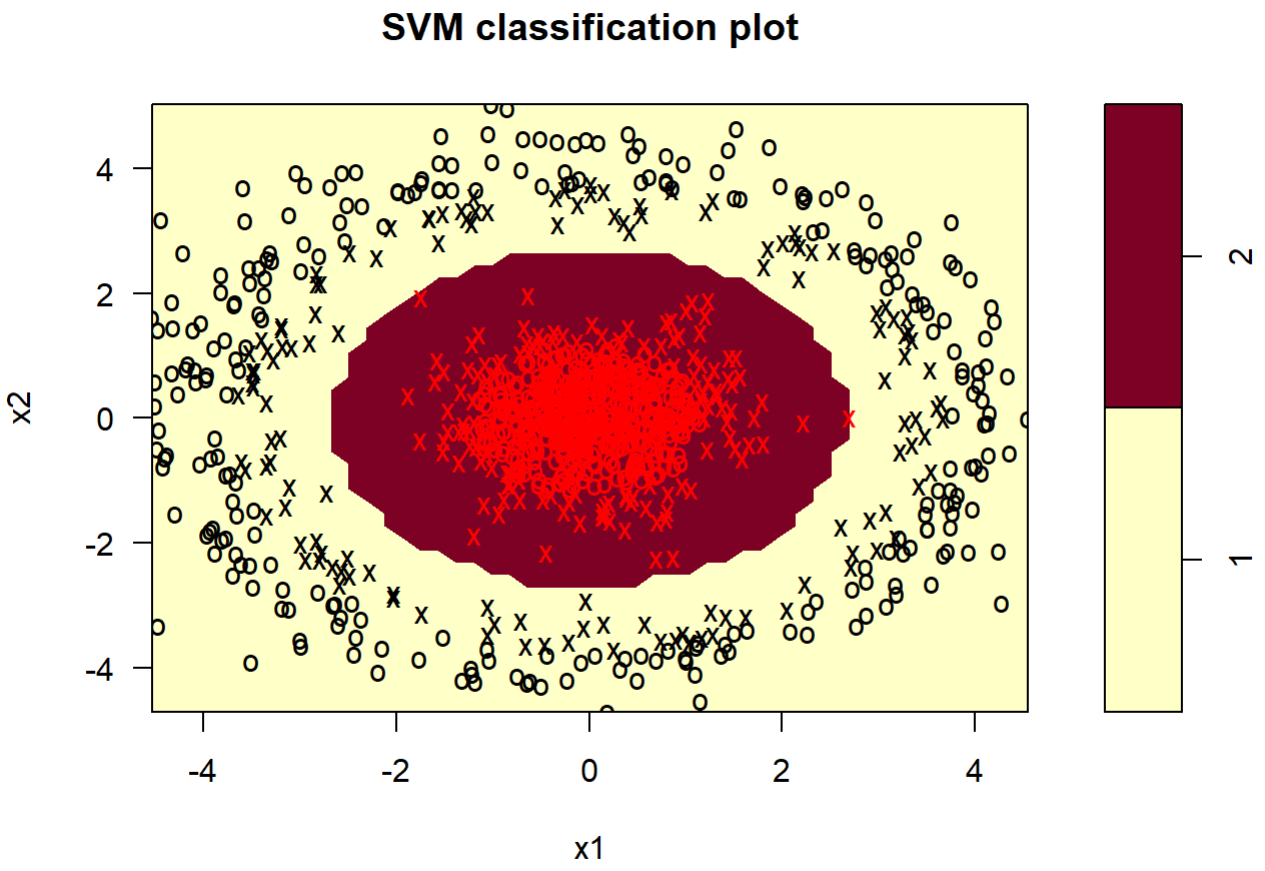
$$K(x, x') = \tanh(\kappa_1 \langle x, x' \rangle + \kappa_2)$$

Diferencias entre kernels

Ejemplo clásico

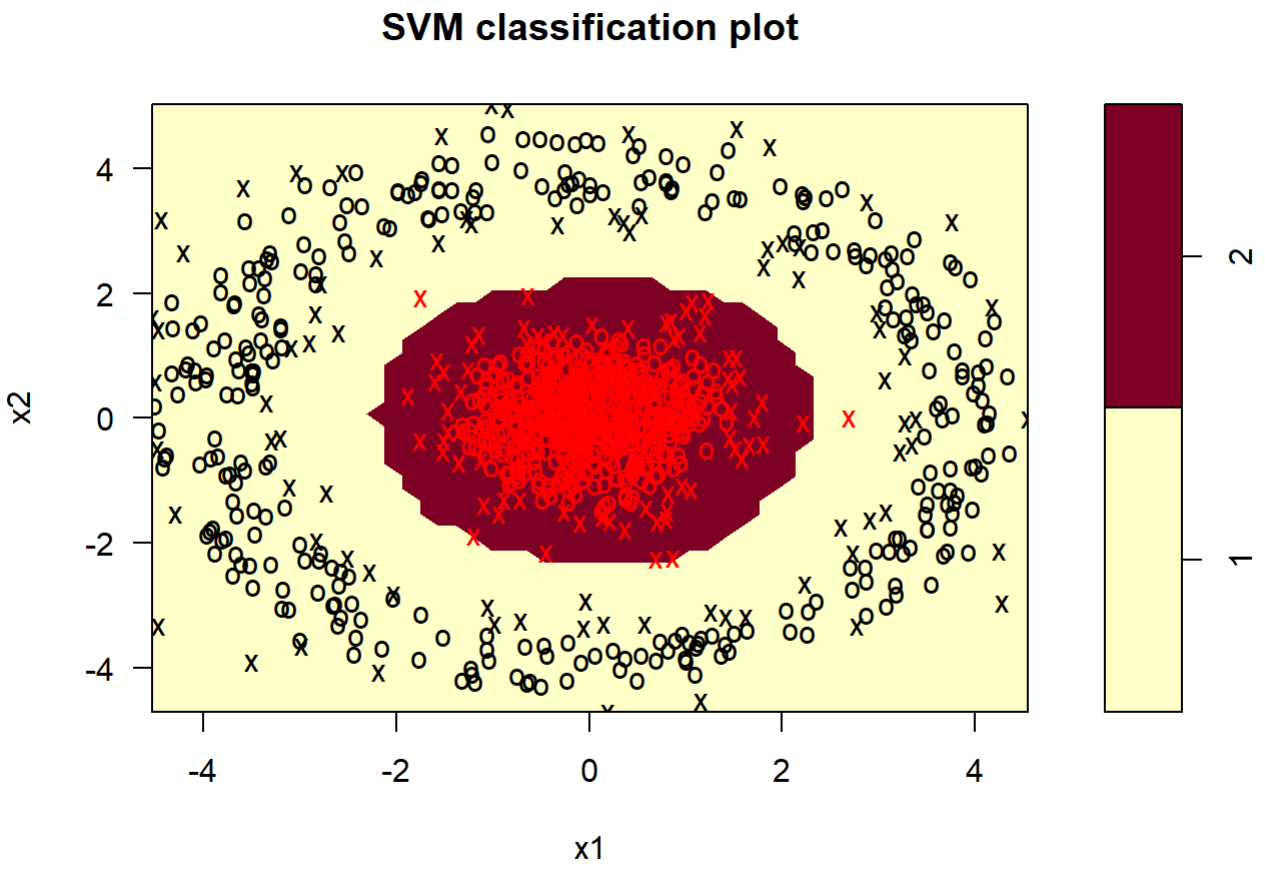
Modelo polinomial, grado 2

```
tune.out = tune(svm, Y ~ x2 + x1 , data = datos_ejemplo, kernel = "polynomial", degree = 2 , sca
le = FALSE, ranges = list ( cost = c(0.001 , 0.01 , 0.1 , 1 ,5 ,10 ,100) ))

plot(tune.out$best.model, datos_ejemplo)
```

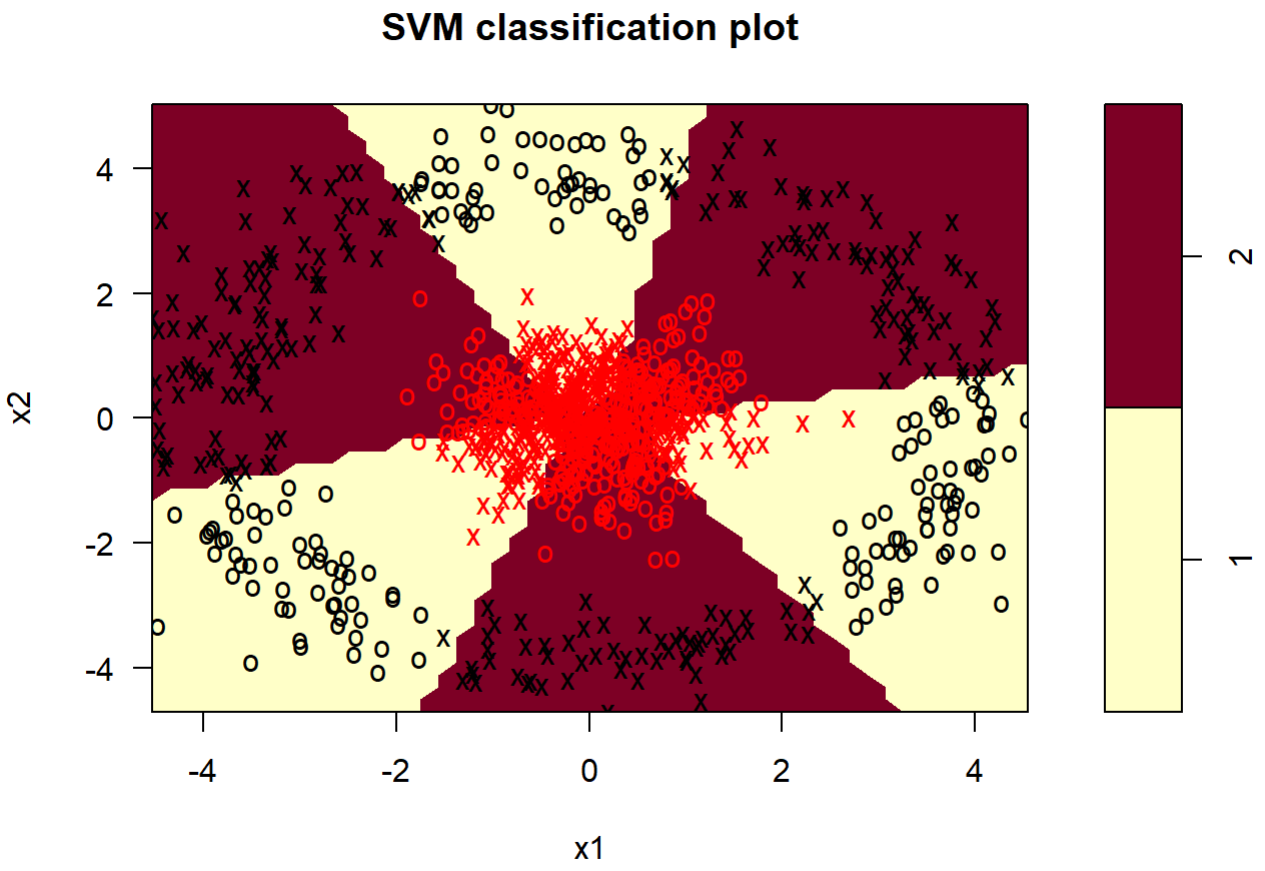


Base radial

```
tune.out = tune(svm,Y ~ x2 + x1 , data = datos_ejemplo, kernel = "radial" , scale = FALSE,
                ranges = list(cost = c(0.1 ,1 ,10 ,100 ,1000),
                              gamma = c(0.5 ,1 ,2 ,3 ,4)))

plot(tune.out$best.model, datos_ejemplo)
```




```
tune.out = tune(svm,Y ~ x2 + x1 , data = datos_ejemplo, kernel = "sigmoid" , scale = FALSE,
                ranges = list(cost = c(0.1 ,1 ,10 ,100),
                              gamma = c(0.5 ,1 ,2 ,3)))

plot(tune.out$best.model, datos_ejemplo)
```

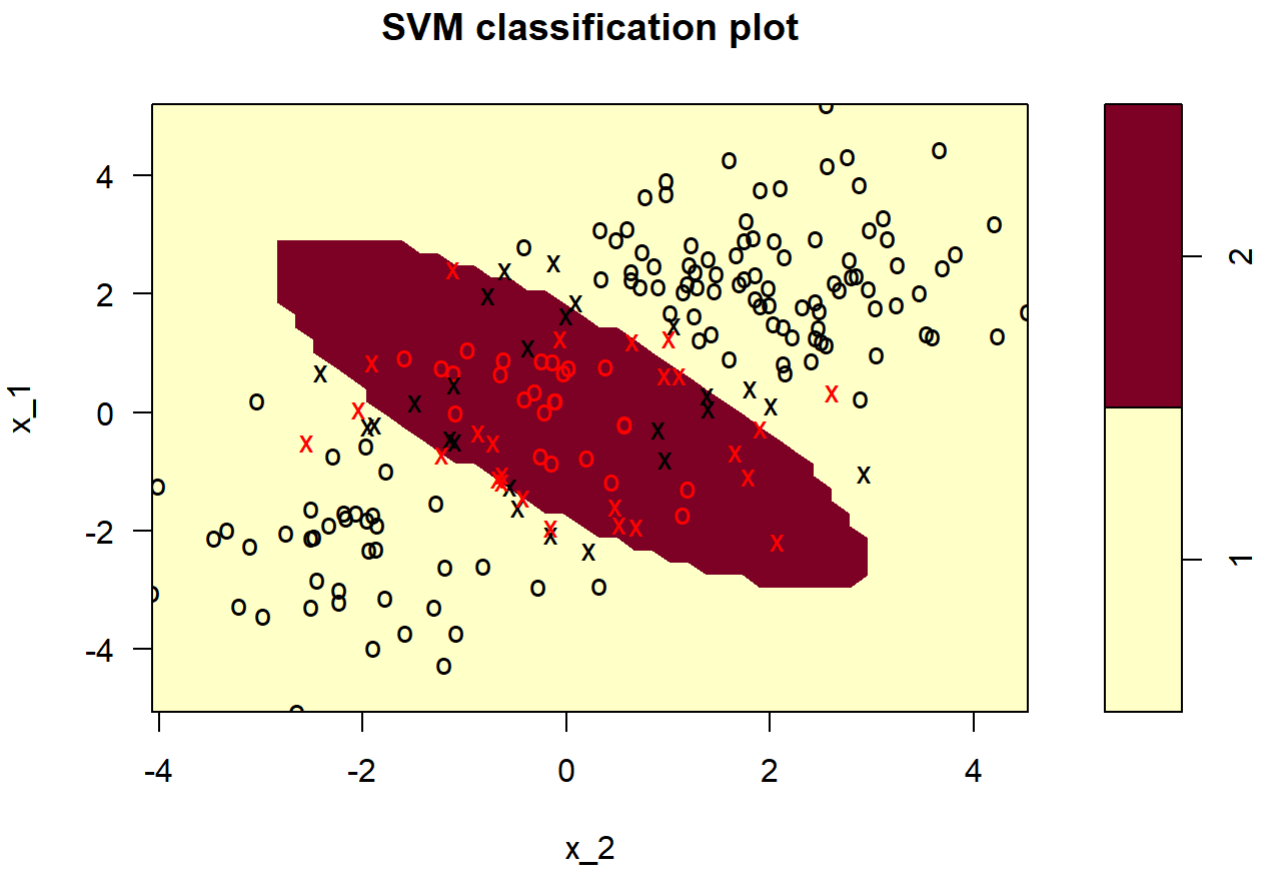


Ejemplo de la clase pasada

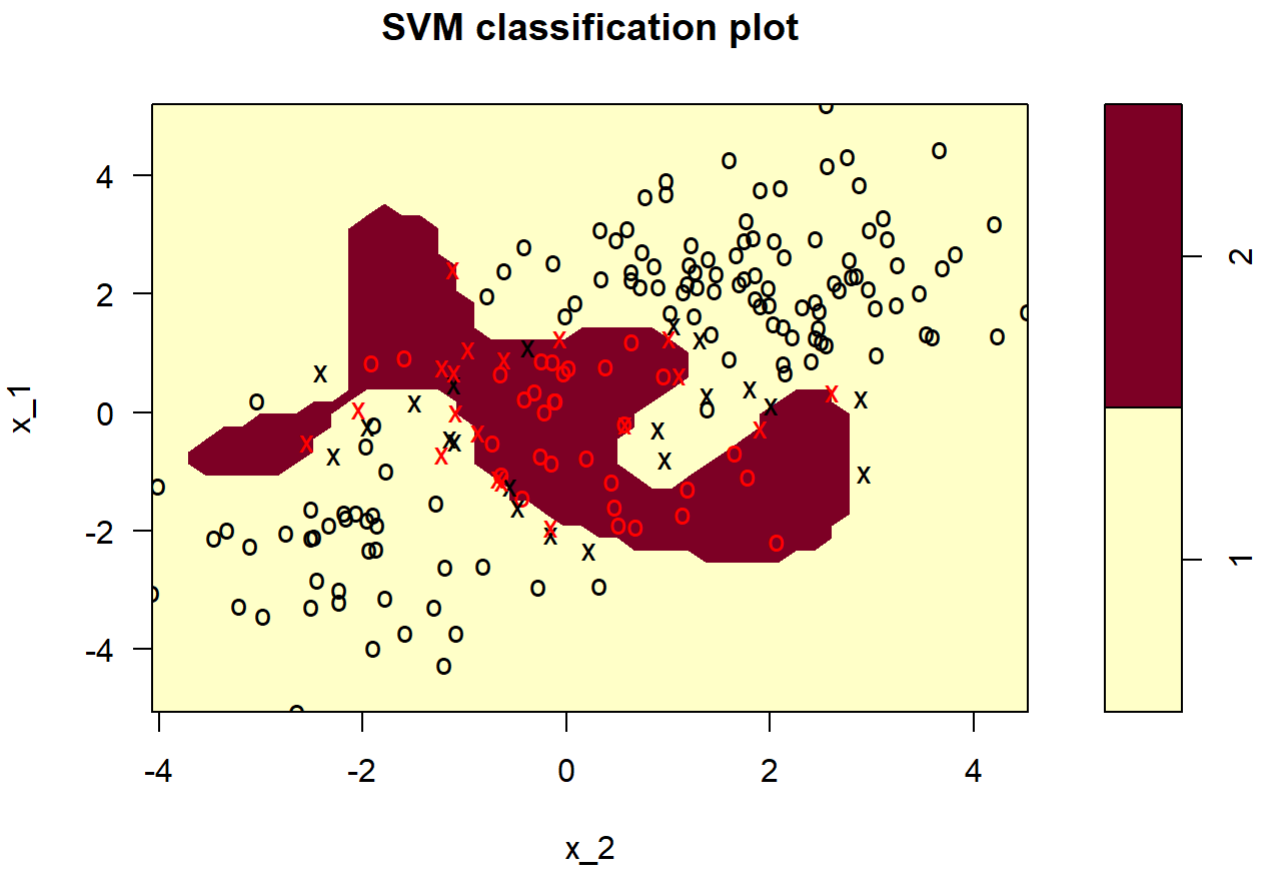
```
set.seed(2020)

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))


ggplot(data = dat, aes(x.1,x.2, color = y)) + geom_point()
```

Polinomial

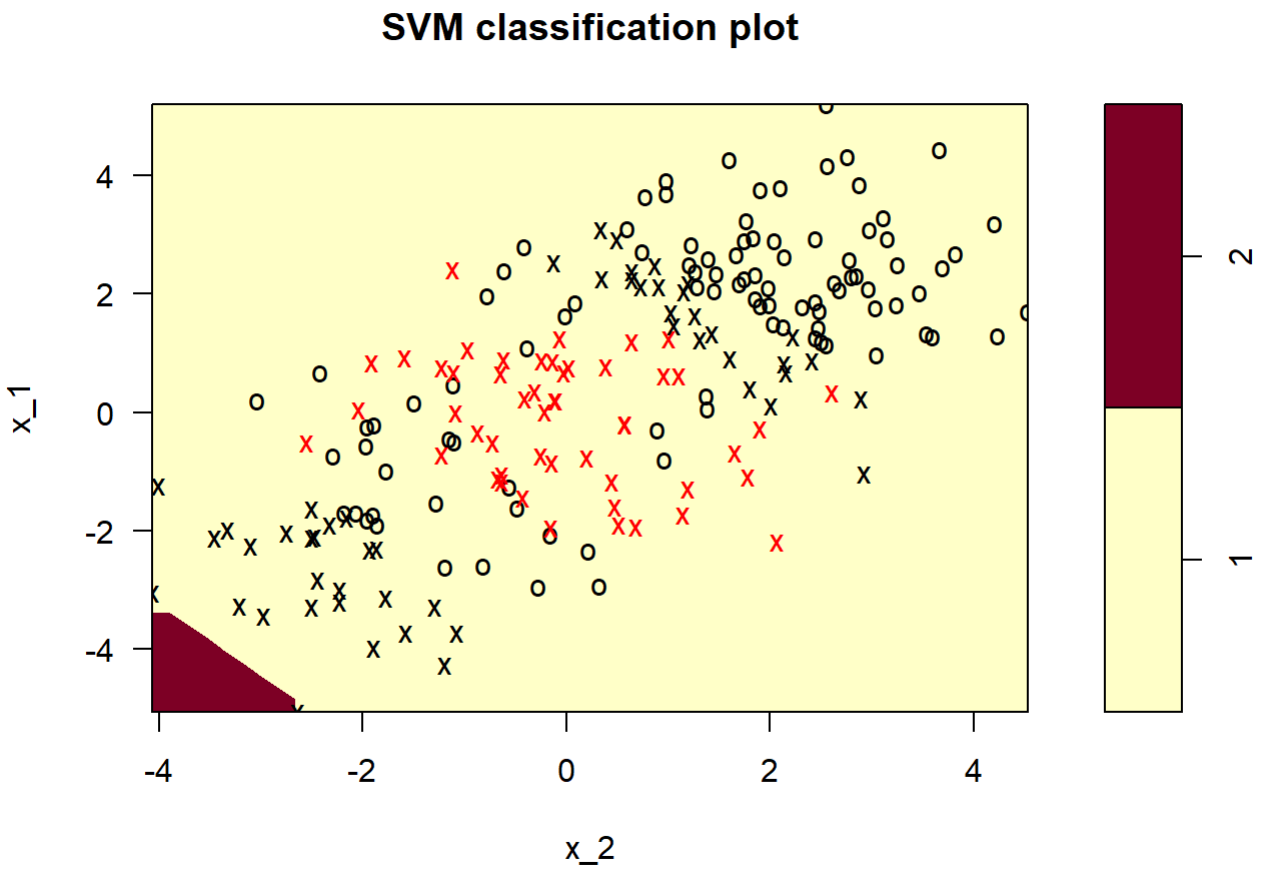
```
names(dat) = c("x_1","x_2","y")
svm.fit = svm(y ~ x_1 + x_2 , data = dat, kernel = "polynomial" ,cost = 10 , scale = FALSE,
              degree = 2)

plot(svm.fit, dat)
```




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

plot(tune.out$best.model,dat)
```




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

plot(tune.out$best.model,dat)
```


SVM classification plot
x_1
x_2
4
2
0
-2
-4
-4
-2
0
2
4
2
1