SVM - parte 2

Fernando Anorve

4/16/2020

Ayer vimos

1. Hiperplano de margen maximal (caso separable)
2. Clasificador por soporte vectorial (caso no separable)

Habíamos visto comentado un poco acerca de los márgenes. En el problema de optimización

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N}\xi_i$$

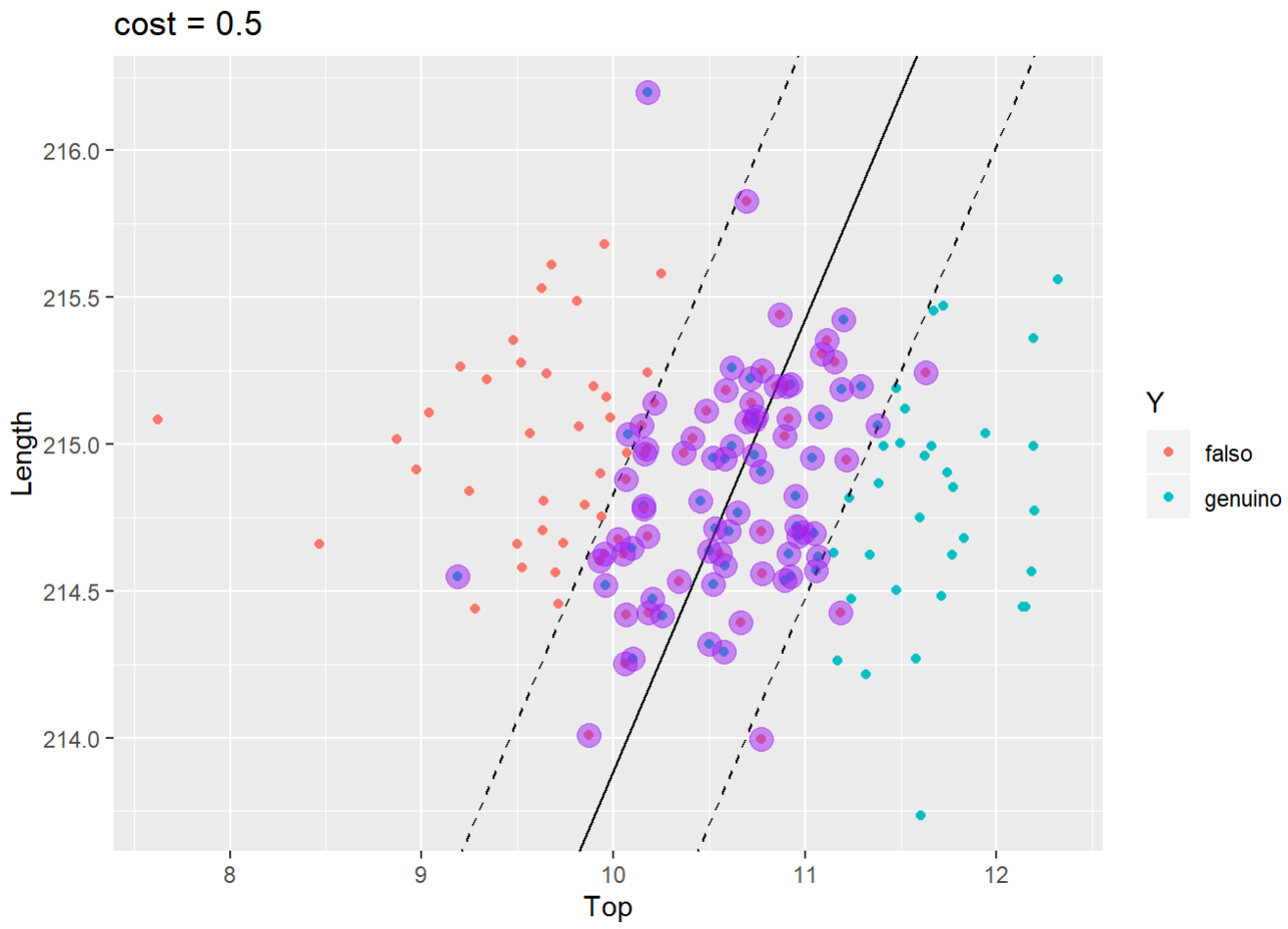
sujeto a

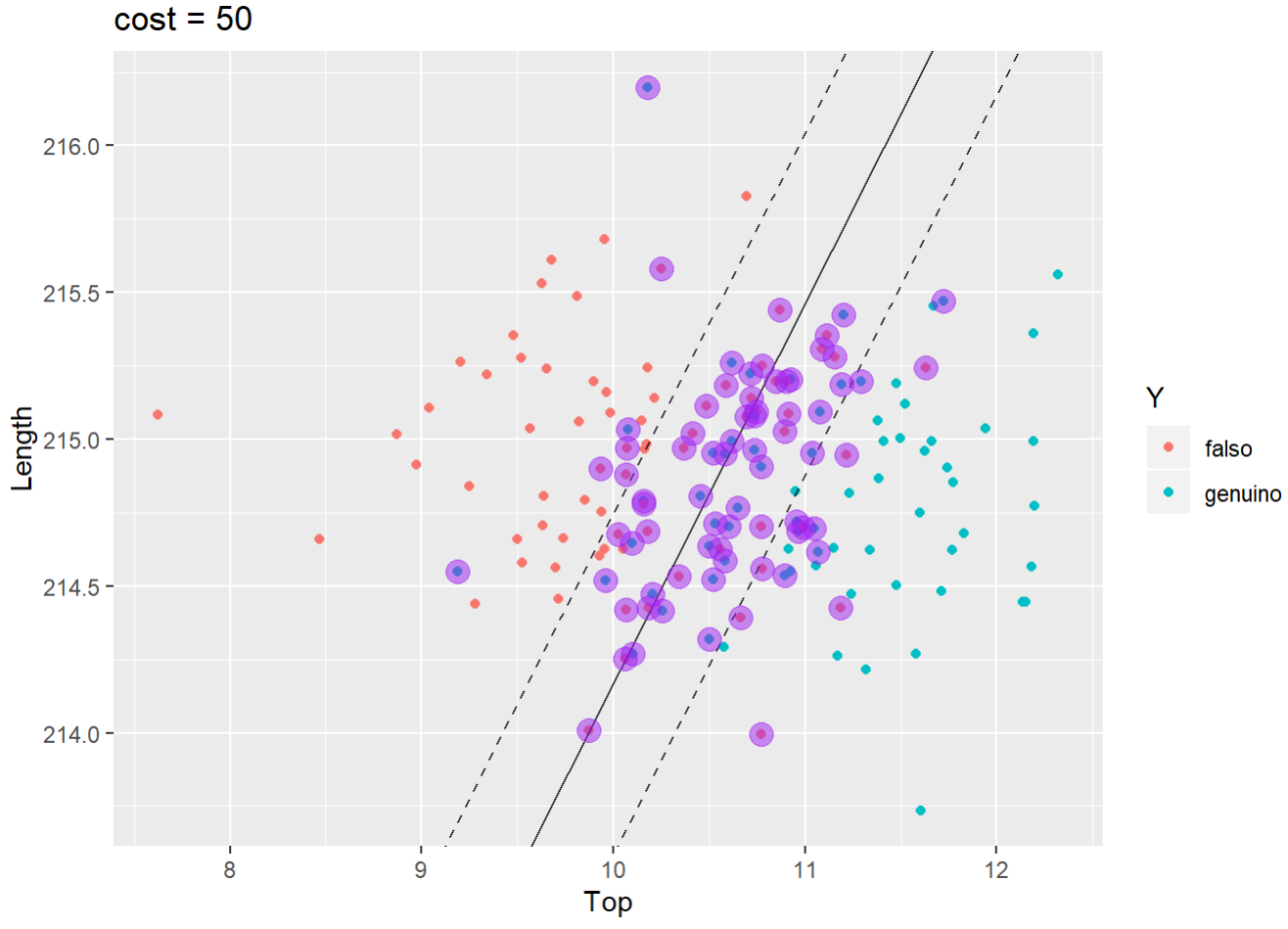
$$\xi_i \geq 0, y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i \forall i$$

Un mayor costo C, penaliza más las observaciones que infringen los márgenes, por lo que éstos tienden a ser más pequeños. Y un menor costo C, tiende a hacer los márgenes más grandes.

Para escoger el valor de C que más nos conviene, se puede usar validación cruzada. La función $tune$ ya tiene implementadas pruebas de validación cruzada de 10 dobleces para comparar distintos valores de C

Ya antes habíamos visto qué pasaba con distintos costos


cost = 0.5
Length
216.0
215.5
215.0
214.5
214.0
8
9
10
11
12
Top
Y
falso
genuino


cost = 50
Length
216.0
215.5
215.0
214.5
214.0
8
9
10
11
12
Top
Y
falso
genuino

Para comparar distintos valores de C utilizamos la función tune()

```
set.seed(1)
tune.out = tune(svm, Y~Top + Length, data = banknote.train , kernel = "linear",
ranges = list ( cost = c(0.001 , 0.01 , 0.1 , 1 ,5 ,10 ,100) ))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.22
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-03 0.5666667 0.07200823
## 2 1e-02 0.2533333 0.15650504
## 3 1e-01 0.2200000 0.18067090
## 4 1e+00 0.2266667 0.18908487
## 5 5e+00 0.2333333 0.18921540
## 6 1e+01 0.2266667 0.18908487
## 7 1e+02 0.2200000 0.18605721
```

Dicha función recomienda usar C = 1, cuyos resultados serían (nótese que ya no llamamos a la función svm)

```
svmfit = tune.out$best.model

w <- t(svmfit$coefs) %*% svmfit$SV

slope_1 <- -w[1]/w[2]

intercept_1 <- svmfit$rho/w[2]

p <- ggplot(data = banknote.train , aes(x = Top, y = Length,color = Y)) + geom_point() +
  geom_abline(slope = slope_1,intercept = intercept_1) +
    geom_abline(slope = slope_1,
                intercept = intercept_1-1/w[2],
                linetype = "dashed") +
    geom_abline(slope = slope_1,
                intercept = intercept_1+1/w[2],
                linetype = "dashed") +
  ggtitle("cost = 0.5")

df_sv <- banknote.train[svmfit$index,]

p1 <- p + geom_point(data = df_sv,
                     aes(x = Top, y = Length),
                     color = "purple",
                     size = 4, alpha = 0.5)


svm.pred = predict(svmfit,banknote.test,type = "class")

conf_table = table(svm.pred,banknote.test$Y)

conf_table
```

```
##
## svm.pred  falso genuino
##   falso      18       3
##   genuino     6      23
```

```
(conf_table[1,1] + conf_table[2,2])/sum(conf_table)
```

```
## [1] 0.82
```

Mismo ejemplo, más variables

La función svm no sólo funciona con un par de variables. De hecho podemos usar todas las demás variables para predecir Y. La desventaja es que ya no es tan trivial mostrarlo en 2D. Aun así, podemos predecir el conjunto de prueba con bastante exactitud

```
svmfit = svm(Y ~ . , data = banknote.train, kernel = "linear", cost = 100 , scale = FALSE)

svm.pred = predict(svmfit,banknote.test,type = "class")

conf_table = table(svm.pred,banknote.test$Y)

conf_table
```

```
##
## svm.pred  falso genuino
##   falso      24       0
##   genuino     0      26
```

```
(conf_table[1,1] + conf_table[2,2])/sum(conf_table)
```

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

Máquinas de soporte vectorial

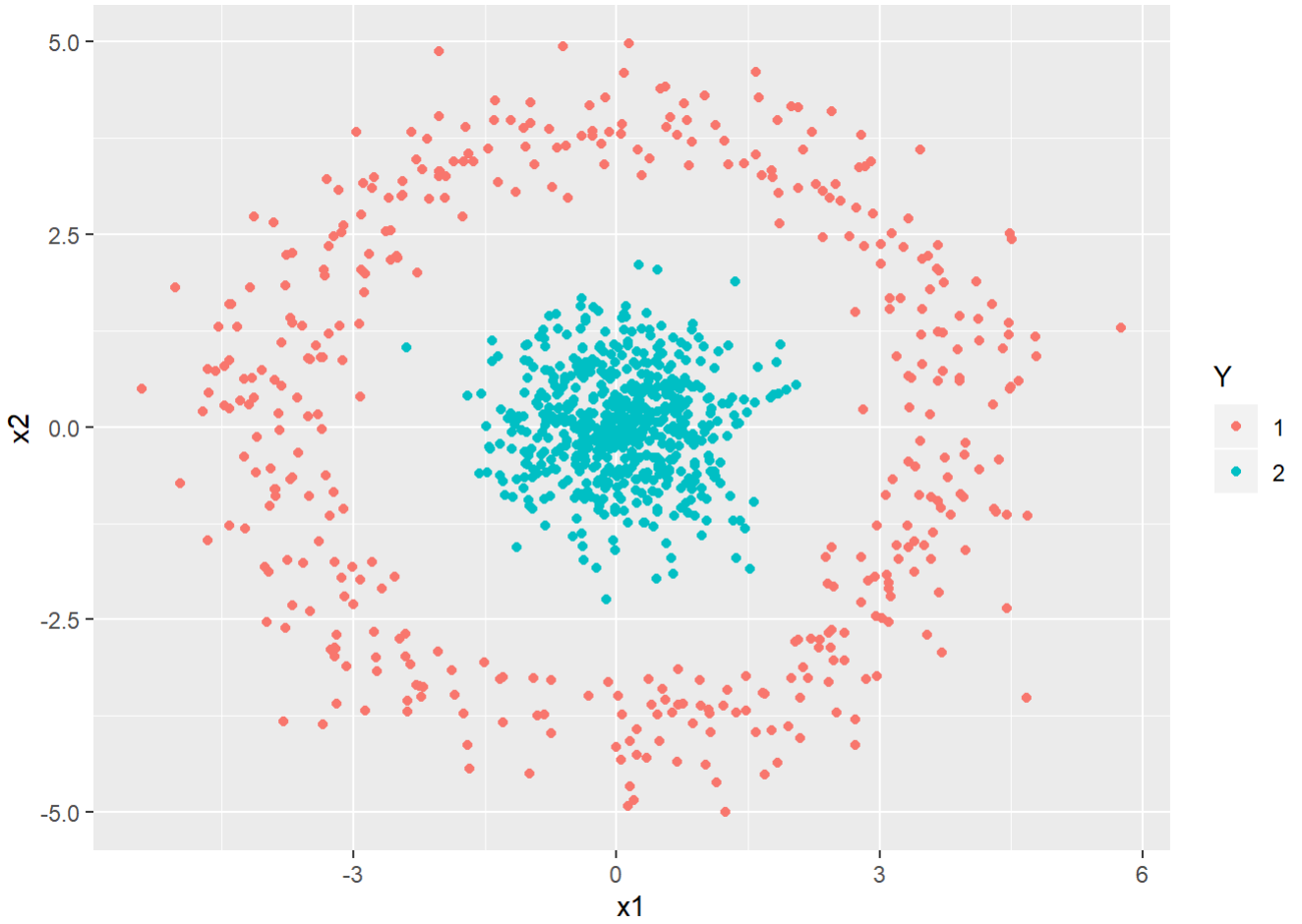
Hemos estudiado cómo separar el espacio utilizando un hiperplano para clasificar observaciones

¿Esto siempre es posible?

Veamos un ejemplo

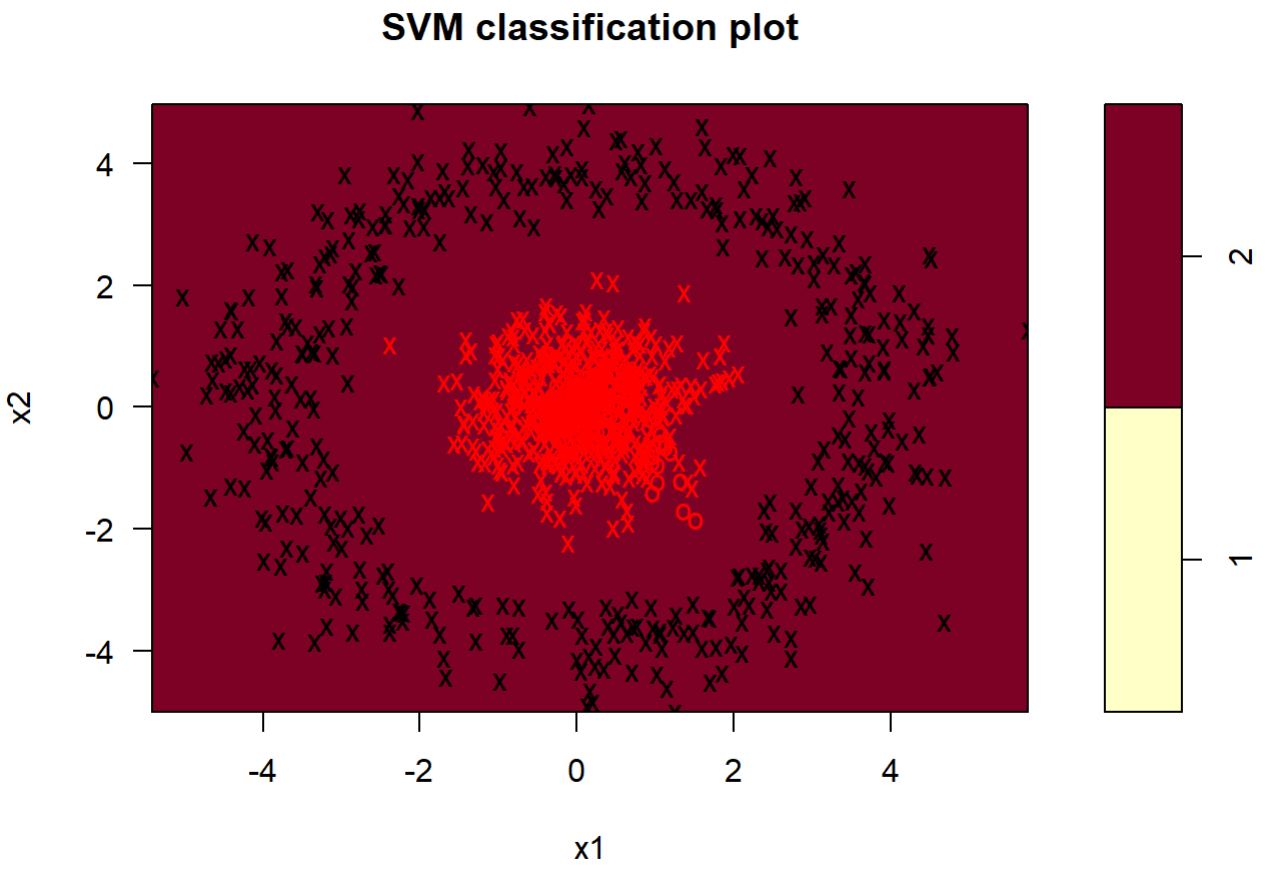
```
x1 = rep(0,1000)
x2 = rep(0,1000)
y = rep(0,1000)

lambd = runif(1000);

y[which(lambd < 0.4)] = 1;
y[which(lambd >= 0.4)] = 2;

for (i in 1:1000) {
    if(y[i] == 1) {
        r = rgamma(1,shape = 20, scale = 0.1)+2
        u = runif(1, max = 2*pi)
        x1[i] = r*cos(u)
        x2[i] = r*sin(u)
    }
    else {
        r = sqrt(2*rexp(1,rate = 1)) * 0.7
        u = runif(1, max = 2*pi)
        x1[i] = r*cos(u)
        x2[i] = r*sin(u)
    }
}

datos_ejemplo = data.frame(x1,x2, "Y" = as.factor(y) )
```

```
ggplot(data = datos_ejemplo, aes(x1,x2, color = Y)) + geom_point()
```

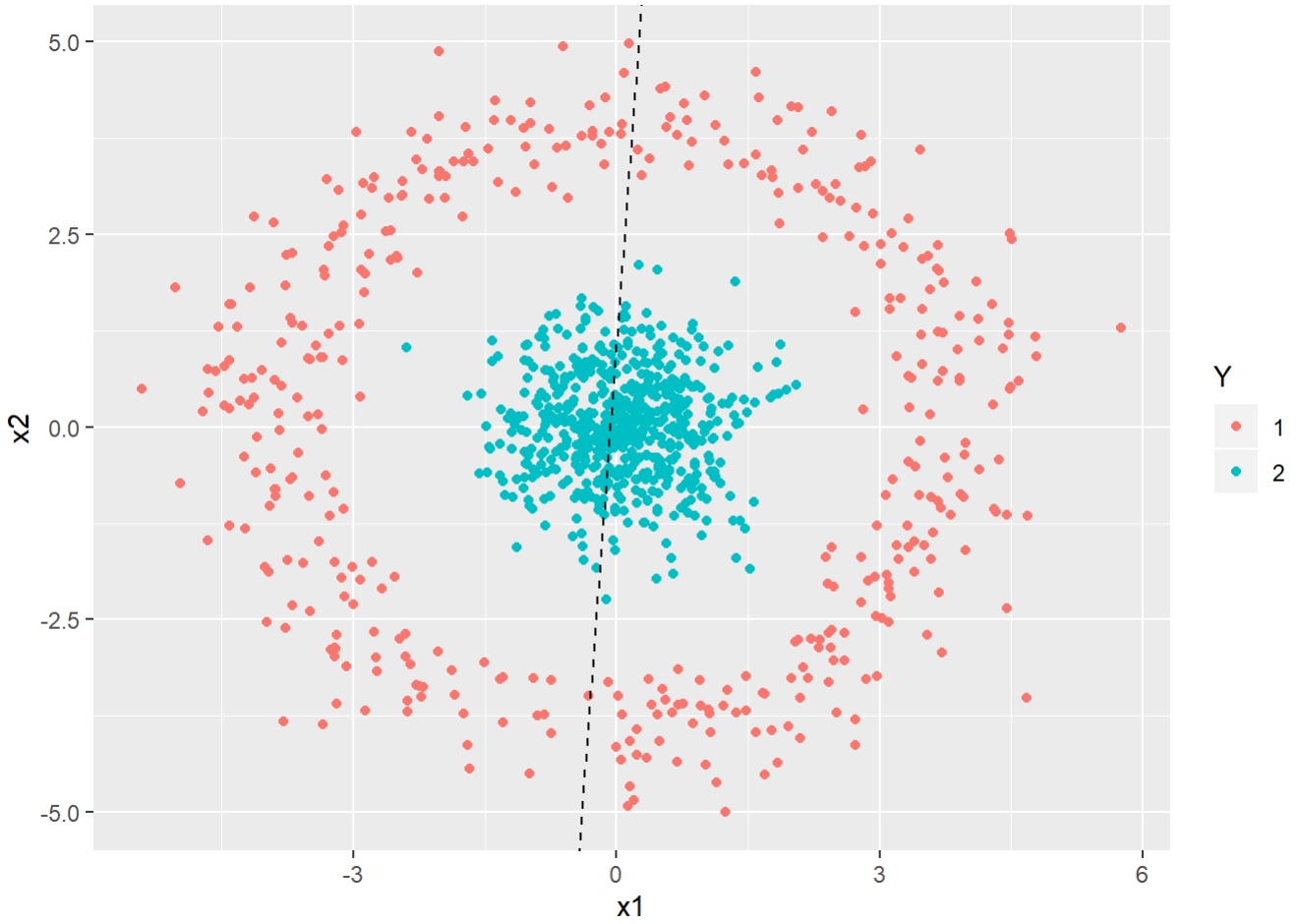


No hace falta mucha técnica para adivinar qué ocurrirá en este caso:

```
svmfit = svm(Y ~ x2 + x1 , data = datos_ejemplo, kernel = "linear", cost = 1 , scale = FALSE)

plot(svmfit, datos_ejemplo)
```




```
w <- t(svmfit$coefs) %*% svmfit$SV

slope_1 <- -w[1]/w[2]

intercept_1 <- svmfit$rho/w[2]

p <- ggplot(data = datos_ejemplo , aes(x = x1, y = x2,color = Y)) + geom_point() +
  geom_abline(slope = slope_1,intercept = intercept_1) +
    geom_abline(slope = slope_1,
                intercept = intercept_1-1/w[2],
                linetype = "dashed") +
    geom_abline(slope = slope_1,
                intercept = intercept_1+1/w[2],
                linetype = "dashed")
p
```

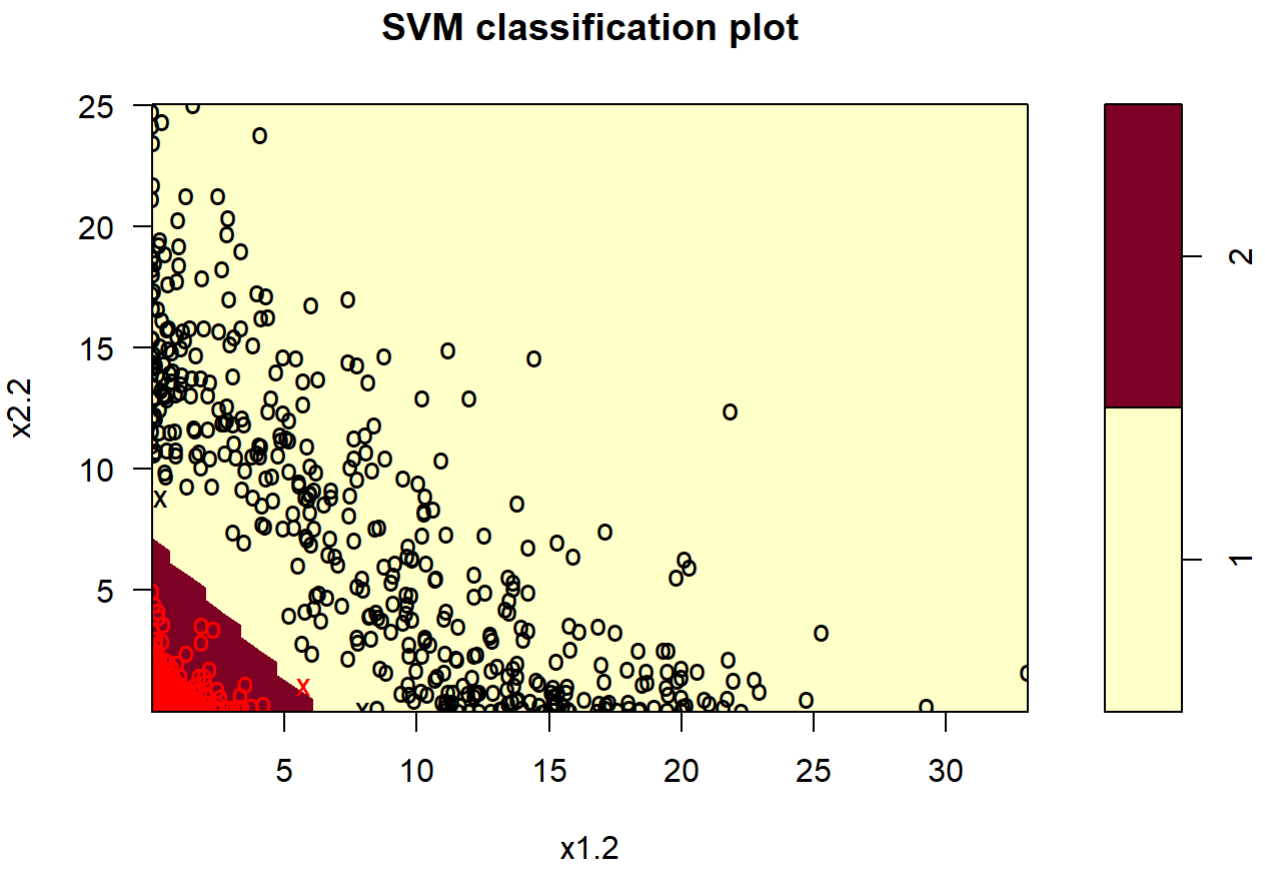


En resumen: hay fallas. No importa cómo dibujemos la gráfica en 2-D.

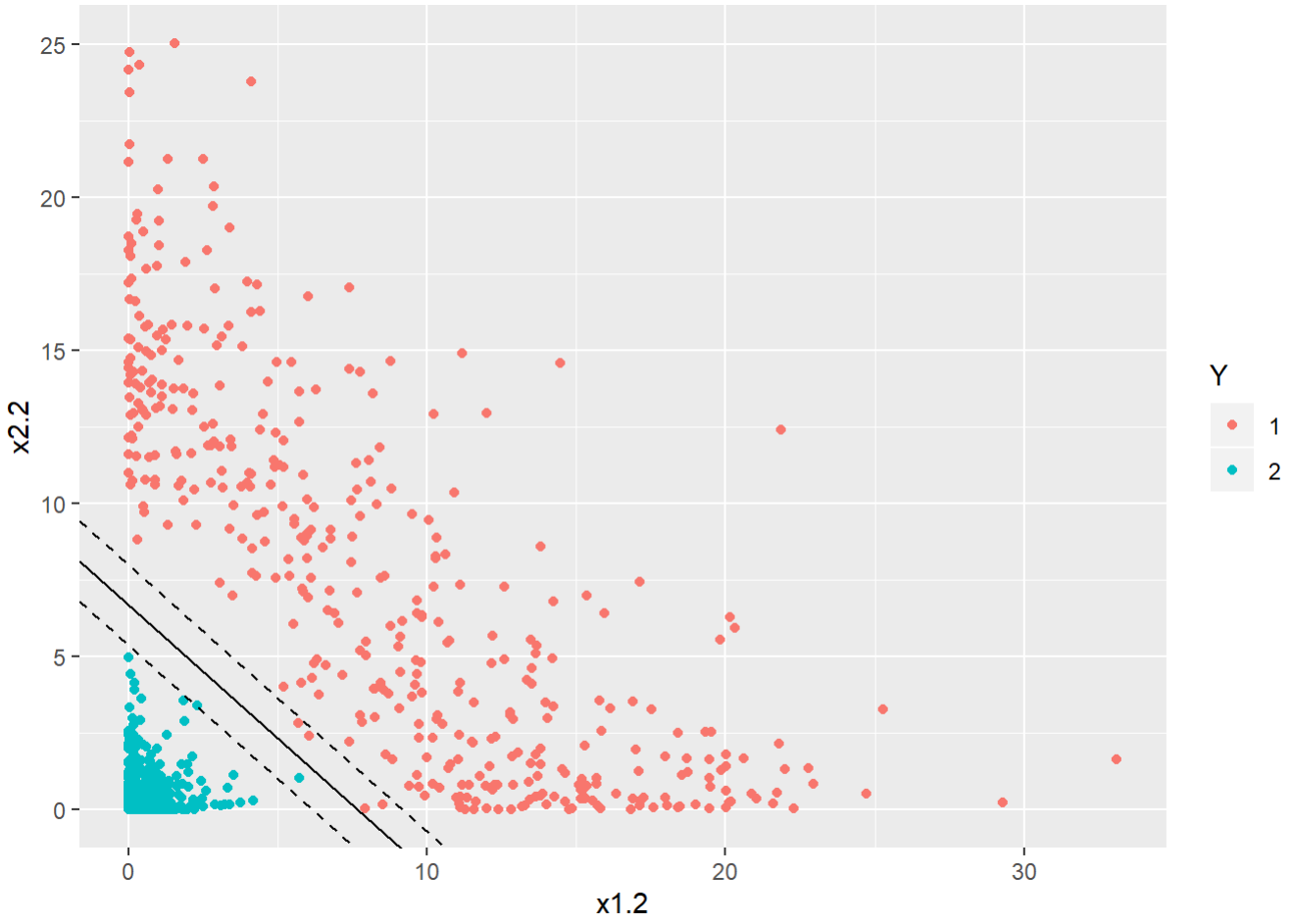
La pregunta ahora es, ¿qué procede?

Lógicamente, calcular hiperplanos no nos va ayudar mucho en esta ocasión. Lo que podría ayudarnos es agregar variables de mayor orden a la función $f(x)$. En este caso, usar x_1^2 y x_2^2 podría ayudar.

```
datos_ejemplo2 = datos_ejemplo
datos_ejemplo2$x1 = datos_ejemplo$x1^2
datos_ejemplo2$x2 = datos_ejemplo$x2^2
names(datos_ejemplo2)[1:2] = c("x1.2","x2.2")

svmfit = svm(Y ~ x2.2 + x1.2 , data = datos_ejemplo2, kernel = "linear" , scale = FALSE)

plot(svmfit, datos_ejemplo2)
```




```
w <- t(svmfit$coefs) %*% svmfit$SV

slope_1 <- -w[1]/w[2]

intercept_1 <- svmfit$rho/w[2]

p <- ggplot(data = datos_ejemplo2 , aes(x = x1.2, y = x2.2,color = Y)) + geom_point() +
  geom_abline(slope = slope_1,intercept = intercept_1) +
    geom_abline(slope = slope_1,
                intercept = intercept_1-1/w[2],
                linetype = "dashed") +
    geom_abline(slope = slope_1,
                intercept = intercept_1+1/w[2],
                linetype = "dashed")
p
```



¿Qué cosas cambian desde esta nueva perspectiva?

En la imagen original (que aparece de nuevo abajo), los puntos azules(?) y los puntos rojos están divididos por la circunferencia de radio aproximadamente 2.6. Esto quiere decir que un buen criterio de clasificación sería

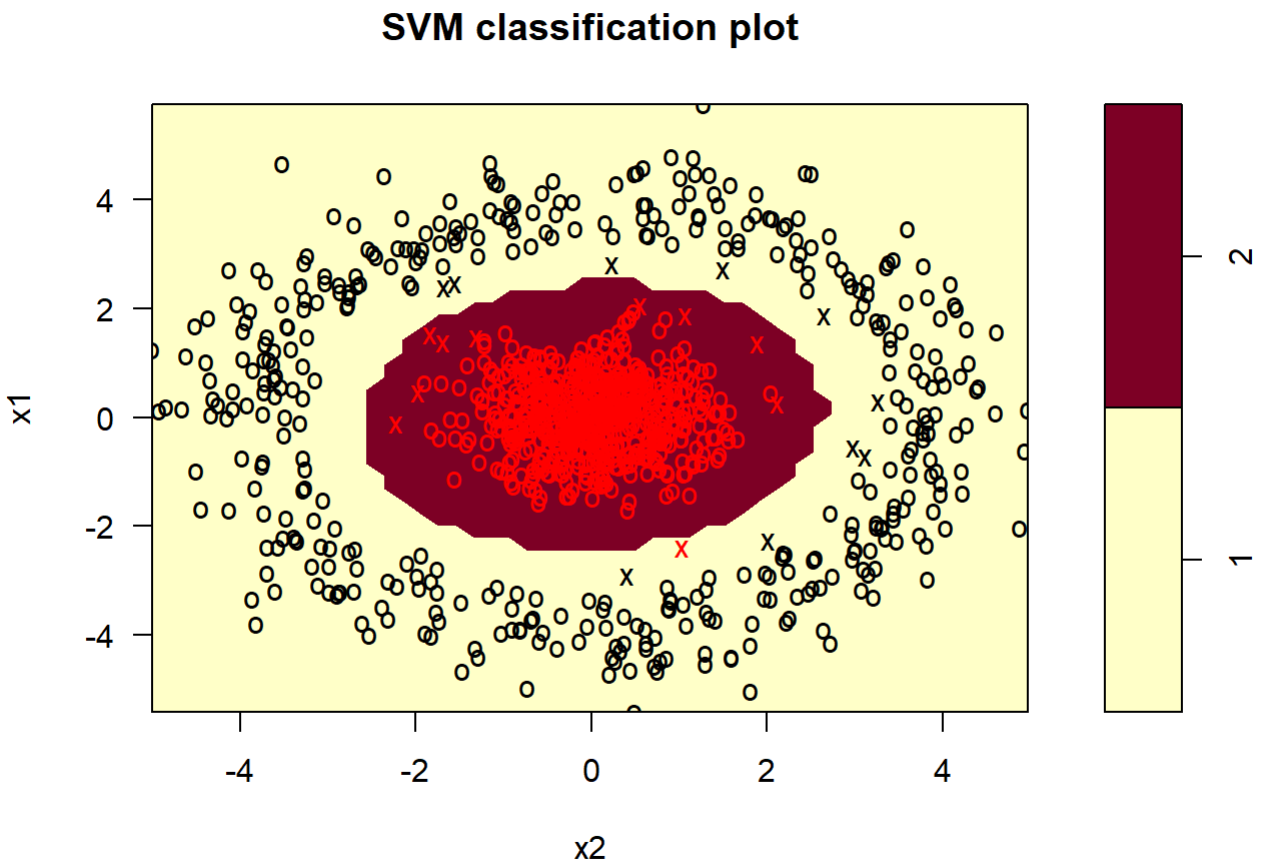
- $x_1^2 + x_2^2 \leq 2.6^2$ se clasifica como azul verdoso
- $x_1^2 + x_2^2 \geq 2.6^2$ se clasifica como rojo

¡Es por ello que un nuevo espacio definido por los cuadrados de estas variables funcionan bien!

¿Cómo generalizar?

- En los primeros ejemplo habíamos utilizado $f(x) = x^T\beta + \beta_0$ como base de la clasificación. La preimagen de cero tiene una forma bastante regular, es un hiperplano.
- En este nuevo ejemplo, cambiamos por $f(x) = \beta_0 + x_1^2\beta_1 + x_2^2\beta_2$
- Cuando f deja de ser una combinación lineal, la preimagen de 0 respecto a f deja de ser forzosamente un hiperplano para volverse en general una curva/superficie de nivel cero
- La curva de nivel cero de $f(x) = x_1^2 + x_2^2 - 2.6^2$ es una circunferencia que podríamos usar para clasificar en el problema anterior
- Notamos que ahora vamos a usar una función polinomial para f en vez de una con términos lineales. Eso se lo podemos indicar directamente a la función svm()

```
svm_model<- svm(Y ~ ., data = datos_ejemplo, type = "C-classification",
      kernel = "polynomial", degree = 2)

plot(svm_model, datos_ejemplo)
```

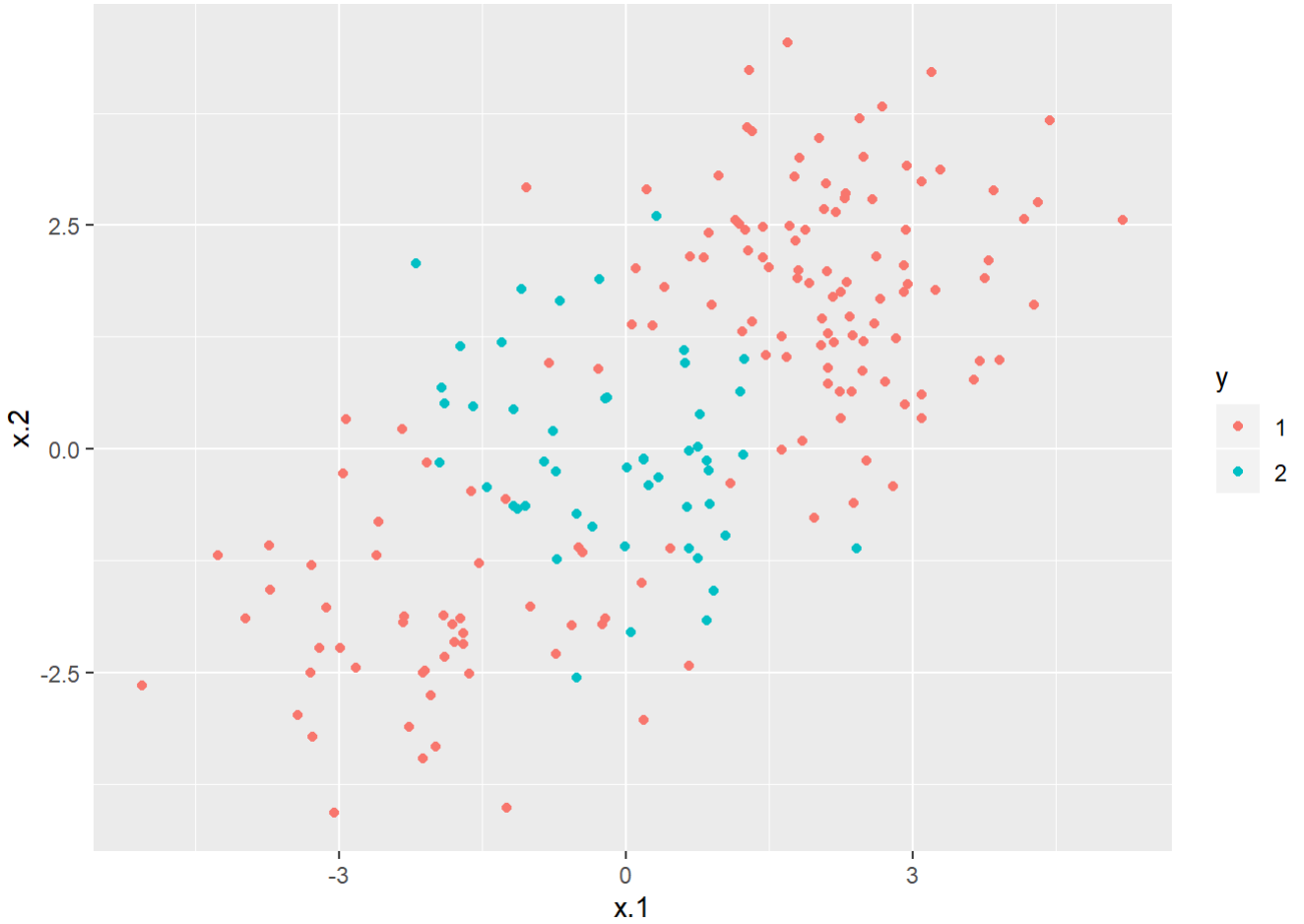


- Para cambiar la naturaleza de f, tuvimos que usar el parámetro kernel de lineal a polinomial. Además, indicamos que íbamos a usar un polinomio de grado 2.
- En general, podemos ver a f de la forma $f(x) = \beta_0 + \sum \alpha_i K(x, x_i)$ (varía un poco según el autor)
- $K(x, x_i)$ es el kernel de nuestro modelo

Otro ejemplo

Otro ejemplo donde no se puede pensar en hiperplannos sería:

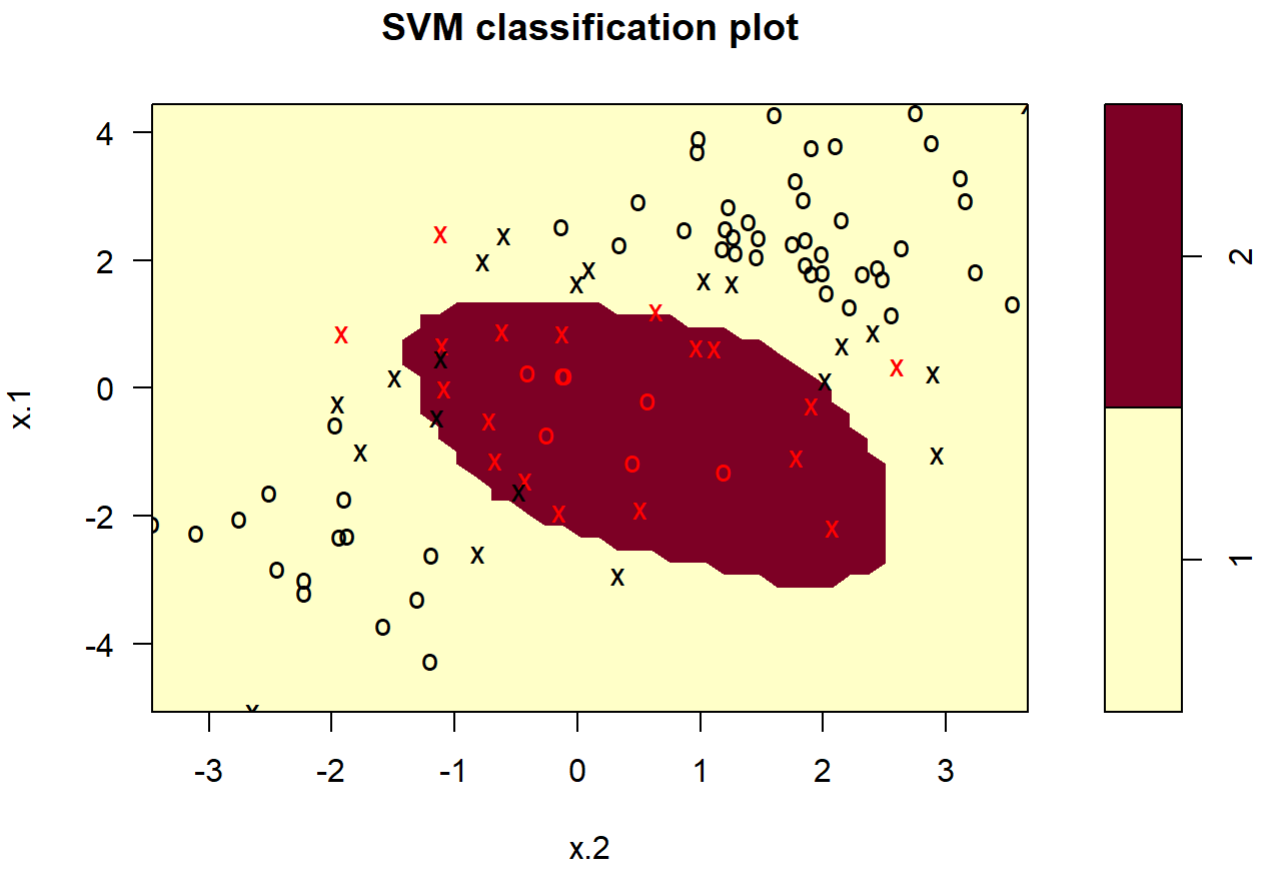
```
set.seed(2020)

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()
```



Ecogemos conjunto de entrenamiento y además utilizamos un kernel radial, donde el parámertro $\gamma = 1$.

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

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



Sin embargo, también podemos explorar los distintos resultados de afinar los parámetros de afinamiento. Notamos que no es necesario hacer uno a la vez

```
set.seed (1)
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     1
##
## - best performance: 0.08
##
## - Detailed performance results:
##     cost gamma error dispersion
## 1  1e-01   0.5  0.25 0.17795130
## 2  1e+00   0.5  0.09 0.07378648
## 3  1e+01   0.5  0.10 0.06666667
## 4  1e+02   0.5  0.12 0.12292726
## 5  1e+03   0.5  0.11 0.09944289
## 6  1e-01   1.0  0.25 0.17795130
## 7  1e+00   1.0  0.08 0.07888106
## 8  1e+01   1.0  0.11 0.05676462
## 9  1e+02   1.0  0.13 0.08232726
## 10 1e+03   1.0  0.12 0.07888106
## 11 1e-01   2.0  0.25 0.17795130
## 12 1e+00   2.0  0.10 0.08164966
## 13 1e+01   2.0  0.12 0.06324555
## 14 1e+02   2.0  0.13 0.06749486
## 15 1e+03   2.0  0.13 0.06749486
## 16 1e-01   3.0  0.25 0.17795130
## 17 1e+00   3.0  0.12 0.07888106
## 18 1e+01   3.0  0.13 0.06749486
## 19 1e+02   3.0  0.14 0.06992059
## 20 1e+03   3.0  0.12 0.07888106
## 21 1e-01   4.0  0.25 0.17795130
## 22 1e+00   4.0  0.11 0.08755950
## 23 1e+01   4.0  0.14 0.06992059
## 24 1e+02   4.0  0.14 0.06992059
## 25 1e+03   4.0  0.14 0.06992059
```

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

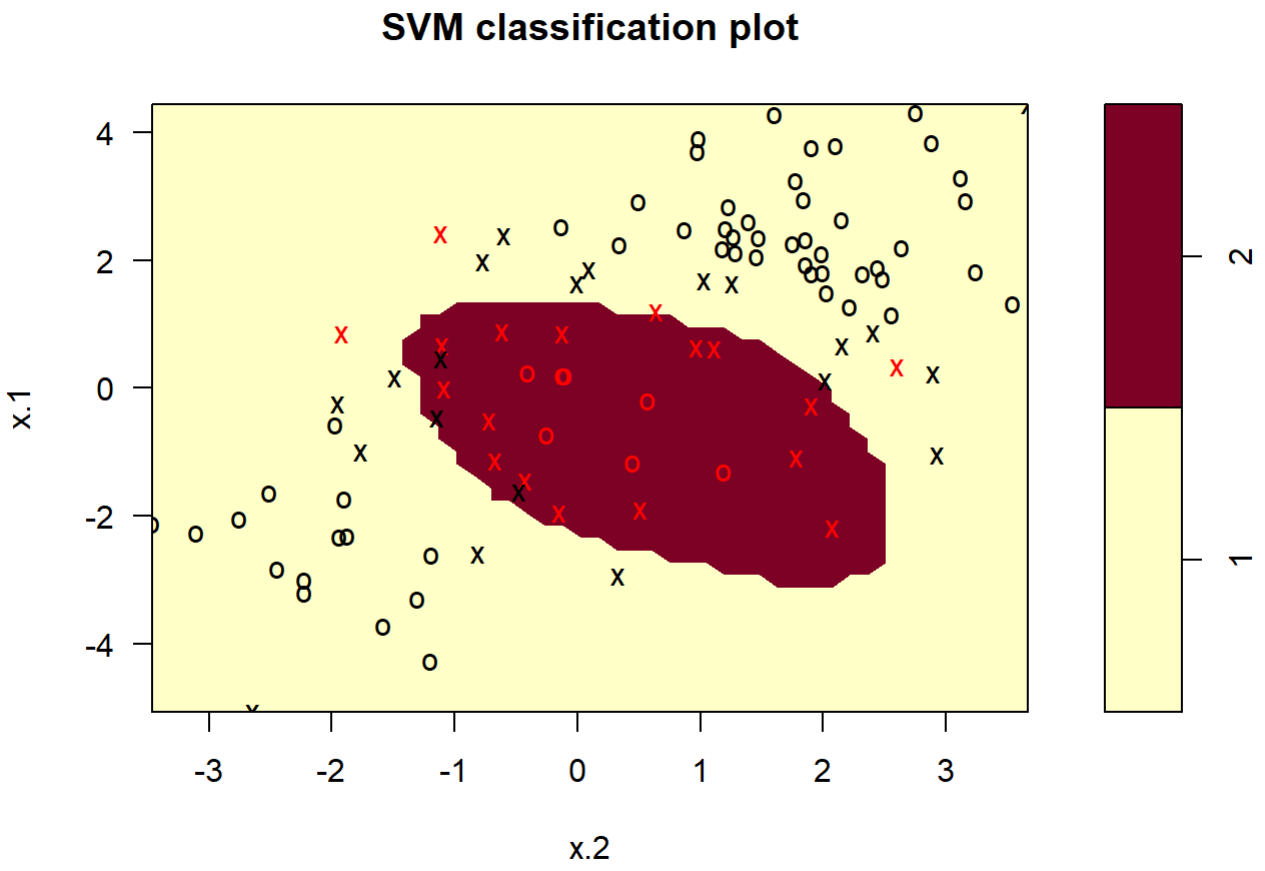


Tabla de confusión del mejor modelo

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

pred.best.mod
```

```
##      pred
## true  1  2
##    1 55 20
##    2 19  6
```

```
(pred.best.mod[1,1] + pred.best.mod[2,2])/sum(pred.best.mod)
```

```
## [1] 0.61
```