Anexo: Ejemplos de implementación en R

Pablo Alfaro Goicoechea, Carlos Morales Aguilera, Carlos S. Sánchez Muñoz

16/12/2020

Primer ejemplo: Comparativa modelos en Iris

El primer ejemplo consiste en evaluar como se comporta **SVM** en un ejemplo sencillo como es el conjunto de datos *iris*, para ello se comparará con diversos modelos como *árboles de decisión*, *random forest* o *Knn*. Es un ejemplo interesante ya que sin demasiado esfuerzo ni demasiado ajuste se puede comprobar la eficacia de esta herramienta en un problema de clasificación sencillo con tres clases.

Lo primero es hacer un análisis exploratorio de los datos:

```
data(iris)
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

A continuación se lleva a cabo una normalización de los datos, excepto la variable clasificatoria:

```
scaled = as.data.frame(scale(iris[,1:4]))
scaled$Species = iris$Species
summary(scaled)
```

```
##   Sepal.Length       Sepal.Width       Petal.Length      Petal.Width
##  Min.   :-1.86378   Min.   :-2.4258   Min.   :-1.5623   Min.   :-1.4422
##  1st Qu.:-0.89767   1st Qu.:-0.5904   1st Qu.:-1.2225   1st Qu.:-1.1799
##  Median :-0.05233   Median :-0.1315   Median : 0.3354   Median : 0.1321
##  Mean   : 0.00000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000
##  3rd Qu.: 0.67225   3rd Qu.: 0.5567   3rd Qu.: 0.7602   3rd Qu.: 0.7880
##  Max.   : 2.48370   Max.   : 3.0805   Max.   : 1.7799   Max.   : 1.7064
```

```
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

Realizamos una visualización inicial de la distribución de los datos, observando como se relacionan entre sí sus variables.

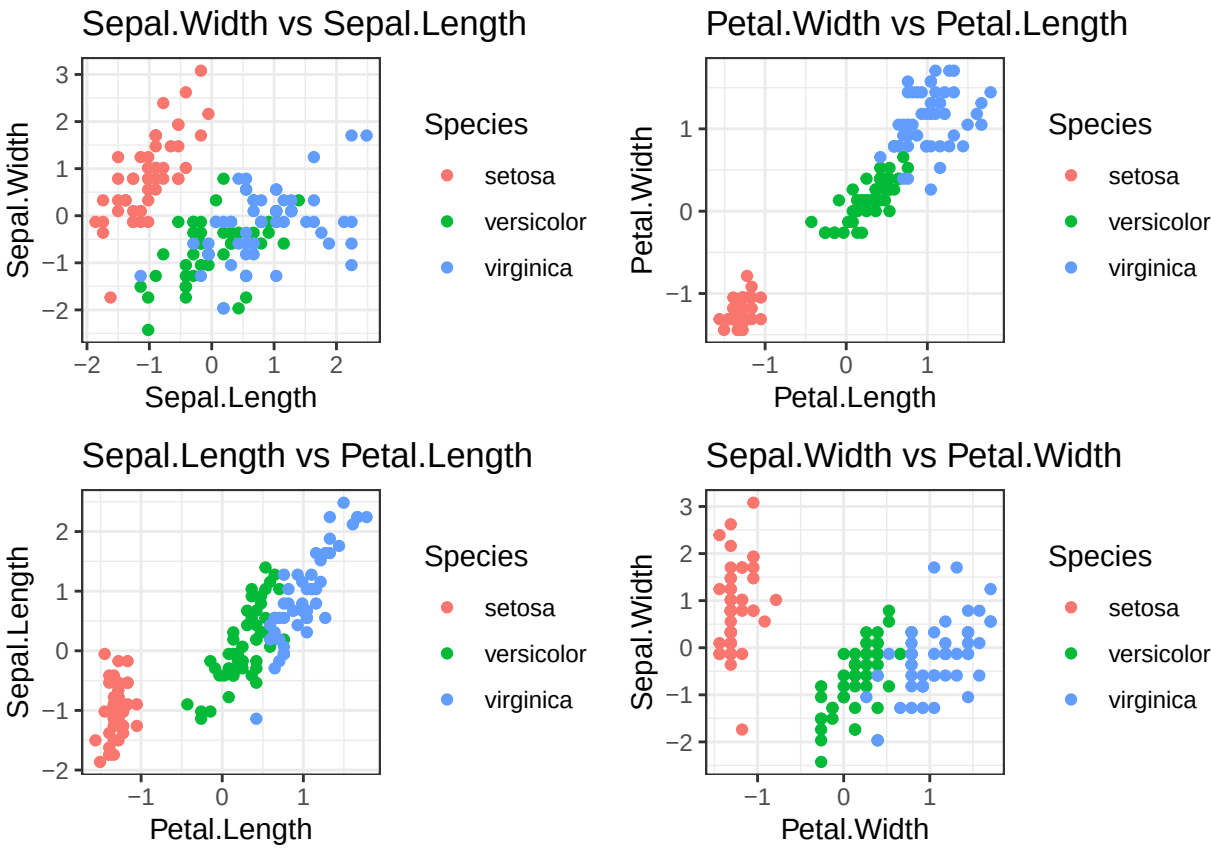
```
# Relationship between Sepal characteristics
plot1 = ggplot(scaled,aes(x =Sepal.Length,y = Sepal.Width,color = Species)) +
  geom_point() + ggtitle("Sepal.Width vs Sepal.Length") +
  theme_bw()

# Relationship between Petal characteristics
plot2 = ggplot(scaled,aes(x =Petal.Length,y = Petal.Width,color = Species)) +
  geom_point() + ggtitle("Petal.Width vs Petal.Length") +
  theme_bw()

# Relationship between Sepal and Petal Length
plot3 = ggplot(scaled,aes(x =Petal.Length,y = Sepal.Length,color = Species)) +
  geom_point() + ggtitle("Sepal.Length vs Petal.Length") +
  theme_bw()

# Relationship between Sepal and Petal width
plot4 = ggplot(scaled,aes(x =Petal.Width,y = Sepal.Width,color = Species)) +
  geom_point()  + ggtitle("Sepal.Width vs Petal.Width") +
  theme_bw()

grid.arrange(plot1,plot2,plot3,plot4,nrow = 2)
```



Creamos conjuntos de *train* y *tests* haciendo un reparto del 75%-25% respectivamente:

```
# Sample size
sample_size =  round(0.75*nrow(scaled))
# Random training index
train_ind = sample(seq_len(nrow(scaled)), size = sample_size)
# Create subsets
train = scaled[train_ind, ]
test = scaled[-train_ind, ]
```

Modelo de árboles de decisión con *Rpart*:

```
# Decision tree with Rpart
model.rpart = rpart(Species ~ . ,data =train)
# Predict
pred.rpart = predict(model.rpart,newdata = test,type = "class")
# Create the confusion matrix
matrix<-confusionMatrix(test$Species, pred.rpart)
# Save accuracy
rpart_accuracy <- matrix$overall["Accuracy"]
# Print table
kable(matrix$table)
```

	setosa	versicolor	virginica
setosa	12	0	0
versicolor	0	17	0
virginica	0	1	8

Modelo de Knn con *Knn*:

```
# Save class
cl = train$Species
# Predict
pred.knn = knn(train[,1:4],test[,1:4],cl,k=3)
# Create the confusion matrix
matrix<-confusionMatrix(test$Species, pred.knn)
# Save accuracy
knn_accuracy <- matrix$overall["Accuracy"]
# Print table
kable(matrix$table)
```

	setosa	versicolor	virginica
setosa	12	0	0
versicolor	0	16	1
virginica	0	1	8

Modelo de Random Forest con *randomForest*:

```
# Random Forest with randomForest
model.rf = randomForest(Species ~ .,data = train)
# Predict
pred.rf = predict(model.rf,newdata = test)
# Create the confusion matrix
matrix<-confusionMatrix(test$Species, pred.rf)
# Save accuracy
rf_accuracy <- matrix$overall["Accuracy"]
# Print table
kable(matrix$table)
```

	setosa	versicolor	virginica
setosa	12	0	0
versicolor	0	16	1
virginica	0	0	9

Modelo de SVM con *SVM*:

```
# SVM with svm
model.svm = svm(Species ~ . ,data = train, method="C-classification", kernel="radial",
          gamma=0.1, cost=10)
# Predict
pred.svm = predict(model.svm,newdata = test)
```

```r
# Create the confusion matrix
matrix<-confusionMatrix(test$Species, pred.svm)
# Save accuracy
svm_accuracy <- matrix$overall["Accuracy"]
# Print table
kable(matrix$table)
```

	setosa	versicolor	virginica
setosa	12	0	0
versicolor	0	16	1
virginica	0	0	9

A continuación, podemos observar los resultados obtenidos con los diferentes modelos:

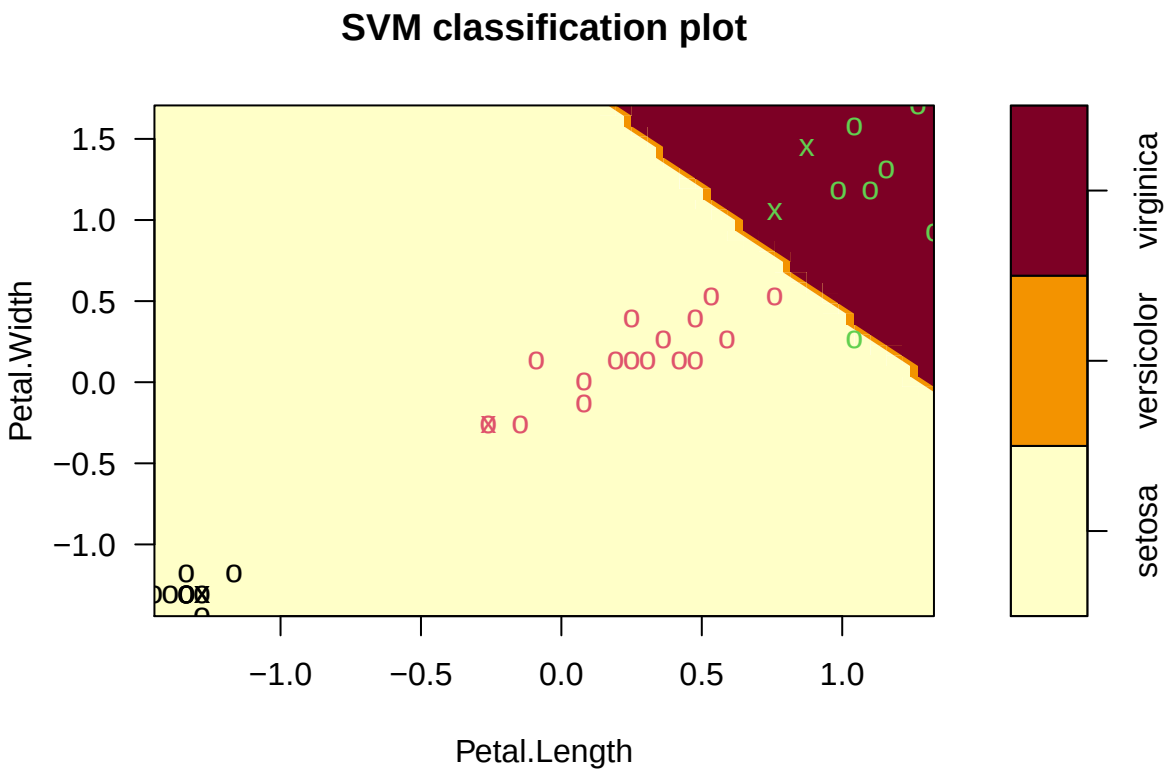
```r
models = data.frame(Technique = c("Decision Tree","kNN","SVM","Random Forest"),Accuracy_Percentage = c(
models
```

```
##       Technique Accuracy_Percentage
## 1 Decision Tree           0.9736842
## 2           kNN           0.9473684
## 3           SVM           0.9736842
## 4 Random Forest           0.9736842
```

Como se puede observar, se obtienen los mismos resultados en este sencillo ejemplo, por lo que se observar que en ejemplos linealmente separables *SVM* funciona al mismo nivel que algunos de los algoritmos más conocidos. La elección de parámetros, kernel, y otros elementos no es una tarea trivial, ya que debe adaptarse al problema en particular para obtener unos buenos resultados.

A continuación se demuestra como se verían los resultados en una gráfica sencilla:

```r
plot(model.svm, test, Petal.Width ~ Petal.Length,
          slice=list(Sepal.Width=3, Sepal.Length=4))
```



Al tratarse de un plano 2D, no se puede apreciar del todo como se realizan las divisiones de forma gráfica, aunque si se puede apreciar que los puntos verdes corresponden a la clase *virginica*, los rojos a la clase *versicolor* y los negros a la *setosa*, y la división es correcta. También se puede observar que el punto verde que se encuentra en la sección correspondiente a *versicolor*, es el único error del conjunto de test.

Segundo ejemplo: Observaciones con función no lineal en 2 dimensiones

Para el siguiente ejemplo, se utiliza el conjunto de datos publicado en el libro *Elements of Satistical Learning*, que contiene observaciones simuladas (2 predictores) con funciones no lineales en un espacio bidimensional.

```
# Obtain data
load(url("https://web.stanford.edu/~hastie/ElemStatLearn/datasets/ESL.mixture.rda"))
```

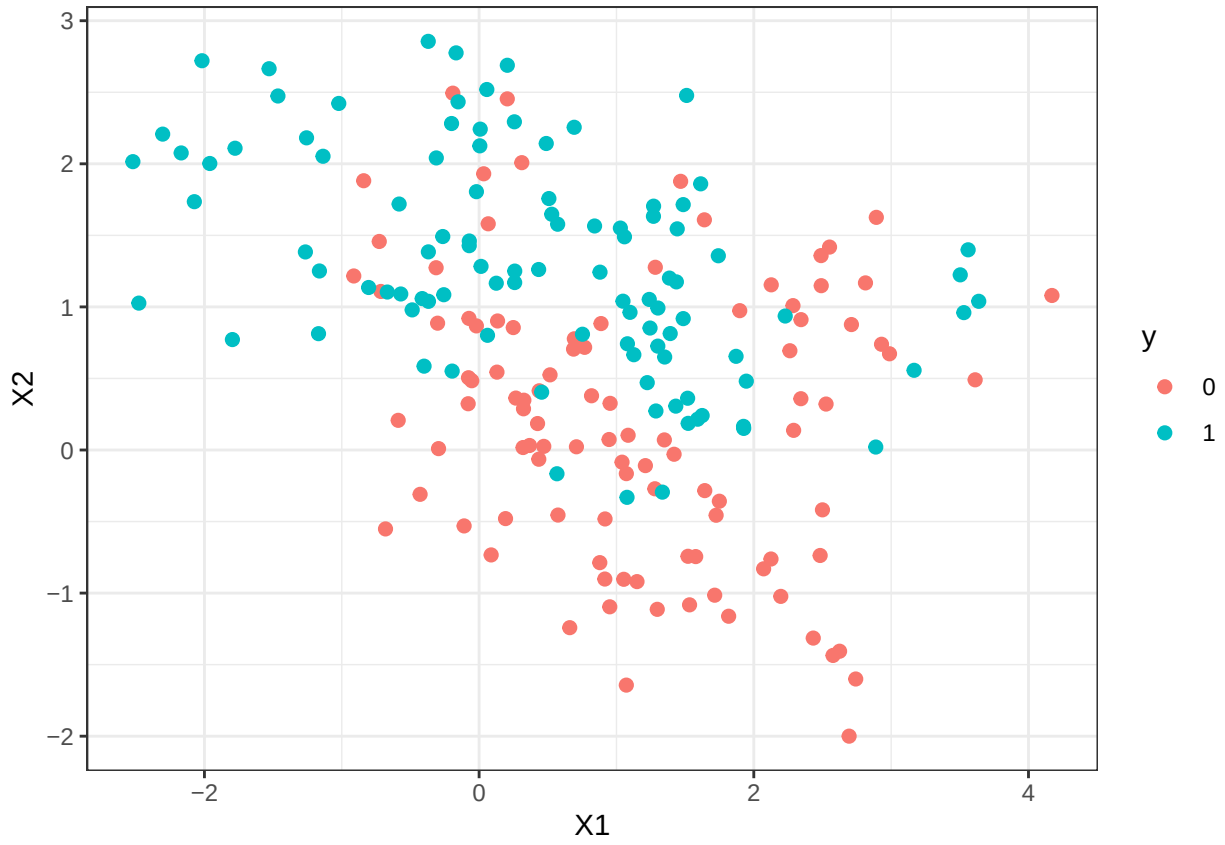
El objeto *ESL.mixture* es una lista que contiene almacenados el valor de los dos predictores en el elemento x y el valor de la clase a la que pertenece cada observación en el elemento y.

```
# Create dataframe
data <- data.frame(ESL.mixture$x, y = ESL.mixture$y)
# Transform classification variable to factor
data$y <- as.factor(data$y)
# Print summary
summary(data)
```

```
##        X1                  X2                y
##  Min.   :-2.52082   Min.   :-1.99985   0:100
##  1st Qu.:-0.07147   1st Qu.: 0.09555   1:100
##  Median : 0.85970   Median : 0.86139
##  Mean   : 0.78467   Mean   : 0.75602
##  3rd Qu.: 1.54344   3rd Qu.: 1.43527
##  Max.   : 4.17075   Max.   : 2.85581
```

A continuación vemos como se distribuyen los datos en un espacio bidimensional:

```
# Plot data
ggplot(data = data, aes(x = X1, y = X2, color = y)) +
  geom_point(size =2) +
  theme_bw()
```



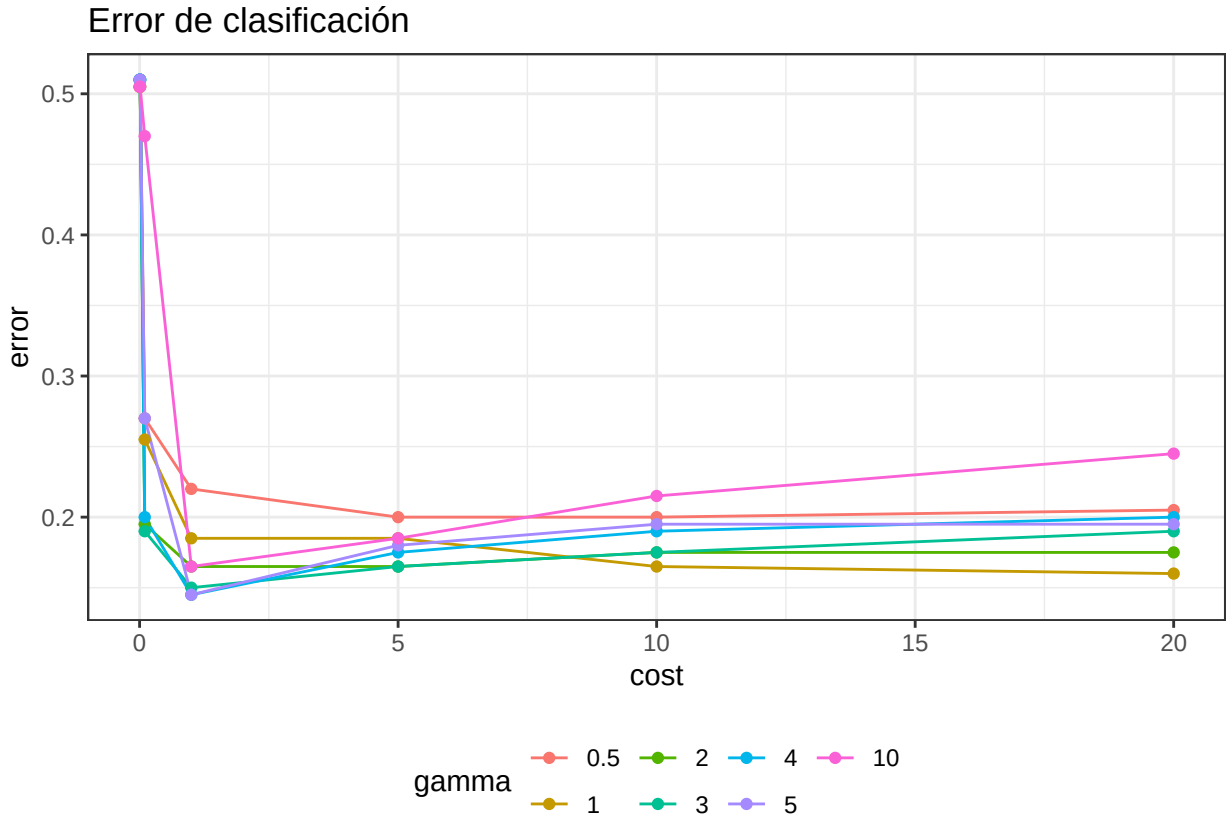
Como no conocemos la mejor configuración posible para el problema, vamos a utilizar la función tune que nos permite probar distintos hiperparámetros, en nuestro caso, con el modelo de *SVM* para encontrar la mejor configuración posible.

El kernel escogido es de tipo *radial*, ya que se trata de un kernel no lineal que nos permite un mayor ajuste que un kernel *polinomial*. En este tipo de kernel está la tarea de determminar el valor γ apropiado y un coste de penalización. Para ello visualizaremos los diferentes errores de clasificación obtenidos y seleccionaremos el mejor modelo obtenido.

```
# Train model with tune, using different hyperparameters
tune_svm <- tune("svm", y ~ X1 + X2, data = data, 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)))

# Plot different performances
ggplot(data = tune_svm$performances, aes(x = cost, y = error, color = as.factor(gamma)))+
  geom_line() +
  geom_point() +
  labs(title = "Error de clasificación", color = "gamma") +
  theme_bw() +
  theme(legend.position = "bottom")
```



A continuación se observan los mejores parámetros y guardamos el mejor modelo de *SVM* construido para el problema:

```
# Print best parameters for problem with SVM
tune_svm$best.parameters
```

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

```
# Save best model
model.svm <- tune_svm$best.model
```

Ya solo quedaría comprobar el resultado obtenido del modelo, para ello se realizará la predicción con el mejor modelo obtenido, se visualizará su matriz de confusión y se visualizará el modelo mediante una gráfica 2D.

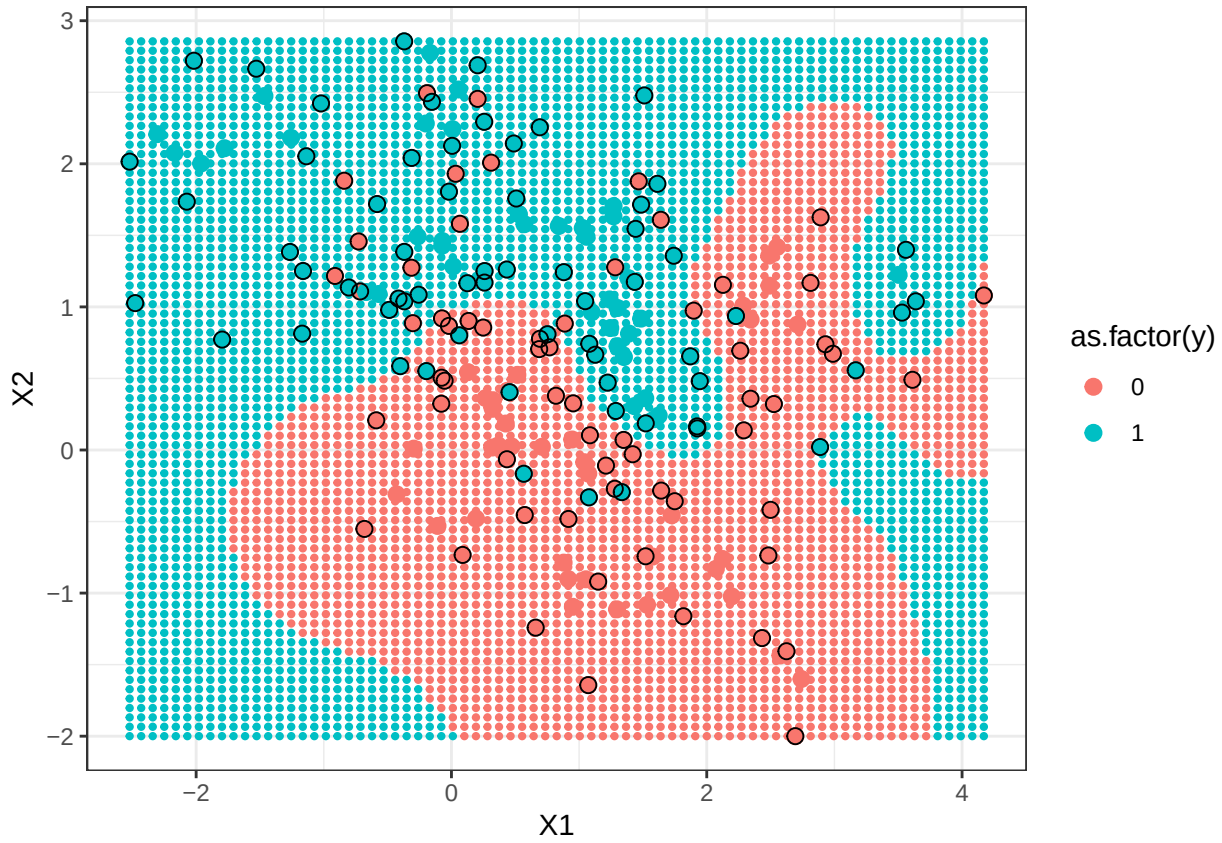
```r
# Obtain ranges
range_X1 <- range(data$X1)
range_X2 <- range(data$X2)

# Interpolate new points for tests
new_x1 <- seq(from = range_X1[1], to = range_X1[2], length = 75)
new_x2 <- seq(from = range_X2[1], to = range_X2[2], length = 75)
new_points <- expand.grid(X1 = new_x1, X2 = new_x2)

# Predict
predictions <- predict(object = model.svm, newdata = new_points)

# Get region color
regions <- data.frame(new_points, y = predictions)

ggplot() +
  # Plot regions
  geom_point(data = regions, aes(x = X1, y = X2, color = as.factor(y)),
             size = 0.8) +
  # Plot points
  geom_point(data = data, aes(x = X1, y = X2, color = as.factor(y)),
             size = 2.5) +
  # Plot points that acts ass vector support
  geom_point(data = data[model.svm$index, ],
             aes(x = X1, y = X2, color = as.factor(y)),
             shape = 21, colour = "black",
             size = 2.5) +
             theme_bw()
```


3
2
1
0
−1
−2
X2
−2
0
2
4
X1
as.factor(y)
0
1