

PEC3: Diagnostic prediction of cancers using gene expression profiling

Roger_Massaguer

2023-01-03

Sumario

Algoritmo Support Vector Machine	2
Fortalezas y debilidades del algoritmo SVM	2
Recoger datos de var. predictora y la clase	2
Exploracion de los datos	3
Entrenar el modelo SVM lineal con datos aportados	5
Evaluacion del rendimiento del algoritmo	5
Modelo SVM con kernel rbf	7
Modelo SVM lineal con 3-fold crossvalidation	8
Modelo SVM con kernel gaussiano con 3-fold crossvalidation	9
Referencias:	11

Algoritmo Support Vector Machine

Los algoritmos SVM (*Support Vector Machine*) procesan datos distribuidos en un espacio multidimensional, y los utilizan para definir una superficie (hiperplano) que divide dicho espacio en regiones homogéneas que agrupan observaciones afines. Estos algoritmos se usan en el campo del aprendizaje automático tanto para clasificar observaciones, como para hacer predicciones. Este pequeño script pondrá en aplicación una de sus funciones más sencillas, la clasificación de un conjunto de observaciones en dos grupos diferenciados.

Cuando las observaciones de ambos grupos están claramente separadas y pueden separarse completamente mediante una línea, plano, o su equivalente multidimensional (hiperplano), hablamos de **datos separables linealmente**. Sin embargo, en la vida real es muy común que la relación entre variables no sea lineal. En estos casos, se pueden seguir dos estrategias: aplicación de la *slack variable* (variable fluctuante) y el uso del *kernel trick* (cambio de la función kernel).

El caso de la variable *slack* consiste en añadir una variable al algoritmo que “permite” dejar observaciones de entrenamiento fuera del grupo al que propiamente pertenecen. Aquí lo que produce el algoritmo es una optimización del hiperplano para minimizar la distancia de esas observaciones hasta el límite que define su clasificación correcta.

El *kernel trick* es matemáticamente más complejo y consiste en añadir nuevas dimensiones a los datos y usarlas para identificar las relaciones no lineales entre las variables. Aquí las superficies que definen los límites entre grupos ya no son hiperplanos, sino curvas de mayor o menor complejidad.

Fortalezas y debilidades del algoritmo SVM

Fortalezas	Debilidades
- Se puede usar tanto para problemas de clasificación como para predicción numérica.	- Encontrar el modelo más adecuado requiere probar varias combinaciones de kernels y parámetros.
- Poco sensible al ruido en los datos y poco propenso al sobreajuste (overfitting).	- El entrenamiento puede ser lento, en especial si el set de datos tiene un gran número de características o ejemplos.
- Puede ser de más fácil uso que las redes neurales, especialmente debido a la existencia de varios algoritmos SVM que han sido adaptados para su uso en ciencia de datos.	- El modelo resultante del entrenamiento es de tipo caja negra, complejo y difícil (si no imposible) de interpretar.
- Popular debido a su alta precisión y a la fama debida a su uso ganador en competiciones de minería de datos.	

Recoger datos de var. predictora y la clase

```
#Lectura de datos:  
data <- read.csv("C:/Users/Usuario/Desktop/UOC_8.22/ML/PEC3/data3.csv")  
class <- read.csv("C:/Users/Usuario/Desktop/UOC_8.22/ML/PEC3/class3.csv", sep="")
```

```
observaciones <- nrow(data)
variables <- ncol(data)
```

El primer conjunto de datos denominado data3.csv esta formado por **102** muestras con **5563** variables.

El segundo conjunto de datos denominado class3.csv corresponde a la clase con notación numérica.

Añadir variable clase como factor al conjunto de datos:

```
lab.tumor <- c("B-like", "ERBB2p", "Nrm", "Lum-B.C")
clase.lab.tumor <- factor(class$x, labels=lab.tumor)
clase.lab.tumor
```

```
## [1] B-like B-like
## [10] B-like B-like
## [19] B-like B-like B-like B-like B-like B-like B-like B-like B-like ERBB2p ERBB2p
## [28] ERBB2p ERBB2p ERBB2p ERBB2p ERBB2p ERBB2p ERBB2p ERBB2p ERBB2p ERBB2p
## [37] ERBB2p ERBB2p ERBB2p ERBB2p ERBB2p ERBB2p ERBB2p ERBB2p ERBB2p ERBB2p
## [46] ERBB2p ERBB2p ERBB2p ERBB2p ERBB2p ERBB2p Nrm Nrm Nrm
## [55] Nrm Nrm Nrm Nrm Nrm Nrm Nrm Nrm Nrm Nrm
## [64] Nrm Nrm Nrm Nrm Nrm Nrm Nrm Nrm Nrm Nrm
## [73] Nrm Nrm Nrm Nrm Nrm Nrm Nrm Lum-B.C Lum-B.C
## [82] Lum-B.C Lum-B.C Lum-B.C Lum-B.C Lum-B.C Lum-B.C Lum-B.C Lum-B.C Lum-B.C Lum-B.C
## [91] Lum-B.C Lum-B.C Lum-B.C Lum-B.C Lum-B.C Lum-B.C Lum-B.C Lum-B.C Lum-B.C Lum-B.C
## [100] Lum-B.C Lum-B.C Lum-B.C
## Levels: B-like ERBB2p Nrm Lum-B.C
```

```
data$class <- clase.lab.tumor
data$class <- as.factor(class$x)
```

Exploracion de los datos

Extraer las 6 primeras muestras de las 5 primeras variables

```
head(data[,1:5])
```

```
##          V1          V2          V3          V4          V5
## 1  1.1043638  1.4011377  0.1712629  2.7061461 -1.3629389
## 2  1.2803340 -1.6223058  1.3399947  1.3140683 -1.1419608
## 3  0.9130290  0.1367017 -1.6160803 -0.3744629 -1.0826116
## 4  0.2391991  0.3446135 -1.0406707  2.6105714 -0.1102325
## 5  0.9949015 -0.1480161  0.3895256 -1.0517833  1.5284879
## 6 -0.6098422  1.8347321  1.6965736  0.9448399  0.1629590
```

Extraer las 6 ultimas observaciones de las 5 ultimas variables

```
tail(data[,5560:5564])
```

```
##          V5560          V5561          V5562          V5563 class
## 97  1.1827351  0.56943947  0.7269450 -0.06584812     4
```

```

## 98 -1.7883522 0.98311220 3.2240096 -0.89758072      4
## 99 -1.1023023 -1.35315429 2.8120933 -0.42529442      4
## 100 0.8995841 -0.42647449 0.9197074 0.27932703      4
## 101 -1.1487759 1.98913351 2.8632942 1.58779831      4
## 102 1.6037020 -0.05275914 3.0462997 -1.21177966      4

```

Muestra la estructura de las anteriores 9 variables

```
str(data[1:6, (ncol(data)-8):ncol(data)])
```

```

## 'data.frame':   6 obs. of  9 variables:
## $ V5556: num -0.364 1.087 0.438 1.77 1.44 ...
## $ V5557: num 0.984 3.34 -0.218 0.107 1.163 ...
## $ V5558: num -0.215 2.248 0.149 -0.847 -0.77 ...
## $ V5559: num -1.443 1.801 -0.494 1.436 1.574 ...
## $ V5560: num 2.16 -1.081 -0.921 0.308 -0.772 ...
## $ V5561: num 1.253 3.112 -0.328 2.515 3.297 ...
## $ V5562: num 1.08 -0.86 -2.784 -0.325 1.507 ...
## $ V5563: num -2.61 -1.33 1.54 1.43 -3.16 ...
## $ class: Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1

```

Breve estadistica descriptiva de las 9 anteriores variables es:

```
summary(data[, (ncol(data)-8):ncol(data)])
```

	V5556	V5557	V5558	V5559	
## Min.	-2.61213	-3.2000	-2.8988	-3.8974	
## 1st Qu.:-0.77095	0.17088	0.2806	0.2028	0.3301	
## Median :	-0.01851	0.2132	0.2067	0.4029	
## Mean	0.70450	1.0635	1.1553	1.1776	
## 3rd Qu.:	3.09805	3.3399	3.4640	4.5988	
## Max. :	3.1529	3.7790	4.2537	4.6032	
	V5560	V5561	V5562	V5563	class
## Min. :-2.3569	Min. :-2.2349	Min. :-3.2784	Min. :-3.1612	1:25	
## 1st Qu.:-0.7315	1st Qu.:-0.4018	1st Qu.:-0.5647	1st Qu.:-0.6985	2:26	
## Median :	0.2147	0.3901	0.6783	0.2659	3:28
## Mean	0.1745	0.5098	0.5645	0.2270	4:23
## 3rd Qu.:	1.0926	1.3866	1.7810	0.9307	
## Max. :	3.1529	3.7790	4.2537	4.6032	

Numero de observaciones para cada clase

```
table(data$class)
```

```

##
## 1 2 3 4
## 25 26 28 23

```

Dividir el dataset en una parte de entrenamiento y en otra de test.

Generamos un indice de unos y doses para el numero de filas de nuestro conjunto de datos —nrow— con las probabilidades deseadas, en nuestro ejemplo un 67% y un 33%. Con ese indice creamos los dos subconjuntos: data_training y data_test.

```

set.seed(12345)
ind <- sample(2, nrow(data), replace = TRUE, prob = c(0.67, 0.33))
data_training <- data[ind == 1, ]
data_test <- data[ind == 2, ]

```

Entrenar el modelo SVM lineal con datos aportados

Usando la función ksvm del paquete kernlab.

```

install.packages("kernlab", repos = "http://cran.us.r-project.org")

## Installing package into 'C:/Users/Usuario/AppData/Local/R/win-library/4.2'
## (as 'lib' is unspecified)

## package 'kernlab' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##   C:\Users\Usuario\AppData\Local\Temp\Rtmp4sTTkT\downloaded_packages

library(kernlab)
set.seed(1234567)
data_model1 <- ksvm(class ~ ., data = data_training,
                     kernel = "vanilladot")

## Setting default kernel parameters

data_model1

## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc (classification)
## parameter : cost C = 1
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 64
##
## Objective Function Value : -0.0012 -0.002 -0.0015 -0.0013 -0.0011 -0.0017
## Training error : 0

```

Evaluación del rendimiento del algoritmo

Valoración con los datos del test y clasificar las muestras de los datos de test con la función predict.

```

data_predict1 <- predict(data_model1, data_test)

res <- table(data_predict1, data_test$class)

```

Matriz de confusión con las predicciones y las clases.

La función confusionMatrix del paquete caret genera esta matriz.

```
install.packages("caret", repos = "http://cran.us.r-project.org")

## Installing package into 'C:/Users/Usuario/AppData/Local/R/win-library/4.2'
## (as 'lib' is unspecified)

## package 'caret' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
## C:\Users\Usuario\AppData\Local\Temp\Rtmp4sTTkT\downloaded_packages

library(caret)

## Loading required package: ggplot2

##
## Attaching package: 'ggplot2'

## The following object is masked from 'package:kernlab':
## alpha

## Loading required package: lattice

(conf_mat.s1 <- confusionMatrix(res))

## Confusion Matrix and Statistics
##
## 
## data_predict1  1  2  3  4
##               1 10  0  0  1
##               2  0  9  0  0
##               3  0  0 10  0
##               4  0  0  1  7
##
## Overall Statistics
##
##           Accuracy : 0.9474
##             95% CI : (0.8225, 0.9936)
##   No Information Rate : 0.2895
##   P-Value [Acc > NIR] : < 2.2e-16
##
##           Kappa : 0.9296
##
## McNemar's Test P-Value : NA
##
## Statistics by Class:
```

```

##          Class: 1 Class: 2 Class: 3 Class: 4
## Sensitivity      1.0000   1.0000   0.9091   0.8750
## Specificity     0.9643   1.0000   1.0000   0.9667
## Pos Pred Value   0.9091   1.0000   1.0000   0.8750
## Neg Pred Value   1.0000   1.0000   0.9643   0.9667
## Prevalence       0.2632   0.2368   0.2895   0.2105
## Detection Rate   0.2632   0.2368   0.2632   0.1842
## Detection Prevalence 0.2895   0.2368   0.2632   0.2105
## Balanced Accuracy 0.9821   1.0000   0.9545   0.9208

```

El algoritmo de SVM lineal tiene un valor de precision de 0.95 y un estadistico K = 0.93. Los valores de sensibilidad y especificidad varian segun el factor, obteniendo como valor medio 0.94 y 0.98 respectivamente.

Modelo SVM con kernel rbf

SVM con funcion gaussiana o rbf.

```

set.seed(12345)
data_model2 <- ksvm(class ~ ., data = data_training,
                     kernel = "rbfdot")
data_model2

## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
## parameter : cost C = 1
##
## Gaussian Radial Basis kernel function.
## Hyperparameter : sigma =  8.96024081868479e-05
##
## Number of Support Vectors : 64
##
## Objective Function Value : -13.8628 -18.746 -16.0586 -14.6053 -12.6101 -17.3222
## Training error : 0

```

Evaluar rendimiento del modelo con los datos de test. Se debe de clasificar las muestras de los datos de test con la funcion predict.

```

data_predict2 <- predict(data_model2, data_test)
res <- table(data_predict2, data_test$class)

```

Se obtiene la matriz de confusion con las predicciones y las clases reales. La función confusionMatrix del paquete caret genera esta matriz y calcula diferentes del rendimiento del algoritmo.

```
(conf_mat.s2 <- confusionMatrix(res))
```

```

## Confusion Matrix and Statistics
##
## 
## data_predict2  1  2  3  4

```

```

##          1 3 0 0 0
##          2 0 9 0 0
##          3 7 0 11 3
##          4 0 0 0 5
##
## Overall Statistics
##
##           Accuracy : 0.7368
##             95% CI : (0.569, 0.866)
##   No Information Rate : 0.2895
##   P-Value [Acc > NIR] : 1.513e-08
##
##           Kappa : 0.6422
##
## McNemar's Test P-Value : NA
##
## Statistics by Class:
##
##           Class: 1 Class: 2 Class: 3 Class: 4
## Sensitivity      0.30000  1.00000  1.00000  0.6250
## Specificity       1.00000  1.00000  0.6296  1.0000
## Pos Pred Value    1.00000  1.00000  0.5238  1.0000
## Neg Pred Value    0.80000  1.00000  1.00000 0.9091
## Prevalence        0.26316  0.2368  0.2895  0.2105
## Detection Rate    0.07895  0.2368  0.2895  0.1316
## Detection Prevalence 0.07895  0.2368  0.5526  0.1316
## Balanced Accuracy  0.65000  1.00000  0.8148  0.8125

```

El algoritmo de SVM de funcion RBF tiene un valor de precision de 0.74 y un estadistico k = 0.64. Como se puede esperar, los valores de sensibilidad y especificidad varian segun la clase, obteniendo como valor medio 0.73 y 0.90 respectivamente. En resumen, se puede decir que el modelo SVM con la funcion RBF obtiene una menor precision que el modelo de SVM con funcio lineal. Ademas, el nuevo modelo obtenido de SVM con la función RBF tiene menor valor de kappa que el modelo mas sencillo de SVM con función lineal.

Modelo SVM lineal con 3-fold crossvalidation

Realizar el algoritmo de SVM con la funcion lineal con 3-fold crossvalidation usando el paquete caret.

El modelo de entrenamiento es:

```

set.seed(12345)
model_sc <- train(class ~ ., data, method='svmLinear',
                    trControl= trainControl(method='cv', number=3),
                    tuneGrid= NULL, trace = FALSE)
model_sc

## Support Vector Machines with Linear Kernel
##
## 102 samples
## 5563 predictors
##    4 classes: '1', '2', '3', '4'
##

```

```

## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 68, 67, 69
## Resampling results:
##
##   Accuracy   Kappa
##   0.9607843  0.9475915
##
## Tuning parameter 'C' was held constant at a value of 1

```

El modelo obtenido con el algoritmo de SVM lineal con 3-fold crossvalidation tiene una precision de 0.96 y un valor k de 0.95. Finalmente, podemos decir que el modelo obtenido con la funcion ‘svmLinear’ y 3-fold crossvalidation obtiene mayor precision que el modelo SVM de función lineal con particion train/test . Ademas, el modelo obtenido con la funcion ‘svmLinear’ y 3-fold crossvalidation tiene mayor valor de kappa que el modelo SVM de funcion lineal con partición train/test .

Modelo SVM con kernel gaussiano con 3-fold crossvalidation

Evaluacion rendimiento del algoritmo SVM con kernel RBF para diferentes valores del hiperparámetro C y sigma.

Se ha escogido como valores de sigma y C:

```
(mysigma <- c(.008, .009, .01, .011, .012, .013))
```

```
## [1] 0.008 0.009 0.010 0.011 0.012 0.013
```

```
(myC <- c(0.9, 1, 1.1, 1.25, 1.35))
```

```
## [1] 0.90 1.00 1.10 1.25 1.35
```

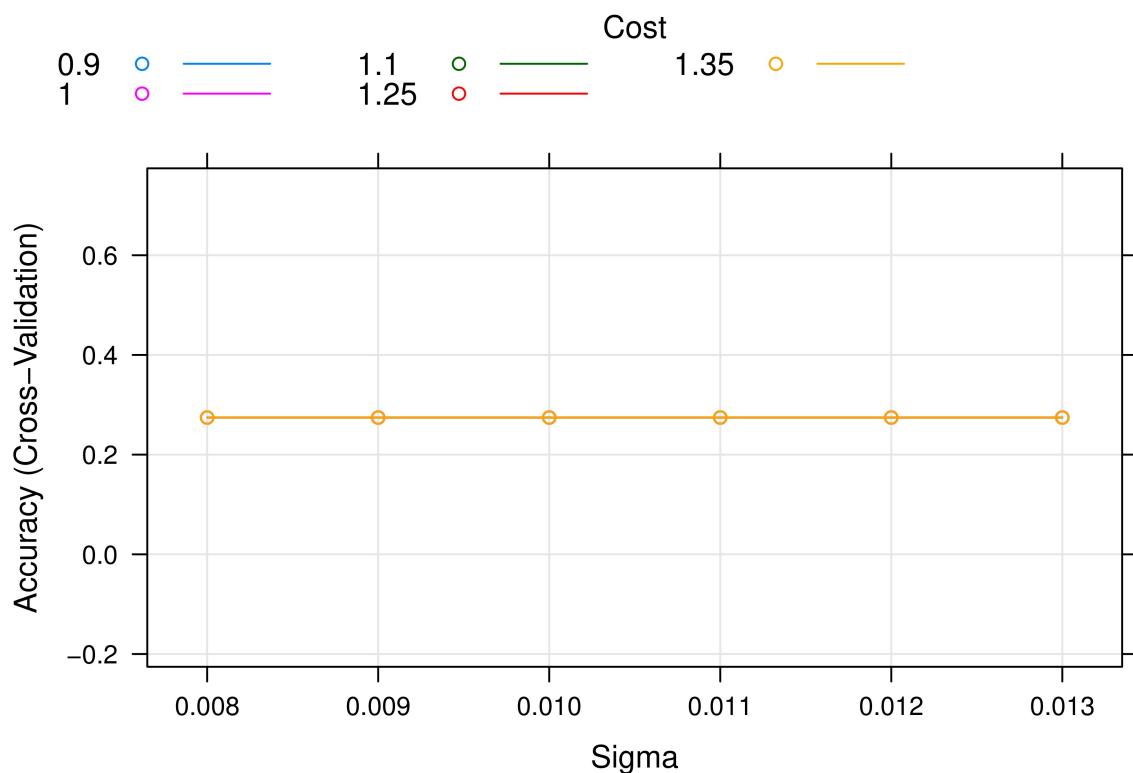
La evolucion del rendimiento es:

```

set.seed(12345)

grid <- expand.grid(sigma = mysigma, C = myC)
model_sr <- train(class ~ ., data, method='svmRadial',
                    trControl= trainControl(method='cv', number=3),
                    tuneGrid= grid, trace = FALSE)
plot(model_sr)

```



```
model_sr
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 102 samples
## 5563 predictors
##    4 classes: '1', '2', '3', '4'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 68, 67, 69
## Resampling results across tuning parameters:
##
##   sigma  C    Accuracy  Kappa
##   0.008  0.90  0.2743825  0
##   0.008  1.00  0.2743825  0
##   0.008  1.10  0.2743825  0
##   0.008  1.25  0.2743825  0
##   0.008  1.35  0.2743825  0
##   0.009  0.90  0.2743825  0
##   0.009  1.00  0.2743825  0
##   0.009  1.10  0.2743825  0
##   0.009  1.25  0.2743825  0
##   0.009  1.35  0.2743825  0
##   0.010  0.90  0.2743825  0
##   0.010  1.00  0.2743825  0
```

```

##  0.010  1.10  0.2743825  0
##  0.010  1.25  0.2743825  0
##  0.010  1.35  0.2743825  0
##  0.011  0.90  0.2743825  0
##  0.011  1.00  0.2743825  0
##  0.011  1.10  0.2743825  0
##  0.011  1.25  0.2743825  0
##  0.011  1.35  0.2743825  0
##  0.012  0.90  0.2743825  0
##  0.012  1.00  0.2743825  0
##  0.012  1.10  0.2743825  0
##  0.012  1.25  0.2743825  0
##  0.012  1.35  0.2743825  0
##  0.013  0.90  0.2743825  0
##  0.013  1.00  0.2743825  0
##  0.013  1.10  0.2743825  0
##  0.013  1.25  0.2743825  0
##  0.013  1.35  0.2743825  0
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.013 and C = 0.9.

```

Por tanto, el mejor modelo de SVM con kernel gaussiano de los revisados basado en la “accuracy” tiene como hiperparametros:

```
model_sr$bestTune
```

```

##   sigma   C
## 26 0.013 0.9

```

Su rendimiento es

```
model_sr$results[which.max(model_sr$results[,3]),3:6]
```

```

##   Accuracy Kappa AccuracySD KappaSD
## 1 0.2743825    0 0.01060156    0

```

Referencias:

Lantz, B. (2019). Machine Learning with R Second Edition all your data analysis problems.

Machines, S. V. (2000). Unidad 6: Predicción del tipo de tejido normal / tumoral en cáncer de colon usando el algoritmo de Support Vector Machines . 3.

Gu, M. L., & Semanales, E. (s.d.). Comentario general sobre los contenidos de la segunda unidad Materiales de estudio Actividades complementarias. 7, 7-8.

6deeb154c3668d9105eca4b937a411cf6d2e4617 @ materials.campus.uoc.edu. (s.d.). https://materials.campus.uoc.edu/cdocent/PID_00285309/

516557e1427b51fc2ba024d847c80668fb17353c @ materials.campus.uoc.edu. (s.d.). https://materials.campus.uoc.edu/cdocent/PID_00285305/