Tarea de SVM

Cedric Prieels

Diciembre 2016

Resolución del ejercicio

```
rm(list=ls())
if (!require(e1071)) install.packages("e1071")
```

```
## Loading required package: e1071
```

```
library(e1071)
if (!require(ROCR)) install.packages("ROCR")
```

```
## Loading required package: ROCR

## Loading required package: gplots

##
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':
##
##     lowess
```

```
library(ROCR)
if (!require(rgl)) install.packages("rgl")
```

```
## Loading required package: rgl
```

```
library(rgl)
setwd('/Users/ced2718/Documents/Universite/Modelizacion/')
```

Primero, abrimos los ficheros de datos con R.

```
veg<-read.table("veg.txt", header=T)
environ<-read.table("environ.txt", header=T)
```

Queremos poder distinguir dos estados diferentes : la presencia, y la ausencia de una especia. Por lo tanto, cambiamos en el fichero de entrada todo que lo que es mayor que 0 para poner 1 (que significa que la especie es presente). También queremos estandarizar la matriz de factores ambientales, para tener datos con una media y una varianza nulas. Quitarmos entonces a cada dato su media para centrar el valor y escalamos. Por fin, creamos un nuevo dataframe llamado *data*, que vamos a usar en todo el ejercicio.

```
veg[veg>0]<-1
environ_scaled<-scale(environ, center=T, scale=T)
data <- data.frame(environ_scaled, veg)
```

Para hacer el ejercicio, lo más fácil consiste en separar este dataframe en diferentes dataframes, cada uno correspondiendo a un tipo de vegetación diferente. También creamos un vector y una lista con los nombres de todas las especies.

```
Cal_vul <- data.frame(x1 = environ_scaled, y = as.factor(veg[,1]))
Car_nig <- data.frame(x2 = environ_scaled, y = as.factor(veg[,2]))
Cyt_oro <- data.frame(x3 = environ_scaled, y = as.factor(veg[,3]))
Eri_ara <- data.frame(x4 = environ_scaled, y = as.factor(veg[,4]))
Eri_arb <- data.frame(x5 = environ_scaled, y = as.factor(veg[,5]))
Eri_tet <- data.frame(x6 = environ_scaled, y = as.factor(veg[,6]))
Eup_pol <- data.frame(x7 = environ_scaled, y = as.factor(veg[,7]))
Fes_agr <- data.frame(x8 = environ_scaled, y = as.factor(veg[,8]))
Fes_esk <- data.frame(x9 = environ_scaled, y = as.factor(veg[,9]))
Gen_obt <- data.frame(x10 = environ_scaled, y = as.factor(veg[,10]))
Jun_nan <- data.frame(x11 = environ_scaled, y = as.factor(veg[,11]))
Jun_tri <- data.frame(x12 = environ_scaled, y = as.factor(veg[,12]))
Luz_cae <- data.frame(x13 = environ_scaled, y = as.factor(veg[,13]))
Nar_str <- data.frame(x14 = environ_scaled, y = as.factor(veg[,14]))
Vac_myr <- data.frame(x15 = environ_scaled, y = as.factor(veg[,15]))
Vac_uli <- data.frame(x16 = environ_scaled, y = as.factor(veg[,16]))

names <- list("Cal_vul", "Car_nig", "Cyt_oro", "Eri_ara", "Eri_arb",
              "Eri_tet", "Eup_pol", "Fes_agr", "Fes_esk", "Gen_obt",
              "Jun_nan", "Jun_tri", "Luz_cae", "Nar_str", "Vac_myr", "Vac_uli")

list <- list(Cal_vul, Car_nig, Cyt_oro, Eri_ara, Eri_arb,
             Eri_tet, Eup_pol, Fes_agr, Fes_esk, Gen_obt,
             Jun_nan, Jun_tri, Luz_cae, Nar_str, Vac_myr, Vac_uli)
```

Ahora queremos repartir los datos en muestras de train y de test (con una repartición del 70/30%). Por supuesto, no vamos a coger los primeros elementos del dataframe para usar los como train, porque hay que coger una muestra aleatorio dentro del dataframe. Esto se hace mediante el uso de la función *sample*.

```
#Fijamos la semilla para siempre obtener los mismo resultados
set.seed(1)
length.train <- round(nrow(data)*0.7)
length.test <- ceiling(nrow(data)*0.3)

train <- sample(nrow(data), length.train)
```

Lo que queremos hacer es realizar un modelo basado en vectores soporte para cada una de las especies en nuestros dataframes para predecir si una especie va a encontrarse o no en una localidad, con unas características ambientales dadas.

Ya que tenemos hecha toda la preparación del ejercicio, podemos empezar a usar las funciones *svm* y *tune* de R. Vamos a estudiar tres diferentes tipos de *kerneles* en este ejercicio : un kernel lineal, un kernal radial y un kernel de tipo polinomial.

Kernel lineal

En este caso, es necesario usar la función *tune* primera para determinar el mejor valor de coste de este problema, para tener un número de vectores soporte adecuado. El coste se puede ver como una penalización : aumentar su valor nos permite dar una mejor clasificación, pero penaliza la capacidad de generalización (problema habitual de sobreajuste si tenemos demasiados parámetros libres). Siempre hay que preocuparse por su valor porque influya mucho en los resultados. En R, existe una función que nos permite probar distintos valores para ayudarnos, y para sacar el mejor modelo usamos $best.model.

Primero, vamos a definir una función que nos permite calcular los parámetros más importantes de los kerneles lineales en cada caso. Esta función devuelve una lista con diferentes variables : el mejor valor de coste, el número de vectores soporte, el número de datos bien clasificados, y el area debajo de la curva ROC para muestras de train y de test.

```
cost <- c(1e-3, 0.001, 0.01, 0.1, 1, 10)

linear_svm <- function(data, train, cost) {

  #Entrenamiento
  tune.out <- tune(method = svm, y~., data = data[train,],
                       kernel = "linear", ranges = list(cost = cost))
  best.model <- tune.out$best.model
  best.cost <- tune.out$best.parameters[1]
  number.vectors <- length(best.model$index)

  #Usamos la función predict para los datos de test
  ypred <- predict(best.model, newdata = data[-train,])
  predictions <- table(predicted = ypred, observed = data[-train,]$y)
  correct.number <- sum(diag(predictions))

  #Area debajo de la ROC en entrenamiento
  fitted.train <- attributes(predict(best.model, data[train,], decision.values = TRUE))$decision.values
  auc.train <- performance(prediction(fitted.train, data[train,]$y), measure = "auc")@y.values

  # Area debajo de la ROC en test
  fitted.test <- attributes(predict(best.model, data[-train,], decision.values = TRUE))$decision.values
  auc.test <- performance(prediction(fitted.test, data[-train,]$y), measure = "auc")@y.values

  list(best_cost = as.numeric(best.cost), number_vectors = number.vectors,
       correctly_predicted = correct.number, auc_train = round(as.numeric(auc.train),3),
       auc_test = round(as.numeric(auc.test),3))

}
```

Ahora aplicamos esta función a todos los dataframe que contienen los datos de cada especie y ponemos los resultados obtenidos en una tabla.

```
best_costs <- rep("NA", length(list))
number_vectors <- rep("NA", length(list))
correct_predictions <- rep("NA", length(list))
auc_train <- rep("NA", length(list))
auc_test <- rep("NA", length(list))

for(i in 1:length(list)){
```

```
  output <- linear_svm(list[[i]], train, cost)
  best_costs[i] <- output$best_cost
  number_vectors[i] <- output$number_vectors
  correct_predictions[i] <- output$correctly_predicted
  auc_train[i] <- output$auc_train
  auc_test[i] <- output$auc_test

}

linear_kernel <- data.frame(v1=best_costs, v2=number_vectors,
                            v3=correct_predictions, v4=auc_train, v5=auc_test)
colnames(linear_kernel) <- c("Best cost value", "Number of suport vectors",
                             "Correctly predicted", "AUC train", "AUC test")
rownames(linear_kernel) <- names
linear_kernel
```

```
##         Best cost value Number of suport vectors Correctly predicted
## Cal_vul           0.001                      110                 104
## Car_nig              10                      111                 104
## Cyt_oro           0.001                       77                 112
## Eri_ara           0.001                       20                 124
## Eri_arb             0.1                       87                 104
## Eri_tet              10                      141                 104
## Eup_pol               1                      106                 100
## Fes_agr             0.1                      135                  92
## Fes_esk            0.01                      170                 110
## Gen_obt               1                      116                 109
## Jun_nan              10                      240                  73
## Jun_tri              10                       42                 111
## Luz_cae             0.1                      116                 102
## Nar_str           0.001                      101                 112
## Vac_myr             0.1                      157                  90
## Vac_uli               1                       52                 119
##         AUC train AUC test
## Cal_vul     0.644    0.515
## Car_nig      0.14    0.156
## Cyt_oro     0.209    0.148
## Eri_ara     0.067    0.565
## Eri_arb      0.13    0.113
## Eri_tet     0.262    0.142
## Eup_pol     0.066     0.13
## Fes_agr     0.099    0.177
## Fes_esk     0.954    0.934
## Gen_obt     0.083    0.081
## Jun_nan     0.687    0.595
## Jun_tri     0.021    0.207
## Luz_cae     0.936     0.88
## Nar_str     0.825    0.734
## Vac_myr     0.857    0.754
## Vac_uli     0.046    0.096
```

Kernel radial

Ahora volvemos a hacer exactamente lo mismo que antes, pero usando kerneles de tipo radiales. Por lo tanto, en este caso tenemos que preocuparnos también del valor del parámetro *gamma*, y no solamente del coste. Como antes, creamos primero una función que nos devuelve una lista con los parámetros más importantes.

```
gamma <- c(0.5, 1, 2, 3)

radial_svm <- function(data, train, cost) {

  #Entrenamiento
  tune.out <- tune(method = svm, y~., data = data[train,],
                     kernel = "radial", ranges = list(cost = cost, gamma = gamma))
  best.model <- tune.out$best.model
  best.cost <- tune.out$best.parameters[1]
  best.gamma <- tune.out$best.parameters[2]
  number.vectors <- length(best.model$index)

  #Usamos la función predict para los datos de test
  ypred <- predict(best.model, newdata = data[-train,])
  predictions <- table(predicted = ypred, observed = data[-train,]$y)
  correct.number <- sum(diag(predictions))

  #Area debajo de la ROC en entrenamiento
  fitted.train <- attributes(predict(best.model, data[train,], decision.values = TRUE))$decision.values
  auc.train <- performance(prediction(fitted.train, data[train,]$y), measure = "auc")@y.values

  # Area debajo de la ROC en test
  fitted.test <- attributes(predict(best.model, data[-train,], decision.values = TRUE))$decision.values
  auc.test <- performance(prediction(fitted.test, data[-train,]$y), measure = "auc")@y.values

  list(best_gamma = as.numeric(best.gamma), best_cost = as.numeric(best.cost),
       number_vectors = number.vectors, correctly_predicted = correct.number,
       auc_train = round(as.numeric(auc.train),3), auc_test = round(as.numeric(auc.test),3))

}
```

Calculamos ahora los parámetros importantes devueltos por nuestra función y imprimimos por pantalla en una tabla los resultados obtenidos para cada especie.

```
best_gammas <- rep("NA", length(list))
best_costs <- rep("NA", length(list))
number_vectors <- rep("NA", length(list))
correct_predictions <- rep("NA", length(list))
auc_train <- rep("NA", length(list))
auc_test <- rep("NA", length(list))

for(i in 1:length(list)){

  output <- radial_svm(list[[i]], train, cost)
  best_gammas[i] <- output$best_gamma
  best_costs[i] <- output$best_cost
  number_vectors[i] <- output$number_vectors
  correct_predictions[i] <- output$correctly_predicted
```

```
  auc_train[i] <- output$auc_train
  auc_test[i] <- output$auc_test

}

radial_kernel <- data.frame(v1 = best_gammas, v2=best_costs,
                            v3=number_vectors, v4=correct_predictions,
                            v5=auc_train, v6=auc_test)
colnames(radial_kernel) <- c("Best gamma value", "Best cost value",
                             "Number of suport vectors", "Correctly predicted",
                             "AUC train", "AUC test")
rownames(radial_kernel) <- names
radial_kernel
```

```
##         Best gamma value Best cost value Number of suport vectors
## Cal_vul                3               1                       263
## Car_nig              0.5               1                       204
## Cyt_oro              0.5              10                       147
## Eri_ara              0.5           0.001                        15
## Eri_arb                2               1                       251
## Eri_tet              0.5               1                       227
## Eup_pol                1               1                       241
## Fes_agr              0.5               1                       200
## Fes_esk              0.5               1                       184
## Gen_obt              0.5              10                       185
## Jun_nan                3               1                       284
## Jun_tri              0.5               1                       148
## Luz_cae              0.5               1                       187
## Nar_str                2               1                       248
## Vac_myr              0.5               1                       215
## Vac_uli              0.5           0.001                        69
##         Correctly predicted AUC train AUC test
## Cal_vul                 104         1     0.48
## Car_nig                 102     0.046    0.169
## Cyt_oro                 109     0.002    0.212
## Eri_ara                 124     0.129    0.089
## Eri_arb                 102     0.002     0.18
## Eri_tet                 105     0.057    0.186
## Eup_pol                 102     0.017    0.117
## Fes_agr                  90     0.034    0.246
## Fes_esk                 109     0.989    0.941
## Gen_obt                  98     0.012    0.177
## Jun_nan                  72     0.992     0.64
## Jun_tri                 106     0.001    0.117
## Luz_cae                  95     0.988    0.803
## Nar_str                 113         1    0.774
## Vac_myr                  98     0.948    0.808
## Vac_uli                 118      0.04    0.268
```

Kernel polinomial

Por fin, intentamos un otro tipo de kernel para separar nuestros puntos : los polinomios, de diferentes grados (en este caso, consideramos polinomios de grado 2, 3 y 4 como los polinomios de grado 1 ya están cubiertos

por la función de tipo lineal). Primero, volvemos a definir una función de la misma manera que antes.

```
degree <- c(2, 3, 4)

poli_svm <- function(data, train, cost, degree) {

  #Entrenamiento
  tune.out <- tune(method = svm, y~., data = data[train,],
                       kernel = "polynomial", ranges = list(cost = cost, degree = degree))
  best.model <- tune.out$best.model
  best.cost <- tune.out$best.parameters[1]
  best.degree <- tune.out$best.parameters[2]
  number.vectors <- length(best.model$index)

  #Usamos la función predict para los datos de test
  ypred <- predict(best.model, newdata = data[-train,])
  predictions <- table(predicted = ypred, observed = data[-train,]$y)
  correct.number <- sum(diag(predictions))

  #Area debajo de la ROC en entrenamiento
  fitted.train <- attributes(predict(best.model, data[train,], decision.values = TRUE))$decision.values
  auc.train <- performance(prediction(fitted.train, data[train,]$y), measure = "auc")@y.values

  # Area debajo de la ROC en test
  fitted.test <- attributes(predict(best.model, data[-train,], decision.values = TRUE))$decision.values
  auc.test <- performance(prediction(fitted.test, data[-train,]$y), measure = "auc")@y.values

  list(best_degree = as.numeric(best.degree), best_cost = as.numeric(best.cost),
       number_vectors = number.vectors, correctly_predicted = correct.number,
       auc_train = round(as.numeric(auc.train),3), auc_test = round(as.numeric(auc.test),3))

}
```

E imprimimos por pantalla una tabla con todos los resultados obtenidos.

```
best_degrees <- rep("NA", length(list))
best_costs <- rep("NA", length(list))
number_vectors <- rep("NA", length(list))
correct_predictions <- rep("NA", length(list))
auc_train <- rep("NA", length(list))
auc_test <- rep("NA", length(list))

for(i in 1:length(list)){

  output <- poli_svm(list[[i]], train, cost, degree)
  best_degrees[i] <- output$best_degree
  best_costs[i] <- output$best_cost
  number_vectors[i] <- output$number_vectors
  correct_predictions[i] <- output$correctly_predicted
  auc_train[i] <- output$auc_train
  auc_test[i] <- output$auc_test

}
```

```
poli_kernel <- data.frame(v1 = best_degrees, v2=best_costs,
                          v3=number_vectors, v4=correct_predictions,
                          v5=auc_train, v6=auc_test)
colnames(poli_kernel) <- c("Best degree value", "Best cost value",
                           "Number of suport vectors", "Correctly predicted",
                           "AUC train", "AUC test")
rownames(poli_kernel) <- names
poli_kernel
```

```
##         Best degree value Best cost value Number of suport vectors
## Cal_vul                 2           0.001                       105
## Car_nig                 2              10                       132
## Cyt_oro                 2              10                        68
## Eri_ara                 4             0.1                        30
## Eri_arb                 2              10                        96
## Eri_tet                 2              10                       164
## Eup_pol                 2              10                       160
## Fes_agr                 2              10                       176
## Fes_esk                 3              10                       156
## Gen_obt                 3              10                       155
## Jun_nan                 2              10                       229
## Jun_tri                 2              10                        62
## Luz_cae                 2              10                       151
## Nar_str                 4              10                        95
## Vac_myr                 2              10                       175
## Vac_uli                 2               1                        65
##         Correctly predicted AUC train AUC test
## Cal_vul                 104     0.747    0.653
## Car_nig                 101     0.113    0.249
## Cyt_oro                 112      0.07    0.166
## Eri_ara                 124     0.003    0.984
## Eri_arb                 103     0.092    0.192
## Eri_tet                 100     0.125    0.244
## Eup_pol                 102     0.055     0.16
## Fes_agr                  91     0.089    0.245
## Fes_esk                 109     0.978    0.935
## Gen_obt                 100     0.051    0.132
## Jun_nan                  68     0.829    0.584
## Jun_tri                 111     0.003    0.273
## Luz_cae                  93     0.956    0.762
## Nar_str                 110     0.931    0.474
## Vac_myr                  86     0.901     0.74
## Vac_uli                 121     0.092    0.146
```

Conclusion

Ahora podemos comparar los resultados obtenidos con cada método distinto. Ponemos en una tabla el mejor método y sus parámetros asociados, para cada tipo de especie.

Vemos con esta tabla que cos estos datos, el kernal de tipo radial parece el menos eficiente porque en general (expecto una especia), los kerneles de tipo lineal y polynomial dan resultados mejores. También podemos observar que en general, los kerneles polinomiales de grado menos alto están más representados. Vemos por

Nombre	Kernel AUC test	Coste	γ	Grado	Vec. sop.	Pred. Corr.	AUC train	
Cal_vul	Polynomial	0.001	/	2	105	104	0.747	0.653
Cal_nig	Lineal	10	/	1	111	104	0.14	0.156
Cyt_oro	Polynomial	10	/	2	68	112	0.07	0.166
Eri_ara	Radial	0.001	0.5	/	15	124	0.129	0.089
Eri_arb	Lineal	0.1	/	1	87	104	0.13	0.113
Eri_tet	Lineal	10	/	1	141	104	0.254	0.138
Eup_pol	Lineal	1	/	1	106	100	0.066	0.13
Fes_agr	Lineal	0.1	/	1	135	92	0.099	0.177
Fes_esk	Polynomial	10	/	3	156	109	0.978	0.935
Gen_obt	Lineal	1	/	1	116	109	0.083	0.081
Jun_nan	Polynomial	10	/	2	229	68	0.829	0.584
Jun_tri	Lineal	10	/	1	42	111	0.021	0.207
Luz_cae	Lineal	0.1	/	1	116	102	0.936	0.88
Nar_str	Polynomial	10	/	4	95	110	0.931	0.474
Vac_myr	Lineal	0.1	/	1	157	90	0.857	0.754
Vac_uli	Lineal	1	/	1	52	119	0.046	0.096

Table 1: Resultados del mejor kernel para diferentes tipos de especie.

fin que el número minimo de vectores soporte es 15 (Eri_ara) y que el número máximo de vectores soporte obtenido vale 229 (gran riesgo de sobreajuste, igual estaría mejor cambiar los valores posible de coste, de gamma y de grado para intentar encontrar una solución con un número de vectores soporte menor).

Bibliografia

R Markdown, *Markdown basics*, http://rmarkdown.rstudio.com/authoring_basics.html. Consultado por última vez el 29 de octubre 2016.

R Development Core Team (2008). *R: A language and environment for statistical computing. R Foundation for Statistical Computing*, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

How do I make a list of data frames?, STACKOVERFLOW, http://stackoverflow.com/questions/17499013/how-do-i-make-a-list-of-data-frames, consultado por última vez el 10 de diciembre 2016.

How to assign from a function which returns more than one value?, STACKOVERFLOW, http://stackoverflow.com/questions/1826519/how-to-assign-from-a-function-which-returns-more-than-one-value, consultado por última vez el 12 de diciembre 2016.