Tema7_Ejercicio_SVM

Fran Camacho

2025-03-05

Tema 7 - Ejercicio SVM

Ejercicio equivalente al de redes de neuronas, pero usando "support vector machines".

Paso 1 – Carga de los datos

Mismos datos que en la parte de redes de neuronas.

```
# import the CSV file
bank_raw <- read.csv(file.path("Chapter07/Bank", "bank.csv"), sep = ";", stringsAsFactors = TRUE)
```

Paso 2 – Exploración y preparación de los datos

Importar librerías necesarias:

```
#https://www.jstatsoft.org/article/view/v011i09
if (!require(kernlab)) install.packages('kernlab', dependencies = T)
```

```
## Cargando paquete requerido: kernlab
```

```
library(kernlab)

if (!require(caret)) install.packages('caret', dependencies = T)
```

```
## Cargando paquete requerido: caret

## Cargando paquete requerido: ggplot2

##
## Adjuntando el paquete: 'ggplot2'

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

## Cargando paquete requerido: lattice
```

```r
library(caret)

# LIBSVM   https://www.csie.ntu.edu.tw/~cjlin/libsvm/
if (!require(e1071)) install.packages('e1071', dependencies = T)
```

```
## Cargando paquete requerido: e1071
```

```r
library(e1071)
```

Preparamos el dataset de igual manera que para las redes de neuronas:

```r
#scale numeric variables (neither day nor month)
maxs <- apply(bank_raw[c(1,6,12,13,14,15)], 2, max)
mins <- apply(bank_raw[c(1,6,12,13,14,15)], 2, min)

bank_norm <- data.frame(scale(bank_raw[c(1,6,12,13,14,15)], center = mins, scale = maxs - mins))

#hot encoding of categorical features
dummies <- dummyVars(" ~ job + marital + education + default + housing + loan + contact + poutcome", da
bank_hot_encoded_feat <-  data.frame(predict(dummies, newdata = bank_raw))

#encoding month (name to number)
month_to_number <- function(month_name) {
  month_and_number <- c("jan"=1,"feb"=2,"mar"=3,"apr"=4,"may"=5,"jun"=6,"jul"=7,"aug"=8,"sep"=9,"oct"=1
  return(month_and_number[as.character(month_name)])
}
bank_raw$month_num <- sapply(bank_raw$month, month_to_number)


#put all features in the same dataframe
bank_processed <- cbind(bank_norm,as.numeric(bank_raw$day),bank_raw$month_num,bank_hot_encoded_feat,ban
names(bank_processed)[7:8] <- c("day","month")
names(bank_processed)[41] <- "y"
head(bank_processed,5)
```

```
##         age    balance   duration   campaign     pdays previous day month
## 1 0.1617647 0.06845546 0.02482622 0.00000000 0.0000000     0.00  19    10
## 2 0.2058824 0.10875022 0.07149950 0.00000000 0.3899083     0.16  11     5
## 3 0.2352941 0.06258976 0.05991394 0.00000000 0.3795872     0.04  16     4
## 4 0.1617647 0.06428102 0.06454816 0.06122449 0.0000000     0.00   3     6
## 5 0.5882353 0.04446920 0.07348560 0.00000000 0.0000000     0.00   5     5
##   job.admin. job.blue.collar job.entrepreneur job.housemaid job.management
## 1          0               0                0             0              0
## 2          0               0                0             0              0
## 3          0               0                0             0              1
## 4          0               0                0             0              1
## 5          0               1                0             0              0
##   job.retired job.self.employed job.services job.student job.technician
## 1           0                 0            0           0              0
## 2           0                 0            1           0              0
## 3           0                 0            0           0              0
## 4           0                 0            0           0              0
```

```
## 5          0                 0            0           0             0
##   job.unemployed job.unknown marital.divorced marital.married marital.single
## 1              1           0                0               1              0
## 2              0           0                0               1              0
## 3              0           0                0               0              1
## 4              0           0                0               1              0
## 5              0           0                0               1              0
##   education.primary education.secondary education.tertiary education.unknown
## 1                 1                   0                  0                 0
## 2                 0                   1                  0                 0
## 3                 0                   0                  1                 0
## 4                 0                   0                  1                 0
## 5                 0                   1                  0                 0
##   default.no default.yes housing.no housing.yes loan.no loan.yes
## 1          1           0          1           0       1        0
## 2          1           0          0           1       0        1
## 3          1           0          0           1       1        0
## 4          1           0          0           1       0        1
## 5          1           0          0           1       1        0
##   contact.cellular contact.telephone contact.unknown poutcome.failure
## 1                1                 0               0                0
## 2                1                 0               0                1
## 3                1                 0               0                1
## 4                0                 0               1                0
## 5                0                 0               1                0
##   poutcome.other poutcome.success poutcome.unknown  y
## 1              0                0                1 no
## 2              0                0                0 no
## 3              0                0                0 no
## 4              0                0                1 no
## 5              0                0                1 no
```

Finalmente, creamos los conjuntos de entrenamiento y validación de igual manera que para las redes de neuronas:

```r
#Set seed to make the process reproducible
set.seed(9)

#partitioning data frame into training (75%) and testing (25%) sets
train_indices <- createDataPartition(bank_processed$y, times=1, p=.75, list=FALSE)

#create training set
bank_processed_train <- bank_processed[train_indices, ]

#create testing set
bank_processed_test  <- bank_processed[-train_indices, ]


#view number of rows in each set
nrow(bank_processed_train)  # 3391
```

```
## [1] 3391
```

```r
nrow(bank_processed_test)   # 1130
```

```
## [1] 1130
```

Paso 3: Entrenamiento del modelo

Vamos a comparar dos kernels de la librería kernlab.

```r
#train the model vanilladot
model_vanilladot <- ksvm(y ~ ., data=bank_processed_train, kernel="vanilladot")
```

```
##  Setting default kernel parameters
```

```r
model_vanilladot
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 983
##
## Objective Function Value : -718.0508
## Training error : 0.105868
```

```r
#train the model rbfdot
model_rbfdot <- ksvm(y ~ ., data=bank_processed_train, kernel="rbfdot")
model_rbfdot
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.0177605577375513
##
## Number of Support Vectors : 949
##
## Objective Function Value : -672.7457
## Training error : 0.093483
```

El error es ligeramente más pequeño con el kernel rbf ("radial basis function").

[

EXTRA:

Probamos también con la librería LIBSVM.

```r
#train the model e1071
model_e1071 <- svm (y ~ ., data=bank_processed_train, scale = FALSE)  # default values:  kernel = RBF,
model_e1071
```

```
##
## Call:
## svm(formula = y ~ ., data = bank_processed_train, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  952
```

]

Paso 4 – Predicción del modelo

Con la red entrenada podemos realizar predicciones usando el dataset de validación y obteniendo la pertinente matriz de confusión:

```r
#Confusion matrix vanilladot
prediction_vanilladot <- predict(model_vanilladot, bank_processed_test)
confusionMatrix(as.factor(bank_processed_test$y), as.factor(prediction_vanilladot), positive="yes", mod
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  987  13
##        yes 112  18
##
##                Accuracy : 0.8894
##                  95% CI : (0.8696, 0.9071)
##     No Information Rate : 0.9726
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1876
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.58065
##             Specificity : 0.89809
##          Pos Pred Value : 0.13846
##          Neg Pred Value : 0.98700
##               Precision : 0.13846
##                  Recall : 0.58065
##                      F1 : 0.22360
##              Prevalence : 0.02743
##          Detection Rate : 0.01593
```

```
##    Detection Prevalence : 0.11504
##       Balanced Accuracy : 0.73937
##
##        'Positive' Class : yes
##
```

```r
#Confusion matrix rbfdot
prediction_rbfdot <- predict(model_rbfdot, bank_processed_test)
confusionMatrix(as.factor(bank_processed_test$y), as.factor(prediction_rbfdot), positive="yes", mode =
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  988  12
##        yes 110  20
##
##                Accuracy : 0.892
##                  95% CI : (0.8725, 0.9095)
##     No Information Rate : 0.9717
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2111
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.62500
##             Specificity : 0.89982
##          Pos Pred Value : 0.15385
##          Neg Pred Value : 0.98800
##               Precision : 0.15385
##                  Recall : 0.62500
##                      F1 : 0.24691
##              Prevalence : 0.02832
##          Detection Rate : 0.01770
##    Detection Prevalence : 0.11504
##       Balanced Accuracy : 0.76241
##
##        'Positive' Class : yes
##
```

```r
#Confusion matrix e1071
prediction_e1071 <- predict(model_e1071, bank_processed_test)
confusionMatrix(as.factor(bank_processed_test$y), as.factor(prediction_e1071), positive="yes", mode = "
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  1000    0
##        yes  129    1
##
##                Accuracy : 0.8858
```

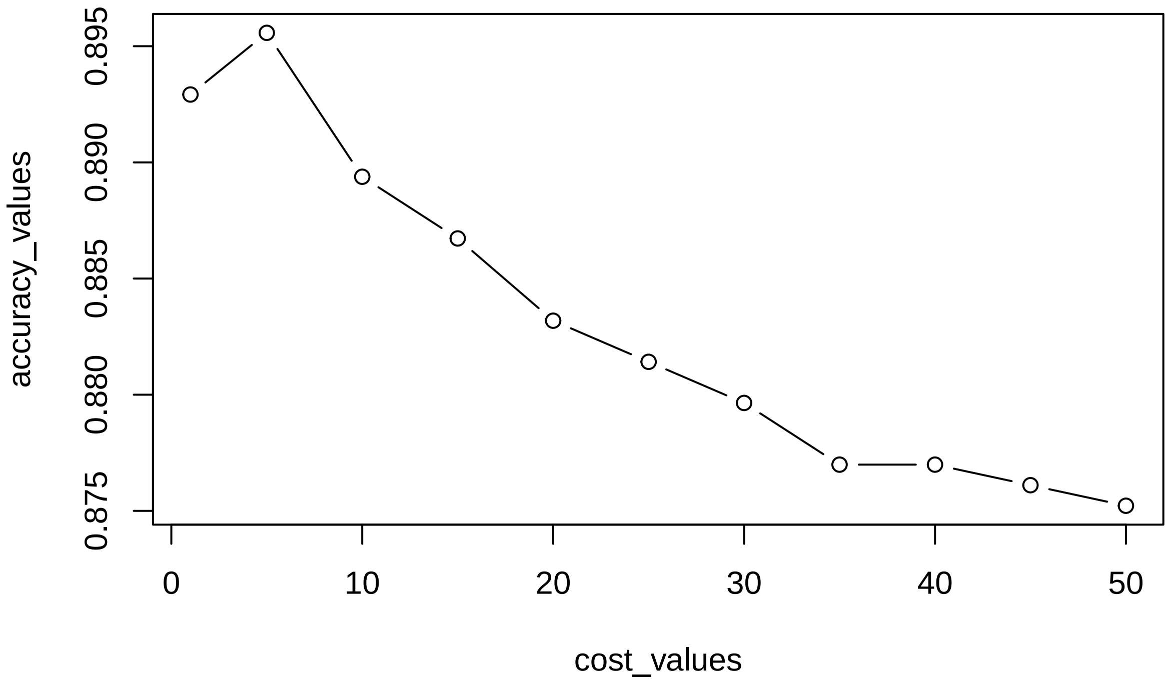
```
##                  95% CI : (0.8658, 0.9038)
##     No Information Rate : 0.9991
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0135
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.000000
##             Specificity : 0.885740
##          Pos Pred Value : 0.007692
##          Neg Pred Value : 1.000000
##               Precision : 0.007692
##                  Recall : 1.000000
##                      F1 : 0.015267
##              Prevalence : 0.000885
##          Detection Rate : 0.000885
##    Detection Prevalence : 0.115044
##       Balanced Accuracy : 0.942870
##
##        'Positive' Class : yes
##
```

Con los valores por defecto, se obtiene la misma exactitud .. pero con muchos falsos positivos (El modelo predice 129 positivos que en realidad son negativos. Solo predice correctamente 1 de los 130!)

Paso 5 – Mejora del modelo

Como se explica en el libro, vamos a intentar averiguar si con algún valor del parámetro coste (parámetro C en la función ksvm), se puede obtener una exactitud mejor que con el valor por defecto (C=1):

```r
set.seed(12345)

cost_values <- c(1, seq(from = 5, to = 50, by = 5))

accuracy_values <- sapply(cost_values, function(x) {

  m <-ksvm (y ~ ., data=bank_processed_train, kernel="rbfdot", C = x)
  pred <- predict(m, bank_processed_test)

  agree <- ifelse(pred == bank_processed_test$y, 1, 0)
  accuracy <- sum(agree) / nrow(bank_processed_test)

  return (accuracy)
})

plot(cost_values, accuracy_values, type = "b")
```



El mejor resultado se obtiene para C=5.

Examinamos con más detalle los valores alrededor de 5:

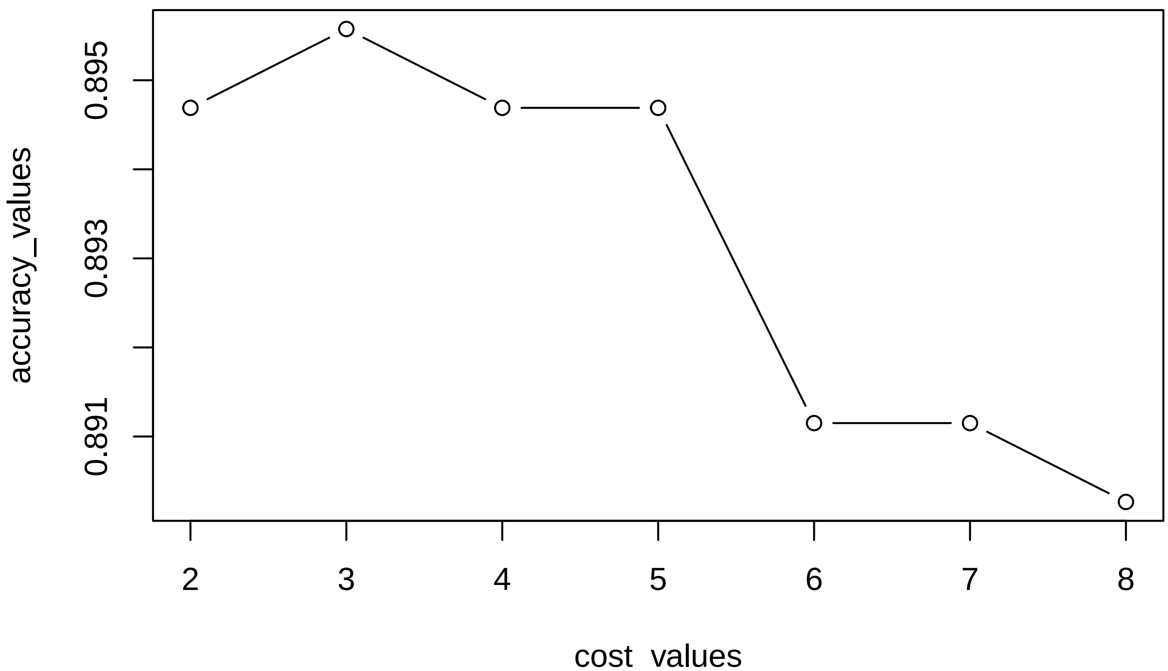
```
set.seed(12345)

cost_values <- c(seq(from = 2, to = 8, by = 1))

accuracy_values <- sapply(cost_values, function(x) {

  m <- ksvm(y ~ ., data=bank_processed_train, kernel="rbfdot", C = x)
  pred <- predict(m, bank_processed_test)

  agree <- ifelse(pred == bank_processed_test$y, 1, 0)
  accuracy <- sum(agree) / nrow(bank_processed_test)

  print(sprintf("C: %f - acc: %f", x, accuracy))

  return (accuracy)
})
```

```
## [1] "C: 2.000000 - acc: 0.894690"
## [1] "C: 3.000000 - acc: 0.895575"
## [1] "C: 4.000000 - acc: 0.894690"
## [1] "C: 5.000000 - acc: 0.894690"
## [1] "C: 6.000000 - acc: 0.891150"
```

```
## [1] "C: 7.000000 - acc: 0.891150"
## [1] "C: 8.000000 - acc: 0.890265"
```

La gráfica:

```
plot(cost_values, accuracy_values, type = "b")
```



Aunque por muy poco, el mejor valor de la exactitud se da para C=3. Entonces entrenamos el modelo y hacemos la predicción con ese valor.

```
#train the model

model_rbfdot_C <- ksvm(y ~ ., data=bank_processed_train, kernel="rbfdot", C = 3)
model_rbfdot_C
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 3
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.0175374530882441
##
## Number of Support Vectors : 959
##
```

```
## Objective Function Value : -1758.389
## Training error : 0.070186
```

C = 5

Support Vector Machine object of class "ksvm"

SV type: C-svc (classification) parameter : cost C = 5

Gaussian Radial Basis kernel function. Hyperparameter : sigma = 0.0182756619058051

Number of Support Vectors : 969

Objective Function Value : -2627.335 Training error : 0.055146

```r
#Confusion matrix
prediction_rbfdot <- predict(model_rbfdot_C, bank_processed_test)
confusionMatrix(as.factor(bank_processed_test$y), as.factor(prediction_rbfdot))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  982  18
##        yes  99  31
##
##                Accuracy : 0.8965
##                  95% CI : (0.8772, 0.9136)
##     No Information Rate : 0.9566
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3024
##
##  Mcnemar's Test P-Value : 1.403e-13
##
##             Sensitivity : 0.9084
##             Specificity : 0.6327
##          Pos Pred Value : 0.9820
##          Neg Pred Value : 0.2385
##              Prevalence : 0.9566
##          Detection Rate : 0.8690
##    Detection Prevalence : 0.8850
##       Balanced Accuracy : 0.7705
##
##        'Positive' Class : no
##
```

C_= 5

Confusion Matrix and Statistics

```
      Reference
```

Prediction no yes no 977 23 yes 98 32

```
           Accuracy : 0.8929
```

Al igual que con las redes de neuronas, parece como si hubiera un muro en el 88-89%, y no consigo pasar de esta exactitud.

[

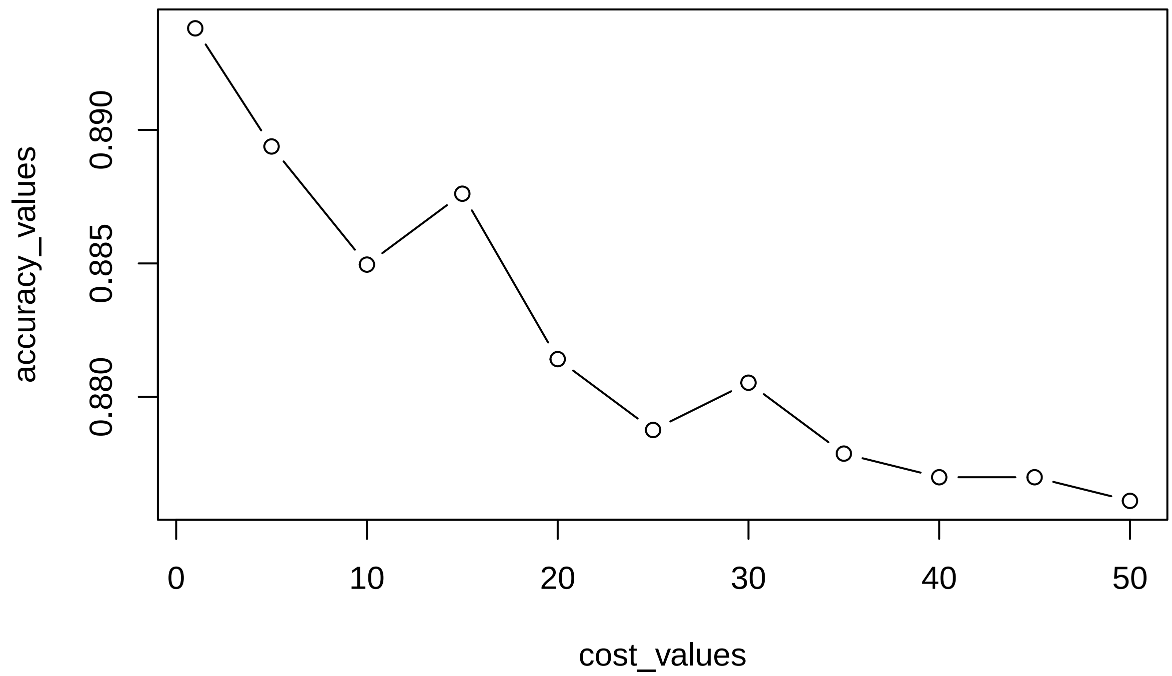
EXTRA:

Intentamos mejorar también el resultado del modelo **e10701** variando el parámetro coste:

```
set.seed(12345)

cost_values <- c(1, seq(from = 5, to = 50, by = 5))

accuracy_values <- sapply(cost_values, function(x) {

  m <- svm(y ~ ., data=bank_processed_train, cost = x, cross = 5)
  pred <- predict(m, bank_processed_test)

  agree <- ifelse(pred == bank_processed_test$y, 1, 0)
  accuracy <- sum(agree) / nrow(bank_processed_test)

  print(sprintf("C: %f - acc: %f", x, accuracy))

  return (accuracy)
})
```

```
## [1] "C: 1.000000 - acc: 0.893805"
## [1] "C: 5.000000 - acc: 0.889381"
## [1] "C: 10.000000 - acc: 0.884956"
## [1] "C: 15.000000 - acc: 0.887611"
## [1] "C: 20.000000 - acc: 0.881416"
## [1] "C: 25.000000 - acc: 0.878761"
## [1] "C: 30.000000 - acc: 0.880531"
## [1] "C: 35.000000 - acc: 0.877876"
## [1] "C: 40.000000 - acc: 0.876991"
## [1] "C: 45.000000 - acc: 0.876991"
## [1] "C: 50.000000 - acc: 0.876106"
```

```
plot(cost_values, accuracy_values, type = "b")
```



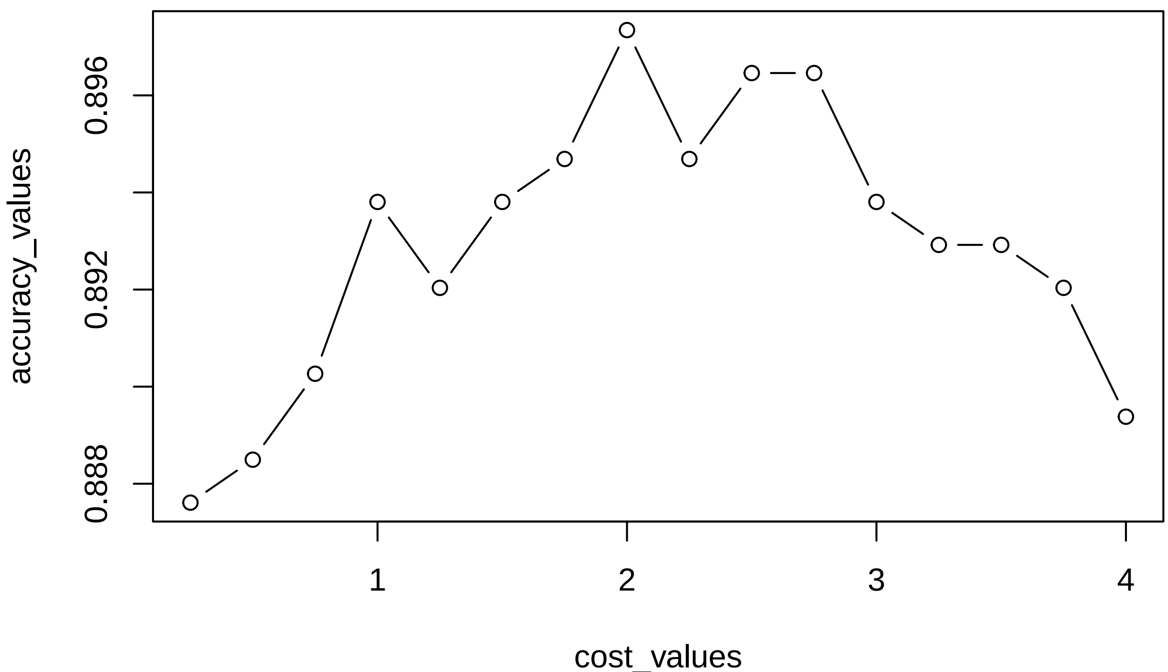
El mejor resultado parece ser con cost=1 …

```
set.seed(12345)

cost_values <- c(seq(from = 0.25, to = 4, by = 0.25))

accuracy_values <- sapply(cost_values, function(x) {

  m <- svm(y ~ ., data=bank_processed_train, cost = x)
  pred <- predict(m, bank_processed_test)

  agree <- ifelse(pred == bank_processed_test$y, 1, 0)
  accuracy <- sum(agree) / nrow(bank_processed_test)

  print(sprintf("C: %f - acc: %f", x, accuracy))

  return (accuracy)
})
```

```
## [1] "C: 0.250000 - acc: 0.887611"
## [1] "C: 0.500000 - acc: 0.888496"
## [1] "C: 0.750000 - acc: 0.890265"
## [1] "C: 1.000000 - acc: 0.893805"
## [1] "C: 1.250000 - acc: 0.892035"
## [1] "C: 1.500000 - acc: 0.893805"
## [1] "C: 1.750000 - acc: 0.894690"
```

```
## [1] "C: 2.000000 - acc: 0.897345"
## [1] "C: 2.250000 - acc: 0.894690"
## [1] "C: 2.500000 - acc: 0.896460"
## [1] "C: 2.750000 - acc: 0.896460"
## [1] "C: 3.000000 - acc: 0.893805"
## [1] "C: 3.250000 - acc: 0.892920"
## [1] "C: 3.500000 - acc: 0.892920"
## [1] "C: 3.750000 - acc: 0.892035"
## [1] "C: 4.000000 - acc: 0.889381"
```

```r
plot(cost_values, accuracy_values, type = "b")
```



Pues es mejor con 2.

```r
#train the model e1071
model_e1071_C2 <- svm(y ~ ., data=bank_processed_train, cost = 2 , scale = FALSE)
model_e1071_C2
```

```
##
## Call:
## svm(formula = y ~ ., data = bank_processed_train, cost = 2, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
```

```
##  SVM-Kernel:  radial
##        cost:  2
##
## Number of Support Vectors:  961
```

```r
#Confusion matrix e1071
prediction_e1071 <- predict(model_e1071_C2, bank_processed_test)
confusionMatrix(as.factor(bank_processed_test$y), as.factor(prediction_e1071), positive="yes", mode = "
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  996   4
##        yes 126   4
##
##                Accuracy : 0.885
##                  95% CI : (0.8649, 0.903)
##     No Information Rate : 0.9929
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0452
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.50000
##             Specificity : 0.88770
##          Pos Pred Value : 0.03077
##          Neg Pred Value : 0.99600
##               Precision : 0.03077
##                  Recall : 0.50000
##                      F1 : 0.05797
##              Prevalence : 0.00708
##          Detection Rate : 0.00354
##    Detection Prevalence : 0.11504
##       Balanced Accuracy : 0.69385
##
##        'Positive' Class : yes
##
```

]

Mejorar el modelo con "grid search".

Después de haber visto cómo se intenta mejorar un modelo en Python con la función GridSearchCV de la librería scikit-learn, he buscado cómo hacerlo de manera equivalente en R, para así probar no solo con valores del parámetro coste, sino también del parámetro gamma*. Voy a utilizar la librería caret (también se puede hacer con la librería e1071):

Probamos con los mismos valores que en Python

*lo que en Python se llama gamma, en R se llama sigma … ?!

```r
#gridsearch with caret
set.seed(12345)

# 5-fold cross-validation
control <- trainControl(method = "cv", number = 5)

#https://topepo.github.io/caret/available-models.html
grid <- expand.grid(C = c(1,3,10,100,1000),
                    sigma = c(0.1, 0.01, 0.001, 0.0001))

# https://topepo.github.io/caret/train-models-by-tag.html#Support_Vector_Machines
svmGridSearch <- train(y ~ ., data = bank_processed_train, method = "svmRadial",
                       trControl = control,
                       tuneGrid = grid)
```

Mostrar el contenido del "grid search":

```r
svmGridSearch
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 3391 samples
##   40 predictor
##    2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 2713, 2713, 2712, 2713, 2713
## Resampling results across tuning parameters:
##
##   C     sigma  Accuracy   Kappa
##      1  1e-04  0.8846951  0.0000000
##      1  1e-03  0.8941303  0.2292384
##      1  1e-02  0.8944253  0.2368032
##      1  1e-01  0.8897077  0.1235625
##      3  1e-04  0.8846951  0.0000000
##      3  1e-03  0.8941303  0.2292384
##      3  1e-02  0.8938362  0.2674598
##      3  1e-01  0.8817457  0.2050404
##     10  1e-04  0.8941303  0.2292384
##     10  1e-03  0.8941303  0.2292384
##     10  1e-02  0.8938362  0.3244388
##     10  1e-01  0.8734909  0.2158880
##    100  1e-04  0.8941303  0.2292384
##    100  1e-03  0.8964906  0.2992693
##    100  1e-02  0.8841012  0.3481747
##    100  1e-01  0.8693593  0.2202765
##   1000  1e-04  0.8941303  0.2292384
##   1000  1e-03  0.8911839  0.3236708
##   1000  1e-02  0.8587429  0.2965707
##   1000  1e-01  0.8675903  0.2141514
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.001 and C = 100.
```

C sigma Accuracy 100 1e-03 0.8964906 -> 89.65%