SVM

Laura Belmonte

2025-12-08

```
library(recipes)
library(e1071)
library(mlbench)
library(ggplot2)
library(ISLR)
library(caret)
library(pROC)
library(dplyr)
load("~/Documents/GitHub/Mineria/DATA/dataaaaaaaaaaaaaaa.RData")
bd<-data_reducida
```

Dumificar

```
set.seed(123)

rec <- recipe(Exited ~ ., data = bd) %>%
  step_dummy(all_nominal_predictors(), -group, one_hot = TRUE, keep_original_cols = FALSE)

# Prepara y aplica a un NUEVO objeto
bd_procesado <- prep(rec) %>% bake(new_data = NULL)
bd<-bd_procesado
#summary(bd) se ha dummificado correctamente
```

```
#Dividir la base de datos
trainbase<-bd[bd$group=="train",]
trainbase$group<-NULL
testbase<-bd[bd$group=="test",]
testbase$group<-NULL
ind <- sample(1:nrow(trainbase), 0.7*nrow(trainbase))
train <- trainbase[ind,]
test <- trainbase[-ind,]
```

Mejor kernel

Comenzamos con los diferentes kernels posibles dejando fijo los valores expuestos por la función.

```r
threshold <- 0.20
evaluar_kernel <- function(
    kernel_name,
    cost_val = 10,
    gamma_val = 0.1,
    degree_val = 3,
    coef0_val = 0.0,
    train_data,
    test_data
) {
    svm.model <- svm(
        Exited ~ .,
        data = train_data,
        cost = cost_val,
        kernel = kernel_name,
        gamma = gamma_val,
        degree = degree_val,
        coef0 = coef0_val,
        probability = TRUE
    )
    svm.pred_prob_matrix <- predict(
        svm.model,
        test_data,
        probability = TRUE
    )
    svm.pred_prob <- attr(svm.pred_prob_matrix, "probabilities")[, "1"]
    svm.pred_class <- ifelse(svm.pred_prob >= threshold, "1", "0")

    cm <- confusionMatrix(
        as.factor(svm.pred_class),
        as.factor(test_data$Exited),
        positive = "1"
    )

    roc_obj <- roc(test_data$Exited, svm.pred_prob, quiet = TRUE)
    auc_val <- auc(roc_obj)

    return(data.frame(
        Kernel = kernel_name,
        Accuracy = cm$overall["Accuracy"],
        Precision = cm$byClass["Precision"],
        Recall = cm$byClass["Recall"],
        Specificity = cm$byClass["Specificity"],
        F1_Score = cm$byClass["F1"],
        AUC = auc_val
    ))
}

kernels_a_probar <- c("linear", "polynomial", "radial", "sigmoid")
cost_val <- 10
gamma_val <- 0.1
degree_val <- 3
coef0_val <- 0.0
```

```
resultados_kernels <- lapply(kernels_a_probar, function(k) {
    evaluar_kernel(
        kernel_name = k,
        cost_val = cost_val,
        gamma_val = gamma_val,
        degree_val = degree_val,
        coef0_val = coef0_val,
        train_data = train,
        test_data = test
    )
})
resultados_finales <- do.call(rbind, resultados_kernels) %>%
    mutate(across(where(is.numeric), ~ round(., 4))) %>%
    select(Kernel, F1_Score, Recall, Precision, Accuracy, Specificity, AUC) %>%
    arrange(desc(F1_Score))
print(resultados_finales)
```

```
##               Kernel F1_Score Recall Precision Accuracy Specificity    AUC
## Accuracy1 polynomial   0.4085 0.3649    0.4639   0.7876      0.8939 0.6934
## Accuracy3    sigmoid   0.3582 0.5640    0.2624   0.5938      0.6013 0.6187
## Accuracy2     radial   0.3535 0.2773    0.4875   0.7962      0.9267 0.6276
## Accuracy      linear   0.1526 0.0877    0.5873   0.8043      0.9845 0.6195
```

Los resultados muestran que el kernel polynomial ofrece el mejor rendimiento general con un F1-Score de 0.4085 y un AUC de 0.6934, logrando el mejor equilibrio entre, precisión (0.4639) y recall (0.3649).

El kernel sigmoid, aunque alcanza el mayor recall (0.5640), presenta una precisión muy baja (0.2624), indicando un exceso de falsos positivos. Los kernels radial y linear muestran comportamiento excesivamente conservador con specificities superiores a 0.92, detectando muy pocos casos positivos.

En conclusión, el kernel polynomial es la opción más adecuada para este problema de clasificación desbalanceada.

Reducida plus

```
bd<-data_reducida_plus
```

Dumificar

```
set.seed(123)

rec <- recipe(Exited ~ ., data = bd) %>%
  step_dummy(all_nominal_predictors(), -group, one_hot = TRUE, keep_original_cols = FALSE)

# Prepara y aplica a un NUEVO objeto
bd_procesado_plus <- prep(rec) %>% bake(new_data = NULL)
bd<-bd_procesado_plus
#summary(bd) se ha dummificado correctamente
```

```r
#Dividir la base de datos
trainbase<-bd[bd$group=="train",]
trainbase$group<-NULL
testbase<-bd[bd$group=="test",]
testbase$group<-NULL
ind <- sample(1:nrow(trainbase), 0.7*nrow(trainbase))
train <- trainbase[ind,]
test <- trainbase[-ind,]
```

Mejor kernel

Comenzamos con los diferentes kernels posibles dejando fijo los valores expuestos por la función.

```r
threshold <- 0.20
evaluar_kernel <- function(
    kernel_name,
    cost_val = 10,
    gamma_val = 0.1,
    degree_val = 3,
    coef0_val = 0.0,
    train_data,
    test_data
) {
    svm.model <- svm(
        Exited ~ .,
        data = train_data,
        cost = cost_val,
        kernel = kernel_name,
        gamma = gamma_val,
        degree = degree_val,
        coef0 = coef0_val,
        probability = TRUE
    )
    svm.pred_prob_matrix <- predict(
        svm.model,
        test_data,
        probability = TRUE
    )
    svm.pred_prob <- attr(svm.pred_prob_matrix, "probabilities")[, "1"]
    svm.pred_class <- ifelse(svm.pred_prob >= threshold, "1", "0")

    cm <- confusionMatrix(
        as.factor(svm.pred_class),
        as.factor(test_data$Exited),
        positive = "1"
    )

    roc_obj <- roc(test_data$Exited, svm.pred_prob, quiet = TRUE)
    auc_val <- auc(roc_obj)

    return(data.frame(
        Kernel = kernel_name,
```

```
        Accuracy = cm$overall["Accuracy"],
        Precision = cm$byClass["Precision"],
        Recall = cm$byClass["Recall"],
        Specificity = cm$byClass["Specificity"],
        F1_Score = cm$byClass["F1"],
        AUC = auc_val
    ))
}

kernels_a_probar <- c("linear", "polynomial", "radial", "sigmoid")
cost_val <- 10
gamma_val <- 0.1
degree_val <- 3
coef0_val <- 0.0

resultados_kernels <- lapply(kernels_a_probar, function(k) {
    evaluar_kernel(
        kernel_name = k,
        cost_val = cost_val,
        gamma_val = gamma_val,
        degree_val = degree_val,
        coef0_val = coef0_val,
        train_data = train,
        test_data = test
    )
})
resultados_finales <- do.call(rbind, resultados_kernels) %>%
    mutate(across(where(is.numeric), ~ round(., 4))) %>%
    select(Kernel, F1_Score, Recall, Precision, Accuracy, Specificity, AUC) %>%
    arrange(desc(F1_Score))
print(resultados_finales)
```

```
##               Kernel F1_Score Recall Precision Accuracy Specificity    AUC
## Accuracy1 polynomial   0.4227 0.4763    0.3800   0.7386      0.8045 0.6894
## Accuracy2     radial   0.4067 0.4005    0.4132   0.7652      0.8570 0.6442
## Accuracy3    sigmoid   0.3081 0.5166    0.2195   0.5338      0.5381 0.5570
## Accuracy      linear   0.1526 0.0877    0.5873   0.8043      0.9845 0.5177
```

#Con la base de datos FAMD

```
# Cargar librerías necesarias
library(FactoMineR)
library(e1071)
library(caret)
library(pROC)
library(dplyr)

# Dividir por group
trainbase <- bd[bd$group == "train",]
trainbase$group <- NULL
testbase <- bd[bd$group == "test",]
testbase$group <- NULL
```

```r
# Subdividir trainbase en train (70%) y test (30%)
set.seed(123)
ind <- sample(1:nrow(trainbase), 0.7*nrow(trainbase))
train_original <- trainbase[ind,]
test_original <- trainbase[-ind,]

# Variables a excluir del PCA
cols_excluir <- c("Exited")

# Aplicar PCA en train (todas las variables son numéricas)
pca_model <- PCA(train_original[, !names(train_original) %in% cols_excluir],
                 ncp = 9,
                 scale.unit = TRUE,
                 graph = FALSE)

# Transformar train a dimensiones PCA
train <- as.data.frame(pca_model$ind$coord)
train$Exited <- train_original$Exited

# Transformar test usando el mismo modelo PCA
test_coords <- predict(pca_model,
                       newdata = test_original[, !names(test_original) %in% cols_excluir])
test <- as.data.frame(test_coords$coord)
test$Exited <- test_original$Exited

# Verificar
cat("Columnas en train:", names(train), "\n")
```

```
## Columnas en train: Dim.1 Dim.2 Dim.3 Dim.4 Dim.5 Dim.6 Dim.7 Dim.8 Dim.9 Exited
```

```r
# Función para evaluar kernels
threshold <- 0.20

evaluar_kernel <- function(
    kernel_name,
    cost_val = 10,
    gamma_val = 0.1,
    degree_val = 3,
    coef0_val = 0.0,
    train_data,
    test_data
) {
    svm.model <- svm(
        Exited ~ .,
        data = train_data,
        cost = cost_val,
        kernel = kernel_name,
        gamma = gamma_val,
        degree = degree_val,
        coef0 = coef0_val,
        probability = TRUE
    )
    svm.pred_prob_matrix <- predict(
```

```
        svm.model,
        test_data,
        probability = TRUE
    )
    svm.pred_prob <- attr(svm.pred_prob_matrix, "probabilities")[, "1"]
    svm.pred_class <- ifelse(svm.pred_prob >= threshold, "1", "0")
    cm <- confusionMatrix(
        as.factor(svm.pred_class),
        as.factor(test_data$Exited),
        positive = "1"
    )
    roc_obj <- roc(test_data$Exited, svm.pred_prob, quiet = TRUE)
    auc_val <- auc(roc_obj)
    return(data.frame(
        Kernel = kernel_name,
        Accuracy = cm$overall["Accuracy"],
        Precision = cm$byClass["Precision"],
        Recall = cm$byClass["Recall"],
        Specificity = cm$byClass["Specificity"],
        F1_Score = cm$byClass["F1"],
        AUC = auc_val
    ))
}

# Evaluar todos los kernels
kernels_a_probar <- c("linear", "polynomial", "radial", "sigmoid")

resultados_kernels <- lapply(kernels_a_probar, function(k) {
    evaluar_kernel(
        kernel_name = k,
        cost_val = 10,
        gamma_val = 0.1,
        degree_val = 3,
        coef0_val = 0.0,
        train_data = train,
        test_data = test
    )
})

resultados_finales <- do.call(rbind, resultados_kernels) %>%
    mutate(across(where(is.numeric), ~ round(., 4))) %>%
    select(Kernel, F1_Score, Recall, Precision, Accuracy, Specificity, AUC) %>%
    arrange(desc(F1_Score))

print(resultados_finales)
```

```
##               Kernel F1_Score Recall Precision Accuracy Specificity    AUC
## Accuracy2     radial   0.4322 0.3876    0.4884   0.7886      0.8936 0.6549
## Accuracy1 polynomial   0.4244 0.4312    0.4178   0.7571      0.8425 0.6944
## Accuracy3    sigmoid   0.3452 0.4885    0.2669   0.6152      0.6484 0.5704
## Accuracy      linear   0.2230 0.1445    0.4884   0.7910      0.9603 0.6581
```