UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Tecnura

https://revistas.udistrital.edu.co/ojs/index.php/Tecnura/issue/view/1215

DOI: https://doi.org/10.14483/22487638.20326

INVESTIGACIÓN

Implementación del método de optimización de máquinas de soporte vectorial en *R*

Implementation of the Support Vector Machine Optimization Method in R

Diego Armando Poveda García [1], Martha Elva Ramírez Guzmán [2], Yolanda Margarita Fernández Ordóñez [3] y Erwin Fabián García López [4]

Fecha de Recepción: 8 de enero de 2023 **Fecha de Aceptación**: 16 de octubre de 2024

Cómo citar: Poveda García, D. A., Ramírez Guzmán, M. E., Fernández Ordóñez, Y. M. y García López, E. F. (2024). Implementación del método de optimización de máquinas de soporte vectorial en R. *Tecnura*, 28(80), 99-118. https://doi.org/10.14483/22487638.20326

Resumen

Contexto: La optimización de máquinas de soporte vectorial (SVM, por su sigla en inglés) ha sido un tema recurrente en diversos foros y publicaciones relacionadas con la inteligencia artificial (IA) y el aprendizaje automático. A pesar de su relevancia, existe una notable ausencia de directrices claras para la selección y optimización de los hiperparámetros asociados a las distintas funciones kernel utilizadas en SVM, lo cual puede comprometer la eficacia de estas herramientas en tareas de clasificación y regresión.

Objetivo: Este estudio tiene como objetivo general presentar un método detallado y riguroso para la optimización de los hiperparámetros en SVM, mediante el uso de la librería e1071 en *R*. Se busca, así, proporcionar una guía clara para futuros investigadores y practicantes en el campo, que les permita implementar de manera efectiva SVM en sus proyectos.

Metodología: Se emplearon las librerías e1071 y scales en R para estandarizar los datos y ajustar los modelos SVM con diferentes tipos de kernel (lineal, radial, sigmoide y polinomial). Se procedió a la optimización de hiperparámetros mediante la función "tune", con análisis de conjuntos de entrenamiento y prueba para verificar la bondad de ajuste. Se utilizó una metodología detallada para seleccionar los hiperparámetros óptimos de cada kernel, a partir del estudio de la eficacia de la clasificación y la minimización del error.

Resultados: Los resultados destacan la correcta selección y optimización de los hiperparámetros en SVM, y demuestran mejoras significativas en la clasificación al aplicar el método propuesto. Se encontró que la elección de

[1]Economista. M. Sc. Estadística. Científico de datos corporativo. Email: dapovedag@unal.edu.co

[2]Licenciada en Actuaría, M. Sc. Estadística. Ph. D. Statistics. Profesora investigadora titular. Email: martharg@colpos.mx

[3]Licenciada en Matemáticas. M. Sc. Computación. Ph. D. Computación. Profesora investigadora titular. Email: yfernand@colpos.mx

[4]Administrador de Empresas. Especialista en Administración Financiera. Magíster en Educación. Profesor investigador ocasional. Email: efgarcial@unal.edu.co

hiperparámetros óptimos varía considerablemente entre los diferentes tipos de kernel y que su adecuada optimización contribuye a la precisión del modelo.
Conclusiones: La implementación detallada y el ajuste de los hiperparámetros en SVM son cruciales para maximizar su desempeño. Este artículo proporciona un código optimizado y una metodología clara para su implementación en R, lo cual facilita la tarea de investigadores y analistas en el campo del aprendizaje automático. Se enfatiza la necesidad de considerar la especificidad de cada conjunto de datos y la relevancia de la experiencia en la selección de hiperparámetros. También, se sugiere, para futuras investigaciones, explorar y expandir las fronteras de las SVM en la solución de problemas complejos de clasificación.

Palabras clave: máquina de soporte vectorial, optimización, *R*, librería e1070, aprendizaje supervisado.

Abstract

Context: The optimization of Support Vector Machines (SVM) has been a recurring theme in various forums and publications related to artificial intelligence and machine learning. Despite its relevance, there is a notable absence of clear guidelines for the selection and optimization of parameters associated with the different kernel functions used in SVMs, which can compromise the effectiveness of these tools in classification and regression tasks. **Objective:** The overall objective of this study is to present a detailed and rigorous method for the optimization of parameters in SVMs, using the e1071 library in R for this purpose. Thus, it seeks to provide a clear guide for future researchers and practitioners in the field, enabling them to effectively implement SVMs in their projects.
Methodoly: The e1071 and scales libraries in R were used to standardize the data and adjust the SVM models with different types of kernels (linear, radial, sigmoid, and polynomial). Parameter optimization was carried out using the tune function, analyzing training and test sets to verify the goodness of fit. A detailed methodology was used to select the optimal parameters for each kernel, based on the analysis of classification efficacy and error minimization.
Results: The results highlight the importance of correct parameter selection and optimization in SVMs, showing significant improvements in classification when applying the proposed method. It was found that the choice of optimal parameters varies considerably among the different types of kernels, and their proper optimization contributes to the precision of the model.
Conclusions: Detailed implementation and parameter adjustment in SVMs are crucial to maximizing their performance. This article contributes to the existing literature by providing optimized code and a clear methodology for its implementation in R, facilitating the work of researchers and analysts in the field of machine learning. It emphasizes the need to consider the specificity of each data set and the relevance of experience in parameter selection, inviting future research to explore and expand the boundaries of SVMs in solving complex classification problems.

Keywords: Support Vector Machine, Optimization, R, e1071 library, Supervised Learning.

Introducción

Aprendizaje supervisado

Para entender el aprendizaje supervisado usaremos un ejemplo. Suponga que usted está con un grupo de amigos y juega a adivinar una película con base en mímicas. Usted ve que su equipo ejecuta una serie de movimientos que deberían darle alguna idea de la película que

debe adivinar. Con esa serie de movimientos, usted hace un escrutinio basado en las películas que conoce; aunque la interpretación tiene un grado de incidencia alto, usted podrá acertar con la película, si ya la ha visto, debido a que los movimientos le serán familiares. Básicamente, usted no puede acertar el nombre de la película si no la conoce. Esto es aprendizaje supervisado, "porque el entrenamiento se realizó a partir de datos conocidos o *inputs*, los cuales están etiquetados [...] con la finalidad de obtener un resultado [...] que también es conocido y etiquetado" (Paredes, 2020, p. 10). En esencia, en el aprendizaje supervisado se debe saber cuál es el resultado para determinar las diferencias presentes, dada una clasificación previa.

Entre los métodos de aprendizaje supervisado se pueden considerar las máquinas de soporte vectorial (SVM, por su sigla en inglés).

Máquinas de soporte vectorial

La inteligencia artificial (IA) depende en gran medida de la forma en que se diferencian patrones. Las SVM utilizan las funciones kernel para segmentar los datos y brindar información sobre datos desconocidos, a partir de la información provista por datos conocidos (Méndez, 2003). Un kernel es una "función definida positiva que permite la transformación de los datos" (Rodríguez *et al.*, 2009, p. 172) y se utiliza para transformar datos no lineales y reducir su dimensionalidad para crear una función de separación.

En el contexto de este artículo, interesan los resultados de clasificación binaria (pero no se debe entender que la clasificación binaria es la única posible dentro de las SVM) para los cuales, según Méndez (2003), se deben considerar los siguientes conjuntos de datos:

$$X \subset \mathbb{R}^n, \textit{siendo } X \textit{ el conjunto de observaciones de las variables de entrada}$$

$$Y = \{-1, 1\},\ Y \textit{ un conjunto que contiene la variable salida que toma valores binarios}$$

$$S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_i, y_i)\},\ \textit{siendo } S \textit{ el conjunto de entrenamiento, con } x_i, y_i \subset X, Y$$

Sobre esta última parte se debe resaltar que el tamaño de observaciones del conjunto entrenamiento es estrictamente inferior al conjunto de observaciones presentes en los conjuntos de entrada X y de salida Y. Esto, debido a que es necesario comprobar el grado de precisión del algoritmo con datos que no están incluidos en el aprendizaje para probar la bondad de ajuste, es decir, el porcentaje de error y acierto (Amat, 2017). Este ejercicio se verifica con una partición de los datos denominado *conjunto de prueba*, que indica el acierto basado en una muestra no involucrada en el entrenamiento.

En principio, podemos suponer que la segmentación de los datos es posible a partir de una partición lineal, es decir, una línea recta que divida los datos. En este caso, dado un conjunto

de observaciones X, se define una clasificación Y, y se elige un conjunto S de observaciones y respuesta para entrenar el algoritmo (figura **1**).

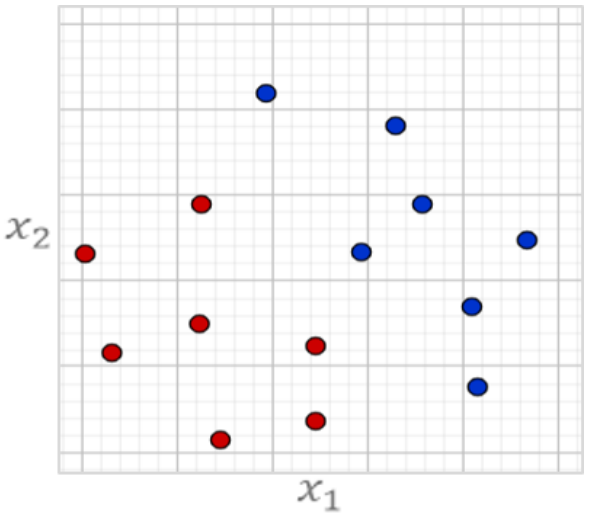

Figura 1. Ordenación de entrenamiento datos clasificados

Nota: elaborada a partir del uso de Geogebra.org.

En este sentido, los datos pueden ser divididos con una recta (figura **2**). No obstante, surge el interrogante: ¿cuál de las posibles rectas divisorias es la más adecuada para el conjunto de datos? La respuesta involucra el grado de error determinado y se refiere a la distancia máxima que puede ocurrir entre dos puntos de clasificación diferente sin que se reduzca el acierto de la separación.

Así, el algoritmo de clasificación considera un hiperplano $f(x) =< w, x > +b$ (función de una recta) que clasifica correctamente los datos. En este caso, se considera un error la clasificación equivocada de una observación, con lo que la minimización del error sucede cuando se presenta el mayor porcentaje de acierto. Para minimizar el error de clasificación, el algoritmo aplica el método de multiplicadores de Lagrange, en combinación con las condiciones complementarias de Karush-Kuhn-Tucker (KKT). Este proceso asegura que se identifique el hiperplano óptimo que maximiza la distancia entre las clases, y así reduce la clasificación incorrecta de las observaciones. Aunque la demostración y el desarrollo matemático completo de este método exceden el alcance de este artículo, puede ser consultado en obras de referencia como las de Méndez (2003), Betancourt (2005) y Campo (2016). El nombre de soporte vectorial se da debido a que las rectas deben pasar por, al menos, un punto que análogamente será una barrera a partir de la cual se definirá la regla de clasificación para los puntos que contenga dicha barrera; así, la frontera se convierte en un vector que contiene o soporta los datos (figura **3**).

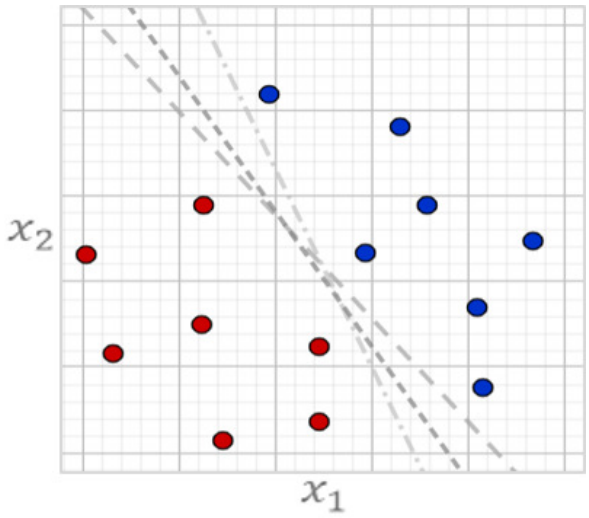

Figura 2. Segmentación de datos clasificados

Nota: elaborada a partir del uso de Geogebra.org.

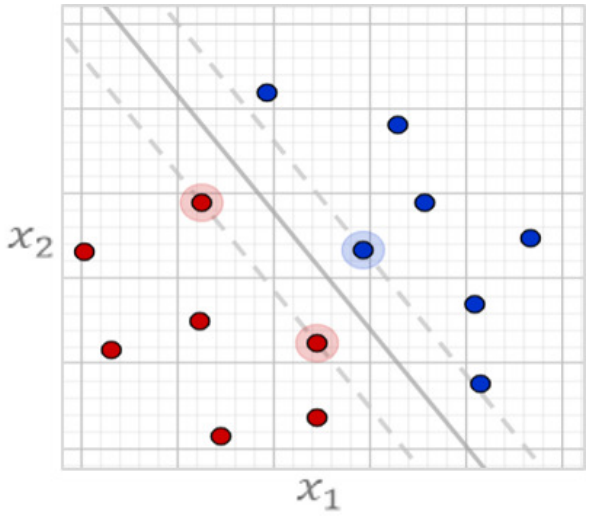

Figura 3. Vectores soporte

Nota: elaborada a partir del uso de Geogebra.org.

El objetivo de las máquinas de soporte vectorial es crear una zona de clasificación en la que cualquier dato nuevo, pueda ser clasificado (figura **4**).

Lo anterior ocurre cuando se tiene un kernel tipo lineal, pero también se pueden tener kernel tipo radial (figura **5**), sigmoide (figura **6**) y polinomial (figura **7**), estos responden a las siguientes funciones y hiperparámetros (tabla 1).

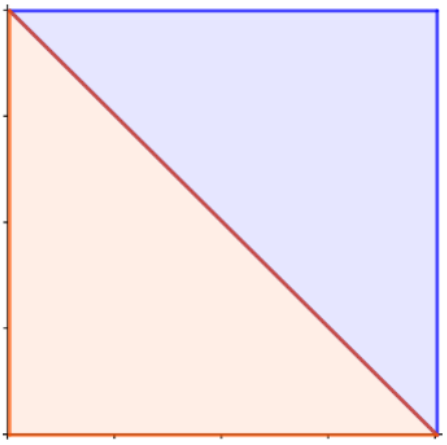

Figura 4. Clasificación por kernel lineal

Nota: elaborada a partir del uso de Geogebra.org.

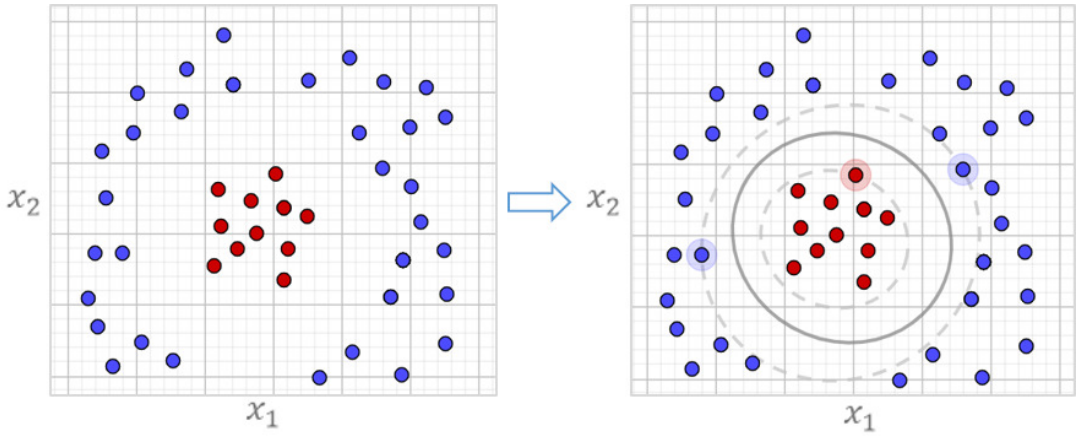

Figura 5. Clasificación por kernel radial

Nota: elaborada a partir del uso de Geogebra.org.

La forma en la que se presenta la función sigmoidal (figura **6**) permite ampliar el entendimiento de un kernel. Este, al ser una transformación, toma un conjunto de datos en R^{n-1} y expresa su ordenamiento en R^n, este encuentra patrones al incorporar una nueva dimensión para la clasificación.

Vale la pena señalar que para manipular correctamente los datos frente a un algoritmo SVM, se recomienda trabajar, en primer lugar, con datos estandarizados.

Por consiguiente, el problema abordado en este artículo se refiere a la ausencia de optimización de la totalidad de hiperparámetros asociados a las funciones de los diferentes kernel (*véase*

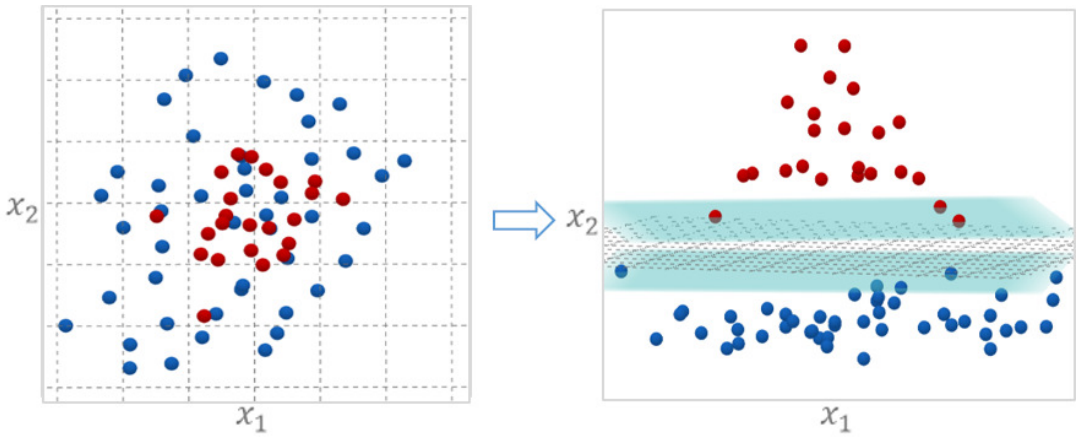

Figura 6. Clasificación por kernel sigmoide* (misma gráfica en 2D y 3D)

*Las funciones sigmoide tienen una forma más asociada a una ese (S); no obstante, su graficación es compleja y su idea principal se expresa la gráfica.
Nota: elaborada a partir del uso de Geogebra.org.

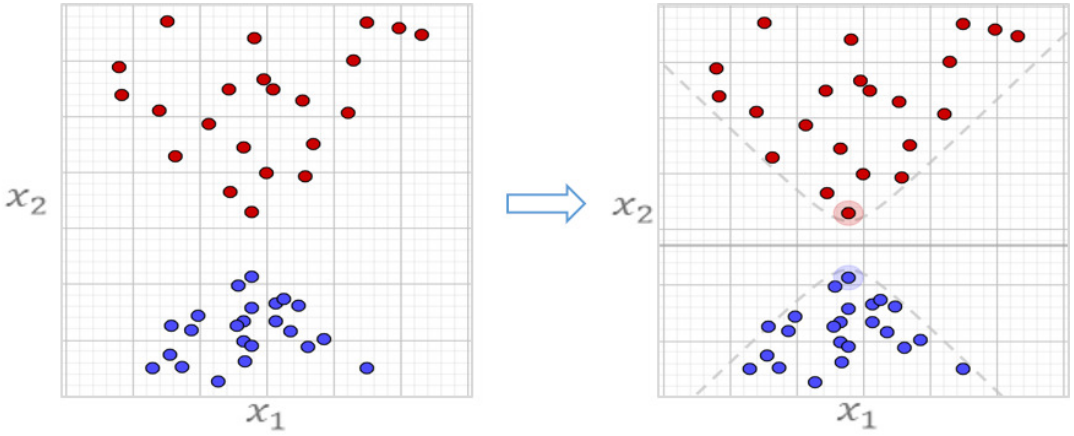

Figura 7. Clasificación por kernel polinomial grado 2

Nota: elaborada a partir del uso de Geogebra.org.

tabla 1). Una función lineal consta de un factor: v, por lo que la optimización debe constar de un conjunto de valores para un parámetro. La función radial consta de dos factores: v, γ, por lo que la optimización debe constar de un conjunto de valores para dos hiperparámetros. La función sigmoidal consta de tres factores: v, γ, τ, por lo que la optimización debe tener de un conjunto de valores para tres hiperparámetros. La función polinomial consta de cuatro factores; v, γ, τ, p, por lo que la optimización debe integrar de un conjunto de valores para cuatro hiperparámetros.

Tabla 1. Kernel usados en máquinas de soporte vectorial

Kernel	**Función**	**Término de regularización**	γ	τ	p
Lineal	$u' \cdot v$	✓			
Radial	$\exp(-\gamma\|u-v\|^2)$	✓	✓		
Sigmoide	$\tanh[\gamma(u' \cdot v)+\tau]$	✓	✓	✓	
Polinomial	$[\gamma(u' \cdot v)+\tau]^p$	✓	✓	✓	✓

Nota: tomada de librería e1071 (Chih *et al.*, 2021).

En las máquinas de soporte vectorial (MSV), los hiperparámetros cumplen un papel clave en la optimización del modelo. A continuación, se detalla la función de cada uno:

- *Término de regularización* (v). Controla la penalización por errores en la clasificación de los puntos. Un valor bajo de v permite más errores, busca un margen más amplio entre las clases (más tolerante a los puntos mal clasificados); mientras que un valor alto busca minimizar los errores, pero a costa de reducir el margen, lo que puede llevar a sobreajuste.

- γ *(gamma)*. Se usa en los kernel radial, sigmoide y polinomial. Define la influencia de un solo punto de entrenamiento, es decir, cuánto afecta un punto a la decisión del modelo. Un valor alto de gamma hace que los puntos cercanos tengan una influencia grande, lo que resulta en un modelo que podría sobreajustarse; mientras que un valor bajo de gamma permite que los puntos más alejados también influyan.

- τ *(tau)*. Se usa en los kernel sigmoide y polinomial. Actúa como un desplazamiento en las funciones kernel, afectando el margen de decisión. Ayuda a controlar la relación entre los vectores de soporte y el resto de los datos.

- *p (grado polinómico)*. Se usa en el kernel polinomial. Representa el grado del polinomio en la función de decisión. Un valor más alto de p genera una frontera de decisión más compleja y no lineal; mientras que un valor bajo, crea una frontera más simple.

No obstante, es recurrente que se presente cualquiera de los siguientes tres escenarios:

- Se omiten las funciones con kernel sigmoidal o polinomial.

- Se carece de un código que minimice el error dado un conjunto de hiperparámetros.

- Se utiliza un conjunto de valores para uno o dos hiperparámetros que no están relacionados directamente con el tipo de kernel utilizado. Por ejemplo, se utiliza un conjunto de valores para dos hiperparámetros en una función polinomial que depende de cuatro hiperparámetros.

Debido a esto, este trabajo presenta un código que permite comprender la forma de optimizar todos los hiperparámetros de una determinada SVM para kernel tipo lineal, radial, sigmoidal y polinomial.

Metodología

Al observar el código que circula en diferentes foros acerca de SVM, es común encontrar tres grupos de código: (a) los que directamente no plantean máquinas de soporte vectorial del tipo polinomial (Amat, 2017; Tinoco, 2019; Sosa, 2021); (b) aquellos que al intentarlo no concluyen cómo se pueden elegir los hiperparámetros (Berrendero, 2016; Calvo, 2016); o (c) directamente aquellos que, al no incluir todos los hiperparámetros necesarios, omiten información que podría optimizar los resultados (Gil, 2018; Vargas, 2020; Cuenca y León, 2019). Para remediar esta situación, como metodología se integran las sugerencias de los creadores de la librería e1070 (Chih *et al.*, 2021) utilizada regularmente en R para SVM.

Para implementar una SVM se utilizan las librerías e1071 y scales, esta última para normalizar los datos, es decir, transformarlos de un rango de $R+$ (reales positivos) a un rango entre 0 y 1, con el fin de reducir el impacto que pueden tener las variables con rango más alto. Para el diseño del código en R, se siguieron las recomendaciones de Amta (2017) y Gomila (2018).

Resultados

Durante la implementación de este código, se utilizó la versión 4.4.1 de R para Windows (82 MB, 64-bits), descargada desde el sitio oficial (https://cran.rstudio.com/) y el entorno de ejecución $RStudio$, versión 2024.09.0-375, descargado desde (https://posit.co/download/rstudio-desktop/). Al momento de realizar este proceso, los paquetes instalados fueron:
- scales, versión 1.3.0.
- e1071, versión 1.7-16.

En primer lugar, se deben instalar los paquetes que se utilizarán. Por ahora se sugiere no hacer llamado a las librerías, ya que algunas de estas pueden compartir el nombre de las funciones, lo que podría ocasionar conflictos o resultados inesperados. De esta manera, se minimizan posibles interferencias entre las librerías.

Al inicio, y debido a que este conjunto de datos es aleatorio, se debe plantar una semilla que garantice que la elección de observaciones sea siempre la misma; sin una semilla, los resultados no serían replicables:

```
install.packages("scales")

## package 'scales' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\User\AppData\Local\Temp\RtmpOwBVzL\downloaded_packages

install.packages("e1071")

## package 'e1071' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\User\AppData\Local\Temp\RtmpOwBVzL\downloaded_packages

set.seed(123)
```

Para realizar el ejercicio, se construye una base ficticia con las funciones "rnorm" y "sample", para luego construir la tabla de trabajo:

```
n = 100
variable_a1 <- rnorm(n)
variable_a2 <- rnorm(n)
variable_a3 <- rnorm(n)
respuesta <- sample(0:1, n, replace = TRUE)
df_desarrollo <- data.frame(variable_a1, variable_a2, variable_a3,
                            respuesta)
```

Este código permitirá crear un "dataframe" de trabajo con 100 observaciones, 3 variables; en este caso, variable_1, variable_2 y variable_3, y la variable objetivo, denominada aquí "respuesta". Las SVM son parte del aprendizaje supervisado, por lo que es imperativo que el modelo se entrene con los resultados; de la calidad de estos dependerá la capacidad del modelo de aprender y así replicar correctamente el proceso para nuevos conjuntos de datos.

Para iniciar con el proceso de SVM, es necesario reescalar los datos para garantizar un rendimiento óptimo del modelo. La razón principal es que las SVM se basan en la distancia entre los puntos de datos y los hiperplanos (en el caso de datos linealmente separables) o las fronteras de decisión (en el caso de los no lineales). Las SVM son sensibles a la magnitud de las características; si estas tienen escalas muy diferentes (por ejemplo, una característica está en el rango de [0, 1] y otra en el rango de [0, 1000]), aquella con un rango mayor dominará las distancias y afectará de manera desproporcionada la optimización del modelo. Esto puede llevar a resultados incorrectos.

Esta librería permite estandarizar las variables utilizadas con la función "rescale":

```
library(scales)
r_variable_a1 = rescale(df_desarrollo$variable_a1)
r_variable_a2 = rescale(df_desarrollo$variable_a2)
r_variable_a3 = rescale(df_desarrollo$variable_a3)
respuesta_desarrollo = factor(df_desarrollo$respuesta)
```

Dentro del conjunto de datos debe existir una variable binaria que será predicha con el conjunto de variables independientes, en este ejemplo: tres variables. Así, la variable debe estar codificada como tipo factor para producir resultados también binarios, a partir de la implementación de la SVM. Esta transformación se logra con la función "factor", implementada sobre la variable dependiente:·

Dentro del conjunto de datos debe existir una variable binaria que será predicha con el conjunto de variables independientes, en este ejemplo: tres variables. Así, la variable debe estar codificada como tipo factor para producir resultados también binarios, a partir de la implementación de la SVM. Esta transformación se logra con la función "factor", implementada sobre la variable dependiente:

```
df_desarrollo_estand = data.frame(r_variable_a1, r_variable_a2,
                      r_variable_a3, respuesta_desarrollo)
```

El proceso de SVM inicia con la elección de una muestra de datos de entrenamiento sobre el conjunto total de datos, y unos datos de prueba que son una partición del conjunto total de datos, esto se conoce como *validación cruzada*. Para construir las muestras de entrenamiento y prueba se requiere la función "sample". Dentro de esta, se utiliza "nrow" que muestra el número de filas que posee un "data.frame" determinado. La opción "size" considera el tamaño de la muestra elegido (en este caso es 75 % de los datos) y la opción "replace" tiene como opciones "T" (*true*) o "F" (*false*), para permitir o negar la muestra con reemplazo, es decir, la opción de considerar más de una vez cada observación para incluirla en la muestra. En este caso, la orden es "tome el total de observaciones del conjunto de datos y tome aleatoriamente y sin reemplazo una muestra del 75 % de los datos":

```
df_mod = sample(1:nrow(df_desarrollo_estand),
                size = 0.75*nrow(df_desarrollo_estand),
                replace = F)
```

Es prudente resaltar que la partición de datos en un conjunto de entrenamiento y un conjunto de prueba, utilizando una proporción del 75 % para el entrenamiento y el 25 % para la

prueba, es una recomendación común por varias razones relacionadas con el rendimiento y la evaluación de modelos de *machine learning*. Sin embargo, esta proporción no es una regla estricta, sino una convención que ha demostrado funcionar bien en muchos escenarios.

El conjunto de datos almacenado en "df_mod" muestra las posiciones de los datos aleatorios elegidos, por lo que es necesario ejecutar el siguiente código que permite elegir un conjunto de datos de un conjunto determinado, en este caso se da la orden de exclusivamente las observaciones posicionadas dentro de la muestra elegida. El contenido en las llaves está posicionado como filas y columnas, por lo que antes de la coma, la acción afecta las filas y después de la coma, la acción afecta las columnas. En este caso, se requiere afectar las filas:

```
df_train = df_desarrollo_estand[df_mod, ]
```

Al incluir la resta (-) se da la orden de que considere todos los datos, excepto las posiciones señaladas en la muestra:

```
df_test = df_desarrollo_estand[-df_mod, ]
```

Así, se obtiene un conjunto de entrenamiento y prueba que será útil para medir la fiabilidad del modelo. Se recomienda, antes de avanzar, observar con la función "summary" para determinar que la proporción de ceros y unos dentro de ambos conjuntos sea similar, con el fin de obtener resultados de prueba más confiables.

```
summary(df_train)

##  r_variable_a1     r_variable_a2     r_variable_a3     respuesta_desarrollo
##  Min.   :0.0000   Min.   :0.07278   Min.   :0.02199   0:36
##  1st Qu.:0.3954   1st Qu.:0.25072   1st Qu.:0.31880   1:39
##  Median :0.5072   Median :0.34714   Median :0.45610
##  Mean   :0.5277   Mean   :0.37045   Mean   :0.49792
##  3rd Qu.:0.6361   3rd Qu.:0.47786   3rd Qu.:0.68275
##  Max.   :1.0000   Max.   :1.00000   Max.   :1.00000

summary(df_test)

##  r_variable_a1     r_variable_a2     r_variable_a3     respuesta_desarrollo
##  Min.   :0.07618   Min.   :0.0000   Min.   :0.0000   0:11
##  1st Qu.:0.42395   1st Qu.:0.2243   1st Qu.:0.2696   1:14
##  Median :0.54230   Median :0.3222   Median :0.3314
##  Mean   :0.55157   Mean   :0.3587   Mean   :0.3602
##  3rd Qu.:0.69626   3rd Qu.:0.4731   3rd Qu.:0.4632
##  Max.   :0.89497   Max.   :0.7845   Max.   :0.7223
```

Para crear SVM se utiliza la librería e1071, del paquete con el mismo nombre (Chih *et al.*, 2021):

```
library(e1071)
```

En primer lugar, se utiliza la función "tune" para modelar cuatro tipos de kernel de las máquinas, con esto se incluye un vector de costo con valores, 0.001, 0.01, 0.1, 1, 5, 10 y 20 para un kernel lineal. El mismo costo y un vector gamma con valores 0.5, 1, 2, 3, 4, 5 y 10 para un kernel radial. Este costo y vector gamma, más un conjunto de valores para la constante tao con los mismos valores que el vector gamma para un kernel sigmoide, y finalmente, los mismos valores más un grado p con valores entre 2 y 4 para un kernel polinomial (debido al coste computacional para los polinomios, se reducen a 6 los valores de 3 vectores adicionales, se eliminan los valores superiores a 5 de las tres variables consideradas).

Para una SVM con un kernel lineal, se eligen costos para la función:

$$u' \cdot v$$

Así,

```
svm_l = tune('svm', respuesta_desarrollo ~ .,
             data = df_train, kernel = 'linear',
             ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 20)),
             scale = F)
```

En este caso, "cost" hace referencia a v.

Para una SVM con un kernel radial, se eligen costos para la función:

$$exp(-\gamma u - v^2)$$

De la siguiente forma:

```
svm_r = tune('svm', respuesta_desarrollo ~ .,
             data = df_train, 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)),
             scale = F)
```

En este caso, "cost" hace referencia a v y gamma a γ.

Para una SVM con un kernel sigmoide, se eligen costos para la función:

$$\tanh[\gamma(u' \cdot v) + \tau]$$

Donde,

```
svm_s = tune('svm', respuesta_desarrollo ~ .,
             data = df_train, kernel = 'sigmoid',
             ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 20),
                           gamma = c(0.5, 1, 2, 3, 4, 5, 10),
                           coef0 = c(1, 2, 3, 4, 5, 10, 100)),
             scale = F)
```

Aquí, "cost" hace referencia a v, gamma a γ y coef0 a τ.

Para una SVM con un kernel polinomial, se eligen costos para la función:

$$[\gamma(u') + \tau]^p$$

Luego,

```
svm_p = tune('svm', respuesta_desarrollo ~ .,
             data = df_train, kernel = 'polynomial',
             ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5),
                           gamma = c(0.5, 1, 2, 3, 4, 5),
                           coef0 = c(1, 2, 3, 4, 5),
                           degree = c(2,3,4)),
             scale = F)
```

En este caso, "cost" hace referencia a v, gamma a γ, coef0 a τ; y "degree", a p. El investigador tiene libertad de modificar los valores en cada uno de los hiperparámetros elegidos hasta lograr una SVM eficiente para sus objetivos.

El siguiente código, permite observar el error asociado a cada forma del kernel. El investigador debería elegir una o dos funciones que menor coste generen y contrastarla con sus resultados. No en todos los casos, la SVM que menor error arroje es la que más se ajusta a los datos:

```
View(svm_l$performances)
View(svm_p$performances)
View(svm_r$performances)
View(svm_s$performances)
```

Al observar el dato de "error" generado por el código anterior en cada una de las tablas, se debe elegir aquella SVM con menor error y sobre esta seleccionar los hiperparámetros que provoquen tal nivel de error.

La siguiente orden permite especificar los mejores hiperparámetros para cada tipo de SVM:

```
View(svm_l$best.parameters)
View(svm_r$best.parameters)
View(svm_s$best.parameters)
View(svm_p$best.parameters)
```

Una vez observados, los mejores hiperparámetros se utilizan en la siguiente función. En este caso hipotético, la función de menor error fue una SVM con kernel tipo polinomial, con el siguiente comportamiento:

$$[5(u' \cdot 5) + 1]4 \tag{1}$$

Escrito en código como:

```
mod_svm = svm(respuesta_desarrollo ~ .,
              data = df_train, kernel = 'polynomial',
              cost=5, gamma = 5, coef0=1, degree=4)
```

Nótese que la base de datos utilizada en esta implementación es el conjunto de datos de entrenamiento. En caso de que se requiera hacer una gráfica:

```
plot(mod_svm,
     data=df_train, r_variable_a3 ~ r_variable_a1)
```

Una salida posible para este conjunto de datos hipotético se ilustra en la figura **8**.

Sobre el código, se debe resaltar que la graficación se da por parejas de variables; por lo que, al ejecutar el código, estas deben modificarse (en este caso son 3 variables y siempre son parejas de elementos, por lo que el resultado (3) es igual a $\binom{3}{2}$ hasta encontrar una combinación en la que gráficamente se observe una división clara de franjas de color (*véase* figura **8**).

Para construir la matriz de confusión, sobre lo cual se recomienda ver el apartado de métricas de desempeño de Ávila *et al.* (2022), debe utilizarse una función de predicción sobre los datos de entrenamiento:

```
pred.0 = predict(mod_svm, data=df_train)
```

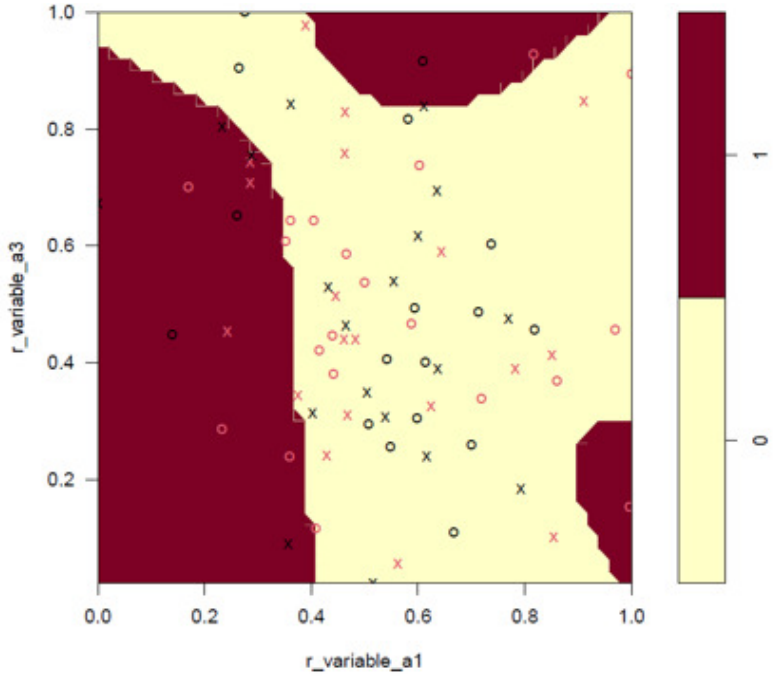

Figura 8. Máquina de soporte vectorial con kernel polinomial grado 4

Nota: tomada como resultado de R.

La matriz de confusión surge al construir una tabla comparativa entre el vector de predicción creado por el modelo y el vector de datos binarios del conjunto de entrenamiento:

```
m_con.0 = table(df_train$respuesta_desarrollo,
                pred.0)
```

Usando la función construida sobre los datos de entrenamiento, se implementa la máquina para los datos de prueba a través del código "newdata" que integra los nuevos datos a la máquina construida. Sobre estos datos también se construye una matriz de confusión. A grandes rasgos, el objetivo es que las matrices generadas no difieran proporcionalmente, y en general, que los elementos fuera de la diagonal principal se aproximen a 0:

```
pred.1 = predict(mod_svm, newdata =df_test)
m_con.1 = table(df_test$respuesta_desarrollo,pred.1)
```

Finalmente, y una vez revisada la matriz de confusión (proceso que escapa del objetivo de este documento), incluidas las métricas que se generan de ella, y entendiendo que el modelo se ajusta a los datos de entrenamiento y prueba, se procede a implementar el modelo en la tabla de despliegue, que por definición no tiene construida la variable "respuesta", en este caso "respuesta_despliegue", que es la que se espera predecir con el modelo:

```
set.seed(123)
m=2500
variable_b1 <- rnorm(m)
variable_b2 <- rnorm(m)
variable_b3 <- rnorm(m)
df_despliegue = data.frame(variable_b1, variable_b2,
                           variable_b3)
r_variable_b1 = rescale(df_despliegue$variable_b1)
r_variable_b2 = rescale(df_despliegue$variable_b2)
r_variable_b3 = rescale(df_despliegue$variable_b3)
df_despliegue_estand = data.frame(r_variable_b1,
                      r_variable_b2, r_variable_b3)
names(df_despliegue_estand) = c("r_variable_a1",
                      "r_variable_a2", "r_variable_a3")
```

Sobre este nuevo conjunto de datos se debe garantizar que los nombres son iguales, para que la predicción pueda realizarse:

```
respuesta_despliegue = predict(mod_svm,
                       newdata = df_despliegue_estand)
```

Luego se integran a dicho conjunto de variables como una variable adicional:

```
despliegue = data.frame(df_despliegue_estand,
                      respuesta_despliegue)
head(despliegue)

##   r_variable_a1 r_variable_a2 r_variable_a3 respuesta_despliegue
## 1     0.3863460     0.3734523     0.4095113                    0
## 2     0.4376486     0.5637984     0.6369838                    1
## 3     0.7155022     0.3695475     0.3179516                    1
## 4     0.4843518     0.3954506     0.6865559                    1
## 5     0.4934816     0.5941889     0.6073321                    1
## 6     0.7397879     0.4043158     0.5258313                    0
```

Conclusiones

En el presente trabajo se ha demostrado la relevancia de las máquinas de soporte vectorial (SVM, por su sigla en inglés) como una herramienta robusta para la clasificación de datos, y se ha resaltado el impacto de una correcta selección de los hiperparámetros y kernels. El proceso

detallado, desde la generación de las variables predictoras hasta la evaluación del modelo mediante validación cruzada, proporciona una guía completa para la implementación efectiva de SVM en tareas de clasificación.

Una de las conclusiones más importantes del estudio es la necesidad de ajustar adecuadamente los hiperparámetros del modelo SVM. Los experimentos con distintos tipos de kernel (lineal, radial, sigmoide y polinómico) muestran que cada uno requiere configuraciones específicas de los hiperparámetros *cost*, *gamma*, *coef0*, y *degree*, para obtener un rendimiento óptimo. En particular, se observó que el kernel lineal es efectivo cuando los datos son linealmente separables; mientras que los no lineales, como el radial y el polinómico, ofrecen mayor flexibilidad en la representación de patrones complejos. Estos resultados evidencian la importancia de emplear una metodología de ajuste de hiperparámetros, ya que la selección incorrecta de estos puede resultar en modelos con bajo rendimiento.

El preprocesamiento es un paso crucial en la creación de modelos robustos. En este estudio, el rescalado de las variables predictoras ha sido un componente esencial para asegurar el rendimiento adecuado del SVM. Las SVM son altamente sensibles a la magnitud de las características, y sin un reescalado adecuado, el modelo puede verse influenciado de manera desproporcionada por variables de mayor escala. La utilización de la función "rescale" ha garantizado que todas las variables predictoras estén en la misma escala, lo cual es particularmente importante cuando se emplean los tipos no lineales.

La implementación de validación cruzada ha sido clave para asegurar que el modelo no solo se ajuste bien a los datos de entrenamiento, sino que también generalice bien a nuevos conjuntos de datos. Mediante la función "tune" en R se exploraron diferentes combinaciones de hiperparámetros y se seleccionaron aquellos que minimizaban el error de clasificación.

El análisis de las matrices de confusión, obtenidas tanto para los datos de entrenamiento como para los de prueba, ha facilitado la cuantificación del rendimiento del modelo en términos de precisión. Aunque los modelos SVM suelen ser robustos, hay que ajustar cuidadosamente los hiperparámetros para evitar problemas como el sobreajuste o la subestimación de la complejidad de los datos. A través de este proceso, se identificaron los mejores modelos, lo que demuestra que la correcta sintonización de los hiperparámetros puede aumentar considerablemente la exactitud de las predicciones.

Los resultados en este estudio presentan una técnica estándar para la implementación de SVM. Sin embargo, la elección del kernel y de los hiperparámetros debe estar alineada con las características particulares del conjunto de datos. En aplicaciones futuras, el uso de otras téc-

nicas de optimización de hiperparámetros, como la búsqueda en cuadrícula o la optimización bayesiana, podría mejorar los resultados.

Además, sería interesante explorar el rendimiento de SVM en conjuntos de datos más grandes o más dimensionales, donde la escalabilidad del modelo puede convertirse en un desafío. El uso de técnicas como *kernel trick* ha demostrado ser eficaz para manejar la complejidad no lineal de los datos, lo que amplía las posibles aplicaciones de SVM en áreas como la visión por computadora, bioinformática y reconocimiento de patrones.

En resumen, las SVM siguen siendo una técnica de clasificación robusta y flexible, particularmente cuando se implementan con una selección correcta de hiperparámetros y un preprocesamiento cuidadoso de los datos. El ajuste adecuado de los hiperparámetros mediante validación cruzada asegura la generalización del modelo y minimiza el riesgo de sobreajuste. Este trabajo ha destacado los pasos clave para implementar con éxito un modelo SVM y proporciona una guía clara sobre cómo sintonizar los hiperparámetros para maximizar su rendimiento.

Es importante aclarar que el uso del código no puede entenderse como algo rutinario o definitivo, pues cada conjunto de datos presenta un reto diferente, y encontrar la combinación de hiperparámetros adecuada tiene tanto que ver con el método como con la experiencia.

Referencias

Amat, J. (abril de 2017). *Máquinas de vector soporte (support vector machines, SVMs)*. RPubs.com. https://rpubs.com/Joaquin_AR/267926

Ávila, P., González, J., Vargas, D. y Pérez, P. (2022). Predicción del rezago social en México: un enfoque de aprendizaje automático a partir de datos de unidades económicas. *Agrociencias*, *56*(2), 248-263. https://doi.org/10.47163/agrociencia.v56i2.2768

Berrendero, J. (2016). *Máquinas de vectores soporte con R*. Rpubs.com. https://rpubs.com/joser/svm

Betancourt, G. (2005). Las máquinas de soporte vectorial (SMV). *Scientia et Technica*, *11*(27), 67-72. https://dialnet.unirioja.es/descarga/articulo/4838384.pdf

Calvo, D. (3 de octubre de 2016). *SVM-Máquinas de vectores de soporte en R*. Diegocalvo.es. https://www.diegocalvo.es/svm-maquinas-de-vectores-de-soporte-en-r/

Campo, E. (2016). *Introducción a las máquinas de vector soporte (SVM) en aprendizaje supervisado* [Tesis de grado]. Universidad de Zaragoza. https://zaguan.unizar.es/record/59156/files/TAZ-TFG-2016-2057.pdf

Chih, C., Chih, L., Dimitriadou, E., Hornik, K., Leisch, F., Meyer, D. y Weingessel, A. (2021). *Misc functions of the Department of Statistics, probability theory group (formerly: e1071)*, TU Wien. Cran.R-project.org. https://cran.r-project.org/web/packages/e1071/e1071.pdf

Cuenca, D. y León, D. (2019). *Support vectormachine.Rpubs.com.* https://rpubs.com/Dario_BSC/SVM

Gil, C. (junio de 2018). *Máquinas de vectorsoporte.Rpubs.com.* https://rpubs.com/Cristina_Gil/SVM

Gomila, J. (2018). 24 - *Support vector machines en RStudio* [Archivo de video]. YouTube. https://youtu.be/_JdK4FMzd28

Méndez, H. (2003). *Algunas propiedades del kernel empleado en máquinas de soporte vectorial para clasificación* [Tesis de posgrado]. Centro de Investigación en Matemáticas A.C. https://cimat.repositorioinstitucional.mx/jspui/bitstream/1008/11/2/TE%20118.pdf

Paredes, D. (26 de 06 de 2020). *Data Science con R. Análisis de datos y algoritmos de predicción con R*. Bookdown.org. https://bookdown.org/dparedesi/data-science-con-r/

Rodríguez, E., Vinante, C. y Leal, M. (2009). Enfoque óptimo del método kernel cuadrados mínimos parciales. *Saber, 21*(2), 172-178. https://www.redalyc.org/pdf/4277/427739440010.pdf

Sosa, S. (23 de abril de 2021). *Suport vector machine.* Rpubs.com. https://rpubs.com/sebas_Alf/759517

Tinoco, C. (2019). Laboratorio 9.6. *Máquinas de vectores de soporte*. Rpubs.com. https://rpubs.com/cesar_tinoco/511674

Vargas, M. (30 de abril de 2020). *Máquinas de Soporte Vectorial (support vector machine, SVM) en regresión.* Rpubs.com: https://rpubs.com/movo/607465