# Máquinas de Soporte Vectorial

Ana Toxqui, *MMOP, CIMAT,*

***Abstract*—El enfoque tradicional de Máquinas de Soporte Vectorial (SVM) implica resolver problemas de optimización cuadrática, y Gurobi proporciona una infraestructura robusta y eficiente para abordar estas tareas, por lo que este trabajo presenta la implementación eficiente del algoritmo de SVM utilizando la herramienta de optimización Gurobi.**

***Index Terms*—Maquina de Soporte Vectorial, Optimización, Minimización, Clasificación, SVM.**

## I. Introducción

LAS Máquinas de Soporte Vectorial (SVM, por sus siglas en inglés) han emergido como una herramienta poderosa en el ámbito del aprendizaje automático y la inteligencia artificial. Es considera una de las herramientas más importantes y robustas ya que han demostrado ser eficaces en una amplia variedad de aplicaciones, esencialmente son un método de clasificación o regresión basado en hiperplanos de margen máximo propuesto por Vladimir Vapnik. Poseen capacidad para manejar conjuntos de datos complejos y de alta dimensionalidad, así como su capacidad para generalizar bien a partir de datos de entrenamiento limitados. En este articulo encontraremos una implementación de SVM usando Gurobi y se presenta su utilidad para clasificar ciertos conjuntos de datos.

## II. Marco teórico

Las Máquinas de Soporte Vectorial fueron introducidas por por Cortés y Vapnik, como una extensión de la teoría de la complejidad computacional y la teoría de juegos. El enfoque principal de las SVM es encontrar un hiperplano de separación óptimo entre dos clases en un espacio de características. Este hiperplano se define como aquel que maximiza la distancia entre las muestras más cercanas de cada clase, también conocidas como vectores de soporte, es decir, dada una muestra de entrenamiento $\{(x_1, y_1), \ldots, (x_n, y_n)\} \subseteq \mathbb{R}^p \times \{-1, +1\}$, el objetivo de SVM es obtener un hiperplano que separa los datos ( $(x \in \mathbb{R}^p)$ en sus dos clases diferentes $(y \in \{-1, +1\})$.

Entre todos los hiperplanos posibles que pueden obtener tal separación entre las clases, SVM busca el que tiene el margen máximo (distancia máxima entre las clases y el hiperplano de separación). Denotemos por $\mathcal{H}$ un hiperplano en $\mathbb{R}^p$ en la forma $\mathcal{H} = \{z \in \mathbb{R}^p : \omega' z + \omega_0 = 0\}$ para algunos $\omega \in \mathbb{R}^p$ y $\omega_0 \in \mathbb{R}$. Este hiperplano inducirá una subdivisión del espacio de datos $\mathbb{R}^p$ en tres regiones: el medio espacio +1 (positivo) $\mathcal{H}^+ = \{z : omega' z + \omega_0 > 1\}$, el medio espacio -1 (negativo) $\mathcal{H}^- = \{z : \omega' z + omega_0 < -1\}$ y la tira $\mathcal{S} = \{z : -1 \leq \omega' z + \omega_0 \leq 1\}$. Así, el SVM se puede formular como el siguiente Problema No Lineal (PNL):

$$\begin{aligned} \min \quad & \frac{1}{2}\omega^t \omega \\ & \text{restringido a:} \\ & y_i(\omega^t x_i + b) \geq 1 \ \forall i = 1, \ldots, n. \end{aligned}$$

El cual al aplicarle sus condiciones KKT (Karush-Kuhn-Tucker) se puede reformular de la siguiente manera:

$$\min L(\alpha_i) = \sum_i^p \alpha_i - \frac{1}{2} \sum_i^p \sum_j^p \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

restringido a:

$$\begin{aligned} & \sum_{i=1}^p \alpha_i y_i = 0 \\ & \alpha_i \geq 0 \ \forall i = 1, \ldots, n. \end{aligned}$$

La solución de este problema de optimización se puede obtener de diversas formas y permite recuperar el hiperplano mediante $w = \sum_{i=1}^l \alpha_i y_i x_i$, donde $l$ es el número de vectores de soporte, los cuales corresponden a $\alpha_i > 0$.

## III. Implementación de SVM con Gurobi

Para mostrar como SVM realiza la clasificación de datos linealmente separables se utilizó Gurobi, el cual es un software de optimización que se utiliza para resolver problemas de programación lineal, programación entera mixta, programación cuadrática y otros problemas de optimización matemática. A continuación se muestra el código que se implemento en Python.

```
# Datos de entrenamiento
X = X_entrenamiento
y = Y_entrenamiento

# Crear el modelo
modelo = gp.Model("SVM_dual")

# Variables
m = len(X)
alfa = modelo.addMVar(m, lb=0.0,
vtype=GRB.CONTINUOUS, name="alfa")

# Funcion objetivo
objetivo = 0.5 * gp.quicksum(alfa[i] * alfa[j] *
y[i] * y[j] * np.dot(X[i], X[j])
for i in range(m) for j in range(m))
- gp.quicksum(alfa[i] for i in range(m))
```

```
# Restricciones
modelo.addConstr(gp.quicksum(
alfa[i] * y[i] for i in range(m)) == 0,
name="restriccion_lineal")

# Configurar el modelo para minimizar
modelo.setObjective(objetivo, GRB.MINIMIZE)

# Resolver el modelo
modelo.optimize()

# Obtener los resultados
alfa_optimo = np.array([alfa[i].x
for i in range(m)])

# Acceder a las variables de decision
valores_alfa = alfa.x

indices_soporte = np.where(alfa_optimo > 9e-2)[0]

# Parametros del hiperplano
w = np.sum((alfa_optimo[indices_soporte] *
y[indices_soporte, np.newaxis]) *
X[indices_soporte, :], axis=0)

# Termino independiente
b = np.mean(y[indices_soporte] -
np.dot(X[indices_soporte, :], w))
```

## IV. Resultados

Se probó que la implementación de SVM fuera correcta mediante 4 conjuntos de datos, 3 con 70 datos linealmente separables y uno con 30 datos linealmente separables, de esta manera se mostró que no se necesitan muchos datos de entrenamiento para obtener el hiperplano deseado, lo cual es ventaja sobre otros métodos. Una vez obtenida la solución del problema de optimización se recupero el vector $\omega$ y el número $b$ de manera que se pudo gráficar el hiperplano, los vectores de soporte y los planos paralelos que pasan por ellos.

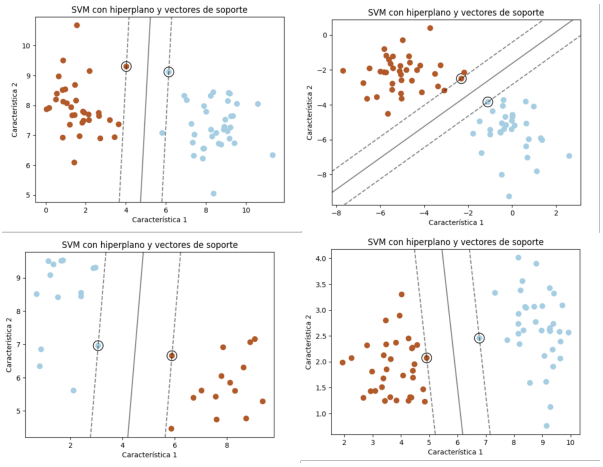


Fig. 1. Clasificación de datos para 4 conjuntos distintos

## V. Análisis y Conclusión

En nuestra implementación, aprovechamos las capacidades de programación lineal y cuadrática de Gurobi para optimizar la función de margen de SVM desde su aplicación de KKT's y encontrar el hiperplano óptimo que maximiza la separación entre clases. La interfaz de Gurobi en Python facilitó la integración de este software de optimización, con los ejemplos generados se probó que el método SVM se implemento correctamente. Se propone como trabajo a futuro explorar el método con datos de mayor dimensionalidad y aplicando el truco del Kernel, con distintos kernel's.

## VI. Bibliografía:

Barber, D. (2016). Bayesian reasoning and machine learning. Cambridge University Press.