

Support Vector Machine

Es un modelo enfocado a clasificación binaria, pero se puede extender a clasificación multiclase y regresión.

Definición

Definamos un *hiperplano*

Sea f un hiperplano definido por la siguiente *función lineal*:

$$f(x_1, \dots, x_p) = w_1x_1 + \dots + w_px_p + b = 0$$

en forma matricial, se expresa como:

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = 0$$

- \mathbf{x} un vector de variables independientes (un punto sobre el hiperplano).
- \mathbf{w} vector de coeficiente asociado a las variables independientes (o vector de pesos).
- b un escalar denominado término independiente (o el bias).

En un espacio de 2 dimensiones corresponde a una línea pero en un espacio de 3 o más dimensiones nos referimos a un hiperplano.

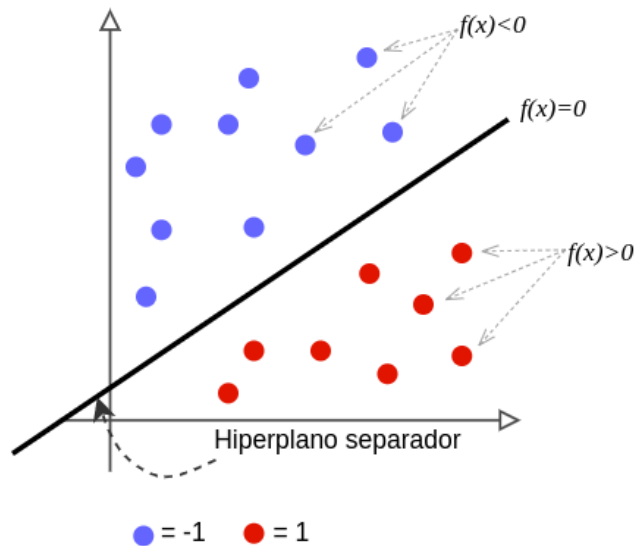
Hiperplano separador

Sea X el conjunto de datos de p características y y la variable respuesta, tal que:

$$X = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}, \quad \mathbf{x}_i \in \mathbb{R}^{1 \times p}, \quad y_i \in \{-1, 1\}, \quad i = 1, \dots, n$$

- \mathbf{x}_i es un vector correspondiente a la observación i .
- y_i es la respuesta esperada correspondiente a \mathbf{x}_i .
- n es el número de observaciones.

Supongamos un hiperplano ideal que separa completamente el conjunto de datos X en dos clases:



Dado este escenario, observemos que:

- Cualquier punto sobre el hiperplano cumple con: $f(\mathbf{x}) = 0$.
- Los puntos que están a los lados del hiperplano cumplen: $f(\mathbf{x}_i) > 0$ o $f(\mathbf{x}_i) < 0$.

Tomando en cuenta estas observaciones y dado que las respuestas $y_i \in \{-1, 1\}$, podemos deducir lo siguiente:

$$y_i f(\mathbf{x}_i) > 0, \quad i = 1, \dots, n$$

Significa que, si la función lineal (hiperplano) separa todos los puntos correctamente, entonces al hacer la multiplicación de la respuesta esperada y_i con la respuesta estimada $f(\mathbf{x}_i)$, el resultado siempre será positivo.

Si se cumple la anterior expresión para todas las observaciones x_i entonces f es un **hiperplano separador**.

Regla de clasificación

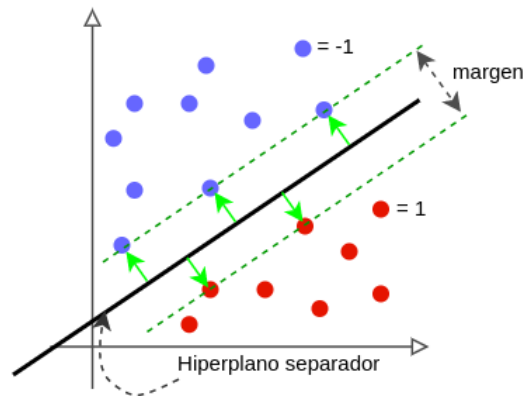
Dado que $f(\mathbf{x}_i) > 0$ o $f(\mathbf{x}_i) < 0$ podemos determinar una regla de clasificación de la siguiente manera:

$$\text{sign}(f(\mathbf{x}_i)) = \begin{cases} +1, & \text{si } f(\mathbf{x}_i) > 0 \\ -1, & \text{si } f(\mathbf{x}_i) < 0 \end{cases}$$

como se puede observar solo necesitamos conocer el signo para hacer la clasificación.

¿Cómo encontrar el mejor *hiperplano separador*?

Debemos plantear un problema de optimización tal que la distancia mínima entre el hiperplano y los puntos más cercanos a este sea lo más grande posible, en otras palabras, **maximizar el margen**.



- Los puntos que están sobre el límite de margen (línea punteada) se denominan *support vectors*.

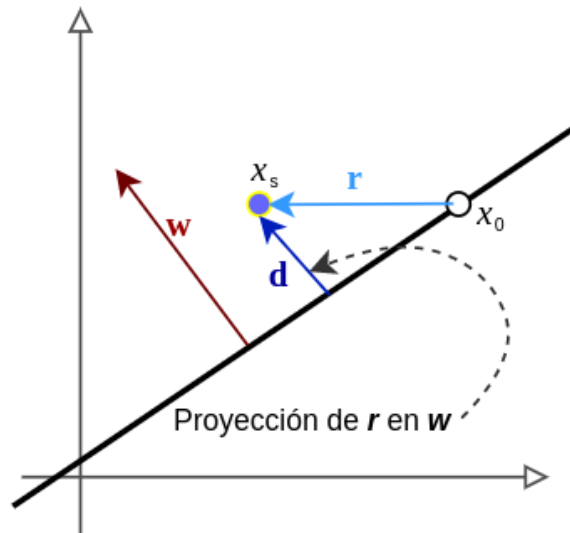
Para resolver este problema se recurre a métodos de *optimización dual*.

Planteamiento del Problema primal

Sea x_0 un punto sobre el hiperplano separador y x_s uno de los puntos de X (es decir, una observación) **más cercano al hiperplano**.

Tomando estos dos puntos, podemos formar un vector $\mathbf{r} = x_s - x_0$.

Luego, por definición sabemos que el vector de pesos \mathbf{w} es ortogonal al hiperplano, proyectando el vector \mathbf{r} sobre el vector \mathbf{w} obtenemos un vector \mathbf{d} cuyo módulo (longitud) es la mínima distancia entre x_s y el hiperplano, a esta distancia la denominamos *margen*.



y se expresa como:

$$d = \|\mathbf{d}\| = \left\| \frac{\mathbf{r}^T \mathbf{w}}{\mathbf{w}^T \mathbf{w}} \mathbf{w} \right\|$$

por simplicidad omitiremos cálculos intermedios, llegando a la siguiente expresión:

$$d = \frac{|\mathbf{w}^T \mathbf{x}_s + b|}{\|\mathbf{w}\|}$$

bajo el supuesto que nuestro modelo clasifica correctamente, entonces tenemos que $|\mathbf{w}^T \mathbf{x}_s + b| = 1$, reemplazando esto en la anterior expresión, obtenemos que el *margen* es:

$$d = \frac{1}{\|\mathbf{w}\|}$$

definida esta métrica, debemos encontrar el mejor *margen*, por lo tanto necesitamos maximizar d

$$d = \max_{\mathbf{w}, b} \left(\frac{1}{\|\mathbf{w}\|} \right) \quad \text{sujeto a: } y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1$$

Nota: Recordar que y_i es 1 o -1, además si el modelo predice correctamente para todas las observaciones, entonces: $y_i f(\mathbf{x}_i) \geq 1 \implies y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1$.

Para expresarlo en términos de minimización, vamos a invertir el término \mathbf{w} , de manera que suba al numerador. Además, **por conveniencia vamos a elevar al cuadrado y dividir entre dos**, esto para facilitar el cálculo de la derivada (esto es solo un truco matemático). Resultando en la siguiente expresión:

$$\begin{aligned}
d &= \min_{\mathbf{w}, b} \left(\frac{1}{2} \|\mathbf{w}\|^2 \right) \\
&= \min_{\mathbf{w}, b} \left(\frac{1}{2} \mathbf{w}^T \mathbf{w} \right) \quad \text{sujeto a: } y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1
\end{aligned}$$

Luego, para resolver este problema podemos aplicar el método de **multiplicadores de Lagrange**, obteniendo la siguiente expresión:

$$\begin{aligned}
L(\mathbf{w}, b, \alpha) &= \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^n \alpha_i [y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1] \\
&\quad \text{sujeto a: } \alpha \geq 0
\end{aligned}$$

- α es un multiplicador de Lagrange asociado a cada par (\mathbf{x}_i, y_i) .
- Los puntos \mathbf{x}_i cuyo $\alpha > 0$ serán los *support vectors*, por ende los puntos cuyo $\alpha = 0$ se descartan.

obteniendo así la definición del **problema primal**.

Finalmente, **notemos que L tiene una forma cuadrática** en \mathbf{w} , además depende de tres variables. El siguiente paso será encontrar una expresión equivalente que solamente dependa de los multiplicadores de Lagrange.

Planteamiento del Problema dual

Tomando en cuenta el problema primal, donde obtuvimos L (una función que depende de tres variables), ahora el objetivo será obtener una función que solo dependa de α (multiplicadores de Lagrange).

Derivamos L respecto a \mathbf{w} y b e igualamos a cero:

$$\begin{aligned}
\frac{\partial L}{\partial \mathbf{w}} &= \mathbf{w} - \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i = 0 \\
\frac{\partial L}{\partial b} &= - \sum_{i=1}^n \alpha_i y_i = 0
\end{aligned}$$

de las anteriores expresiones, obtenemos que:

$$\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i \quad (1)$$

$$\sum_{i=1}^n \alpha_i y_i = 0 \quad (2)$$

finalmente, reemplazando (1) y (2) en L y reordenando, tenemos:

$$L(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{m=1}^n \alpha_i \alpha_m y_i y_m \mathbf{x}_i \cdot \mathbf{x}_m$$

sujeto a: $\alpha_i \geq 0$ y $\sum_{i=1}^n \alpha_i y_i = 0$

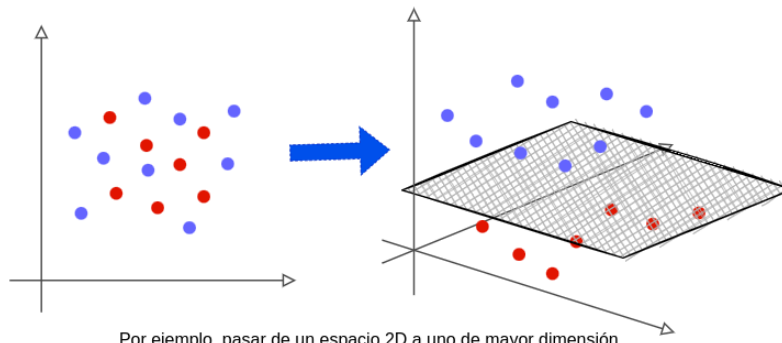
De esta forma obtenemos una función que solo depende de los multiplicadores de Lagrange α , obteniendo así la definición del **problema dual**.

Función Kernel

Notemos que la función $L(\alpha)$ tiene un producto escalar de dos vectores $\mathbf{x}_i \cdot \mathbf{x}_m$, este puede ser expresado como una **función kernel**:

$$k(\mathbf{x}_i, \mathbf{x}_m) = \mathbf{x}_i \cdot \mathbf{x}_m$$

Este "truco" del kernel es muy útil para transformar espacios no linealmente separables en espacios linealmente separables, tan solo modificando el kernel.



Por ejemplo, pasar de un espacio 2D a uno de mayor dimensión, tal que sea posible separar linealmente los puntos.

Existen diferentes tipos de kernel:

- Kernel Lineal: $k(\mathbf{x}_i, \mathbf{x}_m) = \mathbf{x}_i \cdot \mathbf{x}_m$
- Kernel Polinómico de grado d : $k(\mathbf{x}_i, \mathbf{x}_m) = (1 + (\mathbf{x}_i \cdot \mathbf{x}_m))^d$
- Kernel de Base Radial: $k(\mathbf{x}_i, \mathbf{x}_m) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_m\|^2)$
- Kernel de Red Neuronal: $k(\mathbf{x}_i, \mathbf{x}_m) = \tanh(\lambda_1(\mathbf{x}_i \cdot \mathbf{x}_m) + \lambda_2)$

Dado que se está cambiando el espacio, la **función de predicción** va a cambiar.

$$f(x_j) = \sum_{i=1}^n \alpha_i y_i k(x_j, x_i) + b$$

- x_j es un nuevo punto.
- x_i una observación del conjunto de datos de entrenamiento.

luego para determinar b se puede resolver mediante $y_i f(x_i) = 1$, dando como resultado:

$$b = (y_i - \sum_{m=1}^n \alpha_i y_i k(x_i, x_m))_{\mathbb{S}}$$

- \mathbb{S} es un subconjunto conformado por los *support vectors*: $(\mathbf{x}_i, y_i) \in \mathbb{S}$. Recordar que los *support vectors* son los puntos con $0 < \alpha_i < C$.
- x_m una observación del conjunto de datos de entrenamiento.
- El resultado final será el promedio de todos los b calculados.

Importante: estas expresiones son para cuando se tiene un modelo con un kernel no lineal solamente.

Tomando en cuenta la definición del problema dual y la función kernel, **finalmente obtenemos un problema de optimización:**

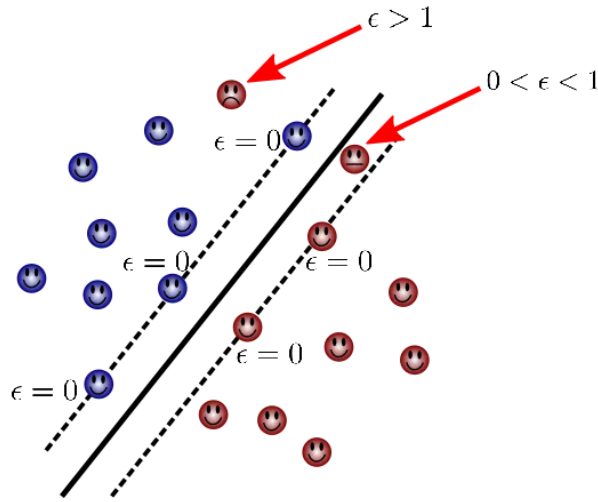
$$\begin{aligned} \max_{\alpha} L(\alpha) &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{m=1}^n \alpha_i \alpha_m y_i y_m \mathbf{k}(\mathbf{x}_i, \mathbf{x}_m) \\ \text{sujeto a: } \alpha_i &\geq 0 \quad \text{y} \quad \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned}$$

Nota:

- K es la función kernel.
- $\alpha_i \geq 0$ es una restricción obtenida en el problema primal.
- Los puntos x_i cuyos α_i son mayor a cero, son los *support vectors*.

Soft-margin SVM

Todo lo analizado anteriormente tiene una limitante práctica, porque definimos un modelo que no permite errores, en otras palabras, el modelo **separa todos los puntos** correctamente, pero en problemas de la vida real esto no es posible.



Para permitir errores vamos a introducir una nueva variable de error ϵ , recordemos la expresión $y_i f(\mathbf{x}_i) \geq 1$, por lo tanto ahora tenemos:

$$y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \epsilon_i$$

Considerando esta nueva expresión dentro del **problema primal**, tenemos:

$$\min \quad d = \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^n \epsilon_i \quad \text{sujeto a: } y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \epsilon_i$$

- C es un parámetro de regularización, define cuan estricto o permisivo será el margen. Si C es muy grande la clasificación errónea general será menor, es decir, casi no se permiten errores; en cambio si C es menor, se permite que las observaciones puedan estar del lado incorrecto.

escribiendo en términos de **multiplicadores de Lagrange**, tenemos:

$$L = \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^n \epsilon_i - \sum_{i=1}^n \alpha_i [y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 + \epsilon_i] - \sum_{i=1}^n \mu_i \epsilon_i$$

sujeto a: $\alpha_i \geq 0, \quad \mu_i \geq 0$

- μ es un nuevo multiplicador de Lagrange para los errores ϵ .

Siguiendo el mismo procedimiento del **problema dual**, calcular las derivadas de L respecto \mathbf{w} , b y ϵ e igualar a cero:

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{w}} &= \mathbf{w} - \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i = 0 \\ \frac{\partial L}{\partial b} &= - \sum_{i=1}^n \alpha_i y_i = 0 \\ \frac{\partial L}{\partial \epsilon_i} &= C - \alpha_i - \mu_i = 0 \end{aligned}$$

del cálculo anterior obtenemos las siguientes expresiones:

$$\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i, \quad \sum_{i=1}^n \alpha_i y_i = 0, \quad \alpha_i = C - \mu_i$$

Recordar que $\alpha_i \geq 0$ y $\mu_i \geq 0$, dado que $\alpha_i = C - \mu_i$ entonces la restricción de α_i cambia a:
 $0 \leq \alpha_i \leq C$.

Remplazando estas expresiones en la función L y reordenando, obtenemos la siguiente expresión:

$$\begin{aligned} \operatorname{argmax}_{\alpha} \quad L(\alpha) &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{m=1}^n \alpha_i \alpha_m y_i y_m \mathbf{k}(\mathbf{x}_i, \mathbf{x}_m) \\ \text{sujeto a:} \quad &0 \leq \alpha_i \leq C \quad \text{y} \quad \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned}$$

- Los puntos \mathbf{x}_i asociados a un $0 < \alpha_i < C$ son los *support vectors*.
- Notar que esta función es cuadrática en α , por lo que se trata de un problema de *programación cuadrática*.

Notemos que esta expresión es la misma que se obtuvo en el planteamiento del problema dual, excepto que ahora α_i tiene un límite superior en C .

Implementación

En primer lugar vamos a convertir $L(\alpha)$ en su forma matricial:

$$\operatorname{argmax}_{\alpha} \quad L(\alpha) = \mathbf{1}^T \alpha - \frac{1}{2} \alpha^T (\mathbf{H}) \alpha$$

- $\alpha \in \mathbb{R}^n$ el vector de variables α_i .
- $\mathbf{1} \in \mathbb{R}^n$ el vector de unos.
- $\mathbf{H} \in \mathbb{R}^{n \times n}$ la matriz tal que:
 - $\mathbf{H} = \mathbf{y}^T \mathbf{y} * \mathbf{K}$ el operador $*$ indica una multiplicación *elemento-a-elemento*.
 - $\mathbf{K} \in \mathbb{R}^{n \times n}$ la matriz kernel donde $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_m)$, en forma matricial:
 $\mathbf{K} = k(\mathbf{X}, \mathbf{X})$.
 - $\mathbf{y} \in \mathbb{R}^n$ el vector de respuestas (o etiquetas) y_i

Por lo general, las librerías requieren que la función objetivo de optimización esté en **términos de minimización**, para esto solamente hay que invertir el signo:

$$\operatorname{argmin}_{\alpha} \quad L(\alpha) = \frac{1}{2} \alpha^T \mathbf{H} \alpha - \mathbf{1}^T \alpha \quad \text{sujeto a: } 0 \leq \alpha \leq C, \quad \mathbf{y}^T \alpha = 0$$

Finalmente tenemos un problema de optimización que se puede resolver con **Programación Cuadrática** (en ingles QP).

Programación Cuadrática o QP

Nos permite resolver problemas de optimización donde la **función objetivo es cuadrática** en las variables y las restricciones son lineales:

$$\min_x \quad \frac{1}{2}x^T Px + q^T x, \quad \text{sueto a: } Gx \leq h, \quad Ax = b$$

- P es una matriz simétrica (define la parte cuadrática).
- q es un vector (parte lineal).
- G, h son restricciones de desigualdad.
- A, b son restricciones de igualdad.

Existen librerías de python como `cvxopt`/`quadprog` o `scikit-learn` para resolver este tipo de problemas..

Aplicando esto al problema dual planteado (minimización de $L(\alpha)$), podemos verificar que precisamente se trata del mismo problema, porque:

- La función L tiene una variable cuadrática en α .
- Sus restricciones son lineales: en $0 \leq \alpha \leq C, \quad y^T \alpha = 0$

Implementación con CVXOPT

Para resolver esta optimización, haremos uso de la librería `cvxopt` de python, usando el método correspondiente a [Quadratic Program](#).

Primero hay que preparar los argumentos (matrices):

$$P = \mathbf{H}_{n \times n}$$

$$q = -\mathbf{1}_{n \times 1}$$

$$G = \text{vstack}(-I_n, I_n) = \begin{pmatrix} -1 & 0 & \cdots & 0 \\ 0 & -1 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & -1 \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{pmatrix}_{2n \times n}$$

$$h = \text{vstack}(0_{n \times 1}, C_{n \times 1}) = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ C \\ \vdots \\ C \end{pmatrix}_{2n \times 1}$$

$$A = \mathbf{y}_{1 \times n}$$

$$b = 0_{1 \times 1}$$

luego llamar a la función `solvers.qp()`:

```
def min_dual_problem(X, y, C, kernel="linear"):
    n = X.shape[0]
    y = y.reshape((1,n)) # conversión a array 2D

    #K = kernel_matrix(X, X, kernel)
    K = X @ X.T # para este ejercicio forzamos a kernel lineal
    H = (y.T @ y) * K

    # Conversión a formato cvxopt
    P = matrix(H)
    q = matrix(-np.ones((n, 1)))
    G = matrix(np.vstack([-np.eye(n), np.eye(n)]))
    h = matrix(np.vstack([np.zeros((n, 1)), C * np.ones((n, 1))]))
    A = matrix(y)
    b = matrix(0.0)

    solvers.options['show_progress'] = False
    solution = solvers.qp(P, q, G, h, A, b)

    # retornar alphas encontrados
```

```
return np.array(solution['x']).flatten()
```

Para encontrar los mejores parámetros \mathbf{w} y b , calculamos:

$$\mathbf{w} = \mathbf{X}^T \alpha \mathbf{y}$$
$$b = (\mathbf{y} - \mathbf{X}\mathbf{w})_{\in S}$$

- S es el subconjunto correspondiente a los *support vectors*.

```
def get_w_b(alphas, X, y, C):  
    n = X.shape[0]  
  
    # Recordar que los x_i con 0 < alphas < C son los support vectors  
    support = (alphas > 1e-5) & (alphas < C - 1e-5)  
  
    w = X.T @ (alphas * y)  
    b = y[support] - (X[support] @ w)  
    b = np.mean(b)  
    return w, b, support  
  
# entrenar el modelo  
C = 1.0 # parámetro de regularización  
alphas = min_dual_problem(X, y, C)  
w, b, S = get_w_b(alphas, X, y, C)  
print('Alphas: ', alphas[S][:10])  
print('w: ', w)  
print('b: ', b)
```

Para realizar las predicciones:

$$\hat{y} = \text{sign}(f(\mathbf{x}_i))$$
$$= \text{sign}(\mathbf{w}^T \mathbf{x}_i + b = 0)$$

```
# predecir  
def predict(X, w, b):  
    return np.sign(X @ w + b)
```

Kernel de base radial

En caso de cambiar a un kernel no lineal, cambia la expresión del cálculo de b :

$$b = (y_i - \sum_{m=1}^n \alpha_i y_i k(x_i, x_m))_{\mathbb{S}}$$

- \mathbb{S} es un subconjunto conformado por los *support vectors*: $(\mathbf{x}_i, y_i) \in \mathbb{S}$. Recordar que los *support vectors* son los puntos con $0 < \alpha_i < C$.
- $x_m x_i$ una observación del conjunto de datos de entrenamiento.
- Luego se debe promediar para obtener un único b .

```
def get_b_rbf(alphas, X, y, C, gamma):
    # Los support vectors son 0 < alphas < C
    support = (alphas > 1e-5) & (alphas < C - 1e-5)
    b = y[support] - np.sum(alphas * y * kernel_matrix(X[support], X, "radial", gamma), axis=1)
    b = np.mean(b)
    return b, support
```

y también cambia la función de estimación $f(\mathbf{x}_i)$:

$$f(x_j) = \sum_{i=1}^n \alpha_i y_i k(x_j, x_i) + b$$

- x_j es una nueva observación.
- x_i una observación del conjunto de datos de entrenamiento.

La regla de clasificación es la misma $\hat{y} = \text{sign}(f(\mathbf{x}_i))$

```
def predict_rbf(X_new, X, y, b, alphas, gamma):
    f_x = np.sum(alphas * y * kernel_matrix(X_new, X, "radial", gamma), axis=1) + b
    return np.sign(f_x)
```

Referencias

- The Elements of Statistical Machine Learning - Hastie, Tibshirani, Friedman (2008)
- Support Vector Machines for Classification - Oscar Contreras (2019) <https://medium.com/data-science/support-vector-machines-for-classification-fc7c1565e3>
- CVXOPT Documentation <https://cvxopt.org/userguide/coneprog.html#quadratic-programming>