Deconstructing the Support Vector Machine: A Mathematical Analysis

Jenish Paudel

20 December, 2025

Prerequisites

This article assumes a conceptual understanding of what SVMs are and a solid foundation in linear algebra and matrix calculus. Readers should be comfortable working with vectors and matrices, including operations such as multiplication, transposition, and dot products, as well as solving systems of linear equations. An understanding of Gram matrices helps. The discussion relies heavily on the dual formulation of support vector classifiers. Familiarity with constrained optimization, particularly the use of Lagrange multipliers, will also be a massive aid in following the mathematical derivations.

Approach

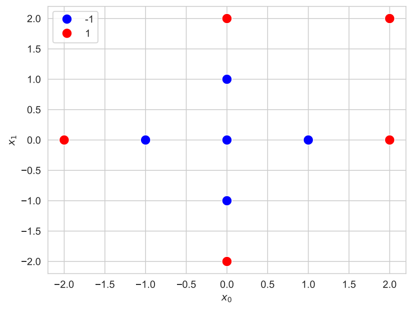
This article systemically develops the mathematical framework of support vector machines, progressing from the foundational hard-margin formulation to it's practical extensions. it begins with the primal optimization problem, derives its dual counterpart using lagrange multiplies and establishes KKT conditions that characterize optimal solutions.

From this foundation, we naturally extend to the soft-margin formulation, showing when slack variable and the regularization parameter C are introduced. This discussion also introduces kernel methods, showing how the kernel tricks emerge. Though the core remains hard-margin theory, theoretical extensions are presented, providing readers with a complete picture of the SVM framework while maintaining logical continuity.

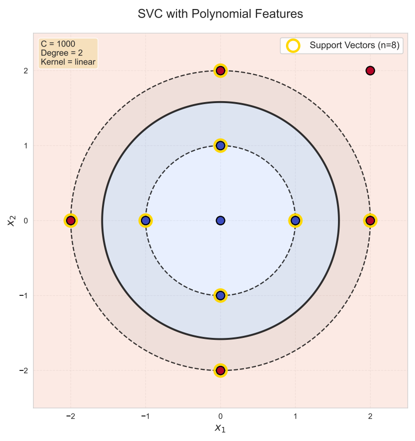
With the theoretical foundation established, we transition to a concrete computational example that instantiates the hard-margin case in detail. A dataset of concentric circles that are inherently non-linearly separable in a two-dimensional space is used as our testbed. For pedagogical clarity, we explicitly construct second-degree polynomial features rather than employing a kernel, making every step transparent.

The following dataset is not linearly separable in the two-dimensional input *space. However, under an appropriate polynomial feature transformation, the data is mapped to a higher-dimensional space in which linear separability is achieved.*

$$\mathrm{X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & -1 \\ -1 & 0 \\ 0 & 0 \\ 0 & 2 \\ 2 & 0 \\ 0 & -2 \\ -2 & 0 \\ 2 & 2 \end{bmatrix} \qquad \mathrm{y} = \begin{bmatrix} +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix} \quad (0)$$



(a) dataset scatterplot in R^2



(b) Dataset scatterplot in R^2 with decision boundary, support vectors and margins

Figure 1: Visualization of support vector classification (a) in the original feature space and (b) also in 2D space but flattened after polynomial transformation and decision boundary, in higher dimension were found.

The dataset presented above will be used as a pedagogical example to demonstrate the sequence of transformations that culminate in the derivation of support vector classifier decision boundaries.

Understanding The Problem

Support Vector Machines, with respect to classification, form a framework that identifies an optimal separating hyperplane between distinct classes. The core linear model is characterized by the decision function:

$$f(\mathbf{x}) = \left(\mathbf{w}^\top \mathbf{x} + b\right),$$

where the decision boundary is given by $\mathbf{w}^\top \mathbf{x} + b = 0$.
In this formulation, $\mathbf{w} \in R^d$ represents the normal vector to the hyperplane, dictating its orientation in the feature space, while $b \in R$ denotes the bias term that offsets the boundary from the origin.

Geometric Interpretation: For a weight vector $\mathbf{w}$ data point $\mathbf{x}_i$:

- $\mathbf{w}^\top \mathbf{x}_i + b = 0$: On the decision boundary
- $\mathbf{w}^\top \mathbf{x}_i + b > 0$: On the positive side
- $\mathbf{w}^\top \mathbf{x}_i + b < 0$: On the negative side

Note that we have not yet introduced the margin constraints. The marginal hyperplanes follow the same form but with constants ±1 instead of 0:

$$\mathbf{w}^\top \mathbf{x}_i + b = \pm 1$$

The values -1 and $+1$ are chosen because the class labels $y_i \in \{-1, +1\}$. The constraints require $y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1$ for all training points, ensuring they are correctly classified with a margin of at least 1.

The primary objective of the SVM is to maximize the distance between these two marginal hyperplanes that "sandwich" the decision boundary, thereby creating the widest possible margin between the classes. For two parallel planes with the same normal vector w:

$$\mathbf{w}^\top \mathbf{x} + b_1 = c_1 \quad \text{and} \quad \mathbf{w}^\top \mathbf{x} + b_2 = c_2,$$

the perpendicular distance between them is:

$$d = \frac{|c_2 - c_1|}{\|\mathbf{w}\|}.$$

For the SVM marginal hyperplanes $\mathbf{w}^\top \mathbf{x} + b = \pm 1$, substituting $c_1 = 1$ and $c_2 = -1$ into the distance formula yields:

$$d = \frac{|1 - (-1)|}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|}.$$

Thus, the core optimization problem is revealed: we must identify a weight vector $\mathbf{w}$ that maximizes the margin $\frac{2}{\|\mathbf{w}\|}$. To solve this, an equivalent and easier minimization problem can be derived which leads to the following primal optimization formulation:

$$\begin{aligned} \underset{\mathbf{w},b}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{subject to} \quad & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \quad \forall i = 1, \dots, n. \end{aligned}$$

The quadratic objective $\frac{1}{2}\|\mathbf{w}\|^2$ is mathematically equivalent to minimizing $\|\mathbf{w}\|$ for optimization purposes, while the linear inequality constraints enforce correct classification with a margin of at least one. Given the constrained primal optimization problem, one may derive an equivalent dual formulation through the method of Lagrange multipliers. The dual problem often provides computational advantages and reveals deeper structural insights. The Lagrangian for the primal SVM problem is:

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left[y_i(\mathbf{w}^\top \mathbf{x}_i + b) - 1 \right],$$

where $\boldsymbol{\alpha} = [\alpha_1, \dots, \alpha_n]^\top \geq \mathbf{0}$ are the Lagrange multipliers. By constructing the dual function and simplifying, we obtain the quadratic programming dual in standard minimization form:

$$\begin{aligned} \underset{\boldsymbol{\alpha}}{\text{minimize}} \quad & \frac{1}{2}\boldsymbol{\alpha}^\top \mathbf{Q} \boldsymbol{\alpha} - \mathbf{1}^\top \boldsymbol{\alpha} \\ \text{subject to} \quad & \boldsymbol{\alpha} \geq \mathbf{0}, \\ & \mathbf{y}^\top \boldsymbol{\alpha} = 0, \end{aligned}$$

where $\mathbf{1} = [1, \dots, 1]^\top$, $\mathbf{y} = [y_1, \dots, y_n]^\top$, and $\mathbf{Q} \in R^{n \times n}$ is the symmetric positive-semidefinite matrix with entries $Q_{ij} = y_i y_j \mathbf{x}_i^\top \mathbf{x}_j$. In matrix notation:

$$\mathbf{Q} = \begin{bmatrix} y_1 y_1 \mathbf{x}_1^\top \mathbf{x}_1 & y_1 y_2 \mathbf{x}_1^\top \mathbf{x}_2 & \cdots & y_1 y_n \mathbf{x}_1^\top \mathbf{x}_n \\ y_2 y_1 \mathbf{x}_2^\top \mathbf{x}_1 & y_2 y_2 \mathbf{x}_2^\top \mathbf{x}_2 & \cdots & y_2 y_n \mathbf{x}_2^\top \mathbf{x}_n \\ \vdots & \vdots & \ddots & \vdots \\ y_n y_1 \mathbf{x}_n^\top \mathbf{x}_1 & y_n y_2 \mathbf{x}_n^\top \mathbf{x}_2 & \cdots & y_n y_n \mathbf{x}_n^\top \mathbf{x}_n \end{bmatrix} = \text{diag}(\mathbf{y})\, \mathbf{X}\mathbf{X}^\top\, \text{diag}(\mathbf{y}) = (\mathbf{y}\mathbf{y}^\top) \odot (\mathbf{X}\mathbf{X}^\top),$$

with $\mathbf{X} = [\mathbf{x}_1 \cdots \mathbf{x}_n]^\top \in R^{n \times d}$ and $\odot$ denoting the Hadamard (element-wise) product. with $\mathbf{X} = [\mathbf{x}_1 \cdots \mathbf{x}_n]^\top \in R^{n \times d}$.The dual objective contains terms of the form $y_i y_j \mathbf{x}_i^\top \mathbf{x}_j$. Crucially, the dependence on the data appears exclusively through inner products $\mathbf{x}_i^\top \mathbf{x}_j$. This allows the extension to nonlinear classification via the kernel trick: replace $\mathbf{x}_i^\top \mathbf{x}_j$ with a kernel function $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$, where ϕ is an implicit mapping to a higher-dimensional space.

The discussion so far only addresses the hard-margin support vector machine, which requires perfect linear separability of the training data. This formulation has two significant limitations: it is applicable only to linearly separable datasets, and it is highly sensitive to outliers. A single well-placed outlier making it impossible to satisfy all constraints simultaneously.
To accommodate non-separable data and improve robustness to outliers, the soft-margin SVM introduces slack variables $\xi_i \geq 0$ that permit controlled margin violations. The primal optimization problem becomes:

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \quad i = 1,\dots,n, \\ & \xi_i \geq 0, \quad i = 1,\dots,n, \end{aligned}$$

where $C > 0$ is a regularization parameter that balances margin width against the total amount of slack. Each slack variable ξ_i quantifies the extent to which the i-th training point violates the margin condition.
The dual formulation of the soft-margin SVM stays structurally identical to the hard margin's, with a single modification to the constraints. Soft-margin imposes an upper-bound constraint to α and replaces $\alpha \geq 0$, with $0 \leq \alpha \leq C$
And finally, using this dual problem formulation, one obtains the optimal Lagrange multiplier vector $\boldsymbol{\alpha}^*$. These values are then used to recover the primal parameters: the weight vector $\mathbf{w}$ and the bias scalar b, via the relations:

$$\mathbf{w} = X^\top(\boldsymbol{\alpha}^* \odot \mathbf{y})$$
$$b = \frac{1}{n_s}\left(\mathbf{1}^\top \mathbf{y}_{\mathcal{S}} - \mathbf{1}^\top X_{\mathcal{S}}\mathbf{w}\right),$$

where $\mathbf{y}_{\mathcal{S}}$ contains the labels and $X_{\mathcal{S}}$ the feature vectors of the margin support vectors ($0 < \alpha_i^* < C$), and $n_s = |\mathcal{S}|$.
For the soft-margin SVM, after computing $\mathbf{w}$ and b, the slack variable ξ_i for each training point $\mathbf{x}_i$ is given by:

$$\xi_i = \max\bigl(0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)\bigr).$$

Identifying Support Vectors: The Lagrange multipliers directly determine which training points are support vectors:

- If $\alpha_i^* > 0$, then $\mathbf{x}_i$ is a support vector.
- If $\alpha_i^* = 0$, then $\mathbf{x}_i$ is not a support vector.

For Soft Margin, margin support vectors (used to compute b) satisfy $0 < \alpha_i^* < C$.

- If $\alpha_i^* = 0$, then $\mathbf{x}_i$ is not a support vector.
- If 0 ¡ $\alpha_i^* < C$, then $\mathbf{x}_i$ is a support vector exactly on the margin
- If $\alpha_i^* = C$, then $\mathbf{x}_i$ is a support vector inside the margin or misclassified.

At this point, the training phase of the support vector classifier is complete. The model parameters $(\mathbf{w}, b)$ are fully determined and can be used to classify new data points via $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$.

Step-by-Step Mathematical Example of a linearSVC

The set of features and label I will be working with is:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & -1 \\ -1 & 0 \\ 0 & 0 \\ 0 & 2 \\ 2 & 0 \\ 0 & -2 \\ -2 & 0 \\ 2 & 2 \end{bmatrix} \qquad y = \begin{bmatrix} +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix}$$

The given dataset cannot be separated by a straight line in its original form. It only becomes linearly separable when transformed into a higher-dimensional space. In this example, a second-degree polynomial transformation will be used.
In real applications, such transformations are usually done using a **kernel** (like rbf and polynomial kernel). The kernel trick allows us to work in a higher-dimensional space without actually creating all the new features, which saves a lot of computation.
For clarity, this example will explicitly create the polynomial features so the transformation is visible. The topic and mathematics of kernel tricks will be discussed later.

Generating polynomial features: Let $x = (x_1, x_2, \dots, x_n)$ be the input features and let $d \in N$ be the maximum polynomial degree.

- For each feature x_i, define the exponent set

$$A_i = \{0, 1, \dots, d\}.$$

- Form the Cartesian product of exponent sets

$$A = A_1 \times A_2 \times \cdots \times A_n = \{0, \dots, d\}^n.$$

- Each element $\alpha = (\alpha_1, \dots, \alpha_n) \in A$ defines a monomial

$$x^\alpha = x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}.$$

- Select the subset
  $$A_d = \left\{ \alpha \in A \,\middle|\, \sum_{i=1}^{n} \alpha_i \leq d \right\},$$
  which enforces the total-degree constraint.

The polynomial feature map is constructed by generating all monomials of the form
$$x_1^{\alpha_1} x_2^{\alpha_2} \dots x_n^{\alpha_n},$$
where each α_i is a non-negative integer and the total degree satisfies
$$\sum_{i=1}^{n} \alpha_i \leq d$$
for a chosen degree d. Each such monomial corresponds to a column in the transformed feature matrix.

Given a dataset with two features x_1 and x_2, a second-degree polynomial transformation yields all monomials of degree $d \leq 2$. The resulting feature set is:
$$S = \left\{ x_1^2, x_1 x_2, x_2^2, x_1, x_2, 1 \right\},$$
where the constant term $x_1^0 x_2^0 = 1$ is included as the intercept.
These features can be organized into a transformed feature matrix as follows:

x_1	x_2	x_1x_2	x_1^2	x_2^2	Intercept
0	1	0	0	1	1
1	0	0	1	0	1
0	-1	0	0	1	1
-1	0	0	1	0	1
0	0	0	0	0	1
0	2	0	0	4	1
2	0	0	4	0	1
0	-2	0	0	4	1
-2	0	0	4	0	1
2	2	4	4	4	1

Table 1: Polynomial feature matrix (degree = 2) for the previous 2D dataset. Columns correspond to the monomials in S.

Now using this new transformed dataset, next goal is to find the α vector through solving the dual problem:

$$\begin{aligned} \underset{\boldsymbol{\alpha}}{\text{minimize}} \quad & \frac{1}{2}\boldsymbol{\alpha}^\top \mathbf{Q}\boldsymbol{\alpha} - \mathbf{1}^\top\boldsymbol{\alpha} \\ \text{subject to} \quad & \boldsymbol{\alpha} \geq \mathbf{0}, \\ & \mathbf{y}^\top\boldsymbol{\alpha} = 0, where \mathrm{Q} = (\mathrm{yy}^\top) \odot (\mathbf{XX}^\top) \end{aligned}$$

First, solving for gram matrix Q:

$$yy^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

and

$$XX^T = \begin{bmatrix} 3 & 1 & 1 & 1 & 1 & 7 & 1 & 3 & 1 & 7 \\ 1 & 3 & 1 & 1 & 1 & 1 & 7 & 1 & 3 & 7 \\ 1 & 1 & 3 & 1 & 1 & 3 & 1 & 7 & 1 & 3 \\ 1 & 1 & 1 & 3 & 1 & 1 & 3 & 1 & 7 & 3 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 7 & 1 & 3 & 1 & 1 & 21 & 1 & 13 & 1 & 21 \\ 1 & 7 & 1 & 3 & 1 & 1 & 21 & 1 & 13 & 21 \\ 3 & 1 & 7 & 1 & 1 & 13 & 1 & 21 & 1 & 13 \\ 1 & 3 & 1 & 7 & 1 & 1 & 13 & 1 & 21 & 13 \\ 7 & 7 & 3 & 3 & 1 & 21 & 21 & 13 & 13 & 57 \end{bmatrix}$$

and finally,

$$Q = yy^T \odot XX^T = \begin{bmatrix} 3 & 1 & 1 & 1 & 1 & -7 & -1 & -3 & -1 & -7 \\ 1 & 3 & 1 & 1 & 1 & -1 & -7 & -1 & -3 & -7 \\ 1 & 1 & 3 & 1 & 1 & -3 & -1 & -7 & -1 & -3 \\ 1 & 1 & 1 & 3 & 1 & -1 & -3 & -1 & -7 & -3 \\ 1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 \\ -7 & -1 & -3 & -1 & -1 & 21 & 1 & 13 & 1 & 21 \\ -1 & -7 & -1 & -3 & -1 & 1 & 21 & 1 & 13 & 21 \\ -3 & -1 & -7 & -1 & -1 & 13 & 1 & 21 & 1 & 13 \\ -1 & -3 & -1 & -7 & -1 & 1 & 13 & 1 & 21 & 13 \\ -7 & -7 & -3 & -3 & -1 & 21 & 21 & 13 & 13 & 57 \end{bmatrix}$$

Now that $\mathbf{Q}$ is known, the remaining task is solving the constrained quadratic program. We introduce Lagrangian multipliers:

- scalar multiplier $\lambda \in R$ for the equality constraint $\mathbf{y}^\top \boldsymbol{\alpha} = 0$,
- vector multiplier $\boldsymbol{\mu} \geq \mathbf{0}$ for the inequality constraints $\boldsymbol{\alpha} \geq \mathbf{0}$.

$$\mathcal{L}(\boldsymbol{\alpha}, \lambda, \boldsymbol{\mu}) = \frac{1}{2}\boldsymbol{\alpha}^\top \mathbf{Q}\boldsymbol{\alpha} - \mathbf{1}^\top \boldsymbol{\alpha} + \lambda(\mathbf{y}^\top \boldsymbol{\alpha}) - \boldsymbol{\mu}^\top \boldsymbol{\alpha}.$$

Since the problem is convex, the KKT conditions are both necessary and sufficient. The stationarity condition yields

$$\nabla_{\boldsymbol{\alpha}} \mathcal{L} = \mathbf{Q}\boldsymbol{\alpha} - \mathbf{1} + \lambda \mathbf{y} - \boldsymbol{\mu} = \mathbf{0},$$

and together with primal feasibility ($\boldsymbol{\alpha} \geq \mathbf{0}$, $\mathbf{y}^\top \boldsymbol{\alpha} = 0$), dual feasibility ($\boldsymbol{\mu} \geq \mathbf{0}$), and complementary slackness ($\mu_i \alpha_i = 0$), these fully characterize the optimum.
The complementary slackness condition $\mu_i \alpha_i = 0$ introduces a combinatorial element: for each data point, either $\alpha_i = 0$ (the point is not a support vector) or $\mu_i = 0$ (the point is a support vector). With $n = 10$ points, this results in $2^{10} = 1024$ possible configurations, making direct analytical solution impractical.
In practice, this convex quadratic program is solved efficiently using numerical optimization. The dual formulation fits the standard form of a quadratic program (QP), allowing us to employ established solvers. For transparency and reproducibility, the solution was obtained via the `scipy.optimize.minimize` function using the SLSQP (Sequential Least Squares Quadratic Programming) method, which is well-suited for small to medium-scale constrained problems.
Once the optimal $\boldsymbol{\alpha}^*$ is obtained, any component below a small numerical tolerance (here 10^{-5}) is treated as zero, as it corresponds to points lying strictly inside the margin. The remaining non-zero α_i^* identify the support vectors. The weight vector in the expanded polynomial feature space is then recovered as

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i^* y_i \mathbf{x}_i,$$

and the bias b is computed from any support vector satisfying $0 < \alpha_i^*$.
The Python implementation below computes $\boldsymbol{\alpha}^*$ for our example dataset. Note that the labels $\mathbf{y}$ have been sign-adjusted to maintain consistency with the standard SVM formulation where the margin boundaries satisfy $y_i(\mathbf{w}^\top \mathbf{x}_i + b) = 1$.

```python
import numpy as np
from scipy.optimize import minimize

# My data ---------
X_poly = np.array([[0,1,0,0,1,1],[1,0,0,1,0,1],
                   [0,-1,0,0,1,1],[-1,0,0,1,0,1],
                   [0,0,0,0,0,1],[0,2,0,0,4,1],
                   [2,0,0,4,0,1],[0,-2,0,0,4,1],
                   [-2,0,0,4,0,1],
                   [2,2,4,4,4,1]])
y = np.array([1,1,1,1,1,-1,-1,-1,-1,-1])

# Compute Q matrix
K = X_poly @ X_poly.T
Q = np.outer(y, y) * K

# Solve QP
def objective(alpha):
    return 0.5 * alpha @ Q @ alpha - alpha.sum()

constraints = [
    {'type': 'eq', 'fun': lambda a: y @ a},
    {'type': 'ineq', 'fun': lambda a: a}
]
bounds = [(0, None) for _ in range(10)]
alpha0 = np.ones(10) * 0.1

result = minimize(objective, alpha0, bounds=bounds,
                  constraints=constraints, method='SLSQP')

# Extract with threshold
alpha = result.x
threshold = 1e-5  # Anything below this is too small, so
    just consider it a zero

for i, a in enumerate(alpha):
    if abs(a) > threshold:
        print(f"a{i}={a:10.6f}")
    else:
        print(f"a{i}={0}")
```

Solving for the alpha vector

article amsmath amssymb booktabs array
The solver yields the following alpha vector:

$$\boldsymbol{\alpha} = \begin{bmatrix} 0.126169 \\ 0.126170 \\ 0.096063 \\ 0.096065 \\ 0.000000 \\ 0.118632 \\ 0.118626 \\ 0.103607 \\ 0.103601 \\ 0.000000 \end{bmatrix}.$$

Eight of the ten coefficients are non-zero, identifying the corresponding data points as support vectors. The two zero coefficients correspond to the origin $(0, 0)$ and the point $(2, 2)$.
With $\boldsymbol{\alpha}$ determined, the weight vector $\mathbf{w}$ and the bias b are computed via:

$$\mathbf{w} = \mathbf{X}^\top(\boldsymbol{\alpha} \odot \mathbf{y}), \tag{1}$$

$$b = \frac{1}{n_S}\big(\mathbf{1}^\top \mathbf{y}_S - \mathbf{1}^\top \mathbf{X}_S \mathbf{w}\big), \tag{2}$$

as explained previously.
Computing $\mathbf{w}$:

$$\mathbf{X} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & -1 & 0 & 0 & 1 & 1 \\ -1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 & 4 & 1 \\ 2 & 0 & 0 & 4 & 0 & 1 \\ 0 & -2 & 0 & 0 & 4 & 1 \\ -2 & 0 & 0 & 4 & 0 & 1 \\ 2 & 2 & 4 & 4 & 4 & 1 \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix}.$$

First compute $\boldsymbol{\alpha} \odot \mathbf{y}$:

$$\boldsymbol{\alpha}^* \odot \mathbf{y} = \begin{bmatrix} 0.126169 \\ 0.126170 \\ 0.096063 \\ 0.096065 \\ 0.000000 \\ -0.118632 \\ -0.118626 \\ -0.103607 \\ -0.103601 \\ 0.000000 \end{bmatrix}.$$

Then, according to (1),

$$\mathbf{w} = \begin{bmatrix} 0 & 1 & 0 & -1 & 0 & 0 & 2 & 0 & -2 & 2 \\ 1 & 0 & -1 & 0 & 0 & 2 & 0 & -2 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 \\ 0 & 1 & 0 & 1 & 0 & 0 & 4 & 0 & 4 & 4 \\ 1 & 0 & 1 & 0 & 0 & 4 & 0 & 4 & 0 & 4 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0.126169 \\ 0.126170 \\ 0.096063 \\ 0.096065 \\ 0.000000 \\ -0.118632 \\ -0.118626 \\ -0.103607 \\ -0.103601 \\ 0.000000 \end{bmatrix} = \begin{bmatrix} 0.000055 \\ 0.000056 \\ 0 \\ -0.666673 \\ -0.666724 \\ 0.000001 \end{bmatrix}.$$

Computing b

The support-vector set is $S = \{1, 2, 3, 4, 6, 7, 8, 9\}$ ($n_S = 8$). Using (2) with the corresponding sub-matrices:

$$\mathbf{X}_S = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & -1 & 0 & 0 & 1 & 1 \\ -1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 2 & 0 & 0 & 4 & 1 \\ 2 & 0 & 0 & 4 & 0 & 1 \\ 0 & -2 & 0 & 0 & 4 & 1 \\ -2 & 0 & 0 & 4 & 0 & 1 \end{bmatrix}, \qquad \mathbf{y}_S = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix}.$$

Compute $\mathbf{1}^\top \mathbf{y}_S = 0$ (since there are four $+1$ and four -1 labels). Next,

$$\mathbf{1}^\top \mathbf{X}_S \mathbf{w} = \mathbf{1}^\top \begin{bmatrix} -0.666617 \\ -0.666617 \\ -0.666779 \\ -0.666727 \\ -2.666783 \\ -2.666581 \\ -2.667007 \\ -2.666801 \end{bmatrix} = -13.333962.$$

Thus,

$$b = \frac{0 - 13.333962}{8} = 1.66674525$$

Final Decision Boundary

The separating hyperplane in the six-dimensional polynomial feature space is:

$$\begin{bmatrix} 0.000055 \\ 0.000056 \\ 0 \\ -0.666673 \\ -0.666724 \\ 0.000001 \end{bmatrix}^T \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_1.x_2 \\ x_1^2 \\ x_2^2 \\ 1 \end{bmatrix} + 1.66674525 = 0$$

This describes a circle of radius $\sqrt{2.5} \approx 1.581$ in the original (x_1, x_2)-plane, which aligns the flattened graph shown at the very start.

From Explicit Features to Kernel Methods

The explicit polynomial feature construction used here, while pedagogically valuable, becomes computationally prohibitive for high-degree polynomials or many input features. Notice, however, that throughout our derivation, the data appears only in the form of inner products:

- The Gram matrix $\mathbf{X}\mathbf{X}^\top$ contains all pairwise dot products $\mathbf{x}_i \cdot \mathbf{x}_j$
- The $\mathbf{Q}$ matrix is built from $\mathbf{X}\mathbf{X}^\top$ via label weighting
- The weight vector recovery uses $\mathbf{X}^\top(\boldsymbol{\alpha} \odot \mathbf{y})$, which depends on $\mathbf{X}$

Kernel methods exploit this observation through the *kernel trick*: instead of explicitly computing the transformed features $\phi(\mathbf{x}) = [x_1, x_2, x_1x_2, x_1^2, x_2^2, 1]$, we work directly with a kernel function $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$ that computes inner products in the high-dimensional space without ever constructing $\phi(\mathbf{x})$.
For our polynomial example, the kernel function would be:

$$K(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{x}^\top \mathbf{z})^2,$$

which, when expanded, yields exactly the same inner products as our explicit six-dimensional mapping. The dual optimization becomes:

$$\min_{\boldsymbol{\alpha}} \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) - \sum_{i=1}^{n} \alpha_i,$$

and the decision function transforms to:

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b.$$

The support vectors and their coefficients α_i remain identical to those we computed; only their representation changes from explicit weights in R^6 to a weighted sum of kernel evaluations.

Thus, while we have worked with explicit features for clarity, the entire mathematical development transfers directly to kernel-based implementations—the most common form of SVMs in practice. The dual formulation, support-vector selection, and margin optimization remain unchanged; only the method of computing inner products differs.

Conclusion

This example demonstrated the complete mathematical workflow of a hard-margin support vector classifier support vector classifier with polynomial feature transformation. Beginning with a dataset that is not linearly separable in its original two-dimensional space, I applied a second-degree polynomial mapping to create a six-dimensional feature space where linear separation becomes possible.
The dual formulation of the SVM transformed the primal optimization problem into a quadratic program, which was solved numerically to obtain the lagrange multipliers α. These coefficients identified the support vectors. From α, we recovered the weight vector w and bias b, yielding the explicitly decision boundary in the polynomial feature space. The final boundary simplifies to the circle $x_1^2 + x_2^2 = 2.5$.
While this example used explicit polynomial features for pedagogical clarity, production implementations typically employ kernel tricks that compute inner products in the transformed space without explicitly constructing the high-dimensional features. The kernel trick allows the same mathematical operations(calculating $\mathbf{Q}_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$ where K is a kernel function) while working entirely in the original feature space. The weight vector $\mathbf{w}$ and bias b become implicit in the kernel expansion, but the core dual quadratic programming formulation remains unchanged.
The mathematical foundations demonstrated here remain central to both explicit feature-based and kernel-based SVM implementations, providing a unified theoretical framework for maximum-margin classification.