

# Assignment 8: Linear SVM - The Widest Street Strategy

Data Science and Machine Learning Module

December 2025

## 1 Introduction

In previous assignments, you used Logistic Regression to find a "boundary" based on probability. In this assignment, we shift to a **Geometric Perspective**. Support Vector Machines (SVM) do not just look for a line that separates data; they look for the **widest possible margin** that keeps the two classes apart.

## 2 Conceptual Background

### 2.1 The Margin and the Support Vectors

The Margin is defined by two parallel lines. The distance between these lines is the **Margin**. The data points that lie exactly on these lines are called **Support Vectors**.

- If you move a non-support vector point, the street stays the same.
- If you move a Support Vector, the entire street must move or shrink.

### 2.2 The Hyperparameter $C$ (The Slack Penalty)

In the real world, data is often messy and overlapping. We use the parameter  $C$  to decide how "strict" we want to be.

- **Small  $C$  (Soft Margin):** We allow some points to be on the wrong side of the line if it means we can have a **wider margin**. This helps with generalization (Low Variance).
- **Large  $C$  (Hard Margin):** We are very strict. We want **zero mistakes**, even if it means the margin becomes very narrow or crooked (High Variance).

## 3 Mathematical Implementation: The Dual Form

To use advanced features later (Kernels), we solve the SVM in its **Dual Form**. Instead of finding the weights  $w$ , we find the **influence**  $\alpha_i$  of each point.

### 3.1 The Objective Function

You must implement the following function to be minimized:

$$\min_{\alpha} f(\alpha) = \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) - \sum_{i=1}^n \alpha_i$$

Subject to:

$$0 \leq \alpha_i \leq C \quad \text{and} \quad \sum_{i=1}^n \alpha_i y_i = 0$$

## 4 Task 1: The Optimizer Setup

You will use `scipy.optimize.minimize`. Unlike previous solvers, this one requires you to explicitly define **Bounds** and **Constraints**.

```
1 from scipy.optimize import minimize
2
3 def my_kernel(x1, x2, type='linear'):
4     if type == 'linear':
5         return np.dot(x1, x2)
6     pass
7
8 # 1. Complete the Dual Objective Function
9 def dual_objective(alpha, K, y):
10     # K is the Gram Matrix (dot product of all xi, xj)
11     # y is the vector of labels (+1 or -1)
12
13     # Hint: term2 = 0.5 * sum(alpha_i * alpha_j * y_i * y_j * K_ij)
14     # Vectorized hint: 0.5 * np.dot(alpha * y, np.dot(K, alpha * y))
15     for i in range(n):
16         for j in range(n):
17             term2 += alpha[i] * alpha[j] * y[i] * y[j] * K[i, j]
18     term2 = 0.5 * term2
19     term1 = np.sum(alpha)
20     return term2 - term1
21
22 # 2. Defining Constraints for the solver
23 # Bounds: Every alpha must be between 0 and C
24 bounds = [(0, C_val) for _ in range(len(y_train))]
25
26 # Equality Constraint: sum(alpha * y) = 0
27 cons = {'type': 'eq', 'fun': lambda a: np.dot(a, y_train)}
28
29 # INITIAL GUESS: Start by assuming everyone has 0 influence
30 initial_guess = np.zeros(len(y_train))
31
32 # THE OPTIMIZER CALL
33 # Students must fill in the blanks (...)
34 result = minimize(
35     fun=...,          # The function we want to minimize
36     x0=...,          # The starting values for alpha
37     args=(K, y_train), # Extra variables needed by the objective function
38     method='SLSQP',   # The specific 'solver' engine
39     bounds=...,       # The 0 to C limits
40     constraints=...    # The sum(alpha * y) = 0 rule
```

```

41 )
42
43 # Extract the optimized alphas
44 alpha_optimized = result.x
45
46 # TASK: Get Weights (w)
47 # w = sum over i (alpha_i * y_i * x_i)
48 w = np.zeros(X_train.shape[1]) # Start with 0s (2D vector)
49 for i in range(len(alpha_optimized)):
50     # Add the contribution of point i to the total weight vector
51     w += alpha_optimized[i] * y_train[i] * X_train[i]
52
53 # TASK: Get Bias (b)
54 # Pick a point that is a Support Vector (0 < alpha < C)
55 # For this point, y*(w.x + b) = 1. Therefore: b = y - (w.x)
56 b = 0
57 for i in range(len(alpha_optimized)):
58     if 0.0001 < alpha_optimized[i] < (C_val - 0.0001):
59         # We found a point on the margin! Calculate b and stop.
60         b = y_train[i] - np.dot(w, X_train[i])
61         break

```

## 5 Task 2: Experiments

### 5.1 Part 1: Perfectly Separable Data

Load `linear_separable.csv`. Split into 80% training and 20% validation.

1. Train the SVM using  $C = 1.0$ .
2. Plot the data points, the decision boundary ( $w \cdot x + b = 0$ ), and the gutters ( $w \cdot x + b = \pm 1$ ).
3. Highlight the Support Vectors (points where  $\alpha > 0$ ).

### 5.2 Part 2: Overlapping Data

Load `linear_overlapping.csv`.

1. **The C-Sweep:** Train three models with  $C = 0.01$ ,  $C = 1$ , and  $C = 100$ .
2. **Visualization:** Plot the three results side-by-side. Observe how the "Street Width" decreases as  $C$  increases.
3. **Validation:** Iterate  $C$  through values  $[10^{-3}, 10^{-2}, \dots, 10^3]$ .
4. **Report:** Plot **Validation Accuracy vs  $C$**  and identify the optimal  $C$ .

## 6 Submission Requirements

- Python code implementing the `dual_objective` and the `minimize` loop.
- A single plot showing the "Street" (boundary + margins) for the non-overlapping case.
- The Validation Accuracy plot for the overlapping case and the chosen optimal  $C$ .