Department of Electrical Engineering

Indian Institute of Technology, Kharagpur

Algorithms, AI and ML Laboratory (EE22202)

Spring, 2025-26

Report 3: Support Vector Machines

Name: Arijit Dey

Roll No: 24IE10001

Support Vector Machines

Three labeled datasets are given. The input variables lie in R^2. Matrix A contains a collection of vectors in R^2 with label 1 and matrix B contains a collection of vectors in R^2 with label -1.

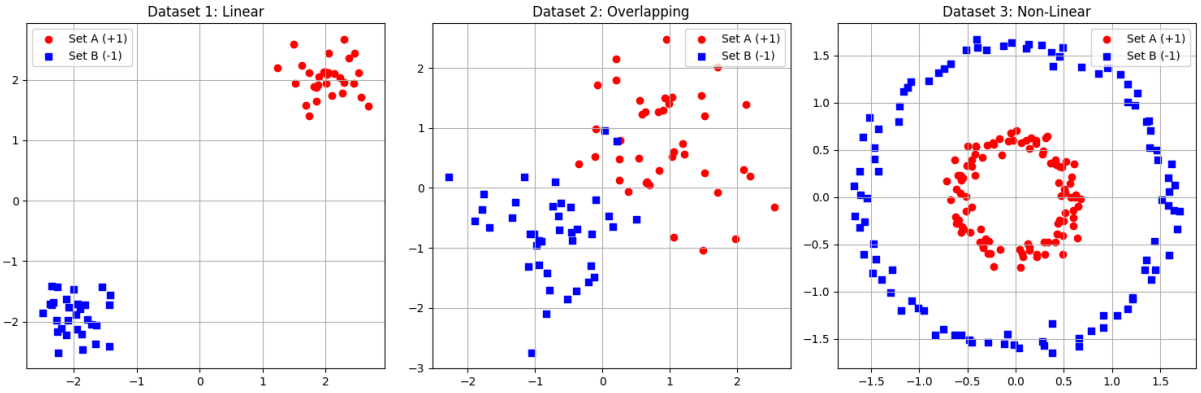
1 Visualizing the Datasets

We plot all the three datasets on the R^2 plane. Each point in set A is denoted as a red circle and set B as blue square.

Visualize datasets using `matplotlib`

```
import numpy as np
import matplotlib.pyplot as plt

def generate_svm_datasets():
    np.random.seed(0)
    A1 = np.random.randn(30, 2) * 0.3 + np.array([2, 2])
    B1 = np.random.randn(30, 2) * 0.3 + np.array([-2, -2])
    # Labels: A1 -> +1, B1 -> -1

    np.random.seed(1)
    A2 = np.random.randn(40, 2) * 0.8 + np.array([0.8, 0.8])
    B2 = np.random.randn(40, 2) * 0.8 + np.array([-0.8, -0.8])

    np.random.seed(2)
    n_points = 100
    theta = np.linspace(0, 2*np.pi, n_points)

    r_inner = 0.6
    A3 = np.c_[r_inner * np.cos(theta), r_inner * np.sin(theta)]
    A3 += np.random.randn(n_points, 2) * 0.07

    r_outer = 1.6
    B3 = np.c_[r_outer * np.cos(theta), r_outer * np.sin(theta)]
    B3 += np.random.randn(n_points, 2) * 0.07

    return (A1, B1), (A2, B2), (A3, B3)

(setA1, setB1), (setA2, setB2), (setA3, setB3) = generate_svm_datasets()

plt.figure(figsize=(15, 5))
datasets = [(setA1, setB1, "Dataset 1: Linear"),
            (setA2, setB2, "Dataset 2: Overlapping"),
            (setA3, setB3, "Dataset 3: Non-Linear")]

for i, (A, B, title) in enumerate(datasets):
    plt.subplot(1, 3, i+1)
    plt.scatter(A[:, 0], A[:, 1], c='red', marker='o', label='Set A (+1)')
    plt.scatter(B[:, 0], B[:, 1], c='blue', marker='s', label='Set B (-1)')
    plt.title(title)
    plt.legend()
```

```
    plt.grid(True)

plt.tight_layout()
plt.savefig("fig1.png", bbox_inches='tight')
plt.show()
```



2 Checking the Existance of a Linear Classifier

For a dataset to be separable by linear classifier, there must exist a weight vector w and a bias b such that for every point x_i with label $y_i \in \{1, -1\}$:

$$y_i(w^T x_i + b) \geq 1 \tag{1}$$

To find the separating hyperplane, we treat this as a Hard Margin SVM optimization problem. The best hyperplane (with the maximum margin) associated with the linear classifier can be obtained by solving the following optimization problem:

$$\min_{w \in R^2} \quad f(w) = \frac{1}{2} \|w\|_2^2$$

constrainted to above Equation 1 condition.

Linear Classifier using `cvxpy`

```
import cvxpy as cp

def solve_linear_svm(setA, setB):
    # Prepare data: A is +1, B is -1
    X = np.vstack([setA, setB])
    y = np.hstack([np.ones(len(setA)), -np.ones(len(setB))])

    N, D = X.shape

    w = cp.Variable(D)
    b = cp.Variable()

    # Constraints: y_i * (w^T * x_i + b) >= 1
    constraints = [cp.multiply(y, X @ w + b) >= 1]

    # Objective: Minimize 0.5 * ||w||^2
    prob = cp.Problem(cp.Minimize(0.5 * cp.sum_squares(w)), constraints)

```

```
    try:
        prob.solve()
        if prob.status == cp.OPTIMAL:
            return w.value, b.value
        else:
            return None, None
    except:
        return None, None

# Test the datasets
datasets = [ (setA1, setB1, "Dataset 1"), (setA2, setB2, "Dataset 2"), (setA3, setB3,
"Dataset 3") ]

for A, B, name in datasets:
    w, b = solve_linear_svm(A, B)
    if w is not None:
        print(f"{name}: Linearly Separable!")
        print(f"   Hyperplane: {w[0]:.2f}x + {w[1]:.2f}y + {b:.2f} = 0")
    else:
        print(f"{name}: Not Linearly Separable.")
```

```
Result
Dataset 1: Linearly Separable!
   Hyperplane: 0.34x + 0.32y + -0.03 = 0
Dataset 2: Not Linearly Separable.
Dataset 3: Not Linearly Separable.
```

3 Finding Solution for the Primal and Dual problem and Verification of KKT conditions

The Primal problem defined as

$$\min_{w \in R^2} \quad f(w) = \frac{1}{2} \|w\|_2^2$$
$$\text{Subject to} \quad y_i(w^T x_i + b) \geq 1$$

is the direct formulation of the SVM goal i.e to find a hyperplane that maximizes the margin while correctly classifying data. Here we directly adjust the orientation and position of the hyperplane in the input feature space.

The Dual problem defined as

$$\max_{\lambda \geq 0} \quad \left(\sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j \hat{y}_i \hat{y}_j (\hat{x}_i)^T \hat{x}_j \right)$$
$$\text{Subject to} \quad \sum_{i=1}^{N} \lambda_i \lambda_i \hat{y}_i = 0 \quad 0 \leq \lambda_i \quad \forall i \in \{N\}$$

is usful for determining how much importance or weight each individual data point has in defining the final boundary.

Let (w^*, b^*) be the primal optimal solution and λ^* be the dual optimal solution. Then the KKT conditions are defined as:

1. $\lambda_i^* \geq 0, \quad \sum_{i=1}^{N} \lambda_i^* \hat{y}_i = 0$
2. $1 - \hat{y}_i\left((w^*)^T \hat{x}_i + b^*\right) \leq 0$
3. $\lambda_i^*\left[1 - \hat{y}_i\left((w^*)^T \hat{x}_i + b^*\right)\right] = 0$
4. $w^* = \sum_{i=1}^{N} \lambda_i^* \hat{x}_i \hat{y}_i$

Solution for Primal and Dual problem

```python
X = np.vstack([setA1, setB1])
y = np.hstack([np.ones(len(setA1)), -np.ones(len(setB1))])
N, d = X.shape

# --- PRIMAL PROBLEM ---
w_p = cp.Variable(d)
b_p = cp.Variable()
primal_obj = cp.Minimize(0.5 * cp.sum_squares(w_p))
primal_con = [cp.multiply(y, X @ w_p + b_p) >= 1]
prob_p = cp.Problem(primal_obj, primal_con)
prob_p.solve()

# --- DUAL PROBLEM ---
alpha = cp.Variable(N)
P = np.outer(y, y) * (X @ X.T)
dual_obj = cp.Maximize(cp.sum(alpha) - 0.5 * cp.quad_form(alpha, cp.psd_wrap(P)))
dual_con = [alpha >= 0, cp.sum(cp.multiply(alpha, y)) == 0]
prob_d = cp.Problem(dual_obj, dual_con)
prob_d.solve()

# --- PARAMETER RECOVERY ---
w_from_dual = np.sum((alpha.value[:, None] * y[:, None] * X), axis=0)
sv_indices = np.where(alpha.value > 1e-5)[0]
b_from_dual = y[sv_indices[0]] - w_from_dual @ X[sv_indices[0]]

print("-" * 30)
print("SVM SOLVER RESULTS (Dataset 1)")
print("-" * 30)
print(f"Primal weights (w): {w_p.value}")
print(f"Dual weights (w):   {w_from_dual}")
print(f"Primal bias (b):    {b_p.value:.4f}")
print(f"Dual bias (b):      {b_from_dual:.4f}")
print("\n" + "-" * 30)

# --- VERIFICATION OF KKT CONDITION ---
print("KKT CONDITION VERIFICATION")
print("-" * 30)

# 1. Stationarity
stationarity_err = np.linalg.norm(w_p.value - w_from_dual)
print(f"1. Stationarity (||w_p - w_d||): {stationarity_err:.2e}")

# 2. Primal Feasibility
```

```
min_margin = np.min(y * (X @ w_p.value + b_p.value))
print(f"2. Primal Feasibility (Min Margin >= 1): {min_margin:.4f}")

# 3. Dual Feasibility
min_alpha = np.min(alpha.value)
print(f"3. Dual Feasibility (Min alpha >= 0): {min_alpha:.2e}")

# 4. Complementary Slackness
# alpha_i * (y_i(w.T x_i + b) - 1) should be 0
margin_gap = y * (X @ w_p.value + b_p.value) - 1
slackness = np.abs(alpha.value * margin_gap)
print(f"4. Max Complementary Slackness: {np.max(slackness):.2e}")
print(f"   Number of Support Vectors: {len(sv_indices)}")
print("-" * 30)
```

```
Result
------------------------------
SVM SOLVER RESULTS (Dataset 1)
------------------------------
Primal weights (w): [0.33696807 0.31613427]
Dual weights (w):   [0.33696807 0.31613427]
Primal bias (b):    -0.0286
Dual bias (b):      -0.0286

------------------------------
KKT CONDITION VERIFICATION
------------------------------
1. Stationarity (||w_p - w_d||): 7.85e-17
2. Primal Feasibility (Min Margin >= 1): 1.0000
3. Dual Feasibility (Min alpha >= 0): -9.29e-23
4. Max Complementary Slackness: 2.37e-17
   Number of Support Vectors: 2
------------------------------
```

4 Plotting the Hyperplane of Dataset 1

```
def plot_svm_results(setA, setB, w, b, alpha):
    plt.figure(figsize=(10, 8))

    # Plot the data points
    plt.scatter(setA[:, 0], setA[:, 1], c='blue', label='Class +1 (A1)', alpha=0.7)
    plt.scatter(setB[:, 0], setB[:, 1], c='red', label='Class -1 (B1)', alpha=0.7)

    # Define plot limits
    ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # Create grid to evaluate model
    xx = np.linspace(xlim[0], xlim[1], 30)
```

```
    yy = np.linspace(ylim[0], ylim[1], 30)
    YY, XX = np.meshgrid(yy, xx)
    xy = np.vstack([XX.ravel(), YY.ravel()]).T

    # Calculate decision boundary and margins: Z = w[0]*x + w[1]*y + b
    Z = (xy @ w + b).reshape(XX.shape)

    # Plot decision boundary and margins
    # level 0 is the hyperplane, levels -1 and 1 are the margins
    ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
               linestyles=['--', '-', '--'])

    # Highlight Support Vectors (where alpha > 1e-5)
    X_all = np.vstack([setA, setB])
    sv_idx = np.where(alpha > 1e-5)[0]
    plt.scatter(X_all[sv_idx, 0], X_all[sv_idx, 1], s=150,
                linewidth=1.5, facecolors='none', edgecolors='green',
                label='Support Vectors')

    plt.title("SVM Linear Classifier: Hyperplane and Margins")
    plt.legend()
    plt.grid(True, linestyle=':', alpha=0.6)
    plt.savefig("fig2.png", bbox_inches='tight')
    plt.show()

# Run the plotting function using values from the previous code block
plot_svm_results(setA1, setB1, w_p.value, b_p.value, alpha.value)
```