

# Support Vector Machines (SVMs) Assignment

Oscar Moxon

March 2024

**Department of Engineering/Informatics, King's  
College London**

**Pattern Recognition, Neural Networks and Deep Learning  
(7CCSMPNN)**

**Assignment: Support Vector Machines (SVMs)**

## Q1.

Write down the first 7 digits of your student ID as  $s_1s_2s_3s_4s_5s_6s_7$ .

k = 23073895

## Q2.

Find  $R_1$ , which is the remainder of  $\frac{s_1+s_2+s_3+s_4+s_5+s_6+s_7}{4}$ . Table 1 shows the multi-class methods to be used corresponding to the value of  $R_1$  obtained.

$R_1$	Method
0	One against one
1	One against all
2	Binary decision tree
3	Binary coded

Table 1: Table 1:  $R_1$  and its corresponding multi-class method.

```
summed = 0
for i in range(len(str(k))):
    value = int(str(k)[i])
    summed += value
print("added:", summed)

r1 = summed%4
print("r1:", r1)
```

output of code:

```
added: 37
r1: 1
```

we will use One Against All method as we have remainder of 1

### Q3.

Create a linearly separable two-dimensional dataset of your own, which consists of 3 classes. List the dataset in the format as shown in Table 2. Each class should contain at least 10 samples and all three classes have the same number of samples.

```
import numpy as np
import pandas as pd

np.random.seed(0)

# Class 1: Centered around (2, 2)
class_1 = np.random.randn(10, 2) + [2, 2]

# Class 2: Centered around (8, 2)
class_2 = np.random.randn(10, 2) + [8, 2]

# Class 3: Centered around (5, 8)
class_3 = np.random.randn(10, 2) + [5, 8]

# Make new pandas dataframe with both X and Y combined as each class
class_1_rounded = np.round(class_1, 3)
class_2_rounded = np.round(class_2, 3)
class_3_rounded = np.round(class_3, 3)

df = pd.DataFrame({
    "Sample of Class 1": [f"({x}, {y})" for x, y in class_1_rounded],
    "Sample of Class 2": [f"({x}, {y})" for x, y in class_2_rounded],
    "Sample of Class 3": [f"({x}, {y})" for x, y in class_3_rounded]
})

df
```

Sample	Sample of Class 1	Sample of Class 2	Sample of Class 3
0	(3.764, 2.4)	(5.447, 2.654)	(3.951, 6.58)
1	(2.979, 4.241)	(8.864, 1.258)	(3.294, 9.951)
2	(3.868, 1.023)	(10.27, 0.546)	(4.49, 7.562)
3	(2.95, 1.849)	(8.046, 1.813)	(3.747, 8.777)
4	(1.897, 2.411)	(9.533, 3.469)	(3.386, 7.787)
5	(2.144, 3.454)	(8.155, 2.378)	(4.105, 8.387)
6	(2.761, 2.122)	(7.112, 0.019)	(4.489, 6.819)
7	(2.444, 2.334)	(7.652, 2.156)	(4.972, 8.428)
8	(3.494, 1.795)	(9.23, 3.202)	(5.067, 8.302)
9	(2.313, 1.146)	(7.613, 1.698)	(4.366, 7.637)

Table 2: Dataset: Linearly separable two-dimensional dataset

#### Q4.

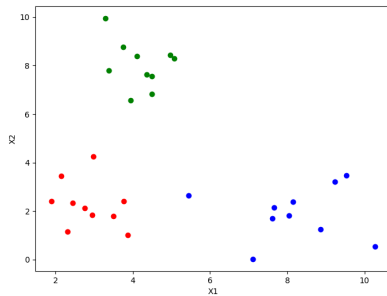
Plot the dataset in Q3 to show that the samples are linearly separable. Explain why your dataset is linearly separable.

```
import matplotlib.pyplot as plt

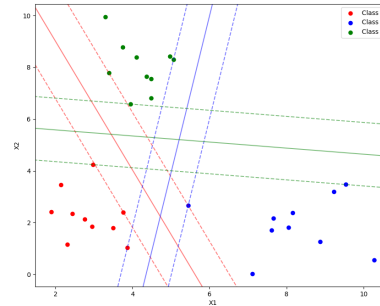
plt.figure(figsize=(8, 6))
plt.scatter(class_1[:, 0], class_1[:, 1], color='red', label='Class 1')
plt.scatter(class_2[:, 0], class_2[:, 1], color='blue', label='Class 2')
plt.scatter(class_3[:, 0], class_3[:, 1], color='green', label='Class 3')
plt.xlabel('X1')
plt.ylabel('X2')

plt.show()
```

Plot (a) displays the dataset with colours corresponding to classes. Plot (b) displays the dataset with the linear hyperplanes separating each class in one-against-all fashion.



(a) Linearly separable dataset (no hyperplanes). Classes: 1 = Red, 2 = Blue, 3 = Green.



(b) Linearly separable dataset with hyperplanes in one vs. all fashion.

The decision boundaries demonstrate the linear separability of the classes, with each boundary marking the divide between the selected class and all other classes (marking the gutter).

Because we are using one-against-all multi-class classification, we need to compare each class against all others. The dataset is linearly separable because the classes are clearly divided by linear hyperplanes between each class and all others.

For each hyperplane, all samples from the selected class are positive samples, while all other samples are negatives.

The dashed lines represent the margin on either side of the hyperplane, and the support vectors are indicated by the edge-marked circles.

This is linearly separable as there exists at least one hyperplane that can separate the classes without any errors (i.e., no misclassifications).

More specifically, this is because the decision function of the SVM (essentially a weighted sum of the attributes plus a bias) assigns the correct class based on the sign of the function (positive for one class, negative for the other). There are no instances where the function assigns the same sign to points from different classes.

Class 1 is linearly separated from the others by the steepest negative gradient hyperplane (red line).

Class 2 is linearly separated from the others by the blue hyperplane, the only positive gradient line.

Class 3 is linearly separated from the others by the green hyperplane, the shallow negative gradient line.

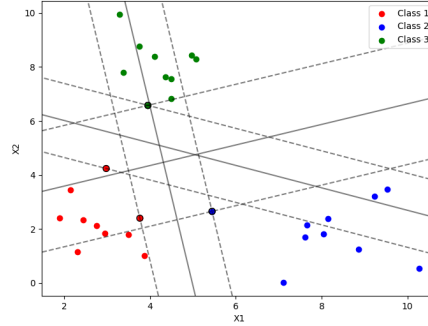


Figure 2: Linearly separable one against one.

For linearly separable data, each class exhibits distinct separation characteristics. Class 1 relies on two distinct support vectors for its hyperplanes, while Class 2 and Class 3 rely on just one support vector each.

Class 1 (Red) is linearly separated from Class 2 (Blue) by the steep negative gradient hyperplane, with intercept above 10.

Class 1 (Red) is linearly separated from Class 3 (Green) by the shallow negative gradient hyperplane, with intercept at approximately 6.3.

Class 2 (Blue) is linearly separated from Class 3 (Green) by the only positive gradient hyperplane, with intercept at approximately 3.5.

## Q5.

According to the method obtained in Q2, draw a block diagram at SVM level to show the structure of the multi-class classifier constructed by linear SVMs. Explain the design (e.g., number of inputs, number of outputs, number of SVMs used, class label assignment, etc.) and describe how this multi-class classifier works.

In our case, of One Against All, we need as many classifiers as there are classes. In the case of One Against One, we have  $R(R-1)/2$  binary SVMs.

Each SVM is a binary classifier that separates one class from all others, and all SVMs operate in parallel. These are (positive / negative):

SVM1 = A / Not A (B,C)  
SVM2 = B / Not B (A, C)  
SVM3 = C / Not C (A, B)

Each SVM will output a score indicating the confidence that a given sample belongs to its designated class against the other classes.

Once each of these classifiers are trained, we put a new sample in and run inference with each of the SVMs in parallel (3 at once), according to the information of each case, we apply a method to determine if the point is A, B, or C. Typically this is determined with Winner-Takes-All approach where a score is generated that represents the likelihood of the input belonging to one of the classes. The highest score determines the output.

If the sample does not strongly belong to any class, this is an indication that the model is unsure, and special handling might be required, such as assigning it to a default class or flagging it for manual review. This method creates a unique output during inference, while majority voting may not definitively produce an output.

So the block diagram at the SVM level represents the use all three of the hyperplanes when making a classification; if a value is Not B, Not C, and A, then it will output A as the final classification.

Meanwhile, in the case of One Against One, we have three pairwise SVMs for 3 classes: 1 v 2, 1 v 3, and 2 v 3. The collective output imposes voting on top.

Number of Inputs: One input feature vector.

Number of Outputs: One class label.

Number of SVMs: Three SVMs for three classes.

Class Label Assignment: The class label is assigned to the class corresponding to the SVM with the highest confidence score for the given input feature vector.

Decision Logic: run all SVMs, get confidence scores and select the class corresponding to the highest score.

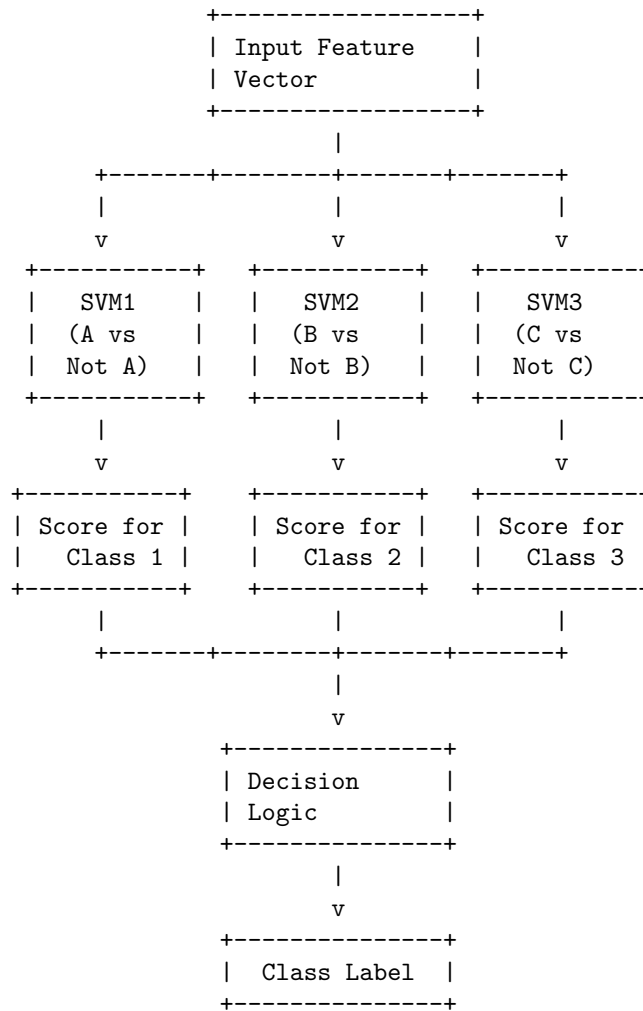


Figure 3: Block diagram at SVM level of multi-class OAA classifier for linear SVMs.

## Q6.

According to your dataset in Q3 and the design of your multi-class classifier in Q5, identify the support vectors of the linear SVMs by “inspection” and design their hyperplanes by hand. Show the calculations and explain the details of your design.

Use Tutorial -

We can identify the support vectors of the linear SVMs by inspection:

class 1: (3.764, 2.4), (3.868, 1.023), (2.979, 4.241)

class 2: (5.447, 2.654), (9.533, 3.469)

class 3: (3.386, 7.787), (5.067, 8.302), (3.951, 6.58)

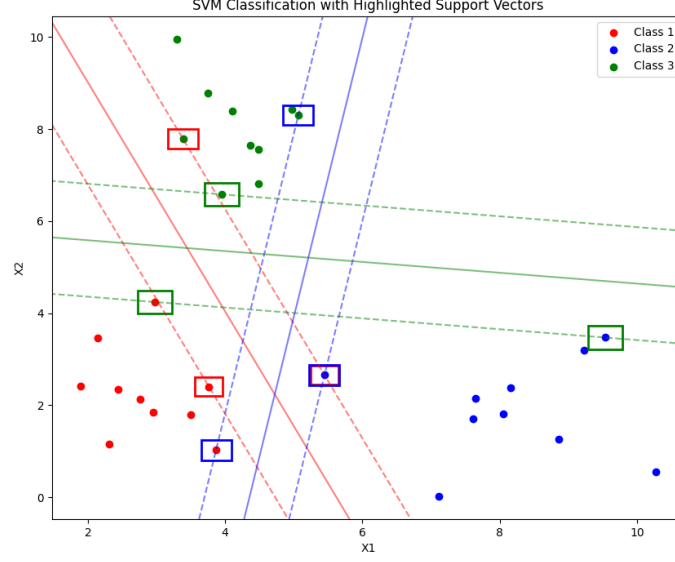


Figure 4: Figure 3: Identified Support Vectors (using Inspection).

Next, we will sort these according to the hyperplane they belong to:

Hyperplane 1: [(c1a', (3.764, 2.4)), (c2a', (5.447, 2.654)), (c3a', (3.386, 7.787))],  
Hyperplane 2: [(c1b', (3.868, 1.023)), (c2b', (5.447, 2.654)), (c3b', (5.067, 8.302))],  
Hyperplane 3: [(c1c', (2.979, 4.241)), (c2c', (9.533, 3.469)), (c3c', (3.951, 6.58))]

Given the support vectors and assuming a one-against-all SVM classifier, the equations we must solve for parameters of the first hyperplane can be written as:

$$\left( \begin{bmatrix} 3.764 \\ 2.4 \end{bmatrix} \lambda_1 - \begin{bmatrix} 5.447 \\ 2.654 \end{bmatrix} \lambda_2 - \begin{bmatrix} 3.386 \\ 7.787 \end{bmatrix} \lambda_3 \right)^T \begin{bmatrix} 3.764 \\ 2.4 \end{bmatrix} + w_0 = 1 \quad (1)$$

$$\left( \begin{bmatrix} 3.764 \\ 2.4 \end{bmatrix} \lambda_1 - \begin{bmatrix} 5.447 \\ 2.654 \end{bmatrix} \lambda_2 - \begin{bmatrix} 3.386 \\ 7.787 \end{bmatrix} \lambda_3 \right)^T \begin{bmatrix} 5.447 \\ 2.654 \end{bmatrix} + w_0 = -1 \quad (2)$$

$$\left( \begin{bmatrix} 3.764 \\ 2.4 \end{bmatrix} \lambda_1 - \begin{bmatrix} 5.447 \\ 2.654 \end{bmatrix} \lambda_2 - \begin{bmatrix} 3.386 \\ 7.787 \end{bmatrix} \lambda_3 \right)^T \begin{bmatrix} 3.386 \\ 7.787 \end{bmatrix} + w_0 = -1 \quad (3)$$

Expanding out equation 1, we get:

$$(1.1) \quad (3.764 \cdot 3.764)\lambda_1 + (5.447 \cdot 3.764)\lambda_2 + (3.386 \cdot 3.764)\lambda_3 + (2.4 \cdot 2.4)\lambda_1 + (2.654 \cdot 2.4)\lambda_2 + (7.787 \cdot 2.4)\lambda_3 + w_0 = 1$$

$$(1.2) \quad (14.1677 + 5.7600)\lambda_1 + (20.5025 + 6.3696)\lambda_2 + (12.7449 + 18.6888)\lambda_3 + w_0 = 1,$$



$$(1.3) \quad 19.9277\lambda_1 + 26.8721\lambda_2 + 31.4337\lambda_3 + w_0 = 1.$$

Expanding out equation 2, we get:

$$(2.1) \quad (5.447 \cdot 3.764)\lambda_1 + (2.654 \cdot 2.4)\lambda_1 - (5.447 \cdot 5.447)\lambda_2 - \\ (2.654 \cdot 2.654)\lambda_2 - (3.386 \cdot 5.447)\lambda_3 - (7.787 \cdot 2.654)\lambda_3 + w_0 = -1$$

$$(2.2) \quad 20.5025\lambda_1 + 6.3696\lambda_1 - 29.6698\lambda_2 - 7.0437\lambda_2 - 18.4435\lambda_3 - 20.6667\lambda_3 + w_0 = -1$$

$$(2.3) \quad 26.8696\lambda_1 - 36.7135\lambda_2 - 39.1102\lambda_3 + w_0 = -1$$

Expanding out equation 3:

$$(3.1) \quad (3.386 \cdot 3.764)\lambda_1 + (7.787 \cdot 2.4)\lambda_1 - (5.447 \cdot 3.386)\lambda_2 - \\ (2.654 \cdot 7.787)\lambda_2 - (3.386 \cdot 3.386)\lambda_3 - (7.787 \cdot 7.787)\lambda_3 + w_0 = -1$$

$$(3.2) \quad 12.7449\lambda_1 + 18.6888\lambda_1 - 18.4435\lambda_2 - \\ 20.6667\lambda_2 - 11.4650\lambda_3 - 60.6374\lambda_3 + w_0 = -1$$

$$(3.3) \quad 31.4337\lambda_1 - 39.1102\lambda_2 - 72.1024\lambda_3 + w_0 = -1$$

We will use the condition  $\sum_{i=1}^n \lambda_i y_i = 0$ , which is part of the Karush-Kuhn-Tucker (KKT) conditions for SVMs. This materialises as  $\lambda_1 - \lambda_2 - \lambda_3 = 0$ .

Given the support vectors and assuming a one-against-all SVM classifier, the equations to solve for the hyperplane coefficients can be written in matrix form as:

$$\begin{bmatrix} 19.9277 & -26.8721 & -31.4337 & 1 \\ 26.8696 & -36.7135 & -39.1102 & 1 \\ 31.4337 & -39.1102 & -72.1024 & 1 \\ 1 & -1 & -1 & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ w_0 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ -1 \\ 0 \end{bmatrix}$$

```
A = np.array([
    [19.9277, -26.8721, -31.4337, 1],
    [26.8696, -36.7135, -39.1102, 1],
    [31.4337, -39.1102, -72.1024, 1],
    [1, -1, -1, 0]
])

B = np.array([1, -1, -1, 0])

# Solve for X
X = np.linalg.solve(A, B)
```

output:

```
array([0.72829567, 0.67670285, 0.05159282, 6.29292223])
```

$$\lambda_1 = 0.7283$$

$$\lambda_2 = 0.6767$$

$$\lambda_3 = 0.0516$$

$$w_0 = 6.2929$$

Each hyperplane can be represented as a function of the input features, where the coefficients are derived from the support vectors weighted by their respective  $\lambda$  values, and the bias term acts as the intercept.

We can perform a variety of methods to solve for the Lagrangian multipliers. These provide insights into the importance of each support vector in defining the decision boundary. Non-zero Lagrangian multipliers indicate that the corresponding support vectors lie on the margin of their classes and are essential for constructing the hyperplane.

The bias term adjusts the distance of the hyperplane from the origin along the direction of the normal vector, effectively positioning the hyperplane in the feature space to optimally separate the classes.

We can now calculate the weight vector;

$$w = 0.7283 \cdot \begin{pmatrix} 3.764 \\ 2.4 \end{pmatrix} - 0.6767 \cdot \begin{pmatrix} 5.447 \\ 2.654 \end{pmatrix} - 0.0516 \cdot \begin{pmatrix} 3.386 \\ 7.787 \end{pmatrix}$$

```
# Calculating the weight vector w
lambda_values = [0.7283, 0.6767, 0.0516]
class_labels = [1, -1, -1]
support_vectors = [(3.764, 2.4), (5.447, 2.654), (3.386, 7.787)]

# Compute w
w = sum(l * y * np.array(x) for l, y, x in zip(lambda_values, class_labels, support_vectors))
```

output:

```
array([-1.1193813, -0.449851 ])
```

Plotting this on the graph, we perform:

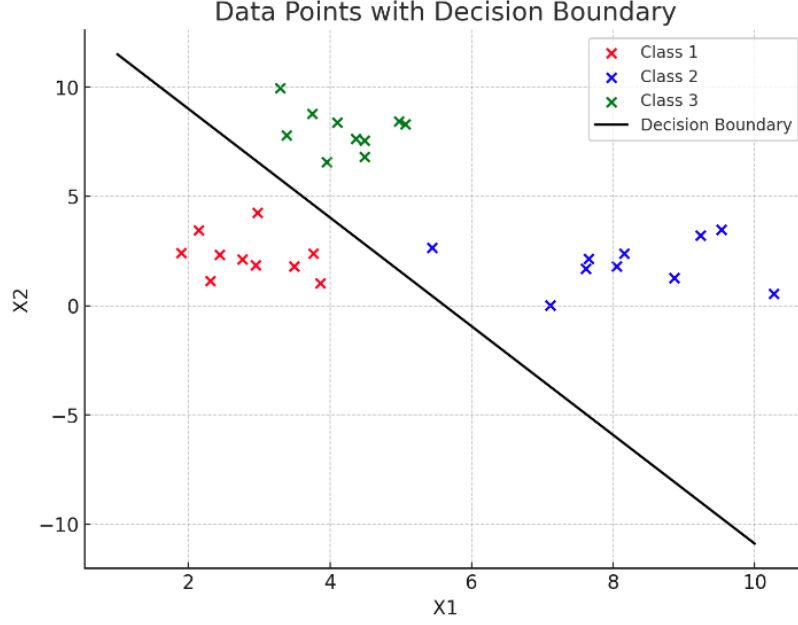


Figure 5: Strong predictive hyperplane designed by hand.

Now we repeat this process for the other two hyperplanes. For hyperplane 2:

$$\left( \begin{bmatrix} 3.868 \\ 1.023 \end{bmatrix} \lambda_1 - \begin{bmatrix} 5.447 \\ 2.654 \end{bmatrix} \lambda_2 - \begin{bmatrix} 5.067 \\ 8.302 \end{bmatrix} \lambda_3 \right)^T \begin{bmatrix} 3.868 \\ 1.023 \end{bmatrix} + w_0 = 1 \quad (4)$$

$$\left( \begin{bmatrix} 3.868 \\ 1.023 \end{bmatrix} \lambda_1 - \begin{bmatrix} 5.447 \\ 2.654 \end{bmatrix} \lambda_2 - \begin{bmatrix} 5.067 \\ 8.302 \end{bmatrix} \lambda_3 \right)^T \begin{bmatrix} 5.447 \\ 2.654 \end{bmatrix} + w_0 = -1 \quad (5)$$

$$\left( \begin{bmatrix} 3.868 \\ 1.023 \end{bmatrix} \lambda_1 - \begin{bmatrix} 5.447 \\ 2.654 \end{bmatrix} \lambda_2 - \begin{bmatrix} 5.067 \\ 8.302 \end{bmatrix} \lambda_3 \right)^T \begin{bmatrix} 5.067 \\ 8.302 \end{bmatrix} + w_0 = -1 \quad (6)$$

Expanding out equation 5:

$$(4.1) \quad (3.868 \cdot 3.868)\lambda_1 - (5.447 \cdot 3.868)\lambda_2 - (5.067 \cdot 3.868)\lambda_3 + (1.023 \cdot 1.023)\lambda_1 - (2.654 \cdot 1.023)\lambda_2 - (8.302 \cdot 1.023)\lambda_3 + w_0 = 1$$

$$(4.2) \quad (14.9614 + 1.0465)\lambda_1 - (21.0689 + 2.7150)\lambda_2 - (19.5992 + 8.4929)\lambda_3 + w_0 = 1,$$

$$(4.3) \quad 16.0079\lambda_1 - 23.7839\lambda_2 - 28.0921\lambda_3 + w_0 = 1.$$

Expanding out equation 6:

$$(5.1) \quad (3.868 \cdot 5.447)\lambda_1 + (1.023 \cdot 2.654)\lambda_1 - (5.447 \cdot 5.447)\lambda_2 - \\ (2.654 \cdot 2.654)\lambda_2 - (5.067 \cdot 5.447)\lambda_3 - (8.302 \cdot 2.654)\lambda_3 + w_0 = -1$$

$$(5.2) \quad (21.0699 + 2.7150)\lambda_1 - (29.6698 + 7.0437)\lambda_2 - (27.5999 + 22.0335)\lambda_3 + w_0 = -1$$

$$(5.3) \quad 23.7849\lambda_1 - 36.7135\lambda_2 - 49.6334\lambda_3 + w_0 = -1.$$

Expanding out equation 7:

$$(6.1) \quad (3.868 \cdot 5.067)\lambda_1 + (1.023 \cdot 8.302)\lambda_1 - (5.447 \cdot 5.067)\lambda_2 - \\ (2.654 \cdot 8.302)\lambda_2 - (5.067 \cdot 5.067)\lambda_3 - (8.302 \cdot 8.302)\lambda_3 + w_0 = -1$$

$$(6.2) \quad (19.5992 + 8.4929)\lambda_1 - (27.5999 + 22.0335)\lambda_2 - (25.6745 + 68.9232)\lambda_3 + w_0 = -1$$

$$(6.3) \quad 19.5992\lambda_1 - 49.6334\lambda_2 - 94.5977\lambda_3 + w_0 = -1.$$

$$\begin{bmatrix} 16.0079 & -23.7839 & -28.0921 & 1 \\ 23.7849 & -36.7135 & -49.6334 & 1 \\ 19.5992 & -49.6334 & -94.5977 & 1 \\ 1 & -1 & -1 & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ w_0 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ -1 \\ 0 \end{bmatrix}$$

For the final hyperplane:

$$\left( \begin{bmatrix} 2.979 \\ 4.241 \end{bmatrix} \lambda_1 - \begin{bmatrix} 9.533 \\ 3.469 \end{bmatrix} \lambda_2 - \begin{bmatrix} 3.951 \\ 6.58 \end{bmatrix} \lambda_3 \right)^T \begin{bmatrix} 2.979 \\ 4.241 \end{bmatrix} + w_0 = 1 \quad (7)$$

$$\left( \begin{bmatrix} 2.979 \\ 4.241 \end{bmatrix} \lambda_1 - \begin{bmatrix} 9.533 \\ 3.469 \end{bmatrix} \lambda_2 - \begin{bmatrix} 3.951 \\ 6.58 \end{bmatrix} \lambda_3 \right)^T \begin{bmatrix} 9.533 \\ 3.469 \end{bmatrix} + w_0 = -1 \quad (8)$$

$$\left( \begin{bmatrix} 2.979 \\ 4.241 \end{bmatrix} \lambda_1 - \begin{bmatrix} 9.533 \\ 3.469 \end{bmatrix} \lambda_2 - \begin{bmatrix} 3.951 \\ 6.58 \end{bmatrix} \lambda_3 \right)^T \begin{bmatrix} 3.951 \\ 6.58 \end{bmatrix} + w_0 = -1 \quad (9)$$

Expanding out equation 8:

$$(7.1) \quad (2.979 \cdot 2.979)\lambda_1 - (9.533 \cdot 2.979)\lambda_2 - (3.951 \cdot 2.979)\lambda_3 + \\ (4.241 \cdot 4.241)\lambda_1 - (3.469 \cdot 4.241)\lambda_2 - (6.58 \cdot 4.241)\lambda_3 + w_0 = 1$$

$$(7.2) \quad (8.8744 + 17.9860)\lambda_1 - (28.3988 + 14.7120)\lambda_2 - (11.7700 + 27.9057)\lambda_3 + w_0 = 1,$$

$$(7.3) \quad 26.8477\lambda_1 - 43.1128\lambda_2 - 39.6766\lambda_3 + w_0 = 1.$$

Expanding out equation 9:

$$(8.1) \quad (2.979 \cdot 9.533)\lambda_1 - (9.533 \cdot 9.533)\lambda_2 - (3.951 \cdot 9.533)\lambda_3 + \\ (4.241 \cdot 3.469)\lambda_1 - (3.469 \cdot 3.469)\lambda_2 - (6.58 \cdot 3.469)\lambda_3 + w_0 = 1$$

$$(8.2) \quad (28.3988 + 14.7120)\lambda_1 - (90.8781 + 12.0340)\lambda_2 - (37.6649 + 22.8260)\lambda_3 + w_0 = 1,$$

$$(8.3) \quad 43.1108\lambda_1 - 102.9121\lambda_2 - 60.4909\lambda_3 + w_0 = 1.$$

Expanding out equation 10:

$$(9.1) \quad (2.979 \cdot 3.951)\lambda_1 - (9.533 \cdot 3.951)\lambda_2 - (3.951 \cdot 3.951)\lambda_3 + \\ (4.241 \cdot 6.58)\lambda_1 - (3.469 \cdot 6.58)\lambda_2 - (6.58 \cdot 6.58)\lambda_3 + w_0 = 1$$

$$(9.2) \quad (11.7700 + 27.9058)\lambda_1 - (37.6649 + 22.8260)\lambda_2 - (15.6104 + 43.2964)\lambda_3 + w_0 = 1,$$

$$(9.3) \quad 39.6758\lambda_1 - 60.4909\lambda_2 - 58.9068\lambda_3 + w_0 = 1.$$

$$\begin{bmatrix} 26.8477 & -43.1128 & -39.6766 & 1 \\ 43.1108 & -102.9121 & -60.4909 & 1 \\ 39.6758 & -60.4909 & -58.9068 & 1 \\ 1 & -1 & -1 & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ w_0 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ -1 \\ 0 \end{bmatrix}$$

```
# Define the bias term from previous calculation

# Plot the data points
plt.figure(figsize=(8, 6))
plt.scatter(class_1[:, 0], class_1[:, 1], color='red', label='Class 1')
plt.scatter(class_2[:, 0], class_2[:, 1], color='blue', label='Class 2')
plt.scatter(class_3[:, 0], class_3[:, 1], color='green', label='Class 3')

# Plot the decision boundary: w^T x + b = 0
# For a 2D plot lets rearrange this equation to y = (-w_1/w_2)x - (b/w_2)
x_values = np.linspace(1, 10, 300)
```

```

y_values = (-w[0]/w[1]) * x_values - (b/w[1])

plt.plot(x_values, y_values, 'k-', label='Decision Boundary')

plt.xlabel('X1')
plt.ylabel('X2')
plt.legend()
plt.title('Data Points with Decision Boundary')
plt.show()

```

We can then plot these hyperplanes on a single plot as:

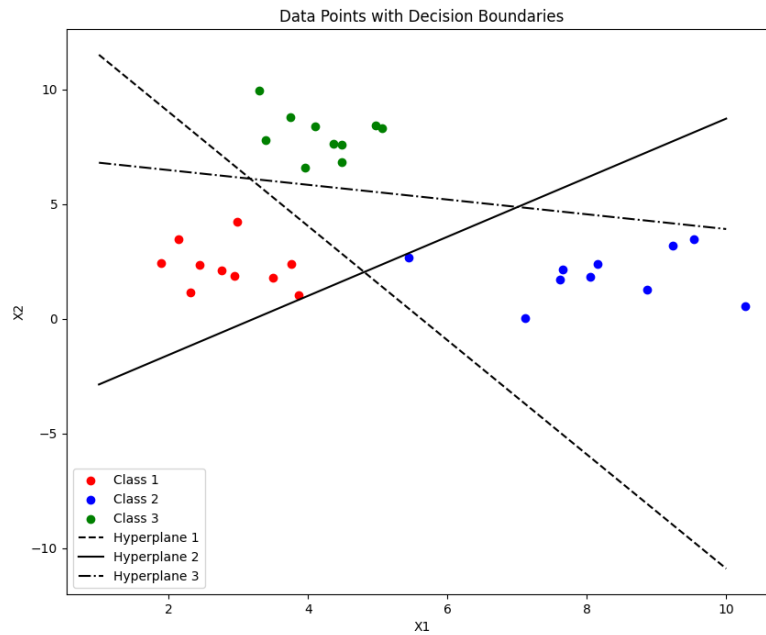


Figure 6: All predicted hyperplanes.

## Q7.

Produce a test dataset by averaging the samples for each row in Table 2, i.e., (sample of class 1 + sample of class 2 + sample of class 3)/3. Summarise the results in the form of Table 3, where N is the number of SVMs in your design and “Classification” is the class determined by your multi-class classifier. Explain how to get the “Classification” column using one test sample. Show the calculations for one or two samples to demonstrate how to get the contents in the table.

```
class_1_avg = np.mean(class_1, axis=0)
class_2_avg = np.mean(class_2, axis=0)
class_3_avg = np.mean(class_3, axis=0)
print("Class 1 Average:", class_1_avg, "\nClass 2 Average:", class_2_avg, "\nClass 3
    ↪ Average:", class_3_avg)
```

output:

```
Class 1 Average: [2.8614 2.2775]
Class 2 Average: [8.1922 1.9193]
Class 3 Average: [4.1867 8.023 ]
```

```
w_vectors = [np.array([-6.41420604, 4.98189282]), np.array([-1.1193813, -0.449851]), np.
    ↪ array([10.38442692, 32.33282478])]
biases = [20.7131053, 6.29292223, -230.11026363]

# Averaged class samples
class_averages = [np.array([2.8614, 2.2775]), np.array([8.1922, 1.9193]), np.array([4.1867,
    ↪ 8.023])]

# Classify each averaged sample
classifications = []
for avg in class_averages:
    decision_values = [np.dot(w, avg) + b for w, b in zip(w_vectors, biases)]
    classified_as = np.argmax(decision_values) + 1 # +1 to match class numbering starting
    ↪ from 1
    classifications.append(classified_as)

# Results
for i, classification in enumerate(classifications, start=1):
    print(f"Class {i} Average Classified as: Class {classification}")
```

Test Sample	Output of SVM 1	Output of SVM 2	Output of SVM 3	Classification
(2.86 2.28)	1	-1	-1	Class 1
(8.19 1.92)	-1	1	-1	Class 2
(4.19 8.02)	-1	-1	1	Class 3

Table 3: Table 3: Summary of classification accuracy.

```

# Hyperplane parameters (w vectors and b biases)
w_vectors = [np.array([-1.1193813, -0.449851])]
biases = [6.29292223]

# Calculate and print the classification for the Class 1 Average
decision_values_class_1 = [np.dot(w, class_1_avg) + b for w, b in zip(w_vectors, biases)]
classification_class_1 = np.argmax(decision_values_class_1) + 1 # Class numbers start at 1
print(f"Class 1 Average is classified as Class {classification_class_1} with decision
    ↪ values {decision_values_class_1}")

```

output:

Class 1 Average is classified as Class 1 with decision values [2.0653889256800007]

```

# Given hyperplane parameters
w_vectors = [np.array([-6.41420604, 4.98189282]), np.array([-1.1193813, -0.449851]), np.
    ↪ array([10.38442692, 32.33282478])]
biases = [20.7131053, 6.29292223, -230.11026363]

# Averaged class samples
class_1_avg = np.array([2.8614, 2.2775])
class_2_avg = np.array([8.1922, 1.9193])
class_3_avg = np.array([4.1867, 8.023])

# Function to calculate the decision values and classification
def classify_sample(sample, w_vectors, biases):
    decision_values = [np.dot(w, sample) + b for w, b in zip(w_vectors, biases)]
    classification = np.argmax(decision_values) + 1 # +1 as class indices start from 1
    return classification, decision_values

# Calculate the classification for the averaged samples
class_1_classification, class_1_d_values = classify_sample(class_1_avg, w_vectors, biases)
class_2_classification, class_2_d_values = classify_sample(class_2_avg, w_vectors, biases)
class_3_classification, class_3_d_values = classify_sample(class_3_avg, w_vectors, biases)

(class_1_classification, class_1_d_values), (class_2_classification, class_2_d_values), (
    ↪ class_3_classification, class_3_d_values)

```

output:

```

((1, [13.705757034693999, 2.0653889256800007, -126.75825600466199]),
 (2, [-22.271606531462, -3.7406722801599983, -82.982570815722]),
 (3, [33.828474967191994, -2.00274603171, 72.77246976590405]))

```

In a one-vs-all (OvA) SVM classification setting, the classification of a sample is determined by the SVM with the highest decision value; the farthest from the decision boundary in the positive direction. This is because a higher decision value indicates that the sample is more likely to belong to the class that SVM is responsible for.

There is some discrepancy about the signs of the values in these results which draw attention to some of the decision boundaries and their classification accuracies; strong



classifiers would have positive, high values for the class they are classifying. So while these are accurate, they could do with improved precision techniques.

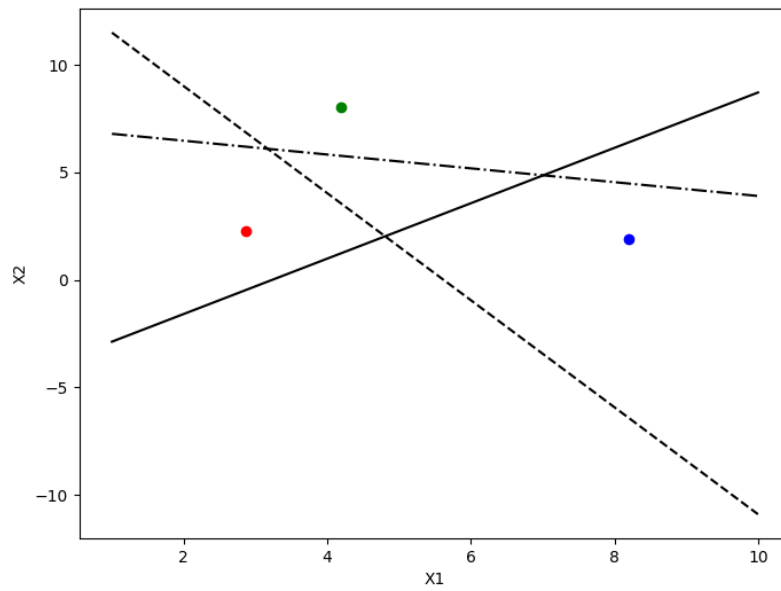


Figure 7: Averaged classes as input to multi-class SVM.