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 $s_1 s_2 s_3 s_4 s_5 s_6 s_7$.

```
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.

```python
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)

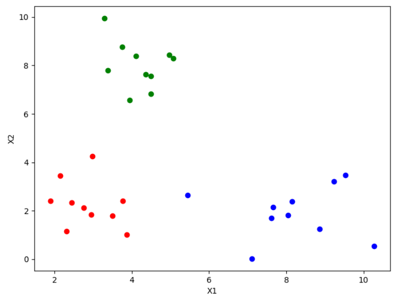
Table 2: Dataset: Linearly separable two-dimensional dataset

Q4.

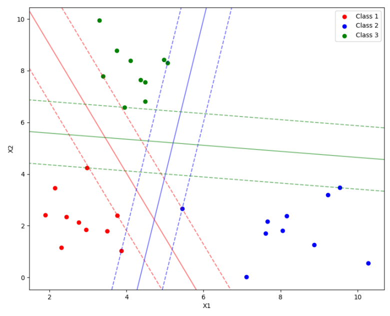
Plot the dataset in Q3 to show that the samples are linearly separable. Explain why your dataset is linearly separable.

```python
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.

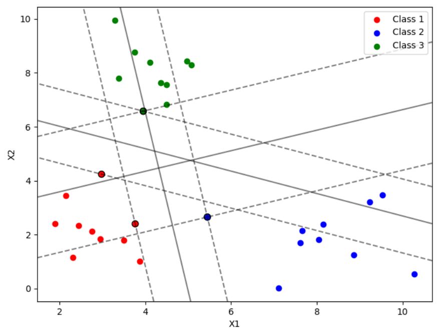


Figure 2: Linearly separable one against one.

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 f 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.

```
              +-------------------+
              | Input Feature     |
              | Vector            |
              +-------------------+
                        |
      +-------+---------+-------+-------+
      |                 |               |
      v                 v               v
 +-----------+    +-----------+   +-----------+
 |   SVM1    |    |   SVM2    |   |   SVM3    |
 |  (A vs    |    |  (B vs    |   |  (C vs    |
 |  Not A)   |    |  Not B)   |   |  Not C)   |
 +-----------+    +-----------+   +-----------+
      |                 |               |
      v                 v               v
+-----------+     +-----------+   +-----------+
| Score for |     | Score for |   | Score for |
|   Class 1 |     |   Class 2 |   |   Class 3 |
+-----------+     +-----------+   +-----------+
      |                 |               |
      +-------+---------+-------+-------+
                        |
                        v
               +----------------+
               | Decision       |
               | Logic          |
               +----------------+
                        |
                        v
               +----------------+
               |  Class Label   |
               +----------------+
```

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: [[5.447 2.654] [3.386 7.787] [3.764 2.4]]
class 2: [[3.868 1.023] [5.067 8.302] [5.447 2.654]]
class 3: [[2.979 4.241] [9.533 3.469] [3.951 6.58]]

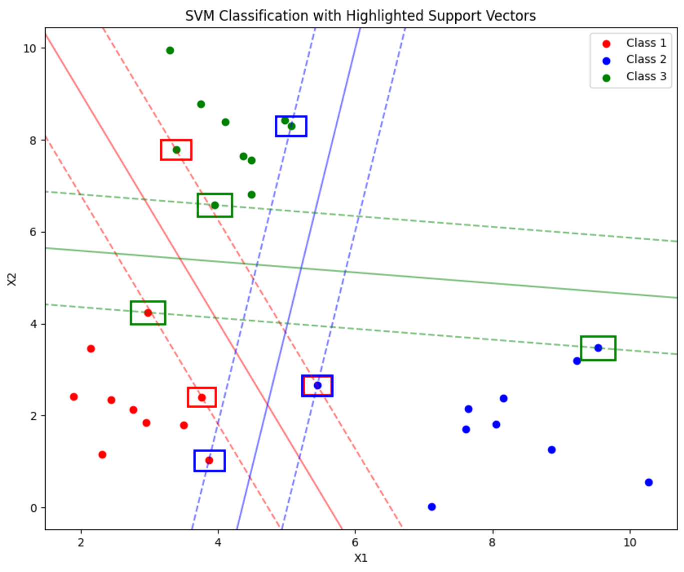


Figure 4: Figure 3: Identified Support Vectors (using Inspection).

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', (5.447, 2.654)), ('c2b', (5.447, 2.654)), ('c3b', (5.067, 8.302))],
Hyperplane 3: [('c1c', (3.951, 6.58)), ('c2c', (9.533, 3.469)), ('c3c', (2.979, 4.241))]

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:

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

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

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

Expanding out equation 1, we get:

$$\begin{aligned}(1.1)\quad &(3.764\cdot 3.764)\lambda_1+(5.447\cdot 3.764)\lambda_2+(3.386\cdot 3.764)\lambda_3+\\ &(2.4\cdot 2.4)\lambda_1+(2.654\cdot 2.4)\lambda_2+(7.787\cdot 2.4)\lambda_3+w_0=1\end{aligned}$$

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

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

Expanding out equation 2, we get:

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

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

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

Expanding out equation 3:

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

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

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

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 λ 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;

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

```python
def calculate_weight_vector(lambda_values, support_vectors, labels):
    """
    Calculate the weight vector w.

    Parameters:
    - lambda_values: The Lagrange multipliers.
    - support_vectors: The support vectors with their labels.

    Returns:
    - The weight vector w.
    """
    w = np.sum([l * label * sv for l, label, sv in zip(lambda_values, labels,
        ↪ support_vectors)], axis=0)
    return w

lambda_values = [0.72891037 0.67733387 0.0515765]
bias = 6.29712509

w = calculate_weight_vector(lambda_values, support_vectors_1, labels)
print("Weight Vector:", w)
```

output:

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

Plotting this on the graph, we get:

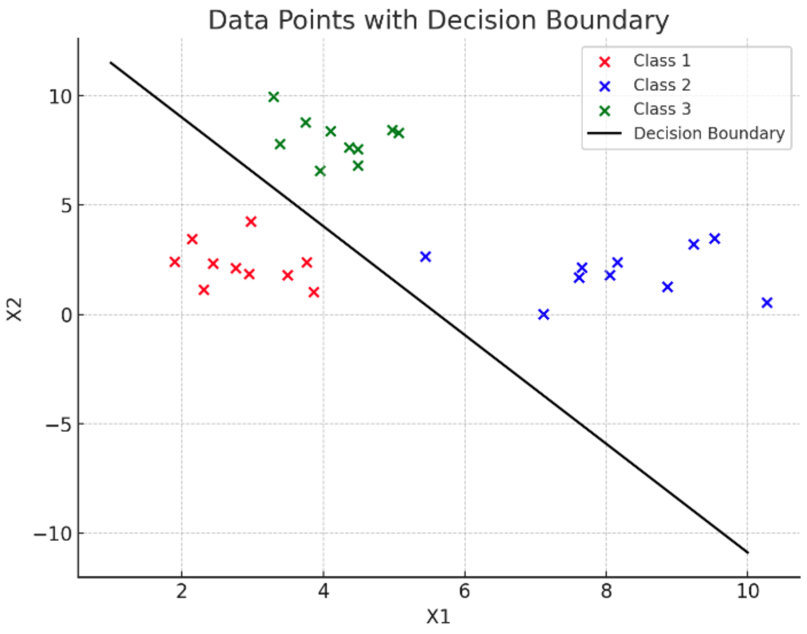


Figure 5: Strong predictive hyperplane designed by hand.

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

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

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

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

Expanding out equation 5:

$$\begin{aligned}(4.1)\quad &(3.868\cdot 3.868)\lambda_1-(5.447\cdot 3.868)\lambda_2-(5.067\cdot 3.868)\lambda_3+\\ &(1.023\cdot 1.023)\lambda_1-(2.654\cdot 1.023)\lambda_2-(8.302\cdot 1.023)\lambda_3+w_0=1\end{aligned}$$

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

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

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:

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

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

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

$$\begin{bmatrix} 36.7135 & -49.6334 & -23.7840 & 1 \\ 49.6334 & -94.5976 & -28.0921 & 1 \\ 23.784038 & -28.092102 & -16.007953 & 1 \\ 1 & -1 & -1 & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ w_0 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ -1 \\ 0 \end{bmatrix}$$

```
Parameters: [ 1.19643182 0.3026235 0.89380832 -6.6466081 ]
Bias: -6.646608104951451
Weight Vector: [ 1.52632027 -0.25141613]
w1: 1.526320270179029
w2: -0.2514161291310145
```

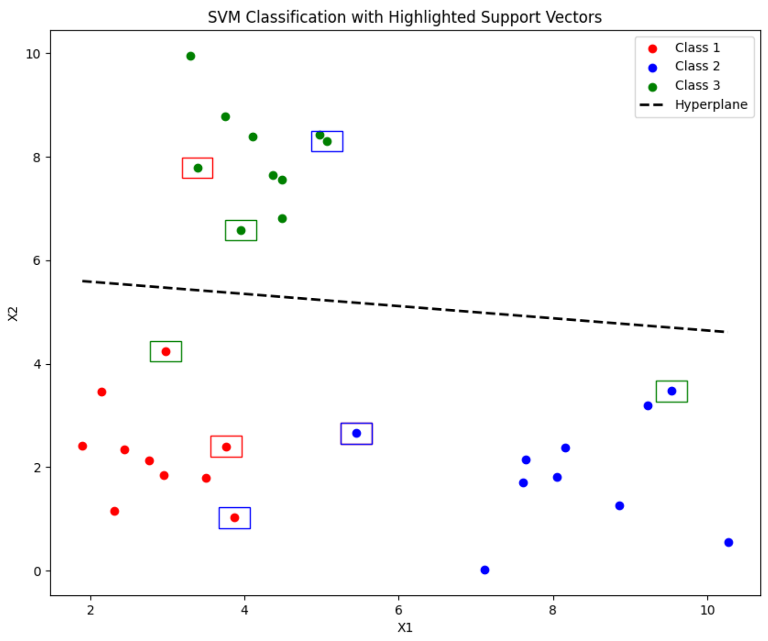


Figure 6: Predictive hyperplane designed by hand (svm 2).

For the final hyperplane:

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

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

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

Expanding out equation 8:

$$\begin{aligned}(7.1)\quad &(2.979\cdot 2.979)\lambda_1-(9.533\cdot 2.979)\lambda_2-(3.951\cdot 2.979)\lambda_3+\\ &(4.241\cdot 4.241)\lambda_1-(3.469\cdot 4.241)\lambda_2-(6.58\cdot 4.241)\lambda_3+w_0=1\end{aligned}$$

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

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

Expanding out equation 9:

$$(8.1)\quad \begin{aligned}&(2.979 \cdot 9.533)\lambda_1 - (9.533 \cdot 9.533)\lambda_2 - (3.951 \cdot 9.533)\lambda_3 +\\ &(4.241 \cdot 3.469)\lambda_1 - (3.469 \cdot 3.469)\lambda_2 - (6.58 \cdot 3.469)\lambda_3 + w_0 = 1\end{aligned}$$

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

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

Expanding out equation 10:

$$(9.1)\quad \begin{aligned}&(2.979 \cdot 3.951)\lambda_1 - (9.533 \cdot 3.951)\lambda_2 - (3.951 \cdot 3.951)\lambda_3 +\\ &(4.241 \cdot 6.58)\lambda_1 - (3.469 \cdot 6.58)\lambda_2 - (6.58 \cdot 6.58)\lambda_3 + w_0 = 1\end{aligned}$$

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

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

$$\begin{bmatrix} 58.9068 & -60.4909 & -39.6766 & 1 \\ 60.4909 & -102.9121 & -43.1108 & 1 \\ 39.6758 & -43.1108 & -26.8605 & 1 \\ 1 & -1 & -1 & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ w_0 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ -1 \\ 0 \end{bmatrix}$$

```
Parameters: [ 0.33685638 0.03530754 0.30154884 -4.74315254]
Bias: -4.743152537376741
Weight Vector: [0.09601877 0.8151645 ]
w1: 0.09601876594741798
w2: 0.8151644974344217
```

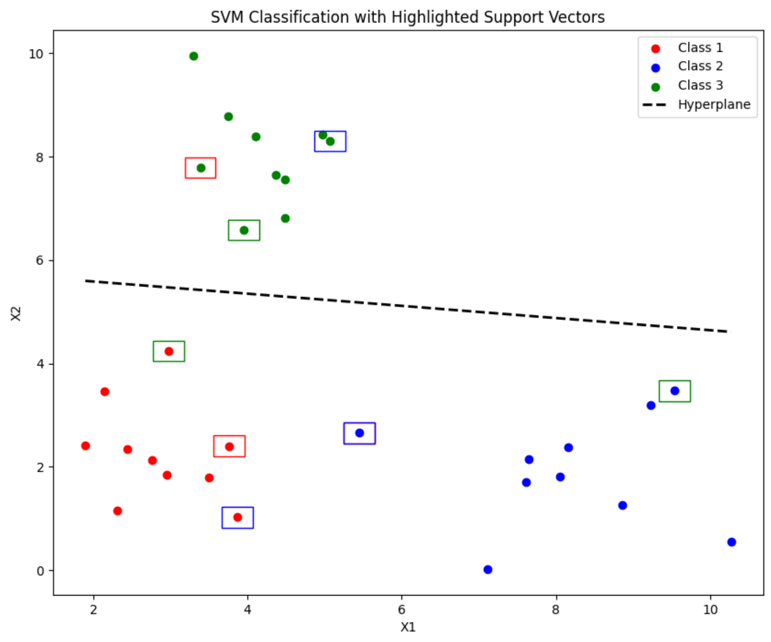


Figure 7: Predictive hyperplane designed by hand (svm 3).

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

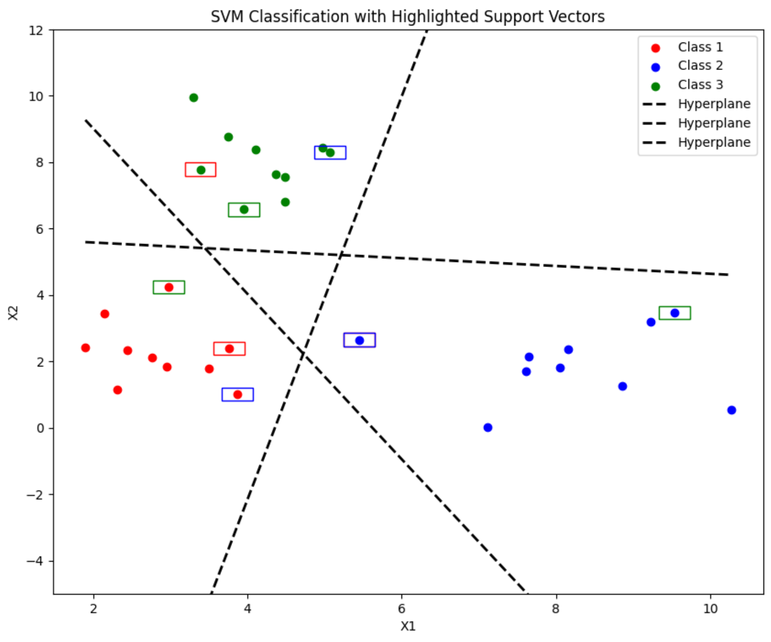


Figure 8: All predicted hyperplanes.

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.

To get the averaged dataset we can use:

```
average_points = []
for i, (class_1_point, class_2_point, class_3_point) in enumerate(zip(class_1, class_2,
    ↪ class_3)):
    # Unpack the x and y components directly
    x1, y1 = class_1_point
    x2, y2 = class_2_point
    x3, y3 = class_3_point

    # Compute the averages using a more concise approach
    x_avg = (x1 + x2 + x3) / 3
    y_avg = (y1 + y2 + y3) / 3

    print(f"Average for Class {i}: ({x_avg}, {y_avg})")
    average_points.append((x_avg, y_avg))
```

Output is displayed in the 'Test Sample' column below:

Table 3: Summary of classification accuracy.

Test Sample	Output of SVM 1	Output of SVM 2	Output of SVM 3	Class
(4.39, 3.88)	-1	-1	-1	N/A
(5.05, 5.15)	-1	-1	-1	N/A
(6.21, 3.04)	-1	1	-1	Class 2
(4.91, 4.15)	-1	-1	-1	N/A
(4.94, 4.56)	-1	-1	-1	N/A
(4.8, 4.74)	-1	-1	-1	N/A
(4.79, 2.99)	-1	-1	-1	N/A
(5.02, 4.31)	-1	-1	-1	N/A
(5.93, 4.43)	-1	1	-1	Class 2
(4.76, 3.49)	-1	-1	-1	N/A

Using points as np.array([above])

```
# SVM Parameters for each classifier
svm_params = [
    {'Weight Vector': [-1.12045698, -0.44988542], 'Bias': 6.297125088118931},
    {'Weight Vector': [1.52632027, -0.25141613], 'Bias': -6.646608104951451},
    {'Weight Vector': [0.09601877, 0.8151645], 'Bias': -4.743152537376741},
]
```

```
for point in points:
    for i, params in enumerate(svm_params):
        classification = np.sign(np.dot(params['Weight Vector'], point) + params['Bias'])
        print(f"Classification result for SVM {i + 1}, point {point}: {classification}")
```

To demonstrate these findings, only two of the ten sample points fall on the positive side of any function, and these are correctly classified as Class 2.

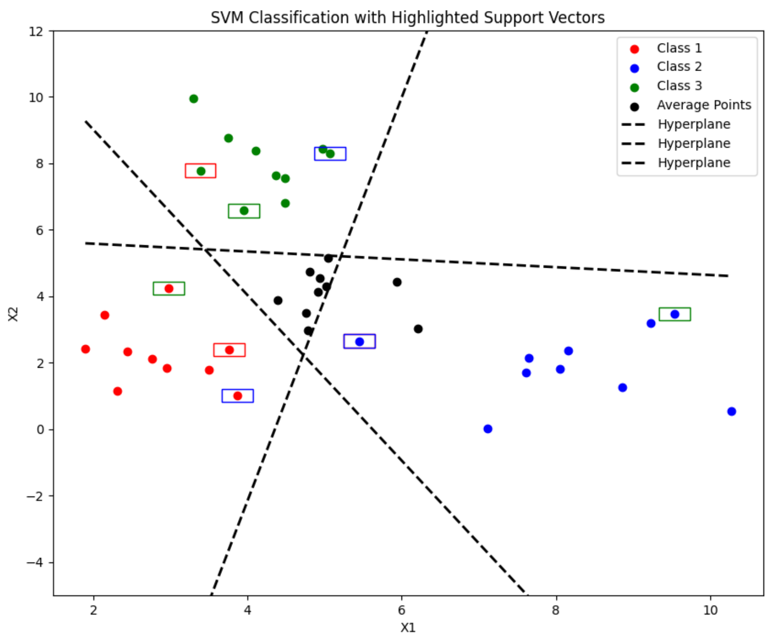


Figure 9: Averaged classes as input to multi-class SVM.

However, it may be useful to classify points based on a measure that considers their closeness to a class.

The average points are skewed towards Class 2, and this is not demonstrated in the above classifications.

Test Sample	Classification
(4.387, 3.878)	Class 1
(5.046, 5.150)	Class 3
(6.209, 3.044)	Class 2
(4.914, 4.146)	Class 2
(4.939, 4.556)	Class 2
(4.801, 4.740)	Class 2
(4.787, 2.987)	Class 2
(5.023, 4.306)	Class 2
(5.930, 4.433)	Class 2
(4.764, 3.494)	Class 2

Table 4: Summary of classification for averaged class points.

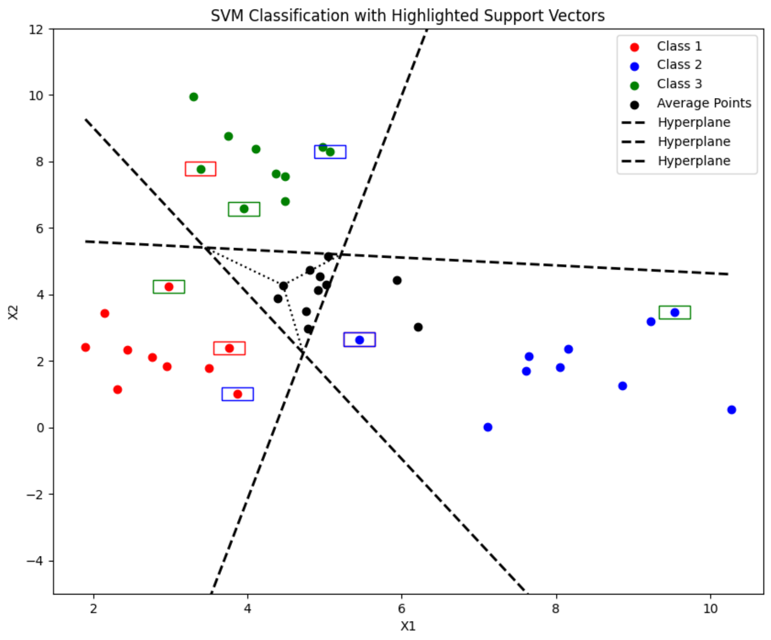


Figure 10: Averaged classes as input to multi-class SVM, with distance as confidence metric, visualised with segmented N/A area.

This graph shows that all but two points are classified by class 2. These two points are shown in the above table. Distance metrics available in the code;

```
def distance_to_hyperplane(point, weight, bias):
    norm_w = np.linalg.norm(weight)
    distance = np.abs(np.dot(weight, point) + bias) / norm_w
    return distance

# Initialize a list to hold the classification results
final_classifications = [None] * len(points)

for i, point in enumerate(points):
    distances = []
    already_classified = False

    for params in svm_params:
        classification = np.sign(np.dot(params['Weight Vector'], point) + params['Bias'])
        if classification > 0:
            already_classified = True
            final_classifications[i] = svm_params.index(params) + 1 # Classify as the current SVM's class
            break
        else:
            distances.append(distance_to_hyperplane(point, params['Weight Vector'], params['Bias']))

    if not already_classified:
        # Find the hyperplane with the minimum distance
        closest_svm_index = np.argmin(distances)
        final_classifications[i] = closest_svm_index + 1 # Classify as the SVM closest to the point

# Print the final classifications
for point, classification in zip(points, final_classifications):
    print(f"Point {point} is classified as Class {classification}")
```

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 method calculates a higher confidence value for the sample closest to the hyperplane in hyperspace. For a 2D hyperspace, we simply use Euclidean distance.