ROLL NO.: A-25	**NAME:SHRADDHA SAWANT**
CLASS:T.E.	**BATCH:A-2**
DATE OF PERFORMANCE:	**DATE OF SUBMISSION:27/3/2025**
GRADE:	

Experiment No.06

Aim: Implementation of Support Vector Machines Algorithm.

Outcome:

After completing this practical, the **Support Vector Machine (SVM) algorithm** was successfully implemented, demonstrating its effectiveness in classification tasks. Hands-on experience was gained in data preprocessing, training an SVM model, tuning hyperparameters like kernel functions, and evaluating performance using accuracy, confusion matrix, and decision boundary visualization. This knowledge enhances the ability to apply SVM for real-world classification problems, especially in high-dimensional spaces.

Theory:

Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression tasks. While it can handle regression problems, SVM is particularly well-suited for classification tasks.

SVM aims to find the optimal hyperplane in an N-dimensional space to separate data points into different classes. The algorithm maximizes the margin between the closest points of different classes.

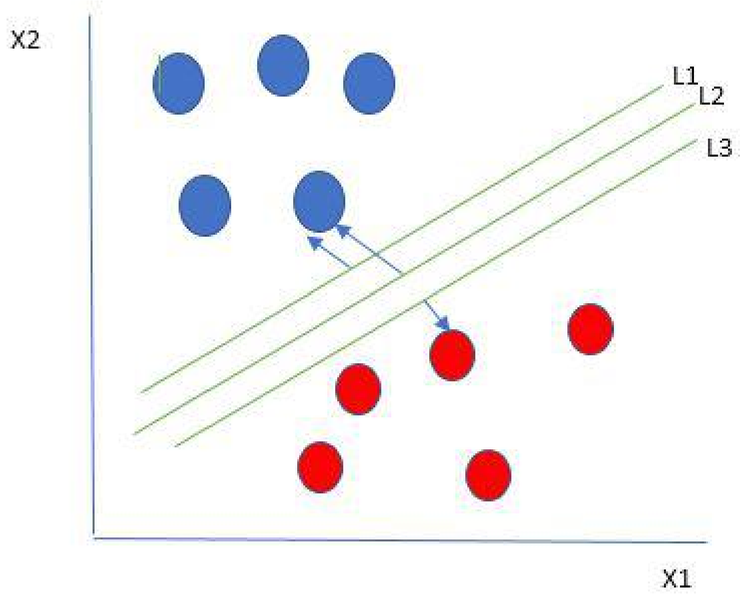
Support Vector Machine (SVM) Terminology

- **Hyperplane**: A decision boundary separating different classes in feature space, represented by the equation **wx + b = 0** in linear classification.
- **Support Vectors**: The closest data points to the hyperplane, crucial for determining the hyperplane and margin in SVM.
- **Margin**: The distance between the hyperplane and the support vectors. SVM aims to maximize this margin for better classification performance.
- **Kernel**: A function that maps data to a higher-dimensional space, enabling SVM to handle non-linearly separable data.
- **Hard Margin**: A maximum-margin hyperplane that perfectly separates the data without misclassifications.
- **Soft Margin**: Allows some misclassifications by introducing slack variables, balancing margin maximization and misclassification penalties when data is not perfectly separable.

- **C**: A regularization term balancing margin maximization and misclassification penalties. A higher C value enforces a stricter penalty for misclassifications.
- **Hinge Loss**: A loss function penalizing misclassified points or margin violations, combined with regularization in SVM.
- **Dual Problem**: Involves solving for Lagrange multipliers associated with support vectors, facilitating the kernel trick and efficient computation.

How does Support Vector Machine Algorithm Work?

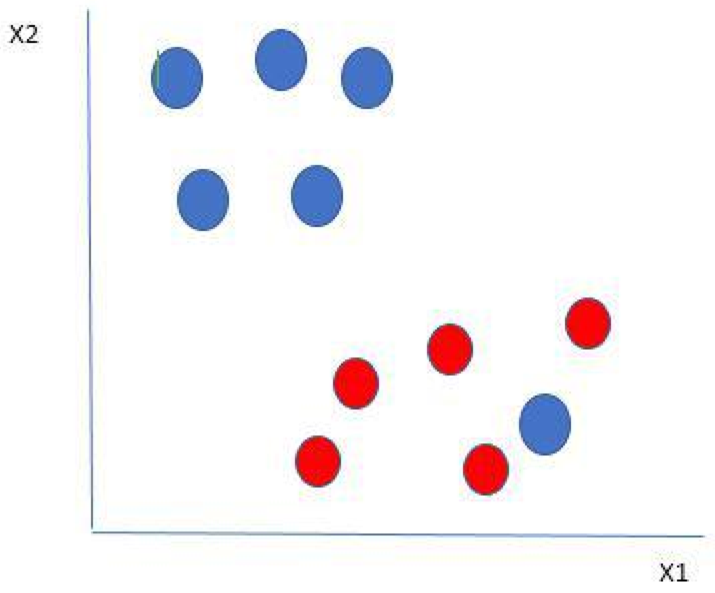
The key idea behind the SVM algorithm is to find the hyperplane that best separates two classes by maximizing the margin between them. This margin is the distance from the hyperplane to the nearest data points (**support vectors**) on each side.



Multiple hyperplanes separate the data from two classes

The best hyperplane, also known as the **"hard margin,"** is the one that maximizes the distance between the hyperplane and the nearest data points from both classes. This ensures a clear separation between the classes. So, from the above figure, we choose L2 as hard margin.

Let's consider a scenario like shown below:



A soft margin allows for some misclassifications or violations of the margin to improve generalization. The SVM optimizes the following equation to balance margin maximization and penalty minimization:

Objective Function=(1margin)+λ∑penalty Objective Function=(margin1)+λ∑penalty

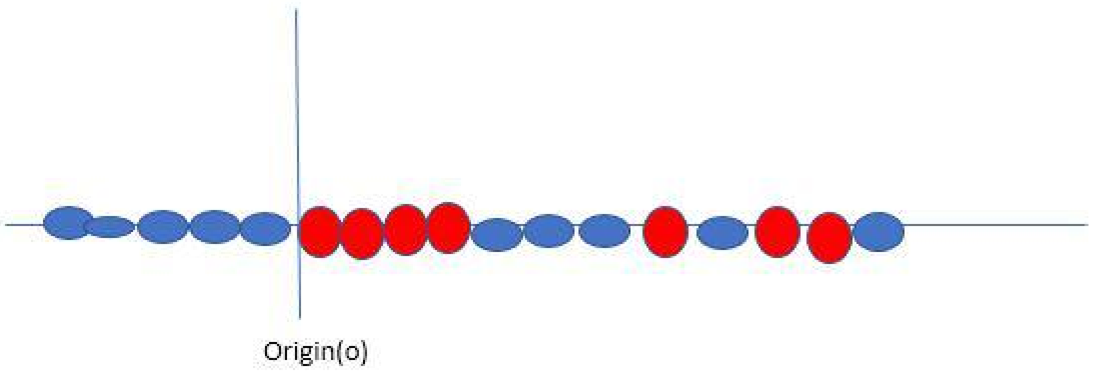
The penalty used for violations is often **hinge loss**, which has the following behavior:

- If a data point is correctly classified and within the margin, there is no penalty (loss = 0).
- If a point is incorrectly classified or violates the margin, the hinge loss increases proportionally to the distance of the violation.

Till now, we were talking about linearly separable data(the group of blue balls and red balls are separable by a straight line/linear line).

What to do if data are not linearly separable?

When data is not linearly separable (i.e., it can’t be divided by a straight line), SVM uses a technique called **kernels** to map the data into a higher-dimensional space where it becomes separable. This transformation helps SVM find a decision boundary even for non-linear data.



Original 1D dataset for classification

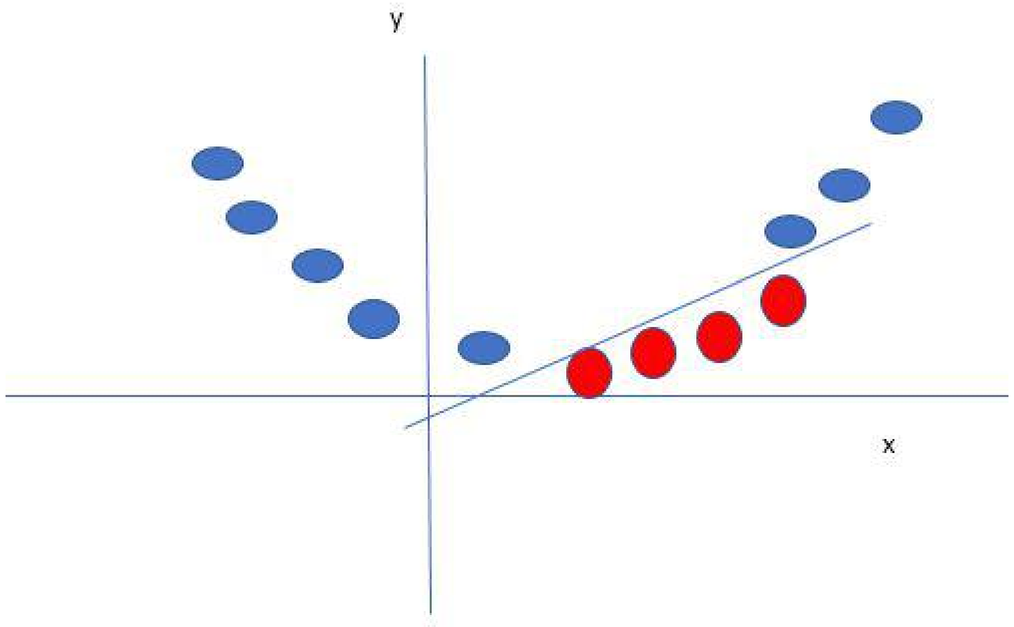
A **kernel** is a function that maps data points into a higher-dimensional space without explicitly computing the coordinates in that space. This allows SVM to work efficiently with non-linear data by implicitly performing the mapping.

For example, consider data points that are not linearly separable. By applying a kernel function, SVM transforms the data points into a higher-dimensional space where they become linearly separable.

- **Linear Kernel**: For linear separability.
- **Polynomial Kernel**: Maps data into a polynomial space.
- **Radial Basis Function (RBF) Kernel**: Transforms data into a space based on distances between data points.

wTx+*b*=0
Where:

- w*w* is the normal vector to the hyperplane (the direction perpendicular to it).
- b*b* is the offset or bias term, representing the distance of the hyperplane from the origin along the normal vector w*w*.



Mapping 1D data to 2D to become able to separate the two classes

In this case, the new variable y is created as a function of distance from the origin.

Mathematical Computation: SVM

Consider a binary classification problem with two classes, labeled as +1 and -1. We have a training dataset consisting of input feature vectors X and their corresponding class labels Y.

The equation for the linear hyperplane can be written as:

wTx+b=0

A soft margin allows for some misclassifications or violations of the margin to improve generalization. The SVM optimizes the following equation to balance margin maximization and penalty minimization:

Objective Function=(1margin)+λ∑penalty Objective Function=(margin1)+λ∑penalty

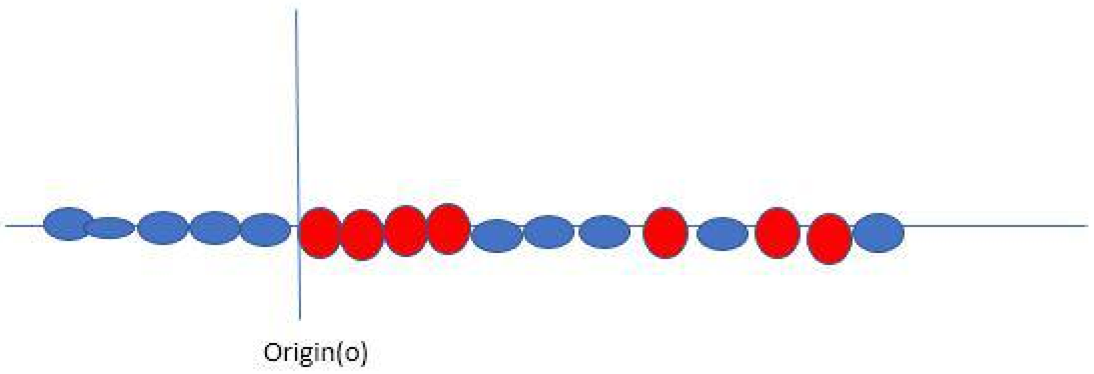
The penalty used for violations is often **hinge loss**, which has the following behavior:

- If a data point is correctly classified and within the margin, there is no penalty (loss = 0).
- If a point is incorrectly classified or violates the margin, the hinge loss increases proportionally to the distance of the violation.

Till now, we were talking about linearly separable data(the group of blue balls and red balls are separable by a straight line/linear line).

What to do if data are not linearly separable?

When data is not linearly separable (i.e., it can’t be divided by a straight line), SVM uses a technique called **kernels** to map the data into a higher-dimensional space where it becomes separable. This transformation helps SVM find a decision boundary even for non-linear data.

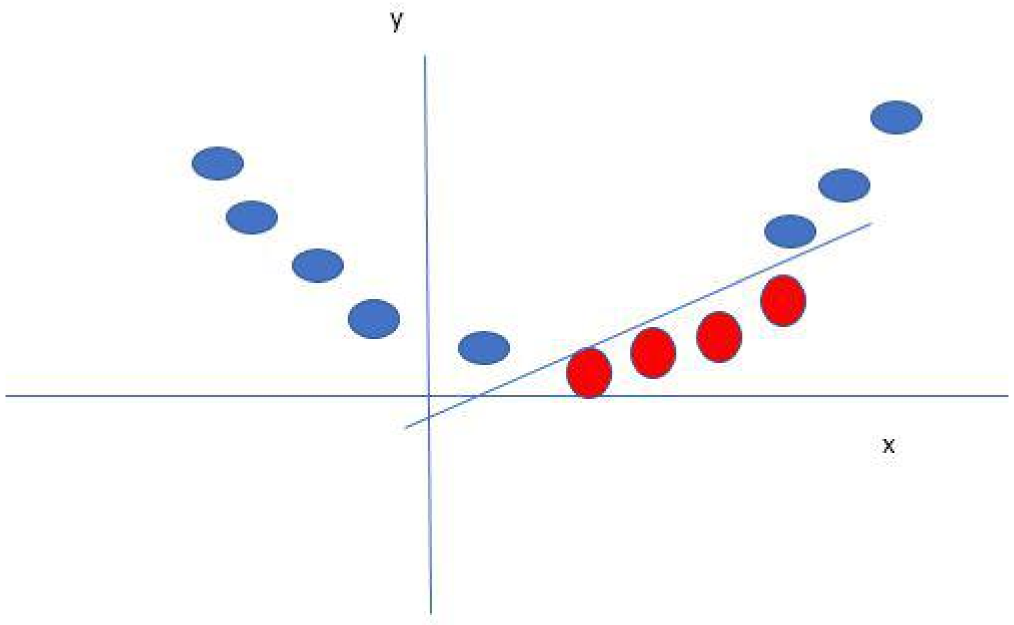


Original 1D dataset for classification

A **kernel** is a function that maps data points into a higher-dimensional space without explicitly computing the coordinates in that space. This allows SVM to work efficiently with non-linear data by implicitly performing the mapping.

For example, consider data points that are not linearly separable. By applying a kernel function, SVM transforms the data points into a higher-dimensional space where they become linearly separable.

- **Linear Kernel**: For linear separability.
- **Polynomial Kernel**: Maps data into a polynomial space.
- **Radial Basis Function (RBF) Kernel**: Transforms data into a space based on distances between data points.



Mapping 1D data to 2D to become able to separate the two classes

In this case, the new variable y is created as a function of distance from the origin.

Mathematical Computation: SVM

Consider a binary classification problem with two classes, labeled as +1 and -1. We have a training dataset consisting of input feature vectors X and their corresponding class labels Y.

The equation for the linear hyperplane can be written as:

$$wTx+b=0$$

Advantages of Support Vector Machine (SVM)

- **High-Dimensional Performance**: SVM excels in high-dimensional spaces, making it suitable for **image classification** and **gene expression analysis**.

- **Nonlinear Capability**: Utilizing **kernel functions** like **RBF** and **polynomial**, SVM effectively handles **nonlinear relationships**.

- **Outlier Resilience**: The **soft margin** feature allows SVM to ignore outliers, enhancing robustness in **spam detection** and **anomaly detection**.

- **Binary and Multiclass Support**: SVM is effective for both **binary classification** and **multiclass classification**, suitable for applications in **text classification**.

- **Memory Efficiency**: SVM focuses on **support vectors**, making it memory efficient compared to other algorithms.
-

Disadvantages of Support Vector Machine (SVM)

- **Slow Training**: SVM can be slow for large datasets, affecting performance in **SVM in data mining** tasks.

- **Parameter Tuning Difficulty**: Selecting the right **kernel** and adjusting parameters like **C** requires careful tuning, impacting **SVM algorithms**.
- **Noise Sensitivity**: SVM struggles with noisy datasets and overlapping classes, limiting effectiveness in real-world scenarios.

- **Limited Interpretability**: The complexity of the **hyperplane** in higher dimensions makes SVM less interpretable than other models.

- **Feature Scaling Sensitivity**: Proper **feature scaling** is essential; otherwise, SVM models may perform poorly.

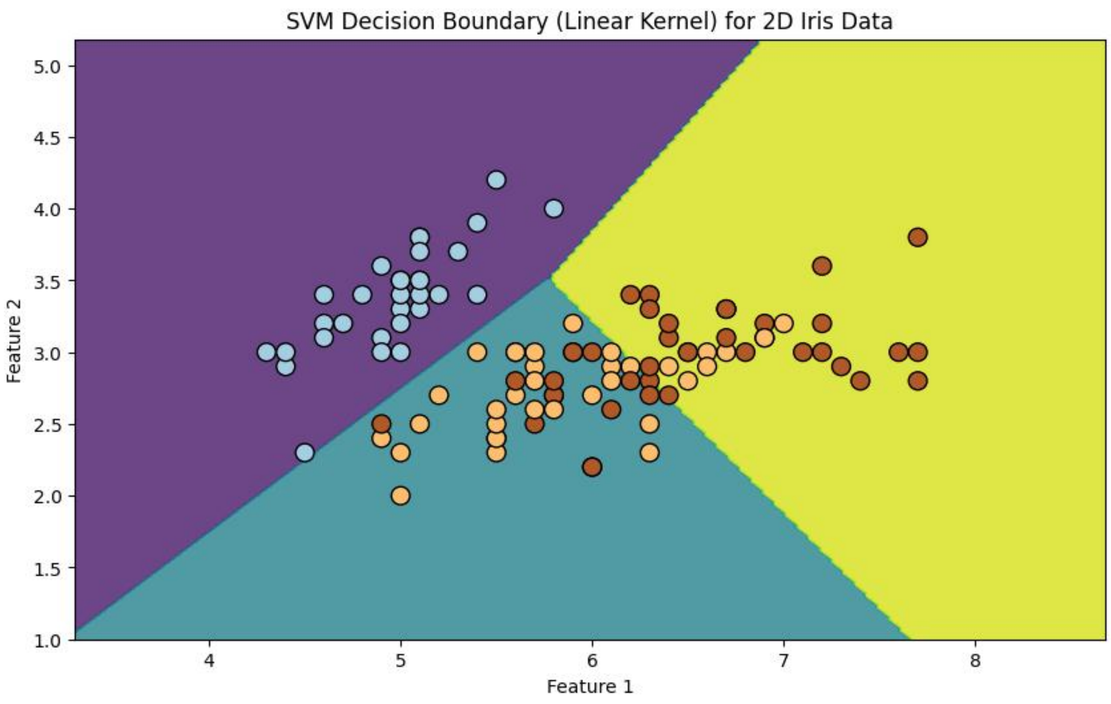
CODE:

```python
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score

# Load the dataset (Iris dataset in this case)
iris = datasets.load_iris()
X = iris.data  # Features
y = iris.target  # Target labels

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create an SVM model (Support Vector Classifier)
svm = SVC(kernel='linear')  # You can change the kernel to 'rbf' or 'poly' if needed

# Train the model
svm.fit(X_train, y_train)

# Predict on the test set
y_pred = svm.predict(X_test)

# Evaluate the model
```

```python
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))

# Optional: Visualize the decision boundary (works best for 2D data)
# For simplicity, we'll use only the first two features of the Iris dataset.
X_2D = X[:, :2]
X_train_2D, X_test_2D, y_train_2D, y_test_2D = train_test_split(X_2D, y, test_size=0.3,
random_state=42)

svm_2D = SVC(kernel='linear')
svm_2D.fit(X_train_2D, y_train_2D)

# Plot the decision boundaries
plt.figure(figsize=(10, 6))
h = .02  # Step size in the mesh
x_min, x_max = X_train_2D[:, 0].min() - 1, X_train_2D[:, 0].max() + 1
y_min, y_max = X_train_2D[:, 1].min() - 1, X_train_2D[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

Z = svm_2D.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.8)
plt.scatter(X_train_2D[:, 0], X_train_2D[:, 1], c=y_train_2D, edgecolors='k', marker='o',
s=100, cmap=plt.cm.Paired)
plt.title("SVM Decision Boundary (Linear Kernel) for 2D Iris Data")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

OUTPUT:

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```



Conclusion:

Successfully implemented the **Support Vector Machine (SVM) algorithm**, observed its ability to classify data by finding the optimal hyperplane, and gained practical insights into kernel functions, margin maximization, and real-world applications of SVM in machine learning.