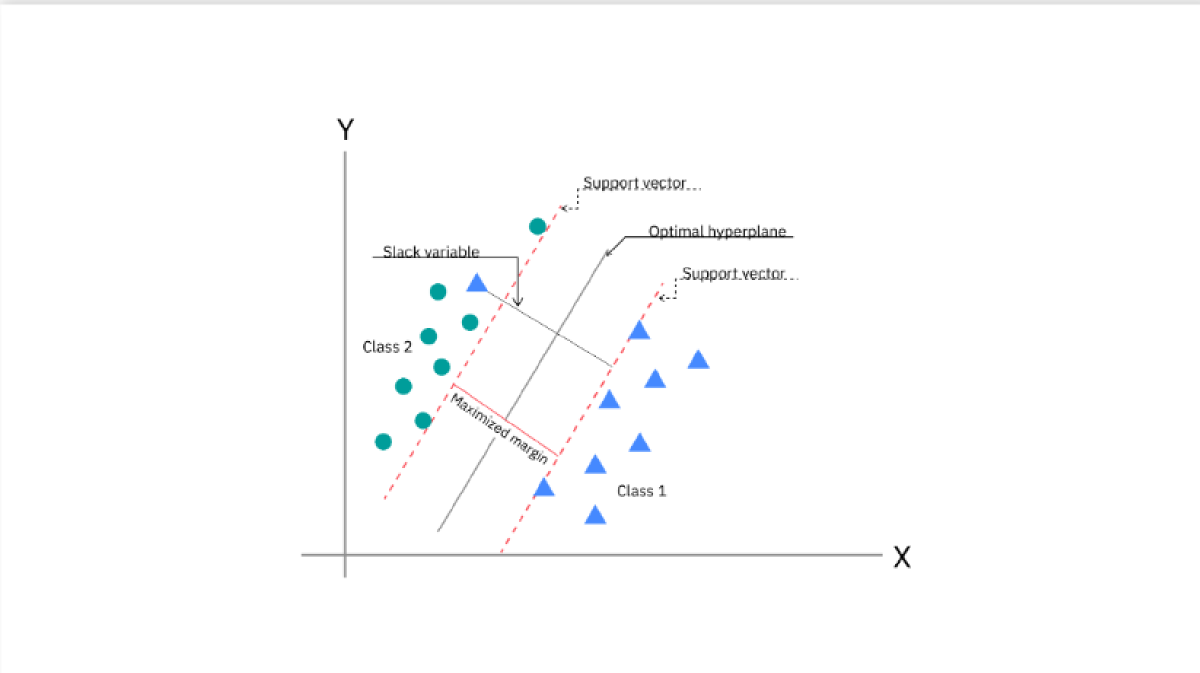
Support Vector Machine (SVM)

SVM is a powerful supervised learning algorithm used for both **classification** and **regression** tasks, though it is primarily used for classification.



How SVM Works in Machine Learning

1. **Separating Data with a Hyperplane**
   - SVM finds the optimal **hyperplane** that best separates different classes in the dataset.
   - The hyperplane is chosen to **maximize the margin** between the nearest data points of different classes.
2. **Support Vectors**
   - The data points closest to the hyperplane are called **support vectors**.
   - These points determine the position and orientation of the hyperplane.
3. **Margin and Optimization**
   - The **margin** is the distance between the hyperplane and the nearest support vectors.
   - A **larger margin** improves generalization and reduces overfitting.

4. **Kernel Trick**
   - If the data is not linearly separable, SVM uses a **kernel function** to transform data into a higher-dimensional space where it can be separated.
   - Common kernel functions:
     - **Linear Kernel**: Used when data is linearly separable.
     - **Polynomial Kernel**: Maps input space into a higher-degree polynomial.
     - **Radial Basis Function (RBF) Kernel**: Suitable for complex non-linear data.
     - **Sigmoid Kernel**: Used for neural network-like decision functions.

Types of SVM

1. **Linear SVM**: Used when the data is linearly separable.
2. **Non-Linear SVM**: Uses kernel tricks to handle complex, non-linearly separable data.
3. **Support Vector Regression (SVR)**: An extension of SVM for regression problems.

SVM Implementation in Python using Scikit-Learn

Here's how you can implement SVM for classification using Python:

```
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 accuracy_score


# Load dataset (Iris dataset)
iris = datasets.load_iris()
X = iris.data[:, :2]  # Taking only first two features for visualization
```

```
y = iris.target

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train SVM classifier
svm_model = SVC(kernel='linear', C=1.0)  # Using a linear kernel
svm_model.fit(X_train, y_train)

# Make predictions
y_pred = svm_model.predict(X_test)

# Evaluate model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Visualizing Decision Boundary
def plot_decision_boundary(X, y, model):
    h = 0.02  # Step size in the mesh
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

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

    plt.contourf(xx, yy, Z, alpha=0.3)
```

```
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', marker='o')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title('SVM Decision Boundary')
    plt.show()


plot_decision_boundary(X_train, y_train, svm_model)
```

Advantages of SVM in ML

✓ Works well for high-dimensional data.
✓ Effective in cases where the number of dimensions is greater than the number of samples.
✓ Handles non-linearly separable data using kernel trick.
✓ Robust to overfitting in high-dimensional spaces.

Disadvantages of SVM

❌ Computationally expensive for large datasets.
❌ Not effective when there is a lot of noise in the data.
❌ Requires careful tuning of kernel parameters and regularization.

Applications of SVM in Machine Learning

📌 **Image Classification** (Face recognition, digit classification)
📌 **Text Categorization** (Spam detection, sentiment analysis)
📌 **Medical Diagnosis** (Cancer detection)
📌 **Fraud Detection** (Credit card fraud)

Explanation of Attributes in SVM

When using **SVC (Support Vector Classification)** in sklearn.svm, several key attributes control how the model behaves. Below is a detailed explanation of the most important ones:

1.Kernel (kernel)

- **Defines how the data is transformed before classification.**
- Different kernels work for different types of data distributions.

Kernel Type	Description	Best For
"linear"	Uses a straight-line decision boundary	Linearly separable data
"rbf" (Radial Basis Function)	Creates a complex, non-linear decision boundary	Data that is **not** linearly separable
"poly" (Polynomial)	Uses polynomial curves for separation	More complex relationships
"sigmoid"	Similar to a neural network activation function	Rarely used

✅ **Example:**

python

```
model = SVC(kernel="rbf")  # Uses RBF kernel (good for non-linear data)
```

---

2. Regularization Parameter (C)

- Controls the trade-off between **maximizing margin** and **minimizing classification error**.

Value of C	Effect
Low (C → 0.1, 1)	Wider margin, **more misclassifications**, but generalizes better
High (C → 100, 1000)	**Narrower margin**, tries to classify every point correctly, but may **overfit**

✅ **Example:**

```
model = SVC(kernel="linear", C=10)  # Balances margin width and classification
```

3.Gamma (gamma) (Used in RBF, Poly, and Sigmoid Kernels)

- Controls the influence of a **single data point** on the decision boundary.

Value of gamma	Effect
Low (gamma → 0.01, 0.1)	Smoother decision boundary (less overfitting)
High (gamma → 1, 10, 100)	Complex decision boundary (risk of overfitting)

Example:

```
model = SVC(kernel="rbf", C=1, gamma=0.1)  # Well-balanced setting
```

---

4. Degree (degree) (Used only in poly Kernel)

- Defines the degree of the polynomial kernel.

Example:

```
model = SVC(kernel="poly", degree=3)  # Uses a cubic polynomial decision boundary
```

---

5. Class Weight (class_weight)

- Adjusts the importance of different classes (useful for imbalanced data).

Option	Effect
"balanced"	Adjusts class weights automatically based on data distribution
{1:1, 2:5}	Manually set weights (e.g., gives class 2 5x more importance)

Example:

```
model = SVC(kernel="linear", class_weight="balanced")  # Handles class imbalance
```

6.Probability (probability)

- Enables probability estimates (default is False).
- **Setting it to True allows .predict_proba() for probability outputs.**

Example:

```
model = SVC(kernel="rbf", probability=True)
```

- **For simple, linearly separable data** → Use kernel="linear", small C.
- **For complex, non-linear data** → Use kernel="rbf", tune C & gamma.
- **For imbalanced classes** → Use class_weight="balanced".

Ref: https://www.ibm.com/think/topics/support-vector-machine