Register No: 21MIS1044

Name: Srinivasan JP

Support Vector Mechanism in Gradient descent

In [ ]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import datasets
import matplotlib.pyplot as plt
```

Generating the dataset with the random numbers with 2 features and 2 classees 1 and -1

In [ ]:
```
# Generate a synthetic dataset
X, y = datasets.make_classification(n_samples=1000, n_features=2, n_classes=2, n_clust
# Add a bias term to the features
X = np.c_[X, np.ones(X.shape[0])]

# Map labels to -1 and 1 for SVM
y = np.where(y == 0, -1, 1)
print(X,y, sep="\n")
```

```
[[-1.26656013  0.56020728  1.        ]
 [ 1.30895197  1.2918687   1.        ]
 [-1.67704005  1.05762366  1.        ]
 ...
 [-0.97972283  0.0568245   1.        ]
 [ 0.53862088  2.23919241  1.        ]
 [ 0.7870762   0.73113952  1.        ]]
[-1  1 -1  1  1  1  1  1 -1 -1  1 -1  1 -1  1 -1 -1 -1  1  1 -1  1 -1 -1
 -1  1 -1  1 -1 -1  1 -1 -1 -1  1  1 -1  1  1  1 -1  1  1  1  1  1  1  1
  1 -1 -1  1  1  1  1  1  1  1  1 -1  1 -1 -1  1 -1 -1 -1  1 -1  1  1  1
  1  1  1  1 -1 -1  1  1  1  1 -1  1  1  1 -1 -1 -1 -1  1 -1 -1  1 -1 -1
  1 -1  1  1  1  1 -1  1  1 -1 -1  1  1 -1  1  1 -1 -1  1  1 -1  1  1 -1
 -1  1 -1  1  1 -1 -1  1  1 -1 -1 -1  1 -1  1  1  1 -1 -1  1  1  1 -1 -1
 -1  1  1 -1  1  1 -1 -1 -1  1 -1 -1  1  1  1 -1 -1  1  1 -1 -1  1 -1 -1
  1  1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1  1  1 -1  1 -1  1 -1  1 -1  1 -1
 -1  1 -1 -1  1 -1 -1  1 -1 -1  1  1  1 -1  1  1 -1  1  1  1 -1 -1  1  1
 -1  1 -1 -1  1 -1  1 -1  1  1  1  1 -1  1  1  1 -1 -1  1 -1  1  1 -1 -1
  1  1 -1 -1  1 -1  1  1 -1  1 -1 -1 -1 -1 -1  1  1 -1 -1 -1  1  1 -1  1
  1  1  1  1 -1  1 -1 -1  1  1 -1 -1  1  1 -1  1 -1  1 -1 -1 -1  1  1  1
 -1  1 -1 -1  1  1  1  1  1  1 -1  1  1 -1 -1 -1  1 -1  1 -1 -1  1 -1  1
 -1  1 -1 -1 -1 -1  1  1 -1 -1  1  1 -1 -1 -1 -1 -1 -1 -1 -1  1  1 -1 -1
 -1 -1  1 -1  1  1  1 -1 -1  1 -1  1  1 -1 -1 -1 -1  1 -1  1 -1  1  1 -1
  1 -1  1 -1  1 -1  1 -1  1 -1  1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1  1  1  1
  1  1 -1 -1  1 -1  1 -1  1  1  1  1  1  1  1  1 -1  1 -1 -1 -1  1 -1  1
 -1  1  1 -1  1  1  1  1  1 -1 -1  1  1 -1  1  1  1 -1 -1  1 -1 -1  1 -1
  1 -1  1 -1  1  1  1  1 -1 -1  1 -1  1  1  1  1 -1 -1  1 -1  1 -1 -1  1
 -1  1  1 -1  1  1 -1  1 -1 -1 -1  1  1 -1  1 -1  1  1 -1 -1 -1  1 -1 -1
  1  1  1  1 -1 -1 -1 -1  1  1  1  1 -1 -1  1 -1 -1 -1 -1 -1  1 -1 -1 -1
 -1  1 -1 -1  1 -1 -1  1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1
  1 -1 -1  1 -1  1 -1  1  1 -1 -1  1  1  1 -1 -1  1 -1 -1 -1  1 -1 -1  1
  1  1 -1 -1 -1  1  1 -1 -1 -1  1 -1  1  1  1  1  1  1 -1 -1 -1 -1 -1  1
  1  1 -1  1  1  1 -1 -1  1  1  1  1 -1 -1 -1  1  1 -1 -1  1  1  1 -1  1
 -1  1  1 -1 -1  1 -1  1 -1 -1 -1  1 -1 -1  1 -1 -1 -1 -1 -1  1 -1  1  1
 -1 -1 -1  1 -1 -1 -1  1 -1  1  1 -1  1  1 -1 -1 -1 -1 -1  1  1 -1 -1  1
 -1 -1  1 -1 -1  1  1 -1  1  1 -1  1  1 -1  1  1 -1  1  1  1  1 -1  1 -1
 -1  1  1 -1 -1  1  1 -1 -1 -1  1 -1 -1 -1  1 -1 -1  1 -1 -1  1 -1  1 -1
  1 -1  1  1 -1 -1 -1  1  1 -1 -1  1 -1  1 -1 -1  1  1 -1 -1 -1 -1 -1  1
  1 -1 -1  1 -1  1  1 -1  1 -1  1 -1  1  1 -1 -1  1 -1 -1 -1 -1  1  1 -1
 -1 -1  1  1  1  1  1 -1 -1 -1 -1  1 -1  1 -1  1 -1 -1 -1  1  1  1 -1  1
  1  1  1  1  1  1  1  1 -1  1 -1 -1 -1  1  1  1  1 -1  1  1 -1  1  1  1
 -1  1 -1  1 -1  1 -1 -1 -1  1 -1  1 -1  1 -1 -1 -1 -1  1  1 -1  1 -1  1
  1 -1 -1  1 -1  1  1 -1  1 -1 -1 -1  1  1  1  1  1 -1 -1  1 -1  1 -1 -1
  1  1  1  1 -1 -1  1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1
  1  1  1  1 -1  1  1 -1 -1  1 -1 -1 -1 -1  1  1  1  1 -1  1  1 -1  1  1
  1  1 -1 -1 -1  1  1  1  1 -1  1  1 -1  1  1 -1  1 -1  1 -1  1  1  1 -1
  1 -1  1  1  1  1  1  1 -1  1 -1  1 -1 -1 -1 -1  1  1  1 -1  1  1  1 -1
 -1  1 -1 -1 -1  1 -1  1 -1  1 -1 -1 -1 -1 -1 -1  1]
```

Using the train_test_split the train dataset splitted to 80% and test is 20%

In [ ]:
```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
```

To predict using gradient decent algorithm the class is defined

In [ ]:
```
class SVM:
    def __init__(self, learning_rate=0.01, lambda_param=0.01, n_iters=1000):
```

```
        self.lr = learning_rate
        self.lambda_param = lambda_param
        self.n_iters = n_iters
        self.weights = None

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)

        for _ in range(self.n_iters):
            for i in range(n_samples):
                if y[i] * (np.dot(X[i], self.weights)) < 1:
                    self.weights -= self.lr * (2 * self.lambda_param * self.weights -

    def predict(self, X):
        return np.sign(np.dot(X, self.weights))
```

The model is initialized

In [ ]:
```
# Instantiate SVM model
svm_model = SVM(learning_rate=0.01, lambda_param=0.01, n_iters=1000)
```

Training the model with the train data set which is splitted form the generated dataset

In [ ]:
```
# Train the model
svm_model.fit(X_train, y_train)
```

In [ ]:
```
# Make predictions on the test set
predictions = svm_model.predict(X_test)
```

Calculating the accuracy of the model

In [ ]:
```
# Evaluate the accuracy
accuracy = np.mean(predictions == y_test)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

Ploting the SVM hyperplane and the dataset data as scatter plot

In [ ]:
```
# Plot the decision boundary
def plot_decision_boundary(X, y, model):
    h = .02
    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(), np.ones(xx.ravel().shape[0])])

    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.3)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm, edgecolors='k')
    plt.title("SVM Decision Boundary")
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.show()
```

```
# Plot decision boundary
plot_decision_boundary(X_test[:, :2], y_test, svm_model)
```

```
Accuracy: 95.00%
```