ML LAB 07 - SVM GRADIENT DESCENT

Name : Tulasi Raman R

Register Number : 21MIS1170

Building dataset

In [21]:

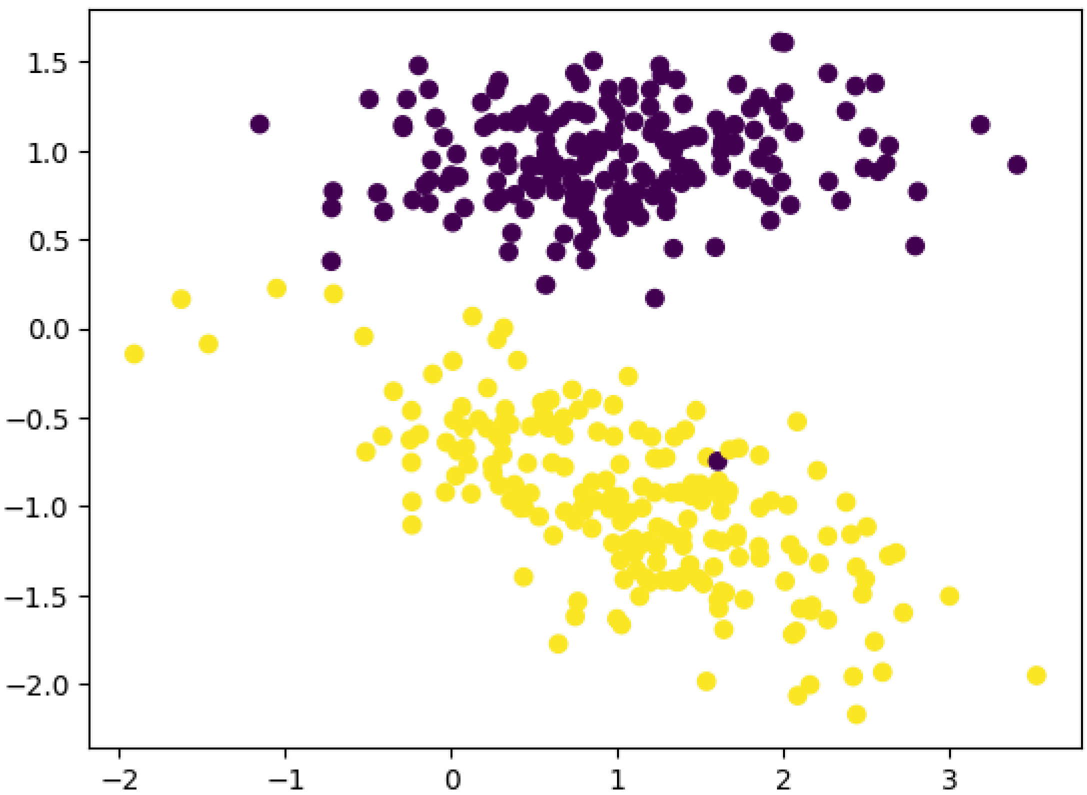
```
from matplotlib import pyplot as plt
from sklearn.datasets import make_classification

X, Y = make_classification(n_classes=2, n_samples=400, n_clusters_per_class=1, r

# Convert our Y-Labels into {1,-1}

Y[Y==0] = -1 #Broadcasting
print(Y)
```

```
[-1  1 -1  1 -1  1  1 -1 -1  1 -1  1  1 -1  1  1 -1 -1 -1  1  1  1  1 -1
 -1 -1  1  1  1 -1 -1  1 -1 -1 -1 -1 -1 -1  1  1  1 -1 -1  1  1 -1 -1 -1
  1  1 -1 -1  1  1 -1  1 -1  1  1  1 -1  1  1 -1 -1  1  1 -1 -1 -1 -1  1
  1 -1  1 -1  1 -1  1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1 -1  1 -1  1 -1  1  1
 -1  1  1  1 -1 -1  1 -1  1 -1  1  1  1  1  1  1 -1  1  1 -1 -1  1  1 -1
  1  1 -1 -1 -1  1  1  1 -1 -1 -1  1 -1 -1 -1 -1 -1  1 -1  1  1  1 -1 -1
  1  1 -1 -1  1  1  1  1  1  1 -1  1 -1  1  1 -1  1  1  1  1 -1  1 -1  1
  1  1 -1  1  1  1 -1 -1 -1 -1  1 -1 -1 -1 -1 -1  1  1  1  1 -1  1  1 -1
 -1  1  1  1  1  1 -1 -1  1 -1  1 -1 -1  1  1  1 -1  1  1 -1 -1 -1  1  1
 -1  1  1  1 -1  1 -1  1 -1 -1 -1  1  1  1  1  1  1  1  1 -1 -1  1 -1  1
  1  1  1 -1  1  1 -1  1  1  1  1 -1  1  1 -1  1 -1 -1 -1 -1 -1  1 -1 -1
  1  1  1 -1 -1  1 -1 -1  1 -1 -1 -1 -1  1 -1  1 -1 -1 -1  1  1  1 -1  1
 -1  1 -1  1 -1  1  1  1 -1  1 -1 -1 -1 -1  1 -1 -1  1 -1 -1 -1 -1 -1  1
  1 -1  1  1 -1 -1 -1  1  1 -1 -1 -1 -1  1  1  1  1  1  1 -1  1 -1 -1  1
 -1 -1  1  1  1 -1  1 -1  1 -1 -1 -1 -1 -1 -1  1 -1 -1  1  1  1  1 -1  1
 -1  1  1 -1  1 -1 -1 -1 -1 -1  1 -1 -1  1 -1 -1  1 -1 -1  1  1 -1 -1  1
  1  1 -1  1  1 -1  1 -1 -1 -1 -1 -1 -1 -1  1 -1]
```

Visualizing Dataset

In [22]:

```
plt.scatter(X[:,0], X[:,1], c=Y)
plt.show()
```



Applying Gradient Descent

In [23]:

```
import numpy as np

class SVM:
    def __init__(self, C=1.0):
        self.C = C
        self.W = 0
        self.b = 0

    def hingeLoss(self, W, b, X, Y):
        loss = 0.0
        loss += .5 * np.dot(W, W.T)
        m = X.shape[0]
        for i in range(m):
            ti = Y[i] * (np.dot(W, X[i].T) + b)
            loss += self.C * max(0, (1 - ti))
        return loss[0][0]

    def fit(self, X, Y, batch_size=100, learning_rate=0.001, maxItr=300):
        no_of_features = X.shape[1]
        no_of_samples = X.shape[0]
        n = learning_rate
        c = self.C

        # Init the model parameters
        W = np.zeros((1, no_of_features))
        bias = 0

        # Initial Loss
        # Training from here…
        # Weight and Bias update rule that we discussed!
```

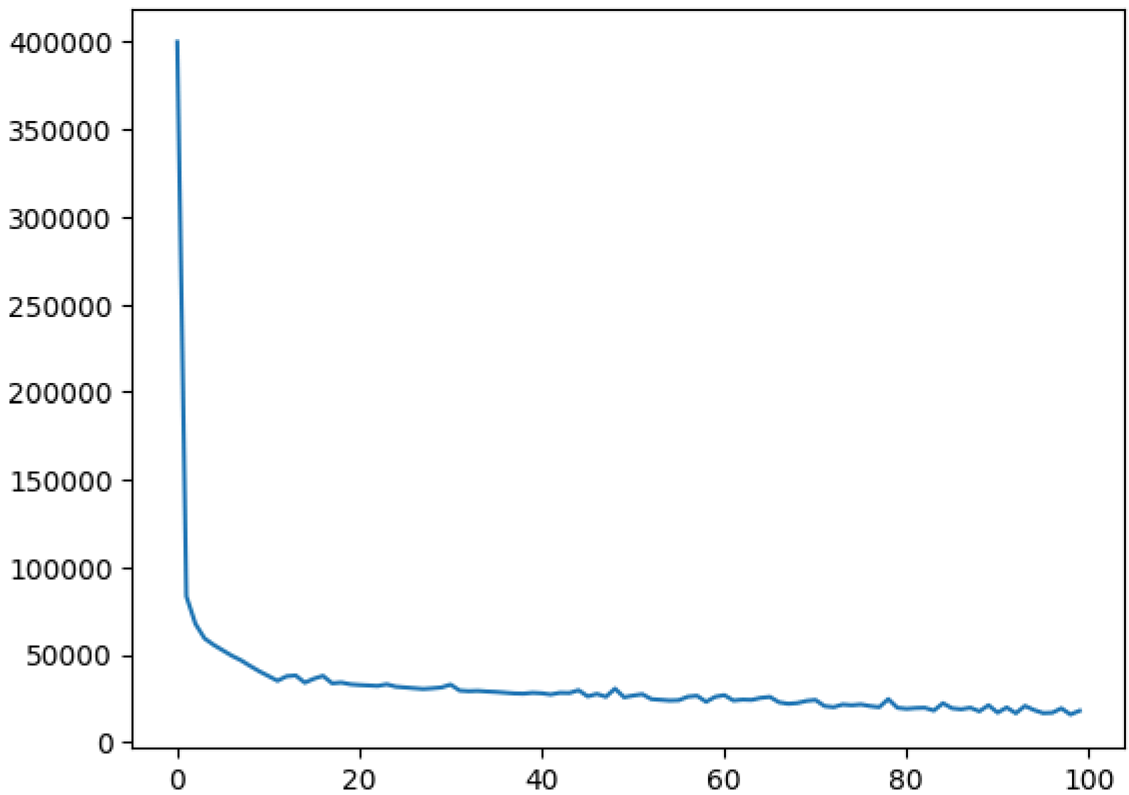
```
        losses = []
        for i in range(maxItr):
            # Training Loop
            l = self.hingeLoss(W, bias, X, Y)
            losses.append(l)
            ids = np.arange(no_of_samples)
            np.random.shuffle(ids)
            # Batch Gradient Descent(Paper) with random shuffling
            for batch_start in range(0, no_of_samples, batch_size):
                # Assume 0 gradient for the batch
                gradw = 0
                gradb = 0
                # Iterate over all examples in the mini batch
                for j in range(batch_start, batch_start + batch_size):
                    if j < no_of_samples:
                        i = ids[j]
                        ti = Y[i] * (np.dot(W, X[i].T) + bias)
                        if ti > 1:
                            gradw += 0
                            gradb += 0
                        else:
                            gradw += c * Y[i] * X[i]
                            gradb += c * Y[i]
                # Gradient for the batch is ready! Update W, B
                W = W - n * W + n * gradw
                bias = bias + n * gradb
        self.W = W
        self.b = bias
        return W, bias, losses

# Usage:
mySVM = SVM(C=1000)
W, b, losses = mySVM.fit(X, Y, maxItr=100)
print(losses[0])
print(losses[-1])
```

```
400000.0
18018.430494335094
```

- The first loss gives you an initial idea of the loss before any optimization or training has occurred.
- The last loss lets you see how well the model has converged or improved after training.

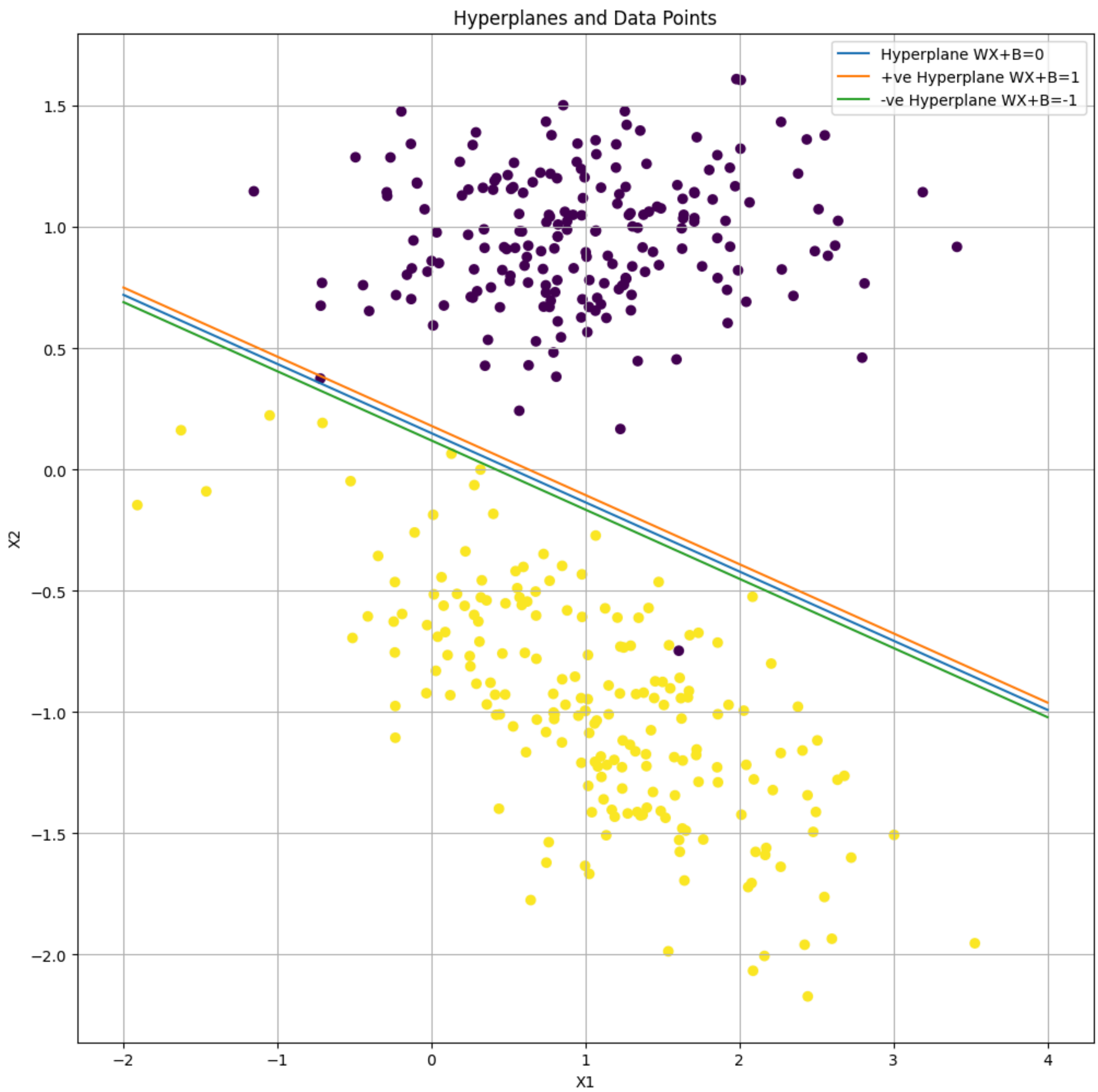
In [24]:

```
plt.plot(losses)
plt.show()
```



In [25]:

```
import numpy as np
import matplotlib.pyplot as plt

def plotHyperplane(w1, w2, b, X, Y):
    plt.figure(figsize=(12, 12))
    x_1 = np.linspace(-2, 4, 10)
    x_2 = -(w1 * x_1 + b) / w2
    x_p = -(w1 * x_1 + b + 1) / w2
    x_n = -(w1 * x_1 + b - 1) / w2

    plt.plot(x_1, x_2, label="Hyperplane WX+B=0")
    plt.plot(x_1, x_p, label="+ve Hyperplane WX+B=1")
    plt.plot(x_1, x_n, label="-ve Hyperplane WX+B=-1")
    plt.legend()
    plt.scatter(X[:,0], X[:,1], c=Y)
    plt.xlabel('X1')
    plt.ylabel('X2')
    plt.title('Hyperplanes and Data Points')
    plt.grid(True)
    plt.show()

# Usage:
plotHyperplane(W[0,0], W[0,1], b, X, Y)
```



Model Interpretation:

Why Gradient Descent?

- We are using gradient descent to optimize the model parameters (W and b) that minimize the hinge loss function. By using gradient descent, we can find the optimal hyperplane that separates the two classes of data points.

What does loss value explain?

- 17162 is the loss value of our model after training. Earlier in the first iteration we had 400000 which is significantly high, and after multiple training we have 17162, approxiately the loss value has been reduced by 95.71 %.