PAGE :
DATE : / /

SVM

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

class SVM:
    def __init__(self, learning_rate=0.001,
                 lambda_param=0.01, n_iters=1000):

        self.lr = learning_rate
        self.lambda_param = lambda_param
        self.n_iters = n_iters
        self.w = None
        self.b = None
    def fit(self, x, y):
        y = np.where(y <= 0, -1, 1)
        n_samples, n_features = x.shape
        self.w = np.zeros(n_features)
        self.b = 0
        for _ in range(self.n_iters):
            for idx, x_i in enumerate(x):
                condition = y[idx] * (np.dot(x_i,
                    self.w) + self.b) >= 1
                if condition:
                    self.w -= self.lr * (2 * self.
                        lambda_param * self.w)
                else:
                    self.w -= self.lr * (2 * self.
                        lambda_param * self.w -
                        np.dot(x_i, y[idx]))
```

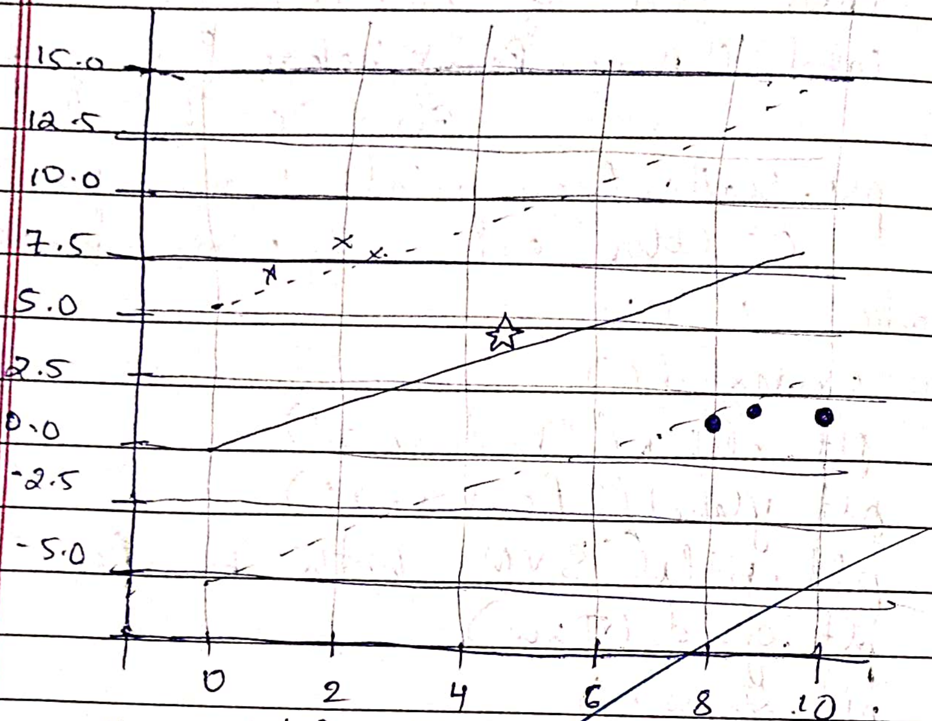
PAGE :
DATE : / /

```
        self.b += self.lr * y[idx]
def predict(self, x):
        approx = np.dot(x, self.w) + self.b
        return np.sign(approx)
def visualize(self, x, y, new_point=None,
                    prediction=None):
    def get_hyperplane(x, w, b, offset):
        return (-w[0] * x + b + offset) /
                    w[1]

    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)

    for i, sample in enumerate(X):
        if y[i] == 1:
            plt.scatter(sample[0], sample[1],
    marker='o', color='blue', label='class +1' if
    i == 0 else "")
        else:

            plt.scatter(sample[0], sample[1],
    marker='x', color='red', label='class -1' if i==0
                    else "")
    x0 = np.linspace(np.min(x[:, 0]) - 1, np.max
                (x[:, 0]) + 1, 100)
    x1 = get_hyperplane(x0, self.w, self.b, 0)
    x1_m = get_hyperplane(x0, self.w, self.b, -1)
    x1_p = get_hyperplane(x0, self.w, self.b, 1)

    ax.plot(x0, x1, 'k-', label='Decision Bound')
    ax.plot(x0, x1_m, 'k--', label='Margin')
    ax.plot(x0, x1_p, 'k--')
if newpoint is not None:
```

PAGE:
DATE: / /

```
colos = 'green' if prediction == 1 else 'orang'
label = f'New Point : class {"1" if prediction==1
                  else "0" }'
plt.scatter(new_point[0], new_point[1],
      c = colos, s=100, edgecolors='black', label=label,
 marker= "*")
 ax.legend()
 plt.xlabel("Feature 1")
 plt.ylabel("Feature 2')
 plt.title("SVM withe new Point prediction')
 plt.grid(True)
 plt.show()

if __name__ == "__main__":
    x = np.array([[
            [1, 7],
            [2, 8],
            [3, 8],
            [8, 1],
            [9, 2],
            [10, 2]
    ])
y= np.array([0, 0, 0, 1, 1, 1])
newpoint = np.array([[5, 5]])
svm = SVM()
svm.fit(X, Y)
prediction = svm.predict(new_point)[0]
svm.visualize(X, Y, new_point=new_point[0], prediction-prediction)
print(f "new point {new_point[0]} classified as: {'class 1' if
                prediction == 1 else 'class 0'}")
```

o/p



New point [5 5] classified as: class 0