**PRACTICAL – 7**

**#WAP in python for support vector machine**

import numpy as np

import cvxopt

from sklearn.datasets.samples\_generator import make\_blobs

from sklearn.model\_selection import train\_test\_split

from matplotlib import pyplot as plt

from sklearn.svm import LinearSVC

from sklearn.metrics import confusion\_matrix

class SVM:

def fit(self, X, y):

n\_samples, n\_features = X.shape

K = np.zeros((n\_samples, n\_samples))

for i in range(n\_samples):

for j in range(n\_samples):

K[i,j] = np.dot(X[i], X[j])

P = cvxopt.matrix(np.outer(y, y) \* K)

q = cvxopt.matrix(np.ones(n\_samples) \* -1)

A = cvxopt.matrix(y, (1, n\_samples))

b = cvxopt.matrix(0.0)

G = cvxopt.matrix(np.diag(np.ones(n\_samples) \* -1))

h = cvxopt.matrix(np.zeros(n\_samples))

solution = cvxopt.solvers.qp(P, q, G, h, A, b)

a = np.ravel(solution['x'])

sv = a > 1e-5

ind = np.arange(len(a))[sv]

self.a = a[sv]

self.sv = X[sv]

self.sv\_y = y[sv]

self.b = 0

for n in range(len(self.a)):

self.b += self.sv\_y[n]

self.b -= np.sum(self.a \* self.sv\_y \* K[ind[n], sv])

self.b /= len(self.a)

self.w = np.zeros(n\_features)

for n in range(len(self.a)):

self.w += self.a[n] \* self.sv\_y[n] \* self.sv[n]

def project(self, X):

return np.dot(X, self.w) + self.b

def predict(self, X):

return np.sign(self.project(X))

X, y = make\_blobs(n\_samples=250, centers=2,random\_state=0, cluster\_std=0.60)

y[y == 0] = -1

tmp = np.ones(len(X))

y = tmp \* y

plt.scatter(X[:, 0], X[:, 1], c=y, cmap='winter')

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0)

def f(x, w, b, c=0):

return (-w[0] \* x - b + c) / w[1]

plt.scatter(X\_train[:, 0], X\_train[:, 1], c=y\_train, cmap='winter')

a0 = -4; a1 = f(a0, svm.w, svm.b)

b0 = 4; b1 = f(b0, svm.w, svm.b)

plt.plot([a0,b0], [a1,b1], 'k')

a0 = -4; a1 = f(a0, svm.w, svm.b, 1)

b0 = 4; b1 = f(b0, svm.w, svm.b, 1)

plt.plot([a0,b0], [a1,b1], 'k--')

a0 = -4; a1 = f(a0, svm.w, svm.b, -1)

b0 = 4; b1 = f(b0, svm.w, svm.b, -1)

plt.plot([a0,b0], [a1,b1], 'k--')

y\_pred = svm.predict(X\_test)

confusion\_matrix(y\_test, y\_pred)

svc = LinearSVC()

svc.fit(X\_train, y\_train)

plt.scatter(X\_train[:, 0], X\_train[:, 1], c=y\_train, cmap='winter');

ax = plt.gca()

xlim = ax.get\_xlim()

w = svc.coef\_[0]

a = -w[0] / w[1]

xx = np.linspace(xlim[0], xlim[1])

yy = a \* xx - svc.intercept\_[0] / w[1]

plt.plot(xx, yy)

yy = a \* xx - (svc.intercept\_[0] - 1) / w[1]

plt.plot(xx, yy, 'k--')

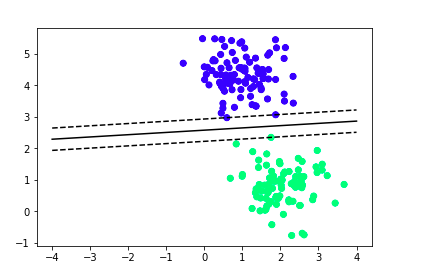
yy = a \* xx - (svc.intercept\_[0] + 1) / w[1]

plt.plot(xx, yy, 'k--')

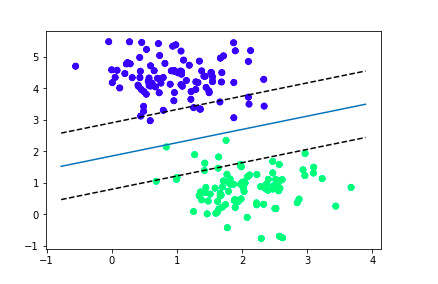
y\_pred = svc.predict(X\_test)

confusion\_matrix(y\_test, y\_pred)

**OUTPUT:**

****

After training our model

****