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
```

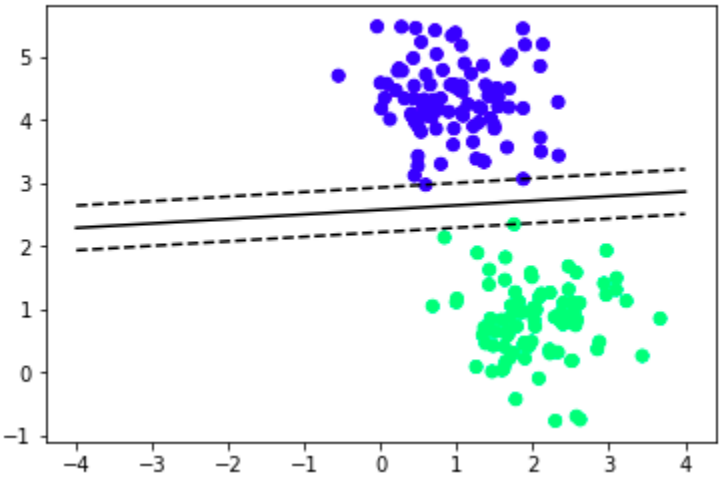
```
ind = np.arange(len(a))[sv]
self.a = a[sv]
self.sv = X[sv]
self.sv_y = y[sv]
self.b = 0
for n in range(len(self.a)):
        self.b += self.sv_y[n]
        self.b -= np.sum(self.a * self.sv_y * K[ind[n], sv])
        self.b /= len(self.a)
        self.w = np.zeros(n_features)
for n in range(len(self.a)):
        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