MACHINE LEARINING

LAB ASSESSMENT – IV

15BCE0082 VOLETI RAVI

**CODE:**

import numpy as np

from numpy import linalg

import cvxopt

import cvxopt.solvers

def linear\_kernel(x1, x2):

return np.dot(x1, x2)

def polynomial\_kernel(x, y, p=3):

return (1 + np.dot(x, y)) \* p

def gaussian\_kernel(x, y, sigma=5.0):

return np.exp(-linalg.norm(x-y)\*\*2 / (2 \* (sigma \*\* 2)))

class SVM(object):

def \_\_init\_\_(self, kernel=linear\_kernel, C=None):

self.kernel = kernel

self.C = C

if self.C is not None: self.C = float(self.C)

def fit(self, X, y):

n\_samples, n\_features = X.shape

# Gram matrix

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

for i in range(n\_samples):

for j in range(n\_samples):

K[i,j] = self.kernel(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)

if self.C is None:

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

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

else:

tmp1 = np.diag(np.ones(n\_samples) \* -1)

tmp2 = np.identity(n\_samples)

G = cvxopt.matrix(np.vstack((tmp1, tmp2)))

tmp1 = np.zeros(n\_samples)

tmp2 = np.ones(n\_samples) \* self.C

h = cvxopt.matrix(np.hstack((tmp1, tmp2)))

# solve QP problem

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

# Lagrange multipliers

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

# Support vectors are non zero lagrange multipliers

sv = a > 1e-5

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

self.a = a[sv]

self.sv = X[sv]

self.sv\_y = y[sv]

print "%d support vectors out of %d points" % (len(self.a), n\_samples)

# Intercept

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)

# Weight vector

if self.kernel == linear\_kernel:

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]

else:

self.w = None

def project(self, X):

if self.w is not None:

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

else:

y\_predict = np.zeros(len(X))

for i in range(len(X)):

s = 0

for a, sv\_y, sv in zip(self.a, self.sv\_y, self.sv):

s += a \* sv\_y \* self.kernel(X[i], sv)

y\_predict[i] = s

return y\_predict + self.b

def predict(self, X):

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

if \_\_name\_\_ == "\_\_main\_\_":

import pylab as pl

def gen\_lin\_separable\_data():

# generate training data in the 2-d case

mean1 = np.array([0, 2])

mean2 = np.array([2, 0])

cov = np.array([[0.8, 0.6], [0.6, 0.8]])

X1 = np.random.multivariate\_normal(mean1, cov, 100)

y1 = np.ones(len(X1))

X2 = np.random.multivariate\_normal(mean2, cov, 100)

y2 = np.ones(len(X2)) \* -1

return X1, y1, X2, y2

def gen\_non\_lin\_separable\_data():

mean1 = [-1, 2]

mean2 = [1, -1]

mean3 = [4, -4]

mean4 = [-4, 4]

cov = [[1.0,0.8], [0.8, 1.0]]

X1 = np.random.multivariate\_normal(mean1, cov, 50)

X1 = np.vstack((X1, np.random.multivariate\_normal(mean3, cov, 50)))

y1 = np.ones(len(X1))

X2 = np.random.multivariate\_normal(mean2, cov, 50)

X2 = np.vstack((X2, np.random.multivariate\_normal(mean4, cov, 50)))

y2 = np.ones(len(X2)) \* -1

return X1, y1, X2, y2

def gen\_lin\_separable\_overlap\_data():

# generate training data in the 2-d case

mean1 = np.array([0, 2])

mean2 = np.array([2, 0])

cov = np.array([[1.5, 1.0], [1.0, 1.5]])

X1 = np.random.multivariate\_normal(mean1, cov, 100)

y1 = np.ones(len(X1))

X2 = np.random.multivariate\_normal(mean2, cov, 100)

y2 = np.ones(len(X2)) \* -1

return X1, y1, X2, y2

def split\_train(X1, y1, X2, y2):

X1\_train = X1[:90]

y1\_train = y1[:90]

X2\_train = X2[:90]

y2\_train = y2[:90]

X\_train = np.vstack((X1\_train, X2\_train))

y\_train = np.hstack((y1\_train, y2\_train))

return X\_train, y\_train

def split\_test(X1, y1, X2, y2):

X1\_test = X1[90:]

y1\_test = y1[90:]

X2\_test = X2[90:]

y2\_test = y2[90:]

X\_test = np.vstack((X1\_test, X2\_test))

y\_test = np.hstack((y1\_test, y2\_test))

return X\_test, y\_test

def plot\_margin(X1\_train, X2\_train, clf):

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

# given x, return y such that [x,y] in on the line

# w.x + b = c

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

pl.plot(X1\_train[:,0], X1\_train[:,1], "ro")

pl.plot(X2\_train[:,0], X2\_train[:,1], "bo")

pl.scatter(clf.sv[:,0], clf.sv[:,1], s=100, c="g")

# w.x + b = 0

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

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

pl.plot([a0,b0], [a1,b1], "k")

# w.x + b = 1

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

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

pl.plot([a0,b0], [a1,b1], "k--")

# w.x + b = -1

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

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

pl.plot([a0,b0], [a1,b1], "k--")

pl.axis("tight")

pl.show()

def plot\_contour(X1\_train, X2\_train, clf):

pl.plot(X1\_train[:,0], X1\_train[:,1], "ro")

pl.plot(X2\_train[:,0], X2\_train[:,1], "bo")

pl.scatter(clf.sv[:,0], clf.sv[:,1], s=100, c="g")

X1, X2 = np.meshgrid(np.linspace(-6,6,50), np.linspace(-6,6,50))

X = np.array([[x1, x2] for x1, x2 in zip(np.ravel(X1), np.ravel(X2))])

Z = clf.project(X).reshape(X1.shape)

pl.contour(X1, X2, Z, [0.0], colors='k', linewidths=1, origin='lower')

pl.contour(X1, X2, Z + 1, [0.0], colors='grey', linewidths=1, origin='lower')

pl.contour(X1, X2, Z - 1, [0.0], colors='grey', linewidths=1, origin='lower')

pl.axis("tight")

pl.show()

def test\_linear():

X1, y1, X2, y2 = gen\_lin\_separable\_data()

X\_train, y\_train = split\_train(X1, y1, X2, y2)

X\_test, y\_test = split\_test(X1, y1, X2, y2)

clf = SVM()

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

y\_predict = clf.predict(X\_test)

correct = np.sum(y\_predict == y\_test)

print "%d out of %d predictions correct" % (correct, len(y\_predict))

plot\_margin(X\_train[y\_train==1], X\_train[y\_train==-1], clf)

def test\_non\_linear():

X1, y1, X2, y2 = gen\_non\_lin\_separable\_data()

X\_train, y\_train = split\_train(X1, y1, X2, y2)

X\_test, y\_test = split\_test(X1, y1, X2, y2)

clf = SVM(gaussian\_kernel)

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

y\_predict = clf.predict(X\_test)

correct = np.sum(y\_predict == y\_test)

print "%d out of %d predictions correct" % (correct, len(y\_predict))

plot\_contour(X\_train[y\_train==1], X\_train[y\_train==-1], clf)

def test\_soft():

X1, y1, X2, y2 = gen\_lin\_separable\_overlap\_data()

X\_train, y\_train = split\_train(X1, y1, X2, y2)

X\_test, y\_test = split\_test(X1, y1, X2, y2)

clf = SVM(C=0.1)

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

y\_predict = clf.predict(X\_test)

correct = np.sum(y\_predict == y\_test)

print "%d out of %d predictions correct" % (correct, len(y\_predict))

plot\_contour(X\_train[y\_train==1], X\_train[y\_train==-1], clf)

test\_soft()



