PURVANG LAPSIWALA

UIN = 662689378

ANSWER (1):

The given problem is to design SVM, which can classify any given pattern with the help of using kernels if needed. To solve this example, I am using CVXOPT library in python to solve maximization problem of Lagrange's multipliers.

Quadratic Programming is one of the numerical optimization method.

Quadratic programming has standard form as

$$\begin{array}{rl} \text{minimize} & (1/2)x^T P x + q^T x \\ \text{subject to} & Gx \le h \\ & Ax = b \end{array}$$

Main challenge was to convert out problem to standard problem and instead of minimizing given problem, we have to maximize it.

So, P is replaced by $d_i d_j k(x_i x_j)$

X is replaced by alpha and q is replaced by matrix containing all ones.

To convert problem into maximization, apply minus sign to entire equation.

For constrain conversion, G is also replace with matrix containing ones A is replaced by desired output matrix.

By doing so, we can successfully set up our problem to solve.

Kernel used for this problem is polynomial kernel. By using different value for polynomial exponent, found 5 as good value. If polynomial exponent value is increase too much, it is observed that it could lead to over fitting.

Python Code:

```
import numpy as np

import cvxopt

import cvxopt.solvers

import matplotlib.pyplot as plt



n_samples = 100

x = np.random.uniform(0,1,(n_samples ,2))

d = []

c1 = []

c2 = []

for i in range(n_samples):

   if x[i][1] < (0.2 * np.sin(10*x[i][0])) + 0.3:

      d.append(1)

      c1.append(x[i])

   elif (x[i][1] - 0.8)**2 + (x[i][0] - 0.5)**2 < 0.15**2:

      d.append(1)

      c1.append(x[i])

   else:

      d.append(-1)

      c2.append(x[i])


def polynomial_kernel(x, y, p=5):

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


y = np.asarray(d).astype(float)


K = np.zeros((n_samples, n_samples))
```

```
for i in range(n_samples):
    for j in range(n_samples):
        K[i,j] = polynomial_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)
G = cvxopt.matrix(np.diag(np.ones(n_samples) * -1))
h = cvxopt.matrix(np.zeros(n_samples))

# solve QP problem
solution = cvxopt.solvers.qp(P, q, G, h, A, b)

# Lagrange multipliers
alpha = np.ravel(solution['x'])
svs = alpha > 1e-5
sv1_x = []
sv1_y = []
sv2_x = []
sv2_y = []
for i in range(n_samples):
    if alpha[i]>1e-5:
        if y[i] == 1:
            sv1_x.append(x[i])
            sv1_y.append(y[i])
        if y[i] == -1:
            sv2_x.append(x[i])
            sv2_y.append(y[i])
```

```
sv_x = sv1_x + sv2_x
sv_y = sv1_y + sv2_y

theta = sv_y[1]
for i in range(n_samples):
    theta -= alpha[i]*y[i]*polynomial_kernel(x[i], sv_x[1])

x_coord = np.linspace(0.0, 1.0, num=1000)
y_coord = np.linspace(0.0, 1.0, num=1000)
h = []
h_plus = []
h_minus = []

for i in range(len(x_coord)):
    for j in range(len(y_coord)):
        descriminant = theta
        for k in range(n_samples):
            descriminant += alpha[k]*y[k]*polynomial_kernel(x[k], np.asarray([x_coord[i], y_coord[j]]))
        if -0.1 < descriminant < 0.1:
            h.append([x_coord[i], y_coord[j]])
        elif 0.9 < descriminant < 1.1:
            h_plus.append([x_coord[i], y_coord[j]])
        elif -1.1 < descriminant < -0.9:
            h_minus.append([x_coord[i], y_coord[j]])

fig, ax = plt.subplots(figsize=(10,10))
plt.scatter(*zip(*c1), c = 'red', label = 'Class 1')
plt.scatter(*zip(*c2), c = 'green', label = 'Class -1')
```

```
plt.scatter(*zip(*h_plus), c = 'red',s=1, label = 'Hyperplane 1')
plt.scatter(*zip(*h), c = 'blue',s=1, label = 'Margin')
plt.scatter(*zip(*h_minus), c = 'green', s=1, label = 'Hyperplane -1')
plt.scatter(*zip(*sv_x), facecolors = 'yellow', edgecolors='black',label='Support Vectors')
plt.legend(loc = 'best')
plt.show()
```