Problem 4: Support Vector Machines

Find a perfect classifier for this data set using support vector machines. Your solution should explain the optimization problem that you solved and provide the learned parameters, the optimal margin, and the support vectors.

$$\max_\lambda \sum_{i=1}^{m} \lambda_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y_i y_j \lambda_i \lambda_j x_i^T x_j$$
$$\text{s.t.} \quad \forall i : \alpha_i \geq 0 \wedge \sum_{i=1}^{m} y_i \alpha_i = 0$$

Solving the above Dual problem of SVM using Quadratic programming, we obtain the Optimal Lagrange multipliers λ^*.

Using the Optimal Lagrange multipliers λ^*, we can compute

- support vectors $x^{(i)}$ at $\lambda_i > 0$
- weights $w = \sum_{i=1}^{m} \lambda_i y^{(i)} x^{(i)}$
- bias $b = (1/y^{(i)}) - w^T x^{(i)}$
- optimal margin $margin = \frac{1}{\|w\|}$

In [1]:

```
import numpy as np
import cvxopt
import cvxopt.solvers
```

In [2]:

```
mystery = np.loadtxt('mystery.data', delimiter=',')
X = mystery[:, 0:4]
Y = mystery[:, 4:]
```

Transforming Input to higher dimension using the Feature vector

$$\phi(x_0, x_1, x_2, x_3) = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_0^2 \\ x_1^2 \\ x_2^2 \\ x_3^2 \end{pmatrix}$$

In [3]:

```
# Increase the dimension of the input using feature vector
X = np.hstack((X, X*X))
```

In [4]:

```
def SVM(X, Y):
    m, n = X.shape
    # Create Gram matrix
    K = np.zeros((m, m))
    for i in range(m):
        for j in range(m):
            K[i,j] = np.dot(X[i], X[j])

    # P = Combination of Yi Yj Xi Xj
    P = cvxopt.matrix(np.outer(Y,Y) * K)
    q = cvxopt.matrix(np.ones(m) * -1)

    # Constraints λY = 0
    A = cvxopt.matrix(Y, (1,m))
    b = cvxopt.matrix(0.0)
    # λ >= 0
    G = cvxopt.matrix(np.diag(np.ones(m) * -1))
    h = cvxopt.matrix(np.zeros(m))

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

    # Solver produces λ
    λ = np.array(solution['x'])
    idx = (λ > 1e-6).nonzero()[0]

    w = np.zeros(X.shape[1])
    for i in range(len(λ)):
        w += Y[i] * λ[i] * X[i]

    b = []
    for i in idx:
        b.append((1 / Y[i]) - (np.dot(w, X[i])))

    support_vectors = X[idx]

    return w, b, support_vectors
```

In [5]:

```
w, b, support_vectors = SVM(X,Y)
```

```
     pcost       dcost       gap    pres   dres
 0: -2.5311e+02 -6.0405e+02  4e+03  5e+01  2e+00
 1: -6.5476e+02 -8.8230e+02  2e+03  3e+01  1e+00
 2: -1.3010e+03 -1.5700e+03  2e+03  3e+01  1e+00
 3: -4.0749e+03 -4.5796e+03  2e+03  2e+01  1e+00
 4: -6.7789e+03 -7.4971e+03  3e+03  2e+01  1e+00
 5: -2.1814e+04 -2.3205e+04  4e+03  2e+01  1e+00
 6: -5.2785e+04 -5.5669e+04  6e+03  2e+01  1e+00
 7: -4.6671e+05 -4.7900e+05  2e+04  2e+01  1e+00
 8: -1.4875e+06 -1.5242e+06  5e+04  2e+01  1e+00
 9: -2.2870e+06 -2.3652e+06  1e+05  2e+01  1e+00
10: -5.3817e+06 -5.7337e+06  4e+05  2e+01  1e+00
11: -9.5744e+06 -1.0655e+07  1e+06  2e+01  1e+00
12: -2.1129e+07 -2.6050e+07  5e+06  2e+01  9e-01
13: -3.8938e+07 -5.4585e+07  2e+07  2e+00  1e-01
14: -4.0657e+07 -4.0847e+07  2e+05  2e-02  9e-04
15: -4.0683e+07 -4.0685e+07  2e+03  2e-04  9e-06
16: -4.0683e+07 -4.0683e+07  2e+01  2e-06  1e-07
17: -4.0683e+07 -4.0683e+07  2e-01  2e-08  7e-08
Optimal solution found.
```

In [6]:

```
print('Weights:\n', w)
```

```
Weights:
 [  477.60571516   -41.6590162    692.21551248 -8337.44743044
  2372.74410874    89.38381436   163.13728297  2340.94969885]
```

In [7]:

```
# As discussed in lecture, solvers yeild multiple biases
print('Biases:')
print([i[0] for i in b])
```

```
Biases:
[4090.100284005357, 4095.7957858383616, 4090.1002840748884, 4090.100284023
0203, 4060.792269600218, 4090.1002840924, 4071.602960761495, 4088.30054986
66307, 4090.1002840125784, 4122.735333723589, 4090.1002840258648, 4090.100
284003734, 4122.1608159281, 4090.1002840013534, 4101.268473076709, 4090.10
0283995995]
```

In [8]:

```
print('Optimal Margin: ', 1/np.linalg.norm(w))
```

```
Optimal Margin:  0.00011086052243733817
```

In [9]:

```python
print('Support vectors: ', len(support_vectors))
for i in support_vectors:
    print(i)
```

```
Support vectors:  16
[1.85298680e-02 5.07821481e-01 7.13645133e-01 6.97747974e-01
 3.43356008e-04 2.57882657e-01 5.09289376e-01 4.86852235e-01]
[0.32838322 0.11309668 0.77191448 0.79256252 0.10783554 0.01279086
 0.59585196 0.62815535]
[0.35339144 0.83894779 0.13388726 0.69814794 0.12488551 0.70383339
 0.0179258  0.48741055]
[6.77600524e-01 1.73155580e-02 2.26607511e-01 9.14466038e-01
 4.59142470e-01 2.99828549e-04 5.13509640e-02 8.36248135e-01]
[0.02751111 0.61371726 0.42361196 0.64529572 0.00075686 0.37664887
 0.1794471  0.41640656]
[0.56116788 0.94932549 0.95929542 0.9899408  0.31490939 0.9012189
 0.92024771 0.97998279]
[0.62291456 0.84209279 0.378042   0.89868175 0.38802254 0.70912027
 0.14291575 0.80762889]
[0.06156021 0.65899664 0.0993981  0.60866886 0.00378966 0.43427657
 0.00987998 0.37047779]
[0.19385208 0.83338556 0.74510427 0.74385454 0.03757863 0.69453149
 0.55518037 0.55331957]
[0.26674025 0.12199103 0.15085275 0.66790593 0.07115036 0.01488181
 0.02275655 0.44609833]
[7.74849111e-01 5.09021209e-01 2.57518000e-02 9.75527528e-01
 6.00391145e-01 2.59102591e-01 6.63155203e-04 9.51653958e-01]
[0.16933499 0.62115084 0.42461128 0.67708347 0.02867434 0.38582836
 0.18029473 0.45844202]
[0.19660474 0.60903473 0.52287111 0.70673033 0.03865342 0.3709233
 0.2733942  0.49946776]
[2.42704748e-01 1.89393990e-02 9.82653180e-01 8.02619526e-01
 5.89055947e-02 3.58700834e-04 9.65607272e-01 6.44198104e-01]
[0.50743509 0.14759421 0.92070003 0.92946127 0.25749038 0.02178405
 0.84768855 0.86389825]
[6.02701480e-01 6.25450545e-01 1.96828890e-02 8.19732602e-01
 3.63249074e-01 3.91188384e-01 3.87416119e-04 6.71961539e-01]
```