Math3017 Coursework

Chenyang Wu, Student ID:32588925

June 19, 2024

1 Q1-1

Let x_i be the production increased in month i (in thousands), y_i production decreased in month i (in thousands) and I_i be the inventory at the end of month i (in thousands). Let $i = 1 \dots 4$ be October, November, December and January respectively.

Objective function: min $\sum_{i=1}^{4}(2x_i + 0.5y_i + I_i)$

subject to:

$I_1 = 12 + x_1 - y_1 - 10$

$I_2 = 12 + x_1 - y_1 + x_2 - y_2 + I_1 - 16$

$I_3 = 12 + x_1 - y_1 + x_2 - y_2 + x_3 - y_3 + I_1 + I_2 - 10$

$I_4 = 12 + x_1 - y_1 + x_2 - y_2 + x_3 - y_3 + x_4 - y_4 + I_1 + I_2 + I_3 - 12$

$I_4 = 0$

$x_i, y_i \geq 0 \quad \forall i = 1 \cdots 4$

$I_i \geq 0 \quad \forall i = 1 \cdots 4$

By combining identical terms inside the functions, we get:

Objective function: min $\sum_{i=1}^{4}(2x_i + 0.5y_i + I_i)$

subject to:

$$-x_1 + y_1 + I_1 = 2 \tag{1}$$
$$-x_1 - x_2 + y_1 + y_2 - I_1 + I_2 = -4 \tag{2}$$
$$-x_1 - x_2 - x_3 + y_1 + y_2 + y_3 - I_1 - I_2 + I_3 = 2 \tag{3}$$
$$-x_1 - x_2 - x_3 - x_4 + y_1 + y_2 + y_3 + y_4 - I_1 - I_2 - I_3 + I_4 = 0 \tag{4}$$
$$I_4 = 0 \tag{5}$$

$x_i, y_i \geq 0 \quad \forall i = 1 \cdots 4$

$I_i \geq 0 \quad \forall i = 1 \cdots 4$

2 Q1-2

[1]:
```
from cvxopt import matrix, solvers
import numpy as np
solvers.options['show_progress'] = False

c = matrix([2.,2.,2.,2.,0.5,0.5,0.5,0.5,1.,1.,1.,1.])

size = 12
diagonal = -1.
```

```
M = matrix(np.diag([diagonal]*(size)))

h = matrix([0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.])

G = matrix([[-1., 0., 0., 0.,1.,0.,0.,0., 1., 0., 0.,0.],
            [-1.,-1., 0., 0.,1.,1.,0.,0.,-1., 1., 0.,0.],
            [-1.,-1.,-1., 0.,1.,1.,1.,0.,-1.,-1., 1.,0.],
            [-1.,-1.,-1.,-1.,1.,1.,1.,1.,-1.,-1.,-1.,1.],
            [0.  ,0.  , 0., 0.,0.,0.,0.,0., 0., 0., 0.,1.]]).T


p = matrix([2000.,-4000.,2000.,0.,0.])

sol = solvers.lp(c,M,h,G,p)
print(sol['x'])
print('the value of the primal objective function is', sol['primal␣
 ↪objective'])
print('the value of the dual objective function is', sol['dual objective'])
print('the value of the slackness condition is', sol['gap'])
```

```
[ 1.00e+03]
[ 3.74e-09]
[-8.57e-11]
[ 1.23e-10]
[-5.23e-10]
[-2.15e-10]
[ 5.00e+03]
[-9.85e-11]
[ 3.00e+03]
[-2.07e-10]
[ 1.00e+03]
[ 4.94e-25]

the value of the primal objective function is 8499.999999825435
the value of the dual objective function is 8500.00000000112
the value of the slackness condition is 1.1317455252610281e-08
```

3 Q2-1

Objective Function

The primal form of the Support Vector Machine (SVM) problem is given by:

$$\min_{\beta,\beta_0,\xi} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{n}\xi_i \tag{6}$$

Here, ξ are slack variables representing the non-negative cost of misclassification of data points.

Constraints

The constraints for the SVM are as follows:

- For each data point: $y_i(\beta^T x_i + \beta_0) \geq 1 - \xi_i$
- Slack variables $\xi_i \geq 0$

Using CVXOPT

To solve this using CVXOPT, we convert the problem into the following quadratic programming (QP) form:

$$\begin{aligned} \min_{x} \quad & \frac{1}{2}x^T P x + q^T x \\ \text{subject to} \quad & Gx \leq h \\ & Ax = b \end{aligned}$$

Here, x includes β, β_0, and the slack variables ξ.

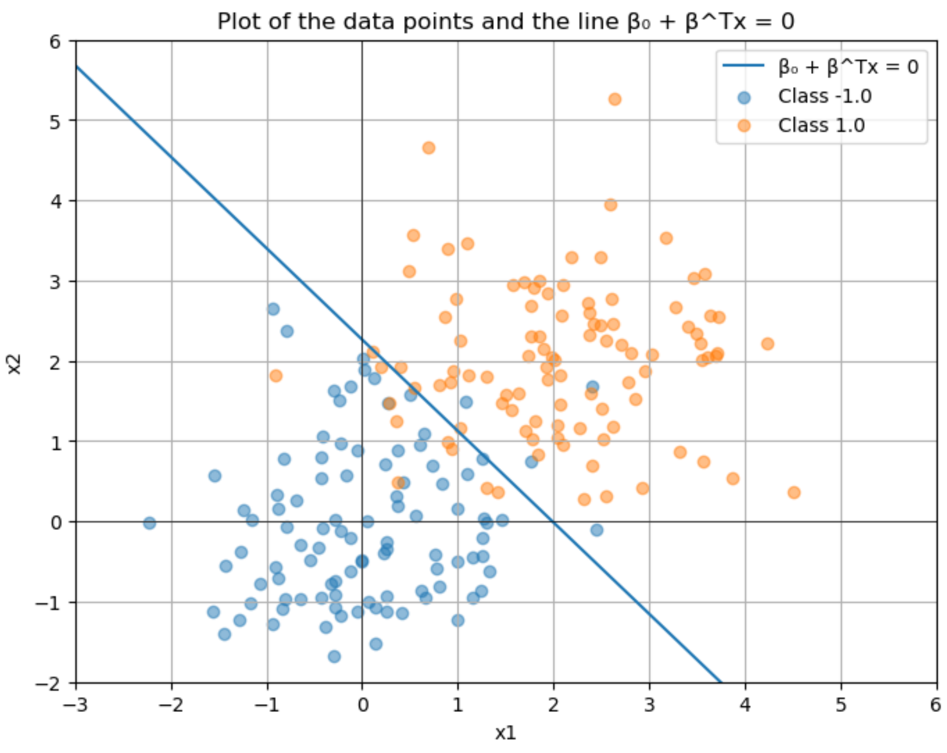
[2]:
```python
import pandas as pd

path = 'xy_train.csv'
data = pd.read_csv(path,header=None)
import numpy as np
from cvxopt import matrix, solvers
X = np.stack((data[0],data[1]),axis=1)
y = np.array(data[2].tolist()).astype(float).reshape(-1,1)
```

[3]:
```python
def solve_svm(X, y, C=1.0):
    n_samples, n_features = X.shape

    P = np.zeros((n_features + 1 + n_samples, n_features + 1 + n_samples))
    P[:n_features, :n_features] = np.eye(n_features)
    q = np.hstack([np.zeros(n_features + 1), C * np.ones(n_samples)])
    G_top = np.hstack([y* X, y, -np.eye(n_samples)])
    G_bottom = np.hstack([np.zeros((n_samples, n_features + 1)), -np.
↪eye(n_samples)])
    G = np.vstack([G_top, G_bottom])

    # h vector setup
    h = np.hstack([-np.ones(n_samples), np.zeros(n_samples)])

    # Convert numpy arrays to cvxopt matrices
    P = matrix(P)
    q = matrix(q)
    G = matrix(G)
    h = matrix(h)

    # Solve QP problem
    solution = solvers.qp(P, q, G, h)

    # Extract solutions for beta and beta0
    beta = np.array(solution['x'][:n_features])
    beta0 = np.array(solution['x'][n_features])

    return beta.ravel(), beta0.item(), solution['primal objective']
b, b_0, optimal = solve_svm(X, y, C=1.0)

print('Optimal criterion value:', optimal)
print('optimal coefficients β:', b)
print('Intercept β_o:', b_0)
```

```
Optimal criterion value: 36.74893289728945
optimal coefficients β: [-1.41967191 -1.24607478]
Intercept β_o: 2.8237275969550932
```

4 Q2-2

```
[4]: import matplotlib.pyplot as plt

     x1_values = np.linspace(-10, 10, 400)
     x2_values = -(b_0 + b[0]*x1_values) / b[1]
     data1 = pd.read_csv('xy_train.csv', header=None)
     x_1 = data1.iloc[:, :-1]
     x_2 = data1.iloc[:, -1]
     plt.figure(figsize=(8, 6))
     plt.plot(x1_values, x2_values, label='β_o + β^Tx = 0')
     for label in np.unique(y):
         plt.scatter(x_1[x_2 == label][0], x_1[x_2 == label][1], label=f'Class␣
      ↪{label}', alpha=0.5)
     plt.xlabel('x1')
     plt.ylabel('x2')
     plt.title('Plot of the data points and the line β_o + β^Tx = 0')
     plt.axhline(0, color='black',linewidth=0.5)
     plt.axvline(0, color='black',linewidth=0.5)
     plt.xlim(-3,6)
     plt.ylim(-2,6)
     plt.grid(True)
     plt.legend()
     plt.show()
```



5 Q2-3

Problem Definition

The dual problem is typically derived from the primal SVM problem using Lagrange multipliers. For a linear SVM, the dual problem aims to maximize the following objective function:

$$\max_{w} \left(-\frac{1}{2} w^T \hat{X}^T \hat{X} w + e^T w \right) \tag{7}$$

Subject to:

$$0 \leq w \leq Ce,$$
$$w^T y = 0$$

where:

- X is the feature matrix $\mathbb{R}^{n \times p}$ augmented by labels y (i.e., each row x_i is scaled by corresponding y_i).
- e is the vector of ones in $\mathbb{R}^n$.
- w are the dual variables corresponding to each data point.
- C is the penalty parameter for the error term.

Solution with CVXOPT

Objective function:

$$\min_{w} \frac{1}{2} w^T P w + q^T w \tag{8}$$

where $P = \hat{X}^T \hat{X}$ and $q = -e$.

Constraints:

- $Gw \leq h$ where G is an identity matrix scaled by -1 and 1 for the constraints $0 \leq w$ and $w \leq C$ respectively, and h is a vector of zeros and $C \times e$.
- $Aw = b$ where $A = y^T$ and $b = 0$ for the constraint $w^T y = 0$.

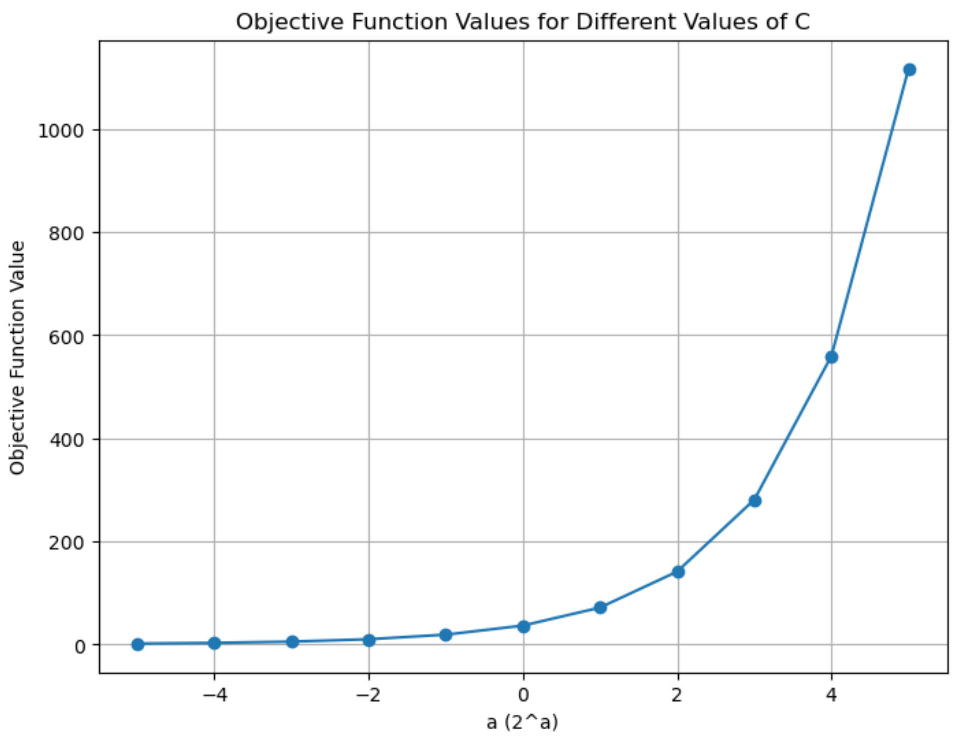
[5]:
```python
def solve_svm_dual(X, y, C=1.0):
    n_samples, n_features = X.shape

    # Create the kernel matrix (since SVM is linear, use dot product)
    K = np.dot(X, X.T)
    P = matrix(np.outer(y, y) * K)
    q = matrix(-np.ones(n_samples))

    # Equality constraints: y^T w = 0
    A = matrix(y, (1, n_samples), 'd')
    b = matrix(0.0)

    # Inequality constraints: 0 <= w_i <= C for all i
    G = matrix(np.vstack((-np.eye(n_samples), np.eye(n_samples))))
    h = matrix(np.hstack((np.zeros(n_samples), np.ones(n_samples) * C)))

    # Solve QP problem
    solution = solvers.qp(P, q, G, h, A, b)
    alphas = np.array(solution['x'])  # Extract solution for alpha
    # Compute the weight vector w using the dual variables
    w = np.sum(alphas * y * X, axis=0)

    # Objective value from the solution
    obj = solution['primal objective']
    # Extract and return the solution
    return  w, obj

w, obj = solve_svm_dual(X, y, C=1.0)

print('Optimal criterion value:', obj)
print('Optimal w:', w)
```

```
Optimal criterion value: -36.748932426022364
Optimal w: [1.41967191 1.24607478]
```

- The optimal value we obtained here in Q2-3 is exactly the negative of the primal objective value obtained in Q2-1. This reflects the property o SVM where the primal and dual formulations are related such that their objective values are negatives of each other due to the transformation used in formulating the dual.
- The optimal weight vector 2 obtained from the dual formulation here is also the negative of the values of β from Q2-1. This again matches the expectations from SVM theory, where the weights derived from the dual problem correspond to the same separating hyperplane as those derived form the primal. But they are expressed in different sign due to the nature of the dual problem.

6 Q2-4

```
[6]: # Iterate over different C values and compute the objective function values
a_values = np.arange(-5, 6)
C_values = 2.0 ** a_values
criterion_values = []
for C in C_values:
    criterion_value = solve_svm(X, y, C)[2]
    criterion_values.append(criterion_value)

# Plot the criterion values against a_values
plt.figure(figsize=(8, 6))
plt.plot(a_values, criterion_values, marker='o')
plt.xlabel('a (2^a)')
plt.ylabel('Objective Function Value')
plt.title('Objective Function Values for Different Values of C')
plt.grid(True)
plt.show()
```



- From the plot above we can see that the objective function values changes with different regularization strength. This reflects the balance between maximizing the margin and minimizing the classification error. For lower values of C, the model is less strict about misclassifications, which might increase the margin but also potentially increase the number of misclassifications. For higher values of C, the model puts a stronger penalty on misclassifications, pushing the optimizer to find solutions that try to correctly classify more or all points. And the higher the C value, the higher the optimal objective function.