MA4270 Computational Exercise Report

Peck Kai Ting
A0130651U

August 12, 2018

1 Problem 1: Perceptron

1.1 Question 1: Plotting

The plot of the dataset is as follows.

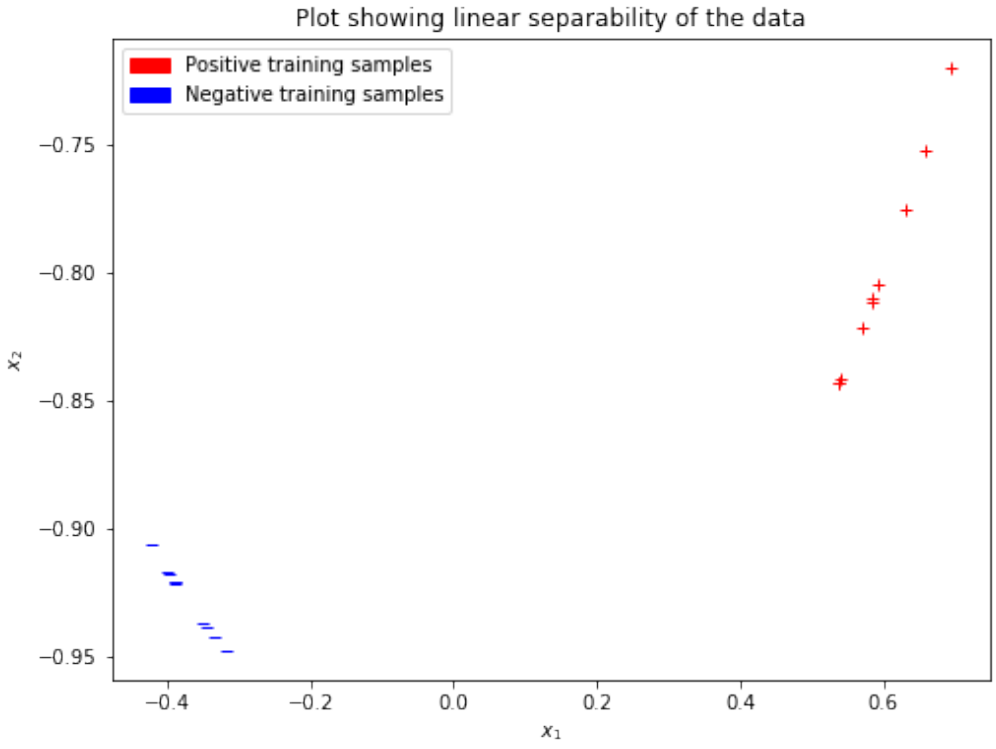

Figure 1: Plot of dataset Problem1.csv showing linear separability

Clearly, from **Figure 1**, the dataset is linearly separable.

1.2 Question 2: Primal SVM

To solve for the optimal θ^*, we solve the primal SVM without offset and without slack variables. A snippet of the code is shown as follows.

```
P = np.identity(p)
q = np.zeros(p)
G = -np.multiply(np.matrix(y).T,X)
h = -1*np.ones(n)

theta_star = quadprog_solve_qp(P, q, G=G, h=h)
# Normalize theta
theta_star = (1.0/np.linalg.norm(theta_star)) * theta_star
# Calculate gamma_star
gamma_star = calculate_gamma(X, y, theta_star, n)
```

Results
$\theta^* = [0.99257345, 0.12164684]^T$
$\gamma^* = 0.4314845982168759$

1.3 Question 3: Standard Perceptron

The code for the standard perceptron algorithm can be found in the appendix.

(a) Run perceptron with zero vector as starting point

Running the algorithm using natural order of the dataset and setting the zero vector as the starting point, for the dataset to be successfully classified,
Number of updates required, $k = 2$
Converged solution, $\theta = [0.98365688, 0.18005314]^T$
Corresponding gamma, $\gamma = 0.37745488374724534$

(b) Run perceptron with different starting points

We generate 10 random starting points from a uniform distribution over [0,1) and run the perceptron using these as starting points.

```
num_iter = 10
for i in range(0,num_iter):
    # Generate theta_zero for each iteration and run perceptron
    theta_zero_i = np.random.rand(2)
    k_i, theta_i, gamma_i = std_perceptron(X, y, theta_zero_i)
```

Results

Iteration	$\theta^{(0)}$	No. of updates k	Converged solution θ	γ
1	[0.67360871, 0.33054755]	0	[0.898,0.441]	0.112
2	[0.92907048, 0.81037498]	1	[1.000,-0.022]	0.297
3	[0.68588925, 0.28404271]	0	[0.924,0.383]	0.175
4	[0.18071892, 0.05301995]	0	[0.960,0.282]	0.279
5	[0.89741097, 0.7844022]	1	[0.999,-0.041]	0.280
6	[0.81425816, 0.4654737]	0	[0.868,0.496]	0.049
7	[0.85759134, 0.53144607]	0	[0.850,0.527]	0.013
8	[0.28672446, 0.32191657]	2	[0.931,0.364]	0.194
9	[0.115096, 0.20357841]	2	[0.945,0.327]	0.233
10	[0.00366389, 0.1913366]	2	[0.937,0.348]	0.211

Generally, for different starting points used, the parameter vectors obtained are different. The gamma values and number of iterations also differ in general. We also note that for the standard perceptron algorithm, the converged solution is not unique, i.e. different starting points and order of data can result in different solutions.

(c) Proof of upper bound

Proof. Suppose $\theta^{(0)}$ is drawn from boundary of the unit circle.

Let $a = \langle \theta^*, \theta^{(0)} \rangle$
$\gamma^* = \min_{t \in \{1,\dots,n\}} y_t \langle \theta^*, x_1 \rangle$
$R = \max_{t \in \{1,\dots,n\}} \|\mathbf{x}_t\|$
Note that for this particular dataset, R = 1

Part 1:

$$\begin{aligned}(\theta^*)^T\theta^{(k)} &= (\theta^*)^T\theta^{(k-1)} + y_t(\theta^*)^T\mathbf{x}_t \\ &\geq (\theta^*)^T\theta^{(k-1)} + \gamma \qquad (since\, \gamma^* = \min_{t \in \{1,\dots,n\}} y_t\langle \theta^*, x_1\rangle)\end{aligned} \tag{1}$$

Hence, by induction on k,

$$\langle \theta^*, \theta^{(k)} \rangle = (\theta^*)^T\theta^{(k)} \geq (\theta^*)^T\theta^{(0)} + k\gamma = \langle \theta^*, \theta^{(0)} \rangle + k\gamma \tag{2}$$

Part 2:

$$\begin{aligned}\|\theta^{(k)}\|^2 &= \|\theta^{(k-1)} + y_t\mathbf{x}_t\|^2 \\ &= \|\theta^{(k-1)}\|^2 + 2y_t(\theta^{(k-1)})^T\mathbf{x}_t + \|\mathbf{x}_t\|^2 \\ &\leq \|\theta^{(k-1)}\|^2 + \|\mathbf{x}_t\|^2 \\ &\leq \|\theta^{(k-1)}\|^2 + R^2 \\ &= \|\theta^{(k-1)}\|^2 + 1 \qquad (since\, R = 1)\end{aligned} \tag{3}$$

where the third last line holds since updates are made only when $y_t(\theta^{(k-1)})^T\mathbf{x}_t < 0$.
Hence, by induction on k, and since $\theta^{(0)}$ lies on the boundary of the circle i.e. $\|\theta^{(0)}\| = 1$

$$\|\theta^{(k)}\|^2 \leq \|\theta^{(0)}\|^2 + k = 1 + k \tag{4}$$

By the Cauchy-Schwarz inequality, applying (2) and (4) and the fact that $\|\theta^*\| = 1$ since θ^* has been normalized,

$$\begin{aligned}1 &\geq \frac{\langle \theta^*, \theta^{(k)} \rangle}{\|\theta^*\|\|\theta^{(k)}\|} \\ &\geq \frac{\langle \theta^*, \theta^{(0)} \rangle + k\gamma}{\sqrt{1+k}} \qquad (since\, \|\theta^*\| = 1) \\ &= \frac{a + k\gamma^*}{\sqrt{1+k}}\end{aligned} \tag{5}$$

Hence, we have

$$\sqrt{1+k} \geq a + k\gamma^* \tag{6}$$

Case 1: $a + k\gamma^* \leq 0$.
Then, since $\gamma^* > 0$, we have the following inequality:

$$k \leq -\frac{a}{\gamma^*} \tag{7}$$

Case 2: $a + k\gamma^* > 0$
Provided that $a + k\gamma^* > 0$, squaring on both sides gives

$$1 + k \geq (a + k\gamma^*)^2 = a^2 + 2ak\gamma^* + k^2(\gamma^*)^2 \tag{8}$$

$$k^2(\gamma^*)^2 + k(2a\gamma^* - 1) + (a^2 - 1) \leq 0 \tag{9}$$

Solving for the roots of the quadratic equation with respect to k, we obtain the roots k_0

$$\begin{aligned}k_0 &= \frac{-(2a\gamma^* - 1) \pm \sqrt{(2a\gamma^* - 1)^2 - 4(\gamma^*)^2(a^2 - 1)}}{2(\gamma^*)^2} \\ &= \frac{1 - 2a\gamma^* \pm \sqrt{(2a\gamma^* - 1)^2 - 4(\gamma^*)^2(a^2 - 1)}}{2(\gamma^*)^2}\end{aligned} \tag{10}$$

Hence,

$$k \leq \frac{1 - 2a\gamma^* + \sqrt{(2a\gamma^* - 1)^2 - 4(\gamma^*)^2(a^2 - 1)}}{2(\gamma^*)^2} \tag{11}$$

Combining the upper bounds for k for both cases in (7) and (11),

$$k \leq \max \left\{ -\frac{a}{\gamma^*}, \frac{1 - 2a\gamma^* + \sqrt{(2a\gamma^* - 1)^2 - 4(\gamma^*)^2(a^2 - 1)}}{2(\gamma^*)^2} \right\} \tag{12}$$

□

(d) Run perceptron with starting points from the unit circle

We generate a random starting point from the boundary of the unit circle and run the perceptron 10000 times in this way.

```
# Generate a point from the boundary of a d-sphere
def sample_unit_sphere(d):
    x = np.random.normal(0.0,1.0,(d))
    norm_x = np.linalg.norm(x)
    point = x/norm_x
    return point
```

Plot of results of first 100 runs

The plot of the results for the first 100 runs is as shown in Figure 2.

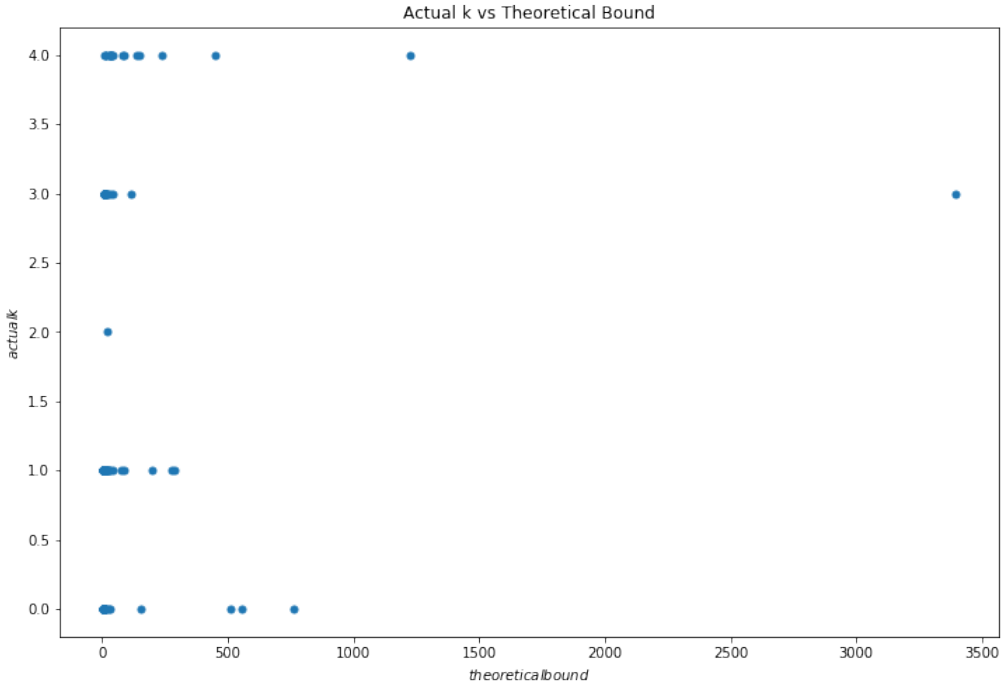

Figure 2: Plot of results of first 100 runs

The average number of updates required to successfully classify the dataset over the 10000 runs is 1.8044. The average of γ's is 0.24296256802801977. The γ^* obtained from part 2 is 0.4314845982168759, which is larger than the average of the γ's obtained in this part.

We also note that the theoretical bounds can be very large compared to the actual number of iterations needed. For example, there is a point where the theoretical bound is over 3000, but actual number of iterations needed is only 3.

1.4 Question 4: Non-linearly separable dataset

After changing the label of the first sample to -1 and label of third sample to +1, the new plot of the dataset is as shown in Figure 3.

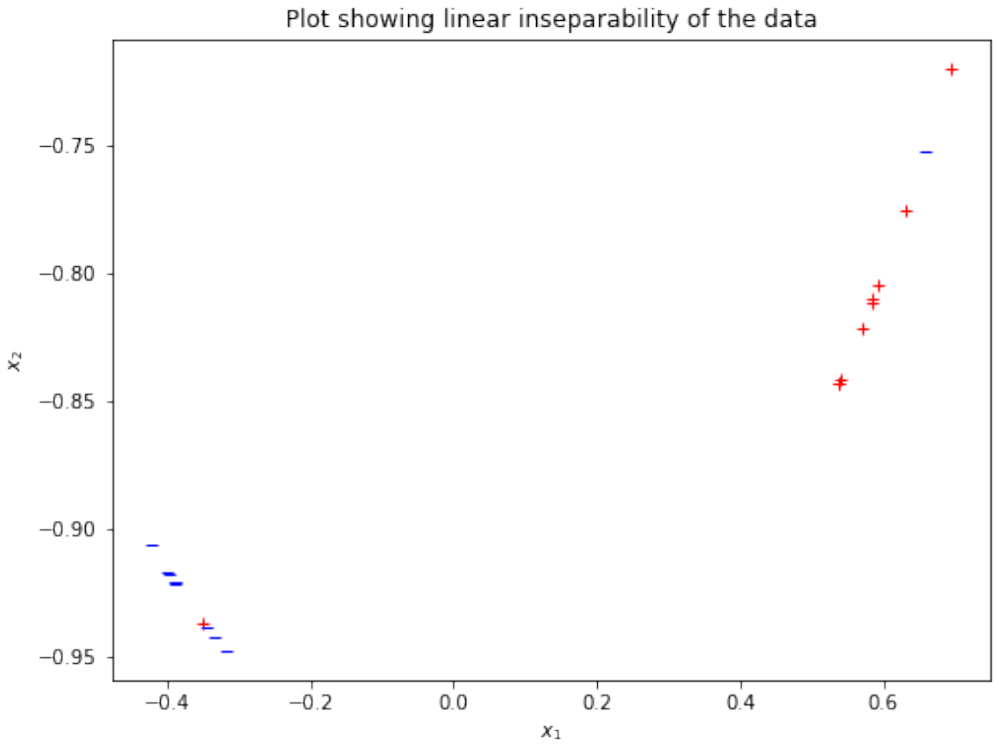

Figure 3: Plot of updated dataset

Clearly, the amended dataset is not linearly separable.
We run part 2 again.

```
# Run SVM
G = -np.multiply(np.matrix(y).T,X)
theta_star = quadprog_solve_qp(P, q, G=G, h=h)

# Run Perceptron
theta_zero = np.zeros(p)
k, theta, gamma = std_perceptron(X, y, theta_zero)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-17-881320d70e6d> in <module>()
      1 # Run SVM
      2 G = -np.multiply(np.matrix(y).T,X)
----> 3 theta_star = quadprog_solve_qp(P, q, G=G, h=h)
      4
      5 # Run Perceptron

<ipython-input-5-9928f0877a0e> in quadprog_solve_qp(P, q, G, h, A, b, initvals)
     14         qp_b = -h
     15         meq = 0
---> 16     return quadprog.solve_qp(qp_G, qp_a, qp_C, qp_b, meq)[0]

quadprog\quadprog.pyx in quadprog.solve_qp()

ValueError: constraints are inconsistent, no solution
```

Figure 4: Screenshot of SVM error

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-48-21ca5c3dff7a> in <module>()
      5 # Run Perceptron
      6 theta_zero = np.random.rand(2)
----> 7 k, theta, gamma = std_perceptron(X, y, theta_zero)

<ipython-input-45-bd18e77af62c> in std_perceptron(X, y, theta_zero)
     20         # raise exception if exceed 10000 iterations
     21         if k > 100000:
---> 22             raise Exception('exceeded 100000 iterations')
     23         isSolution = True
     24         for t in range(0,n):

Exception: exceeded 100000 iterations
```

Figure 5: Screenshot of standard perceptron error

For the SVM without offset and without slack, the algorithm terminates with *ValueError: constraints are inconsistent, no solution.* Hence, no feasible solution can be obtained using the SVM without offset and without slack.

For the standard perceptron algorithm using a random starting point from the uniform distribution [0,1), the algorithm does not converge after 100000 iterations (a sufficiently large number). Hence, we may conclude that the algorithm does not converge.

2 Problem 2: Hard SVM

2.1 Question 1: Plotting

The plot of the dataset is as shown in Figure 6.

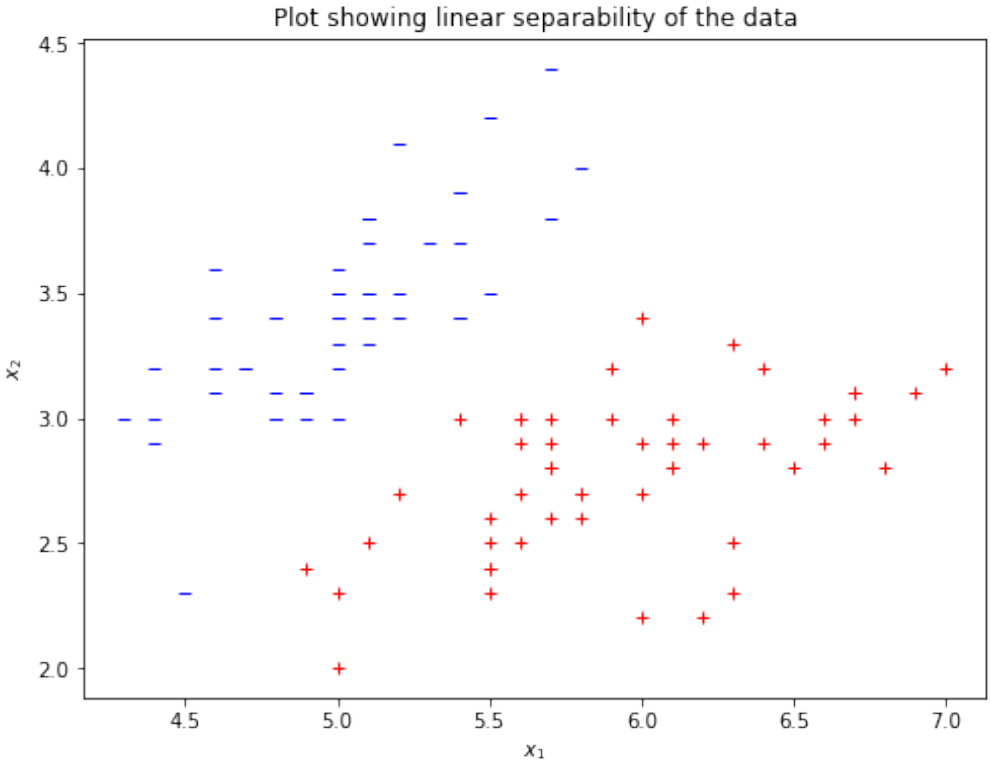

Figure 6: Plot of dataset iris1.csv

2.2 Question 2: Solve primal SVM with offset

A snippet of the code for this part is shown below.

```
# Set matrices for quadprog
P = np.identity(p+1)
# make the last row a zero row so that we don't include the offset in the objective function;
# however a matrix with zero row is not positive definite so use eps
for i in range(0,p+1):
    if i == p:
        P[i][i] = 3e-8
    else:
        P[i][i] = P[i][i] + 3e-8
q = np.zeros(p+1)
G = -1*np.concatenate((np.multiply(np.matrix(y).T,X),np.matrix(y).T),axis=1)
h = -1*np.ones(n)

theta_hardsvm = quadprog_solve_qp(P, q, G=G, h=h)

# Normalize theta
theta_star = theta_hardsvm[0:p]
theta_zero = theta_hardsvm[p]
obj_value = 0.5*math.pow(np.linalg.norm(theta_star),2)
```

Results

$\theta^* = [6.31578947, -5.26315789]^T$
$\theta_0^* = -17.31578947378911$
Optimal objective value = 33.79501385068385

2.3 Question 3: Plot solution

The plot of the line $\theta^T x + \theta_0 = 0$ is as shown in Figure 7. The support vectors are indicated with a black 'x' in the plot.

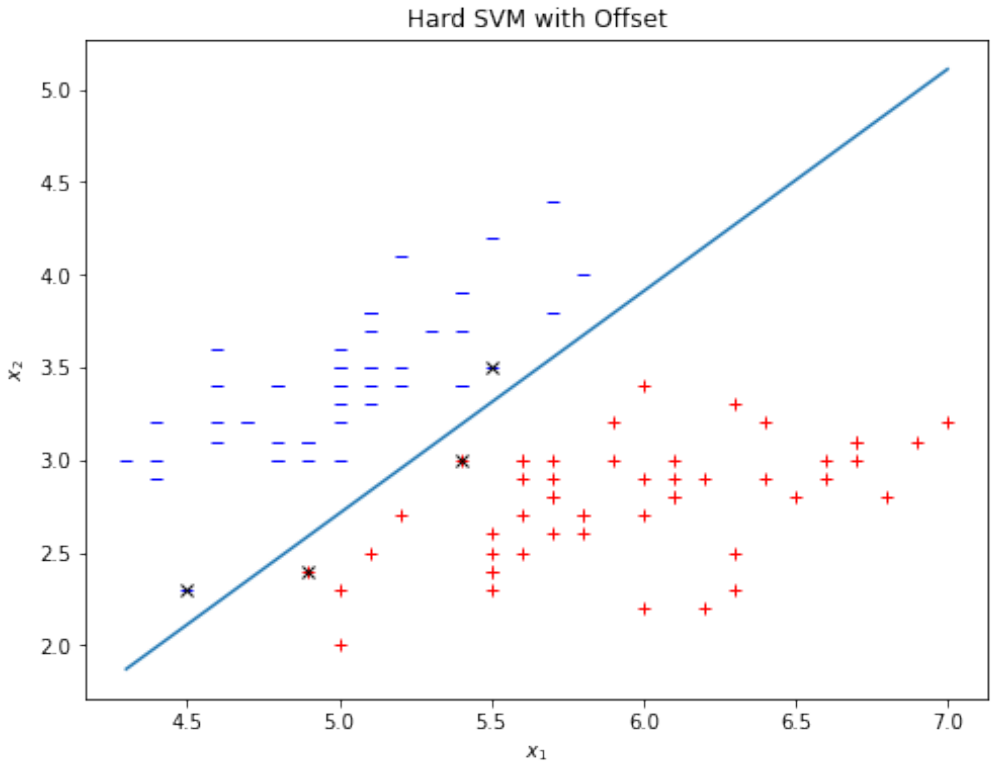

Figure 7: Plot of dataset with SVM solution

2.4 Question 4: Solve dual problem of SVM

We solve the dual problem using the quadprog function, with the matrices as defined below.

```
# Set up matrices
eps = 3e-8

Q = []
for i in range(0,n):
    temp = []
    for j in range(0,n):
        if i==j:
            temp.append(y[i]*y[j]*np.dot(X.values[i], X.values[j]) + eps)
        else:
            temp.append(y[i]*y[j]*np.dot(X.values[i], X.values[j]))
    Q.append(temp)

P = np.asarray(Q)
q = -np.ones(n)
G = -np.identity(n)
h = np.zeros(n)
A = np.matrix(y)
b = [0]

theta_dual = quadprog_solve_qp(P, q, G, h, A, b)
```

The non-zero (larger than 10^{-6}) entries of the optimal $\alpha = (\alpha_1, ..., \alpha_n)^T$ vector and their indices are the following:
$\alpha_{37} = 15.900268956553191$
$\alpha_{42} = 17.894727581879124$
$\alpha_{58} = 16.398883492860477$
$\alpha_{85} = 17.396112841146532$

Time taken to solve primal problem: 0.0002425967179391364
Time taken to solve dual problem: 0.004495503796405842

The optimal objective function value for the dual problem is 33.79501739019088

We observe that the dual optimal objective value is the same as the primal. The primal problem takes a shorter time to solve. As $n = 100 > 2 = d$, the dimension d of the data is significantly smaller than the size n of the dataset. The primal problem is in dimension d = 2, whereas the dual problem is in dimension n = 100. Hence, the primal problem requires a shorter time.

2.5 Question 5: Calculations

$\sum_{i=1}^{n} \alpha_i y_i x_i = [6.31578508, -5.26315588]^T$

By arbitrarily picking a $j \in \{1, 2, ..., n\}$ which satisfies $\alpha_j > 0$, we obtain
$y_j - \sum_{i=1}^{n} \alpha_i y_i x_i^T x_j = -17.315772347562643$

Note that $\sum_{i=1}^{n} \alpha_i y_i x_i = \hat{\theta}$ and $y_j - \sum_{i=1}^{n} \alpha_i y_i x_i^T x_j = \hat{\theta_0}$. This corresponds to the formulas for these two parameters in the lecture notes, using the KKT conditions for optimality.

3 Problem 3: Soft-SVM and Cross-validation

The code for the function *soft_svm* can be located in the Appendix.

3.1 Question 1: Solve soft-SVM

We solve the relaxed (primal) SVM problem by setting C=100.

```
# Run soft SVM
theta_star, theta_zero, obj_value, num_misclassified_egs = soft_svm(n,p,X.values,y,100)
```

Results
$\hat{\theta} = [-1.84782609, -3.26086957, 4.67391304, 10.86956522]^T$
$\hat{\theta_0} = -20.413043478222097$
Optimal objective function value = 654.1942344040862
Number of misclassified samples = 3

3.2 Question 2: 10-fold cross validation

For each $C \in \{1, 100, 10000\}$, run 10-fold cross validation and average the resultant 10 errors for each C. The code for this part can be located in the Appendix document.

The misclassified rate (averages of the 10 test errors) for each C is as follows:
Let average test error for each C be avg_C
C = 1, $avg_1 = 0.4$
C = 100, $avg_{100} = 0.6$
C = 10000, $avg_{10000} = 0.7$

Hence, the best choice of C,

$$C^* = \arg\min_C avg_C = 1$$