5.

From the problem desciption, we have the primal-dual sol:

$$(b^*, \tilde{w}^*, \alpha^*)$$

obtained by using $z_n^T = [1\ x_n^T]^T$ instead of x_n^T.

The $\tilde{w}^*$ is a $d+1$ dimensional vector:

$$\tilde{w}^* = [b^*, \tilde{w}_1^*, \cdots, \tilde{w}_d^*]$$

If we plug in this solution into primal problem P_1, we will get the minimized obejctive value:

$$\frac{1}{2}\tilde{\mathbf{w}}^{*T}\tilde{\mathbf{w}}^* + C\sum_{n=1}^{N}\xi_n$$

subject to:

$$y_n(\tilde{\mathbf{w}}^{*T}z_n) \geq 1 - \xi_n, \quad \forall n = 1, \cdots, N$$

We got the above inequality since we have:

$$\begin{aligned}
\tilde{\mathbf{w}}^{*T}z_n &= \begin{bmatrix} b^* & \tilde{w}_1^* & \cdots & \tilde{w}_d^* \end{bmatrix} \begin{bmatrix} 1 \\ x_{n1} \\ \vdots \\ x_{nd} \end{bmatrix} \\
&= \begin{bmatrix} \tilde{w}_1^* & \cdots & \tilde{w}_d^* \end{bmatrix} \begin{bmatrix} x_{n1} \\ \vdots \\ x_{nd} \end{bmatrix} + b^* \\
&= \mathbf{w}^{*T}\mathbf{x}_n + b^*
\end{aligned}$$

From the objective function of P1, we can see that:

$$\frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{N}\xi_n$$

is convex, because $\mathbf{w}^T\mathbf{w}$ is quadratic, and $C\sum_{n=1}^{N}\xi_n$ is linear (so also convex), and the sum of convex functions is convex.

And the first constraint:

$$y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 - \xi_n, \quad \forall n = 1, \cdots, N$$

Consider the 2 cases,

1. $y_n = 1$:

$$\begin{aligned} \mathbf{w}^T\mathbf{x}_n + b &\geq 1 - \xi_n \\ \mathbf{w}^T\mathbf{x}_n &\geq 1 - \xi_n - b \end{aligned}$$

2. $y_n = -1$:

$$\begin{aligned} -\mathbf{w}^T\mathbf{x}_n - b &\geq 1 - \xi_n \\ \mathbf{w}^T\mathbf{x}_n &\leq -1 + \xi_n - b \end{aligned}$$

The 2 results both define a halfspace, and halfspaces are convex.

For the second constraint:

$$\xi_n \geq 0$$

This is also a halfspace, so this constraint is also convex.

Therefore, the primal problem P_1 is convex.

Since if any convex optimization problem with differentiable objective and constraints satisfies Slater's condition, then the KKT conditions are sufficient and necessary for optimality.

For prove that Slater's condition holds, we need to find a point $(\mathbf{w}^0, b^0, \xi^0)$ such that if:

$$\begin{aligned} g_n(\mathbf{w}, b, \xi) &= -y_n(\mathbf{w}^T\mathbf{x}_n + b) + 1 - \xi_n \leq 0, \quad \forall n = 1, \cdots, N \\ h_n(\xi) &= -\xi_n \leq 0, \quad \forall n = 1, \cdots, N \end{aligned}$$

$(\mathbf{w}^0, b^0, \xi^0)$ satisfies:

$$\begin{aligned} g_n(\mathbf{w}^0, b^0, \xi^0) &< 0 \\ h_n(\xi^0) &< 0 \end{aligned}$$

Let $(\mathbf{w}^0, b^0, \xi^0) = (\mathbf{0}, 0, 1.5)$, then:

$$g_n(\mathbf{0}, 0, 1.5) = -y_n(0+0) + 1 - 1.5 = -0.5 < 0, \quad \forall n = 1, \cdots, N$$
$$h_n(1.5) = -1.5 < 0 \quad \forall n = 1, \cdots, N$$

Thus, Slater's condition holds.

Finally, to claim that $(b^*, \mathbf{w}^*, \alpha^*)$ is also a solution to P_1, we need to show that the KKT conditions are satisfied.

KKT conditions:

1. primal feasibility:

$$y_n(\mathbf{w}^{*T}\mathbf{x}_n + b^*) \geq 1 - \xi_n^*, \quad \forall n = 1, \cdots, N$$
$$\xi_n^* \geq 0, \quad \forall n = 1, \cdots, N$$

By our derivation above, we know that:

$$y_n(\mathbf{w}^{*T}\mathbf{x}_n + b^*) \geq 1 - \xi_n^*, \quad \forall n = 1, \cdots, N$$

since $\tilde{\mathbf{w}}^{*T} z_n = \mathbf{w}^{*T}\mathbf{x}_n + b^*$.

Also, $\xi_n^* \geq 0$ is satisfied because it is the same as what Dr.Threshold get in his optimal solution.

2. dual feasibility:

Again, since α^* is the same as what Dr.Threshold get in his optimal solution, the following still holds:

$$0 \leq \alpha_n^* \leq C, \quad \forall n = 1, \cdots, N$$

3. complementary slackness:

From Dr.Threshold's solution, we know that:

$$\alpha_n^*(1 - \xi_n^* - y_n(\tilde{\mathbf{w}}^{*T}\mathbf{z}_n)) = 0, \quad \forall n = 1, \cdots, N$$

and:

- if $\alpha_n^* > 0$, then $1 - \xi_n^* - y_n(\tilde{\mathbf{w}}^{*T}\mathbf{z}_n) = 0$
- if $\alpha_n^* = 0$, then $1 - \xi_n^* - y_n(\tilde{\mathbf{w}}^{*T}\mathbf{z}_n) < 0$.

So converting the above equation, we can rewrite it as:

$$\alpha_n^*(1 - \xi_n^* - y_n(\mathbf{w}^{*T}\mathbf{x}_n + b^*)) = 0, \quad \forall n = 1, \cdots, N$$

Similar results can be obtained:

- if $\alpha_n^* > 0$, then $1 - \xi_n^* - y_n(\mathbf{w}^{*T}\mathbf{x}_n + b^*) = 0$
- if $\alpha_n^* = 0$, then $1 - \xi_n^* - y_n(\mathbf{w}^{*T}\mathbf{x}_n + b^*) < 0$.

Thus, KKT holds and $(b^*, \mathbf{w}^*, \alpha^*)$ is also an optimal solution to the original problem.

6.

If we only consider the original constraint (i.e. the constraints for example 1 to N), the lagrange function with lagrange multipliers α_n and β_n is:

$$\mathcal{L}(b, \mathbf{w}, \xi, \alpha, \beta) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{N}\xi_n + \sum_{n=1}^{N}\alpha_n\left(1 - \xi_n - y_n(b + \mathbf{w}^T\mathbf{\Phi}(\mathbf{x_n}))\right) + \sum_{n=1}^{N}\beta_n(-\xi_n)$$

For the anchor pseudo-example, we have the constraint:

$$y_0(\mathbf{w}^T\mathbf{\Phi}(\mathbf{x_0}) + b) \geq 1$$

Since we have $\mathbf{x}_0 = \mathbf{0}$ and $y_0 = -1$, the constraint becomes:

$$-b \geq 1 \Rightarrow b \leq -1$$

Convert into canonical form:

$$b + 1 \leq 0$$

So we can add the term $\gamma_0(b + 1)$, where γ_0 is the corresponding lagrange multiplier.

Thus we have the new lagrange function:

$$\mathcal{L}(b, \mathbf{w}, \xi, \alpha, \beta, \gamma_0) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{N}\xi_n + \sum_{n=1}^{N}\alpha_n\left(1 - \xi_n - y_n(b + \mathbf{w}^T\mathbf{\Phi}(\mathbf{x_n}))\right) + \sum_{n=1}^{N}\beta_n(-\xi_n) + \gamma_0(b+1)$$

For the lagrange dual, we need to solve:

$$\max_{\alpha,\beta,\gamma_0 \geq 0} \min_{b,\mathbf{w},\xi} \mathcal{L}(b, \mathbf{w}, \xi, \alpha, \beta, \gamma_0)$$

Taking the partial derivative of each variable, we get:

$$\frac{\partial \mathcal{L}}{\partial b} = 0 \Rightarrow -\sum_{n=1}^{N}\alpha_n y_n + \gamma_0 = 0 \tag{1}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^{N}\alpha_n y_n \mathbf{\Phi}(\mathbf{x_n}) \tag{2}$$

$$\frac{\partial \mathcal{L}}{\partial \xi_n} = 0 \Rightarrow C - \alpha_n - \beta_n = 0 \tag{3}$$

Since $\alpha_n \geq 0$ and $\beta_n \geq 0$, we have $0 \leq \alpha_n \leq C$.

From equation (2), we can substitute $\mathbf{w}$ into the Lagrangian and obtain the dual:

$$\frac{1}{2}\left\| \sum_{n=1}^{N} \alpha_n y_n \Phi(\mathbf{x}_n) \right\|^2 + C\sum_{n=1}^{N} \xi_n + \sum_{n=1}^{N} \alpha_n \left(1 - \xi_n - y_n(b + \mathbf{w}^T \mathbf{\Phi}(\mathbf{x_n}))\right) + \sum_{n=1}^{N} \beta_n(-\xi_n) + \gamma_0(b+1)$$

Converting $||\mathbf{w}||^2$ to the proper form, can write:

$$||\mathbf{w}||^2 = \sum_{n=1}^{N}\sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m \mathbf{\Phi}(\mathbf{x_n})^T \mathbf{\Phi}(\mathbf{x_m})$$

Plug in back to the dual, we get:

$$\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m K(x_n, x_m) + C\sum_{n=1}^{N} \xi_n + \sum_{n=1}^{N} \alpha_n \left(1 - \xi_n - y_n(b + \mathbf{w}^T \mathbf{\Phi}(\mathbf{x_n}))\right) + \sum_{n=1}^{N} \beta_n(-\xi_n) + \gamma_0(b+1)$$

Expand the terms and we get:

$$\begin{aligned}
&\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m K(x_n, x_m) + C\sum_{n=1}^{N} \xi_n \\
&+ \sum_{n=1}^{N} \alpha_n - \sum_{n=1}^{N} \alpha_n \xi_n - \sum_{n=1}^{N} \alpha_n y_n b - \sum_{n=1}^{N} \alpha_n y_n \mathbf{w}^T \mathbf{\Phi}(\mathbf{x_n}) \\
&- \sum_{n=1}^{N} \beta_n \xi_n + \gamma_0 b + \gamma_0
\end{aligned}$$

From equation (1), we have:

$$-\sum_{n=1}^{N} \alpha_n y_n + \gamma_0 = 0 \Rightarrow (-\sum_{n=1}^{N} \alpha_n y_n + \gamma_0) b = 0$$

And from equation (3), we have:

$$(C - \alpha_n - \beta_n) \sum_{n=1}^{N} \xi_n = 0$$

The Largrangian is simplified to:

$$\frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m K(x_n, x_m) + \sum_{n=1}^{N} \alpha_n - \sum_{n=1}^{N} \alpha_n y_n \mathbf{w}^T \mathbf{\Phi}(\mathbf{x_n}) + \gamma_0 \quad (4)$$

Observe that the term that contain $\mathbf{w}$ can also be expanded by (2) as:

$$\begin{aligned} -\sum_{n=1}^{N} \alpha_n y_n \mathbf{w}^T \mathbf{\Phi}(\mathbf{x_n}) &= -\sum_{n=1}^{N} \alpha_n y_n \left(\sum_{m=1}^{N} \alpha_m y_m \mathbf{\Phi}(\mathbf{x_m}) \right)^T \mathbf{\Phi}(\mathbf{x_n}) \\ &= -\sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m \mathbf{\Phi}(\mathbf{x_m})^T \mathbf{\Phi}(\mathbf{x_n}) \\ &= -\sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m K(x_n, x_m) \end{aligned}$$

Again, plug into the previous simplified Lagrangian (4), we get:

$$-\frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m K(x_n, x_m) + \sum_{n=1}^{N} \alpha_n + \gamma_0$$

From the problem description, we knew that $y_n = +1 \quad \forall n,\ n \neq 0$, so we have:

$$-\frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m K(x_n, x_m) + \sum_{n=1}^{N} \alpha_n + \gamma_0$$

In order to use the QP solver, we first need to convert the above maximization problem into a minimization problem:

$$\min_{\alpha, \gamma_0 \geq 0} \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m K(x_n, x_m) - \sum_{n=1}^{N} \alpha_n - \gamma_0 \quad (*)$$

With constraints from $(1), (3)$:

$$\sum_{n=1}^{N} \alpha_n - \gamma_0 = 0 \quad (5)$$

$$C - \alpha_n - \beta_n = 0 \tag{6}$$

To convert into the QP form, we define:

$$\begin{aligned}
\alpha &= [\alpha_1 \ \alpha_2 \ \cdots \ \alpha_N]^T \\
\mathbf{u} &= [\gamma_0 \ \alpha_1 \ \alpha_2 \ \cdots \ \alpha_N]^T \\
\mathbf{Q} &= [K(x_n, x_m)]_{N \times N} \quad \text{(Gram matrix)}
\end{aligned}$$

Then the original problem $(*)$ can be written as:

$$\min_{\mathbf{x} \geq 0} \frac{1}{2} \mathbf{u}^T \mathbf{Q} \mathbf{u} - \mathbf{1}^T \mathbf{u}$$

So:

$$Q = [K(x_n, x_m)]_{N \times N} \quad \text{and} \quad \mathbf{p} = -\mathbf{1}$$

subject to:

$$\begin{aligned}
\sum_{n=1}^{N} \alpha_n &= \gamma_0 \qquad \text{by (5)} \\
\alpha_n &= C - \beta_n \qquad \text{by (6)}
\end{aligned}$$

Thus for each row in A,

$$\mathbf{a}_n^T = y_n [1 \ \mathbf{\Phi}(\mathbf{x_n})^\mathbf{T}] = \begin{cases} [1 \ \mathbf{\Phi}(\mathbf{x_n})^\mathbf{T}] & \forall n \neq 0 \\ [1 \ \mathbf{0^T}] & n = 0 \end{cases}$$

Finally, we have each element in $\mathbf{c}$ as $c_n = 0$

7.

Plugging in $\alpha = \mathbf{1}$ and $b = 0$ into $h_{\alpha,b}(\mathbf{x})$, we get:

$$\hat{h}(\mathbf{x}) = \text{sign}\left(\sum_{n=1}^{N} y_n K(\mathbf{x}_n, \mathbf{x})\right)$$

Consider arbitrary $\mathbf{x}_n$ and $\mathbf{x}_m$, the result of the Gaussian kernel is:

$$K(\mathbf{x}_n, \mathbf{x}_m) = \begin{cases} \exp(-\gamma\|\mathbf{x}_n - \mathbf{x}_m\|^2) & \text{if } n \neq m \\ 1 & \text{if } n = m \end{cases}$$

Since the problem assumed that $||\mathbf{x}_n - \mathbf{x}_m|| \geq \epsilon \quad \forall n \neq m$, for the case $n \neq m$, we have:

$$K(\mathbf{x}_n, \mathbf{x}_m) = \exp(-\gamma\|\mathbf{x}_n - \mathbf{x}_m\|^2) \leq \exp(-\gamma\epsilon^2)$$

Consider the condition that $\gamma > \frac{\ln(N-1)}{\epsilon^2}$, we have:

$$\exp(-\gamma\|\mathbf{x}_n - \mathbf{x}_m\|^2) \leq \exp(-\frac{\ln(N-1)}{\epsilon^2}\epsilon^2) = \exp(-\ln(N-1)) = \frac{1}{N-1}$$

Thus, $\exp(-\gamma\|\mathbf{x}_n - \mathbf{x}_m\|^2) \to 0$ when γ sufficiently large.

Therefore, we can observe that when γ is large enough, for the prediction of an arbitrary $\mathbf{x}_m$, we have:

$$\begin{aligned} \hat{h}(\mathbf{x}_m) &= \text{sign}\left(\sum_{n=1}^{N} y_n K(\mathbf{x}_n, \mathbf{x}_m)\right) \\ &= \text{sign}\left(\sum_{n \in \{1\ldots N\} \setminus m} y_n \exp(-\gamma\|\mathbf{x}_n - \mathbf{x}_m\|^2) + y_m \times 1\right) \\ &\approx \text{sign}\,(y_m) \end{aligned}$$

This means that the prediction made by $\hat{h}$ is the same as the label of $\mathbf{x}_m$, which results in $E_{in}(\hat{h}) = 0$.

8.

We need to show that $K(x, x') = \exp(2\cos(x - x') - 2)$ is a valid kernel.

By the definition of a valid kernel, it should be a symmetric and positive semidefinite function, so it is equivalent to show that $\mathbf{K}$ is a valid (i.e. symmetric and positive semidefinite) Gram matrix, where $\mathbf{K}$ is defined as follows:

$$\mathbf{K} = \begin{bmatrix} K(x_1, x_1) & K(x_1, x_2) & \cdots & K(x_1, x_N) \\ K(x_2, x_1) & K(x_2, x_2) & \cdots & K(x_2, x_N) \\ \vdots & \vdots & \ddots & \vdots \\ K(x_N, x_1) & K(x_N, x_2) & \cdots & K(x_N, x_N) \end{bmatrix}$$

To prove the symmetric property, we need to show that for arbitrary $x_i, x_j, \quad K(x_i, x_j) = K(x_j, x_i)$.

Plug in to the definition, and using the property $\cos(x) = \cos(-x)$, we have:

$$K(x_i, x_j) = \exp(2\cos(x_i - x_j) - 2) = \exp(2\cos(x_j - x_i) - 2) = K(x_j, x_i)$$

Thus, $K(x_i, x_j) = K(x_j, x_i)$ holds.

To prove the positive semidefinite property, we need to show that for arbitrary $\mathbf{z} \in \mathbb{R}^N$, $\mathbf{z}^\top \mathbf{K} \mathbf{z} \geq 0$.

Let $\mathbf{z} = \begin{bmatrix} z_1 & z_2 & \cdots & z_N \end{bmatrix}^\top$, then we have:

$$\mathbf{z}^\top \mathbf{K} \mathbf{z} = \sum_{i=1}^{N} \sum_{j=1}^{N} z_i z_j K(x_i, x_j)$$

Substitute the definition of $K(x_i, x_j)$, we have:

$$\begin{aligned} \mathbf{z}^\top \mathbf{K} \mathbf{z} &= \sum_{i=1}^{N} \sum_{j=1}^{N} z_i z_j \exp(2\cos(x_i - x_j) - 2) \\ &= e^{-2} \sum_{i=1}^{N} \sum_{j=1}^{N} z_i z_j \exp(2\cos(x_i - x_j)) \end{aligned}$$

Since $e^{-2} > 0$, we only need to show that:

$$\sum_{i=1}^{N} \sum_{j=1}^{N} z_i z_j \exp(2\cos(x_i - x_j)) \geq 0$$

This is equivalent to:

$$\sum_{i=1}^{N}\sum_{j=1}^{N} z_i z_j \exp(2(\cos x_i \cos x_j + \sin x_i \sin x_j))$$

If we define the vectors:

$$\mathbf{v}_i = \begin{bmatrix}\cos x_i \\ \sin x_i\end{bmatrix}, \quad \mathbf{v}_j = \begin{bmatrix}\cos x_j \\ \sin x_j\end{bmatrix}$$

Then the above equation can be written as:

$$\sum_{i=1}^{N}\sum_{j=1}^{N} z_i z_j \exp(2\mathbf{v}_i^\top \mathbf{v}_j)$$

We further define the matrix $\mathbf{V}$ with:

$$\mathbf{V}_{ij} = \exp(2\mathbf{v}_i^\top \mathbf{v}_j)$$

We can easily see that $\mathbf{V}$ is a symmetric matrix, and since $\exp(2\mathbf{v}_i^\top \mathbf{v}_j)$ is a positive definite function of the inner product $\mathbf{v}_i^\top \mathbf{v}_j$, $\mathbf{V}$ is a positive (semi)definite matrix.

And we can rewrite the above equation:

$$\mathbf{z}^\top \mathbf{K}\mathbf{z} = e^{-2}\mathbf{z}^\top \mathbf{V}\mathbf{z}$$

Since $\mathbf{V}$ is a positive semidefinite matrix, $e^{-2}\mathbf{z}^\top \mathbf{V}\mathbf{z} \geq 0$ holds for arbitrary $\mathbf{z} \in \mathbb{R}^N$.

Thus, $\mathbf{z}^\top \mathbf{K}\mathbf{z} \geq 0$ holds for arbitrary $\mathbf{z} \in \mathbb{R}^N$, and $\mathbf{K}$ is also a positive semidefinite matrix.

Therefore, we have shown that $K(x, x')$ is a valid kernel.

9.

From the definition of scaled decision stumps $g_{i,\theta}(x) = [\![x_i \geq \theta]\!]$, we obtain:

$$\begin{aligned}\boldsymbol{\Phi}_{ds}(\mathbf{x}) &= [g_{1,\theta_1}(\mathbf{x}) \cdots g_{d,\theta_k}(\mathbf{x})]^T \\ &= [[\![x_1 \geq \theta_1]\!] \cdots [\![x_d \geq \theta_k]\!]]^T\end{aligned}$$

where d is the dimension of $\mathbf{x}$.

Also, we have $\theta \in \{\theta_1 = L + 0.5, \theta_2 = L + 1.5, \cdots, \theta_k = R - 0.5\}$, so:

$$\boldsymbol{\Phi}_{ds}(\mathbf{x}) = \begin{bmatrix} [\![x_1 \geq L + 0.5]\!] \\ [\![x_2 \geq L + 1.5]\!] \\ \vdots \\ [\![x_{d-1} \geq R - 1.5]\!] \\ [\![x_d \geq R - 0.5]\!] \end{bmatrix}$$

Substituting this result into the definition of $K_{ds}(\mathbf{x}, \mathbf{x}')$, we obtain:

$$\begin{aligned}K_{ds}(\mathbf{x}, \mathbf{x}') &= \boldsymbol{\Phi}_{ds}(\mathbf{x})^T \boldsymbol{\Phi}_{ds}(\mathbf{x}') \\ &= \begin{bmatrix} [\![x_1 \geq L + 0.5]\!] & [\![x_2 \geq L + 1.5]\!] & \dots & [\![x_{d-1} \geq R - 1.5]\!] & [\![x_d \geq R - 0.5]\!] \end{bmatrix} \\ &\quad \times \begin{bmatrix} [\![x'_1 \geq L + 0.5]\!] \\ [\![x'_2 \geq L + 1.5]\!] \\ \vdots \\ [\![x'_{d-1} \geq R - 1.5]\!] \\ [\![x'_d \geq R - 0.5]\!] \end{bmatrix} \\ &= \sum_{i=1}^{d} [\![x_i \geq \theta_i]\!][\![x'_i \geq \theta_i]\!]\end{aligned}$$

Therefore, we got the definition of $K_{ds}(\mathbf{x}, \mathbf{x}')$, and the meaning is to count how many dimensions i of the 2 vectors satisfy the condition that both:

$$x_i \geq \theta_i \quad \text{and} \quad x'_i \geq \theta_i$$

10.

The result of the experiment is shown in the following figure:

```
Complete Results Table
------------------------------------------------
Support Vectors for each combination:
      Q=2  Q=3  Q=4
C
0.1   505  547  575
1.0   505  547  575
10.0  505  547  575

Best Combination(s):
------------------------------------------------
   C  Q  Support Vectors
 0.1  2              505
 1.0  2              505
10.0  2              505

Minimum number of support vectors: 505
```

Figure 1: result table

From the figure, we can see that no matter how the value C is set, the number of support vectors in always the same, and only differs when the value of Q is changed.

If we recall what the parameter C represents, it is the regularization parameter, which trades off the size of the margin and the total amount of the slack.

This means that if we have a larger valuer of C, then we're giving the slack variables larger weights, meaning that our model will press more emphasis on minimizing the distance from the data points to its corresponding positve / negative margin hyperplane.

But as the result shows, the value of C does not affect the number of support vectors, so this might due to the other parameter Q.

If we look at the parameter Q, it is the polynomial degree, so as Q becomes larger, we project the data into a higher dimensional space, which makes the

model more complex (having more intricate boundaries), thus intuitvely, the number of support vectors becomes larger.

To conclude, I think that the polynomial of degree 2 is already enough, so as we increase the value of Q, the classifier will just overfit, and this is also the reason why the value of C does not affect the result, since if we can draw the boundary with a polynomial of degree 2 with the least amount of support vectors, then it means that most of the data points will locate far from the decision boundary. (The degree-2 polynomial kernel already achieves a large margin with minimal misclassifications.)

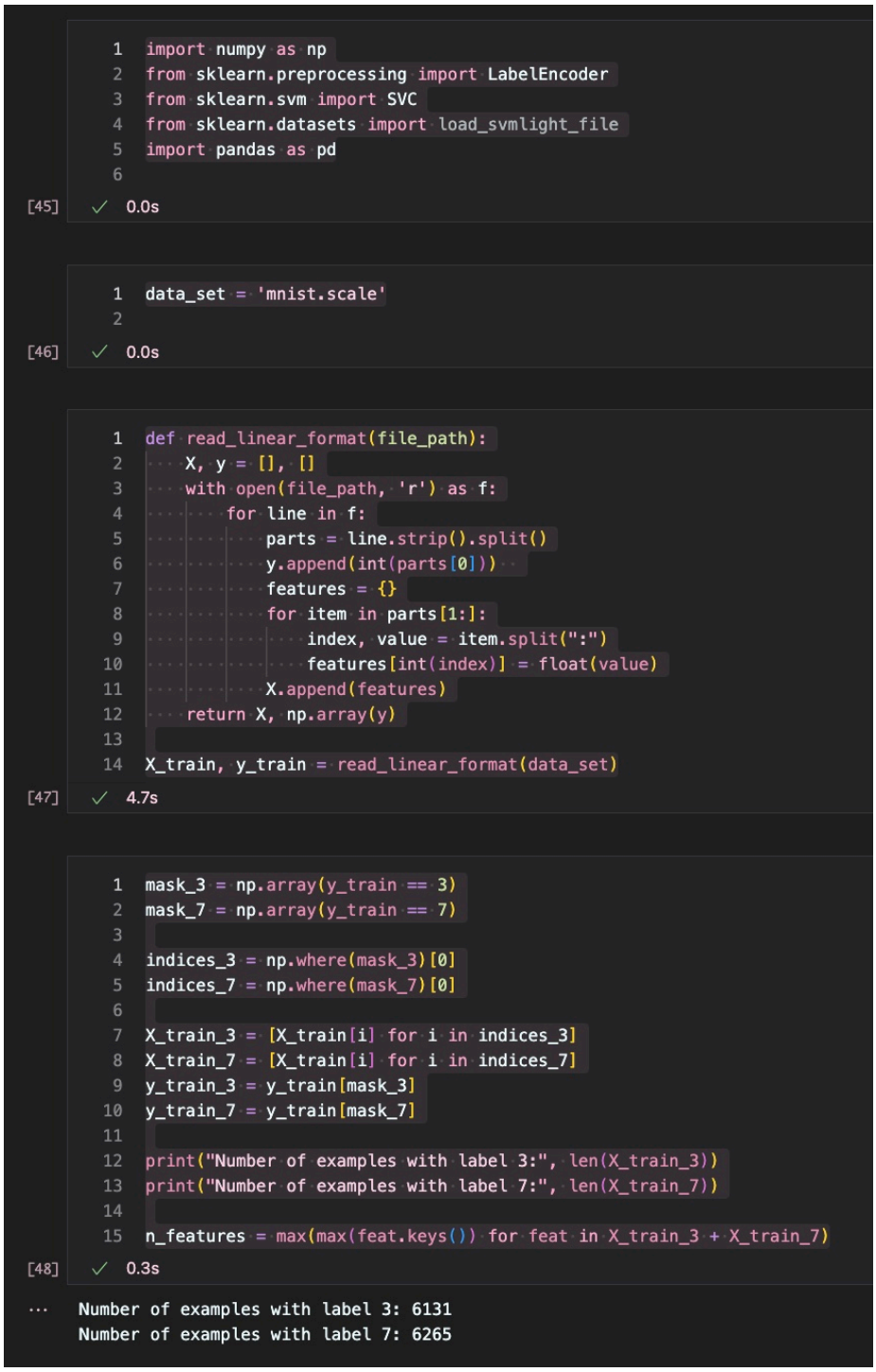

Figure 2: code snapshot 1

```python
def dict_to_array(X_dict, n_features):
    X_dense = np.zeros((len(X_dict), n_features))
    for i, sample in enumerate(X_dict):
        for feat_idx, value in sample.items():
            X_dense[i, feat_idx-1] = value
    return X_dense

X_train_3_dense = dict_to_array(X_train_3, n_features)
X_train_7_dense = dict_to_array(X_train_7, n_features)

X_combined = np.vstack([X_train_3_dense, X_train_7_dense])
```

✓ 0.1s

```python
# Create a LabelEncoder and specify the required mapping
le = LabelEncoder()

# Specify the original labels
le.fit([3, 7])

# Combine the filtered data
y_combined = np.concatenate([y_train_3, y_train_7])
# the mapping is: 3 -> -1, 7 -> 1
y_train_encoded = np.where(y_combined == 3, -1, 1)

# transform the original labels to -1 and 1
y_train_3_encoded = np.full(len(y_train_3), -1)  # All 3s become -1
y_train_7_encoded = np.full(len(y_train_7), 1)  # All 7s become 1

# Verify the results
print("Unique labels after encoding:", np.unique(y_train_encoded))
print("Number of -1 labels:", np.sum(y_train_encoded == -1))
print("Number of 1 labels:", np.sum(y_train_encoded == 1))
```

✓ 0.0s

```
Unique labels after encoding: [-1  1]
Number of -1 labels: 6131
Number of 1 labels: 6265
```

```python
# Use this list to store the result of form (C, Q, amount_of_support_vectors)
result = []

for C in [0.1, 1, 10]:
    for Q in [2, 3, 4]:
        svm_classifier = SVC(C = C, kernel = 'poly', degree = Q, coef0 = 1, gamma = 1)
        svm_classifier.fit(X_combined, y_train_encoded)

        amount_of_support_vectors = svm_classifier.n_support_.sum()
        result.append((C, Q, amount_of_support_vectors))
```

✓ 10.5s

Figure 3: code snapshot 2

11.

The result of the experiment is shown in the following figure:

```
Results Table
------------------------------------------------
Margin for each combination:
      gamma=0.1  gamma=1.0  gamma=10.0
C
0.1    0.041274   0.090780    0.090793
1.0    0.018166   0.009078    0.009079
10.0   0.017794   0.008984    0.008982
```

Figure 1: result table

optimal parameter values

From the figure, we obtain that the largest margin is achieved when the value of γ is 10, and the value of C is 0.1.

perspective from C

As explained in the previous question, the value of C represents a trade-off between the margin and the training error.

For example, when C is large, to minimize the primal problem, we need to prioritize fitting the training data than minimizing the length of the weight vector, so that $||\mathbf{w}||$ may be larger, and the margin is smaller.

We can see this in our resulting table, for each given γ, when C gets larger, the margin is smaller.

perspective from γ

In this problem, we're asked to use the rbf kernel, which is defined as:

$$K(x, x') = \exp(-\gamma||x - x'||^2)$$

The meaning of this kernel is that it measures the similarity between two points x and x' by considering the distance between them, and γ controls the standard for how we consider two points to be similar.

For example, when γ is large, the exponential function decreases rapidly as the distance between x, x' becomes larger. This means that two points need to be very close to each other to be considered similar.

Why this will affect the decision boundary is because for a large γ, $\mathbf{K}(x, x')$ is very small even if x and x' are sligtly distant to each other, so the influence of a single training point will be limited to a small region around it. Therefore, the decision boundary will be influenced by only a few points around it, so the boundary is more intricate.

On the other hand, when γ is small, even for more distant points, they might also affect the decision boundary, so the boundary is smoother.

To conclude, a large γ increases the complexity of the model, and results in a smaller margin, and this conclusion also matches our result.

snapshot of code

The snapshots of my code are shown below: > The beginning of the code is the same as the previous question, so I will only show the part after prepocessing (after relabeling).

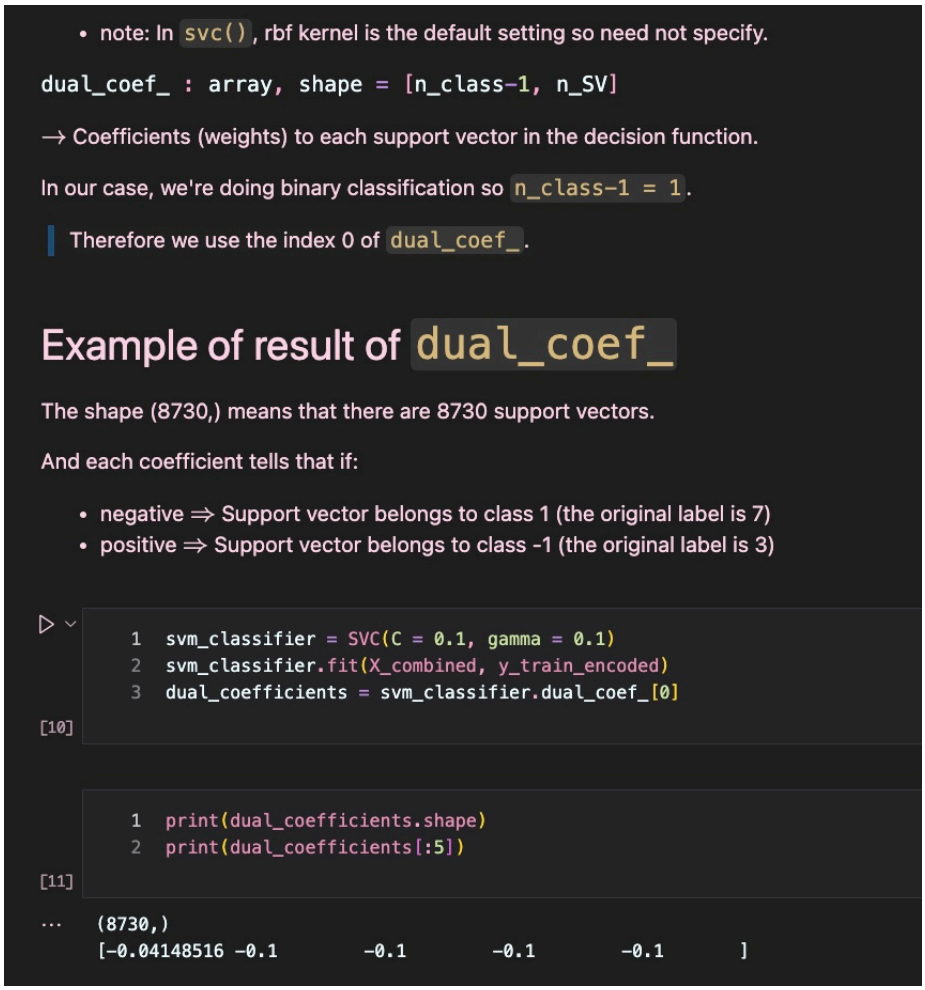

Figure 2: code snapshot 1

The weight vector is defined as:

> see Lecture 4 slide 10

$$\mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{z}_n$$

And

$$\begin{aligned} ||\mathbf{w}||^2 &= \mathbf{w}^T \mathbf{w} \\ &= \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m K(\mathbf{x}_n, \mathbf{x}_m) \\ &= \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m \exp(-\gamma ||\mathbf{x}_n - \mathbf{x}_m||^2) \end{aligned}$$

```python
# Use this list to store the result of form (C, gamma, margin)
result = []

for C in [0.1, 1, 10]:
    for gamma in [0.1, 1, 10]:
        svm_classifier = SVC(C = C, gamma = gamma)
        svm_classifier.fit(X_combined, y_train_encoded)

        support_vectors = svm_classifier.support_vectors_
        dual_coefficients = svm_classifier.dual_coef_[0]
        support_labels = y_train_encoded[svm_classifier.support_]

        w_norm_squared = 0
        num_support_vectors = len(support_vectors)

        for i in range(num_support_vectors):
            for j in range(num_support_vectors):
                w_norm_squared += (
                    dual_coefficients[i] * dual_coefficients[j]
                    * support_labels[i] * support_labels[j]
                    * np.exp(-gamma * np.linalg.norm(support_vectors[i] - support_vectors[j])**2)
                )

        margin = 1.0 / np.sqrt(w_norm_squared)
        result.append((C, gamma, margin))
```

[17]

Figure 3: code snapshot 2

12.

The resulting bar chart of the number of times γ is selected is as follows:

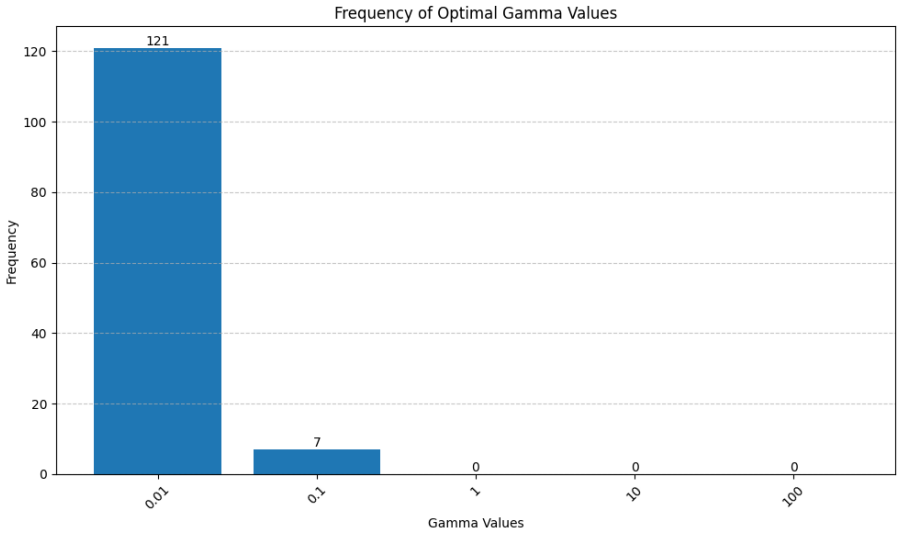

Figure 1: bar chart of γ

From this bar chart we can see that almost all of the optimal γ values are 0.01.

As stated in the previous question 11, the value of γ represents the standard, or the scale for how we consider two points to be similar.

smallest γ being optimal

The smallest γ is optimal in most of the cases implies that the rbf kernel decays more slowly as the distance between two points grows bigger. This would make the decision boundary to be influenced by more points that are farther from it, thus making the decision boundary more smooth (indicating higher generalization ability).

As we take the validation error into consideration, this value evaluates how well the model generalizes, thus the smaller γ is more likely to be optimal.

a sligtly more complex model being optimal

But we still see that there's also some cases with $\gamma = 0.1$ being optimal, indicating that sometimes adding a little bit complexity to the model might be better.

This may due to the selection of our validation set, since we only select 200 points from the training set, this amount of examples might not be enough to represent the whole training set.

Furthermore, if the points in the validation set are near the decision boundary, the model with higher complexity might be able to better fit these points, even though it might not generalize well.

why not the larger γ values?

For $\gamma = 1, 10, 100$, these values are never selected to be optimal.

Again from the meaning of γ, a larger γ makes the decision boundary highly sensitive to the points that are really close to it, thus generating a intricate boundary.

As a result, complex boundaries cannot generalize well on the validation set, making the validation error high.

code snapshot

q12.py > ...

```python
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC
from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from tqdm import tqdm
from multiprocessing import Pool

data_set = 'mnist.scale'

def read_linear_format(file_path):
    X, y = [], []
    with open(file_path, 'r') as f:
        for line in f:
            parts = line.strip().split()
            y.append(int(parts[0]))
            features = {}
            for item in parts[1:]:
                index, value = item.split(":")
                features[int(index)] = float(value)
            X.append(features)
    return X, np.array(y)

X_train, y_train = read_linear_format(data_set)

mask_3 = np.array(y_train == 3)
mask_7 = np.array(y_train == 7)

indices_3 = np.where(mask_3)[0]
indices_7 = np.where(mask_7)[0]

X_train_3 = [X_train[i] for i in indices_3]
X_train_7 = [X_train[i] for i in indices_7]
y_train_3 = y_train[mask_3]
y_train_7 = y_train[mask_7]

n_features = max(max(feat.keys()) for feat in X_train_3 + X_train_7)

def dict_to_array(X_dict, n_features):
    X_dense = np.zeros((len(X_dict), n_features))
    for i, sample in enumerate(X_dict):
        for feat_idx, value in sample.items():
            X_dense[i, feat_idx-1] = value
    return X_dense

X_train_3_dense = dict_to_array(X_train_3, n_features)
X_train_7_dense = dict_to_array(X_train_7, n_features)

X_combined = np.vstack([X_train_3_dense, X_train_7_dense])

le = LabelEncoder()

le.fit([3, 7])

y_combined = np.concatenate([y_train_3, y_train_7])
# the mapping is: 3 -> -1, 7 -> 1
y_train_encoded = np.where(y_combined == 3, -1, 1)

y_train_3_encoded = np.full(len(y_train_3), -1)  # All 3s become -1
y_train_7_encoded = np.full(len(y_train_7), 1)   # All 7s become 1

```

Figure 2: code snapshot 1

```python
def worker(procedure):
    X_train, X_validation, y_train, y_validation = train_test_split(X_combined, y_train_encoded, test_size=200)

    opt_gamma = None
    opt_validation_err = np.inf
    for gamma in [0.01, 0.1, 1, 10, 100]:
        svm_classifier = SVC(C = 1, gamma = gamma)
        svm_classifier.fit(X_train, y_train)
        y_validation_pred = svm_classifier.predict(X_validation)
        validation_err = np.mean(y_validation_pred != y_validation)

        # if tie on E_val, choose the smallest gamma
        if validation_err < opt_validation_err or (validation_err == opt_validation_err and gamma < opt_gamma):
            opt_validation_err = validation_err
            opt_gamma = gamma

    return opt_gamma

def main():
    # Make variables global so they can be accessed by worker processes
    global X_combined, y_train_encoded

    gamma_counts = {0.01: 0, 0.1: 0, 1: 0, 10: 0, 100: 0}

    with Pool(processes=4) as pool:
        optimal_gammas = list(tqdm(pool.imap(worker, range(128)), total=128))

    for gamma in optimal_gammas:
        gamma_counts[gamma] += 1

    # Print and plot results
    print("\nGamma value counts:")
    for gamma, count in gamma_counts.items():
        print(f"gamma = {gamma}: {count} times")

    plt.figure(figsize=(10, 6))
    plt.bar([str(gamma) for gamma in gamma_counts.keys()], gamma_counts.values())
    plt.title('Frequency of Optimal Gamma Values')
    plt.xlabel('Gamma Values')
    plt.ylabel('Frequency')
    plt.xticks(rotation=45)
    plt.grid(axis='y', linestyle='--', alpha=0.7)

    for i, v in enumerate(gamma_counts.values()):
        plt.text(i, v, str(v), ha='center', va='bottom')

    plt.tight_layout()
    plt.show()

if __name__ == '__main__':
    main()
```

Figure 3: code snapshot 2

13.

The soft margin SVM dual is defined as:

$$\begin{aligned} \min_{\alpha} \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m \mathbf{z}_n^T \mathbf{z}_m - \sum_{n=1}^{N} \alpha_n \\ \text{subject to} \sum_{n=1}^{N} y_n \alpha_n = 0 \\ 0 \leq \alpha_n \leq C, \quad n = 1, \cdots, N \end{aligned}$$

We're asked to derive a dual of the above problem.

So first we should rewrite the objective function and the constraints into the canonical form:

Let the objective function be $f_0(\alpha)$:

$$f_0(\alpha) = \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m \mathbf{z}_n^T \mathbf{z}_m - \sum_{n=1}^{N} \alpha_n$$

and the constraints be $f_1(\alpha)$ for the equality constraint and $h_j(\alpha)$ for the inequality constraint:

$$f_1(\alpha) = \sum_{n=1}^{N} y_n \alpha_n$$

For $0 \leq \alpha_n \leq C$, we can split it into two constraints:

$$\begin{aligned} 0 \leq \alpha_n \quad n = 1, \cdots, N \\ \Rightarrow \ -\alpha_n \leq 0 \quad n = 1, \cdots, N \end{aligned}$$

and

$$\alpha_n - C \leq 0 \quad n = 1, \cdots, N$$

So let:

1. $g_n(\alpha_n) = -\alpha_n, \quad n = 1, \cdots, N$
2. $h_n(\alpha_n) = \alpha_n - C, \quad n = 1, \cdots, N$

Therefore, we can construct the Lagrangian:

$$\begin{aligned} L(\alpha, \lambda, \mu, \gamma) &= f_0(\alpha) + \lambda f_1(\alpha) + \sum_{n=1}^{N} \mu_n g_n(\alpha) + \sum_{n=1}^{N} \gamma_n h_n(\alpha) \\ &= \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m \mathbf{z}_n^T \mathbf{z}_m - \sum_{n=1}^{N} \alpha_n + \lambda \sum_{n=1}^{N} y_n \alpha_n + \sum_{n=1}^{N} \mu_n(-\alpha_n) + \sum_{n=1}^{N} \gamma_n(\alpha_n - C) \end{aligned}$$

We need to compare this $L(\alpha, \lambda, \mu, \gamma)$ with the soft-margin SVM primal, which is:

$$\begin{aligned} &\min_{\alpha} \frac{1}{2} \mathbf{w}^T \mathbf{w} - C \sum_{n=1}^{N} \xi_n \\ \text{subject to} \quad &y_n(\mathbf{w}^T \mathbf{z}_n + b) \geq 1 - \xi_n, \quad n = 1, \cdots, N \\ &\xi_n \geq 0, \quad n = 1, \cdots, N \end{aligned}$$

To check if the lagrange dual we derived is the same or similar to the primal, we can check if they would give the same optimal solution.

So first we consider the stationary condition of $L(\alpha, \lambda, \mu, \gamma)$ by taking partial derivatives:

$$\begin{aligned} \frac{\partial L}{\partial \alpha_n} &= \frac{\partial}{\partial \alpha_n} \left(\frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m \mathbf{z}_n^T \mathbf{z}_m - \sum_{n=1}^{N} \alpha_n + \lambda \sum_{n=1}^{N} y_n \alpha_n + \sum_{n=1}^{N} \mu_n(-\alpha_n) + \sum_{n=1}^{N} \gamma_n(\alpha_n - C) \right) \\ &= \sum_{m=1}^{N} \alpha_m y_n y_m \mathbf{z}_n^T \mathbf{z}_m - 1 + \lambda y_n + \mu_n - \gamma_n \end{aligned}$$

Setting the result to 0 gives:

$$\sum_{m=1}^{N} \alpha_m y_n y_m \mathbf{z}_n^T \mathbf{z}_m - 1 + \lambda y_n + \mu_n - \gamma_n = 0 \qquad (*)$$

And the primal constraints must hold, which are:

$$\sum_{n=1}^{N} y_n \alpha_n = 0 \qquad (1\text{-}1)$$

$$0 \leq \alpha_n \leq C, \quad n = 1, \cdots, N \qquad (1\text{-}2)$$

The dual feasibility requires the lagrange multipliers to satisfy:

$$\lambda \geq 0, \quad \mu_n \geq 0, \quad \gamma_n \geq 0, \quad n = 1, \cdots, N \tag{2}$$

Also, the complementary slackness conditions are:

$$\mu_n(-\alpha_n) = 0, \quad \gamma_n(\alpha_n - C) = 0, \quad n = 1, \cdots, N \tag{3}$$

I think that the problems are similar in some sense, the primal problem generates the solution by directly finding the optimal hyperplane, while the dual of the dual deals with the values of the lagrange multipliers α_n, focuses on the weight (importance) of the support vectors.