Computational Optimization Final Project

Ishita Padhiar and Mitchell Falkow

Introduction

For our project we chose to create a solution for the first problem, a support vector machine. The support vector machine aims to find a hyper plane that can classify a set of data points in space into two clusters.

1 Augmented Lagrangian Method

In order to create a solver using the Augmented Lagrangian Method we created a model derived from the following equation:

$$\min_{\mathbf{w},b,\mathbf{t}} \frac{\lambda}{2}||\mathbf{w}||^2 + \sum_{i=1}^{N} t_i \text{ s.t. } y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - t_i, t_i \geq 0, \forall i = 1, \ldots, N \tag{1}$$

In order to make the equations and derivatives easier we changed this format by replacing the summations with vector and matrix multiplication, which resulted in the following:

$$\min_{\mathbf{w},b,\mathbf{t}} \frac{\lambda}{2}||\mathbf{w}||^2 + \mathbf{e}^\top \mathbf{t} \text{ s.t. } D(X^\top \mathbf{w} + b\mathbf{e}) \geq \mathbf{e} - \mathbf{t} \tag{2}$$

We define $\mathbf{e} \in \mathbb{R}^N$ as the vector of ones and $D \in \mathbb{R}^{N \times N}$ as the diagonal matrix where $D_{i,i} = y_i$, for all $i = 1, \ldots, N$. Using this equation we created an Lagrangian and the augmented Lagrangian. Then we used the accelerated proximal gradient method as a sub-solver to get the final approximation for the function.

1.1 Lagrangian Equation

We formed the following Lagrangian function from the initial equation:

$$\mathcal{L}(\mathbf{w}, b, \mathbf{t}, \mathbf{u}) = \frac{\lambda}{2}||\mathbf{w}||^2 + \mathbf{e}^\top \mathbf{t} + \mathbf{u}^\top(\mathbf{e} - \mathbf{t} - D(X^\top \mathbf{w} + b\mathbf{e})) \tag{3}$$

Additionally, we generated the following augmented Lagrangian function, which includes the penalty parameter that we call β:

$$\mathcal{L}_\beta(\mathbf{w}, b, \mathbf{t}, \mathbf{u}) = \frac{\lambda}{2}||\mathbf{w}||^2 + \mathbf{e}^\top \mathbf{t} + \mathbf{u}^\top(\mathbf{e} - \mathbf{t} - D(X^\top \mathbf{w} + b\mathbf{e})) + \frac{\beta}{2}\left|\left|[\mathbf{e} - \mathbf{t} - D(X^\top \mathbf{w} + b\mathbf{e})]_+\right|\right|^2 \quad (4)$$

1.2 Minimizing the Objective function

From our ordinary Lagrangian we find that our original minimization problem has become

$$\min_{\mathbf{w},b,\mathbf{t}} \mathcal{L}(\mathbf{w}, b, \mathbf{t}, \mathbf{u}) = \frac{\lambda}{2}||\mathbf{w}||^2 + \mathbf{e}^\top \mathbf{t} + \mathbf{u}^\top(\mathbf{e} - \mathbf{t} - D(X^\top \mathbf{w} + b\mathbf{e})) \quad (5)$$

We take the partial derivatives of ordinary and augmented Lagrangian functions with respect to $\mathbf{w}$, b, and $\mathbf{t}$. The gradient for the ordinary Lagrangian function is

$$\nabla_w \mathcal{L} = \lambda \mathbf{w} - XD\mathbf{u} \qquad \nabla_b \mathcal{L} = -\mathbf{e}D\mathbf{u} \qquad \nabla_t \mathcal{L} = \mathbf{e} - \mathbf{u} \quad (6)$$

and the gradient for the augmented Lagrangian function is

$$\begin{aligned} \nabla_w \mathcal{L}_\beta &= \lambda \mathbf{w} - XD\mathbf{u} - \beta XD[\mathbf{e} - \mathbf{t} - D(X^\top \mathbf{w} + b\mathbf{e})]_+ \\ \nabla_b \mathcal{L}_\beta &= -\mathbf{e}D\mathbf{u} - \beta \mathbf{e}^\top D[\mathbf{e} - \mathbf{t} - D(X^\top \mathbf{w} + b\mathbf{e})]_+ \\ \nabla_t \mathcal{L}_\beta &= \mathbf{e} - \mathbf{u} - \beta[\mathbf{e} - \mathbf{t} - D(X^\top \mathbf{w} + b\mathbf{e})]_+ \end{aligned} \quad (7)$$

In the outer loop of our `ADM_Solver`, we attempt to minimize the value of our objective function, the ordinary Lagrangian function using the primal and dual residuals

$$\begin{aligned} \text{Primal residual} &= \mathbf{e} - \mathbf{t} - D(X^\top \mathbf{w} + b\mathbf{e}) \\ \text{Dual residual} &= ||\nabla_w \mathcal{L}|| + ||\nabla_b \mathcal{L}|| + ||\,[\nabla_t \mathcal{L}]_+|| \end{aligned}$$

We minimize the primal and dual residuals by using the Accelerated Projected Gradient (APG) method to solve the sub-problem

$$(\mathbf{w}^{(k+1)}, b_{k+1}, \mathbf{t}^{(k+1)}) = \arg\min_{\mathbf{w},b,\mathbf{t}} \mathcal{L}_\beta(\mathbf{w}, b, \mathbf{t}, \mathbf{u})$$

To perform the APG, we define y_w, y_b, y_t as ... respectively. We let ω_k be the extrapolation weight, and τ_k be the term used to calculate ω_k.

We use steepest gradient descent to locate the point at which the augmented Lagrangian, $\mathcal{L}_\beta$, is locally-Lipschitz continuous, which gives us $\alpha = \frac{1}{L}$, where L is the local Lipschitz multiplier.

After calculating the α, we then calculate the APG updates (see Figure 1).

```
w = min(w, y_w - alpha * grady_w);
b = min(b, y_b - alpha * grady_b);
t = max(0, min(t, y_t - alpha * grady_t));

% update t vars, calculate extrapolation weight
Tk1 = (1+sqrt(1+4*Tk0^2))/2;

Wk = (Tk0-1)/Tk1;
Tk0 = Tk1;

% update y vars
y_w = w + Wk * (w - w0);
y_b = b + Wk * (b - b0);
y_t = t + Wk * (t - t0);

w0 = w; b0 = b; t0 = t;

% compute gradient of the augmented Lagrangian function at (w,b,t)
v = e - t - D*(X'*w + b*e);
v_proj = max(0, v);

grad_w = lam*w - X*D*u - beta*X*D*v_proj;
grad_b = -e'*D*u - beta*e'*D*v_proj;
grad_t = e - u - beta*v_proj;

grady_w = (1+Wk)*grad_w - Wk*grad0_w;
grady_b = (1+Wk)*grad_b - Wk*grad0_b;
grady_t = (1+Wk)*grad_t - Wk*grad0_t;
```

Figure 1: Updating using the Accelerated Projected Gradient method

Data Set	Accuracy
`ranData_rho02`	97.50%
`ranData_rho08`	88.00%
`spamData`	N/A

Table 1: Accuracy of Classifications with the Augmented Lagrangian Method

The program stops once the largest residuals—primal or dual—divided by $\frac{1}{\sqrt{N}}$ is less than the tolerance or when the program reaches the maximum number of iterations. The stopping condition for the APG method is when the gradient error–which is equivalent to the dual residual of the ordinary Lagrangian function at that point–is less than the sub-tolerance or it reaches the maximum number of sub-iterations. By default this tolerance, sub-tolerance, iteration limit, and sub-iteration limit are set to 10^{-5}, 10^{-5}, 500, and 1000 respectively, but can be changed by specifying otherwise.

1.3 Results and Accuracy

Using our solver we received the following results, listed in Table 1, and the output in Figure 2 and 3.

```
>> test_model_SVM_rho02
    Testing by student code

    out iter = 1, pres = 3.6144e-04, dres = 1.9077e-01, subit = 10000
    out iter = 2, pres = 2.3400e-03, dres = 8.1329e-02, subit = 10000
    out iter = 3, pres = 4.0311e-04, dres = 9.0602e-02, subit = 6703
    out iter = 4, pres = 9.2058e-04, dres = 1.9565e-02, subit = 10000
    out iter = 5, pres = 1.5329e-04, dres = 3.3276e-02, subit = 10000
    out iter = 6, pres = 6.4820e-04, dres = 2.1036e-03, subit = 10000
    Running time is 133.4563
    classification accuracy on testing data: 98.00%
```

Figure 2: ALM Output for `ranData_rho02`

```
>> test_model_SVM_rho08
    Testing by student code

    out iter = 1, pres = 2.0860e-03, dres = 2.4817e-01, subit = 10000
    out iter = 2, pres = 2.3728e-03, dres = 1.3448e-01, subit = 10000
    out iter = 3, pres = 2.6226e-03, dres = 9.5491e-02, subit = 10000
    out iter = 4, pres = 1.6838e-03, dres = 8.5580e-02, subit = 10000
    out iter = 5, pres = 2.7483e-03, dres = 7.9341e-02, subit = 10000
    out iter = 6, pres = 1.8695e-03, dres = 7.5756e-02, subit = 10000
    out iter = 7, pres = 2.5196e-03, dres = 7.3624e-02, subit = 10000
    out iter = 8, pres = 1.8632e-03, dres = 7.2321e-02, subit = 10000
    out iter = 9, pres = 2.4702e-03, dres = 7.3172e-02, subit = 8331
    out iter = 10, pres = 1.7931e-03, dres = 7.2161e-02, subit = 10000
    out iter = 11, pres = 2.3936e-03, dres = 7.2989e-02, subit = 6576
    out iter = 12, pres = 1.8684e-03, dres = 6.9473e-02, subit = 10000
    out iter = 13, pres = 2.3186e-03, dres = 7.0175e-02, subit = 4785
    out iter = 14, pres = 1.8326e-03, dres = 6.7424e-02, subit = 10000
    out iter = 15, pres = 2.2801e-03, dres = 6.9151e-02, subit = 3669
    out iter = 16, pres = 1.8224e-03, dres = 6.6195e-02, subit = 8705
    out iter = 17, pres = 2.2497e-03, dres = 6.8148e-02, subit = 2346
    out iter = 18, pres = 1.8184e-03, dres = 6.5305e-02, subit = 6211
    out iter = 19, pres = 2.2323e-03, dres = 6.7331e-02, subit = 1242
    out iter = 20, pres = 1.7971e-03, dres = 6.4298e-02, subit = 3448
    out iter = 21, pres = 2.2027e-03, dres = 6.6725e-02, subit = 623
    out iter = 22, pres = 1.7836e-03, dres = 6.3542e-02, subit = 1770
    out iter = 23, pres = 2.1840e-03, dres = 6.4087e-02, subit = 300
    out iter = 24, pres = 1.7436e-03, dres = 6.2827e-02, subit = 722
    out iter = 25, pres = 2.1587e-03, dres = 6.2948e-02, subit = 77
    out iter = 26, pres = 1.6960e-03, dres = 6.2530e-02, subit = 242
    out iter = 27, pres = 2.1656e-03, dres = 6.2735e-02, subit = 51
    out iter = 28, pres = 1.6935e-03, dres = 6.0727e-02, subit = 104
    out iter = 29, pres = 2.0743e-03, dres = 6.2406e-02, subit = 33
    out iter = 30, pres = 1.6169e-03, dres = 5.8564e-02, subit = 71
    out iter = 31, pres = 1.9707e-03, dres = 5.7909e-02, subit = 43
    out iter = 32, pres = 1.3781e-03, dres = 5.1600e-02, subit = 37
    out iter = 33, pres = 1.6937e-03, dres = 5.0909e-02, subit = 31
    out iter = 34, pres = 1.0970e-03, dres = 3.3661e-02, subit = 23
    out iter = 35, pres = 1.0646e-03, dres = 4.3348e-02, subit = 7
    out iter = 36, pres = 8.4857e-04, dres = 1.9367e-02, subit = 29
    out iter = 37, pres = 6.2460e-04, dres = 3.5464e-02, subit = 10
    out iter = 38, pres = 6.5604e-04, dres = 1.1701e-02, subit = 21
    Running time is 377.5199
    classification accuracy on testing data: 88.00%

    Testing by instructor code

    out iter = 1, pres = 2.8486e-03, dres = 1.5961e-03, subit = 1000
    out iter = 2, pres = 4.2314e-05, dres = 3.7069e-04, subit = 1000
    Running time is 0.3937
    classification accuracy on testing data: 87.50%
```

Figure 3: ALM Output for `ranData_rho08`

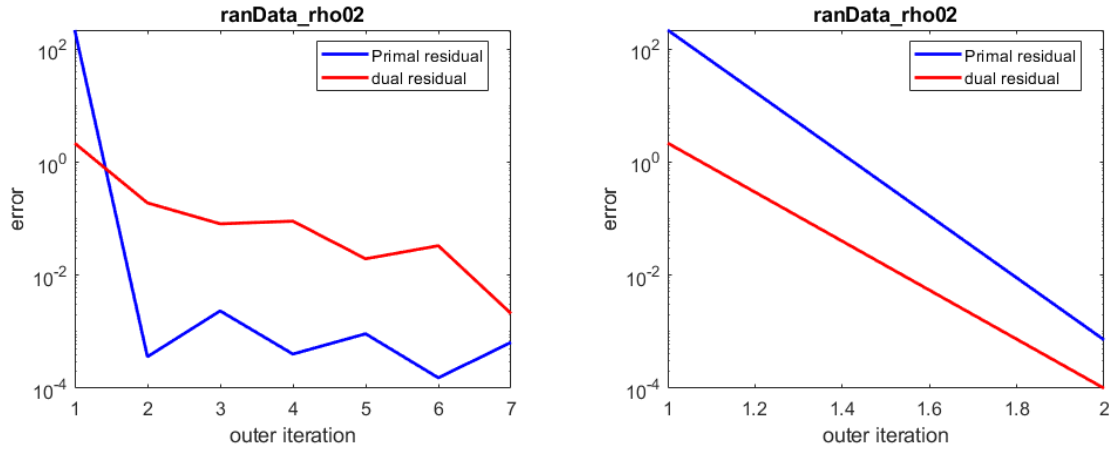

Figure 4: Results from `ranData_rho02` on the ALM solver. (Student on left, instructor on right)

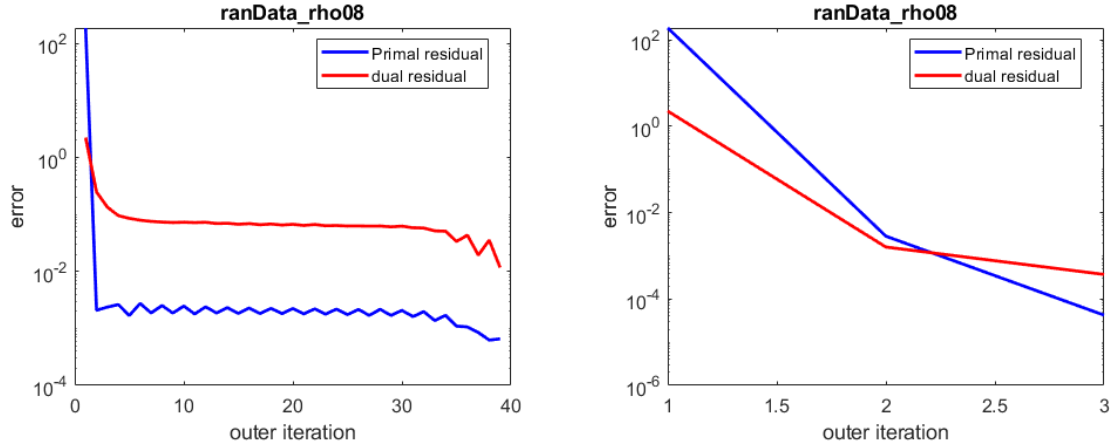

Figure 5: Results from `ranData_rho08` on the ALM solver. (Student on left, instructor on right)

2 Alternating Direction Multipliers Method

In order to create a solver using the Alternating Direction Multipliers Method we created a model derived from the following equation:

$$\min_{\mathbf{w},b,\mathbf{t}} \sum_{i=1}^{N} max(0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)) + \frac{\lambda}{2}||\mathbf{w}||^2 \tag{8}$$

Similarly to the ALM method, in order to make the equations and derivatives easier to calculate, we changed this format by replacing the summations with vector and matrix multiplication, which resulted in the following:

$$\min_{\mathbf{w},b,\mathbf{t}} \left|\left|[\mathbf{e} - D(X^\top \mathbf{w} + b\mathbf{e})]_+\right|\right|_1 + \frac{\lambda}{2}||\mathbf{w}||_2^2 \tag{9}$$

We define $\mathbf{e} \in \mathbb{R}^N$ as the vector of ones and $D \in \mathbb{R}^{N\times N}$ as the diagonal matrix where $D_{i,i} = y_i$, for all $i = 1, \ldots, N$. Using this equation we created an Lagrangian and the augmented Lagrangian. Then we used the accelerated proximal gradient method as a sub-solver to get the final approximation for the function.

2.1 Lagrangian Equation

In order to use the ADMM method for this problem we changed the form of the equation and introduced a variable $\mathbf{z}$ such that

$$\mathbf{z} = \mathbf{e} - D(X^\top \mathbf{w} + b\mathbf{e})$$

This gives us a new objective function:

$$\min_{\mathbf{w},\mathbf{z}} ||\,[\mathbf{z}]_+||_1 + \frac{\lambda}{2}||\mathbf{w}||^2 \text{ s.t. } \mathbf{e} - D(X^\top \mathbf{w} + b\mathbf{e}) - \mathbf{z} = 0 \tag{10}$$

Using this variable $\mathbf{z}$ we generated the following augmented Lagrangian function, which includes the penalty parameter that we call β:

$$\mathrm{L}_\beta(\mathbf{w}, b, \mathbf{z}, \mathbf{v}) = \tfrac{\lambda}{2}||\mathbf{w}||^2 + ||\,[\mathbf{z}]_+||_1 + v^\top(\mathbf{e} + D(X^\top \mathbf{w} + b\mathbf{e}) - \mathbf{z}) + \tfrac{\beta}{2}\left|\left|[\mathbf{e} - D(X^\top \mathbf{w} + b\mathbf{e}) - \mathbf{z}]_+\right|\right|^2$$

2.2 Solving the Subproblems

In order to use the ADMM method, we must solve the subproblems for each individual parameter $\mathbf{w}^{(k)}, b, \mathbf{z}^{(k)}$ and $\mathbf{v}^{(k)}$.

We set the updates to be as follows:

$\mathbf{w}$ update:

$$\mathbf{w}^{(k+1)} = \arg\min_{\mathbf{w}} \mathcal{L}_\beta(\mathbf{w}, b_k, \mathbf{z}^{(k)}, \mathbf{v}^{(k)}) \tag{11}$$

We solve this by setting the derivative to 0.

$$\begin{aligned}\nabla_w \mathcal{L}_\beta(\mathbf{w}, b_k, \mathbf{z}^{(k)}, \mathbf{v}^{(k)}) &= \lambda\mathbf{w} - XD\mathbf{v}^{(k)} - \beta XD(\mathbf{e} - DX^\top\mathbf{w} + b_k D\mathbf{e} - \mathbf{z}^{(k)}) \\ 0 &= \lambda\mathbf{w} - XD\mathbf{v}^{(k)} - \beta XD(\mathbf{e} - DX^\top\mathbf{w} + b_k D\mathbf{e} - \mathbf{z}^{(k)}) \\ 0 &= \lambda\mathbf{w} - XD\mathbf{v}^{(k)} - \beta XD\mathbf{e} + \beta XDDX^\top\mathbf{w} + b_k\beta XDD\mathbf{e} + \beta XD\mathbf{z}^{(k)}\end{aligned}$$

Note that $D^\top D = I \in \mathbb{R}^{N\times N}$.

$$\begin{aligned}\lambda\mathbf{w} + \beta XX^\top\mathbf{w} &= XD\mathbf{v}^{(k)} + \beta XD\mathbf{e} - b_k\beta XDD\mathbf{e} - \beta XD\mathbf{z}^k \\ \mathbf{w}^{(k+1)} &= (\lambda I + \beta XX^\top)^{-1}(XD\mathbf{v}^{(k)} + \beta XD\mathbf{e} - b_k\beta X\mathbf{e} - \beta XD\mathbf{z}^k) \qquad (12)\end{aligned}$$

(13)

b update:

$$b_{k+1} = \arg\min_b \mathcal{L}_\beta(\mathbf{w}^{k+1}, b, \mathbf{z}^{(k)}, \mathbf{v}^{(k)}) \qquad (14)$$

We solve this by setting the derivative to 0.

$$\begin{aligned}\nabla_b \mathcal{L}_\beta(\mathbf{w}^{(k+1)}, b, \mathbf{z}^{(k)}, \mathbf{v}^{(k)}) &= -\mathbf{e}^\top D^\top\mathbf{v}^{(k)} - \beta\mathbf{e}^\top D^\top(\mathbf{e} - DX^\top\mathbf{w}^{(k+1)} - bD\mathbf{e} - \mathbf{z}^{(k)}) \\ 0 &= -\mathbf{e}^\top D^\top\mathbf{v}^{(k)} - \beta\mathbf{e}^\top D^\top(\mathbf{e} - DX^\top\mathbf{w}^{(k+1)} - bD\mathbf{e} - \mathbf{z}^{(k)}) \\ \beta\mathbf{e}^\top D^\top D\mathbf{e}b &= -\mathbf{e}^\top D(\mathbf{v}^{(k)} - \beta(\mathbf{e} - DX^\top\mathbf{w}^{(k+1)} - \mathbf{z}^{(k)}) \\ \beta\mathbf{e}^\top\mathbf{e}b &= \mathbf{e}^\top D(\mathbf{v}^{(k)} - \beta(\mathbf{e} - DX^\top\mathbf{w}^{(k+1)} - \mathbf{z}^{(k)}) \\ b_{k+1} &= (\frac{1}{\beta N})\mathbf{e}^\top D(\mathbf{v}^{(k)} - \beta(\mathbf{e} - DX^\top\mathbf{w}^{(k+1)} - \mathbf{z}^{(k)}) \qquad (15)\end{aligned}$$

$\mathbf{z}$ update:

$$\mathbf{z}^{(k+1)} = argmin_z \mathcal{L}_\beta(\mathbf{w}^{k+1}, b_{k+1}, \mathbf{z}, \mathbf{v}^{(k)}) \qquad (16)$$

We used the proximal mapping of the one norm to get $\mathbf{z}$:

$$\begin{aligned}\mathbf{z}^{k+1} &= \arg\min_z ||z||_1 + \mathbf{v}^{(k)\top}(\mathbf{e} - DX^\top w^{k+1} - b_{k+1}D\mathbf{e} - \mathbf{z}) + \frac{\beta}{2}||\mathbf{e} - DX^\top\mathbf{w}^{k+1} - b_{k+1}D\mathbf{e} - \mathbf{z}||_2^2 \\ \mathbf{z}^{k+1} &= \arg\min_z ||z||_1 + \frac{\beta}{2}||\mathbf{e} - DX^\top\mathbf{w}^{k+1} - b_{k+1}D\mathbf{e} - \mathbf{z} + \frac{v^k}{\beta}||_2^2 \\ \mathbf{z}^{k+1} &= \arg\min_z ||z||_1 + \frac{\beta}{2}||\mathbf{z} - (\mathbf{e} - DX^\top\mathbf{w}^{k+1} - b_{k+1}D\mathbf{e} + \frac{v^k}{\beta})||_2^2 \\ \mathbf{z}^{k+1} &= _{\frac{1}{\beta}||\cdot||_1}\ (\mathbf{e} - DX^\top\mathbf{w}^{k+1} - b_{k+1}D\mathbf{e} + \frac{v^k}{\beta}) \\ \mathbf{z}^{k+1} &= \text{sign}(\mathbf{e} - DX^\top\mathbf{w}^{k+1} - b_{k+1}D\mathbf{e} + \frac{v^k}{\beta}) \cdot \max(0, \max(0, \mathbf{e} - DX^\top\mathbf{w}^{k+1} - b_{k+1}D\mathbf{e} + \frac{v^k}{\beta}) - \frac{1}{\beta}) \qquad (17)\end{aligned}$$

$\mathbf{v}$ update:

$$\mathbf{v}^{k+1} = \mathbf{v}^k + \beta(\mathbf{e} - DX^\top\mathbf{w}^{k+1} - b_{k+1}D\mathbf{e} - z^{k+1}) \qquad (18)$$

In the ADMM algorithm we update these variables one by one and then recalculate the primal and dual residuals for the updated values. The solver stops when the largest residual–primal or dual–is less than the tolerance we set. This is our primal residual:

$$\text{Primal residual} = ||\mathbf{e} - DX^\top \mathbf{w}^{k+1} + b_{k+1} D\mathbf{e} - \mathbf{z}^{k+1}||$$

We were unable to get our ADMM solver to work properly due to some errors in our code. Additionally, we had difficulty deriving the dual feasibility condition.