MATH 173B, HW7

Victor Pekkari — epekkari@ucsd.edu

March 14, 2025

Problem 1

(a)

We try to minimize the following function in P2:

$$\min_{x\in\mathbb{R}^n, z\in\mathbb{R}^m} f(z) + \lambda g(x) \tag{*}$$

From the constraint in P2 we know that:

$$Ax = z$$

This means we can do substitute z for Ax in **(*)** to get P1:

$$\min_{x\in\mathbb{R}^n} f(Ax) + \lambda g(x)$$

This ensures that every point every point-pair (a, b) that we put into $g(a)$ and $f(b)$ will have the same relationship for P1 and P2.

(b)

We can write the augmented Lagrangian as:

$$\mathcal{L}(x, z, y) = f(x) + g(x) + y^T(Ax - z) + \frac{\rho}{2}\|Ax - z\|^2$$

This give us the following ADMM-algorithm:

Initialize: $\rho > 0, x^{(0)} \in \mathbb{R}^n, z^{(0)}, y^{(0)} \in \mathbb{R}$

$$\begin{cases} x^{(t+1)} = \arg\min_x \mathcal{L}(x, z_t, y_t) \\ z^{(t+1)} = \arg\min_z \mathcal{L}(x_{t+1}, z, y_t) \\ y^{(t+1)} = y^{(t)} + \rho(Ax^{(t+1)} - z^{(t+1)}) \end{cases}$$

We can't get a closed form solution for the argmin since we don't know if the functions $f(x)$ and $g(x)$ are convex or not. This means we can't simplify this further. We can use numerical methods to approximate the argmin, many numerical approximator rely on the fact that the function is convex, so our approximator algorithm won't be that effective.

Problem 2

(a)

$$\min_{w,b,z} \sum_{i=1}^{N} \frac{1}{2}(z_i - y_i)^2 + \frac{1}{2}\|w\|^2 \qquad s.t \quad z_i = w^T x_i + b$$

(b)

$$\mathcal{L}_\rho(w, b, z, \nu) = \sum_{i=1}^{N} \left[\frac{1}{2}(z_i - y_i)^2 + \frac{\lambda}{2}\|w\|^2 + \nu_i(w^T x_i + b - z_i) + \frac{\rho}{2}\|w^T x_i + b - z_i\|^2 \right]$$

(c)

The ADMM updates consist of the following steps:
Initialize the variables: $w^{(0)} \in \mathbb{R}^d, b^{(0)}, z_i^{(0)}, u_i^{(0)} \in \mathbb{R}$

Do the thing below for desired number of iterations:

Update w, b

$$(w^{(t+1)}, b^{(t+1)}) = \arg\min_{w,b} \frac{\lambda}{2}\|w\|_2^2 + \frac{\rho}{2}\sum_{i=1}^{N}(w^T x_i + b - z_i^t + u_i^t)^2.$$

Solving for w,

$$w^{(t+1)} = \left(\lambda I + \rho \sum_{i=1}^{N} x_i x_i^T\right)^{-1} \left(\rho \sum_{i=1}^{N} x_i (z_i^t - u_i^t - b^{(t+1)})\right).$$

Solving for b,

$$b^{(t+1)} = \frac{1}{N}\sum_{i=1}^{N}(z_i^t - u_i^t - x_i^T w^{(t+1)}).$$

Update z

$$z_i^{(t+1)} = \arg\min_{z} \frac{1}{2}(z_i - y_i)^2 + \frac{\rho}{2}(z_i - w^{(t+1)} x_i - b^{(t+1)} + u_i^t)^2.$$

Solving for z_i,

$$z_i^{(t+1)} = \frac{y_i + \rho(w^{(t+1)} x_i + b^{(t+1)} - u_i^t)}{1 + \rho}.$$

Update the dual variable u

$$u_i^{(t+1)} = u_i^t + (w^{(t+1)} x_i + b^{(t+1)} - z_i^{(t+1)}).$$

This iterative process continues until convergence...

Problem 3

(a)

This function is convex in both w and b.

$$\min_{w,b} \frac{1}{N} \sum_{i=1}^{N} \log(1 + e^{-w^T x_i y_i}) + \frac{\lambda}{2} \|z\|^2$$

We construct the augmented Lagrangian:

$$\mathcal{L}(w, z, y) = \frac{1}{N} \sum_{i=1}^{N} \left[\log(1 + e^{-w^T x_i y_i}) + \frac{\lambda}{2} \|z\|^2 \right] + y^T (w - z) + \frac{\rho}{2} \|w - z\|^2$$

For the w-update, we solve:

$$w^{(t+1)} = \arg\min_{w} \frac{1}{N} \sum_{i=1}^{N} \log(1 + e^{-w^T x_i y_i}) + \frac{\rho}{2} \|w - z^{(t)} + \frac{y^{(t)}}{\rho}\|^2. \tag{1}$$

Since there is no closed-form solution, we use gradient descent to solve it, since we've been working with GD for 2 quarters I'll leave it out from this solution, it is implemented in my code however...

For the z-update:

$$z^{(t+1)} = \arg\min_{z} \frac{\lambda}{2} \|z\|^2 + \frac{\rho}{2} \|w^{(t+1)} - z + \frac{y^{(t)}}{\rho}\|^2. \tag{2}$$

Taking the derivative and setting it to zero (we can do this because the function is convex in z), we get:

$$z^{(t+1)} = \frac{\rho w^{(t+1)} + y^{(t)}}{\rho + \lambda}. \tag{3}$$

Finally, the dual variable y is updated as:

$$y^{(t+1)} = y^{(t)} + \rho(w^{(t+1)} - z^{(t+1)}). \tag{4}$$

Thus, the ADMM updates are summarized as:

$$\begin{cases} w^{(t+1)} = \arg\min_w \frac{1}{N} \sum_{i=1}^{N} \log(1 + e^{-w^T x_i y_i}) + \frac{\rho}{2} \|w - z^{(t)} + \frac{y^{(t)}}{\rho}\|^2, \\ z^{(t+1)} = \frac{\rho w^{(t+1)} + y^{(t)}}{\rho + \lambda}, \\ y^{(t+1)} = y^{(t)} + \rho(w^{(t+1)} - z^{(t+1)}). \end{cases}$$

Where the w is solved through GD

(b) Performance Evaluation

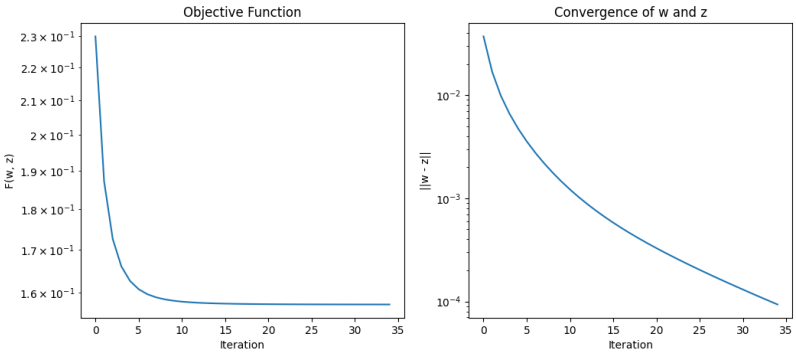

Figure 1: Objective function and convergence of w and z over iterations.

Results:
Training error: 0.0325
Test error: 0.0340

(ii) Performance and Convergence

The objective function shows a fast decrease in the beginning. The decease never stops, though the pace at which the function decreases slows down. The plot of $\|w - z\|$ in log-scale shows an exponential decay. The training error (0.0325) and test error (0.0340) are close, meaning the model generalizes well.

The computational complexity per iteration is primarily determined by the gradient descent step in the w-update. all other operations "inside the iteration" have constant timecomplexity. So the timecomplexity is $O(\#GD - iterations)$.

(iii) Advantages and Disadvantages of ADMM

Advantages:

- Support parallelisation
- Cost function doesn't have to be differentiable
- It provides robust convergence guarantees

Disadvantages:

- Each iteration involves solving a subproblem for w, which can be computationally expensive. If we for example have to run GD for many iterations
- Convergence can be slow

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split


def ADMM(X_train, y_train, lambda_, rho, max_iter=100, tol=1e-4):
    d = X_train.shape[1]
    w, z, y = np.zeros(d), np.zeros(d), np.zeros(d)
    obj_values, w_z_diffs = [], []

    for iter_count in range(max_iter):
        # w-update with gradient descent
        for _ in range(40):
            y_pred = X_train @ w
            neg_pred = -y_train * y_pred
            sig_vals = sig_func(neg_pred)
            term1 = -(X_train.T @ (y_train * sig_vals)) / X_train.shape[0]
            term2 = rho * (w - z + y / rho)
            gradient = term1 + term2
            w = w - 0.1 * gradient

        z = (rho * w + y) / (rho + lambda_)
        y = y + rho * (w - z)

        current_loss = log_func(w, X_train, y_train)
        regularization = (lambda_/2) * np.linalg.norm(z)**2
        obj_values.append(current_loss + regularization)

        primal_residual = np.linalg.norm(w - z)
        w_z_diffs.append(primal_residual)

        if primal_residual < tol:
            break

    return w, obj_values, w_z_diffs

def log_func(w, X, y):
    return np.mean(np.log(1 + np.exp(-y * (X @ w))))

def sig_func(x):
    return 1 / (1 + np.exp(-x))

mnist_data = fetch_openml('mnist_784', version=1, as_frame=False)
features, labels = mnist_data.data, mnist_data.target.astype(int)

digit1, digit2 = 3, 8
selected_indices = (labels == digit1) | (labels == digit2)
features = features[selected_indices]
labels = labels[selected_indices]

binary_labels = np.where(labels == digit1, -1, 1)

scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

X_train, X_test, y_train, y_test = train_test_split(
    scaled_features, binary_labels, test_size=0.2, random_state=42
)

lambda_ = 0.1
rho = 1.0
model_weights, convergence_vals, residuals = ADMM(X_train, y_train, lambda_, rho)

fig = plt.figure(figsize=(12, 5))

ax1 = fig.add_subplot(1, 2, 1)
ax1.plot(convergence_vals)
ax1.set_yscale('log')
ax1.set_xlabel('Iteration')
ax1.set_ylabel('F(w, z)')
ax1.set_title('Objective Function')

ax2 = fig.add_subplot(1, 2, 2)
ax2.plot(residuals)
ax2.set_yscale('log')
ax2.set_xlabel('Iteration')
ax2.set_ylabel('||w - z||')
ax2.set_title('Convergence of w and z')

plt.tight_layout()
plt.show()

train_predictions = np.sign(X_train @ model_weights)
test_predictions = np.sign(X_test @ model_weights)

train_error = np.mean(train_predictions != y_train)
test_error = np.mean(test_predictions != y_test)

print(f"Training error: {train_error:.4f}")
print(f"Test error: {test_error:.4f}")
```