Roya Ghamari

Student ID: 2071969

May 2024

```
import numpy as np
import time
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

Generating synthetic data:

```
def generate_logistic_data(m, d, k, test_size=0.2):

    np.random.seed(0)
    A = np.random.normal(0, 1, (m, d))  # Feature matrix A generated from N(0,1)
    x = np.random.normal(0, 1, (d, k))  # Parameter matrix x
    e = np.random.normal(0, 1, (m, k))  # Noise matrix e

    scores = A @ x + e
    b = np.argmax(scores, axis=1)  # Determine class labels by max scoring class

    A_train, A_test, b_train, b_test = train_test_split(A, b, test_size=test_size, random_state=42)
    return A_train, A_test, b_train, b_test
```

```
m, d, k = 1000, 1000, 50
A_train, A_test, b_train, b_test = generate_logistic_data(m, d, k, test_size=0.2)
```

Generating logistic regression objective function and related functions:

```
def generate_logistic_objective(A, b, k):

    m, d = A.shape

    def f(X):
        exp_terms = np.exp(A @ X)
        sum_exp = exp_terms.sum(axis=1, keepdims=True)
        log_prob = np.log(sum_exp)
        correct_class_scores = exp_terms[np.arange(m), b]
        loss = -np.sum(np.log(correct_class_scores) - log_prob)
        return loss

    def grad_f(X):
        exp_terms = np.exp(A @ X)
        sum_exp = exp_terms.sum(axis=1, keepdims=True)
        probs = exp_terms / sum_exp
        indicator = np.zeros_like(probs)
        indicator[np.arange(m), b] = 1
        gradient = -A.T @ (indicator - probs)
        return gradient

    return f, grad_f
```

```
def softmax(Z):

    e_Z = np.exp(Z - np.max(Z, axis=1, keepdims=True))
    return e_Z / e_Z.sum(axis=1, keepdims=True)

def compute_accuracy(X, A, b):

    predictions = softmax(A @ X)
    predicted_classes = np.argmax(predictions, axis=1)
    return np.mean(predicted_classes == b)
```

```
def gradient_descent(A_train, b_train, A_test, b_test, k, initial_X_gd, num_iterations, step_size_strategy, initial_learning_rate):
    X = initial_X_gd
    m, d = A_train.shape
    accuracies_train = []
    accuracies_test = []
    losses_train = []
    losses_test = []
    times = []
    start_time = time.time()

    # Get the functions for calculating the objective and gradient for training and test data
    f_train, grad_f_train = generate_logistic_objective(A_train, b_train, k)

    f_test, _ = generate_logistic_objective(A_test, b_test, k)

    for i in range(num_iterations):
        # Use the provided function to calculate gradient
        gradient = grad_f_train(X)

        alpha = initial_learning_rate

        X -= alpha * gradient

        if i % 10 == 0 or i == num_iterations:
            current_time = time.time() - start_time
            times.append(current_time)

            accuracy_train = compute_accuracy(X, A_train, b_train)
            accuracy_test = compute_accuracy(X, A_test, b_test)
            accuracies_train.append(accuracy_train)
            accuracies_test.append(accuracy_test)

            loss_train = f_train(X)
            loss_test = f_test(X)
            losses_train.append(loss_train)
            losses_test.append(loss_test)

            print(f"Iteration {i}: Time: {current_time:.2f}s, Train Acc: {accuracy_train:.4f}, Test Acc: {accuracy_test:.4f}, Train Loss: {l

    total_time = time.time() - start_time
    return X, accuracies_train, accuracies_test, losses_train, losses_test, times, total_time
```

```
np.random.seed(42)
initial_X_gd = np.random.normal(0, 1, (d, k))


overall_start_time = time.time()


X_train_opt_gd, accuracies_train_gd, accuracies_test_gd, losses_train_gd, losses_test_gd, train_times_gd, total_train_time_gd = gradient_des
    A_train, b_train, A_test, b_test, k, initial_X_gd, 300, 'fixed', 1e-3
)


total_time = time.time() - overall_start_time


print("GD Training Complete")
print(f"Final GD Train Accuracy: {accuracies_train_gd[-1]:.4f}")
print(f"Final GD Test Accuracy: {accuracies_test_gd[-1]:.4f}")
print(f"Final GD Train Loss: {losses_train_gd[-1]:.4f}")
print(f"Final GD Test Loss: {losses_test_gd[-1]:.4f}")
print(f"GD Total Training Time: {total_train_time_gd:.2f} seconds")
```

```
Iteration 0: Time: 0.03s, Train Acc: 0.0225, Test Acc: 0.0050, Train Loss: 44454639.7166, Test Loss: 2955749.8764
Iteration 10: Time: 0.28s, Train Acc: 0.0850, Test Acc: 0.0050, Train Loss: 32875461.5742, Test Loss: 2898520.6051
Iteration 20: Time: 0.48s, Train Acc: 0.1625, Test Acc: 0.0050, Train Loss: 23546845.4720, Test Loss: 2851438.9550
Iteration 30: Time: 0.61s, Train Acc: 0.2762, Test Acc: 0.0050, Train Loss: 16074452.0250, Test Loss: 2813589.1143
Iteration 40: Time: 0.74s, Train Acc: 0.4425, Test Acc: 0.0050, Train Loss: 10408286.3204, Test Loss: 2784266.5062
Iteration 50: Time: 0.87s, Train Acc: 0.6062, Test Acc: 0.0050, Train Loss: 6491516.6971, Test Loss: 2763965.7762
Iteration 60: Time: 1.01s, Train Acc: 0.7300, Test Acc: 0.0050, Train Loss: 3839805.6122, Test Loss: 2751426.9538
Iteration 70: Time: 1.17s, Train Acc: 0.8325, Test Acc: 0.0050, Train Loss: 2099737.8589, Test Loss: 2747126.3470
Iteration 80: Time: 1.30s, Train Acc: 0.9087, Test Acc: 0.0050, Train Loss: 1131463.6050, Test Loss: 2744391.5826
Iteration 90: Time: 1.44s, Train Acc: 0.9500, Test Acc: 0.0050, Train Loss: 591138.2785, Test Loss: 2740871.8071
Iteration 100: Time: 1.57s, Train Acc: 0.9738, Test Acc: 0.0050, Train Loss: 282235.2277, Test Loss: 2738069.9645
```

```
Iteration 110: Time: 1.69s, Train Acc: 0.9862, Test Acc: 0.0050, Train Loss: 130660.9909, Test Loss: 2736429.9555
Iteration 120: Time: 1.82s, Train Acc: 0.9950, Test Acc: 0.0050, Train Loss: 49516.7908, Test Loss: 2734869.7127
Iteration 130: Time: 1.96s, Train Acc: 0.9975, Test Acc: 0.0050, Train Loss: 14712.2117, Test Loss: 2733894.2544
Iteration 140: Time: 2.08s, Train Acc: 1.0000, Test Acc: 0.0050, Train Loss: 7658.3084, Test Loss: 2733447.8645
Iteration 150: Time: 2.23s, Train Acc: 1.0000, Test Acc: 0.0050, Train Loss: 6292.8036, Test Loss: 2733171.8229
Iteration 160: Time: 2.35s, Train Acc: 1.0000, Test Acc: 0.0050, Train Loss: 5406.1092, Test Loss: 2732944.4988
Iteration 170: Time: 2.50s, Train Acc: 1.0000, Test Acc: 0.0050, Train Loss: 4759.5810, Test Loss: 2732747.9162
Iteration 180: Time: 2.63s, Train Acc: 1.0000, Test Acc: 0.0050, Train Loss: 4261.3690, Test Loss: 2732573.7584
Iteration 190: Time: 2.75s, Train Acc: 1.0000, Test Acc: 0.0050, Train Loss: 3863.3911, Test Loss: 2732417.0082
Iteration 200: Time: 2.89s, Train Acc: 1.0000, Test Acc: 0.0050, Train Loss: 3537.0587, Test Loss: 2732274.2812
Iteration 210: Time: 3.03s, Train Acc: 1.0000, Test Acc: 0.0050, Train Loss: 3264.0200, Test Loss: 2732143.1451
Iteration 220: Time: 3.15s, Train Acc: 1.0000, Test Acc: 0.0050, Train Loss: 3031.8477, Test Loss: 2732021.7766
Iteration 230: Time: 3.31s, Train Acc: 1.0000, Test Acc: 0.0050, Train Loss: 2831.7797, Test Loss: 2731908.7659
Iteration 240: Time: 3.43s, Train Acc: 1.0000, Test Acc: 0.0050, Train Loss: 2657.4378, Test Loss: 2731802.9971
Iteration 250: Time: 3.57s, Train Acc: 1.0000, Test Acc: 0.0050, Train Loss: 2504.0563, Test Loss: 2731703.5691
Iteration 260: Time: 3.69s, Train Acc: 1.0000, Test Acc: 0.0050, Train Loss: 2367.9963, Test Loss: 2731609.7430
Iteration 270: Time: 3.83s, Train Acc: 1.0000, Test Acc: 0.0050, Train Loss: 2246.4267, Test Loss: 2731520.9043
Iteration 280: Time: 3.97s, Train Acc: 1.0000, Test Acc: 0.0050, Train Loss: 2137.1097, Test Loss: 2731436.5364
Iteration 290: Time: 4.10s, Train Acc: 1.0000, Test Acc: 0.0050, Train Loss: 2038.2514, Test Loss: 2731356.2000
GD Training Complete
Final GD Train Accuracy: 1.0000
Final GD Test Accuracy: 0.0050
Final GD Train Loss: 2038.2514
Final GD Test Loss: 2731356.2000
GD Total Training Time: 4.22 seconds
```

```
def gauss_southwell_bcgd(A_train, b_train, A_test, b_test, k, initial_X_bcgd, learning_rate, num_iterations):
    X = initial_X_bcgd
    m, d = A_train.shape
    accuracies_train = []
    accuracies_test = []
    losses_train = []
    losses_test = []
    times = []
    start_time = time.time()

    # Get the functions for calculating the objective and gradient for training and test data
    f_train, grad_f_train = generate_logistic_objective(A_train, b_train, k)

    f_test, _ = generate_logistic_objective(A_test, b_test, k)

    for i in range(num_iterations):
        # Calculate gradients for each class and find the class with the largest gradient norm
        gradient_norms = [np.linalg.norm(grad_f_train(X)[:, j]) for j in range(k)]
        idx = np.argmax(gradient_norms)
        block_gradient = grad_f_train(X)[:, idx]
        X[:, idx] -= learning_rate * block_gradient

        if i % 10 == 0:
            current_time = time.time() - start_time
            times.append(current_time)

            accuracy_train = compute_accuracy(X, A_train, b_train)
            accuracy_test = compute_accuracy(X, A_test, b_test)
            accuracies_train.append(accuracy_train)
            accuracies_test.append(accuracy_test)

            loss_train = f_train(X)
            loss_test = f_test(X)
            losses_train.append(loss_train)
            losses_test.append(loss_test)

            print(f"Iteration {i}: Time: {current_time:.2f}s, Train Acc: {accuracy_train:.4f}, Test Acc: {accuracy_test:.4f}, Train Loss: {l

    total_time = time.time() - start_time
    return X, accuracies_train, accuracies_test, losses_train, losses_test, times, total_time
```

```
np.random.seed(42)


initial_X_bcgd = np.random.normal(0, 1, (d, k))


optimized_X_bcgd, accuracies_train_bcgd, accuracies_test_bcgd, losses_train_bcgd, losses_test_bcgd, times_bcgd, total_time_bcgd = gauss_sout
    A_train, b_train, A_test, b_test, k, initial_X_bcgd, 1e-3, 300
)

print("BCGD Optimization finished.")
if accuracies_train_bcgd:
    print(f"Final BCGD Train Accuracy: {accuracies_train_bcgd[-1]:.4f}")
    print(f"Final BCGD Test Accuracy: {accuracies_test_bcgd[-1]:.4f}")
else:
    print("No BCGD accuracy data available.")

if losses_train_bcgd:
    print(f"Final BCGD Train Loss: {losses_train_bcgd[-1]:.4f}")
    print(f"Final BCGD Test Loss: {losses_test_bcgd[-1]:.4f}")
else:
    print("No BCGD loss data available.")

print(f"BCGD Total time: {total_time_bcgd:.2f} seconds")
```

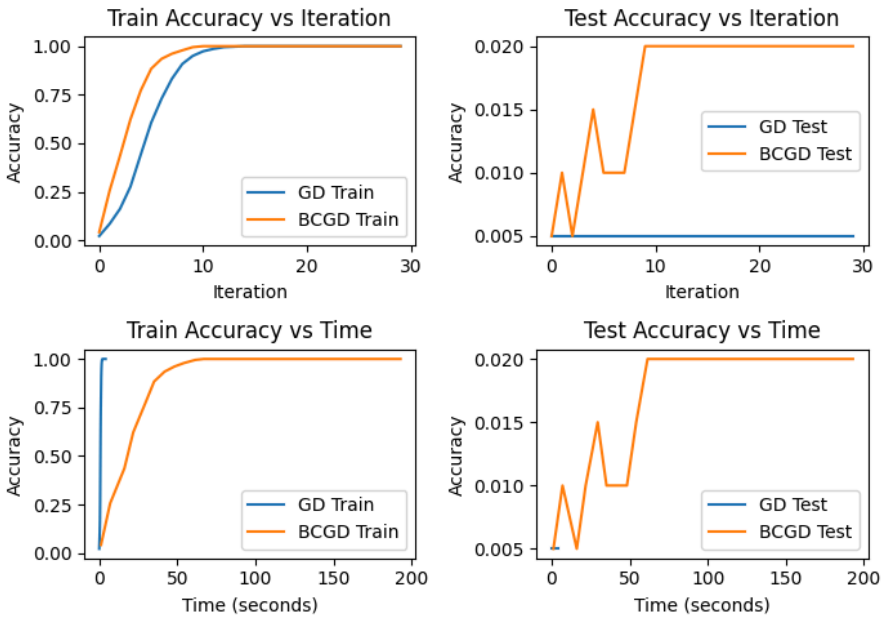
```
Iteration 0: Time: 0.54s, Train Acc: 0.0187, Test Acc: 0.0050, Train Loss: 45729801.7206, Test Loss: 2961943.4759, Updating class 41
Iteration 10: Time: 9.30s, Train Acc: 0.0200, Test Acc: 0.0050, Train Loss: 45386514.6039, Test Loss: 2960035.8381, Updating class 36
Iteration 20: Time: 14.61s, Train Acc: 0.0213, Test Acc: 0.0050, Train Loss: 45060415.0690, Test Loss: 2958200.3506, Updating class 44
Iteration 30: Time: 23.07s, Train Acc: 0.0213, Test Acc: 0.0050, Train Loss: 44745133.2088, Test Loss: 2956374.0679, Updating class 13
Iteration 40: Time: 29.16s, Train Acc: 0.0213, Test Acc: 0.0050, Train Loss: 44439399.4861, Test Loss: 2954996.5802, Updating class 13
Iteration 50: Time: 36.95s, Train Acc: 0.0250, Test Acc: 0.0050, Train Loss: 44142887.1082, Test Loss: 2953700.8741, Updating class 34
Iteration 60: Time: 42.71s, Train Acc: 0.0288, Test Acc: 0.0050, Train Loss: 43851087.6450, Test Loss: 2952375.5106, Updating class 20
Iteration 70: Time: 50.39s, Train Acc: 0.0288, Test Acc: 0.0050, Train Loss: 43564460.3442, Test Loss: 2951363.1667, Updating class 27
Iteration 80: Time: 58.73s, Train Acc: 0.0300, Test Acc: 0.0050, Train Loss: 43281687.1666, Test Loss: 2949768.3405, Updating class 40
Iteration 90: Time: 65.90s, Train Acc: 0.0300, Test Acc: 0.0050, Train Loss: 43001234.2693, Test Loss: 2948733.0689, Updating class 27
Iteration 100: Time: 71.27s, Train Acc: 0.0300, Test Acc: 0.0050, Train Loss: 42724476.9581, Test Loss: 2947914.1336, Updating class 25
Iteration 110: Time: 78.92s, Train Acc: 0.0338, Test Acc: 0.0050, Train Loss: 42450714.7696, Test Loss: 2946477.2189, Updating class 20
Iteration 120: Time: 88.46s, Train Acc: 0.0350, Test Acc: 0.0050, Train Loss: 42179959.1637, Test Loss: 2945726.8956, Updating class 21
Iteration 130: Time: 96.54s, Train Acc: 0.0362, Test Acc: 0.0050, Train Loss: 41911382.9142, Test Loss: 2944715.1283, Updating class 39
Iteration 140: Time: 102.02s, Train Acc: 0.0362, Test Acc: 0.0050, Train Loss: 41644784.6786, Test Loss: 2943492.0575, Updating class 11
Iteration 150: Time: 109.58s, Train Acc: 0.0362, Test Acc: 0.0050, Train Loss: 41379700.2486, Test Loss: 2942148.1125, Updating class 3
Iteration 160: Time: 115.29s, Train Acc: 0.0362, Test Acc: 0.0050, Train Loss: 41117628.1332, Test Loss: 2941014.7010, Updating class 41
Iteration 170: Time: 122.24s, Train Acc: 0.0362, Test Acc: 0.0050, Train Loss: 40856383.9286, Test Loss: 2939840.7869, Updating class 38
Iteration 180: Time: 128.88s, Train Acc: 0.0387, Test Acc: 0.0050, Train Loss: 40597883.8920, Test Loss: 2938737.6759, Updating class 10
Iteration 190: Time: 135.58s, Train Acc: 0.0425, Test Acc: 0.0050, Train Loss: 40341880.3264, Test Loss: 2937734.0190, Updating class 34
Iteration 200: Time: 142.32s, Train Acc: 0.0425, Test Acc: 0.0050, Train Loss: 40085479.4534, Test Loss: 2936676.5767, Updating class 8
Iteration 210: Time: 148.10s, Train Acc: 0.0425, Test Acc: 0.0050, Train Loss: 39832858.6587, Test Loss: 2935107.5723, Updating class 7
Iteration 220: Time: 155.95s, Train Acc: 0.0425, Test Acc: 0.0050, Train Loss: 39580016.9674, Test Loss: 2934287.4660, Updating class 20
Iteration 230: Time: 161.86s, Train Acc: 0.0450, Test Acc: 0.0050, Train Loss: 39330117.2538, Test Loss: 2933326.4434, Updating class 8
Iteration 240: Time: 169.51s, Train Acc: 0.0462, Test Acc: 0.0050, Train Loss: 39082121.6437, Test Loss: 2932532.0037, Updating class 21
Iteration 250: Time: 174.67s, Train Acc: 0.0475, Test Acc: 0.0050, Train Loss: 38836176.3120, Test Loss: 2931075.5358, Updating class 13
Iteration 260: Time: 182.15s, Train Acc: 0.0488, Test Acc: 0.0050, Train Loss: 38591731.6236, Test Loss: 2930075.8854, Updating class 42
Iteration 270: Time: 187.97s, Train Acc: 0.0488, Test Acc: 0.0050, Train Loss: 38348244.9384, Test Loss: 2928709.9620, Updating class 10
Iteration 280: Time: 195.76s, Train Acc: 0.0488, Test Acc: 0.0050, Train Loss: 38105722.8348, Test Loss: 2927859.7111, Updating class 24
Iteration 290: Time: 201.48s, Train Acc: 0.0488, Test Acc: 0.0050, Train Loss: 37867577.4737, Test Loss: 2926862.6897, Updating class 23
BCGD Optimization finished.
Final BCGD Train Accuracy: 0.0488
Final BCGD Test Accuracy: 0.0050
Final BCGD Train Loss: 37867577.4737
Final BCGD Test Loss: 2926862.6897
BCGD Total time: 207.38 seconds
```

```
np.random.seed(42)


initial_X_bcgd = np.random.normal(0, 1, (d, k))


optimized_X_bcgd, accuracies_train_bcgd, accuracies_test_bcgd, losses_train_bcgd, losses_test_bcgd, times_bcgd, total_time_bcgd = gauss_sout
    A_train, b_train, A_test, b_test, k, initial_X_bcgd, 1e-1, 300
)

print("BCGD Optimization finished.")
if accuracies_train_bcgd:
    print(f"Final BCGD Train Accuracy: {accuracies_train_bcgd[-1]:.4f}")
    print(f"Final BCGD Test Accuracy: {accuracies_test_bcgd[-1]:.4f}")
else:
    print("No BCGD accuracy data available.")

if losses_train_bcgd:
    print(f"Final BCGD Train Loss: {losses_train_bcgd[-1]:.4f}")
    print(f"Final BCGD Test Loss: {losses_test_bcgd[-1]:.4f}")
else:
    print("No BCGD loss data available.")

print(f"BCGD Total time: {total_time_bcgd:.2f} seconds")
```

```
Iteration 0: Time: 1.16s, Train Acc: 0.0425, Test Acc: 0.0050, Train Loss: 44325659.6896, Test Loss: 2942268.4235, Updating class 41
Iteration 10: Time: 7.00s, Train Acc: 0.2562, Test Acc: 0.0100, Train Loss: 32554878.1564, Test Loss: 2894033.2662, Updating class 20
Iteration 20: Time: 16.12s, Train Acc: 0.4363, Test Acc: 0.0050, Train Loss: 22200925.3335, Test Loss: 2883401.5371, Updating class 42
Iteration 30: Time: 21.80s, Train Acc: 0.6225, Test Acc: 0.0100, Train Loss: 13571215.6043, Test Loss: 2858590.2120, Updating class 31
Iteration 40: Time: 29.53s, Train Acc: 0.7725, Test Acc: 0.0150, Train Loss: 6340574.5155, Test Loss: 2848917.8569, Updating class 39
Iteration 50: Time: 35.15s, Train Acc: 0.8838, Test Acc: 0.0100, Train Loss: 1275673.6780, Test Loss: 2824782.7568, Updating class 19
Iteration 60: Time: 41.89s, Train Acc: 0.9350, Test Acc: 0.0100, Train Loss: 640142.2161, Test Loss: 2804029.2925, Updating class 25
Iteration 70: Time: 48.23s, Train Acc: 0.9613, Test Acc: 0.0100, Train Loss: 260751.9111, Test Loss: 2810604.2507, Updating class 23
Iteration 80: Time: 54.17s, Train Acc: 0.9788, Test Acc: 0.0150, Train Loss: 133690.9046, Test Loss: 2822670.7309, Updating class 49
Iteration 90: Time: 61.44s, Train Acc: 0.9950, Test Acc: 0.0200, Train Loss: 5889.3260, Test Loss: 2812569.2747, Updating class 28
Iteration 100: Time: 67.01s, Train Acc: 1.0000, Test Acc: 0.0200, Train Loss: 162.4041, Test Loss: 2809667.5675, Updating class 49
Iteration 110: Time: 74.45s, Train Acc: 1.0000, Test Acc: 0.0200, Train Loss: 68.3160, Test Loss: 2809526.9968, Updating class 1
Iteration 120: Time: 80.03s, Train Acc: 1.0000, Test Acc: 0.0200, Train Loss: 56.6456, Test Loss: 2809471.7437, Updating class 11
Iteration 130: Time: 87.76s, Train Acc: 1.0000, Test Acc: 0.0200, Train Loss: 49.5451, Test Loss: 2809461.0666, Updating class 1
Iteration 140: Time: 93.30s, Train Acc: 1.0000, Test Acc: 0.0200, Train Loss: 44.6307, Test Loss: 2809437.4747, Updating class 46
Iteration 150: Time: 100.84s, Train Acc: 1.0000, Test Acc: 0.0200, Train Loss: 41.0037, Test Loss: 2809440.9257, Updating class 30
Iteration 160: Time: 106.37s, Train Acc: 1.0000, Test Acc: 0.0200, Train Loss: 38.0701, Test Loss: 2809430.3337, Updating class 39
Iteration 170: Time: 113.98s, Train Acc: 1.0000, Test Acc: 0.0200, Train Loss: 35.7255, Test Loss: 2809421.8347, Updating class 11
Iteration 180: Time: 119.54s, Train Acc: 1.0000, Test Acc: 0.0200, Train Loss: 33.7357, Test Loss: 2809423.1580, Updating class 0
Iteration 190: Time: 126.91s, Train Acc: 1.0000, Test Acc: 0.0200, Train Loss: 32.0268, Test Loss: 2809415.6785, Updating class 1
Iteration 200: Time: 132.72s, Train Acc: 1.0000, Test Acc: 0.0200, Train Loss: 30.5342, Test Loss: 2809410.2189, Updating class 28
Iteration 210: Time: 139.22s, Train Acc: 1.0000, Test Acc: 0.0200, Train Loss: 29.1993, Test Loss: 2809410.5690, Updating class 6
Iteration 220: Time: 146.12s, Train Acc: 1.0000, Test Acc: 0.0200, Train Loss: 28.0258, Test Loss: 2809403.2300, Updating class 26
Iteration 230: Time: 151.90s, Train Acc: 1.0000, Test Acc: 0.0200, Train Loss: 26.9356, Test Loss: 2809400.1116, Updating class 1
Iteration 240: Time: 159.29s, Train Acc: 1.0000, Test Acc: 0.0200, Train Loss: 25.9498, Test Loss: 2809396.1654, Updating class 0
Iteration 250: Time: 165.01s, Train Acc: 1.0000, Test Acc: 0.0200, Train Loss: 25.0382, Test Loss: 2809389.8516, Updating class 24
Iteration 260: Time: 174.20s, Train Acc: 1.0000, Test Acc: 0.0200, Train Loss: 24.1970, Test Loss: 2809380.7359, Updating class 0
Iteration 270: Time: 179.75s, Train Acc: 1.0000, Test Acc: 0.0200, Train Loss: 23.4203, Test Loss: 2809378.6970, Updating class 26
Iteration 280: Time: 187.34s, Train Acc: 1.0000, Test Acc: 0.0200, Train Loss: 22.6943, Test Loss: 2809378.3033, Updating class 28
Iteration 290: Time: 192.89s, Train Acc: 1.0000, Test Acc: 0.0200, Train Loss: 22.0204, Test Loss: 2809371.3840, Updating class 17
BCGD Optimization finished.
Final BCGD Train Accuracy: 1.0000
Final BCGD Test Accuracy: 0.0200
Final BCGD Train Loss: 22.0204
Final BCGD Test Loss: 2809371.3840
BCGD Total time: 199.77 seconds
```

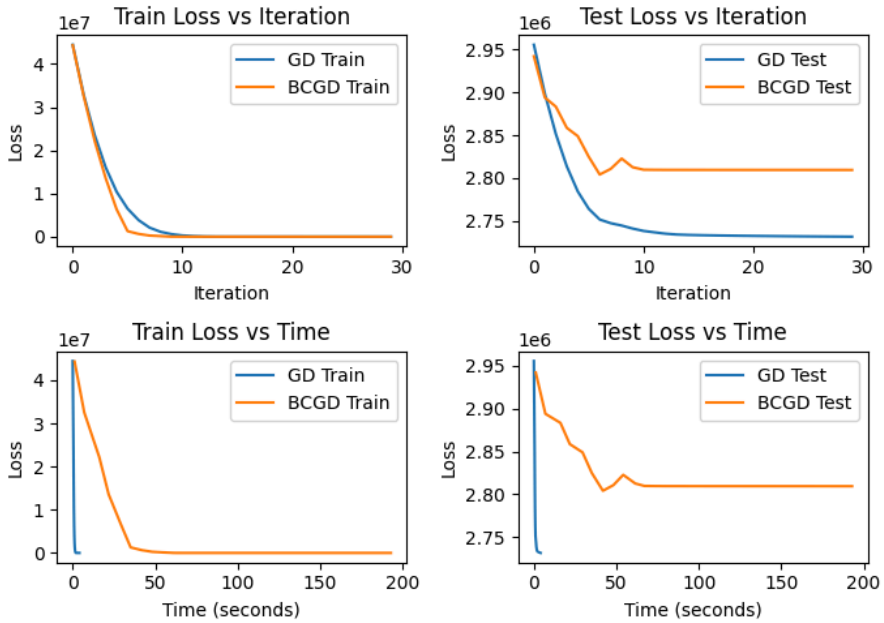
```
import matplotlib.pyplot as plt


# Plotting Accuracy vs Iteration for Train and Test
plt.figure(figsize=(7, 5))

# Accuracy vs Iteration for Train
plt.subplot(2, 2, 1)
plt.plot(accuracies_train_gd, label='GD Train')
plt.plot(accuracies_train_bcgd, label='BCGD Train')
plt.title('Train Accuracy vs Iteration')
plt.xlabel('Iteration')
plt.ylabel('Accuracy')
plt.legend()

# Accuracy vs Iteration for Test
plt.subplot(2, 2, 2)
```

```
plt.plot(accuracies_test_gd, label='GD Test')
plt.plot(accuracies_test_bcgd, label='BCGD Test')
plt.title('Test Accuracy vs Iteration')
plt.xlabel('Iteration')
plt.ylabel('Accuracy')
plt.legend()

# Plotting Accuracy vs Time for Train and Test
# Accuracy vs Time for Train
plt.subplot(2, 2, 3)
plt.plot(train_times_gd, accuracies_train_gd, label='GD Train')
plt.plot(times_bcgd, accuracies_train_bcgd, label='BCGD Train')
plt.title('Train Accuracy vs Time')
plt.xlabel('Time (seconds)')
plt.ylabel('Accuracy')
plt.legend()

# Accuracy vs Time for Test
plt.subplot(2, 2, 4)
plt.plot(train_times_gd, accuracies_test_gd, label='GD Test')
plt.plot(times_bcgd, accuracies_test_bcgd, label='BCGD Test')
plt.title('Test Accuracy vs Time')
plt.xlabel('Time (seconds)')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```




```
import matplotlib.pyplot as plt

# Plotting Loss vs Iteration for Train and Test
plt.figure(figsize=(7, 5))

# Loss vs Iteration for Train
plt.subplot(2, 2, 1)
plt.plot(losses_train_gd, label='GD Train')
plt.plot(losses_train_bcgd, label='BCGD Train')
plt.title('Train Loss vs Iteration')
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.legend()

# Loss vs Iteration for Test
plt.subplot(2, 2, 2)
plt.plot(losses_test_gd, label='GD Test')
plt.plot(losses_test_bcgd, label='BCGD Test')
plt.title('Test Loss vs Iteration')
plt.xlabel('Iteration')
```

```
plt.ylabel('Loss')
plt.legend()

# Plotting Loss vs Time for Train and Test
# Loss vs Time for Train
plt.subplot(2, 2, 3)
plt.plot(train_times_gd, losses_train_gd, label='GD Train')
plt.plot(times_bcgd, losses_train_bcgd, label='BCGD Train')
plt.title('Train Loss vs Time')
plt.xlabel('Time (seconds)')
plt.ylabel('Loss')
plt.legend()

# Loss vs Time for Test
plt.subplot(2, 2, 4)
plt.plot(train_times_gd, losses_test_gd, label='GD Test')
plt.plot(times_bcgd, losses_test_bcgd, label='BCGD Test')
plt.title('Test Loss vs Time')
plt.xlabel('Time (seconds)')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```



Fashion-MNIST data:

```
import torchvision as tv
```

```
# Define transformation pipeline
transform = tv.transforms.Compose([
    tv.transforms.ToTensor(),  # Converts to [0, 1] range
    tv.transforms.Lambda(lambda x: x.view(-1))  # Flattens the image
])

# Load Fashion-MNIST data
FMNIST_train = tv.datasets.FashionMNIST('data/', train=True, download=True, transform=transform)
FMNIST_test = tv.datasets.FashionMNIST('data/', train=False, download=True, transform=transform)
```

```
Xtrain = FMNIST_train.data.numpy().reshape((FMNIST_train.data.shape[0], -1)).astype(np.float32) / 255.0
Xtest = FMNIST_test.data.numpy().reshape((FMNIST_test.data.shape[0], -1)).astype(np.float32) / 255.0
ytrain = FMNIST_train.targets.numpy()
ytest = FMNIST_test.targets.numpy()
```

```
print("Shape of Xtrain:", Xtrain.shape)
print("Shape of Xtest:", Xtest.shape)
```

```
Shape of Xtrain: (60000, 784)
Shape of Xtest: (10000, 784)
```

```
# Using smaller subset for training and testing to save memory
num_samples_train = 10000
num_samples_test = 2000

Xtrain = Xtrain[:num_samples_train]
ytrain = ytrain[:num_samples_train]
Xtest = Xtest[:num_samples_test]
ytest = ytest[:num_samples_test]
```

```
def gradient_descent(A_train, b_train, A_test, b_test, k, initial_X_gd, num_iterations, step_size_strategy, initial_learning_rate):
    X = initial_X_gd
    m, d = A_train.shape
    accuracies_train = []
    accuracies_test = []
    losses_train = []
    losses_test = []
    times = []
    start_time = time.time()

    # Get the functions for calculating the objective and gradient for training and test data
    f_train, grad_f_train = generate_logistic_objective(A_train, b_train, k)

    f_test, _ = generate_logistic_objective(A_test, b_test, k)

    for i in range(num_iterations):
        # Use the provided function to calculate gradient
        gradient = grad_f_train(X)

        alpha = initial_learning_rate

        X -= alpha * gradient

        if i % 10 == 0 or i == num_iterations:
            current_time = time.time() - start_time
            times.append(current_time)

            accuracy_train = compute_accuracy(X, A_train, b_train)
            accuracy_test = compute_accuracy(X, A_test, b_test)
            accuracies_train.append(accuracy_train)
            accuracies_test.append(accuracy_test)

            loss_train = f_train(X)
            loss_test = f_test(X)
            losses_train.append(loss_train)
            losses_test.append(loss_test)

            print(f"Iteration {i}: Time: {current_time:.2f}s, Train Acc: {accuracy_train:.4f}, Test Acc: {accuracy_test:.4f}, Train Loss: {l

    total_time = time.time() - start_time
    return X, accuracies_train, accuracies_test, losses_train, losses_test, times, total_time
```

```
k = 10  # Number of classes
d = Xtrain.shape[1]  # Number of features (784 for MNIST)
np.random.seed(42)

# Initialize weights for logistic regression
initial_X_gd = np.random.normal(0, 1, (d, k))


overall_start_time = time.time()

X_train_opt_gd, accuracies_train_gd, accuracies_test_gd, losses_train_gd, losses_test_gd, train_times_gd, total_train_time_gd = gradient_desce
    Xtrain, ytrain, Xtest, ytest, k, initial_X_gd, 300, 'fixed', 1e-3
)


total_time = time.time() - overall_start_time
```

```
print("Training Complete")
print(f"Final Train Accuracy: {accuracies_train_gd[-1]:.4f}")
print(f"Final Test Accuracy: {accuracies_test_gd[-1]:.4f}")
print(f"Final Train Loss: {losses_train_gd[-1]:.4f}")
print(f"Final Test Loss: {losses_test_gd[-1]:.4f}")
print(f"Total Training Time: {total_train_time_gd:.2f} seconds")
print(f"Total Time (Training + Evaluation): {total_time:.2f} seconds")
```

```
<ipython-input-36-86617081ecf2>:17: RuntimeWarning: divide by zero encountered in log
  loss = -np.sum(np.log(correct_class_scores) - log_prob)
Iteration 0: Time: 0.21s, Train Acc: 0.2566, Test Acc: 0.2500, Train Loss: inf, Test Loss: inf
Iteration 10: Time: 2.07s, Train Acc: 0.3262, Test Acc: 0.3190, Train Loss: 12910697605.1743, Test Loss: 544592874.1184
Iteration 20: Time: 3.74s, Train Acc: 0.6495, Test Acc: 0.6390, Train Loss: 5958641098.8755, Test Loss: 252463648.3764
Iteration 30: Time: 5.42s, Train Acc: 0.6551, Test Acc: 0.6265, Train Loss: 2585938075.0593, Test Loss: 111315220.7542
Iteration 40: Time: 7.12s, Train Acc: 0.7080, Test Acc: 0.6940, Train Loss: 1814586521.8211, Test Loss: 77944173.6520
Iteration 50: Time: 9.87s, Train Acc: 0.6641, Test Acc: 0.6325, Train Loss: 2601369309.4350, Test Loss: 111771584.5538
Iteration 60: Time: 12.90s, Train Acc: 0.6762, Test Acc: 0.6670, Train Loss: 5051896810.8713, Test Loss: 202116995.7370
Iteration 70: Time: 15.37s, Train Acc: 0.7273, Test Acc: 0.7220, Train Loss: 2293259067.8260, Test Loss: 97860780.4142
Iteration 80: Time: 17.09s, Train Acc: 0.7457, Test Acc: 0.7375, Train Loss: 1562851950.0851, Test Loss: 61595670.0395
Iteration 90: Time: 18.82s, Train Acc: 0.6970, Test Acc: 0.6815, Train Loss: 2925782981.4634, Test Loss: 116646437.3360
Iteration 100: Time: 20.50s, Train Acc: 0.5659, Test Acc: 0.5495, Train Loss: 3050248142.5185, Test Loss: 135940543.2313
Iteration 110: Time: 22.09s, Train Acc: 0.7535, Test Acc: 0.7455, Train Loss: 1629208971.9158, Test Loss: 62264794.3758
Iteration 120: Time: 23.66s, Train Acc: 0.7628, Test Acc: 0.7585, Train Loss: 2853951671.8383, Test Loss: 107212943.7980
Iteration 130: Time: 25.53s, Train Acc: 0.7130, Test Acc: 0.6820, Train Loss: 3338851914.7430, Test Loss: 142337446.2678
Iteration 140: Time: 28.08s, Train Acc: 0.7724, Test Acc: 0.7630, Train Loss: 1120014662.3843, Test Loss: 46529991.0054
Iteration 150: Time: 29.96s, Train Acc: 0.7533, Test Acc: 0.7300, Train Loss: 2116422673.0157, Test Loss: 89209653.5398
Iteration 160: Time: 31.63s, Train Acc: 0.7122, Test Acc: 0.7050, Train Loss: 1527286776.0818, Test Loss: 65386889.5464
Iteration 170: Time: 33.29s, Train Acc: 0.7121, Test Acc: 0.7015, Train Loss: 2696108612.6225, Test Loss: 104028982.7593
Iteration 180: Time: 34.94s, Train Acc: 0.7387, Test Acc: 0.7110, Train Loss: 1469231667.6964, Test Loss: 64114224.0493
Iteration 190: Time: 36.55s, Train Acc: 0.7660, Test Acc: 0.7605, Train Loss: 3352920121.2732, Test Loss: 125118663.4122
Iteration 200: Time: 38.16s, Train Acc: 0.7605, Test Acc: 0.7340, Train Loss: 1611948828.7825, Test Loss: 70177850.4597
Iteration 210: Time: 40.52s, Train Acc: 0.7560, Test Acc: 0.7450, Train Loss: 3278242242.1614, Test Loss: 122114616.7057
Iteration 220: Time: 43.28s, Train Acc: 0.7001, Test Acc: 0.6780, Train Loss: 2720036877.1290, Test Loss: 117568997.6799
Iteration 230: Time: 45.03s, Train Acc: 0.7708, Test Acc: 0.7595, Train Loss: 1686061028.3079, Test Loss: 69194810.2649
Iteration 240: Time: 46.71s, Train Acc: 0.7594, Test Acc: 0.7430, Train Loss: 1663314973.4893, Test Loss: 67670532.8427
Iteration 250: Time: 48.38s, Train Acc: 0.7100, Test Acc: 0.7095, Train Loss: 1549387289.3199, Test Loss: 63478565.7604
Iteration 260: Time: 50.02s, Train Acc: 0.7528, Test Acc: 0.7450, Train Loss: 1841112570.1909, Test Loss: 77347949.0606
Iteration 270: Time: 51.55s, Train Acc: 0.7949, Test Acc: 0.7880, Train Loss: 2035690259.9686, Test Loss: 76295893.0198
Iteration 280: Time: 53.09s, Train Acc: 0.6558, Test Acc: 0.6420, Train Loss: 2178048568.4860, Test Loss: 93982199.6097
Iteration 290: Time: 55.58s, Train Acc: 0.8154, Test Acc: 0.7980, Train Loss: 890397580.4706, Test Loss: 36872536.8757
Training Complete
Final Train Accuracy: 0.8154
Final Test Accuracy: 0.7980
Final Train Loss: 890397580.4706
Final Test Loss: 36872536.8757
Total Training Time: 57.89 seconds
Total Time (Training + Evaluation): 57.89 seconds
```

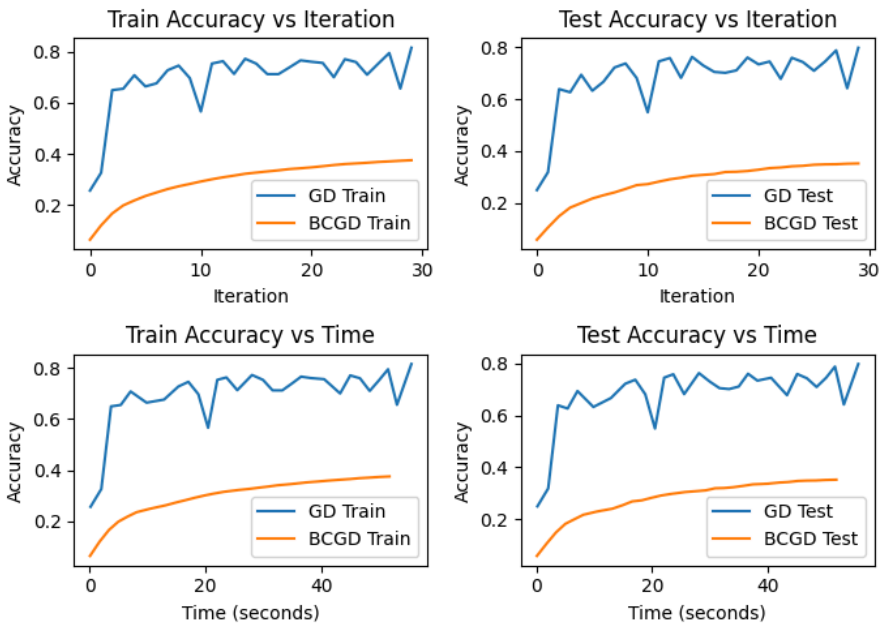
```
def gauss_southwell_bcgd(A_train, b_train, A_test, b_test, k, initial_X_bcgd, learning_rate, num_iterations):
    X = initial_X_bcgd
    m, d = A_train.shape
    accuracies_train = []
    accuracies_test = []
    losses_train = []
    losses_test = []
    times = []
    start_time = time.time()

    f_train, grad_f_train = generate_logistic_objective(A_train, b_train, k)
    f_test, _ = generate_logistic_objective(A_test, b_test, k)

    for i in range(num_iterations):
        # Calculate gradients for each class
        gradient = grad_f_train(X)
        gradient_norms = [np.linalg.norm(gradient[:, j]) for j in range(k)]

        # Find the class with the largest gradient norm
        idx = np.argmax(gradient_norms)

        # Gradient clipping for the selected block
        block_gradient = gradient[:, idx]
        gradient_norm = np.linalg.norm(block_gradient)
        max_norm = 10.0  # Max norm for gradient clipping
        if gradient_norm > max_norm:
            block_gradient = (block_gradient / gradient_norm) * max_norm

        # Update the block corresponding to the selected class
        X[:, idx] -= learning_rate * block_gradient

        if i % 10 == 0:
            current_time = time.time() - start_time
            accuracy_train = compute_accuracy(X, A_train, b_train)
            accuracy_test = compute_accuracy(X, A_test, b_test)
            loss_train = f_train(X)
            loss_test = f_test(X)

            accuracies_train.append(accuracy_train)
            accuracies_test.append(accuracy_test)
            losses_train.append(loss_train)
            losses_test.append(loss_test)
            times.append(current_time)

            print(f"Iteration {i}: Time: {current_time:.2f}s, Train Acc: {accuracy_train:.4f}, Test Acc: {accuracy_test:.4f}, Train Loss: {l

    total_time = time.time() - start_time
    return X, accuracies_train, accuracies_test, losses_train, losses_test, times, total_time
```

```
np.random.seed(42)
initial_X_bcgd = np.random.normal(0, 1, (d, k))


optimized_X_bcgd, accuracies_train_bcgd, accuracies_test_bcgd, losses_train_bcgd, losses_test_bcgd, times_bcgd, total_time_bcgd = gauss_southu
    Xtrain, ytrain, Xtest, ytest, k, initial_X_bcgd, 1e-1, 300
)


print("Optimization finished.")
if accuracies_train_bcgd:
    print(f"Final Train Accuracy: {accuracies_train_bcgd[-1]:.4f}")
    print(f"Final Test Accuracy: {accuracies_test_bcgd[-1]:.4f}")
else:
    print("No accuracy data available.")

if losses_train_bcgd:
    print(f"Final Train Loss: {losses_train_bcgd[-1]:.4f}")
    print(f"Final Test Loss: {losses_test_bcgd[-1]:.4f}")
else:
    print("No loss data available.")

print(f"Total time: {total_time_bcgd:.2f} seconds")
```

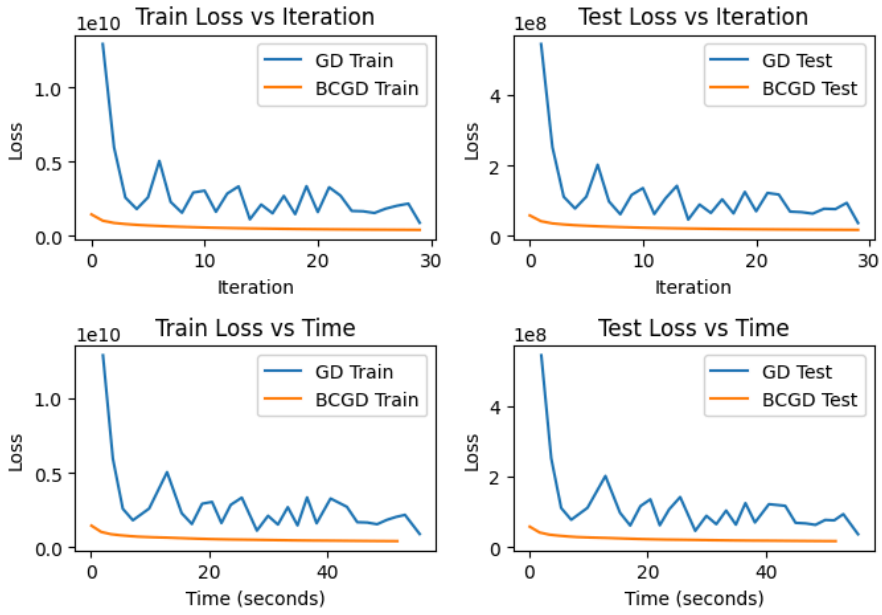
```
Iteration 0: Time: 0.12s, Train Acc: 0.0644, Test Acc: 0.0590, Train Loss: 1453728423.4872, Test Loss: 58385946.1482, Updating class 0
Iteration 10: Time: 1.78s, Train Acc: 0.1204, Test Acc: 0.1060, Train Loss: 1030742531.0184, Test Loss: 41859149.7592, Updating class 0
Iteration 20: Time: 3.41s, Train Acc: 0.1668, Test Acc: 0.1495, Train Loss: 879104510.3598, Test Loss: 35707360.6711, Updating class 0
Iteration 30: Time: 5.02s, Train Acc: 0.1986, Test Acc: 0.1825, Train Loss: 806446618.1057, Test Loss: 32805857.6967, Updating class 0
Iteration 40: Time: 6.57s, Train Acc: 0.2179, Test Acc: 0.2000, Train Loss: 751336550.6340, Test Loss: 30592239.1581, Updating class 0
Iteration 50: Time: 8.14s, Train Acc: 0.2353, Test Acc: 0.2175, Train Loss: 708692172.5004, Test Loss: 28896677.1162, Updating class 0
Iteration 60: Time: 10.50s, Train Acc: 0.2491, Test Acc: 0.2300, Train Loss: 673636919.6300, Test Loss: 27508262.3265, Updating class 0
Iteration 70: Time: 13.18s, Train Acc: 0.2625, Test Acc: 0.2410, Train Loss: 643823704.5949, Test Loss: 26329716.6426, Updating class 0
Iteration 80: Time: 14.94s, Train Acc: 0.2733, Test Acc: 0.2545, Train Loss: 617927049.4686, Test Loss: 25308721.8933, Updating class 0
Iteration 90: Time: 16.58s, Train Acc: 0.2825, Test Acc: 0.2690, Train Loss: 595141924.2488, Test Loss: 24413116.3121, Updating class 0
Iteration 100: Time: 18.21s, Train Acc: 0.2921, Test Acc: 0.2730, Train Loss: 573597989.7295, Test Loss: 23564933.7827, Updating class 0
Iteration 110: Time: 19.85s, Train Acc: 0.3008, Test Acc: 0.2825, Train Loss: 555962069.7781, Test Loss: 22876297.8997, Updating class 0
Iteration 120: Time: 21.53s, Train Acc: 0.3085, Test Acc: 0.2915, Train Loss: 540510684.7788, Test Loss: 22275137.2833, Updating class 0
Iteration 130: Time: 23.20s, Train Acc: 0.3152, Test Acc: 0.2975, Train Loss: 526699516.2851, Test Loss: 21739221.7729, Updating class 0
Iteration 140: Time: 25.71s, Train Acc: 0.3223, Test Acc: 0.3050, Train Loss: 513860009.7635, Test Loss: 21238844.4295, Updating class 0
Iteration 150: Time: 27.72s, Train Acc: 0.3270, Test Acc: 0.3085, Train Loss: 502382649.2786, Test Loss: 20794332.5241, Updating class 0
Iteration 160: Time: 29.26s, Train Acc: 0.3315, Test Acc: 0.3115, Train Loss: 492026877.9200, Test Loss: 20395080.2388, Updating class 0
Iteration 170: Time: 30.87s, Train Acc: 0.3358, Test Acc: 0.3195, Train Loss: 482628586.9882, Test Loss: 20033895.1661, Updating class 0
Iteration 180: Time: 32.45s, Train Acc: 0.3408, Test Acc: 0.3205, Train Loss: 474059936.0215, Test Loss: 19705266.3607, Updating class 0
Iteration 190: Time: 33.99s, Train Acc: 0.3438, Test Acc: 0.3235, Train Loss: 466215295.7148, Test Loss: 19404731.9397, Updating class 0
Iteration 200: Time: 35.64s, Train Acc: 0.3475, Test Acc: 0.3285, Train Loss: 459005140.8774, Test Loss: 19128551.7276, Updating class 0
Iteration 210: Time: 37.40s, Train Acc: 0.3519, Test Acc: 0.3345, Train Loss: 452352848.0715, Test Loss: 18873544.6919, Updating class 0
Iteration 220: Time: 39.91s, Train Acc: 0.3565, Test Acc: 0.3370, Train Loss: 446192704.9328, Test Loss: 18637010.2042, Updating class 0
Iteration 230: Time: 41.93s, Train Acc: 0.3603, Test Acc: 0.3415, Train Loss: 440468411.4686, Test Loss: 18416675.1040, Updating class 0
Iteration 240: Time: 43.49s, Train Acc: 0.3629, Test Acc: 0.3435, Train Loss: 435131691.2799, Test Loss: 18210637.1149, Updating class 0
Iteration 250: Time: 45.07s, Train Acc: 0.3654, Test Acc: 0.3475, Train Loss: 430140950.3921, Test Loss: 18017300.5161, Updating class 0
Iteration 260: Time: 46.63s, Train Acc: 0.3686, Test Acc: 0.3490, Train Loss: 425460101.8213, Test Loss: 17835313.8313, Updating class 0
Iteration 270: Time: 48.24s, Train Acc: 0.3707, Test Acc: 0.3495, Train Loss: 421057669.6487, Test Loss: 17663519.8154, Updating class 0
Iteration 280: Time: 49.91s, Train Acc: 0.3732, Test Acc: 0.3515, Train Loss: 416906157.6812, Test Loss: 17500920.2716, Updating class 0
Iteration 290: Time: 51.79s, Train Acc: 0.3753, Test Acc: 0.3525, Train Loss: 412981566.1221, Test Loss: 17346651.2992, Updating class 0
Optimization finished.
Final Train Accuracy: 0.3753
Final Test Accuracy: 0.3525
Final Train Loss: 412981566.1221
Final Test Loss: 17346651.2992
Total time: 54.10 seconds
```

```
import matplotlib.pyplot as plt

# Plotting Accuracy vs Iteration for Train and Test
plt.figure(figsize=(7, 5))

# Accuracy vs Iteration for Train
plt.subplot(2, 2, 1)
plt.plot(accuracies_train_gd, label='GD Train')
plt.plot(accuracies_train_bcgd, label='BCGD Train')
plt.title('Train Accuracy vs Iteration')
plt.xlabel('Iteration')
plt.ylabel('Accuracy')
plt.legend()

# Accuracy vs Iteration for Test
plt.subplot(2, 2, 2)
plt.plot(accuracies_test_gd, label='GD Test')
plt.plot(accuracies_test_bcgd, label='BCGD Test')
plt.title('Test Accuracy vs Iteration')
plt.xlabel('Iteration')
plt.ylabel('Accuracy')
plt.legend()

# Plotting Accuracy vs Time for Train and Test
# Accuracy vs Time for Train
plt.subplot(2, 2, 3)
plt.plot(train_times_gd, accuracies_train_gd, label='GD Train')
plt.plot(times_bcgd, accuracies_train_bcgd, label='BCGD Train')
plt.title('Train Accuracy vs Time')
plt.xlabel('Time (seconds)')
plt.ylabel('Accuracy')
plt.legend()

# Accuracy vs Time for Test
plt.subplot(2, 2, 4)
plt.plot(train_times_gd, accuracies_test_gd, label='GD Test')
plt.plot(times_bcgd, accuracies_test_bcgd, label='BCGD Test')
plt.title('Test Accuracy vs Time')
plt.xlabel('Time (seconds)')
plt.ylabel('Accuracy')
plt.legend()
```

```
plt.tight_layout()
plt.show()
```




```
import matplotlib.pyplot as plt

# Plotting Loss vs Iteration for Train and Test
plt.figure(figsize=(7, 5))

# Loss vs Iteration for Train
plt.subplot(2, 2, 1)
plt.plot(losses_train_gd, label='GD Train')
plt.plot(losses_train_bcgd, label='BCGD Train')
plt.title('Train Loss vs Iteration')
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.legend()

# Loss vs Iteration for Test
plt.subplot(2, 2, 2)
plt.plot(losses_test_gd, label='GD Test')
plt.plot(losses_test_bcgd, label='BCGD Test')
plt.title('Test Loss vs Iteration')
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.legend()

# Plotting Loss vs Time for Train and Test
# Loss vs Time for Train
plt.subplot(2, 2, 3)
plt.plot(train_times_gd, losses_train_gd, label='GD Train')
plt.plot(times_bcgd, losses_train_bcgd, label='BCGD Train')
plt.title('Train Loss vs Time')
plt.xlabel('Time (seconds)')
plt.ylabel('Loss')
plt.legend()

# Loss vs Time for Test
plt.subplot(2, 2, 4)
plt.plot(train_times_gd, losses_test_gd, label='GD Test')
plt.plot(times_bcgd, losses_test_bcgd, label='BCGD Test')
plt.title('Test Loss vs Time')
plt.xlabel('Time (seconds)')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```



Iris data:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

data = load_iris()
X = data.data
y = data.target

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

```
k = 3  # Number of classes in the Iris dataset
d = X_train.shape[1]  # Number of features (4 for Iris)
np.random.seed(42)

initial_X_gd = np.random.normal(0, 1, (d, k))


overall_start_time = time.time()


X_train_opt_gd, accuracies_train_gd, accuracies_test_gd, losses_train_gd, losses_test_gd, train_times_gd, total_train_time_gd = gradient_desce
    X_train, y_train, X_test, y_test, k, initial_X_gd, 300, 'fixed', 1e-3
)


total_time = time.time() - overall_start_time


print("Training Complete")
print(f"Final Train Accuracy: {accuracies_train_gd[-1]:.4f}")
print(f"Final Test Accuracy: {accuracies_test_gd[-1]:.4f}")
print(f"Final Train Loss: {losses_train_gd[-1]:.4f}")
print(f"Final Test Loss: {losses_test_gd[-1]:.4f}")
print(f"Total Training Time: {total_train_time_gd:.2f} seconds")
print(f"Total Time (Training + Evaluation): {total_time:.2f} seconds")
```

```
Iteration 0: Time: 0.00s, Train Acc: 0.2333, Test Acc: 0.2000, Train Loss: 29244.4613, Test Loss: 2012.7585
Iteration 10: Time: 0.00s, Train Acc: 0.6167, Test Acc: 0.6667, Train Loss: 11703.1318, Test Loss: 725.6659
Iteration 20: Time: 0.00s, Train Acc: 0.7500, Test Acc: 0.8667, Train Loss: 8349.0710, Test Loss: 468.9165
Iteration 30: Time: 0.01s, Train Acc: 0.7750, Test Acc: 0.9000, Train Loss: 7274.8774, Test Loss: 390.1752
Iteration 40: Time: 0.01s, Train Acc: 0.8083, Test Acc: 0.9000, Train Loss: 6755.0224, Test Loss: 353.8965
```

```
Iteration 50: Time: 0.01s, Train Acc: 0.8083, Test Acc: 0.9000, Train Loss: 6437.5076, Test Loss: 332.6239
Iteration 60: Time: 0.01s, Train Acc: 0.8167, Test Acc: 0.9000, Train Loss: 6216.4088, Test Loss: 318.3004
Iteration 70: Time: 0.01s, Train Acc: 0.8167, Test Acc: 0.9000, Train Loss: 6049.7047, Test Loss: 307.7923
Iteration 80: Time: 0.01s, Train Acc: 0.8167, Test Acc: 0.9000, Train Loss: 5917.3769, Test Loss: 299.6314
Iteration 90: Time: 0.02s, Train Acc: 0.8167, Test Acc: 0.9000, Train Loss: 5808.5911, Test Loss: 293.0357
Iteration 100: Time: 0.02s, Train Acc: 0.8083, Test Acc: 0.9000, Train Loss: 5716.9047, Test Loss: 287.5480
Iteration 110: Time: 0.02s, Train Acc: 0.8083, Test Acc: 0.9000, Train Loss: 5638.1977, Test Loss: 282.8812
Iteration 120: Time: 0.02s, Train Acc: 0.8167, Test Acc: 0.9000, Train Loss: 5569.6807, Test Loss: 278.8449
Iteration 130: Time: 0.02s, Train Acc: 0.8250, Test Acc: 0.9000, Train Loss: 5509.3759, Test Loss: 275.3068
Iteration 140: Time: 0.02s, Train Acc: 0.8167, Test Acc: 0.9000, Train Loss: 5455.8276, Test Loss: 272.1717
Iteration 150: Time: 0.02s, Train Acc: 0.8167, Test Acc: 0.9000, Train Loss: 5407.9311, Test Loss: 269.3687
Iteration 160: Time: 0.02s, Train Acc: 0.8167, Test Acc: 0.9000, Train Loss: 5364.8266, Test Loss: 266.8441
Iteration 170: Time: 0.03s, Train Acc: 0.8167, Test Acc: 0.9000, Train Loss: 5325.8305, Test Loss: 264.5558
Iteration 180: Time: 0.03s, Train Acc: 0.8167, Test Acc: 0.9000, Train Loss: 5290.3896, Test Loss: 262.4704
Iteration 190: Time: 0.03s, Train Acc: 0.8167, Test Acc: 0.9000, Train Loss: 5258.0499, Test Loss: 260.5612
Iteration 200: Time: 0.03s, Train Acc: 0.8167, Test Acc: 0.9000, Train Loss: 5228.4334, Test Loss: 258.8060
Iteration 210: Time: 0.03s, Train Acc: 0.8250, Test Acc: 0.9000, Train Loss: 5201.2224, Test Loss: 257.1866
Iteration 220: Time: 0.03s, Train Acc: 0.8250, Test Acc: 0.9000, Train Loss: 5176.1472, Test Loss: 255.6877
Iteration 230: Time: 0.03s, Train Acc: 0.8250, Test Acc: 0.9000, Train Loss: 5152.9770, Test Loss: 254.2963
Iteration 240: Time: 0.04s, Train Acc: 0.8167, Test Acc: 0.9000, Train Loss: 5131.5127, Test Loss: 253.0012
Iteration 250: Time: 0.04s, Train Acc: 0.8250, Test Acc: 0.9000, Train Loss: 5111.5818, Test Loss: 251.7930
Iteration 260: Time: 0.04s, Train Acc: 0.8333, Test Acc: 0.9000, Train Loss: 5093.0335, Test Loss: 250.6634
Iteration 270: Time: 0.04s, Train Acc: 0.8333, Test Acc: 0.9000, Train Loss: 5075.7360, Test Loss: 249.6051
Iteration 280: Time: 0.04s, Train Acc: 0.8333, Test Acc: 0.9000, Train Loss: 5059.5729, Test Loss: 248.6117
Iteration 290: Time: 0.05s, Train Acc: 0.8333, Test Acc: 0.9333, Train Loss: 5044.4413, Test Loss: 247.6776
Training Complete
Final Train Accuracy: 0.8333
Final Test Accuracy: 0.9333
Final Train Loss: 5044.4413
Final Test Loss: 247.6776
Total Training Time: 0.05 seconds
Total Time (Training + Evaluation): 0.05 seconds
```

```
def gauss_southwell_bcgd(A_train, b_train, A_test, b_test, k, initial_X_bcgd, learning_rate, num_iterations):
    X = initial_X_bcgd
    m, d = A_train.shape
    accuracies_train = []
    accuracies_test = []
    losses_train = []
    losses_test = []
    times = []
    start_time = time.time()

    # Get the functions for calculating the objective and gradient for training and test data
    f_train, grad_f_train = generate_logistic_objective(A_train, b_train, k)

    f_test, _ = generate_logistic_objective(A_test, b_test, k)

    for i in range(num_iterations):
        # Calculate gradients for each class and find the class with the largest gradient norm
        gradient_norms = [np.linalg.norm(grad_f_train(X)[:, j]) for j in range(k)]
        idx = np.argmax(gradient_norms)
        block_gradient = grad_f_train(X)[:, idx]
        X[:, idx] -= learning_rate * block_gradient

        if i % 10 == 0:
            current_time = time.time() - start_time
            times.append(current_time)

            accuracy_train = compute_accuracy(X, A_train, b_train)
            accuracy_test = compute_accuracy(X, A_test, b_test)
            accuracies_train.append(accuracy_train)
            accuracies_test.append(accuracy_test)

            loss_train = f_train(X)
            loss_test = f_test(X)
            losses_train.append(loss_train)
            losses_test.append(loss_test)

            print(f"Iteration {i}: Time: {current_time:.2f}s, Train Acc: {accuracy_train:.4f}, Test Acc: {accuracy_test:.4f}, Train Loss: {l

    total_time = time.time() - start_time
    return X, accuracies_train, accuracies_test, losses_train, losses_test, times, total_time
```

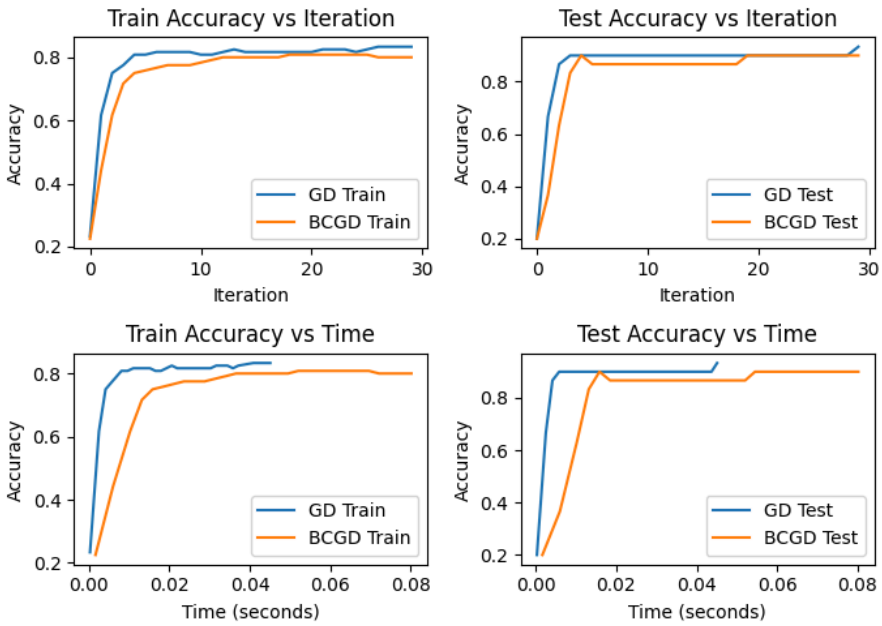
```
np.random.seed(42)
initial_X_bcgd = np.random.normal(0, 1, (d, k))
```

```
optimized_X_bcgd, accuracies_train_bcgd, accuracies_test_bcgd, losses_train_bcgd, losses_test_bcgd, times_bcgd, total_time_bcgd = gauss_southw
    X_train, y_train, X_test, y_test, k, initial_X_bcgd, 1e-3, 300
)


print("Optimization finished.")
if accuracies_train_bcgd:
    print(f"Final Train Accuracy: {accuracies_train_bcgd[-1]:.4f}")
    print(f"Final Test Accuracy: {accuracies_test_bcgd[-1]:.4f}")
else:
    print("No accuracy data available.")

if losses_train_bcgd:
    print(f"Final Train Loss: {losses_train_bcgd[-1]:.4f}")
    print(f"Final Test Loss: {losses_test_bcgd[-1]:.4f}")
else:
    print("No loss data available.")

print(f"Total time: {total_time_bcgd:.2f} seconds")
```

```
Iteration 0: Time: 0.00s, Train Acc: 0.2250, Test Acc: 0.2000, Train Loss: 30721.6661, Test Loss: 2121.0065, Updating class 0
Iteration 10: Time: 0.01s, Train Acc: 0.4417, Test Acc: 0.3667, Train Loss: 16558.1717, Test Loss: 1111.8706, Updating class 0
Iteration 20: Time: 0.01s, Train Acc: 0.6167, Test Acc: 0.6333, Train Loss: 11723.0385, Test Loss: 735.2919, Updating class 2
Iteration 30: Time: 0.01s, Train Acc: 0.7167, Test Acc: 0.8333, Train Loss: 9585.1906, Test Loss: 566.4992, Updating class 2
Iteration 40: Time: 0.02s, Train Acc: 0.7500, Test Acc: 0.9000, Train Loss: 8484.5125, Test Loss: 481.2600, Updating class 2
Iteration 50: Time: 0.02s, Train Acc: 0.7583, Test Acc: 0.8667, Train Loss: 7844.8854, Test Loss: 433.0970, Updating class 2
Iteration 60: Time: 0.02s, Train Acc: 0.7667, Test Acc: 0.8667, Train Loss: 7433.7616, Test Loss: 403.1755, Updating class 0
Iteration 70: Time: 0.02s, Train Acc: 0.7750, Test Acc: 0.8667, Train Loss: 7148.5185, Test Loss: 382.6977, Updating class 2
Iteration 80: Time: 0.03s, Train Acc: 0.7750, Test Acc: 0.8667, Train Loss: 6938.1687, Test Loss: 367.9009, Updating class 2
Iteration 90: Time: 0.03s, Train Acc: 0.7750, Test Acc: 0.8667, Train Loss: 6775.4456, Test Loss: 356.7457, Updating class 0
Iteration 100: Time: 0.03s, Train Acc: 0.7833, Test Acc: 0.8667, Train Loss: 6644.9075, Test Loss: 347.8457, Updating class 0
Iteration 110: Time: 0.03s, Train Acc: 0.7917, Test Acc: 0.8667, Train Loss: 6537.0283, Test Loss: 340.5928, Updating class 0
Iteration 120: Time: 0.04s, Train Acc: 0.8000, Test Acc: 0.8667, Train Loss: 6445.5825, Test Loss: 334.4729, Updating class 2
Iteration 130: Time: 0.04s, Train Acc: 0.8000, Test Acc: 0.8667, Train Loss: 6366.5186, Test Loss: 329.2896, Updating class 0
Iteration 140: Time: 0.04s, Train Acc: 0.8000, Test Acc: 0.8667, Train Loss: 6296.8957, Test Loss: 324.7402, Updating class 0
Iteration 150: Time: 0.04s, Train Acc: 0.8000, Test Acc: 0.8667, Train Loss: 6234.6951, Test Loss: 320.7207, Updating class 0
Iteration 160: Time: 0.05s, Train Acc: 0.8000, Test Acc: 0.8667, Train Loss: 6178.4169, Test Loss: 317.1215, Updating class 0
Iteration 170: Time: 0.05s, Train Acc: 0.8000, Test Acc: 0.8667, Train Loss: 6126.5069, Test Loss: 313.7441, Updating class 1
Iteration 180: Time: 0.05s, Train Acc: 0.8083, Test Acc: 0.8667, Train Loss: 6077.6877, Test Loss: 310.6280, Updating class 2
Iteration 190: Time: 0.05s, Train Acc: 0.8083, Test Acc: 0.9000, Train Loss: 6031.6398, Test Loss: 307.6961, Updating class 2
Iteration 200: Time: 0.06s, Train Acc: 0.8083, Test Acc: 0.9000, Train Loss: 5988.1434, Test Loss: 304.9490, Updating class 1
Iteration 210: Time: 0.06s, Train Acc: 0.8083, Test Acc: 0.9000, Train Loss: 5946.9621, Test Loss: 302.3937, Updating class 0
Iteration 220: Time: 0.06s, Train Acc: 0.8083, Test Acc: 0.9000, Train Loss: 5907.9314, Test Loss: 299.9802, Updating class 2
Iteration 230: Time: 0.06s, Train Acc: 0.8083, Test Acc: 0.9000, Train Loss: 5870.8936, Test Loss: 297.7096, Updating class 2
Iteration 240: Time: 0.07s, Train Acc: 0.8083, Test Acc: 0.9000, Train Loss: 5835.6918, Test Loss: 295.5721, Updating class 1
Iteration 250: Time: 0.07s, Train Acc: 0.8083, Test Acc: 0.9000, Train Loss: 5802.2007, Test Loss: 293.5556, Updating class 2
Iteration 260: Time: 0.07s, Train Acc: 0.8000, Test Acc: 0.9000, Train Loss: 5770.2904, Test Loss: 291.6395, Updating class 2
Iteration 270: Time: 0.08s, Train Acc: 0.8000, Test Acc: 0.9000, Train Loss: 5739.8783, Test Loss: 289.8255, Updating class 1
Iteration 280: Time: 0.08s, Train Acc: 0.8000, Test Acc: 0.9000, Train Loss: 5710.8451, Test Loss: 288.1052, Updating class 2
Iteration 290: Time: 0.08s, Train Acc: 0.8000, Test Acc: 0.9000, Train Loss: 5683.1196, Test Loss: 286.4649, Updating class 2
Optimization finished.
Final Train Accuracy: 0.8000
Final Test Accuracy: 0.9000
Final Train Loss: 5683.1196
Final Test Loss: 286.4649
Total time: 0.08 seconds
```

```
import matplotlib.pyplot as plt

# Plotting Accuracy vs Iteration for Train and Test
plt.figure(figsize=(7, 5))

# Accuracy vs Iteration for Train
plt.subplot(2, 2, 1)
plt.plot(accuracies_train_gd, label='GD Train')
plt.plot(accuracies_train_bcgd, label='BCGD Train')
plt.title('Train Accuracy vs Iteration')
plt.xlabel('Iteration')
plt.ylabel('Accuracy')
plt.legend()

# Accuracy vs Iteration for Test
plt.subplot(2, 2, 2)
plt.plot(accuracies_test_gd, label='GD Test')
plt.plot(accuracies_test_bcgd, label='BCGD Test')
plt.title('Test Accuracy vs Iteration')
plt.xlabel('Iteration')
plt.ylabel('Accuracy')
plt.legend()
```

```
# Plotting Accuracy vs Time for Train and Test
# Accuracy vs Time for Train
plt.subplot(2, 2, 3)
plt.plot(train_times_gd, accuracies_train_gd, label='GD Train')
plt.plot(times_bcgd, accuracies_train_bcgd, label='BCGD Train')
plt.title('Train Accuracy vs Time')
plt.xlabel('Time (seconds)')
plt.ylabel('Accuracy')
plt.legend()

# Accuracy vs Time for Test
plt.subplot(2, 2, 4)
plt.plot(train_times_gd, accuracies_test_gd, label='GD Test')
plt.plot(times_bcgd, accuracies_test_bcgd, label='BCGD Test')
plt.title('Test Accuracy vs Time')
plt.xlabel('Time (seconds)')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```




```
import matplotlib.pyplot as plt


# Plotting Loss vs Iteration for Train and Test
plt.figure(figsize=(7, 5))

# Loss vs Iteration for Train
plt.subplot(2, 2, 1)
plt.plot(losses_train_gd, label='GD Train')
plt.plot(losses_train_bcgd, label='BCGD Train')
plt.title('Train Loss vs Iteration')
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.legend()

# Loss vs Iteration for Test
plt.subplot(2, 2, 2)
plt.plot(losses_test_gd, label='GD Test')
plt.plot(losses_test_bcgd, label='BCGD Test')
plt.title('Test Loss vs Iteration')
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.legend()

# Plotting Loss vs Time for Train and Test
# Loss vs Time for Train
plt.subplot(2, 2, 3)
```

```
plt.plot(train_times_gd, losses_train_gd, label='GD Train')
plt.plot(times_bcgd, losses_train_bcgd, label='BCGD Train')
plt.title('Train Loss vs Time')
plt.xlabel('Time (seconds)')
plt.ylabel('Loss')
plt.legend()
```