In [17]:

```
import numpy as np
import matplotlib.pyplot as plt
import torchvision
```

In [18]:

```
def fit_stump(X, Y, w, gamma):

    projections = X @ w
    a = 0
    b = 0
    c = 0
    min_error = float('inf')

    #chooses the midpoint as a threshold
    unique_projections = np.unique(projections)
    thresholds = (unique_projections[:-1] + unique_projections[1:]) / 2


    for b in thresholds:
        # Indicator function
        I = (projections + b > 0).astype(float)

        a = (np.sum(gamma * Y * I ) - np.sum(gamma * c * I))/ (np.sum(gamma * I) + 1e-1
0)

        c = (np.sum(gamma * Y) - np.sum(gamma * a * I))/ (np.sum(gamma) + 1e-10)

        error = np.sum(gamma * (Y - (a * I + c))**2)

        if error < min_error:
            min_error = error
            optimal_a = a
            optimal_b = b
            optimal_c = c

    return optimal_a, optimal_b, optimal_c, min_error
```

In [19]:

```
def gentle_boost(X, Y, k):
    n= X.shape[0]
    d = X.shape[1]
    gamma = np.ones(n) / n

    W = np.zeros((d, k))
    a_param = np.zeros(k)
    b_param = np.zeros(k)
    c_param = np.zeros(k)

    for t in range(k):
        #prints eveyr 50th iteration so it's easier to follow the process
        print(f"Current k: {t+1}/{k}") if (t + 1) % 50 == 0 or t == k - 1 else None
        w = np.random.randn(d)
        w /= np.linalg.norm(w)

        a, b, c, min_error = fit_stump(X, Y, w, gamma)

        W[:, t] = w
        a_param[t] = a
        b_param[t] = b
        c_param[t] = c

        # Update weights
        fun_x = a * (X @ w + b > 0) + c
        gamma *= np.exp(-Y * fun_x)
```

```
        gamma /= np.sum(gamma)

    return W, a_param, b_param, c_param
```

In [20]:

```
trainset = torchvision.datasets.USPS(root='./data', download=True, train=True)
X_train, Y_train = np.array(trainset.data) / 255., np.array(trainset.targets)

tstset = torchvision.datasets.USPS(root='./data', download=True, train=False)
X_test, Y_test = np.array(tstset.data) / 255., np.array(tstset.targets)

# Reshape the data
X_train = X_train[:, np.newaxis, :, :]
X_test = X_test[:, np.newaxis, :, :]

# Filter the data for digits 0 and 1
mask_train = ((Y_train == 0) | (Y_train == 1)).reshape(-1)
X_train, Y_train = X_train[mask_train], Y_train[mask_train]
mask_test = ((Y_test == 0) | (Y_test == 1)).reshape(-1)
X_test, Y_test = X_test[mask_test], Y_test[mask_test]

# Change labels for binary classification
Y_train[Y_train == 0] = -1
Y_test[Y_test == 0] = -1
#Reshape, otherwise there dimension problem again
X_train = X_train.reshape(X_train.shape[0], -1)
X_test = X_test.reshape(X_test.shape[0], -1)
# Shuffle the training data
inds = np.random.permutation(X_train.shape[0])
X_train, Y_train = X_train[inds], Y_train[inds]
```

In [21]:

```
#testing data
np.unique(Y_train)
np.unique(Y_test)
```

Out[21]:

```
array([-1,  1])
```

In [22]:

```
#testing dimensions
print(X_train.shape, Y_train.shape)
print(X_test.shape, Y_test.shape)
```

```
(2199, 256) (2199,)
(623, 256) (623,)
```

In [23]:

```
k = 1000
W, a_param, b_param, c_param = gentle_boost(X_train, Y_train, k)
```

```
Current k: 50/1000
Current k: 100/1000
Current k: 150/1000
Current k: 200/1000
Current k: 250/1000
Current k: 300/1000
Current k: 350/1000
Current k: 400/1000
Current k: 450/1000
Current k: 500/1000
Current k: 550/1000
Current k: 600/1000
Current k: 650/1000
Current k: 700/1000
Current k: 750/1000
Current k: 800/1000
```

```
Current k: 850/1000
Current k: 900/1000
Current k: 950/1000
Current k: 1000/1000
```

In [24]:

```python
#testing if preds work
pred = X_train @ W
np.sign(pred)
```

Out[24]:

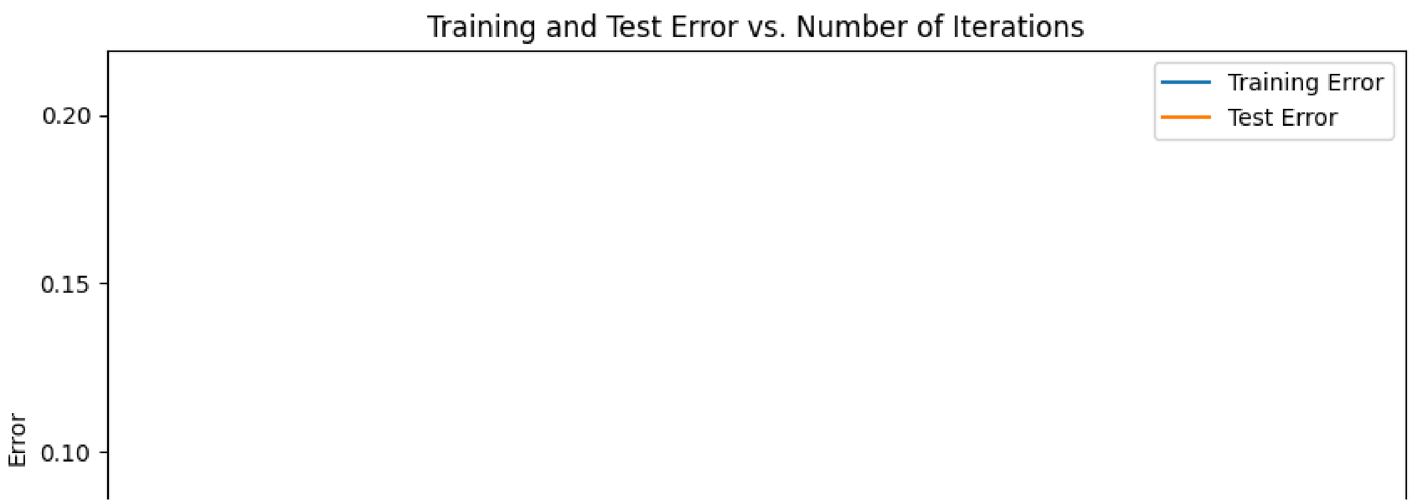
```
array([[-1., -1.,  1., ...,  1., -1., -1.],
       [ 1.,  1., -1., ...,  1.,  1.,  1.],
       [ 1., -1.,  1., ..., -1., -1., -1.],
       ...,
       [ 1., -1.,  1., ...,  1., -1., -1.],
       [-1.,  1.,  1., ...,  1., -1., -1.],
       [-1.,  1.,  1., ...,  1., -1., -1.]])
```

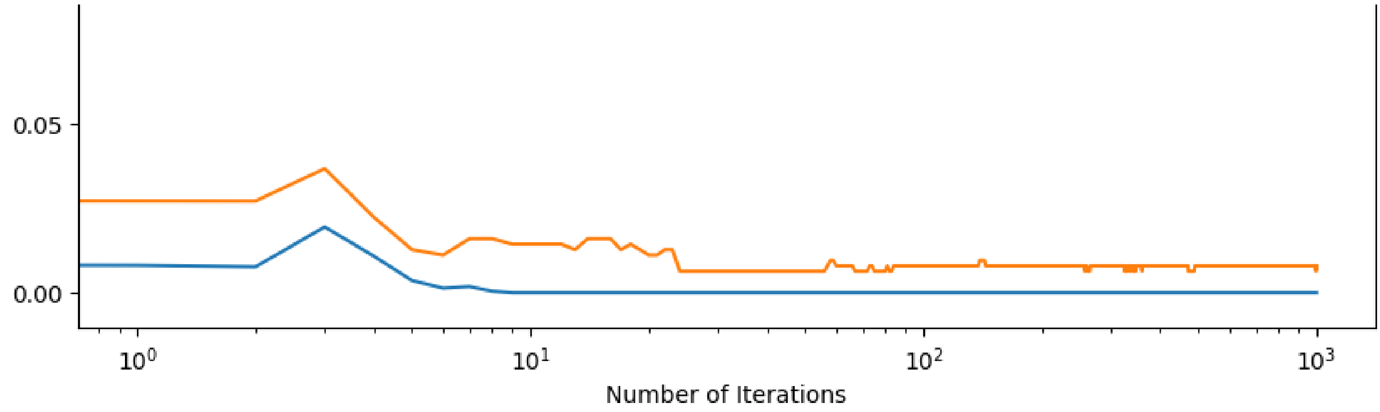
In [25]:

```python
train_errors = []
test_errors = []

for t in range(k):
    predictions_train = np.zeros(X_train.shape[0])
    predictions_test = np.zeros(X_test.shape[0])

    for i in range(t+1):
        I_train = ((X_train @ W[:, i] + b_param[i]) > 0)
        predictions_train += a_param[i] * I_train + c_param[i]

        I_test = ((X_test @ W[:, i] + b_param[i]) > 0)
        predictions_test += a_param[i] * I_test + c_param[i]

    train_error = np.mean(np.sign(predictions_train) != Y_train)
    train_errors.append(train_error)

    test_error = np.mean(np.sign(predictions_test) != Y_test)
    test_errors.append(test_error)
```

In [26]:

```python
# Plot the training and test errors
plt.figure(figsize=(10, 6))
plt.plot(range(k), train_errors, label='Training Error')
plt.plot(range(k), test_errors, label='Test Error')
plt.xlabel('Number of Iterations')
plt.ylabel('Error')
plt.xscale('log')
plt.title('Training and Test Error vs. Number of Iterations')
plt.legend()
plt.show()
```





In [27]:

```
trainset_full = torchvision.datasets.USPS(root='./data', download=True, train=True)
X_train_full, Y_train_full = np.array(trainset_full.data) / 255., np.array(trainset_full
.targets)

tstset_full = torchvision.datasets.USPS(root='./data', download=True, train=False)
X_test_full, Y_test_full = np.array(tstset_full.data) / 255., np.array(tstset_full.targe
ts)

X_train_full = X_train_full.reshape(X_train_full.shape[0], -1)
X_test_full = X_test_full.reshape(X_test_full.shape[0], -1)
```

In [ ]:

```
k_new = 200 #2000, # Running it for k=2000 takes hours, after ~100-200 iterations algorit
hm reaches a point where it doesn't imporve anymore and keeps going since thre iss no sto
pping criterion
classes = np.unique(Y_train_full).shape[0]   # Number of classes
train_errors_all = np.zeros((classes, k_new))
test_errors_all = np.zeros((classes, k_new))


for i in range(classes):
    print(f"Current class: {i}/{classes}")
    Y_train_bin = np.where(Y_train_full == i, 1, -1)
    Y_test_bin = np.where(Y_test_full == i, 1, -1)

    W, a_param, b_param, c_param = gentle_boost(X_train_full, Y_train_bin, k_new)

    for t in range(k_new):
        predictions_train_all = np.zeros(X_train_full.shape[0])
        for j in range(t + 1):
            I_train = (X_train_full @ W[:, j] + b_param[j] > 0)
            predictions_train_all += a_param[j] * I_train + c_param[j]


        train_errors_all[i, t] = np.mean(np.sign(predictions_train_all) != Y_train_bin)

        predictions_test_all = np.zeros(X_test_full.shape[0])
        for j in range(t + 1):
            I_test = (X_test_full @ W[:, j] + b_param[j] > 0)
            predictions_test_all += a_param[j] * I_test + c_param[j]


        test_errors_all[i, t] = np.mean(np.sign(predictions_test_all) != Y_test_bin)

        print(f"Iteration {t + 1}/{k_new}: Training Error = {train_errors_all[i, t]}, Te
sting Error = {test_errors_all[i, t]}")
```

In [30]:

```
# Plot the training and test errors for each digit in one-versus-all classification
plt.figure(figsize=(12, 8))
for i in range(classes):
    plt.plot(range(k_new), train_errors_all[i], label=f'Train Error - Digit {i}')
```

```
    plt.plot(range(k_new), test_errors_all[i], label=f'Test Error - Digit {i}')
plt.xlabel('Number of Iterations')
plt.ylabel('Error')
plt.xscale('log')
plt.title('Test Error vs. Number of Iterations for One-Versus-All Classification')
plt.legend()
plt.show()
```