MATH 173A: Homework #7

Due on Dec 3, 2024 at 23:59pm

Professor Cloninger

Ray Tsai

A16848188

Problem 1

Suppose a function $f : \mathbb{R}^d \to \mathbb{R}$ is L-smooth with $L = 4$ and satisfies the PL-property with parameter $\mu = 2$, i.e.,

$$\frac{1}{2}\|\nabla f(x)\|^2 \geq \mu(f - f^*).$$

Consider the gradient descent method for minimizing f. Let x^* be the global minimum and suppose $x^{(0)}$ is the initialization such that

$$\|x^* - x^{(0)}\| \leq 5.$$

Determine the step size η and the number of steps needed to satisfy

$$|f(x^{(t)}) - f(x^*)| \leq 10^{-4}.$$

Proof. The step size is $\eta = \frac{1}{L} = \frac{1}{4}$. The convergence rate is

$$\begin{aligned} f(x^{(t)}) - f(x^*) &\leq \left(1 - \frac{\mu}{L}\right)^t [f(x^{(0)}) - f(x^*)] \\ &= (0.5)^t [f(x^{(0)}) - f(x^*)]. \end{aligned}$$

Since f is L-smooth and $\|x^* - x^{(0)}\| \leq 5$,

$$\|\nabla f(x^{(0)})\| = \|\nabla f(x^{(0)}) - f(x^*)\| \leq L\|x^* - x^{(0)}\| \leq 4 \cdot 5 = 20.$$

By the PL-condition,

$$f(x^{(0)}) - f(x^*) \leq \frac{1}{2\mu}\|\nabla f(x^{(0)})\|^2 \leq \frac{1}{4} \times 400 = 100.$$

Hence,

$$f(x^{(t)}) - f(x^*) \leq (0.5)^t \times 100 \leq 10^{-4} \implies t \geq 6\log_2 10 \approx 20.$$

□

Problem 2

Consider the following set in $\mathbb{R}^n$ for an integer $s > 0$:

$$B = \{x \in \mathbb{R}^n \mid x_i \geq 0, \text{ for } i = 1, \dots, n \text{ and } x \text{ has at most } s \text{ nonzeros.}\}.$$

(a) Find an expression for the orthogonal projection of a point $x \in \mathbb{R}^n$ onto B (No need for justification).

Proof. Let $x_i^+ = \max(x_i, 0)$, and let $I_s(x)$ be the index set of the s largest components of x. Note that $|I_s(x)| = s$. Define projection $\Pi_B(x)$ by sending

$$x_i \mapsto \begin{cases} x_i & \text{if } i \in I_s(x^+) \\ 0 & \text{otherwise.} \end{cases}$$

□

(b) For the function

$$f(x) = \frac{1}{2}\|Ax - b\|^2,$$

write a projected gradient descent algorithm to solve

$$\min_{x \in \Omega} f(x)$$

for $\Omega = B$, with B from part (a). You need to specify the gradient formula and the projection formula. You do not need to specify the step size for this problem.

Proof. Let $x^{(0)} \in B$, and let μ be the step size. For $t = 1, \dots,$

1. Set $y^{(t+1)} = x^{(t)} - \mu \nabla f(x^{(t)}) = x^{(t)} - \mu A^T(Ax^{(t)} - b) = (I - \mu A^T A)x^{(t)} + \mu A^T b$.
2. Set $y_i^{(t+1)} = \max(0, y_i^{(t+1)})$ for all i.
3. Calculate $I_s(y^{(t+1)})$.
4. Set $x_i^{(t+1)} = \begin{cases} y_i^{(t+1)} & \text{if } i \in I_s(y^{(t+1)}) \\ 0 & \text{otherwise} \end{cases}$.

□

(c) Consider the function in (b) and suppose

$$A = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \quad s = 1$$

for the set B in (a). Does the projected gradient method converge to the global minimizer for any initialization $x^{(0)}$ if the step size $\mu \leq \frac{1}{8}$? Justify your answer.

Proof. No. Consider initializations $x^{(0)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $x^{(0)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$.

Case 1: $x^{(0)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$**.**

Problem 2 continued on next page. . .

Following the steps in (b),

$$y^{(1)} = \begin{bmatrix} 1 \\ \mu \end{bmatrix}.$$

Since $\mu \leq 1$, $x^{(1)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, and thus the algorithm converges to $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$.

Case 2: $x^{(0)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$**.**

Following the steps in (b),

$$y^{(1)} = \begin{bmatrix} 4\mu \\ 1 \end{bmatrix}.$$

Since $4\mu \leq 0.5 \leq 1$, $x^{(1)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, and thus the algorithm converges to $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$. $\square$

But then $f(1,0) = 0.5$ and $f(0,1) = 2$, so the algorithm does converge to the global minimum for all initializations.

Question 3

Consider the optimization problem:

$$f(x) = \frac{1}{2}\|Ax - b\|^2, \tag{1}$$

where $A \in \mathbb{R}^{20\times 50}$ and $b \in \mathbb{R}^{20}$ are from the dataset `HW7Q3.csv`. The file `HW7Q3.csv` contains the data A and b. The first 50 columns form the matrix A and the last column is the vector b. The vector b is generated by setting $b = Ax^*$ for a vector $x^* \in \mathbb{R}^{20}$ that has 2 nonzeros. Note the linear system $Ax = b$ is underdetermined and has a lot of solutions. Write a projected gradient method for the following optimization problem to find the x^*:

$$\text{minimize } f(x) \quad \text{s.t. } x \text{ has at most 2 nonzeros.}$$

You can experiment with the stepsize to make sure $f(x^{(t)})$ converges to 0. You need to submit the code, the plot of $f(x^{(t)}) - f(x^*) = f(x^{(t)})$, and the indices and values of the nonzero entries of x^* you found.

In [84]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [85]:
```
data = np.loadtxt('HW7Q3.csv', delimiter=',')
A = data[:, :-1]
b = data[:, -1]
```

In [86]:
```
d = A.shape[1]
x = np.zeros(d)
T = 10
f_values = []
```

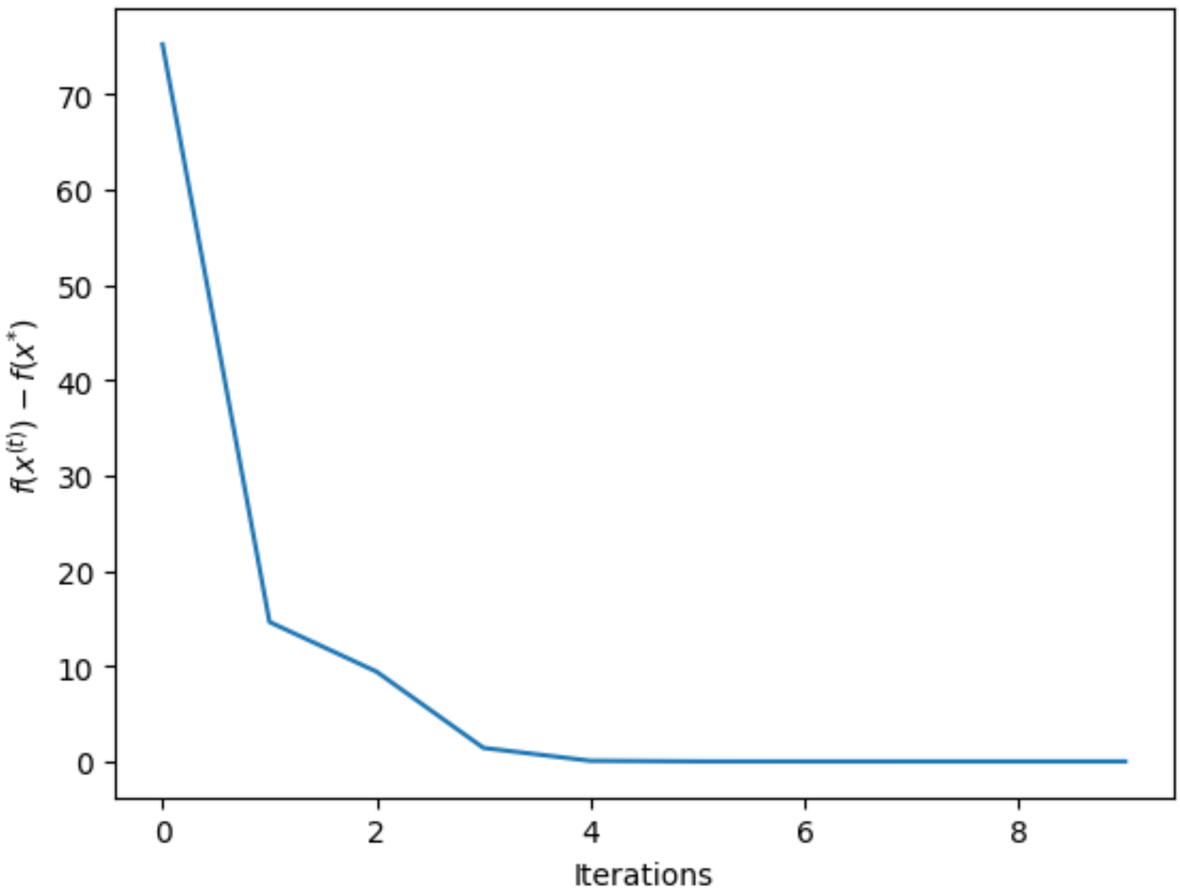
In [87]:
```
def f(x):
    return 1/2 * np.linalg.norm(A @ x - b)**2

def df(x):
    return A.T @ (A @ x - b)
```

In [88]:
```
s = 2
mu = 5e-2

x = np.zeros(d)
```

```
for t in range(T):
    f_values.append(f(x))
    y = x - mu * df(x)
    y_plus = np.maximum(y, 0)
    I_s = np.argpartition(y_plus, -s)[-s:]
    x = np.zeros(d)
    x[I_s] = y[I_s]

print(f_values[-1])
```

```
1.409537444226036e-07
```

In [90]:
```
plt.plot(f_values)
plt.xlabel('Iterations')
plt.ylabel(r'$f(x^{(t)}) - f(x^{*})$')
plt.show()
```



In [92]:
```
nonzero_ind = np.nonzero(x)[0]
nonzero_val = x[nonzero_ind]
print("Nonzero indices:", nonzero_ind)
print("Nonzero values:", nonzero_val)
```

```
Nonzero indices: [ 0 14]
Nonzero values: [0.99997612 2.99992859]
```

Question 4

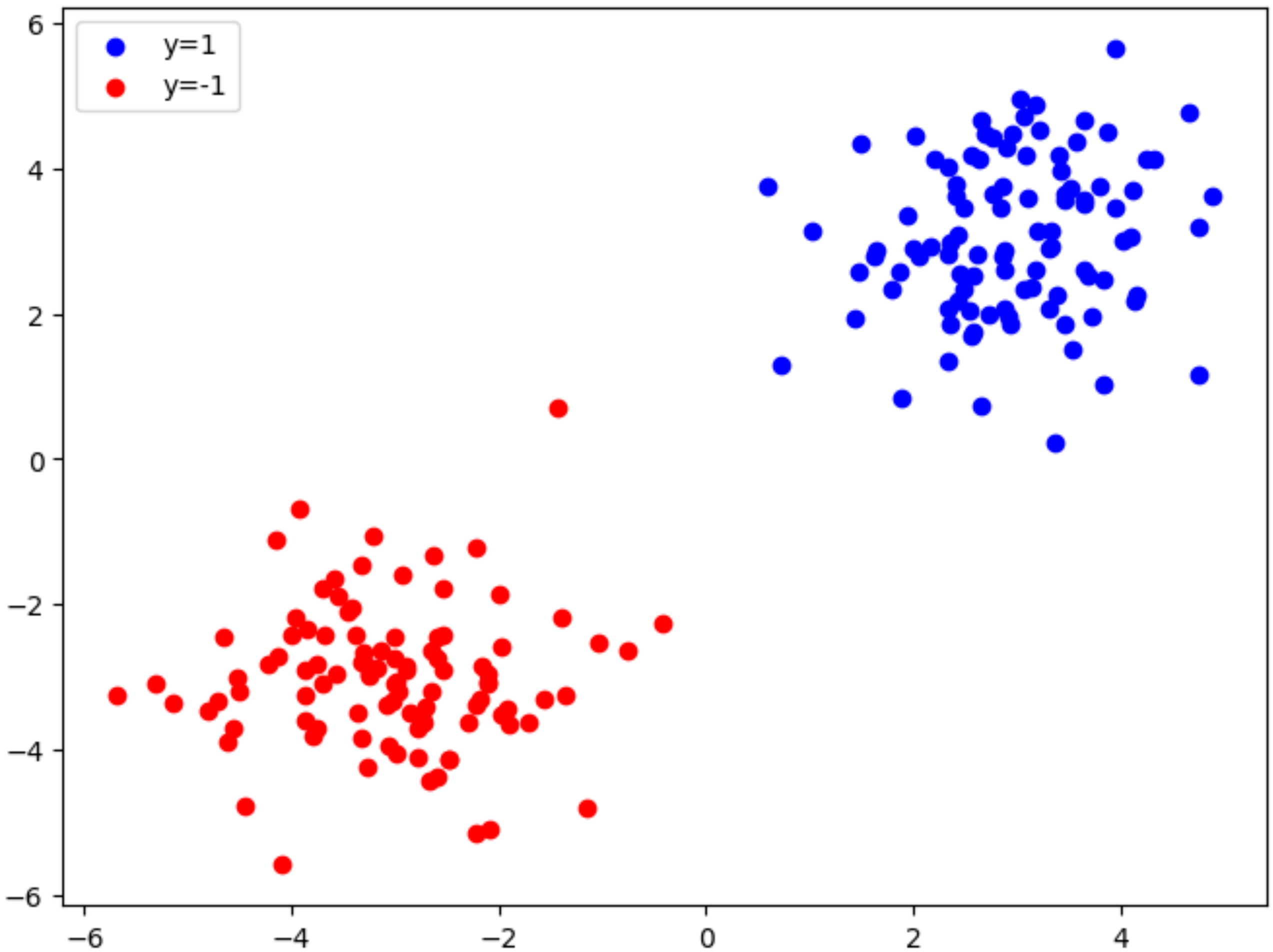
We will implement the SVM algorithm with gradient descent to classify two gaussians in 2D. The dataset is given in `HW7Q4.csv`.

In [33]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Part A

In `HW7Q4.csv`, the first 100 rows are the data for cluster 1: $(x_i, y_i) \in \mathbb{R}^2 \times \mathbb{R}$, $i = 1, \ldots, 100$, with $y_i = 1$ always. The next 100 rows are the data for cluster 2: $(x_i, y_i) \in \mathbb{R}^2 \times \mathbb{R}$, $i = 101, \ldots, 200$, with $y_i = -1$ always. Create and turn in a scatter plot of the feature vectors, i.e., the x_i's, colored by the label, i.e., y_i's (blue for 1 and red for -1).

In [34]:
```
data = pd.read_csv('HW7Q4.csv', header=None)
data.columns = ['x1', 'x2', 'y']

plt.figure(figsize=(8, 6))
plt.scatter(data[:100]['x1'], data[:100]['x2'], color='blue', label='y=1')
plt.scatter(data[100:]['x1'], data[100:]['x2'], color='red', label='y=-1')
plt.legend()
plt.show()
```



Part B

Create a function for the gradient of the loss

$$L(w) = \frac{1}{2}\|w\|^2 + \sum_{i=1}^{n} \max(0, 1 - y_i \langle x_i, w \rangle)$$

$$\nabla L(w) = w + \sum_{i=1}^{n} -y_i x_i \cdot 1_{1-y_i\langle x_i, w\rangle > 0},$$

where $1_{1-y_i\langle x_i,w\rangle>0} = \begin{cases} 1 & \text{if } 1 - y_i\langle x_i, w\rangle > 0 \\ 0 & \text{else} \end{cases}$. Also, here $n = 200$. To compute the gradient, you'll have to compute an indicator of whether $1 - y_i\langle x_i, w\rangle$ is positive or negative at every point, and sum up the contribution of this term for all points where it's positive.

In [43]:
```
n = len(data)
```

```python
def indicator(w, x1, x2, y):
    if 1 - y * (w[0] * x1 + w[1] * x2) > 0:
        return 1
    return 0

def L(w):
    return 1/2 * np.linalg.norm(w)**2 + sum([max(0, 1 - row['y'] * (w[0] * r

def dL(w):
    return w + sum([-row['y'] * np.array([row['x1'], row['x2']]) * indicator
```

Part C

Setting the step size $\mu = 10^{-4}$ and starting at $w^{(0)} = (-1, 1)$, run 1000 iterations of gradient descent. You will create two plots.

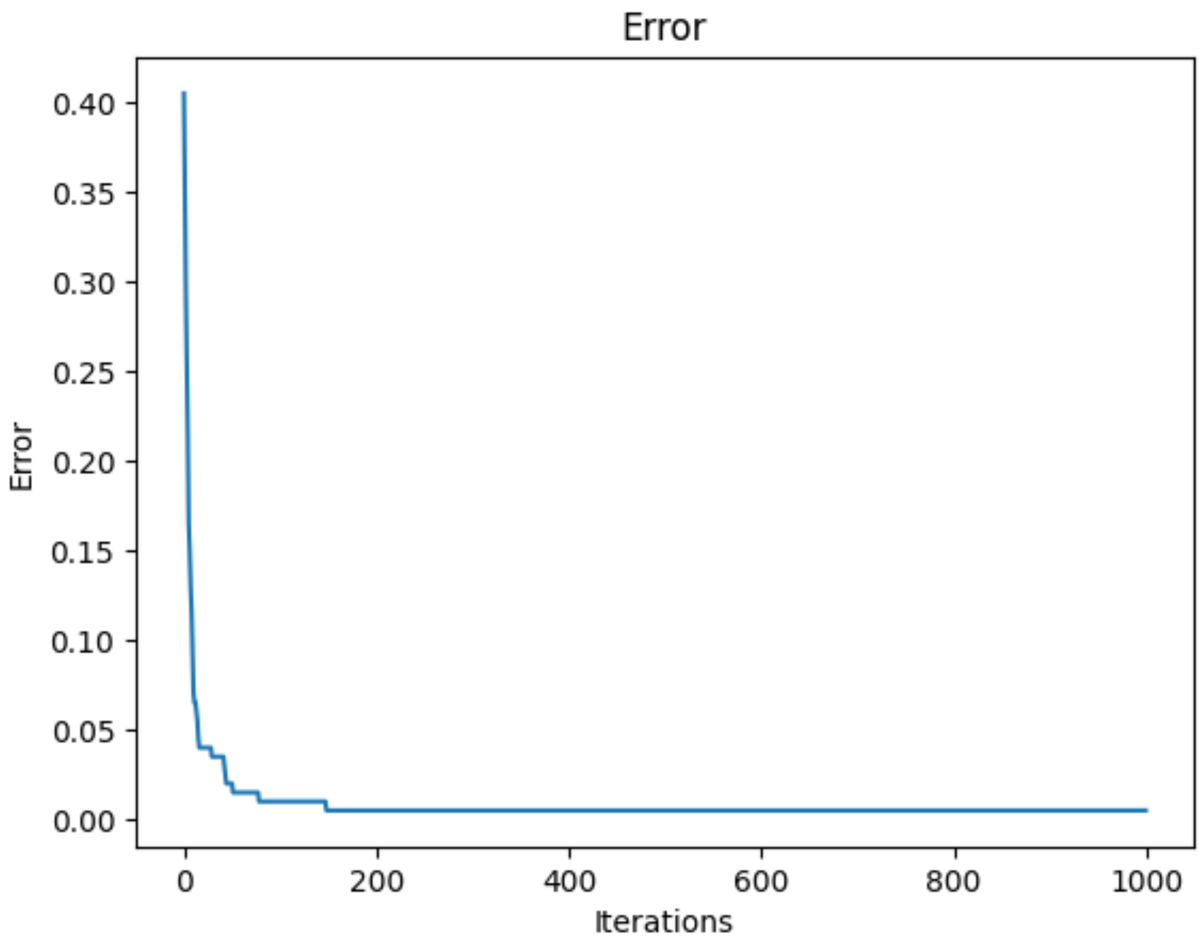
i. Plot the classification error (averaged over all the points) as a function of the iterations. The classification of x_i is determined by $\text{sign}(\langle x_i, w \rangle)$.

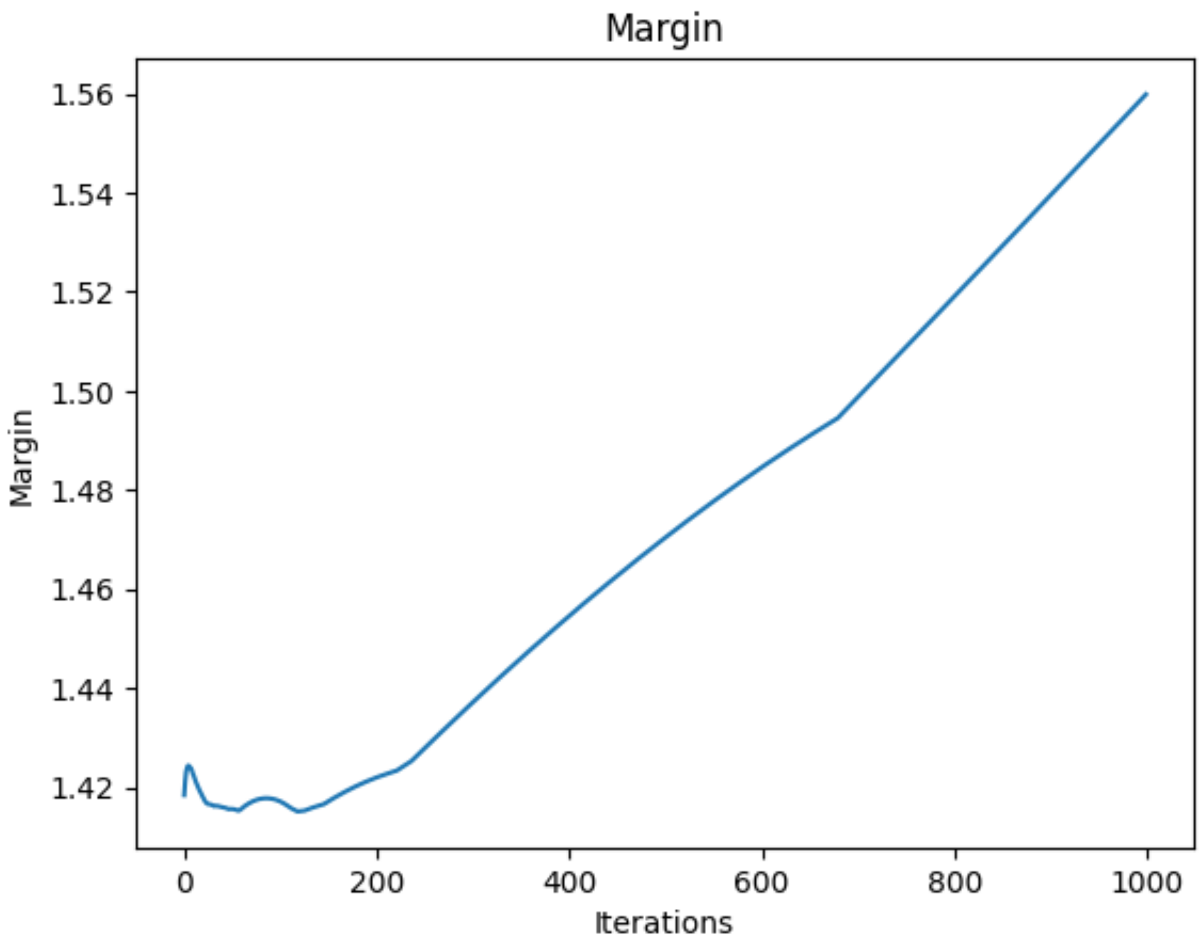
ii. Plot the margin $\frac{2}{\|w\|}$ as a function of the iterations. This shows how much of a gap you have between the classes you've learned.

In [ ]:
```python
T = 1000
mu = 1e-4

def error(w):
    sum = 0
    for _, row in data.iterrows():
        if np.sign(w[0] * row['x1'] + w[1] * row['x2']) != row['y']:
            sum += 1
    return sum / n

w = np.array([-1, 1])
error_val = []
margin_val = []

for t in range(T):
    w = w - mu * dL(w)
    error_val.append(error(w))
    margin_val.append(2 / np.linalg.norm(w))
```

In [ ]:
```python
plt.plot(error_val)
plt.xlabel('Iterations')
plt.ylabel('Error')
plt.title('Error')
```

```
plt.show()

plt.plot(margin_val)
plt.xlabel('Iterations')
plt.ylabel('Margin')
plt.title('Margin')

plt.show()
```

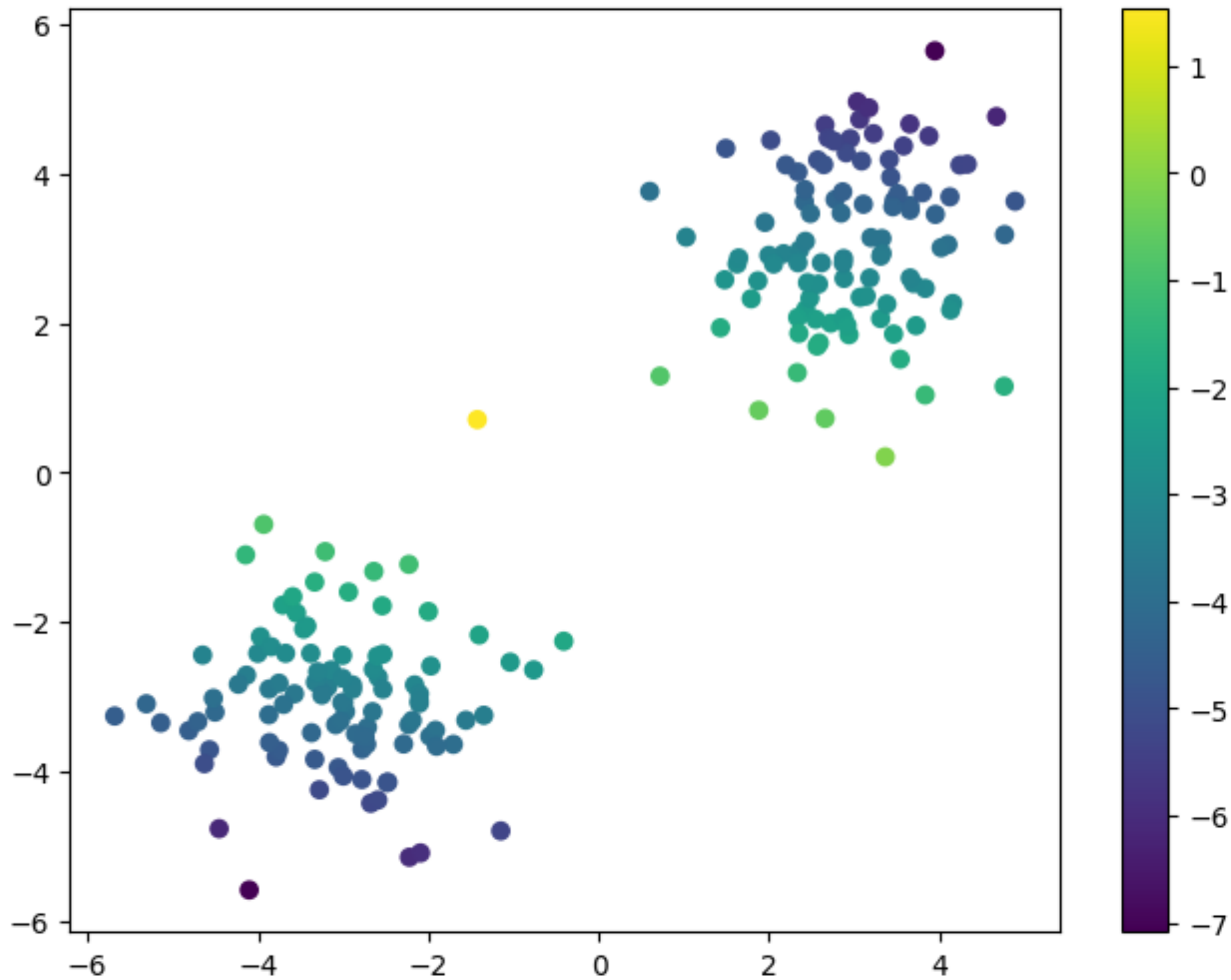




Part D

Create another scatter plot of your data, but this time color the points by the function $f(x_i) = 1 - y_i \cdot \langle x_i, w \rangle$. The numbers closest to 0 (positive numbers or largest negative numbers) will show you which points were "most important" in determining the classification.

In [54]:
```
f_values = [1 - row['y'] * (w[0] * row['x1'] + w[1] * row['x2']) for _, row

plt.figure(figsize=(8, 6))
plt.scatter(data['x1'], data['x2'], c=f_values, label='f(x_i)')
plt.colorbar()
plt.show()
```



6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
1
0
−1
−2
−3
−4
−5
−6
−7