SerraJoseAssignment3

October 19, 2023

Hoeffding's Bound Inequality

$$\mathbb{P}[|E_{in}(g) - E_{out}(g)| > \epsilon] \leq 2Me^{-2\epsilon N} \tag{1}$$

#Question 1

0.1 (a)

1 Equation 1: Error Bound

$$\sqrt{\frac{1}{2N} \cdot \mathbf{ln}(\frac{2M}{\delta})} \tag{2}$$

2 Substitution

$$\sqrt{\frac{1}{2(600)} \cdot \mathbf{ln}(\frac{2 \cdot (1000)}{(0.05)})} = 9.397 \cdot 10^{-2} \tag{3}$$

The full Error Bar is given as follows:

$$E_{out} \leq E_{in} + \sqrt{\frac{1}{2(600)} \cdot \mathbf{ln}(\frac{2 \cdot (1000)}{(0.05)})} \tag{4}$$

(b) This will cause the variance to rise which is not what we want. We want to find balance and examine the error where every parameter is at its minimum.

#Question 2

2.1 (a)

3 Equation 2 : Growth Function

- $X = \mathbb{R}$(one dimensional)
- $\mathcal{H}$ contains h, where each h(x) = sign(x - a) for threshold a
- one dichotomy for $a \in (x_n, x_{n+1})$:

$$m_H(N) \leq N + 1 \tag{5}$$

which is the maximum number of dichotomies for positive rays

OR

$$m_H(N) \le N + 1 \tag{6}$$

which is the maximum number of dichotomies for negative rays

Considering both positive or negative rays the maximum number of dichotomies:

$$m_H(N) \le (N + 1) + (N + 1) \tag{7}$$
$$\le (-\infty, a] \cup [a, \infty) \tag{8}$$
$$\le \mathbf{set_W} + \mathbf{set_V};\ \text{where } \mathbf{set_W} = (-\infty, a] \text{ and } \mathbf{set_V} = [a, \infty) \tag{9}$$
$$\le \mathbf{set_W} + \mathbf{set_V} - (\mathbf{set_W} \cap set_V) \tag{10}$$
$$\le (-\infty, a] + [a, \infty] - [(-\infty, a] \cap [a, \infty)] \tag{11}$$
$$\le (-\infty, \infty) \tag{12}$$
$$\le 2(N + 1) \tag{13}$$

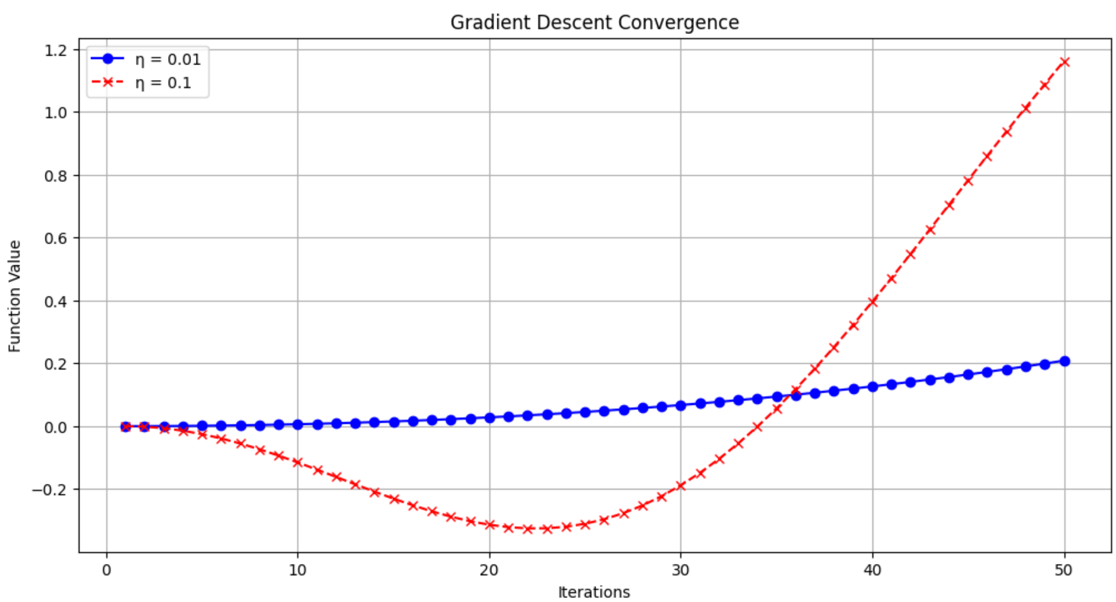
3.1 (b)

The circumstance that one is given in the problem is the Real Number Line therefore the VC-dimension is infinty. $m_h(N) = 2^N$ for all N, then $d_{vc}(H) = \infty$

4 Question 3

[14]:
```
import numpy as np
import matplotlib.pyplot as plt

# Define the function
def f(x, y):
    return x**2 + 2*y**2 + 2*np.sin(2*np.pi*x)*np.sin(2*np.pi*y)
def gradient_descent_Jose(learning_rate, num_iterations):
    x = np.linspace(0,.1,num=num_iterations)
    y = np.linspace(0,.1,num=num_iterations)
    X, Y = np.meshgrid(x, y)
    df_dx, df_dy =  np.gradient(f(X,Y), x, y )
    z = []
    for i in range(num_iterations):
        #x -= learning_rate * df_dx
        #y -= learning_rate * df_dy
        # Update x and y using gradient descent
        x -= learning_rate * df_dx[0]
        y -= learning_rate * df_dy[0]
        z.append(f(x, y))


    return x, y, z
learning_rate_1 = 0.01
```

```
learning_rate_2 = 0.1
num_iterations = 50

# Perform gradient descent
x1, y1, history1 = gradient_descent_Jose(learning_rate_1, num_iterations)
x2, y2, history2 = gradient_descent_Jose(learning_rate_2, num_iterations)

# Plot the convergence
iterations = list(range(1, num_iterations + 1))
plt.figure(figsize=(12, 6))
plt.plot(iterations, history1[0], label='  = 0.01', marker='o', linestyle='-',␣
 ↪color='b')
plt.plot(iterations, history2[0], label='  = 0.1', marker='x', linestyle='--',␣
 ↪color='r')
plt.xlabel('Iterations')
plt.ylabel('Function Value')
plt.legend()
plt.title('Gradient Descent Convergence')
plt.grid(True)
plt.show()
plt.figure(figsize=(12, 6))
```



[14]: <Figure size 1200x600 with 0 Axes>

<Figure size 1200x600 with 0 Axes>

[15]:
```
# Gradient of the function
def gradient_f(x, y):
    df_dx = 2*x + 4*np.pi*np.cos(2*np.pi*x)*np.sin(2*np.pi*y)
    df_dy = 4*y + 4*np.pi*np.sin(2*np.pi*x)*np.cos(2*np.pi*y)
    return df_dx, df_dy
# Define a function for gradient descent
def gradient_descent_with_starting_point(learning_rate, num_iterations,␣
 ↪start_point):
    x, y = start_point
    history = []  # Store function values for plotting

    for i in range(num_iterations):
        df_dx, df_dy = gradient_f(x, y)
        x -= learning_rate * df_dx
        y -= learning_rate * df_dy
        history.append(f(x, y))

    return x, y, history

# List of initial points
initial_points = [(0.1, 0.1), (1, 1), (-0.5, -0.5), (-1, -1)]

# Perform gradient descent for each initial point
results = []
for start_point in initial_points:
    x, y, history = gradient_descent_with_starting_point(learning_rate_1,␣
 ↪num_iterations, start_point)
    min_value = f(x, y)
    results.append((start_point, (x, y), min_value))

# Print results in a table
print("Initial Point     Minimum Location    Minimum Value")
for start_point, (x, y), min_value in results:
    print(f"{start_point}        ({x:.4f}, {y:.4f})        {min_value:.4f}")
```

```
Initial Point     Minimum Location    Minimum Value
(0.1, 0.1)        (0.2438, -0.2379)        -1.8201
(1, 1)        (1.2181, 0.7128)        0.5933
(-0.5, -0.5)        (-0.7314, -0.2379)        -1.3325
(-1, -1)        (-1.2181, -0.7128)        0.5933
```

5 Question 4

The bias variance trade off allows one to note that the increasing of the validation set test size one can better approximate the out-of-sample error since bias nears 0.