Lab 5 - Confirm the Johnson-Lindenstraus lemma using simulations

Turgunboev Dadakhon

Import libraries

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

Function to generate N random high-dimensional vectors

Generates N random vectors in a d-dimensional space. Each vector follows a standard normal distribution.

```
def generate_high_dim_vectors(N, d):
    return np.random.randn(N, d)
```

Function to create a random projection matrix

Creates a random projection matrix of size (n, d). Each element follows a standard normal distribution and is scaled by 1/sqrt(n).

```
def random_projection_matrix(d, n):
    return np.random.randn(n, d) / np.sqrt(n)
```

Function to project high-dimensional vectors to lower dimensions

Projects high-dimensional vectors X using the projection matrix P. The result is a lower-dimensional representation of X.

```
def project_vectors(X, P):
    return X @ P.T
```

Function to compute pairwise Euclidean distances between vectors

Computes the pairwise Euclidean distances between all vectors in X. Returns a symmetric distance matrix.

```
def compute_distances(X):
    N = X.shape[0]
    D = np.zeros((N, N))
    for i in range(N):
        for j in range(i + 1, N):
            D[i, j] = np.linalg.norm(X[i] - X[j])
            D[j, i] = D[i, j] # Symmetric matrix
    return D
```

Function to verify the Johnson-Lindenstrauss lemma through simulations

Runs multiple trials to verify the Johnson-Lindenstrauss lemma. Measures how well pairwise distances are preserved after random projection.

```
def verify_jl_lemma(N, d, n_values, delta_values, epsilon_values,
num_trials=10):
    probabilities = {}
    for delta in delta_values:
        for epsilon in epsilon_values:
            success_rates = []
            for n in n_values:
                count = 0
                for _ in range(num_trials):
                    # Generate high-dimensional vectors
                    X = generate_high_dim_vectors(N, d)
                    # Create a random projection matrix
                    P = random_projection_matrix(d, n)
                    # Project the vectors to lower dimension
                    Y = project_vectors(X, P)
                    # Compute pairwise distances before and after
projection
                    D_x = compute_distances(X)
                    D_y = compute_distances(Y)
                    # Check if distances satisfy the JL lemma
condition
                    valid = np.logical_and(
                        (1 - delta) * D_x <= D_y,
                        D_y <= (1 + delta) * D_x
                    )
                    # Compute the probability of satisfying the
condition
                    success_prob = np.mean(valid[np.triu_indices(N,
k=1)])
                    if success_prob >= 1 - epsilon:
                        count += 1
                success_rates.append(count / num_trials)
```

```
            probabilities[(delta, epsilon)] = success_rates
    return probabilities
```

Function to plot the results

Plots the success probability of satisfying the JL lemma condition as a function of the reduced dimension n.

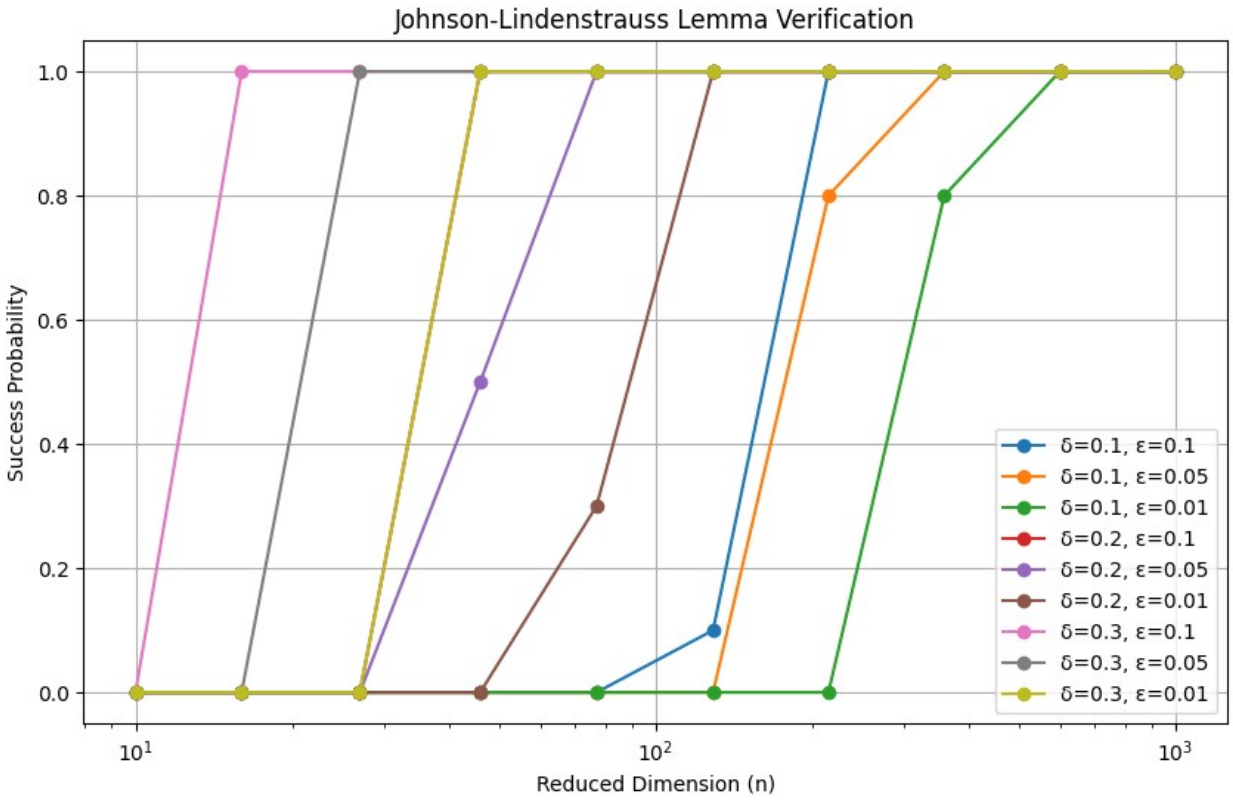
```
def plot_results(n_values, probabilities):
    plt.figure(figsize=(10, 6))
    for (delta, epsilon), success_rates in probabilities.items():
        plt.plot(n_values, success_rates, marker='o',
label=f"δ={delta}, ε={epsilon}")
    plt.xscale("log")
    plt.xlabel("Reduced Dimension (n)")
    plt.ylabel("Success Probability")
    plt.title("Johnson-Lindenstrauss Lemma Verification")
    plt.legend()
    plt.grid()
    plt.show()
```

Experiment parameters

```
N = 100  # Number of vectors
d = 1000  # High-dimensional space
n_values = np.logspace(1, 3, num=10, dtype=int)  # Range of reduced
dimensions in log scale
delta_values = [0.1, 0.2, 0.3]  # Distortion factors
epsilon_values = [0.1, 0.05, 0.01]  # Failure probabilities

# Run the experiment
probabilities = verify_jl_lemma(N, d, n_values, delta_values,
epsilon_values)

# Visualize results
plot_results(n_values, probabilities)
```



Summary

Expected dependency:

- The more n. the higher the probability of success
- For smaller δ and ϵ, the probability os success increase more slowly

Logarithmic scale n:

- Changing the size of the projection exponentially affects the probability of success

Threshold values:

- The lines rise sharply, which corresponds to a sharp increase in the probability of success when the critical n is reached.