PROJECT REPORT : IMPLEMENTATION OF CORESET ALGORITHM

1.INTRODUCTION:

The aim of this project was to implement algorithms for generating coresets as described in a research paper. Coresets are a smaller subset of data points that can accurately represent the original dataset. This report outlines the implementation process and discusses the relative error performance of coresets generated using uniform sampling and practical methods.

2. Algorithms Implementation:

Two algorithms, algo-1 and algo-2, were implemented based on the descriptions provided in the research paper.

Algorithm 1 (algo-1):

Description: Selects k random data points as centers.

Implementation: Implemented in Python using NumPy library.

Pseudocode:

Input: data (input data points), k (number of centers)

Output: centers (subset of k data points)

centers = Randomly select k data points from the input data return centers

Algorithm 2 (algo-2):

Description: Selects k data points based on weights assigned to each data point.

Implementation: Implemented in Python using NumPy library.

Pseudocode:

Input: data (input data points), weights (weight assigned to each data point), k (number of centers)

Output: centers (subset of k data points)

centers = Randomly select k data points with probability proportional to their weights return centers

3. Performance Evaluation:

To evaluate the performance of coresets generated by the implemented algorithms, relative error was computed.

Relative Error Performance:

Relative error is a measure of the difference between the clustering obtained using the coresets and the clustering obtained using the entire dataset. It is calculated using a suitable evaluation metric.

Relative error from Uniformly random Coreset is 1.5379973941340326

4. Implementation of Coreset Generation:

The performance of both uniform sampling-based and practical coresets was computed using provided scripts and files.

Uniform_Sampling.py: Constructs a fixed-size coreset using uniform sampling and returns relative error performance.

wkpp.py: Implements weighted kMeans++ algorithm required for practical coreset generation.

skeleton.py: Provides a skeleton for writing code or taking reference from Uniform_Sampling.py.

5. Performance Comparison:

Both uniform sampling-based and practical coresets were evaluated for their relative error performance.

Uniform_Sampling1.py:

Relative error from Uniformly random Coreset **is** -1.5379973941340326

Skeleton.py:

Relative error from Practical Coreset is 0.8495299208348667

Relative error from Uniformly random Coreset is 1.5071939069065823

6. Conclusion:

The implementation of algo-1 and algo-2 algorithms provided a foundation for generating coresets. By comparing the relative error performance of uniform sampling-based and practical coresets, we gained insights into their effectiveness. Further experimentation and analysis can lead to enhancements in coreset generation techniques, improving their applicability in various domains.