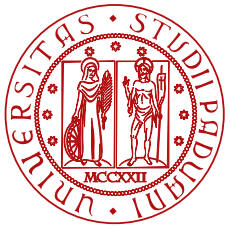

Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Master Thesis in Computer Engineering

Clustering for Large-Scale High-Dimensional Data Visualization

Master Candidate

Paria Tahan

Student ID 2043889

Supervisor

Prof. Matteo Ceccarello

University of Padova

Academic Year
2023/2024

To my husband
parents
and friends

Abstract

Multiple dimension reduction techniques, whether preserving global or local structures, demonstrate impressive visualization performance on many real-world datasets. Techniques such as T-SNE, UMAP, and TriMap are popular choices. However, the main challenge remains the running time, especially when working with large, high-dimensional data. One potential solution is to take a sample subset of the original data to start the embedding process. While this approach might not yield results as accurate as those obtained from the full dataset, it significantly reduces computation time. Center-based clustering is a fundamental primitive in data analysis, which allows to identify key landmark points that are representative of the entirety of the dataset. This thesis introduces a technique combining UMAP and clustering focusing on global structure preservation. We propose four metrics, each providing a different perspective on applying UMAP over k centers. Our results are supported by a series of experiments on both real-world and synthetic datasets, containing up to 15 million points. These experiments demonstrate that our algorithm produces higher-quality solutions than the standard UMAP method.

Contents

1

Introduction

The increasing availability of large-scale, high-dimensional datasets across various fields, such as bioinformatics, finance, and computer vision, has introduced new challenges for data analysis. As datasets grow in both size and complexity, they demand more sophisticated tools for meaningful interpretation and visualization. High-dimensional data often contain rich structures that are difficult to observe directly due to the so-called curse of dimensionality" [7]. This phenomenon highlights the difficulties that arise when analyzing and visualizing data in high-dimensional spaces, where the number of features far exceeds the number of observations, leading to challenges in computational efficiency, data sparsity, and model overfitting [2].

Dimensionality reduction techniques have emerged as a powerful tool to address these challenges. By transforming data from a high-dimensional space into a lower-dimensional representation, these techniques make it easier to visualize and interpret complex datasets while retaining the important structures within the data. There are two primary types of structures that these methods aim to preserve: local and global [1]. Local structure preservation ensures that points close to each other in high-dimensional space remain close in the low-dimensional embedding. Global structure preservation, on the other hand, ensures that the overall arrangement of clusters and groups within the data is retained.

Among the most popular dimensionality reduction techniques are t-Distributed Stochastic Neighbor Embedding (t-SNE), Uniform Manifold Approximation and Projection (UMAP), and Principal Component Analysis (PCA). Each of these

methods has strengths and weaknesses. For instance, t-SNE is widely used for preserving local structures and is particularly effective at revealing clusters within data. However, it struggles with scalability and tends to distort global structures as dataset sizes increase. PCA, applying a linear transformation to the data, is computationally efficient but may fail to capture nonlinear relationships within complex datasets [14]. UMAP, introduced in recent years, has gained prominence due to its ability to preserve both local and global structures, making it a highly effective tool for large-scale visualization tasks [8].

Despite the advantages of UMAP, its application to large-scale datasets remains computationally intensive, mainly when dealing with millions of data points. The high time complexity of UMAP makes it impractical for real-time or interactive data exploration on large datasets [1]. This leads to a need for optimization strategies that can reduce the computational burden without sacrificing the quality of the visualization. One promising approach to this problem is the use of coresets: small, representative subsets of the data that approximate the properties of the original dataset [4]. By selecting a strategically chosen coreset, it is possible to reduce the size of the data that needs to be processed, thus improving computational efficiency while still maintaining the key structures of the data.

The concept of coresets was first introduced as a method in computational geometry for approximating large datasets with a smaller, weighted subset of points. These subsets retain the essential geometric properties of the larger dataset, allowing for efficient approximation algorithms to be applied [4]. In the context of dimensionality reduction, coresets can be applied to preprocess data, selecting a small number of points that are representative of the larger dataset. These points are then used as input to methods such as UMAP, reducing both memory and time requirements. The challenge, however, lies in selecting a coreset that not only reduces the data size but also preserves the underlying structure that dimensionality reduction techniques seek to capture.

This thesis explores the combination of UMAP with k-center clustering for the creation of coresets, focusing on whether this approach can accelerate dimensionality reduction while preserving both local and global data structures. K-center clustering is an efficient clustering algorithm that selects k points (centers) in such a way that the maximum distance between any point and its nearest center is minimized [4]. By leveraging k-center clustering, we ensure that the coresets contain points that are representative of the entire dataset, capturing the

diversity and overall structure of the data. Furthermore, this thesis introduces a custom metric designed to optimize the embedding of these coresets in UMAP, which takes into account both the location and spread (radius) of the selected data points.

The custom metric proposed in this research augments the traditional Euclidean distance by incorporating a radius component, which adjusts the weight of each data point based on its relative importance. This allows UMAP to better capture both local neighborhoods and larger, global structures when applied to the reduced dataset. This thesis systematically evaluates the performance of this approach on both synthetic and real-world datasets, including large-scale datasets. These datasets, which contain millions of data points, provide a robust testing ground for examining the trade-offs between computation time and the quality of the embedding when using coreset-based methods.

The outline of this thesis is as follows: In Chapter 2, we will discuss several dimension reduction techniques and propose a new classification method at the end. Chapter 3 will present a new approach based on clustering for large-scale high-dimensional data. Chapter 4 will delve deeper into the implementation and validation part. Finally, in Chapter 5, we will conclude and discuss potential future work.

2

Related Works

Traditional dimensionality reduction techniques such as Principal Components Analysis (PCA) [12] and classical multidimensional scaling (MDS) [3] are linear techniques that focus on keeping the low-dimensional representations of dissimilar datapoints far apart.

For high-dimensional data that lie on or near a low-dimensional, non-linear manifold (In simpler terms, a manifold is a geometric object that is "locally flat".) it is usually more important to keep the low-dimensional representations of very similar datapoints close together, which is typically not possible with a linear mapping [5].

The challenges that were found among the mentioned algorithms can be noted as 1) not being very successful at visualizing real data. 2) Not being capable of retaining both the local and the global structure of the data in a single map.

In this chapter, we will explore various techniques used for dimensionality reduction. We will thoroughly examine their individual strengths and weaknesses and provide a comprehensive classification to summarize the leading approaches. This exploration will pave the way for the discovery of new methods for visualization and dimensionality reduction.

2.1 T-SNE

T-SNE, which stands for T-Distributed Stochastic Neighbor Embedding, is a powerful technique for visualizing high-dimensional data by mapping each

data point to a location in a two- or three-dimensional map. This method is an extension of the SNE (stochastic neighbor embedding) technique, which we will explore further in the next section to better understand its principles and applications.

T-Distributed Stochastic Neighbor Embedding (T-SNE) [9] focuses on two key aspects: (1) representing dissimilar data points with large differences in distance, and (2) representing similar data points with small differences in distance. The t-SNE cost function is simpler to optimize due to these characteristics and the approximate scale invariance of the Student t-distribution, compared to the cost functions of SNE and UNI-SNE. Specifically, t-SNE introduces long-range forces in the low-dimensional map, which can bring two clusters of similar points back together if they become separated early in the optimization process.

The SNE and UNI-SNE algorithms do not incorporate long-range forces, so they rely on simulated annealing to produce reasonable solutions. Conversely, t-SNE utilizes long-range forces to identify favorable local optima without the need for simulated annealing.

2.1.1 SNE

SNE method transforms the high-dimensional Euclidean distances between data points into conditional probabilities, which are used to process the similarities between the data points [9]. Specifically, the conditional probability $p_{j|i}$ represents the similarity of data point x_j to data point x_i. The probability definition is formulated in equation 2.1.

$$p_{j|i} = \frac{\exp\left(-\left\|x_i - x_j\right\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\left\|x_i - x_k\right\|^2 / 2\sigma_i^2\right)} \tag{2.1}$$

- For a given data point x_i, a Gaussian with a variance σ_i selects x_j as its neighbor, based on reasonable values.
- Because of considering pairwise similarity, we have $p_{i|i} = 0$.

$$q_{j|i} = \frac{\exp\left(-\left\|y_i - y_j\right\|^2\right)}{\sum_{k \neq i} \exp\left(-\left\|y_i - y_k\right\|^2\right)}. \tag{2.2}$$

It is possible to calculate a similar conditional probability $q_{j|i}$ and $\sigma = \frac{1}{\sqrt{2}}$ for low-dimensional data, and $q_{i|i} = 0$. It is explained in equation 2.2.

SNE aims to discover a low-dimensional representation of data that minimizes the difference between $p_{j|i}$ and $q_{j|i}$.

MEASURE: KullbackLeibler divergence (which is in this case equal to the cross-entropy up to an additive constant).

Stochastic Neighbor Embedding (SNE) [9] minimizes the sum of Kullback-Leibler divergences for all data points using gradient descent. SNE conducts a binary search to find the value of σ_i that produces a P_i with a specified perplexity defined in equation 2.3, which is a smooth measure of the effective number of neighbors between 5 and 50, as specified by the user.

$$\text{Perp}\left(P_i\right) = 2^{H(P_i)} \tag{2.3}$$

The parameter H (Shannon entropy) in the above equation has the following definition explained in equation 2.4. Shannon entropy is calculated using the probabilities of different outcomes in a given set of data and provides a quantification of the amount of information inherent in the data.

$$H\left(P_i\right) = -\sum_j p_{j|i} \log_2 p_{j|i} \tag{2.4}$$

The gradient descent of the equation-2.5 is:

$$\frac{\delta C}{\delta y_i} = 2 \sum_j \left(p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j}\right)\left(y_i - y_j\right). \tag{2.5}$$

The present gradient is combined with a gradually decreasing total of past gradients. This guides the adjustments in map point coordinates during each iteration of the gradient search. The gradient update with a momentum term represented in equation 2.6:

$$\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathcal{Y}} + \alpha(t)\left(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)}\right) \tag{2.6}$$

where $\mathcal{Y}^{(t)}$ indicates the solution at iteration t, η indicates the learning rate and $\alpha(t)$ represents the momentum at iteration t.

2.2 Cluster Visualization in Nonlinear Dimensionality Reduction

T-SNE, as explained in the previous section, is a widely used technique for visualizing high-dimensional data, which is particularly effective at preserving local neighborhood structures [9]. T-SNE minimizes the Kullback-Leibler divergence between distributions representing pairwise similarities in high-dimensional and low-dimensional spaces. While this approach works well for local structures, it often fails to reveal large-scale patterns, such as distinct clusters, especially in datasets with complex, high-dimensional relationships [14].

Current efforts to enhance t-SNE's performance by adjusting hyperparameters or using early exaggeration techniques [9] do not completely resolve the challenge of visualizing clusters. This is because t-SNE focuses mainly on preserving local neighborhood structures, which can lead to misrepresentations of larger-scale patterns, as highlighted in recent research studies [14].

2.2.1 Stochastic Cluster Embedding (SCE) for Improved Cluster Visualization

To address these limitations, Stochastic Cluster Embedding (SCE) was proposed by Yang et al. [14]. SCE builds on the Neighbor Embedding (NE) framework by introducing a generalization of Stochastic Neighbor Embedding (SNE), which adjusts the scale of output similarities using a flexible scaling factor. Unlike t-SNE, which uses a fixed scale for neighborhood similarities, SCE dynamically adjusts this parameter based on both local and global structure. This allows SCE to better balance preserving local neighborhoods and highlighting large-scale patterns such as clusters.

The key innovation in SCE is the adaptive scale factor, which is optimized through an efficient asynchronous stochastic block coordinate descent algorithm. This method not only improves visualization quality but also makes the algorithm highly scalable and capable of handling large datasets through parallel processing.

2.2.2 SCE Experimental Results and User Study

Yang et al. [14] demonstrated the effectiveness of SCE across a range of real-world datasets. Compared to state-of-the-art methods like t-SNE, LargeVis, and UMAP, SCE consistently produced clearer and more distinct clusters. For instance, in datasets where t-SNE failed to visualize any apparent clusters, SCE was able to reveal distinct groups with clear separations.

Additionally, a user study involving over 300 participants confirmed that the SCE visualizations were more intuitive and better aligned with human perception of clusters than those generated by t-SNE. SCE's adaptive scaling factor plays a critical role in enhancing the clarity of cluster visualizations.

Stochastic Cluster Embedding represents a significant advancement over t-SNE for cluster visualization tasks in high-dimensional data. By generalizing the SNE framework and introducing an adaptive scaling factor, SCE addresses the core limitations of t-SNE, providing clearer and more accurate visualizations of large-scale patterns. This method is particularly useful for applications that require discovering global structures, such as clustering in large datasets.

2.3 Uniform Manifold Approximation and Projection (UMAP)

Uniform Manifold Approximation and Projection (UMAP) is a dimensionality reduction technique introduced by McInnes et al. [8]. UMAP is based on Riemannian geometry and algebraic topology, using fuzzy simplicial sets to model both local and global structures in the data. It is known for being scalable and competitive with t-SNE, while often preserving more global structure.

2.3.1 Mathematical Foundations of UMAP

UMAP assumes that high-dimensional data lies on a manifold $\mathcal{M}$. The algorithm first constructs a weighted k-nearest neighbor graph based on the distances between data points. The local distances for each point x_i are determined by calculating the distance to its nearest neighbors. For each point x_i, the **local radius** ρ_i is defined as:

$$\rho_i = \min\{d(x_i, x_j) \mid 1 \leq j \leq k, d(x_i, x_j) > 0\} \tag{2.7}$$

where $d(x_i, x_j)$ is the distance between points x_i and x_j which is explained in 2.7.

Next, the **fuzzy membership strength** between points x_i and x_j is determined using a smooth exponential kernel in equation 2.8. This membership strength is defined as:

$$w(x_i, x_j) = \exp\left(-\frac{\max(0, d(x_i, x_j) - \rho_i)}{\sigma_i}\right) \tag{2.8}$$

Where σ_i is a scaling factor that normalizes the membership values by ensuring that the sum of the memberships across k-nearest neighbors is:

$$\sum_{j=1}^{k} \exp\left(-\frac{d(x_i, x_j) - \rho_i}{\sigma_i}\right) = \log_2(k) \tag{2.9}$$

This normalization ensures that the resulting fuzzy simplicial set is locally connected.

2.3.2 Optimization of the Low-Dimensional Embedding

UMAP optimizes the low-dimensional representation Y by minimizing the **cross-entropy** between the high-dimensional fuzzy simplicial set and the low-dimensional one. Let $w(x_i, x_j)$ and $\tilde{w}(y_i, y_j)$ denote the fuzzy memberships in the high- and low-dimensional spaces, respectively. The cross-entropy loss function is given by:

$$C(X, Y) = \sum_{i \neq j} \left(w(x_i, x_j) \log \frac{w(x_i, x_j)}{\tilde{w}(y_i, y_j)} + (1 - w(x_i, x_j)) \log \frac{1 - w(x_i, x_j)}{1 - \tilde{w}(y_i, y_j)} \right) \tag{2.10}$$

The fuzzy membership in the low-dimensional space is approximated by:

$$\tilde{w}(y_i, y_j) = \frac{1}{1 + a\|y_i - y_j\|^{2b}} \tag{2.11}$$

Where a and b are hyperparameters that control the embedding layout.

2.3.3 Embedding Initialization

UMAP uses a **spectral embedding** to initialize the low-dimensional coordinates of the points. This is done using the eigenvectors of the **normalized Laplacian** matrix L, which is computed as:

$$L = D^{-1/2}(D - A)D^{-1/2} \tag{2.12}$$

Where A is the adjacency matrix of the graph, and D is the degree matrix. The top d eigenvectors of L are used to initialize the embedding.

UMAPs ability to balance both local and global structure preservation and its efficient optimization and scalability makes it a powerful tool for manifold learning tasks. The mathematical foundation ensures theoretical soundness and practical effectiveness.

2.4 State-of-art dimension reduction techniques

One significant study that contributes to the field of dimensionality reduction (DR) is the work by Wang et al. [10], which provides an in-depth empirical analysis of leading dimensionality reduction techniques such as t-SNE, UMAP, TriMap, and a novel approach, PaCMAP. Wang et al. highlight a major challenge in DR: the trade-off between preserving local and global structure. In visualizing high-dimensional data, these methods often excel in one area but struggle in another, making it difficult to achieve a holistic representation of the data in a lower-dimensional space.

Moreover, it provides detailed insights into the mechanics behind each algorithm and proposes a new approach, PaCMAP, that seeks to balance both local and global structure preservation. Specifically, the study explores how different algorithms handle neighborhood structure and emphasizes the importance of understanding the loss functions that govern the attractive and repulsive forces between data points in DR methods.

2.4.1 Algorithm Comparisons

t-SNE converts pairwise Euclidean distances between high-dimensional data points into conditional probabilities. However, its sensitivity to hyperparameters, particularly perplexity, can lead to spurious clusters that are not present in

the original data. The t-SNE loss function is expressed as:

$$\text{Loss}_{\text{t-SNE}} = \sum_i \sum_j p_{ij} \log\left(\frac{p_{ij}}{q_{ij}}\right)$$

Where p_{ij} and q_{ij} represent the probability distributions of distances in high and low dimensions, respectively.

UMAP uses a force-directed graph layout approach to model the relationships between neighbors. Its optimization process involves pulling neighbors closer in low-dimensional space while repelling distant points. UMAP's loss function is structured as:

$$\text{Loss}_{\text{UMAP}} = \sum_{i,j} w_{ij} \log\left(1 + a\|y_i - y_j\|^2\right)$$

where w_{ij} represents edge weights, and a is a parameter governing the scale of attractive forces.

TriMap [1] takes a triplet-based approach, simultaneously preserving the relative distances between three points. This algorithm excels in maintaining global structure, but it can sometimes struggle with local structure. Its loss function focuses on ensuring that the relative ordering of distances between triplets is preserved:

$$\text{Loss}_{\text{TriMap}} = \sum_{(i,j,k)} \omega_{i,j,k} \frac{s(y_i, y_k)}{s(y_i, y_j) + s(y_i, y_k)}$$

where $s(y_i, y_j) = \left(1 + \|y_i - y_j\|^2\right)^{-1}$ represents similarity, and ω is a weighting function based on the distances in the original space.

PaCMAP (Pairwise Controlled Manifold Approximation Projection), introduced in [10], combines elements from both local and global preservation techniques. By dynamically adjusting the forces applied to neighbors, mid-near pairs, and further points, PaCMAP achieves a more balanced representation. The loss function of PaCMAP is a combination of three terms:

$$\text{Loss}_{\text{PaCMAP}} = w_{NB} \sum_{i,j} \frac{d_{ij}}{10 + d_{ij}} + w_{MN} \sum_{i,k} \frac{d_{ik}}{10000 + d_{ik}} + w_{FP} \sum_{i,l} \frac{1}{1 + d_{il}}$$

where d_{ij} represents the distance between points i and j in the low-dimensional

space, and weights w_{NB}, w_{MN}, and w_{FP} control the contributions from neighbors, mid-near, and further pairs, respectively.

2.4.2 Contributions to Dimensionality Reduction

Wang et al. [10] contribute several key insights to the field of DR.

- First, the analysis reveals that preserving local and global structures simultaneously is crucial for generating accurate visualizations of high-dimensional data.
- Second, the paper presents a set of design principles for DR algorithms, particularly in how to balance the attractive and repulsive forces between points to maintain both types of structure.
- Finally, PaCMAP is introduced as an algorithm that dynamically adjusts its parameters during the optimization process, allowing it to transition smoothly from local to global structure preservation.

In conclusion, this work provides not only a comparative analysis of existing DR techniques but also a novel approach in PaCMAP that addresses long-standing challenges in the field. By combining the strengths of t-SNE, UMAP, and TriMap, PaCMAP offers a solution that balances both local and global structural preservation, which is essential for reliable data visualization.

2.5 Methods classifications

After exploring various dimension reduction techniques, we aim to establish a classification system to differentiate them based on specific criteria outlined in the upcoming sections.

2.5.1 Classification based on the Challenge Addressed

The primary focus of this classification is to identify and address the main challenge that the method aims to solve within the context of the dimension reduction (DR) problem.

- Local Structure Preservation: These methods focus on preserving the relationships between neighboring points in the high-dimensional space. Example: t-SNE, UMAP
- Global Structure Preservation: Methods that focus on maintaining points' overall distribution often aim to preserve the long-range distances between points. Example: TriMap

- Balanced Approach (Local and Global Structure): Newer methods try to preserve both local and global structures simultaneously. Example: PaCMAP

2.5.2 Classification based on Algorithmic Innovation

In this scenario, the methods are categorized based on the primary innovation that has been implemented.

- Graph-Based Methods: Methods that represent the high-dimensional data as a graph, where nodes represent data points and edges represent distances or similarities. Example: UMAP (uses graph construction and optimization over weighted graphs)
- Probabilistic Methods: These methods use probabilistic models to translate the high-dimensional distances into low-dimensional spaces.

  Example: t-SNE (uses a probabilistic framework for mapping high-dimensional data)
- Triplet-Based Methods: Methods that rely on relationships between triplets of points (e.g., distances between a focal point, its neighbor, and a far point).

  Example: TriMap (uses triplet constraints to maintain global structure)
- Hybrid Methods: Methods that combine different innovations or approaches from the categories as mentioned earlier.

  Example: PaCMAP (combines graph-based and pairwise loss approaches to balance local and global structures)

2.5.3 Classification based on Parameter Sensitivity

Another approach to categorizing these methods is based on their sensitivity to hyperparameter tuning. This refers to how much the performance of the methods can be influenced or altered by adjusting the hyperparameters. Some methods may require extensive hyperparameter tuning to achieve optimal performance, while others may be less sensitive and perform well with default or minimal tuning.

- Highly Parameter-Sensitive: Methods that require careful parameter tuning to work effectively and can produce unstable results if not tuned properly.

  Example: t-SNE (highly sensitive to the perplexity parameter)

- Moderately Parameter-Sensitive: Methods that require some parameter tuning but are generally more robust to variations.
  Example: TriMap
- Adaptive Methods: Methods that dynamically adjust parameters during optimization, leading to less reliance on initial settings.
  Example: PaCMAP (automatically adjusts weights between local and global structure preservation)

2.5.4 Classification based on Dimensionality Reduction Goal

This classification centers on determining whether the method is better suited for particular visualization purposes or embedding tasks.

- Data Visualization: Methods primarily designed for visualizing high-dimensional data in 2D or 3D.
  Example: t-SNE, UMAP
- Data Embedding: Methods aimed at embedding high-dimensional data into lower dimensions for analysis, clustering, or classification.
  Example: TriMap
- General Purpose: Methods that are flexible enough to work well for both visualization and embedding tasks.
  Example: PaCMAP

After detailing the various classification methods, we can consolidate them in a hierarchical chart, viewable in Figure 2.1.

2.6 Conclusion

In this chapter, we have reviewed several dimension reduction techniques, including Principal Components Analysis (PCA), t-SNE, Stochastic Neighbor Embedding (SNE), and Uniform Manifold Approximation and Projection (UMAP). Each method offers distinct advantages, particularly in terms of either preserving local or global structures in the data. However, these techniques often struggle with balancing computational efficiency and maintaining the integrity of both local neighborhoods and large-scale patterns in high-dimensional datasets.

Among the explored methods, UMAP is a highly effective tool due to its strong mathematical foundations in Riemannian geometry and algebraic topology. Its ability to construct a fuzzy topological representation of the data and

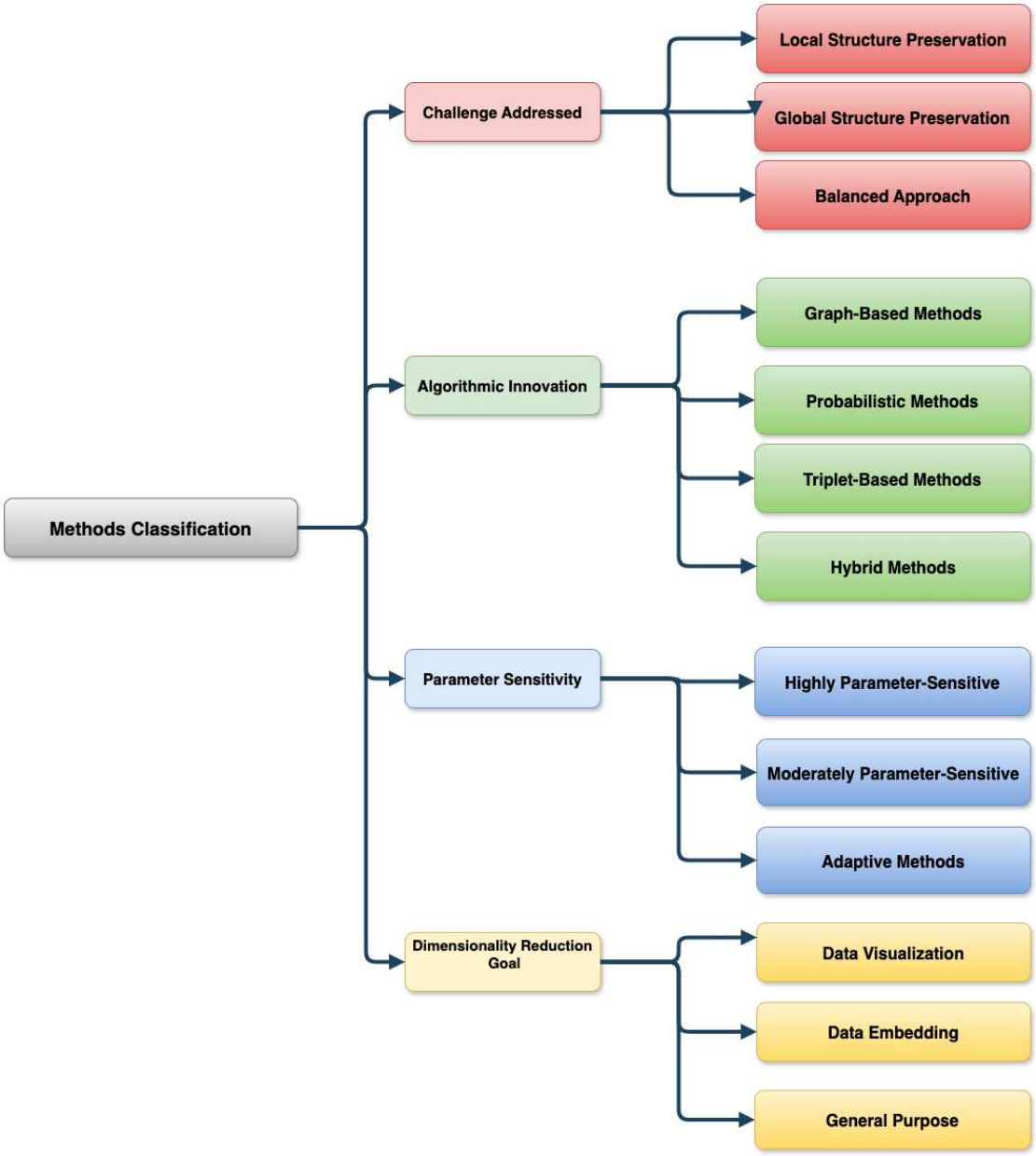

Figure 2.1: Different classification for the dimension reduction (DR) techniques

its optimization techniques allow it to efficiently handle large datasets while preserving both local and global data structures. Unlike t-SNE, which focuses primarily on local neighborhood relationships and often fails to capture global structures, UMAP balances the two. Moreover, UMAP's computational efficiency and scalability make it suitable for large-scale datasets, a key focus of this thesis.

3

Proposed Method

3.1 Introduction

Let us consider a scenario where we are dealing with a substantial number of points, which can be quite expensive to process. Our main objective is to take a high-dimensional object and map it onto a two-dimensional space for visualization purposes. However, when working with a massive dataset containing, for instance, tens of millions points, it may not be feasible to plot all of them due to the limited number of pixels available on the screen.

One potential strategy involves minimizing the computational workload needed to embed the complete set of tens of millions points by carefully selecting a subset of representative data points. These representatives should be chosen in such a way that each point in the original dataset is closely represented. After this selection process, all non-representative points can be discarded. It may be necessary to assign a weight to each representative based on the number of original points it represents. Subsequently, the UMAP algorithm can be applied to these carefully selected points, which we can refer to as "coreset points."

Our initial assumption is that using a smaller dataset will lead to improved performance and reduced run-time due to the reduced data volume. However, it is crucial to verify whether the resulting accuracy meets our requirements. Furthermore, we need to assess the impact on visualization quality when using UMAP, as the selection of K nearest neighbors for each point in UMAP is influenced by the parameter K, which determines the trade-off between local and

global structure representation. If we decide to retain only the coreset points and discard the majority of the points, the K nearest neighbors may extend beyond the local structure, potentially distorting the visualization. We need to investigate whether this distortion occurs and consider potential mitigation strategies, such as incorporating weights. Additionally, we highlighted the importance of evaluating our method against UMAP using appropriate evaluation metrics.

3.2 UMAP as baseline

UMAP will serve as the baseline dimensionality reduction technique for this study due to several factors. First, its computational efficiency allows for handling the large, high-dimensional datasets that are the focus of this thesis, where other methods like t-SNE may falter due to scaling issues. Second, UMAP's ability to preserve both local and global structures ensures that the high-level relationships between data points are maintained, which is critical for effective clustering and visualization.

In this research, UMAP's flexibility in working well with custom distance metrics aligns with the need to experiment with various clustering techniques introduced in Chapter 3. The coreset-based clustering techniques proposed in Chapter 3 and validated in Chapter 4 will benefit from UMAP's properties, making it a robust baseline for comparison in terms of accuracy and efficiency.

By employing UMAP as a baseline, we ensure that the dimensionality reduction is both computationally feasible and theoretically sound, providing a solid foundation for the clustering techniques explored in the following chapters.

3.3 Coreset-based k-Center Clustering

Coreset clustering is an efficient method for solving the k-center clustering problem, particularly useful for handling large datasets. The k-center clustering problem is defined as follows: given a dataset S with n points in a metric space and a parameter k, the goal is to select a subset $T \subseteq S$ of k points (centers) such that the maximum distance between any point in S and its closest center in T is minimized. Formally, the objective is:

$$r_T(S) = \max_{s \in S} \min_{t \in T} d(s, t) \tag{3.1}$$

where $d(s,t)$ is the distance between the points s and t.

This problem is NP-hard, so approximate algorithms are often employed. The key idea behind coreset-based algorithms is to reduce the input size by constructing a smaller subset, called a coreset, which can be used to approximate the solution of the clustering problem on the original dataset [4].

3.3.1 Coreset Construction

A coreset is a small, weighted subset of the input dataset that approximates the properties of the entire dataset with respect to the clustering objective. The coreset size is a function of the desired approximation factor ϵ and the doubling dimension D of the metric space.

The size of the coreset C is bounded by:

$$|C| \leq k \cdot \left(\frac{4}{\epsilon}\right)^{D} \tag{3.2}$$

where D is the doubling dimension of the metric space, which measures how well smaller balls can cover the space.

3.3.2 Algorithm for Coreset-based k-Center Clustering

The coreset-based k-center clustering algorithm is described in Algorithm1:

3.3.3 Final Optimization

The approximation guarantee of the coreset-based k-center clustering algorithm is given by:

$$r_C(S) \leq (2+\epsilon) \cdot r_k^*(S) \tag{3.4}$$

where $r_k^*(S)$ is the optimal radius for the k-center clustering on S, and $r_C(S)$ is the radius obtained by running the algorithm on the coreset C.

3.3.4 Advantages of Coresets

The main advantages of using coresets are:

Algorithm 1 Coreset-based k-Center Clustering

Input: Dataset S, number of centers k, precision parameter ϵ
Output: Set of centers T

1: Partition the dataset S into ℓ subsets: $S_1, S_2, \ldots, S_\ell$
2: **for** each subset S_i **do**
3: Initialize $c_1 \in S_i$ as an arbitrary point
4: **for** $j = 2$ to k **do**
5: Select the next center c_j as:

$$c_j = \arg\max_{s \in S_i} \min_{c \in T} d(s, c) \tag{3.3}$$

6: **end for**
7: Continue until the maximum distance between any point in S_i and its nearest center is less than ϵ times the radius of the current clustering.
8: **end for**
9: Merge the coresets $C_1, C_2, \ldots, C_\ell$ into a single coreset C
10: Apply the k-center algorithm on the coreset C to select the final set of k centers T

- **Efficiency**: By reducing the size of the dataset to a smaller coreset, the algorithm becomes computationally efficient, especially when dealing with large datasets.
- **Scalability**: The method scales well to big data applications, making it feasible to apply clustering to large datasets using distributed systems like MapReduce.
- **Approximation Guarantee**: The coreset-based algorithm provides a provable approximation to the optimal clustering solution, with the approximation factor depending on ϵ.

3.4 Pros and cons of using coreset clustering before using Umap

Among the advantages we can consider:

- Reduced Computational Complexity: Coreset clustering techniques aim to summarize the dataset by selecting a representative subset of points. This reduction in data size can significantly reduce the computational complexity of subsequent processing steps, including dimensionality reduction with UMAP.

- Faster Processing Time: By working with a smaller subset of data, dimensionality reduction techniques like UMAP can run much faster, making it feasible to visualize large datasets in a reasonable amount of time.

- Preservation of Global Structure: Coreset clustering algorithms are designed to preserve the global structure of the dataset. This means that the selected subset of points retains the essential characteristics of the original data, allowing dimensionality reduction techniques like UMAP to capture the underlying relationships between data points.

among the disadvantages we have:

- Loss of Information: Discarding data points to create a corset inevitably leads to some loss of information. Depending on the size and quality of the corset, important features or patterns in the data may be overlooked, potentially affecting the accuracy of the visualization.

- Sensitivity to Coreset Selection: The effectiveness of the coreset clustering approach heavily depends on the choice of parameters and the algorithm used to generate the coreset. Poorly selected parameters or algorithms may result in a coreset that does not accurately represent the original data distribution, leading to biased visualizations.

- Limited Scalability: While coreset clustering can help reduce computational complexity, processing extremely large datasets may still be challenging, especially if the coreset generation step requires significant computational resources.

3.5 Understanding the Impact of Weights on Visualization

Using coreset clustering to handle large, high-dimensional datasets can be an efficient way to manage computational complexity. When it comes to visualizing the data with UMAP or t-SNE, incorporating the weights of the coreset points can indeed influence the accuracy and fidelity of the visualization.

- Weighted Visualization: By applying weights to the coreset points, you can better represent the density and distribution of the original dataset in the reduced-dimensional space. This is particularly important because coresets are essentially representative subsets, and their influence should reflect the original data's distribution.

- Accuracy of Visualization: Weights can improve the accuracy of visualizations by ensuring that regions of higher density in the original dataset are proportionally represented in the low-dimensional embedding. Without weights, the visualization might underrepresent or overrepresent certain clusters, leading to misleading interpretations.

3.5.1 Applying Weights in UMAP and t-SNE

Both UMAP and t-SNE can incorporate sample weights, but their approaches to integrating them differ.

Significance of Weights: Weights reflect the importance of each coreset point. A point with a higher weight indicates that it represents a larger portion of the original dataset. Hence, the distances in the reduced space should reflect these densities.

Weighted Distance: In the context of dimensionality reduction, using weights should influence the distance calculations such that points with higher weights exert more influence on the resulting embedding.

Applying Weights in UMAP: UMAP does not natively support sample weights in its distance metric. However, we can still incorporate weights effectively by adapting the distances through a custom metric. UMAP expects metric functions to take only two arguments, hence a wrapper.

Algorithm 2 UMAP with Weighted Euclidean Distance

Require: Coreset data $X_{\text{coreset}} \in \mathbb{R}^{n \times d}$, weights $w \in \mathbb{R}^n$
Ensure: Low-dimensional embedding $Y \in \mathbb{R}^{n \times p}$

function WEIGHTED_EUCLIDEAN(a, b, w)
    $d \leftarrow \sqrt{\sum_{i=1}^{d} w_i (a_i - b_i)^2}$
    return d
end function
reducer $\leftarrow$ UMAP($metric$ = weighted_euclidean, $metric\_kwds$ = $\{'w' : w\}$)
$Y \leftarrow$ reducer.fit_transform(X_{coreset})
return Y

When using weights to calculate distances for dimensionality reduction techniques like UMAP or t-SNE, the goal is to ensure that the distances reflect the

density and distribution of the original dataset. Simply multiplying the weights in the Euclidean distance calculation might not always be sufficient or appropriate.

A more appropriate method might be to adjust distances by considering the relative importance of each point. This can be done through scaling distances or using a weighted distance metric that inherently considers the density represented by the weights.

3.6 UMAP Clustering

The proposed method can be segmented into three distinct sections for better understanding and organization.

- Data Preparation: We append an index to each data point to easily reference the corresponding weights.
- Custom Metric Function: WeightedEuclideanUMAP uses these indices to look up weights and calculate a weighted Euclidean distance.
- UMAP Initialization: The custom metric is passed to UMAP, enabling it to consider weights during embedding.

UMAP's performance is strongly influenced by the choice of distance metric, as this determines how data point similarity is calculated. We explored various methods for using metrics, creating custom metrics based on the Euclidean distance with added elements such as weights and radii.

3.6.1 weighted euclidean

Explanation: This metric calculates the weighted Euclidean distance between two points. Each point has an associated weight ($weight_a$, $weight_b$), and the average of these weights scales the distance between points. If points are closer but have high weights, the distance is amplified. Usage: Use this metric when each point has a different "importance" or "influence," as the weights represent. It scales distances based on how much weight each point carries.

3.6.2 radius adjusted euclidean

Explanation: This metric introduces the concept of a radius for each point. The distance between two points is inversely scaled by the sum of their radii:

Algorithm 3 weighted euclidean

Input: a, b - two data points, each consisting of a coordinate vector and a weight
Output: Weighted distance d between a and b
Function CustomMetric(a, b)
$point\_a \leftarrow a[:-1]$ ▷ Coordinates of point a
$weight\_a \leftarrow a[-1]$ ▷ Weight of point a
$point\_b \leftarrow b[:-1]$ ▷ Coordinates of point b
$weight\_b \leftarrow b[-1]$ ▷ Weight of point b
$squaredDist \leftarrow$ squaredEuclidean($point\_a, point\_b$)
return $\sqrt{(squaredDist) \cdot \frac{(weight\_a + weight\_b)}{2}}$

points with large radii will appear closer to each other, while points with small radii will appear further apart. Usage: This metric is ideal when each point has a different "influence area" or "spread." For instance, in data with clusters of varying density, points in denser regions can have smaller radii, while points in sparser regions can have larger radii. Why it Works Better: Incorporating radii allows for more nuanced control of distances between points, effectively capturing both local and global relationships in the data. In the case of fashion images, items like Shoes and Bottoms might naturally form denser clusters, while Bags and Tops might be more diverse. The custom metric with radius can account for these varying cluster densities, resulting in better preservation of the underlying structure in the lower-dimensional space.

Algorithm 4 radius adjusted euclidean

Input: a, b - two data points, each consisting of a coordinate vector and a radius
Output: Distance d between a and b, considering radius
Function CustomMetricWithRadius(a, b)
$point\_a \leftarrow a[:-1]$ ▷ Coordinates of point a
$radius\_a \leftarrow a[-1]$ ▷ Radius of point a
$point\_b \leftarrow b[:-1]$ ▷ Coordinates of point b
$radius\_b \leftarrow b[-1]$ ▷ Radius of point b
$squaredDist \leftarrow$ squaredEuclidean($point\_a, point\_b$)
$radiusSum \leftarrow radius\_a + radius\_b$
return $\sqrt{squaredDist} \cdot radiusSum$

3.6.3 RELATIVE WEIGHTED EUCLIDEAN

Explanation: This metric calculates a combination of both weighted distances and Euclidean distance. It divides the Euclidean distance by the weight of one point and multiplies it by the weight of the other point. Usage: This metric can be used in scenarios where there is a more complex relationship between points, combining weights and Euclidean distance in a non-linear way. However, it may not perform well in all cases because it alters the geometry of distances in a more aggressive way.

Algorithm 5 relative weighted euclidean

Input: a, b - two data points, each consisting of a coordinate vector, weight, and radius
Output: Combined distance d between a and b
Function CustomMetricCombination(a, b)
$point\_a \leftarrow a[:-3]$ ▷ Coordinates of point a
$weight\_a \leftarrow a[-2]$ ▷ Weight of point a
$radius\_a \leftarrow a[-1]$ ▷ Radius of point a
$point\_b \leftarrow b[:-3]$ ▷ Coordinates of point b
$weight\_b \leftarrow b[-2]$ ▷ Weight of point b
$radius\_b \leftarrow b[-1]$ ▷ Radius of point b
$squaredDist \leftarrow$ squaredEuclidean($point\_a, point\_b$)
$radiusSum \leftarrow radius\_a + radius\_b$
$weightSum \leftarrow \frac{weight\_a + weight\_b}{2}$
return $\sqrt{(squaredDist \cdot weightSum)} \cdot radiusSum$

3.6.4 WEIGHTED RADIUS ADJUSTED EUCLIDEAN

Explanation: This metric combines both weights and radii for each point in calculating the distance. It scales the squared Euclidean distance by the sum of both weights and radii, which allows it to take into account both the influence (weight) and the spread (radius) of each point. Usage: This metric is useful when both weights and radii are important for determining distances. For example, points with high weights (importance) and large radii (spread) will appear close together, emphasizing their connection. Performance: While this metric is more complex, the combination of weights and radii might sometimes produce less accurate results because it mixes two effects that could distort the original distances between points.

Algorithm 6 weighted radius adjusted euclidean

Input: a, b - two data points, each consisting of a coordinate vector, a weight, and a radius
Output: Weighted and radius-aware distance d between a and b
Function CustomMetricRadiusWeighted(a, b)
$point\_a \leftarrow a[:-3]$ ▷ Coordinates of point a
$weight\_a \leftarrow a[-2]$ ▷ Weight of point a
$radius\_a \leftarrow a[-1]$ ▷ Radius of point a
$point\_b \leftarrow b[:-3]$ ▷ Coordinates of point b
$weight\_b \leftarrow b[-2]$ ▷ Weight of point b
$radius\_b \leftarrow b[-1]$ ▷ Radius of point b
$squaredDist \leftarrow$ squaredEuclidean($point\_a, point\_b$)
return $\sqrt{\left(\frac{squaredDist \cdot (weight\_a + weight\_b)}{2}\right) \cdot (radius\_a + radius\_b)}$

3.6.5 Why the "Custom Metric with Radius" Has Better Results

Adaptive Distances: The custom metric with radius scales distances between points based on their radii, effectively making the algorithm more sensitive to clusters of different densities. In datasets like fashion MNIST, where different categories (e.g., shoes, tops, bottoms) may form clusters of varying density, this adaptability helps preserve the overall data structure better. Cluster Preservation: Points that are part of denser clusters (with smaller radii) will have their distances calculated with more precision, while points in sparser clusters (with larger radii) will be more loosely connected. This helps UMAP maintain both local relationships (within clusters) and global relationships (between clusters) more accurately. Balance Between Global and Local Structure: UMAP is designed to balance global and local data structure preservation. By introducing radius into the metric, the algorithm can better capture global features of the dataset (e.g., clusters of items like bags and shoes that might be farther apart in the original space). The reason the custom metric with radius shows better results is its ability to account for variations in the density and spread of clusters in the dataset. It adapts to different scales of data better than other metrics, leading to improved accuracy in representing high-dimensional data in a lower-dimensional space. This explains why it outperforms the other custom metrics, especially for datasets like fashion MNIST where categories have inherently different structures.

if we use the default rta not the custom, we are computing the random triplet

accuracy over the entire points so It means we are taking a sample over all the points in evaluating the accuracy. If we consider the only points in the coreset we are considering a very small subset. if the embedding is a bit unlucky and the baseline makes a bad embedding for those points then the performance is bad. because it's more difficult and you don't see the extremes but the average behavior. If you consider all the points then the average behavior is a bit better.

This was a good thing to observe, and mainly, this means that evaluating the triplet accuracy only on the points of the coreset is not a good idea. It's a little bit unfair for the baseline. So, for the baseline, we use the umap and then apply random triplet accuracy. But it leaves us with the question of how to evaluate the corset with our customized metrics because it only works with a subset of the entire dataset, and we are not evaluating it on the same thing. One possibility is to say that we don't care, but then we do not compute the right thing. The other possibility is to say that after training the umap embedding on the coreset which means we create the umap object (umap fit with the custom metric and other parameters and then call the transform but ... and it's much faster than from scratch.) So what we can do is, after training on the coreset, we can transform the entire original dataset and then compute the random triplet loss on the entire dataset but embedded in the model trained on the coreset. Then we are comparing more fair.

instead of using a custom metric function let's use another feature of umap API that allows us to pass to the umap object or instead of passing the data metrics you pass the metric of pairwise distances between the points. So, passing a matrix of n*n, represents the distance between two corset points. what we gain by passing the distance metrics instead of raw data is now we are thinking of computing the custom metric using numpy, avoiding the for loops. So we can try to use the same tickss that umap is using, so looking at things like matrix multiplications to avoid for loops, etc. We will do them in our code then we take the result of these distance computations so we have this square metrics and we pass it to the fit method. and this should make things faster. we have to find a way to implement this function basically instead of pair of points for entire set of points in terms of matrix multiplication. To optimize the work don't do it for the all the custom metrics.

3.7 Complexity Analysis

To analyze the time complexity of the proposed method, we'll need to break down to its different components. The complexity will depend on various factors, such as dataset size, the number of points (n), the dimensionality of the data (d), the number of centers (k), and the number of neighbors ($n_{\text{neighbors}}$) used in UMAP. Below is an analysis of the time complexity for different parts of the technique.

3.7.1 Coreset Selection

The `kCenterFFT` function selects k centers from n data points.

- **Initialization**: Picking a random center takes $O(1)$ time.
- **Distance calculation**: In each iteration, the function computes the squared distance between each point and the selected center, which requires $O(n \cdot d)$ time per iteration.
- **Center selection loop**: In each iteration, the algorithm selects the next center by finding the point farthest from the nearest selected center. This process is repeated k times, and in each iteration, the algorithm updates the distances for all n points.

Thus, the overall complexity of `kCenterFFT` is:

$$O(k \cdot n \cdot d)$$

3.7.2 Weight Computation

- **Loop over points**: For each of the n points, the algorithm computes the distance to all k centers.
- **Distance calculation**: Each distance computation takes $O(d)$.

Thus, the overall complexity of `computeWeights` is:

$$O(n \cdot k \cdot d)$$

3.7.3 Radius Computation

- **Loop over centers**: For each of the k centers, the algorithm computes the distance to all n points.

- **Distance calculation**: Each distance calculation takes $O(d)$.

Thus, the overall complexity of `computeRadius` is:

$$O(n \cdot k \cdot d)$$

3.7.4 Distance Matrix Computation

This function computes the distance matrix for the k centers using a custom metric (`custom_metric_with_radius`).

- **Pairwise distance calculation**: The number of pairwise comparisons is $O(k^2)$, and each comparison using the custom metric involves a distance computation of two points in d dimensions, which takes $O(d)$ time.

Thus, the overall complexity of `compute_distance_matrix` is:

$$O(k^2 \cdot d)$$

3.7.5 UMAP Embedding

The complexity of UMAP depends on the number of data points, the number of neighbors, and the dimensionality of the embedding space.

- UMAP typically has a complexity of $O(n \cdot n_{\text{neighbors}} \cdot d)$, but since we are using a precomputed distance matrix with k centers, the complexity reduces to:
$$O(k \cdot n_{\text{neighbors}} \cdot 2)$$
where the factor of 2 accounts for the 2D embedding space.

Thus, the overall complexity for UMAP is:

$$O(k \cdot n_{\text{neighbors}})$$

3.7.6 Triplet Evaluation Using `random_triplet_eval(X, X_new, y)`

- **Triplet generation**: For each point, the algorithm generates 5 random triplets, which requires $O(n)$.
- **Distance calculation**: For each triplet, the distance between points is computed in both the original and new spaces, which takes $O(d)$ for each distance calculation.

Thus, the overall complexity of `random_triplet_eval` is:

$$O(n \cdot d)$$

3.7.7 Overall Complexity

To summarize the dominant terms:

- **Coreset selection**: $O(k \cdot n \cdot d)$
- **Distance matrix computation**: $O(k^2 \cdot d)$
- **UMAP embedding**: $O(k \cdot n_{\text{neighbors}})$
- **Triplet evaluation**: $O(n \cdot d)$

Given that $n \gg k$ in most cases, the coreset selection and UMAP embedding steps dominate the time complexity. Therefore, the overall time complexity of the code is:

$$O(k \cdot n \cdot d + k^2 \cdot d + k \cdot n_{\text{neighbors}} + n \cdot d)$$

If we assume that k is much smaller than n, the leading term is $O(n \cdot d)$, which corresponds to operations involving the full dataset.

4

Implementation and Validation

4.1 Introduction

In this chapter, we describe the methodologies employed in this research to analyze high-dimensional datasets. We first discuss the preprocessing steps applied to different types of datasets, followed by an explanation of the k-Center FFT algorithm used for selecting a coreset. Finally, we delve into the UMAP algorithm and its custom distance metric, which were utilized to generate lower-dimensional embeddings for visualization and evaluation.

4.2 Preprocessing of Datasets

Preprocessing is a critical step in any data analysis pipeline. The datasets considered in this study include the WISDM dataset [11], Fashion MNIST [13], and a household power consumption [6] dataset. Each dataset requires specific preprocessing steps to ensure the data is in the correct format for further analysis. In the following sections we will discuss them more in depth.

4.2.1 Fashion MNIST Preprocessing

The Fashion MNIST dataset contains images of fashion items, represented as numerical vectors. The dataset has the training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28×28 grayscale image, associated (in total 724 pixels) with a label from 10 classes.The preprocessing involves reading

the dataset from a CSV file, extracting the group labels, and converting the data into a numerical format suitable for further analysis. The dataset is then stored as a NumPy array of type `float32`.

4.2.2 Power Consumption Dataset Preprocessing

The household power consumption dataset consists of various electrical measurements recorded over time. This dataset requires cleaning to remove any missing values, followed by conversion into a numerical array. Moreover, the date feature should be removed since we are analyzing numerical values only. Since no group labels are provided, the analysis proceeds without them but we can take one of the features as the label also for better visualizing data later on graphs. Overall, it has 10 features and 2075259 instances.

4.2.3 WISDM Dataset Preprocessing

The WISDM Smartphone and Smartwatch Activity and Biometrics Dataset is a comprehensive dataset that focuses on human activity recognition (HAR) and biometrics using smartphone and smartwatch sensor data. It was collected by the Wireless Sensor Data Mining (WISDM) Lab at Fordham University. The raw data has 6 features and 15630426 samples.

4.3 Coreset Selection Algorithm

To think about clustering as dividing data points Data into k clusters, we will try to select a set Centers of k points that will serve as the centers of these clusters. We would like to choose Centers so that they minimize some distance function between Centers and Data over all possible choices of centers.

The k-Center FFT algorithm is employed to select a representative subset, or coreset, from the entire dataset. The algorithm is efficient and scales well with the size of the dataset. The k-Center FFT algorithm operates as follows:

1. **Initialization:** A random point is selected as the first center.
2. **Distance Calculation:** For each point in the dataset, the distance to the nearest center is calculated.
3. **Center Selection:** The point farthest from its nearest center is selected as the next center.

4. **Repeat:** Steps 2 and 3 are repeated until the desired number of centers (k) is selected.

This algorithm ensures that the selected centers are spread out, providing a good approximation of the dataset's structure.

4.4 Distance Metrics

A key aspect of this research is the use of a custom distance metric, particularly when dealing with high-dimensional data. The metric used is a modified version of the Euclidean distance, where each point is associated with a radius. The distance between two points a and b is calculated using the following formula:

$$\text{distance} = \sqrt{\frac{\sum(a_i - b_i)^2}{2 \cdot (\text{radius}_a + \text{radius}_b)^2 + 1}}$$

This metric accounts for the relative importance of each point based on its radius, which is determined by the k-Center FFT algorithm. The reason of choosing

4.5 UMAP for Dimensionality Reduction

UMAP is a dimensionality reduction technique used to project high-dimensional data into a lower-dimensional space for visualization. UMAP preserves the local structure of the data, making it particularly useful for clustering and classification tasks.

In our thesis, UMAP is used in conjunction with the custom distance metric to generate 2D embeddings of the selected coresets. The UMAP algorithm is configured to use the precomputed distance matrix, calculated using the custom metric, and the number of nearest neighbors is set to n.

In the next sections We will discuss more the performance of this customization and compare the results.

4.6 Experimental proof of using radius adjusted Euclidean Metric

So far, we have discussed 4 different metrics, each with its own potential to improve the performance of our proposed method. In this section, we will compare the results of each metric tested on the same dataset, Fashion MNIST train set. Additionally, all other conditions, such as number of neighbors and coreset size, remain constant. You can find the results of the experiment in table 4.1.

Table 4.1: clustering umap with different custom metrics experimented on fashion mnist train set

Metrics Performance			
Metric	Number of Neighbors	Accuracy	Coreset Size
weighted Euclidean	15	0.587	200
radius adjusted Euclidean	15	0.729	200
weighted radius adjusted Euclidean	15	0.517	200
relative weighted Euclidean	15	0.502	200

Moreover, in the Figure4.1 you can see the performance of each of the metrics for fashion mnist train set.

We conducted experiments using the same conditions for all metrics. First, we analyzed the baseline results, which displayed all clustered points. It was evident that increasing the number of data points could make visualization more challenging, so using the k-center technique seemed like a good idea. We then examined four other plots showing the impact of metrics and coreset clustering on the baseline. We used 15 neighbors for UMAP and set the number of coresets to 200, aiming to focus solely on metric performance without biasing ourselves with these parameters.

The relative weighted Euclidean metric showed the worst performance, even without considering accuracy. Additionally, the weighted radius adjusted Euclidean metric led to widely spread blue points. Comparing the first two metrics (weighted Euclidean and radius adjusted Euclidean) was difficult, so we referred to the table4.1 and found that the radius adjusted metric performed the best.

In summary, we will use this metric for future experiments to concentrate on parameter tuning and to assess if our proposed technique is satisfactory

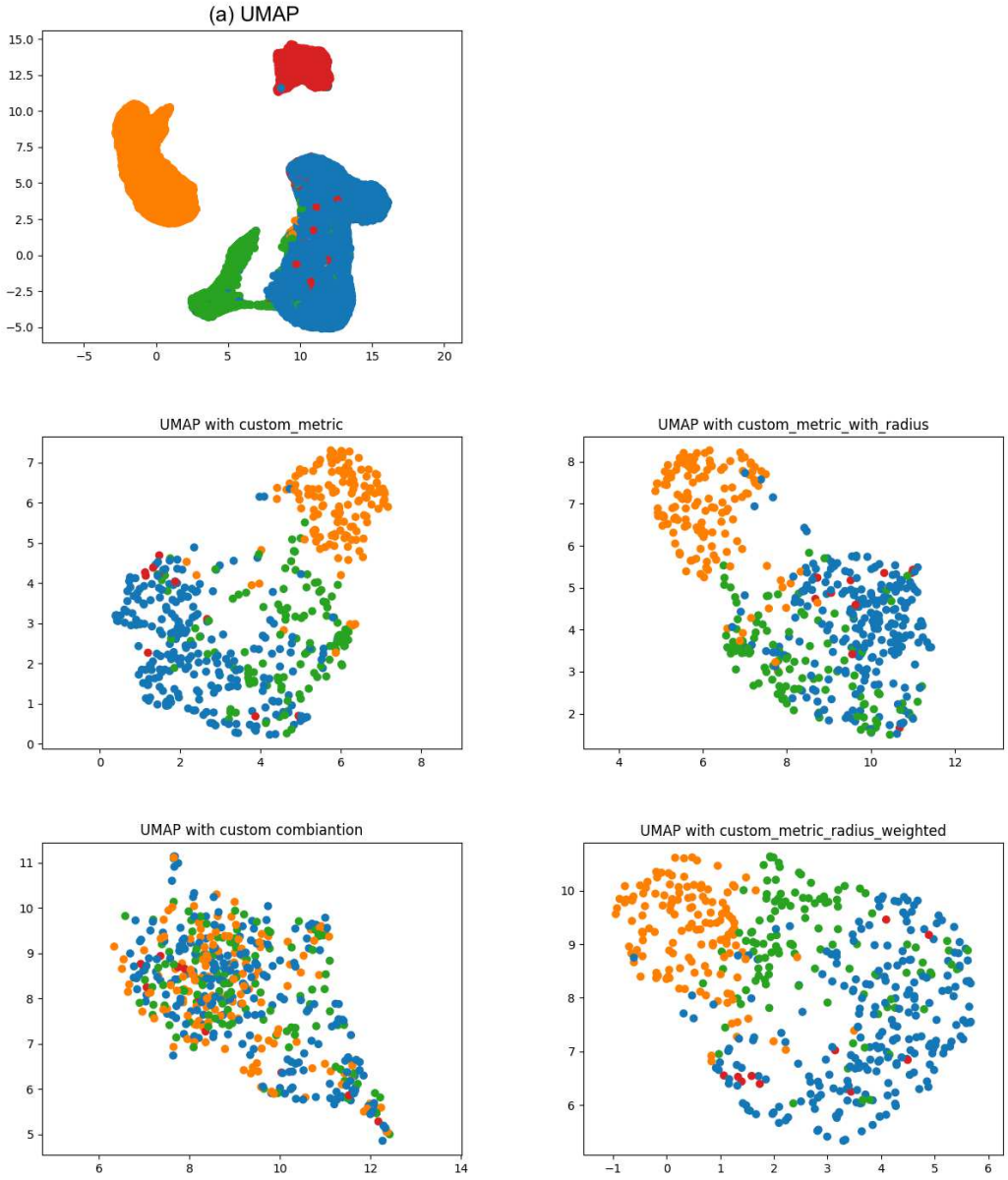

Figure 4.1: visualization of the fashion mnist train set using clustering umap with different custom metrics

compared to the UMAP baseline method.

4.7 Experimental Setup

The experimental setup involves several steps:

1. **Data Loading and Preprocessing:** Depending on the dataset, appropriate preprocessing steps are applied to obtain a clean, numerical representation of the data.
2. **Coreset Selection:** The k-Center FFT (farthest first traversal) algorithm is used to select a coreset from the dataset.
3. **Distance Matrix Calculation:** A distance matrix is computed using the custom metric, where each point is represented by its coordinates and radius.
4. **UMAP Embedding:** UMAP is applied to the distance matrix to obtain a 2D embedding of the coreset.
5. **Evaluation:** The quality of the embedding is evaluated using random triplet accuracy, which compares the relative distances between points in the original and reduced spaces.

4.8 Validation of the Proposed Method

The paper [10] evaluates the performance of various Dimension Reduction (DR) algorithms in preserving both local and global structure in the data using the following metrics and methodology:

Local Structure Preservation Metrics

- KNN Accuracy: Leave-one-out cross-validation with the K-Nearest Neighbors (KNN) classifier is used to assess how well the DR methods preserve neighborhood relationships in the data. The intuition is that labels tend to be similar in small neighborhoods, so the KNN classification accuracy should remain close to the high-dimensional space if local structure is well-preserved. Hyperparameter tuning is performed for the number of neighbors k.
- SVM Accuracy: The accuracy of nonlinear Support Vector Machine (SVM) models with an RBF kernel is measured using 5-fold cross-validation. This metric, similar to KNN accuracy, measures the cohesiveness of neighborhoods in a more flexible manner that is less impacted by data density. The embedding is partitioned into 5 folds, with 4 folds used for training the SVM model and the remaining fold for evaluating accuracy. The Nystrom method is used to approximate the kernel matrix and reduce runtime.

Global Structure Preservation Metrics

- Random Triplet Accuracy: This metric measures the percentage of triplets whose relative distances in the high- and low-dimensional spaces maintain their relative order. Due to computational constraints, a sample of triplets is used rather than considering all triplets. The metric is applied five times, and the mean and standard deviation are reported. Random triplet accuracy does not require labels, allowing evaluation on unlabeled datasets.
- Centroid Triplet Accuracy [1]: For labeled datasets, the centroids of each class are computed in both the high- and low-dimensional spaces. Triplets are constructed using the relative distances between centroids in the high-dimensional space, and the percentage of preserved centroid triplets is reported.

UMAP baseline is a powerful dimensionality reduction technique known for its ability to preserve both local and global structures in high-dimensional data. However, when applying k-center clustering, a significant number of data points are removed, leading to a stronger emphasis on global rather than local structure. Consequently, we opted to employ the Random Triplet Accuracy metric to thoroughly analyze and validate the proposed technique. Given UMAP's proficiency in capturing global structure, this validation approach is deemed to be sufficiently thorough and fair.

4.9 Results

The results, including the dataset name, coreset size, number of neighbors, metric used, computation time, and triplet accuracy, are stored in a CSV file for later analysis. This allows for easy comparison across different datasets and parameter settings.

In the previous sections, we explained that we will be presenting the following results based on UMAP as our baseline. First, we need to determine the running time and accuracy for each dataset. The table4.2 will provide the data from the experiments conducted using the baseline technique:

Table 4.2: clustering umap with different custom metrics experimented on fashion mnist train set

UMAP baseline accuracy		
Dataset	Running Time (seconds)	Accuracy
fashion-mnist	182	0.7392
Power Consumption	1969	0.789
WISDM	1850	0.6316

Now for each of the experimented datasets we provided the separated graphs in order to compare and better visualize the baseline with the proposed technique. In these experiments we tested our system with different number of neighbors n and size of coreset k.

In Figure4.2 UMAP as the baseline, and four different parameter tuning are shown for WISDM dataset. The parameters tuned differently as follows:

- coreset size = 1500, number of neighbors = 15
- coreset size = 1000, number of neighbors = 15
- coreset size = 500, number of neighbors = 15
- coreset size = 100, number of neighbors = 15

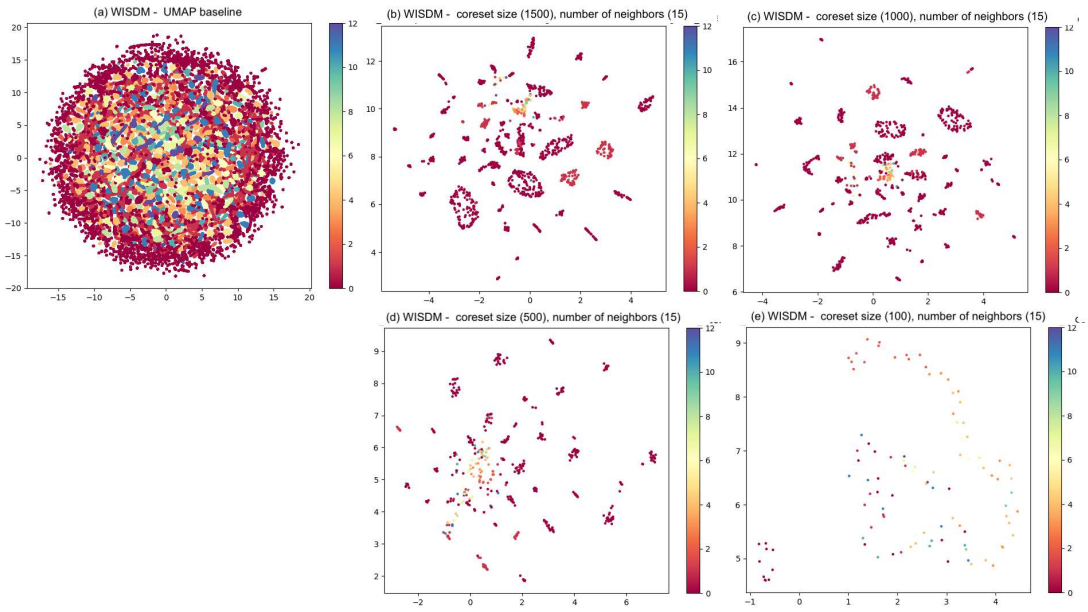

Figure 4.2: WISDM Dataset - A comparing results within the UMAP technique as baseline and 4 different coreset sized UMAP clustering technique

In Figure4.3, The same way of experiment is shown for Power Consumption Dataset.

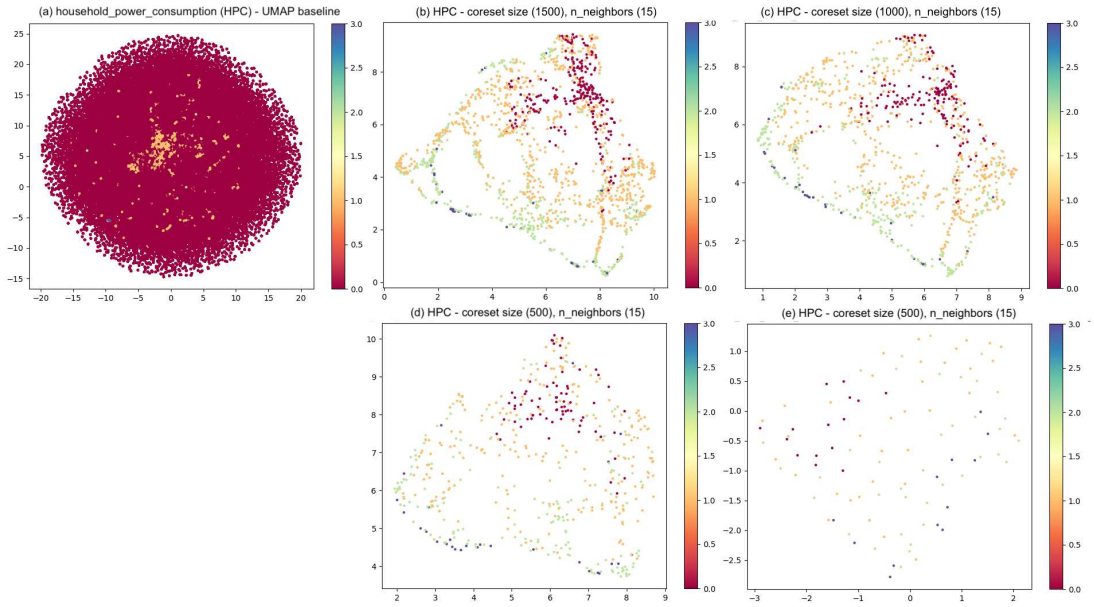

Figure 4.3: Power Consuption Dataset - A comparing results within the UMAP technique as baseline and 4 different coreset sized UMAP clustering technique

In Figure4.4, with that same parameter tuning we showed the clustering for the Fashion MNIST dataset.

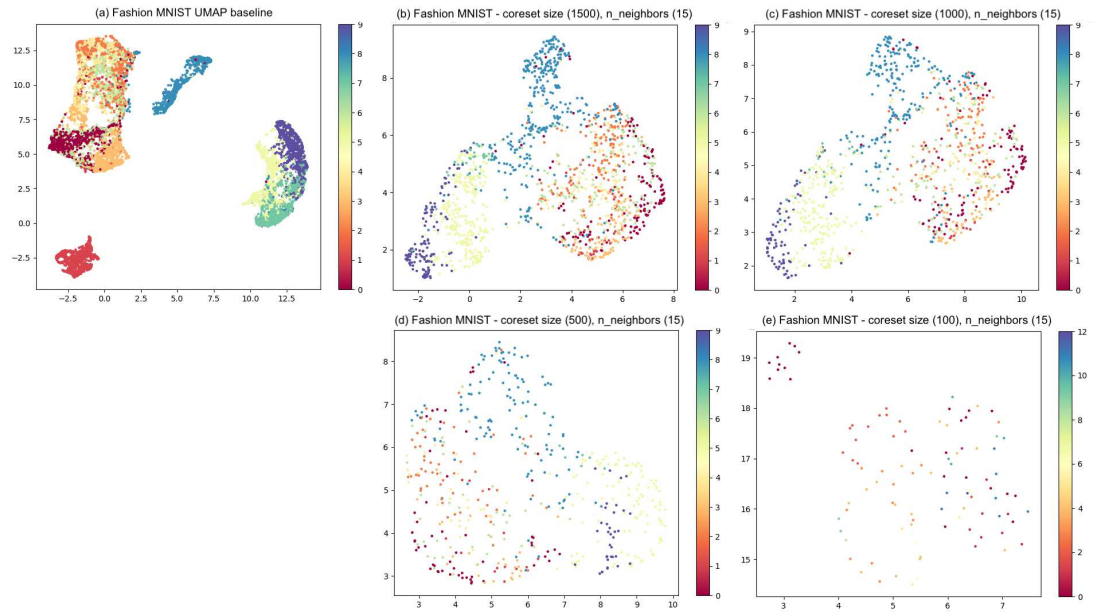

Figure 4.4: Fashion MNIST Dataset - A comparing results within the UMAP technique as baseline and 4 different coreset sized UMAP clustering technique

In table4.3 all the results of the experiments based on different parameters and datasets are represented. Based on that result we can conclude the following:

- For the Fashion MNIST dataset, the highest accuracy achieved is 0.7414, which corresponds to a coreset size of 1000 and a number of neighbors of 5.
- For the Power Consumption dataset, the highest accuracy achieved is 0.8544, which corresponds to a coreset size of 500 and a number of neighbors of 15.
- For the WISDM dataset, the highest accuracy achieved is 0.7680, which corresponds to a coreset size of 100 and a number of neighbors of 5.

Table 4.3: Experiments result based on different parameter tuning for diffent dataset

Results				
Dataset	Coreset Size	Number of Neighbors	Running Time	Accuracy
fashion-mnist	1500	15	42.0664	0.7392
fashion-mnist	1500	10	41.6848	0.7341
fashion-mnist	1500	5	41.9219	0.7203
fashion-mnist	1000	15	37.6964	0.7354
fashion-mnist	1000	10	27.4527	0.7250
fashion-mnist	1000	5	27.2900	0.7414
fashion-mnist	500	15	14.2053	0.7372
fashion-mnist	500	10	14.3026	0.7144
fashion-mnist	500	5	18.1329	0.7156
fashion-mnist	100	15	32.8341	0.7180
fashion-mnist	100	10	33.7102	0.7080
fashion-mnist	100	5	31.9246	0.6560
Power Consumption	1500	15	502.3609	0.8380
Power Consumption	1500	10	549.3834	0.8447
Power Consumption	1500	5	550.6811	0.8344
Power Consumption	1000	15	331.5609	0.8292
Power Consumption	1000	10	342.4512	0.8316
Power Consumption	1000	5	373.7951	0.8188
Power Consumption	500	15	169.2621	0.8544
Power Consumption	500	10	165.9974	0.8376
Power Consumption	500	5	193.0581	0.8276
Power Consumption	100	15	45.1727	0.8060

Continuation of Table 4.3				
Dataset	Coreset Size	Number of Neighbors	Running Time	Accuracy
Power Consumption	100	10	35.8779	0.8020
Power Consumption	100	5	37.7525	0.7980
WISDM	500	15	165.4590	0.6316
WISDM	500	10	162.3355	0.6136
WISDM	500	5	155.4638	0.5428
WISDM	100	15	43.4706	0.7260
WISDM	100	10	34.6703	0.7080
WISDM	100	5	35.8167	0.7680
WISDM	1500	15	465.9893	0.5756
WISDM	1500	10	482.5490	0.5792
WISDM	1500	5	472.4267	0.5680
WISDM	1000	15	306.3049	0.5970
WISDM	1000	10	308.3149	0.6020
WISDM	1000	5	317.6600	0.5438

It is the time to analyze and contrast the specific and overall traits of each experiment by presenting easily understandable graphs.

In Figure 4.5, we can observe that the size of the coreset does not significantly affect the performance across all datasets. This indicates that the accuracy of our proposed technique is not dependent on the number of coresets. This is a positive outcome as it shows that the method has been improved in terms of running time. When working with a higher number of points, the running time would certainly increase. Therefore, our method can effectively support reduced running times.

Figure 4.6 proves our point and illustrates how the size affects the running time. Therefore, we should opt for a smaller size for the k-center clustering part.

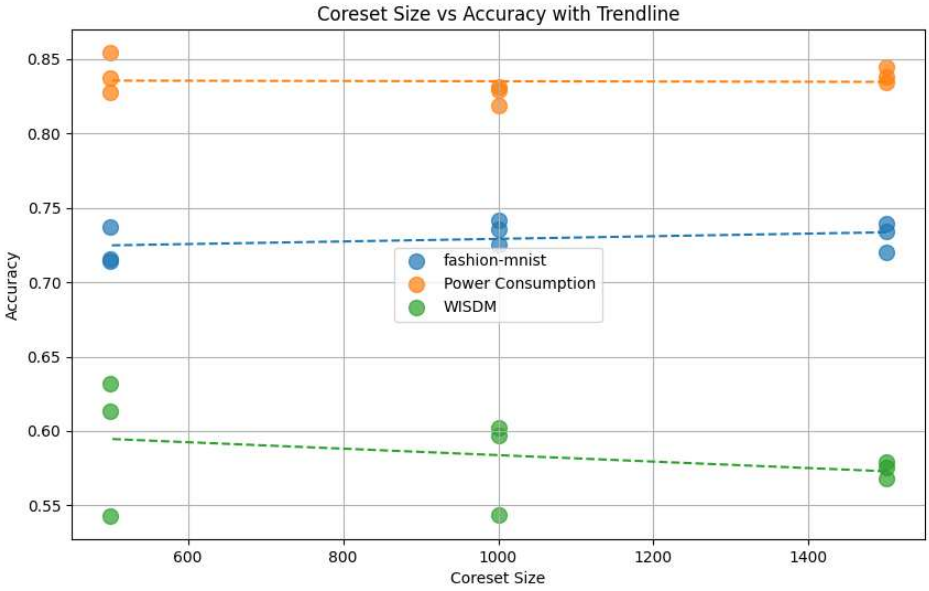

Figure 4.5: coreset vs accuracy

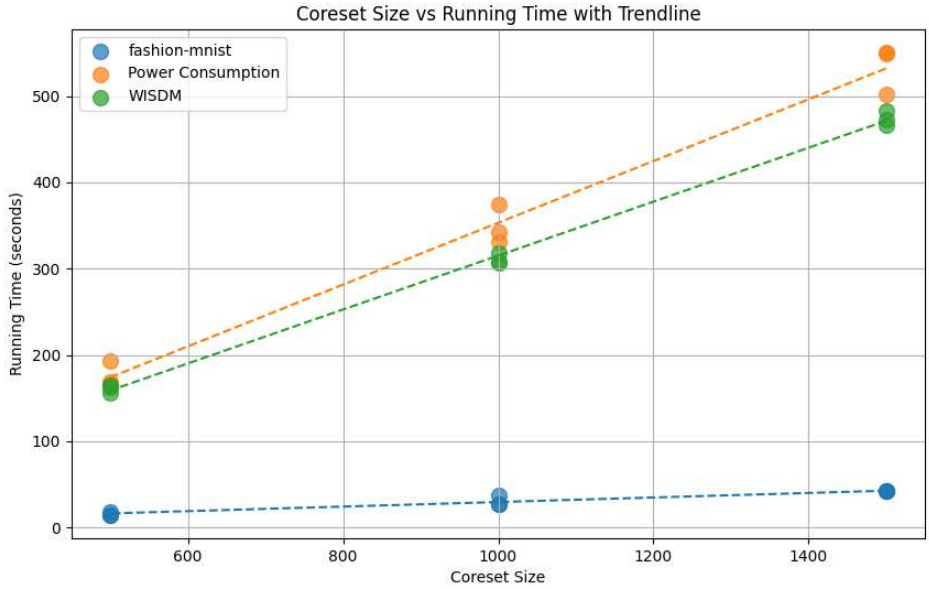

Figure 4.6: relation between the running time and coreset size

In Figure 4.7 the scatter plot shows Running Time vs Accuracy. The color of each point indicates the Number of Neighbors.

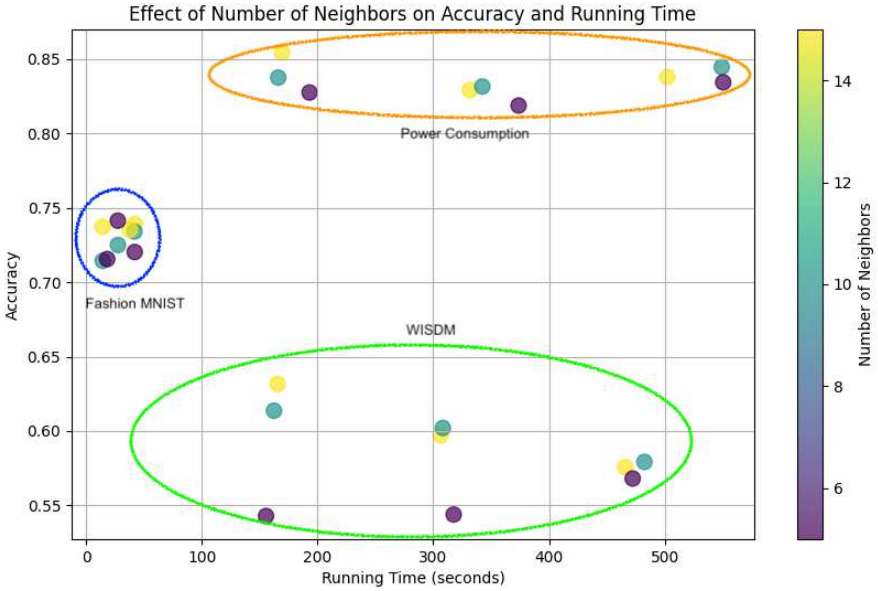

Figure 4.7: the effect of the number of neighbors on the accuracy

Figure 4.8 combines a boxplot and a KDE (Kernel Density Estimate), showing both the range and the distribution (density) of the values. In this way, we have the knowledge of how the accuracy is distributed for each number of neighbors, which is split by dataset.

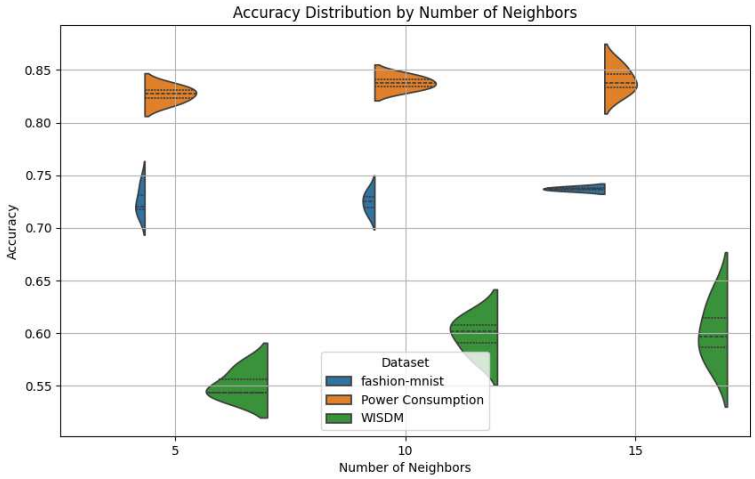

Figure 4.8: the distribution of accuracy and running time for each number of neighbors or coreset size

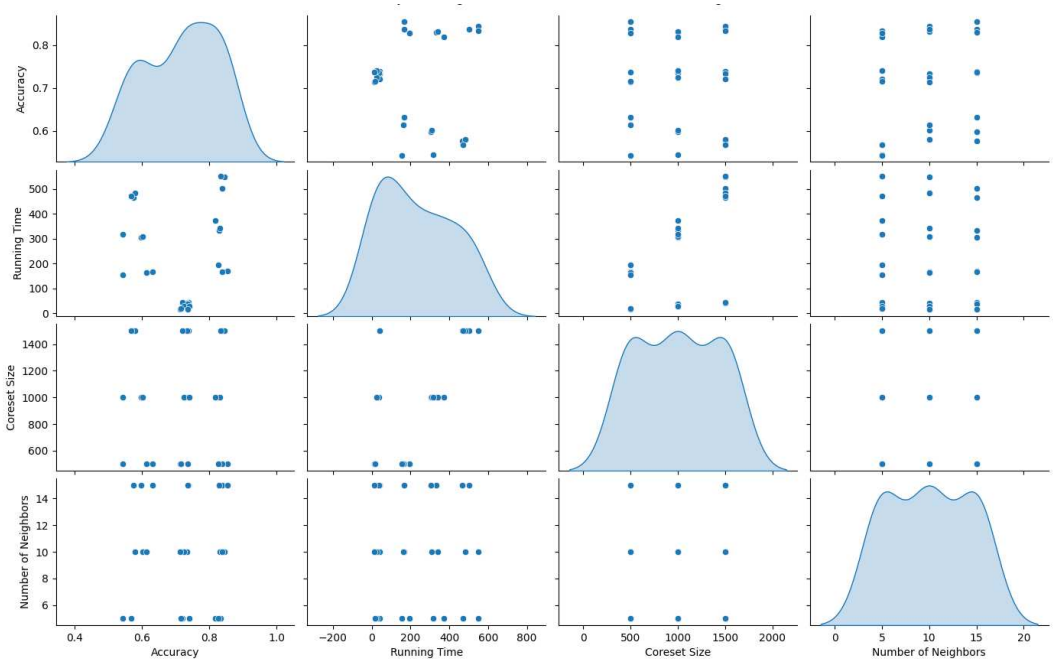

Figure 4.9: The plot shows pairwise scatter plots of accuracy, running time, coreset size, and number of neighbors.

In Figure 4.9 pairwise relationships in the dataset are shown. It is a matrix of scatterplots, showing every combination of variables such as accuracy, running time, coreset size, number of neighbors against each other. The diagonal shows KDE (kernel density estimates) for each variable to visualize the distribution so we can explore all relationships simultaneously.

4.10 Conclusion

In this chapter, we provide a comprehensive overview of the methodologies and implementations. The discussion includes detailed insights into the pre-processing steps, the k-Center FFT algorithm, a custom distance metric, and the UMAP algorithm. Additionally, we thoroughly explain the overall experimental setup. Finally, we compare the results obtained from different parameter tuning methods and find that the proposed method is significantly better than using UMAP as the baseline in terms of complexity and running time. Based on the results, we can choose the method with better Random Triplet Accuracy based on the dataset used.

5

Conclusion and Future Work

The research explored innovative methodologies to address the challenges of large-scale high-dimensional data, focusing on dimensionality reduction and data visualization through UMAP and k-center coreset clustering techniques. By comparing baseline UMAP with custom metrics applied to selected coresets, this study advanced the state of knowledge in terms of efficiency, accuracy, and visualization quality in high-dimensional data embedding.

5.1 Key Findings

The experiments, particularly in Chapter 4, revealed that coreset-based clustering techniques, when combined with custom distance metrics such as the "custom metric with radius," yield competitive results while significantly reducing computational time. A major breakthrough from this study was the ability to efficiently handle large datasets, such as Fashion MNIST, WISDM, and Power Consumption, without significant loss in accuracy.

For instance, the Fashion MNIST dataset demonstrated high accuracy (up to 0.7414) with a relatively small coreset size of 1000 points and 5 neighbors, proving that the approach can maintain both accuracy and computational efficiency. The coreset-based approach outperformed the baseline UMAP in terms of running time without heavily sacrificing accuracy. This finding is significant as it confirms the hypothesis that coresets can be employed to reduce computational overhead in large-scale dimensionality reduction.

Similarly, the WISDM and Power Consumption datasets showed that increasing coreset size led to better accuracy, though the gains diminished beyond a certain point. Results indicated that UMAP with custom metrics offered an optimized balance between local and global data structures, preserving both clusters and relationships within datasets.

5.2 Challenges

Despite these promising results, challenges remain, particularly regarding the trade-offs between coreset size, accuracy, and running time. Larger coresets often lead to better accuracy, but at the cost of increased computation time. This trade-off is crucial in real-world applications where scalability and performance are equally important. Further, the custom metrics demonstrated variability across datasets, indicating that no single metric outperforms in all cases.

Moreover, as shown in Chapter 4, tuning parameters such as the number of neighbors (k) significantly impacts performance, and careful tuning is necessary depending on the dataset and its complexity.

5.3 Future Directions

Several future directions emerge from this work. First, developing adaptive algorithms that can automatically select optimal coreset sizes and metrics for a given dataset can enhance both the scalability and ease of use of this methodology. Automated parameter tuning, particularly for k-neighbors and metric selection, will be critical in applying these techniques in real-time systems where datasets are continuously evolving.

Second, exploring extensions of custom distance metrics beyond the Euclidean space would be beneficial. The introduction of more complex metrics tailored to specific domains, such as cosine similarity for text data, could improve performance for domain-specific tasks like natural language processing and image recognition .

Lastly, integrating this approach with neural networks, such as by leveraging coreset-based clustering in convolutional neural networks (CNNs), could allow for further advances in real-time image classification and object detection tasks. The techniques developed in this thesis could be enhanced by considering graph-

based methods for preserving relationships between points in dynamic, non-Euclidean spaces.

5.4 Conclusion

This research contributes significantly to the field of dimensionality reduction, particularly for large-scale high-dimensional data, by introducing a novel combination of coresets, custom metrics, and UMAP. The findings illustrate that efficient, scalable solutions to dimensionality reduction are attainable without substantial loss in accuracy. These methods have the potential to impact a wide range of fields, from data science to real-time systems, where quick and accurate embeddings of high-dimensional data are critical.

Future research should focus on refining these techniques and extending their applicability to broader domains, including natural language processing, real-time systems, and interactive visualization platforms. The promising results from this work lay a foundation for further advancements in scalable dimensionality reduction techniques.

References

[1] Ehsan Amid and Manfred K Warmuth. "TriMap: Large-scale dimensionality reduction using triplets". In: *arXiv preprint arXiv:1910.00204* (2019).

[2] Miguel A Carreira-Perpinán. "A review of dimension reduction techniques". In: *Department of Computer Science. University of Sheffield. Tech. Rep. CS-96-09* 9 (1997), pp. 1–69.

[3] J Douglas Carroll and Phipps Arabie. "Multidimensional scaling". In: *Measurement, judgment and decision making* (1998), pp. 179–250.

[4] Matteo Ceccarello, Andrea Pietracaprina, and Geppino Pucci. "Solving k-center Clustering (with Outliers) in MapReduce and Streaming, almost as Accurately as Sequentially". In: *arXiv preprint arXiv:1802.09205* (2018).

[5] Mayur Datar et al. "Locality-sensitive hashing scheme based on p-stable distributions". In: *Proceedings of the twentieth annual symposium on Computational geometry*. 2004, pp. 253–262.

[6] Georges Hebrail and Alice Berard. *Individual Household Electric Power Consumption*. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C58K54. 2006.

[7] Piotr Indyk and Rajeev Motwani. "Approximate nearest neighbors: towards removing the curse of dimensionality". In: *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. 1998, pp. 604–613.

[8] Leland McInnes, John Healy, and James Melville. "Umap: Uniform manifold approximation and projection for dimension reduction". In: *arXiv preprint arXiv:1802.03426* (2018).

[9] Laurens Van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE." In: *Journal of machine learning research* 9.11 (2008).

[10] Yingfan Wang et al. "Understanding how dimension reduction tools work: an empirical approach to deciphering t-SNE, UMAP, TriMAP, and PaCMAP for data visualization". In: *Journal of Machine Learning Research* 22.201 (2021), pp. 1–73.

[11] Gary Weiss. *WISDM Smartphone and Smartwatch Activity and Biometrics Dataset*. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5HK59. 2019.

[12] Svante Wold, Kim Esbensen, and Paul Geladi. "Principal component analysis". In: *Chemometrics and intelligent laboratory systems* 2.1-3 (1987), pp. 37–52.

[13] Han Xiao, Kashif Rasul, and Roland Vollgraf. "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms". In: *arXiv preprint arXiv:1708.07747* (2017).

[14] Zhirong Yang et al. "Stochastic cluster embedding". In: *Statistics and Computing* 33.1 (2023), p. 12.

Acknowledgments

Completing this thesis has been both a challenging and rewarding journey, and it would not have been possible without the support of several people who have been by my side throughout this process.

First and foremost, I would like to express my deepest gratitude to my supervisor, Matteo Ceccarello. Your guidance, insightful feedback, and constant support have been crucial to the successful completion of this work. I am truly grateful for the time and effort you have invested in helping me grow. Thank you for always being available to provide advice and encouragement.

I would also like to extend my heartfelt thanks to my husband Mahdi, whose support has been my greatest source of strength. Your encouragement, and patience have carried me through the most challenging moments of this journey. I am incredibly fortunate to have you by my side.

To my friends and my dearest passengers group, thank you for offering encouragement and always knowing when I needed a break. Your friendship, humor, and positive energy have been essential in helping me stay motivated and focused.

I am also deeply grateful to my family for their unconditional love and support. Thank you for always standing by me and for providing me with the strength to keep moving forward.

Finally, I would like to acknowledge everyone else who contributed to this thesis. Thank you all for being part of this journey.

Paria Tahan