Online Graph Clustering with Sparse Grids Density Estimation

Denys Sobchyshak
Computational Science and Engineering (Int. Master’s Program)
Technical University of Munich (TUM)
Department of Informatics

Master’s Thesis

Online Graph Clustering with Sparse Grids
Density Estimation

Author: Denys Sobchyshak
1st examiner: Prof. Dr. Hans-Joachim Bungartz
2nd examiner: Prof. Dr. Thomas Huckle
Advisor: M. Sc. Kilian Röhner
Thesis handed in on: October 26, 2015
I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

October 26, 2015

Denys Sobchyshak
Acknowledgements

This thesis would not have been possible without the direct and indirect contributions of several people, to which I owe my greatest gratitude.

First and foremost, I would like to thank my advisor, Kilian Röhner, for all of the support and open atmosphere that he showed throughout this entire time, for giving me the opportunity to work on this topic and for his excellent skill of laying down complex theories in simple words.

Furthermore, I would like to thank my family and friends for encouragement and motivation that they gave me. I would also like to thank my first teachers in linear algebra and computer hardware, Tatiana Oleshko and Natalia Fomina, for believing in me.
Abstract

During the past century fast-paced developments in technology have substantially increased information flow and the need of its understanding. Within such data-centric environment an ability to make quick strategic decisions based on generalized understanding of immense amounts of rapidly changing data became of great importance and a major deciding factor between success and missed opportunity. Today such problems are tackled by using efficient data mining techniques, which allow an extraction of useful information out of cluttered and not seldom obfuscated data sets and place it into clusters or densities of certain shape.

While the task itself is not novel, methods and approaches of solving it have gone a long way and with recent developments have opened new opportunities. We want to focus on performing graph stream clustering in mini-batches of constantly flowing data, with a goal of capturing topological information and adapting to its change, as more recent data takes over older one, known as concept drift.

We propose to consider dimensionality reduction techniques as means to transform graph topological information into its feature space, which can be done by applying certain weighting scheme based on dissimilarities between graph nodes and its corresponding Laplacian matrix. Furthermore, we utilize density estimation to identify regions of high and low node coupling, which subsequently serve to perform clustering.

While one of the most popular ways to tackle unsupervised learning problem of density estimation is by applying kernel methods, we argue that, because their evaluation depends on all of the data points at any given time, sparse grids approach stands as a better option, since not only it depends on grid points and thus can be used at any point of stream processing, it also poses an efficient way of decreasing the grid size, requiring fewer points stored to perform estimation.

We demonstrate how our algorithm works on real-world data and discuss some of the unexpected findings, when performing clustering on a feature space of high-dimensionality.
Contents

Acknowledgements iii
Abstract iv

I. Introduction and Theory 1

1. Introduction 3

2. Learning in the Context of Data Mining 5
   2.1. Unsupervised Learning 5

3. Clustering 7
   3.1. Graph Theory 7
   3.2. Graph Data 8
   3.3. Feature Space 9

4. Density-Based Clustering 11
   4.1. Kernel Methods 11
   4.2. Grid Methods 12
      4.2.1. Sparse Grids 12
      4.2.2. Clustering 15

5. Dimensionality Reduction 18
   5.1. Linear Reduction 18
      5.1.1. Principal Component Analysis (PCA) 18
      5.1.2. Multidimensional Scaling (MDS) 19
   5.2. Non-Linear Reduction 20
      5.2.1. Isomap 21

6. Distance Measures 23
   6.1. Edit Distances 23
      6.1.1. Levenshtein Distance 23
      6.1.2. Longest Common Subsequence 24
      6.1.3. Damerau-Levenshtein Distance 25
Contents

6.2. Graph-Weighted Edit Distance ........................................... 25
   6.2.1. Wagner-Fisher Algorithm ....................................... 27

7. Stream Processing ................................................................. 29
   7.1. Graph Streams ......................................................... 29

8. Cluster Validation ................................................................. 31
   8.1. Internal .................................................................. 31
   8.2. External .................................................................. 32

9. The Algorithm ....................................................................... 33

II. Numerical experiments .......................................................... 34

10. Numerical experiments .......................................................... 36
    10.1. Data sets .............................................................. 36
    10.2. Mini-batch density convergence .................................. 37
    10.3. Concept drift .......................................................... 39
    10.4. Clustering topology .................................................. 39
    10.5. Clustering other dimensions ...................................... 42

11. Conclusions and future work .................................................. 46

Bibliography ............................................................................. 49
Part I.

Introduction and Theory
1. Introduction

Throughout the last century advances in hardware and software have substantially increased information flow in virtually all spheres of human life. With the age of Internet of Things, which is bringing down final frontiers of connectivity, necessity for skimming through vast amounts of data, while grasping its general characteristic or understanding its overall structure and direction, is increasing on an hourly basis. This is especially true for areas of human networking and information exchanges, where possibilities may come and go without notice. Specifically, understanding current trends and direction of community interests define the difference between success and missed opportunity.

One of most optimal ways to tackle such problems today is by using data mining methods in conjunction with their efficient parallel implementations, which allow an extraction of useful information out of cluttered and not seldom obfuscated data sets in the smallest time possible. Naturally, these data crunching problems are often assigned to complex computer systems due to their processing potentials.

Cluster analysis is one of the many akin tasks and as such its applications vary greatly among different industries, starting from business and marketing, where it is used for market segmentation, product positioning and etc, transitioning to world wide web, where it is used for social network analysis, search result grouping and etc, and coming down to medicine, biology and bioinformatics, where it is used for organism communities comparison, tissue and blood differentiation and etc.

Because amounts of data show no intention of decreasing and only keep growing, data scientists are constantly pressured to come up with novel methods, which will exhibit efficient big data set handling characteristics, faster analysis times, decreased space and system resource usage and so on. In this thesis we will present an approach that conveys all those attributes, specifically our ultimate goal is to present an algorithm that performs real-time clustering of a graph-based data representation, found in multitude of places in today’s online communities.

To understand further material better, the reader should take into account, that our approach is based around an online (sometimes referred to as streaming or mini-batch) learning principle. The difference between online and offline (or batch) learning techniques, is that in online setting the data process-
1. Introduction

ing, which is updating density function in our case, is performed after arrival of every new data point (or object), whereas batch approach is used when we have access to the entire data set at once.

This thesis is split into two parts, that give proper walk through to an unfamiliar reader:

- In **Part I** we provide theoretical background, required to understand our approach as well as give a generalized overview of the algorithm. We start by defining the problem’s realm and its relation to a more general field of knowledge discovery from databases (KDD) in Chapter 2, switching to peculiarity of clustering method applied in Chapter 3 and Chapter 4, after which we will talk about ways to convey topological information into feature space in Chapter 5 and Chapter 6. We will wrap this part by depicting specifics of graph streaming in Chapter 7 and cluster validation in Chapter 8.

- In **Part II** we will present some numerical results and our findings, as well as argue about various aspects and parametrization of the algorithm.
2. Learning in the Context of Data Mining

In a generalized setting data mining can be defined as a process of discovering underlying structure or patterns in data sets with a goal of its future reuse for certain tangible gains, e.g. for prediction, classification, anomaly detection, etc. However, data mining itself is normally carried out as a part of more general process of knowledge discovery from databases (also known as KDD) with numerous steps [10]:

1. Selecting a data set
2. Pre-processing
3. Data reduction and transformation
4. Data mining: searching for patterns
5. Interpretation and evaluation of results

Data mining process is also a largely interdisciplinary field, that uses techniques from database systems, statistics, artificial intelligence and especially machine learning, which focuses on construction of algorithms, that learn and perform predictions on data. In this work we will expand on the machine learning subfield, as our focus lies mainly on the learning task and corresponding involved algorithms, with a purpose of discovering a structure in an evolving system and potentially generalize learned patterns to new unseen samples. In machine learning there exist three main approaches: supervised, unsupervised and reinforcement learning, however, we are primarily focused on the unsupervised setting, where no guiding reference is provided as we aim to discover structure.

2.1. Unsupervised Learning

As mentioned before, in unsupervised learning data points are not supplemented with target values, thus we aim to discover structure on our own. More
formally, to perform further processing, one is given only a data set

\[ S = \{x_1, \ldots, x_N\} \in \mathbb{R}^t, \]

where \( N \) denotes number of samples and \( t \) stand for dimensionality of the data. For example, in supervised learning we are instead presented with a labeled set

\[ S = \{(x_i, y_i)\}_{i=1}^M \in \mathbb{R}^t \times N, \]

where \( t \) represents data dimensionality (length of vector \( x_i \)) and \( y_i \) represents a label (i.e. a class) from an arbitrary set \( N \), assigned to sample \( x_i \), which comprises the major difference between supervised and unsupervised learning, as the former one always has target values. It should also be noted, that depending on the problem, which supervised learning algorithm is trying to solve, \( y_i \) can take on various values.

We can distinguish between several goals of unsupervised learning problems: discovering groups of similar data points within the data set, commonly known as clustering; determination of the data distribution, known as density estimation; and projection of data from a high-dimensional space to a lower-dimensional one, quite often for the purpose of visualization, also known as dimensionality reduction [5]. Not seldom one prefers to use density estimation as a basis for further processing, since due to its fundamental nature it can easily be defined without taking into account data specific context, while the other tasks should include such in their inner workings [12]. Thus in this work we heavily rely on density based clustering as supplemented in more detail further.
3. Clustering

Clustering is one of the most fundamental data mining problems due to its extensive application in various fields, e.g. customer segmentation, target marketing, data summarization, trend detection, social networks analysis, etc. In a very broad sense, it can be defined as when given a set of data points we aim to partition them into groups, where data points are as similar as possible. It can also be viewed as a mixture of feature selection/dimensionality reduction methods with clustering [3].

The problem can be approached differently depending on its specific, e.g. connectivity-based, centroid-based and density-based clustering to name a few. However, even though not seldom those approaches provide varying results, they are based implicitly or explicitly on an assumption, that data is generated from a probability distribution of a given type, e.g. from a mixture of \( k \) Gaussian distributions. In particular, this is the case for EM (Expectation Maximization) clustering and for \( k \)-means. Due to this assumption, the algorithms produce spherical clusters and cannot deal well with datasets, where the actual clusters have non-spherical shapes, which is also known as non-convex results when projected onto multiple dimensions. Due to this reason, we stick to density-based clustering, which, unlike the others, doesn’t assume anything about the data and defines clusters as regions of “high density”.

Before expanding on specifics of our clustering algorithm, we want to pay a little bit more attention towards the data itself. Clustering can be applied to both graph based or topological structures and raw feature spaces (vectorial data). We will further give a brief overview of most popular graph based clustering methods and expand on feature space representations of data, as it represents our main focus in this work.

3.1. Graph Theory

To begin with, let us define a graph to be given as \( G = (V, E) \) where \( V = \{v_i\}_{i=1}^N \) is a set of its \( N \) vertices and \( E = \{e_{i,j}\} \) is a set of its edges, where each edge is defined as a set of two vertices \( e_{i,j} = \{v_i, v_j\} \). This work deals exclusively with undirected graphs without loops (see Figure 3.1), defined as a graph \( G \) in which for each outgoing edge \( e_{i,j} \) there’s a ingoing edge \( e_{j,i} \) and
there are no edges outgoing and ingoing to the same vertex, thus $e_{i,j} \in E$ exists only if $i \neq j$.

![Figure 3.1](image)

Figure 3.1.: An undirected graph with one loop going from node 5 and back into itself. Notice, that there is a path from each node of the graph to any other node, which indicates that this graph is connected. Each node is labeled by its degree.

**Adjacency matrix** $A$ of graph $G$ is defined as a $N \times N$ matrix with value 1 at element $a_{i,j}$ if there is an edge $e_{i,j} \in E$ from vertex $v_i$ to $v_j$ and 0 otherwise.

**Degree matrix** $D$ of graph $G$ is defined as an $N \times N$ matrix with a number of ingoing edges on the diagonal element $d_{i,i}$, which corresponds to the vertex $v_i$:

$$d_{i,i} = |\{e \in E | v_i \in e\}|.$$

**Laplacian** of a graph $G$ is defined as an $N \times N$ matrix $L = D - A$.

**Size** of a graph $G$ is defined as a sum of its edges $|E|$.

**Subgraph** $\hat{G} = (\hat{V}, \hat{E})$ of a graph $G = (V, E)$ is a graph whose vertices $\hat{V} \subset V$ and edges $\hat{E} = \{\{v_i, v_j\} \in E | v_i, v_j \in \hat{V}\}$. See Figure 3.2.

One should note that both Laplacian and adjacency matrix of a undirected graph without loops are always symmetric.

### 3.2. Graph Data

One of the most straightforward ways to preserve structural or topological information of a graph while clustering the data is to act directly upon it, which, depending on the size of a graph, may be referred to as network clustering. There are various approaches to handling this task [27]:

- **Median graph approach** is based on a notion of graph with lowest average distance to other graphs in a set [26] and is commonly used in $k$-means based extensions. Depending on how median graph is defined,
3.3. Feature Space

there are several variants [15], but most of them feature rather poor computational characteristics.

• **Frequent pattern mining** is a probabilistic approach, focused on collecting a set of patterns based on latent cluster labels [29], however, corresponding algorithms usually produce too many clusters, which are hard to interpret.

Despite apparent advantages of methods acting directly on data topology and preserving its information, they normally feature rather high complexity, if applied to generalized case, or are restricted to certain graphs, thus seem to require more research. In addition, our task requires capturing a concept drift of data and due to its streaming nature, processing the entire graph is not possible, which is why we lean in the direction of more traditional methods that operate on feature spaces.

### 3.3. Feature Space

Feature or vectorial space has the most preference among clustering research and is widely used data representation, due to myriad of mathematical possibilities available in a vector space, e.g. distance/similarity computation between two vectorized representations of data instances is rather straightforward. As noted before, despite high efficiency of these methods, they are characterised by an inherent information loss relating to their representational limitations. However, we can tolerate such loss due to gain of higher flexibility with respect to concept drift preservation, which is a topic of itself and will be discussed slightly further. At this point we prefer to discuss some of the most common clustering approaches based on feature space [3]:

![Undirected graph and subgraph](image)
3. Clustering

- **Hierarchical clustering** is based on the idea of objects close together being more related than those farther away, thus connecting them to form clusters based on their distance. In such setting a cluster is described by the maximum distance needed to connect its parts, thus different distances form different clusters, which can be represented using a dendrogram. This approach produces a hierarchy requiring further user actions. In addition, outliers, handled by creating separate clusters, and extreme complexity make this approach a poor choice for big data [9].

- **Centroid-based clustering** is characterised by a central vector, not necessarily being a data point, serving as a central point of a cluster, to which closes points are assigned. Classic example of this is a \( k \)-means clustering algorithm and, as mentioned before, it fails to accurately identify clusters of non-spherical shape [5].

- **Distribution-based clustering** is characterised by assuming a specific form of generative model (e.g a mixture of Gaussians) and then estimating parameters of this model using Expectation Maximization (EM) [8]. This results in a model, using which we can estimate the fit probabilities of underlying data points, where samples with good distribution fit will have high fit probabilities, whereas anomalies will have very low fit probabilities. Quite often these methods suffer from overfitting and are burdened with a rather strong initial assumption on the data [4].

- **Density-based clustering** defines clusters as regions of “high density” as compared to the data set. One of the greatest advantages of this approach, is that a grid-based density estimation method can preserve evaluated function without the need of storing all the data points, as we will describe in the following chapter.
4. Density-Based Clustering

In this thesis we focus on density-based clustering, as not only it can handle clusters of various shapes and sizes, but also this approach doesn’t suffer from model overfitting, as is commonly the problem of distribution-based clustering. As we uncover theoretical details, the rationale of our choice will become more clear. First, let us define what density estimation is: when given a data set \( S = \{x_1, \ldots, x_N\} \in \mathbb{R}^d \) drawn from an unknown distribution \( p(X) \) of a random variable \( X \), we want to construct an estimate \( \hat{p} \) of the probability density function \( p \) on the data set \( S \). There are several ways of handling this task, but all of them can be split into two types:

- **parametric** density estimation approach assumes that the form of underlying distribution is known and aims to estimate only parameters of this distribution, as is done in Gaussian mixture models, where a linear combination of such distributions is taken and their statistical moments are estimated via the EM algorithm

- **non-parametric** density estimation requires only the data itself, as is the case with kernel density estimation [14].

We want to follow an Occam’s razor principle and avoid assumptions about data distribution, thus follow non-parametric approach.

4.1. Kernel Methods

Among numerous non-parametric approaches, kernel density estimation has long been one of the most popular ones [14]. It is based on a linear combination of kernel functions \( K \) centred at data points \( x_i \in S \), which looks as follows for 1D case:

\[
\hat{p}(x) = \frac{1}{N} \sum_{i=1}^{N} K(\frac{x - x_i}{\sigma^2}).
\]

Even though it depends only on the choice of bandwidth \( \sigma > 0 \) and the base kernel function \( K \), which is not seldom simply a Gaussian, the greatest weakness of this method lies in the fact, that evaluation of density function \( \hat{p} \) depends on a myriad of kernels centred at all data points, which becomes rather
4. Density-Based Clustering

expensive once the data set size \( N \) is large enough. Such direct dependence on the size of data set can be decreased by combining several data points into bins and placing kernel functions on top of those instead, however, with increasing dimensionality number of bins can increase exponentially, which makes it infeasible in a high-dimensional setting (with more than four dimensions).

4.2. Grid Methods

One of the promising ways to overcome the problem of storing all the data points and evaluating kernel functions over them was implemented via grid-based density estimation. The idea is to start with a highly overfitted guess \( p_\epsilon \) and then use a spline smoothing to obtain a more generalized approximation \( \hat{p} \) [13].

Suppose we have an initial guess \( p_\epsilon \) of the density function describing the data \( S = \{x_1, \ldots, x_N\} \), then line smoother search is proposed to be done in a functional space \( \nu \) such that:

\[
\hat{p} = \arg \min_{f \in \nu} \int_{\Omega} (f(x) - p_\epsilon(x))^2 dx + \lambda \| \Lambda f \|_{L^2}^2 ,
\]

(4.1)

where left part of \( \hat{p} \) ensures close fit to \( p_\epsilon \) and the right term introduces regularization. Now let \( \delta_{x_i} \) be the Dirac delta function centred on \( x_i \). If we set \( p_\epsilon = \frac{1}{M} \sum_{i=1}^{M} \delta_{x_i} \), we obtain a variational equation after some transformations [13]:

\[
\int_{\Omega} \hat{p}(x) \psi(x) dx + \lambda \int_{\Omega} \Lambda \hat{p}(x) \Lambda \psi(x) dx = \frac{1}{M} \sum_{i=1}^{M} \psi(x_i), \forall \psi \in \nu.
\]

(4.2)

Note that we could use a different initial guess \( p_\epsilon \), but the presented choice is both simple and sufficient, thus we will stick to it. One can find a solution to 4.2 using Galerkin projection in a finite dimensional space. In [13] the density function is discretized with basis functions centred on grid points rather than with kernels centred on data points, which solves the problem to an extent, since, just as with kernel methods, the number of points can grow exponentially with increase in dimensionality. One can overcome this problem by using sparse grids density estimation instead [22], which is properly suited for large data sets.

4.2.1. Sparse Grids

Before we continue with the sparse grids approach, lets take a brief look at its basics [23]. We consider a \( d \)-linear interpolation of a function \( f \) : \( \Omega \rightarrow R \), given
4.2. Grid Methods

on a discretization domain $\Omega = [0, 1]^d$. To construct an interpolant $u$ of $f$, we
discretize the domain $\Omega$ using a regular grid and obtain equidistant grid points
$x_i$ with mesh width $h_n = 2^{-n}$ in each dimension $d$ for some refinement level $n$. Now, by defining a set of piecewise $d$-linear functions $\phi_i(x) \in \nu_n$ we can construct an interpolant $u(x)$ using a weighted sum of basis functions:

$$f(x) \approx u(x) = \sum_i \alpha_i \phi_i(x).$$

By further decomposition of approximation spaces the interpolant can be rewritten as a finite sum:

$$u(x) = \sum_{l \leq n, i \in I_l} \alpha_{l,i} \phi_{l,i}(x),$$

where $I_l$ is a hierarchical index set as defined in [23], $\alpha_{l,i}$ is a hierarchical sur-

plus corresponding to hierarchical basis function $\phi_{l,i}(x)$. One dimensional ex-

ample of this idea is presented on Figure 4.1.

The next step of sparse grids approach would be to use only those approx-
imation subspaces, that contribute most to the overall solution, which is de-
picted in Figure 4.2. A more detailed explanation of the method can be found
in [23].

Continuing the discussion as we left off with equation 4.2, let $\Phi_\ell$ be a set of

hierarchical basis functions of the sparse grid space $\nu_\ell^{(1)}$ of level $\ell$, then we

are
4. Density-Based Clustering

Figure 4.2.: Two-dimensional approximation subspaces up to level 3 in each dimension, indicated by a subindex. The optimal subspace selection is shown in black, with corresponding resulting sparse grid to the right. If grey subspaces are included, we will get a full grid [23].

searching for \( \hat{p} \in \nu_l^{(1)} \) such that

\[
\int_{\Omega} \hat{p}(x)\phi(x)dx + \lambda \int_{\Omega} \Delta \hat{p}(x)\Delta \phi(x)dx = \frac{1}{N} \sum_{i=1}^{N} \phi(x_i), \forall \phi \in \Phi_{\ell}.
\]  

(4.3)

Since \( \hat{p} = \sum_{(l,i) \in I} \alpha_{l,i} \phi_{l,i} \) is a linear combination we can write 4.3 as a system of linear equations

\[(R + \lambda C)\alpha = b,\]  

(4.4)

where \( R_{i,j} = (\phi_i, \phi_j)_{L_2} \), \( C_{i,j} = (\Delta \phi_i, \Delta \phi_j)_{L_2} \) and \( b_i = \frac{1}{N} \sum_{i=1}^{N} \phi_i(x_i) \), for which an arbitrary ordering of \( M \) sparse grids basis functions \( \phi_{l,i} \) and corresponding coefficients \( \lambda_{l,i} \) was used. Note that the matrices \( R, C \) are of size \( M \times M \) and the right hand side \( b \) is of size \( M \), making the number of unknowns independent of data points \( N \), thus allowing us to ensure that the number of grid points grows only moderately with the number of dimensions. A more detailed explanation of the approach can be found in [21] and a convergence analysis of related approach can be found in [24].

Adaptivity

One can further decrease the number of grid points by applying adaptivity to sparse grids. Generally, it can be defined as a process of refining the grid at certain points to improve accuracy of the function \( f_M = \sum_i \alpha_i \phi_i \in \nu_l^{(1)} \), as a
sparse grid interpolant of target function $f$. Naturally, an adaptivity criterion is required to perform such an operation and those tend to be very problem specific. We decided to use a simple but rather popular criterion based on hierarchical coefficients $\alpha$, which essentially refines grid points at which the sharpest density change takes place. Note that refinement process itself poses some constraints, since most sparse grids algorithms [23] require existence of hierarchical parent points as depicted on Figure 4.3.

Figure 4.3.: The regular grid of second level in the left is refined to obtain the sparse grid in the middle. One more refinement step is performed on the middle grid, which requires to create hierarchical parent points (gray) on the right [23].

### 4.2.2. Clustering

As was indicated before, density-based clustering rotates around an idea of finding “high density” regions. We employ a clustering approach discussed in [21] with following steps given for a dataset $S = \{x_1, \ldots, x_M\}$:

1. Construct a similarity graph $G = (S, E)$ to represent the data points of $S$ based on $l \in \mathbb{N}$ nearest neighbours as determined via an Euclidean distance, defined for two arbitrary vectors $\vec{p}, \vec{q} \in \mathbb{R}^k$ as

   $$\text{eucl}(\vec{p}, \vec{q}) = \sqrt{\sum_{i=1}^{k} (p_i - q_i)^2}.$$  

   Here we compute the distance between all points and for every one of them find $l$ closest neighbours.

2. Compute probability density function $\hat{p}$ via sparse grids density estimation as discussed in 4.2.1.

3. Create subgraph $\hat{G}_i = (\hat{S}, \hat{E}) = (S \setminus \hat{S}, E \setminus \hat{E})$ by deleting vertices $\hat{S}$ and related edges $\hat{E}$, at which estimated density function is below a “high density” threshold parameter $\epsilon$. 


4. Density-Based Clustering

4. Depending on remaining connected subgraphs within $\hat{G}$, we assign labels $y_{i1}, \ldots, y_{iM} \in \{1, \ldots, k\}$ to remaining vertices (data points) in $\hat{S}$.

The slowest part among those steps would be construction of similarity graph, since it requires $O(n^2)$ operations, however, it can be easily parallelized. For the last step we have provided our own implementation of finding connected subgraphs, which is a DFS-based (depth-first search) method similar to the one discussed in [7]. Pseudocode sketch is given in Algorithm 4.1.

**Algorithm 4.1** Adjacency matrix DFS-based connected component search

```
function FIND_COMPONENTS(adj) ▷ adjacency matrix as input
define dynamic 2D vector components
define queue[adj.n_cols]
cnt ← 0
for i = 0 .. queue.size() - 1 do
    if adj[i][i] ≠ -1 then ▷ skip removed nodes
        queue.append(i)
while not queue.empty do
    VISIT_NODE(adj, components, queue.next(), queue, cnt)
cnt ← cnt + 1
return components

function VISIT_NODE(adj, components, idx, queue, cnt)
found ← -1
for i = 0 .. queue.size() - 1 do
    if queue[i] == idx then
        found ← i
        break loop
if found ≠ -1 then ▷ prevent checking visited or deleted nodes
    queue.erase(found)
    components[cnt].append(idx)
for i = 0 .. adj.n_cols - 1 do
    if adj[idx][i] ≠ 0 then ▷ indicates an edge
        visit_node(adj, components, i, queue, cnt)
return
```

This algorithm will return a 2D array, where rows represent different connected components and its columns represent corresponding nodes of similarity graph $G$, which are included in particular connected component.

Given that corresponding similarity graph $G$ was built using relatively small number of nearest neighbours $l \in \mathbb{N}$, this algorithm’s worst case complexity will never exceed $O(|V| + |E|)$, but in most cases will be considerably smaller,
since it greatly depends on the “high density” threshold parameter $\epsilon$, which defines how many low density nodes were already removed, thus decreasing the $|V|$ and $|E|$ with each removed node.
5. Dimensionality Reduction

As discussed previously in Section 3.1, conveying graph structural information into vectorized form (feature space) presents multiple opportunities and provides more flexibility in the choice of clustering methods. For the purposes of this work we have decided to use simplistic approach known as a family of dimensionality reduction methods.

Generally, these methods are used to transform data from high-dimensional to lower-dimensional space, while preserving critical information. More formally, we attempt to learn a manifold by deriving a set of degrees of freedom, which can be used to reproduce most of the variability of original data set. In mathematical terms, given a $t$-dimensional random variable $x = (x_1, \ldots, x_t)^T$ we aim to find a lower-dimensional representation $s = (s_1, \ldots, s_d)^T$ with $d < t$ according to some criteria, that preserves critical information of the original data.

There exist a myriad of various methods to achieve this goal as shown in [11], however, we focus on few most relevant for this work with a brief overview of closely related ones.

5.1. Linear Reduction

5.1.1. Principal Component Analysis (PCA)

The most popular technique for linear dimensionality reduction is the principal component analysis (PCA), which for a given data set of dimension $t$ aims to find a linear subspace of dimension $d$, lower than $t$, such that the data points lie mainly on the linear subspace. The main advantage of PCA is that it aims to preserve variability of the data, where each principal component (PC) would account for the highest variance within the data. Detailed derivation of the method via Lagrange multipliers is provided in [16], thus we would like to reflect upon several steps of its essentials, relevant for this work.

Given a $N \times t$ matrix of initial data points $X$, where each column corresponds to the $i$th dimension of the observation, with $i \in 1, \ldots, t$ and with $N$ observations, one obtains

1. Center the data around zero by subtracting dimension wide mean from each data point
5.1. Linear Reduction

2. Calculate the \( t \times t \) covariance matrix \( C = \frac{1}{N-1}XX^T \)

3. Calculate eigenvalues and eigenvectors of \( C \)

4. If the matrix of largest \( d \) eigenvalues is given as \( \Lambda_+ \) (PCs) and \( V_+ \) contains corresponding eigenvectors, the points of lower dimension are given as \( Y = XV_+ \)

As one can notice, the problem of PCA for the purposes of this work lie in its focus on highest variance components (see [16] for relation of PCs with variances) and not the pairwise distances.

5.1.2. Multidimensional Scaling (MDS)

By looking at the problem from a different perspective, one can discover a so known multidimensional scaling (MDS) method, which seeks to find a lower-dimensional approximation of pairwise distances. More formally, the objective of MDS is to construct a representation of given high-dimensional data in low-dimensional Euclidean space, where distances between pairs of points are approximated to the corresponding dissimilarity, which is represented as an \( N \times N \) dissimilarity matrix \( D = [\delta_{i,j}] \), with \( N \) number of points (objects or data instances) and \( \delta_{i,j} \) being the dissimilarity measure (see Chapter 6) between point \( i \) and point \( j \)

\[
D = \begin{pmatrix}
\delta_{1,1} & \delta_{1,2} & \cdots & \delta_{1,N} \\
\delta_{2,1} & \delta_{2,2} & \cdots & \delta_{2,N} \\
\vdots & \vdots & \ddots & \vdots \\
\delta_{N,1} & \delta_{N,2} & \cdots & \delta_{N,N}
\end{pmatrix}
\]

The dissimilarity matrix has to follow certain restrictions:

- symmetricity, so that \( \delta_{i,j} = \delta_{j,i} \)
- non-negativity, so that \( \delta_{i,j} \geq 0 \), \( \forall i, j \)
- zero diagonal entries, so that \( \delta_{i,i} = 0 \), \( \forall i \)

Thus given dissimilarity between original data points \( x_1, \ldots, x_N \in \mathbb{R}^t \) we aim to find \( N \) data points \( y_1, \ldots, y_N \in \mathbb{R}^d \) such that if \( \hat{\delta}_{i,j} \) denotes the Euclidean distance between points \( y_i \) and \( y_j \), then \( \hat{D} \) is similar to \( D \), thus \( \|y_i - y_j\| = \hat{\delta}_{i,j} \sim \delta_{i,j}, \forall i, j \in 1, \ldots, N \) and we seek to minimize

\[
\min_{y \in \mathbb{R}^d} \sum_{i<j} \left( \delta_{i,j} - \hat{\delta}_{i,j} \right)^2.
\] (5.1)
5. Dimensionality Reduction

The beauty of MDS is that it has an analytic solution and does not require iterative schemes. Its detailed derivation is described in [6], however, the overall procedure for computation is as follows

1. Compute the matrix of squared dissimilarities $D^{(2)}$ by squaring each element of the matrix.

2. Apply double centering to $D^{(2)}$ using matrix $J = I - \frac{1}{n}11^T$ as follows

$$B = -\frac{1}{2}JD^{(2)}J,$$

where $1$ is a unit column vector and $I$ an identity matrix.

3. Compute the eigendecomposition of $B = V\Lambda V^T$, where $V$ is a vector of eigenvectors and $\Lambda$ a matrix of eigenvalues at it’s diagonal.

4. If the matrix of first $d$ eigenvalues greater than zero is given as $\Lambda_+$ and $V_+$ contains first $d$ columns of $V$, the points of lower dimension are given as

$$Y = V_+\Lambda_+^{\frac{1}{2}}.$$

Computational complexity of this method is highly dependant on the choice of eigendecomposition algorithm, which is $O(n^3)$ in our case, but since we apply MDS only to a mini-batch, its influence on overall performance of the code is greatly diminished. However, for applications with high loads we recommend to consider faster parallelizable methods, e.g. $O(n \log n)$ as in [30].

Since we aim to convey structural graph data, preserving vertex distances information is the most critical aspect due to which multidimensional scaling (MDS) has higher preference in this thesis. However, we should note that despite having different mathematics, the solution to MDS is identical to the one of PCA and produces same results with respect to Euclidean distance. The advantage, however, is that in MDS the distances are not restricted to Euclidean measure and can stand for various dissimilarities between data instances.

5.2. Non-Linear Reduction

Previously introduced techniques for dimensionality reduction both share one significant drawback: they reconstruct a linear manifold, which might result in substantial information loss for non-linear cases. Thus an approach that utilizes geodesic distances instead of straight line Euclidean (or other alike measure) is applied. An example of such data can be a swirl, e.g. a so called “Swiss roll” as in Figure 5.1.
5.2. Non-Linear Reduction

As PCA was previously noted as a method of lower preference due to its focus on variance and not proximity, we will only look at more advanced versions of MDS.

5.2.1. Isomap

The complete isometric feature mapping (Isomap), as one of the approaches to constructing non-linear manifold in a lower dimensional space, is stimulated by an idea of utilizing original distance measures (dissimilarity matrix entries in MDS) for neighbouring points while measuring geodesic distances between far away ones. This is done by adding up neighbouring point distances sequentially while traversing through a shortest path towards the far away point [28]. The algorithm is comprised of three stages:

1. Determine which points are neighbours based on the dissimilarities $\delta_{i,j}$ in the input space $X$ and either connect each point to all other points within some fixed radius or to all $K$ of its nearest neighbours, so that these relations are represented as a weighted graph $G$ with edges of weight $\delta_{i,j}$, which is an easily parallelizable $O(n^2)$ procedure.

2. Estimate geodesic distances $\tilde{\delta}_{i,j}$ between all pairs of points by computing their shortest path distance in the graph $G$. Floyd-Warshall algorithm is one of the popular choices for this matter [7] with a worst case complexity of $O(|V|^3)$ (where $|V|$ is the number of vertices).
5. Dimensionality Reduction

3. Apply MDS technique to the obtained a geodesic distance matrix $\tilde{D}$, which will best capture the estimated intrinsic geometry of the underlying manifold. See Section 5.1.2 for MDS complexity analysis.

Derivation of the algorithm and proofs of convergence and efficiency can be found in [28].
6. Distance Measures

As discussed in Section 5, conversion of structural graph information into corresponding vectorized form requires a properly defined dissimilarity measure between two graph nodes. Complexity of this problem can be highly dependent on the data set itself, specifically the dimensionality of feature space. In other words, one can utilize up to all available information about the graph node to measure the dissimilarity.

In order to allow a more generic approach to diverse data sets, especially those missing additional feature space dimensions or even question contents, we chose to follow a simplistic approach of utilizing primary text information as given in title of the questions, which brings us to a common problem of measuring string dissimilarity.

6.1. Edit Distances

Throughout the past century there has been developed an entire family of string dissimilarity measures, known as edit distance measures, and despite their numerous variations the basic problem is defined as follows: given two strings \( a \) and \( b \) on an alphabet \( \Sigma \) the edit distance \( \text{dist}(a, b) \) is the minimum number of edit operations required to transform \( a \) into \( b \), where alphabet \( \Sigma \) is defined as a finite set of characters or numbers (e.g. a set of ASCII symbols or a set of bytes from 0 to 255). For the purposes of this work we assume that given strings can be of various lengths (including being equal) and thus consider edit distance variations which depend only on the allowed set of edit operations. In addition we consider only symmetric distances where \( \text{dist}(a, b) = \text{dist}(b, a) \).

6.1.1. Levenshtein Distance

The most used method is known as Levenshtein distance [18], which is composed of three edit operations each having a cost of 1: insertion, deletion and substitution. Consider an example provided in [17] of transforming a word
6. Distance Measures

‘intention’ into ‘execution’ that has a distance measure of 5 operations:

start with → intention
delete i → ntion
substitute n with e → ention
substitute t with x → exion
insert u → exution
substitute n with c → execution

Surely, one can argue that there can be other ways to transform one word into another, however, it is not hard to notice, that for both words first 5 characters are different, which indicates a minimum of 5 substitutions. Even if we try to hold character ‘e’ constant in both prefixes ‘inten’ and ‘execu’, we would still need to insert one character, since both are at different positions. Thus we can see, that 5 is indeed the smallest number of operations, required to transform the word.

More formally, Levenstein distance between two strings $a$ and $b$ of length $|a|$ and $|b|$ respectfully is given by $lev_{a,b}(|a|, |b|)$, where

$$lev_{a,b}(i,j) = \begin{cases} 
\max(i,j) & \text{if } \min(i,j) = 0 \\
\min(lev_{a,b}(i-1,j) + 1, lev_{a,b}(i,j-1) + 1, lev_{a,b}(i-1,j-1) + 1(a_i \neq b_j)) & \text{otherwise}
\end{cases}$$

with $1(a_i \neq b_j)$ equal to 0 when $a_i = b_j$ and 1 otherwise. As minimization is done over costs of operations

- $lev_{a,b}(i-1,j) + 1$ corresponds to a deletion
- $lev_{a,b}(i,j-1) + 1$ corresponds to an insertion
- $lev_{a,b}(i-1,j-1) + 1(a_i \neq b_j)$ corresponds to substitution

6.1.2. Longest Common Subsequence

Another variation of simple edit distance measure is limited to only insertion and deletion as the unit cost edit operations [20]. Qualitatively it measures length of the longest sequence of characters that respects ordering of letters and is present in both strings, while distance would be given as the number of unpaired or unmatched letters. Considering the fact that any substitution operation can be replaced with deletion and insertion, this method places penalty on substitution operation, effectively making it cost twice the other operations.
6.1.3. Damerau-Levenshtein Distance

Extension of edit operations with transposition (i.e., substitution of the form \(ab \rightarrow ba\)) uncovers another variation in the edit distance family. Mathematically, it is defined as a distance between two strings \(a\) and \(b\) of length \(|a|\) and \(|b|\) given by \(dlev_{a,b}(|a|, |b|)\), where

\[
dlev_{a,b}(i, j) = \begin{cases} 
\max(i, j) & \text{if } \min(i, j) = 0 \\
\min \begin{cases} 
\ dlev_{a,b}(i-1, j) + 1 \\
\ dlev_{a,b}(i, j-1) + 1 \\
\ dlev_{a,b}(i-1, j-1) + 1_{(a_i \neq b_j)} \\
\ dlev_{a,b}(i-2, j-2) + 1 \\
\ dlev_{a,b}(i-1, j) + 1 \\
\ dlev_{a,b}(i, j-1) + 1 \\
\ dlev_{a,b}(i-1, j-1) + 1_{(a_i \neq b_j)} 
\end{cases} & \text{if } i, j > 1 \text{ and } a_i = b_{j-1} \text{ and } a_{i-1} = b_j \\
\text{otherwise} 
\end{cases}
\]

with \(1_{(a_i \neq b_j)}\) equal to 0 when \(a_i = b_j\) and 1 otherwise and operations defined as

- \(dlev_{a,b}(i-1, j) + 1\) corresponds to a deletion
- \(dlev_{a,b}(i, j-1) + 1\) corresponds to an insertion
- \(dlev_{a,b}(i-1, j-1) + 1_{(a_i \neq b_j)}\) corresponds to substitution
- \(dlev_{a,b}(i-2, j-2) + 1\) corresponds to transposition between two neighbouring symbols

6.2. Graph-Weighted Edit Distance

As has been shown in Section 5.1 for the case of classic MDS it is absolutely vital for the algorithm to have dissimilarity measured between each and every node of the graph, more formally dissimilarity matrix \(D\) may have 0 elements only on the diagonal. However, this poses a problem for our work, since to convey graph structural information into feature space we heavily rely on the Laplacian matrix.

In order to overcome this problem we decided to introduce an experimental approach of weighted distance measures, that tightens and scales measured dissimilarity in accordance to the node’s overall connectivity. Specifically, using definitions of Chapter 3 for a undirected graph \(G = (V, E)\) of size \(|E|\), we create a dissimilarity matrix \(D = (d)_{i,j} \forall i, j \in 1, \ldots, |V|\) based on the Laplacian...
of a graph \( L \), where each element \( d_{i,j} \in D \), that corresponds to \( v_i \in V \) and is connected to any other vertex \( v_j \in V \) such that \( i \neq j \), is measured by the Levenshtein distance \( \text{dist}(v_i, v_j) \) scaled down using sum of degrees of two connected nodes \( v_i \) and \( v_j \), as compared to the graph size \( |E| \):

\[
d_{i,j} = \frac{\text{dist}(v_i, v_j)}{(\text{deg}(v_i) + \text{deg}(v_j))} |E|, \quad \text{if} \quad l_{i,j} = -1,
\]

where degree of arbitrary vertex \( v_l \) is defined as \( \text{deg}(v_l) = \sum_{e \in E} e_k,l, \forall k \). Applying the Handshaking lemma to undirected graphs without loops and a function \( \text{tr}(A) \), defined as a trace of arbitrary matrix \( A \), one can show that

\[
|E| = \frac{\text{tr}(L)}{2}.
\]

We can now use these definitions to expand our definition of dissimilarity matrix elements \( d_{i,j} \) to cases, where nodes \( v_i \) and \( v_j \) are not connected and when they represent a loop (see 3.1). The first entry for the case of \( l_{i,j} = -1 \) has been show before, thus we provide general definition of \( d_{i,j} \):

\[
d_{i,j} = \begin{cases} 
2 \times \frac{\text{dist}(v_i, v_j)}{(\text{deg}(v_i) + \text{deg}(v_j))} \frac{\text{tr}(L)}{\text{tr}(L)} & \text{if} \ l_{i,j} = -1 \\
2 \times \frac{\text{dist}(v_i, v_j)}{\text{tr}(L)} & \text{if} \ l_{i,j} = 0 \\
0 & \text{if} \ i = j
\end{cases}
\]

Consider an example, where we are given a data set of five questions, that has

![Figure 6.1: An undirected connected graph without loops, that represents topological structure of a data set with five questions.](image)

a topological structure as in Figure 6.1, which can be represented by a Laplacian matrix

\[
L = \begin{pmatrix}
2 & -1 & 0 & -1 & 0 \\
-1 & 2 & -1 & 0 & 0 \\
0 & -1 & 3 & -1 & -1 \\
-1 & 0 & -1 & 2 & 0 \\
0 & 0 & -1 & 0 & 1
\end{pmatrix}
\]
6.2. Graph-Weighted Edit Distance

with \(|E| = \frac{tr(L)}{2} = 5\). In addition assume we have already measured string distances and obtained the following dissimilarity matrix

\[
S = \begin{pmatrix}
0 & 30 & 52 & 44 & 13 \\
30 & 0 & 10 & 33 & 40 \\
52 & 10 & 0 & 18 & 6 \\
44 & 33 & 18 & 0 & 70 \\
13 & 40 & 6 & 70 & 0 \\
\end{pmatrix}.
\]

Then the resulting graph-weighted dissimilarity matrix looks as follows

\[
D = \begin{pmatrix}
0 & 1.5 & 10.4 & 2.2 & 2.6 \\
1.5 & 0 & 0.4 & 6.6 & 8 \\
10.4 & 0.4 & 0 & 0.72 & 0.3 \\
2.2 & 6.6 & 0.72 & 0 & 14 \\
2.6 & 8 & 0.3 & 14 & 0 \\
\end{pmatrix},
\]

where connected nodes have their corresponding edge weights substantially decreased, thus are more tightly coupled from the MDS viewpoint.

6.2.1. Wagner-Fisher Algorithm

Simplicity of Levenstein distance solves the problem well enough for the purposes of this work and thus was given the priority. Among numerous implementations of the distance we have selected the Wagner-Fisher dynamic programming version due to feasibility of its application to general cases of distance computation as discussed in [20].

The general idea is based on building a matrix of distances between all prefixes of both strings step-by-step, which results in final \(\text{dist}(a, b)\) measure, given in the last element of the matrix. A more detailed explanation is given in [20].

**Algorithm 6.1 Wagner-Fisher algorithm**

1: function DIST(a, b)  \(\triangleright\) string \(a\) of size \(m\) and string \(b\) of size \(n\)
2: define \(d[m+1][n+1]\)
3: for \(i = 0 \ldots m\) do
4: \(d[i,0] \leftarrow i\)
5: for \(j = 0 \ldots n\) do
6: \(d[0,j] \leftarrow j\)
7: for \(j = 1 \ldots n\) do
8: for \(i = 1 \ldots m\) do
9: if \(a[i] = b[j]\) then
10: \(d[i,j] \leftarrow d[i-1,j-1] \)

27
6. Distance Measures

11: \hspace{1em} \textbf{else}
12: \hspace{1em} d[i, j] \leftarrow 1 + \min(d[i - 1, j], d[i, j - 1], d[i - 1, j - 1])
13: \hspace{1em} \textbf{return} \ d[m, n]
7. Stream Processing

In recent years advances in hardware technology have made it possible to automatically record various aspects of human life at high rates and frequencies, which has caused massive data storage requirements and, due to its continuous and rapid growth this online data phenomena, is commonly referred to as data streams. There are several problems with those as identified in [3]:

- Because of enormous storage requirements, it is not always possible to persist the data for long term, thus it has to be processed in a single or certain constant number of passes. The processing algorithms must also be rather fast, since otherwise a model construction process won’t be able to catch up with the rate of input.

- Patterns in streaming data are subjected to continuous evolution [1] over time, putting a burden on the algorithm to not only adapt, as in updating cluster models, but also should be available at any time, since the stream of data may never reach its end.

- Different domains may have varying challenges posed to stream clustering algorithms, however, the optimum performance of the algorithm is often achieved by minimizing the number of passes over data, rather than the CPU operations count.

Our data stream is highly dependent on graph topological information, which has its own peculiar aspects.

7.1. Graph Streams

Not seldom graph streams are created as a result of social and information networks and, depending on the application specific scenario, can be differentiated into following streaming and clustering models:

- The stream is a sequence of objects, each of which is a small graph with a subset of nodes and edges. We aim to determine similar objects in stream based on similarities between nodes and edges. For example, the DBLP bibliographic network creates a graph stream out of co-authored papers.
7. Stream Processing

To determine clusters of objects with similar structure a Gmicro algorithm is employed as described in [4].

- The stream is a sequence of objects, each of which is a small graph with a subset of nodes and edges. We aim to determine a dense node set, which frequently occurs in these objects. A method proposed in [2] is a perfect example of such case. It heavily relies on minihash graph stream summarization.

- The stream is a sequence of objects or edges. We aim to determine dense node regions in the graph, which would compose different clusters.

Despite its general definition, the last type of streaming model is the exact scenario we aim to pursue in this thesis. Specifically, we aim to split our data set into mini-batches, which would afterwards be assembled into a connected undirected graph, comprising one object. We then process this object finding dense node sets within the graph. As indicated before, our density-based clustering algorithm works on a feature space representation of data, thus we employ distance based dimensionality reduction method to construct the space out of the topological information.
8. Cluster Validation

Having obtained cluster assignment it is important to check its quality, which is assessed via comparison to a true clustering known a priori, also referred to as external validation, or via assessing compactness and separation of clusters, known as internal validation [19].

8.1. Internal

Despite the variety of available assessment indexes, most of the internal validation methods try to check cluster compactness, i.e. how closely related the objects are in clusters, and separation, i.e. how distinct or well separated clusters are from each other. Thus these methods assign higher score to clusters with high similarity within a cluster and low one between different clusters.

One of the most common examples would be the Dunn index, which aims to identify dense and well-separated clusters by measuring ratio of minimal inter-cluster distance to maximal intra-cluster distance:

\[
D = \frac{\min_{1 \leq i \geq j \geq n} d(i, j)}{\max_{1 \leq k \geq n} d'(k)},
\]

where \(d(i, j)\) represents the distance between clusters \(i\) and \(j\), and \(d'(k)\) measures the intra-cluster distance of cluster \(k\). The family of Dunn’s indices is of course not alone, but just like other measures it suffers from high dependence on shape and distances between clusters.

To capture non-convex cluster shapes, we utilize an expected density measure \(\bar{p}\). Let \(C = \{C_1, \ldots, C_k\}\) be a clustering of a weighted graph \(G = (V, E, w)\), where \(w_{i,j}\) represents a weight of edge \(e_{i,j} \in E\) and let \(G_i = (V_i, E_i, w_i)\) be the induced subgraph of \(G\) with respect to cluster \(C_i\), then the expected density of cluster \(C\) is defined as

\[
\bar{p}(C) = \frac{\sum_{i=1}^{k} |V_i| w(G_i)}{|V| |V|^\theta},
\]

where \(|V|^\theta = w(G) = |V| + \sum_{e \in E} w(e)\), thus \(\theta = \frac{\ln(w(G))}{m(|V|)}\). A more detailed explanation can be found in [31].
8. Cluster Validation

8.2. External

In external cluster validation, the obtained results are to be measured against a priori given cluster assignments, however, for most of our data sets such labeling is not available, thus this method is not used.
9. The Algorithm

In order to understand our approach better, it can be useful to have a step by step description of how all of the above mentioned components glue together. Thus we want to provide a generalized walk through for one mini-batch:

1. We start off by reading the entire mini-batch of predetermined size and normalizing the data, by centering it around 0 and then scaling values into a range of [0, 1] for each of the $d$ feature dimensions.

2. Then we create a Laplacian matrix, based on the tag assigned to different questions, and a dissimilarity matrix, based on measures discussed in Chapter 6.

3. Next step is to convey graph topological information into its feature space representation, for which we use graph-weighted distance measure as given in Section 6.2 and transform it via MDS, as given in Section 5.1.2.

4. Obtained feature space is processed via sparse grids density estimation (see Chapter 4) and target function $\tilde{p}_i$ is obtained, where $i$ identifies the mini-batch number.

5. Using a learning rate parameter $lrate$, we merge obtained estimate $\tilde{p}_i$ with our global density estimation $\tilde{p}$ by adding missing grid points, which might have appeared due to adaptivity (see Section 4.2.1), and merging surpluses as

$$\alpha_j = (1 - lrate) \ast \alpha_j + lrate \ast \alpha_{j,i}$$

for all $j$ in a set of all grid points of a function $\tilde{p}_i$, where $\alpha_{j,i}$ represents a surplus of the density function $\tilde{p}_i$ at grid point $j$.

This routine is performed continuously, unless manually stopped, or used for a finite data set, as is the case for our numerical experiments, nevertheless, at any point in time user can obtain clustering from a globally accessible density estimation.
Part II.

Numerical experiments
10. Numerical experiments

In this chapter we will present numerical results obtained when putting together the entire pipeline of previously mentioned methods and provide some foundation for further research prospects. We focused primarily on capturing concept drift via mini-batch data processing, which can be further extended to streaming data in a 24/7 fashion, and on clustering topological information. Following results will show, that both goals were achieved and some unexpected difficulties have been discovered.

From the technical perspective it should be noted, that the software used for experiments was implemented entirely using C++ programming language with exception of pre- and post-processing scripts, which were done in Python and Octave. The density estimation part of it was based on SG++ framework [23], while other parts with extensive involvement of mathematical operations, which can be found in linear algebra of MDS and so on, were implemented from scratch using Armadillo mathematical library [25].

10.1. Data sets

First and foremost we would like to draw attention to data sets, used for these experiments, and the feature selections approach applied. As data analysis itself was not the focus of this thesis, we have succumbed to using a very simplisitic feature extraction approach from one of the most known graph based data stores: stackexchange.com. It provides data scientists from all over the world with a Data Explorer tool, which can be used to extract any piece of information about any particular subdomain of the Q&A (questions and answers) website. We have selected the following two data sets:

- **bitcoin** - a set of questions and tags related to a cryptographic currency named bitcoin. At the moment of extraction it contained 8445 questions.

- **quant** - a set of questions with tags relating to a topic of mathematical finance, especially the algorithmic side of it. At the moment of extraction it had 4098 different questions.

We have further processed both data sets in a similar manner, and have created a graph with questions as its vertexes and tags representing edges. For
other dimensions we have used questions view count, number of up-votes, user favourites and a date of posting, making it a 4D feature set. During processing we have converted the topological information into a 2D space, making it a total of 6D feature set.

10.2. Mini-batch density convergence

To start off we would like to show that our algorithm converges in the long run to a certain state as in
\[
\lim_{n \to \infty} |\tilde{p}_n(x) - \tilde{p}(x)| \to 0,
\]
where \( n \) is the number of current mini-batch, \( \tilde{p}_n \) is the density function as estimated after processing batch \( n \) and \( \tilde{p} \) is the final density function. This validation is performed by estimating function values at each data point for each batch and comparing them to the reference estimate, which is done once all mini-batches are processed. The results can be viewed on Figure 10.1, 10.2, 10.3 and show, that the form of density function \( \tilde{p} \) is influenced more by the recent mini-batches than by the ones processed longer time ago, i.e it converges to the most recent state of density shape and slowly discards previous information. Notice, that the learning rate substantially influences variability of data,

![Figure 10.1.](image)

(a) Relative error

(b) Variance of relative error

Figure 10.1.: Convergence of relative error and its variance towards zero as obtained at points of quant dataset processed with 750 questions per mini-batch, learning rate 0.1 and more than 2 thousand grid points specifically its higher values discards too much previous information (see Figure 10.2(a)), thus making it obvious that by setting too high of a learning rate
10. Numerical experiments

Figure 10.2.: Convergence of relative error and its variance towards zero as obtained at points of bitcoin dataset processed with 750 questions per mini-batch, learning rate 0.2 and 4552 grid points

Figure 10.3.: Convergence of relative error and its variance towards zero as obtained at points of bitcoin dataset processed with 1000 questions per mini-batch, learning rate 0.1 and more than 4 thousand grid points

one will always have rather big jumps from one shape of $\tilde{p}$ to another, providing quite inaccurate density estimation results and consequently bad clustering.

Moreover, as our selected data sets were rather small compared to ones of potential big data application fields, it should be noted that the learning rate
parameter must decrease substantially. Particularly, even though this matter is rather application specific, we would expect a learning rate of 0.0001 for a data set of around 1000 mini-batches and more. Just as described in previous paragraph, such a high learning rate for such a big data set would discard too much previous data and would adapt too fast to the distributions of newly processed mini-batches, making it capture and describe only a small frame of data.

10.3. Concept drift

Next criteria of the algorithm is its capacity to capture data evolution and adapt the approximated density function $\tilde{p}$ with each new mini-batch, which requires calculating a delta between two numeric function approximations. There are several ways to achieve that, but we prefer to show a visual difference between those estimates, which can be found on Figure 10.9 and 10.10.

One could notice a drastic change in the approximated density function after processing 6th mini-batch as a massive chunk of highly coupled data came in and centred around $[0.6, 0.2]$, which indicates certain topic becoming overly "hot", but then it slowly loses momentum by mini-batch 9. From the raw data in this particular case we can see a higher ratio of "blockchain" tags as compared to previous batches and while already being one of the major tags its ratio rose approximately 18.03%, but over the next 3 batches it’s averaged ratio drops by nearly 45.7%.

10.4. Clustering topology

One of the most interesting aspects of the algorithm is its capacity to identify clusters within topological structure. As indicated in [23] an arbitrary number of nearest neighbours and a good guess of "high density" threshold parameter is enough to perform decent clustering, which is exactly the case for our examples. In addition, presented clustering is based on uniform sampling of estimated density function with varying only topological features (dimensions 1 and 2) while holding others fixed. This approach is used due to an unexpected finding, which will be discussed further in a context of multi-dimensional clustering.

Already when using a simplistic linear reduction technique estimated density function would capture closely coupled regions, which identified "hot" topics as shown on Figure 10.4. Topology clustering certainly changes based on the mini-batch size, as it directly relates to topological information loss and the difference can be seen already after a slight change as depicted in Figure
10. Numerical experiments

10.5. Surely, based on the value to which other dimensions are fixed, topology clustering changes correspondingly as can be seen from Figure 10.6. However,

![Figure 10.4.](image1)

(a) Sampled density function approximation

(b) Clustered sampling

Figure 10.4.: Clustering of sampled data with 8 nearest neighbours and a density threshold of 1.01 on a grid of more than 5 thousand points obtained from processing mini-batches of 1000 questions with a learning rate of 0.1. Only topological features (first two dimensions) were varied, while others were fixed to value 0.5

![Figure 10.5.](image2)

(a) Sampled density function approximation

(b) Clustered sampling

Figure 10.5.: Clustering of sampled data based on bitcoin dataset with 11 nearest neighbours and a density threshold of 1.47 on a grid of 5027 points obtained from processing mini-batches of 1300 questions with a learning rate of 0.1. Only topological features (first two dimensions) were varied, while others were fixed to value 0.5
10.4. Clustering topology

Figure 10.6.: Clustering of sampled data based on bitcoin dataset with 5 nearest neighbours and a density threshold of 2.3 on a grid of 5346 points obtained from processing mini-batches of 1000 questions with a learning rate of 0.1. Only topological features (first two dimensions) were varied, while others were fixed to value 0.3.

it should be noted that it is not always easy to find a good guess of "high density" threshold and the number of nearest neighbours to use for similarity graph, especially in cases of numerous less dense clusters or when those are loosely interconnected via other dimensions, which may result in a clustering as in Figure 10.7. Clearly the black cluster on the top right quarter of Figure 10.7(b) consists of several sub-clusters, and is not identified correctly. We would suggest this topic as a separate research and can provide three ideas for further investigation:

• **Subdomain "high density" threshold** approach would consist of splitting the entire domain into smaller parts while setting different thresholds per subdomain. This might get quite complicated as dimensionality grows.

• **The "high density" threshold estimator** approach could potential supplement the previous idea by finding a certain way to automatically estimate the parameter instead of simply wiping down certain percentage of low density points.

• **Hierarchical threshold** approach would require multiple clusterings over the same data, with an idea of finding loosely coupled clusters and performing further clustering solely on that data with a new threshold.
10. Numerical experiments

(a) Sampled density function approximation  
(b) Clustered sampling

Figure 10.7.: Clustering of sampled data based on bitcoin dataset with 11 nearest neighbours and a density threshold of 0.7 on a grid with more than 5 thousand points obtained from processing mini-batches of 1000 questions with a learning rate of 0.1. Only topological features (first two dimensions) were varied, while others were fixed to value 0.5

10.5. Clustering other dimensions

In this section we would like to shed light upon an unexpected problem that we found, while clustering multiple dimensions. As one might have noticed before, our clusterings and concept drift was dedicated only to the topology information dimensions while we fixed others to certain constant value. However, the actual data points are rather scattered all over 6D and our algorithm fails to determine several clusters given only manual ways of selecting the number of nearest neighbours and the density threshold. As was previously argued [12], clustering algorithms should not discard context of data and as our feature selection process has overly simplified, it is exactly the opposite case. We have noticed that certain dimensions of our data had a rather skewed shape, as can be seen on Figure 10.8, which without doubt has an effect on the overall performance of the algorithm and clustering in particular. Additionally, it seems that high dimensionality of our feature space as compared to its sparsity (having topped only 8445 points) does not allow for proper cluster formation. This problem requires further study, which would benefit greatly from an automatic estimation of required number of nearest neighbours and density threshold.
10.5. Clustering other dimensions

(a) Density over dim2 and dim5

(b) Density over dim3 and dim4

(c) Density over dim1 and dim2

(d) Data clustering

Figure 10.8.: Density estimation over bitcoin data performed on a grid with more than 5 thousand points obtained from processing mini-batches of 1000 questions with a learning rate of 0.1 and clustering done with 11 nearest neighbours and “high density” threshold of 20
10. Numerical experiments

Figure 10.9.: Evolution of density function $\tilde{p}$ with new incoming mini-batch. Visualization is based on bitcoin dataset with learning rate 0.2 and batch size of 1000 questions with varying first two dimensions (topology features) and having others fixed. Please see continuation in Figure 10.10.
10.5. Clustering other dimensions

(a) after batch 7

(b) after batch 8

(c) after batch 9

Figure 10.10.: Continuation of Figure 10.9 for remaining mini-batches
11. Conclusions and future work

In this work we have presented an experimental way of conveying graph topological information into feature space and then expanded it with clustering the space via sparse grids density estimation, providing various arguments towards potential of such approach. As we carried out the experiments, based on stackexchange.com data, we have shown, that topological information can be captured in a streaming manner, with data evolution and automatic adaptation of our algorithm to the concept drift. Where learning rate defined how much of a data frame we want to preserve, influencing the extent to which more recent mini-batches influence general density shape of the data. Specifically, for our 8445 thousand samples data, we have determined, that learning rates of up to 0.3 were viable, however for bigger data sets we expected smaller rates. Numerical results also prove, that chosen combination of Levenstein distance measure, MDS dimensionality reduction and sparse grids density-based clustering, expanded with our improvements of graph-weighted distance measure and grid merge, are a viable graph clustering approach.

As we dealt with high-dimensional feature spaces, we have discovered a phenomena of high clustering dependency on data quantity and a feature selection process. Specifically, we have seen that a straightforward approach to 6D space feature selection, without its proper investigation and analysis, can lead to a highly coupled or skewed sample spaces via multiple dimensions, with most of the domain being covered with near zero density. Inevitably, this made our density function very volatile (with sharp changes) and rather sparse, exhibiting large areas of low density throughout most of the domain, which is also a result of dealing with numerous dimensions by using a rather small sized data set, since 8445 points for 6 dimensions is rather small.

These findings have led us to new ideas, which would make the clustering approach more complex, yet will allow us to find hierarchies or in some cases simply smaller clusters via splitting the sparse grids domain and setting different “high density” thresholds per subdomain, which poses itself as a great future research topic.

Moreover, when making a decision one is not seldom interested in both present and past state of things, which, adapted to our context, means that to judge upon current clustering of data one might be interested in its evolution process, thus how the data changed to give rise to current clustering. Through-
out our numerical results we have used a simplistic approach to show, how our density function evolved, but we did not focus on extending original algorithm with means to persist or visualize clustering evolution. However, the topic itself is rather interesting and is encouraged to be considered for further research.
Bibliography


