Faculty of Sciences
University of Novi Sad

Department of Mathematics and Informatics

MASTER'S THESIS

Submitted to

Faculty of Sciences
Department of Mathematics and Informatics

By

Jelena Petković

SELECTED GRAPH EMBEDDING METHODS

Supervisor: Prof. Dušan Jakovetić

September 2024

Acknowledgments

I am thoroughly thankful to Dr Dušan Jakovetić for his mentorship and dedication to the students throughout the master's studies. I am also greatly thankful to Dr Maja Jolić and Dr Nataša Krklec Jerinkić for kindly reviewing the thesis, providing valuable advice, and being part of the member board.
I am also grateful for all the teachers who instilled knowledge in me in every part of my formal education. I acknowledge the value of community therefore I am extremely thankful to all of my fellow peers in both bachelor's and master's programs - thank you for making the whole process of finishing these degrees more enjoyable!
Last but not least, I want to thank my sister Milica for her uplifting before and while writing this thesis!

I dedicate this thesis to my family and friends whose support, encouragement, and belief in me I have felt continuously throughout my studies.

Table of Contents

List of Figures

List of Tables

Introduction

In recent years networks have attracted a lot of attention since many real-world scenarios can be modeled using them. From social and transportation networks to citation networks, representing and analyzing data in this format is a common practice. While many algorithms deal with networks successfully there is still a need to try and simplify graph structure to one of fewer dimensions. The field of graph embedding is trying precisely to bridge a gap between graph structures and structures of fewer dimensions. Once the mapping is achieved, algorithms and techniques reserved for later structures can be successfully used on the graph data and yield new results. Dimensionality reduction is of great importance in many areas since it enables more efficient data mining, downsizes the complexity, and speeds up computing.The practical reasons for using it are the following:

1. Curse of dimensionality
   When data in hand is high-dimensional we come across a problem known as the "curse of dimensionality". As the number of dimensions increases, the data becomes more sparse which makes it harder for the model to learn similarities among the data and important features. The dimensionality reduction makes data more compact and as such more suitable for further analysis.
2. Efficiency and Scalability
   The large graphs, with thousands or even millions of nodes, are very expensive to store and manipulate. Once we have the lower-dimensional representation of a graph, the needed resources are smaller and the speed of the algorithms manipulating the graphs is increased.
3. Extracting important features
   It happens often that some of the dimensions are redundant and do not provide useful information but rather take away from the model's precision. The dimensionality reduction allows us to filter out unimportant information and focus on the valuable features of the graph structure.
4. Data Visualization
   The high-dimensional data can not be visualized. Once the dimensionality reduction is applied we can showcase the data in 2D or 3D space and recognize the relations and patterns in data.

In this thesis, we provide an overview of graph embedding methods and present in detail three approaches - *Locally Linear Embedding*, *Laplacian Eigenmaps*, and *DeepWalk*.

1. Graph Structure & Embeddings

In this section, we give theoretical definitions of both graphs [1] and graph embeddings [2].

1.1 Mathematical Definition of the Graph

A graph is defined as an ordered pair $G(V, E)$ where V is a set of nodes or vertices and E represents a set of edges i.e. links connecting two nodes and denoting that there is a present relationship among them. Being a non-linear data structure, graphs allow us to model complex systems and scenarios of network data.
Let $G(V, E)$ be a graph containing a set of vertices $V = \{v_1, v_2, ...\}$ and a set of edges among these vertices E. If there is a present link among two vertices $v_i, v_j \in V$ it will be denoted as an element of an edge set $e_{ij} \in E$.
There are many different classifications of graphs depending on which aspect of the graph we focus on. Some of them are presented below:

1. Directed vs. Undirected graphs
   In the directed graph every edge has a determined direction, which is often denoted with an arrow pointed in the wanted direction. On the contrary, an undirected graph has edges that are unordered pairs meaning that edge e_{ij} can be replaced with e_{ji}.
   A real-world example that is modeled with the directed graph would be Google Maps, while undirected graphs could be used to model online social networks - friendship between two individuals is mutual.

2. Weighted vs. Binary graphs
   In weighted graphs, each edge has a specific number called weight. The weight associated with the edge e_{ij} will be denoted by w_{ij}. In general, the edge weight represents a similarity between the two nodes being connected by it. The edge weight being higher indicates that the respective nodes are expected to be more similar. Conversely, in binary graphs, also called unweighted graphs, edges are not assigned numbers.

3. Homogeneous vs. Heterogeneous graphs
   In homogeneous graphs all nodes and edges are consistent i.e. of the same kind. We can think of it as if the nodes and edges are all equal in a hierarchy, for example in a

friendship group.
On the other hand, in heterogeneous graphs, there can be many different types of nodes and edges. An example of a network that could be modeled with a directed heterogeneous graph would be an education network. In the education network, there are nodes representing teachers, as well as nodes representing students. There are also different types of edges modeling different relationships between different types of nodes.

There are three ways of representing graphs:

1. Adjacency matrix
   An adjacency matrix is denoted with A and it shows whether there is an edge between two nodes or not. It is a square matrix of dimensions $|V| \times |V|$, where $|V|$ represents the total number of nodes in the graph. For $a_{ij} \in A$ holds:

$$a_{ij} = \begin{cases} 1 & \text{if there is an edge } e_{ij} \text{ in binary graph} \\ 0 & \text{if there is no edge } e_{ij} \text{ in binary graph} \\ w & \text{where } w \text{ is a corresponding edge weight in a weighted graph} \end{cases} \tag{1.1}$$

   An adjacency matrix is symmetric for undirected graphs. If a graph in hand is sparse, an adjacency matrix is not the best way to represent it due to the high computational cost.

2. Adjacency list
   A better way to represent large sparse graphs would be an adjacency list since it only stores each vertex's adjacent vertices. An adjacency list is also suitable for operations on vertices such as deletion, insertion, and addition of vertices.

3. Incidence matrix
   An incidence matrix is a way to capture node-edge relationships in a graph. It is a matrix of dimensions $|V| \times |E|$ in which each column is assigned a specific edge and each row represents a different node in a directed graph.
   Elements of the matrix for the directed graph are either 1—meaning that there is a connection between column edge and row node in a way that the edge is outgoing from the node, 0—denoting that there is no link between edge and node in hand, and -1—similar to the first case with an exception that column edge is incoming edge to the given node. If the graph in hand is undirected, elements of the matrix are only 0 and 1 depending on whether the connection between the column edge and row node is present.

1.2 Graph Embeddings Formal Setup

There are usually two ways of dealing with graph-based problems. We either use an original graph adjacency matrix, or represent a network in a vector space while trying to preserve its properties. Attaining such an embedding is what allows for using different techniques. There are various definitions of *graph embeddings* - one relates to representing a whole graph in vector space. At the same time, the other assumes representing each sole node in a vector space.
Deriving a vector representation of every node of a graph proposes a few challenges:

1. Choosing the property
   A vector representation of nodes is well-defined if it conserves the graph structure and links among the nodes. But how does one choose which property embedding should preserve? There are many options for the choice of the property since there are many metrics and properties related to graphs.

2. Scaling the network
   Another challenge put in front of the embedding is related to scaling large networks. Many networks in the real world are of considerable size, containing millions of vertices and edges. The embedding must pass the test of scalability and be able to deal with large graphs with minimal information loss. This is not an easy task especially when we aim to preserve the overall properties of the graph.

3. Choosing the number of the dimensions
   Yet another choice worth being considered is choosing the optimal dimensions of the embedding. While leaning toward a higher number of dimensions allows more precise reconstruction there is a bigger computational cost associated with it. On the other hand, choosing fewer dimensional embedding is computationally less expensive but can lead to losing important information about the network. That is not always the case though. Depending on the task at hand, choosing fewer dimensions can be the right choice as in predicting link accuracy while only taking into account local links among nodes.

Let us consider a weighted graph with the edge weights denoted by w_{ij} and present three proximity measures:

1. First-order proximity
   As mentioned before, edge weights are indicators of how similar the nodes connected by it are. They represent the first such a measure and therefore are referred to as first-order proximities. First-proximity measures capture local graph structure property and as such are not sufficient for conserving the global network structure[1].

2. Second-order proximity
   The second-order proximity takes into account how similar the neighborhoods of the two nodes are. Firstly, we compute the neighborhood of each node meaning we look at the weights of links present for each node. Then we say two nodes are similar if their neighborhoods are similar. Second-order proximity allows us to capture global network structure property.

3. Higher-order proximities
   Many metrics can be used to define higher-order proximities such as Rooted PageRank, Katz Index, Common Neighbors, and others. However, second-order proximity gives good enough results in a majority of graph embedding methods.

An embedding maps every vertex to a low-dimensional vector of features while simultaneously trying to preserve graph structure properties and strength of the links among the nodes.
Mathematically, graph embedding can be defined as a mapping $f : v_i \rightarrow y_i \in \mathbb{R}^d$, for every node $v_i \in V$, such that the dimension d is significantly lower than the order of the graph i.e. $|V|$. The mapping f should be able to conserve proximity measures of the graph [2].
Embedding techniques learn representations by solving an optimization problem. For example, if the embedding has a goal of preserving first-order proximity one way to achieve that would be by minimizing the following:

$$\sum_{i,j} w_{ij}||y_i - y_j||^2$$

where (v_i, v_j) and (v_i, v_k) are two node pairs with link strengths w_{ij}, w_{ik} such that $w_{ij} > w_{ik}$.Since the nodes v_i and v_j have a stronger connection they will be mapped to points closer to each other in the embedding space [2].
The $|| \cdot ||$ is notation used for the Euclidean, i.e, L_2 norm that is defined for a vector

$\mathbf{x} = (x_1, x_2, ..., x_n)$ in $\mathbb{R}^n$ as follows

$$||x|| = (\sum_{i=1}^{n} |x_i|^2)^{\frac{1}{2}}$$

Graph embedding techniques have three cornerstones [1]:

1. Preservation of graph structure property
2. Similarity measures of nodes in both the original and mapped space
3. Encoder

2. Overview of Graph Embedding Methods

Graph embedding methods can be categorized into three wide categories: Factorization based, Random Walk based and Deep Learning based [2].
In this chapter, we give an overview of each category.

2.1 Factorization based Methods

Factorization based methods represent the link among the vertices in a matrix form and proceed to factorize this matrix to obtain the new representation of the original graph. Some of the matrices commonly used to capture the links among nodes are the adjacency matrix, node transition probability matrix, Laplacian matrix, Katz similarity matrix, and others. Depending on the properties of the matrix we can use different techniques to factorize it. For example, if the matrix in hand is positive semidefinite, then eigenvalue decomposition is a good choice. On the other hand, for unstructured matrices gradient descent methods can be used to derive the graph embedding.
In the following table, we present the Factorization based methods, along with information about their time complexity and which proximity order they preserve [2].

Table 1. *Factorization based methods*

Nr	**Name of the Method**	**Properties perserved**	**Time Complexity**
1	Locally Linear Embedding (LLE)	1^{st} order proximity	O($\|E\|d^2$)
2	Laplacian Eigenmaps	1^{st} order proximity	O($\|E\|d^2$)
3	Graph Factorization	1^{st} order proximity	O($\|E\|d$)
4	GraRep	1 - k^{th} order proximities	O($\|V\|^3$)
5	HOPE	1 - k^{th} order proximities	O($\|E\|d^2$)

2.2 Random Walk based Methods

Random walks are beneficial if the graph is too large to be observed and measured as a whole. They have been utilized to estimate many properties of the graph such as node similarity and centrality. Some of the embedding methods based on random walks that carry out node representations are $DeepWalk$ and $node2vec$.

Table 2. *Random walk based methods*

Nr	Name of the Method	Properties perserved	Time Complexity
1	DeepWalk	1 - kth order proximities	O($\|V\|d$)
2	*node2vec*	1 - kth order proximities	O($\|V\|d$)

2.3 Deep Learning based Methods

With the rise of deep learning research, there are many deep neural network based approaches applied to graph data. The non-linear structure of the graph data can be captured using deep autoencoders. And they are precisely used to generate an embedding model. The deep learning based methods are Structural Deep Network Embedding (SDNE), Deep Neural Networks for Learning Graph Representations (DNGR), and Graph Convolutional Networks (GCN). In the following table, we list their properties preservation and time complexity information.

Table 3. *Random walk based methods*

Nr	Name of the Method	Properties perserved	Time Complexity
1	SDNE	1st & 2nd order proximities	O($\|V\|\|E\|$)
2	DNGR	1 - kth order proximities	O($\|V\|^2$)
3	GCN	1 - kth order proximities	O($\|E\|d^2$)

3. Locally Linear Embedding

Locally Linear Embedding (LLE) is an unsupervised learning algorithm that derives low-dimensional embeddings of high-dimensional inputs, while preserving information about the neighborhood structure of the graph. This method is presented at [3],[4]. It addresses the problem of nonlinear dimensionality reduction.

3.1 The Intuitive Notion of Manifolds

A *manifold* is a concept that describes topological space that locally resembles Euclidean space, but can have a more complex structure on a global level. It is a generalization of line, curve, plane, or sphere terms. Intuitively, we can think of it as taking an object from $\mathbb{R}^k$ and trying to place it into $\mathbb{R}^n$, where $n > k$ [5].

Let us start with a line, which is of course one dimensional. We can embed a line in two dimensions by wrapping it around into a circle. If we now look at each arc of the circle, locally it seems closer to a one-dimensional line segment. This way we embedded a one-dimensional manifold - a circle, in two dimensions. Some other examples of one-dimensional manifolds are parabolas, hyperbolas, and ellipses.

Moving onto the two-dimensional manifolds, the simplest example is a sphere. Locally, each patch of the sphere resembles a 2D Euclidean plane. More examples of 2D manifolds include torus, double torus, Klein bottle, and cross surface as shown in Figure 1.

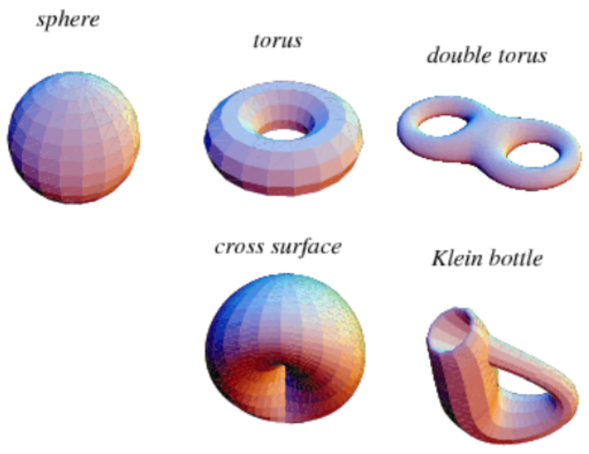

Figure 1. *Non-intersecting closed surfaces in* $\mathbb{R}^3$. *Examples of 2D manifolds, (source:[5])*

The neighborhood of each point on these manifolds locally resembles a two-dimensional plane. An appropriate analogy is Earth. When we stand on the ground Earth looks flat. There are also higher-dimensional manifolds embedded in even larger dimensions. However, they can not be visualized.
In machine learning manifolds are a central term of the *manifold hypothesis*. This hypothesis states that *"real-world high dimensional data (such as images) lie on low-dimensional manifolds embedded in the high dimensional space"* [5].
The idea is that there is some lower-dimensional representation of high-dimensional real-world data. Let us take as an example RGB images having dimensions $512 \times 512 \times 3$. They may lie on a lower-dimensional manifold compared to their original dimensions. This reasoning makes sense because we can, for example, learn to classify these images in a capacity-limited neural network. If not, learning an $512 \times 512 \times 3$ function would be infeasible .
The topic of dimensionality reduction was previously explored by classical approaches that include principal component analysis (PCA) as well as multi-dimensional scaling and neural network based approaches. The majority of these approaches try to solve a nonlinear optimization problem, usually using gradient descent which only guarantees derivation of a local optimum. Most of these approaches do not take into account the structure of the manifold on which data may lie.

3.2 Introduction

LLE tries to discover nonlinear structures in high-dimensional data by making use of the local symmetries of linear embeddings. LLE maps its inputs into a sole global coordinate system of fewer dimensionality and manages to recover global nonlinear structure from locally linear fits.
Locally Linear Embedding is an eigenvector method aimed at solving the problem of nonlinear dimensionality reduction. Its optimizations do not relate to local minima. Two popular eigenvector methods that preceded the LLE are principal component analysis (PCA) and multidimensional scaling (MDS). Both of these methods are designed to mould linear variabilities in data of high dimensions. Although both of these methods are easy to implement and their optimizations do not include local minima, they have their limitations as linear models. LLE addresses the problem of mapping high-dimensional data into a sole global coordinate system of fewer dimensions.
It is based on a simple geometric intuition.
Assume that data is comprised of N real-valued vectors $\vec{x_i}$, $i = 1, ..., N$, of dimensionality D. Data points are sampled from some underlying manifold, and we expect data points and their neighbors to be located on or at least close to a locally linear patch of the manifold. The local geometry of those patches is identified by linear coefficients that allow the

reconstruction of each data point from its neighbors. There are many ways in which we can identify neighbors of the data point. The basic formulation of LLE locates the K neighbors using the Euclidean distance. Alternatively, one can choose different rules based on local metrics.
The cost function used to measure the reconstruction error is the following:

$$\epsilon(W) = \sum_i ||\vec{x_i} - \textstyle\sum_j w_{ij}\vec{x_j}||^2 \tag{1}$$

This cost function summarizes the squared distances among all data points and their reconstructions. Reconstruction weights w_{ij} outline the contribution of the jth data point to the ith reconstruction and are computed by minimizing the cost function with two constraints:

1. Each data point $\vec{x_i}$ is reconstructed using only its neighbors, meaning that $w_{ij} = 0$ if $\vec{x_j}$ is not the neighbor of $\vec{x_i}$.
2. The rows of the weight matrix add to one ($\sum\limits_j w_{ij} = 1$).

Keeping in mind these constraints, optimal weights are achieved by solving a least-squares problem.
The described weights have an important symmetry. Namely, for each data point, they are invariant to rotations, resizing, and translations of both that one data point and its neighbors. This symmetry implies that the reconstruction weights identify the inherent geometric properties of each neighborhood. The latter constraint is responsible for enabling the invariance of translations, while the invariance to rotations and resizing follows from the definition of equation (1).
The idea that the LLE uses to construct its neighborhood-preserving mapping is that the same weights w_{ij} that reconstruct the data point in higher D dimensions should also allow us to remake its embedded manifold coordinates in fewer d dimensions. Intuitively, we can think of it as breaking locally linear patches of the underlying manifold and placing the pieces into desired low-dimensional embedding space. Suppose that we managed to preserve the angles that each data point forms with its nearest neighbors. Under this assumption, placing each patch into the embedding space requires no more than rotation, resizing, and translation of its data point. And those are precise operations to which weights are invariable. That is the reason why we expect that we can use the same weights to rebuild data points from their neighbors in a low-dimensional space.
The final step of the algorithm is reserved for mapping each observation $\vec{x_i}$ from high-dimensional space to a vector $\vec{y_i}$ of fewer dimensions. The vector $\vec{y_i}$ represents the general inner coordinates on the manifold and its d coordinates are chosen in a way to minimize the following embedding cost function:

$$\phi(Y) = \sum_i ||\vec{y_i} - \sum_j w_{ij}\vec{y_j}||^2 \tag{2}$$

The latter cost function, as the first one, is based on locally linear reconstruction errors. The difference is that here the weights w_{ij} are fixated while we optimize for the coordinates $\vec{y_i}$. The cost function (2) explicates a quadratic form in the vectors $\vec{y_i}$ and it can be minimized by finding a solution to a sparse $N \times N$ eigenvector task.
The reconstruction weights w_{ij} for each data point are derived only from its local neighborhood and as such are independent of the weights for the rest of the data points. On the other hand, the embedding coordinates are derived using the global operation that combines all data points into connected components of the network determined by the weight matrix. By computing the bottom eigenvectors one at a time from the second equation we can determine the different dimensions in the embedding space. We should keep in mind that derivation is always coupled across data points which allows the algorithm to exploit overlying local information and uncover the global structure.

3.3 Algorithm

The summary of the Locally Linear Embedding (LLE) algorithm in three steps is:

1. Select the neighbors
   Find the neighbors of each high-dimensional data point, $\vec{x_i}$. This can be done with the K nearest neighbors algorithm.
2. Reconstruct with linear weights
   By solving the least-squares problem in (1), while keeping in mind presented constraints, compute the weights w_{ij} that allow for the best linear reconstruction of $\vec{x_i}$ from its neighbors.
3. Map to embedded coordinates
   Derive the low-dimensional embedding vectors $\vec{y_i}$ that are rebuilt by the reconstruction weights w_{ij}, and are minimizing cost function (2). The constraint that points are solely reconstructed from neighbors allows for highly nonlinear embeddings.

Implementation of the algorithm is simple. Data points are rebuilt using their K nearest neighbors that are determined using either Euclidean distance or normalized dot products. The only free parameter in this algorithm is the number of neighbors, K. Once the neighbors are fixated we search for the optimal weights w_{ij} and coordinates $\vec{y_i}$. The algorithm manages to reach a global minimum of the reconstruction and embedding costs provided in equations (1) and (2), respectively [3].
It can happen, although rarely, that the number of neighbors K is higher than the dimension

of the input D. In such a scenario one solution could be to add a regularization that penalizes the squared measures of the weights to the reconstruction cost.
Although the algorithm takes as input N high dimensional data points $\vec{x_i}$, in many cases, data of this form may not be available. Instead of the high dimensional vectors $\vec{x_i}$ one can have at disposal only the measures of dissimilarity or coupled distance among different data points. With a simple change, LLE can deal with input data in such a form. That way LLE requires only a small part of all feasible pairwise distances. Namely distances between neighboring points and their corresponding neighbors.

Algorithm 1 Locally Linear Embedding Algorithm

```
Our input $X$ is a matrix of dimensions $D \times N$, composed of N data points in D
dimensions. The desired output is a matrix $Y$ having the dimensions $d \times N$, composed
of embedding coordinates of the input points in $d-$ dimensional embedding, $d < D$.
**1. Compute neighbors in input space, X**
1 : for $i = 1, ..., N$:
2 :     compute the Euclidean distance from $\vec{x_i}$ to the rest of the points $\vec{x_j}$
3 :     obtain the K smallest distances
4 :     assign the respective data points to be neighbors of $\vec{x_i}$
**2. Derive reconstruction, linear weights** $W$
5 : for $i = 1, ..., N$:
6 :     compute the matrix $Z$ that contains all of the neighbors of $\vec{x_i}$, but not $\vec{x_i}$ itself
7 :     Deduct $\vec{x_i}$ from every column of matrix $Z$
8 :     Derive the local covariance $C$, as $C = Z^T Z$
9 :     Find a solution for linear system $Cw = 1$, with respect to $w$
10 :    Set the value of $w_{ij}$ to 0 if $x_j$ is not the neighbour of $x_i$
11 :    The remaining elements of the $ith$ row of $W$ should be set to $\frac{w}{sum(w)}$
**3. Derive embedding coordinates** $Y$ **with weights** $W$
12 :    Compute sparse matrix $M$, as $M = (I - W)^T (I - W)$
13 :    Obtain the bottom $d + 1$ eigenvectors of $M$
14 :    The $qth$ row of $Y$ should be set to the $q + 1$ smallest eigenvector
         ▷ the bottom vector $[1, 1, 1, ...]$ with corresponding eigenvalue zero is discarded
```

In the above algorithm $\vec{x_i}$ and $\vec{y_i}$ represent the ith column of matrices X and Y, respectively, i.e., the data and the embedding coordinates of the ith data point. I is the identity matrix and 1 is the column vector of $1s$.
Computing the K nearest neighbors in step 1 can be done in many different ways. Instead of using Euclidean distance, one can try to include all the points within a fixed radius or use some other domain-specific local distance metrics. There are well-run techniques for computing the neighbors of each point such as KD trees.
In step 2, if $K > D$, the local covariance will not be full rank and should be regularized. This can be done by setting it to $C = C + \epsilon I$, where ϵ is a small constant. In this way, we ensure that the linear system in step 2 has a unique solution.

3.4 Discussion

The advantage of the LLE is that it does not involve many free parameters such as the learning rate, convergence criteria, and others. Let us investigate the time complexities of different steps of LLE. Computing the nearest neighbors in the first step has $O(DN^2)$ scalability. Meaning, it scales linearly in the dimensionality of the input and quadratically in the number of data points. In the second step, reconstruction weights w_{ij} are computed in $O(DNK^3)$ time. That is the number of operations needed to solve a system of linear equations of the dimension $K \times K$ for each data point. Deriving the bottom eigenvectors in the third step scales linearly regarding embedding dimensions d and quadratically regarding the number of data points N,i.e., $O(dN^2)$. As we add more dimensions to the embedding space the previous one does not change, hence we do not rerun LLE to derive higher dimensional embeddings. The size of the weight matrix, $N \times K$, restricts the storage requirements of LLE.
LLE demonstrates a universal concept of working with manifolds. Namely, if we analyze overlapping local neighborhoods all together we can derive conclusions about global structure. A big advantage of the LLE is that it does not require solving any big dynamic programming task. It also collects sparse matrices whose structure allows for time and space computational reduction.
Combining LLE with different methods can yield more useful results in statistics and data analysis. One direction would be to understand how to learn a parametric mapping from the observation to the embedding space using the conclusions of the LLE. One can try and use pairs $(\vec{x_i}, \vec{y_i})$ as labeled samples for supervised learning models. Learning such a mapping would make LLE widely applicable in many areas of information science.

3.5 Example

The following example showcases the nonlinear dimensionality reduction of three-dimensional data to a two-dimensional embedding space. The data is selected from a two-dimensional manifold. We can notice that LLE managed to discover the mapping that successfully preserved the neighborhood of the data points [4].

The input to the LLE algorithm was $N = 1000$ points selected from the $Swiss\ roll$ two-dimensional manifold. The algorithm used $k = 15$ neighbors per each data point [6].

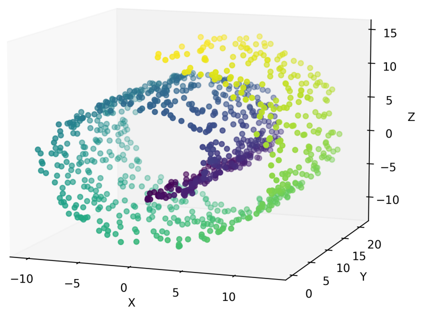

Figure 2. *Original 3D data in the form of a Swiss roll*

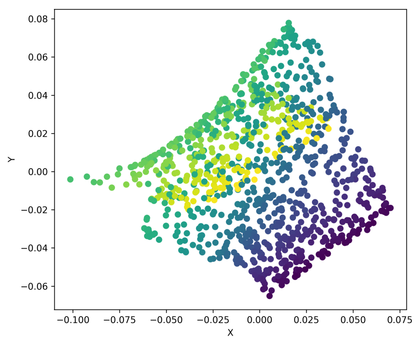

Figure 3. *LLE dimension reduction with k = 15 neighbors*

4. Laplacian Eigenmaps

Laplacian Eigenmaps is a geometrically inspired, dimensionality reduction algorithm presented at [7], [8]. It is a method for "*constructing a representation for data sampled from a low dimensional manifold embedded in a higher dimensional space*" [7] while preserving the local properties and naturally being prone to clustering.

4.1 Introduction

Laplacian Eigenmaps is a method that uses the Laplacian of the graph and derives a low-dimensional representation of the data set that includes local neighborhood information. This representation map can be perceived as a *"discrete approximation to a continuous map that naturally arises from the geometry of the manifold"* [8].
Let us highlight the main points of the algorithm. Its core is simple, having few local computations and a sparse eigenvalue problem, while its solution contemplates the intrinsic geometric shape of the manifold to which the data belongs. This is made possible by observing that the Laplacian matrix of the graph, acquired through data points, can be seen as an estimation of the *Laplace Beltrami* operator established on the manifold.
The Laplace Beltrami operator has a role in providing an appropriate embedding for the manifold. We approximate the manifold by the adjacency graph obtained from data points. The Laplace Beltrami operator is estimated using the weighted Laplacian of the adjacency graph with the optimally chosen weights. The connection between Laplacian and the heat kernel allows us to choose the weights of the graph. As a result, the embedding maps of the data points estimate the eigenmaps of the Laplace Beltrami operator. The graph Laplacian has been broadly utilized for different clustering tasks. The connections between the graph Laplacian and the Laplace Beltrami operator are explicitly used in this framework to explain dimensionality reduction task geometrically.
As we said, this algorithm preserves the local properties of the graph and it is thus less sensitive to outliers and noise. By trying to conserve the local information in the embedding, it highlights the natural clusters of data points. From that perspective, we can see how clustering and dimensionality reduction are connected. Note that this method does not yield meaningful clusters for all data sets. However, we will provide an example for which that is the case.
Given that this method stems from the intrinsic geometric manifold's structure, it has stability concerning embedding. For isometric embeddings, the representation stays the same. In the RGB image example, different choices of height and width of the image, i.e.,

the image resolution, will result in embedding the same underlying manifold into spaces of different dimensions. However, this algorithm will compute similar representations separate from resolution.

4.2 Algorithm

The general dimensionality reduction setup is similar to the graph embedding definition. Having a set of n points $x_1, x_2, ..., x_n \in \mathbb{R}^l$ we are looking for set of points $y_1, y_2, ..., y_n \in \mathbb{R}^d$, where $d << n$, such that the point y_i is an representation of point x_i. In this case points $x_1, x_2, ..., x_n \in M$, where M is a manifold embedded in $\mathbb{R}^l$.
The following algorithm aims to construct the optimal representations, i.e., set of points y_i, $i = 1, ..., n$.

Algorithm 2 Laplacian Eigenmaps Algorithm

```
Our input are $n$ points $x_1, x_2, ..., x_n \in \mathbb{R}^l$. The desired output is a weighted graph $G$ with
$n$ vertices (corresponding to each of $n$ points) and the set of edges joining neighboring
points. The embedding is obtained by deriving the eigenvectors of the graph Laplacian.
1. Compute the (adjacency) graph G
1 : The edge between nodes $i$ and $j$ is present if $x_i$ and $x_j$ are close to each other, for
which we have two variations:
2 :     $(i)$ $||x_i - x_j||^2 < \epsilon$, where $\epsilon \in \mathbb{R}$ is fixed.
3 :     $(ii)$ $i$ is among $j$'s $k$ nearest neighbors, or $j$ is among $i$'s $k$ nearest neighbors.
2. Selecting the weights
4 : Again, we have two variations for choosing the edge's weights:
5 :     $(i)$ If there is a present link between nodes $i$ and $j$, set $w_{ij} = e^{-\frac{||x_i - x_j||^2}{t}}$, $t \in \mathbb{R}$,
else set $w_{ij}$ to 0.
6 :     $(ii)$ If there is a present link among nodes $i$ and $j$ set $w_{ij} = 1$, else $w_{ij} = 0$
3. Eigenmaps
7 : Assume graph G we just obtained is connected, or proceed using its connected
components.
8 : Derive eigenvalues $(\lambda_0, \lambda_1, ..., \lambda_{n-1})$ and eigenvectors $(\mathbf{f_0}, \mathbf{f_1}, ..., \mathbf{f_{n-1}})$, for eigenvec-
tor problem: $L\mathbf{f} = \lambda\Delta\mathbf{f}$
9 : $\Delta$ is a diagonal weight matrix, $\Delta_{ii} = \sum_j w_{ji}$ and $L$ is the Laplacian matrix defined
as $L = \Delta - W$
10 : We dismiss the eigenvector $\mathbf{f_0}$ whose corresponding eigenvalue is $0$
11 : The next $d$ eigenvectors are used for embedding in $d$-dimensional Euclidean space:
$x_i \to (\mathbf{f_1}(i), \mathbf{f_2}(i), ..., \mathbf{f_d}(i))$.
```

Let us explain in more detail the algorithm we just presented.
In the first step, in which we compute the graph G, we presented two options for reasoning if the points x_i and x_j are close. Each of them has its advantages and disadvantages. The first one (line 2) can be referred to as ϵ-neighborhoods. The norm used is the Euclidean norm in $\mathbb{R}^l$. The advantage of using this approach is that it is geometrically motivated and it is symmetric by definition. Disadvantages are that it can often result in graphs with few connected components and it is not obvious how to choose a proper value for ϵ.
The second option (line 3) can be referred to as k nearest neighbors, $k \in \mathbb{N}$. The advantages of choosing this approach are that this relation is symmetric, does not tend to result in disconnected graphs and the value of k is easier to select. On the other hand, its disadvantage is that it is less geometrically intuitive.
In step two we again have two options. The first one (line 5) is called Heat kernel and it has parameter $t \in \mathbb{R}$. The second one (line 6) is a simplification of the first one (for $t = \infty$) as it does not require choosing t.
In the third step, we are solving a general eigenvector problem:

$$L\mathbf{f} = \lambda \Delta \mathbf{f} \qquad \text{i.e.}$$
$$L\mathbf{f}_0 = \lambda_0 \Delta \mathbf{f}_0$$
$$L\mathbf{f}_1 = \lambda_1 \Delta \mathbf{f}_1$$
$$...$$
$$L\mathbf{f}_{n-1} = \lambda_{n-1} \Delta \mathbf{f}_{n-1}$$
$$0 = \lambda_0 \leqslant \lambda_1 \leqslant ... \leqslant \lambda_{n-1}$$

Δ is a diagonal weight matrix, whose entries are column or row sums of matrix W, and L is a Laplacian matrix, defined as $L = \Delta - W$. Laplacian matrix is positive semidefinite and symmetric, and we can think of it as an operator on functions defined on nodes of graph G.
Now, let us discuss why the embedding derived in the Laplacian Eigenmap algorithm optimally conserves local information. The answer is based on standard spectral graph theory.
We compute a weighted graph $G = (V, E)$ with links among the nearby points. Let us also assume that the graph G is connected. We are looking at the special case where $d = 1$, i.e., the problem of mapping the weighted graph G to a line in a way that the points among whom there is a link stay close together. We denote such a map as $\mathbf{y} = (y_1, y_2, ..., y_n)^T$. A good map can be obtained by minimizing the objective function under proper constraints:

$$\sum_{i,j} (y_i - y_j)^2 w_{ij}$$

This objective function together with our selection of weights w_{ij} imposes a penalty if points x_i and x_j, which are close to each other, are mapped apart. By minimizing it we are trying to make sure that if points x_i and x_j are neighboring, then corresponding y_i and y_j will be neighboring as well.

$$\forall \mathbf{y}, \mathbf{y}^T L \mathbf{y} = \tfrac{1}{2} \sum_{i,j} (y_i - y_j)^2 w_{ij}$$

where $L = \Delta - W$, W is symmetric and $\Delta_{ii} = \sum_j w_{ij}$. This relation also implies that L is positive semidefinite.
Our minimization problem becomes finding

$$\underset{\mathbf{y}, \mathbf{y}^T \Delta \mathbf{y} = 1}{\operatorname{argmin}} \mathbf{y}^T L \mathbf{y}$$

We have a constraint $\mathbf{y}^T \Delta \mathbf{y} = 1$ that takes out a random scaling factor in the embedding. The matrix Δ gives us a natural measure of the nodes of the graph. Meaning that, the larger the value Δ_{ii} is, the node corresponding to it is more significant. Matrix L is positive semidefinite. The vector $\mathbf{y}$ that minimizes the objective function is stated by the minimum eigenvalue solution to the general eigenvalue problem:
$L\mathbf{y} = \lambda \Delta \mathbf{y}$
Constraint, as it is now, has a trivial solution, constant function $\mathbf{1}$ that sends all nodes of the graph G to the real number 1. For connected graphs, $\mathbf{1}$ is the only eigenvector with eigenvalue 0. To avoid this trivial case we introduce a constraint of orthogonality and search for

$$\underset{\mathbf{y}^T \Delta \mathbf{1} = 0, \mathbf{y}^T \Delta \mathbf{y} = 1}{\operatorname{argmin}} \quad \mathbf{y}^T L \mathbf{y} \ .$$

In this way, the solution is obtained as an eigenvector with the corresponding smallest nonzero eigenvalue.
In the more general setup, when we are embedding the graph into d-dimensional Euclidean space, the embedding is given as the $n \times d$ matrix $\Upsilon = [\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_d]$. The embedding coordinates of the ith node are given as the ith row of the matrix Υ. Similar to before, we are trying to minimize the following

$$\sum_{i,j} ||\mathbf{y}^{(i)} - \mathbf{y}^{(j)}||^2 w_{ij} = tr(\Upsilon^T L \Upsilon)$$

and $\mathbf{y}^{(i)} = [\mathbf{y}_1(i), \mathbf{y}_2(i), ..., \mathbf{y}_d(i)]^T$ is the ith's node d-dimensional representation. This simplifies to searching for

$$\underset{\Upsilon^T \Delta \Upsilon = I}{\text{argmin}}\, tr(\Upsilon^T L \Upsilon)$$

We have again introduced the constraint, which, in the d-dimensional embedding problem, prevents collapse onto a subspace of dimension less than d (if we request orthogonality to the constant vector) or less than $d-1$ (otherwise). The solution is obtained by the matrix of eigenvectors corresponding to the lowest eigenvalues of $L\mathbf{y} = \lambda \Delta \mathbf{y}$.

4.3 The Laplace Beltrami Operator

As we said in the introduction, there is an analogy between the Laplacian of the graph and the Laplace Beltrami operator on manifolds. We proceed to explain the properties of eigenfunctions of the Laplace Beltrami operator that make them suitable for embedding.
Let us consider a smooth, compact, d-dimensional Riemannian manifold M. The Riemannian structure on the manifold is generated by the standard Riemannian structure on $\mathbb{R}^l$ in case the manifold is embedded in $\mathbb{R}^l$ [9]. Once again we aim to map a manifold to the real line in a way that points which are close to each other on a manifold are close together on a line as well.
Let us denote such a map by φ, $\varphi : M \to \mathbb{R}$. And let us assume that φ is twice differentiable. We are observing two neighboring points $\mathbf{x}, \mathbf{h} \in M$ and their respective mappings, $\varphi(\mathbf{x})$ and $\varphi(\mathbf{h})$.
The first-order Taylor approximation of the mapping φ at the point x tells us that, for any h in the neighborhood of x, the following holds:

$$\varphi(\mathbf{h}) = \varphi(\mathbf{x}) + \nabla\varphi(\mathbf{x})(\mathbf{h}-\mathbf{x}) + o(||\mathbf{h}-\mathbf{x}||)) \qquad \text{i.e.}$$

$$\varphi(\mathbf{h}) - \varphi(\mathbf{x}) = \nabla\varphi(\mathbf{x})(\mathbf{h}-\mathbf{x}) + o(||\mathbf{h}-\mathbf{x}||))$$

Which becomes

$$|\varphi(\mathbf{h}) - \varphi(\mathbf{x})| = |\langle \nabla\varphi(\mathbf{x}), \mathbf{h}-\mathbf{x} \rangle| + o(||\mathbf{h}-\mathbf{x}||))$$

The Cauchy-Schwartz inequality gives us

$$|\varphi(\mathbf{h}) - \varphi(\mathbf{x})| \leqslant ||\nabla\varphi(\mathbf{x})||\, ||\mathbf{h}-\mathbf{x}||_{dist_M} + o(||\mathbf{h}-\mathbf{x}||_{dist_M}))$$

If manifold M is isometrically embedded in $\mathbf{R}^l$ holds the following
$dist_M(\mathbf{x}, \mathbf{h}) = ||\mathbf{h}-\mathbf{x}||_{\mathbb{R}^l} + o(||\mathbf{h}-\mathbf{x}||_{\mathbb{R}^l})$ and

$$|\varphi(\mathbf{h}) - \varphi(\mathbf{x})| \leqslant ||\nabla\varphi(\mathbf{x})||\, ||\mathbf{h}-\mathbf{x}||_{\mathbb{R}^l} + o(||\mathbf{h}-\mathbf{x}||_{\mathbb{R}^l}).$$

Hence, $||\nabla\varphi(\mathbf{x})||$ gives us an estimate of how distant nearby points are, after being mapped

by φ.
By attempting to find the

$$\underset{||\varphi||_{L^2(M)}=1}{\text{argmin}} \int\limits_M ||\nabla\varphi(\mathbf{x})||^2,$$

we are searching for a map that best conserves locality on average.
Minimizing $\int\limits_M ||\nabla\varphi(\mathbf{x})||^2$ corresponds to minimizing $L\mathbf{f} = \frac{1}{2}\sum(f_i - f_j)^2 w_{ij}$, where $\mathbf{f}$ is a function defined on nodes and f_i is the value function f on the ith vertex.
Minimizing $\int\limits_M ||\nabla\varphi(\mathbf{x})||^2$ comes down to finding Laplace Beltrami's operator $(\mathcal{L})$ eigenfunctions. The Laplace Beltrami operator is defined as

$$\mathcal{L}(f) := -div\nabla(f),$$

and div represents the divergence of the vector fields.
It holds the following

$$\int\limits_M ||\nabla f||^2 = \int\limits_M \mathcal{L}(f)f$$

$\mathcal{L}$ is positive semidefinite. The function f that minimizes the left side of the latter equation must be an eigenfunction of $\mathcal{L}$. The spectrum of the Laplace Beltrami operator on a compact manifold is shown to be discrete [9]. Let us denote eigenvalues as $0 = \lambda_0 \leqslant \lambda_1 \leqslant \lambda_2 \leqslant \lambda_3 \leqslant ...$, and corresponding eigenfunctions by f_i, $i = 0, 1, 2, 3, ...$ Again, similar to the graph setting, f_0 is the constant function that sends the whole manifold to a single point. We battle this case by demanding that the embedding map f is orthogonal to f_0. Consequentially f_1 is the optimal embedding map. As in the previous section, the optimal d-dimensional embedding is provided by

$$\mathbf{x} \to (f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_d(\mathbf{x})).$$

4.4 Heat Kernels and Choosing the Weights

When discussing the Laplacian Eigenmaps algorithm, more precisely, its second step in which we choose the weights of the graph, we have offered two options. In this section, we provide a brief intuition of the Heat kernel and how it relates to our weight choice. The Laplace-Beltrami operator, acting on differentiable functions over a manifold M is closely related to heat flow. Let us consider the initial heat distribution given as $\varphi : M \to \mathbb{R}$ and the heat distribution at time $\ddot{t}$, $u(x,t)$. Note that $u(x,0) = \varphi(x)$. The heat equation is given as the following partial differential equation

$$(\tfrac{\partial}{\partial t} + \mathcal{L})u = 0$$

Its solution is given as

$$u(x,t) = \int\limits_M \chi_t(x,y)\ \varphi(y)$$

and χ_t is the heat kernel, the Green's function for this equation. As a consequence,

$$\mathcal{L}\varphi(x) = \mathcal{L}u(x,0) = -(\ \tfrac{\partial}{\partial t}\ [\ \int\limits_M \chi_t(x,y)\ \varphi(y)\ ])_{t=0}.$$

In a suitable coordinate system, we can approximate heat kernel by the Gaussian [9]:

$$\chi_t(x,y) = (4\pi t)^{-\frac{d}{2}} e^{-\frac{||x-y||^2}{4t}}\ (\psi(x,y) + O(t)),$$

and $\psi(x,y)$ is a smooth function for which holds $\psi(x,x) = 1$. With this in mind, when t is small and x and y are close to each other,

$$\chi_t(x,y) \approx (4\pi t)^{-\frac{d}{2}} e^{-\frac{||x-y||^2}{4t}}$$

Detailed explanation is provided in [9].
As $t \to 0$ the heat kernel tends to Dirac's δ-function, i.e., $\lim\limits_{t\to 0} \int\limits_M \chi_t(x,y)\varphi(y) = \varphi(x)$. For small t, we have

$$\mathcal{L}\varphi(x) \approx \tfrac{1}{t}[\ \varphi(x) - (4\pi t)^{-\frac{d}{2}} \int\limits_M\ e^{-\frac{||x-y||^2}{4t}} \varphi(y)\ dy\ ]$$

For n data points on the manifold $\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n}$, the latter comes close to

$$\mathcal{L}\varphi(\mathbf{x_i}) \approx \tfrac{1}{t}[\ \varphi(\mathbf{x_i}) - \tfrac{1}{n}(4\pi t)^{-\frac{d}{2}} \sum_{\mathbf{x}_j, ||\mathbf{x}_j - \mathbf{x}_i|| < \epsilon}\ e^{-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{4t}} \varphi(\mathbf{x_j})\ ],$$

where $\frac{1}{t}$ is a global coefficient. Since the intrinsic dimensionality of the manifold may be unknown it is set $\mu = \frac{1}{n}(4\pi t)^{-\frac{d}{2}}$. Using the fact that the Laplacian of the constant function is zero we get

$$\mu = (\textstyle\sum_{\mathbf{x}_j, ||\mathbf{x}_j - \mathbf{x}_i|| < \epsilon} e^{-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{4t}})^{-1} .$$

Now, there are few feasible approximations for the manifold Laplacian. We aim for the approximation matrix to be positive semidefinite and thus the graph Laplacian is computed with the subsequent weights:

$$w_{ij} = \begin{cases} \mathrm{e}^{-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{4t}} & \text{, when } ||\mathbf{x}_i - \mathbf{x}_j|| < \epsilon \\ 0 & \text{, otherwise} \end{cases} \tag{4.1}$$

4.5 Example

As in the previous chapter, we consider an example of a *Swiss roll*, a flat two-dimensional manifold in $\mathbb{R}^3$. We sample $N = 2000$ random data points from the *Swiss roll* and present it in Figure 4. We obtain its two-dimensional representation using *Laplacian Eigenmps* algorithm [6] with $k = 15$ number of neighbors. We see that the locality, while not the distances on the manifold are preserved.
Note that the weights are binary, either 0 or 1, meaning we do not choose the heat kernel parameter $t \in \mathbb{R}$. This simplified version of the algorithm seems to give good results in practice.

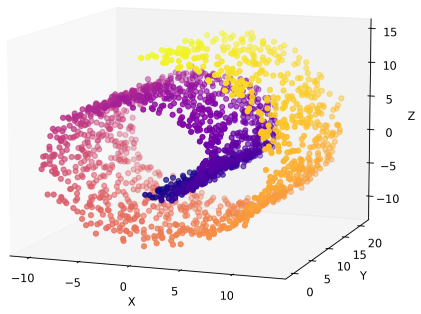

Figure 4. *Random N = 2000 data points on the Swiss Roll*

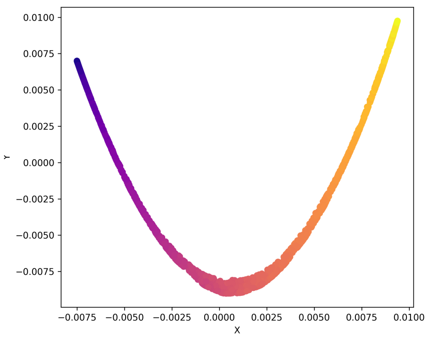

Figure 5. *Two-dimensional Laplacian Eigenmaps representation*

5. DeepWalk

$DeepWalk$ introduced at [10] is an approach that allows for learning latent representations of nodes in a network. Social relations of the nodes are encoded in a continuous vector space and then can be easily processed by known statistical methods. $DeepWalk$ bridges the gap in language modeling and unsupervised feature learning from word sequences to graphs. With $DeepWalk$ the successful deep learning techniques from natural language processing are introduced to network analysis. An algorithm $learns$ social representations of nodes by representing a stream of short random walks. By social representations, we mean latent features of the nodes that describe neighborhood properties and similarities. The social relations of the nodes are encoded into a continuous vector space with a small number of dimensions. Instead of processing the semantic structure of human language neural language models are now generalized to treat a set of randomly-made walks.
The input of a $DeepWalk$ algorithm is a graph while the output is a latent representation. Once the representation is outputted simple linear classifiers such as logistic regression can be used to obtain good results. The other advantage of a $DeepWalk$ is that it is trivially parallelizable.

5.1 Problem Setting

We are trying to classify social network members into single or more categories. Let us denote the graph with $G = (V, E)$ where, as before, V is a set of nodes, i.e., members of the network and E represents a set of edges, $E \subseteq (V \times V)$.
We consider a partially labeled network $G_l = (V, E, X, Y)$, where the attributes $X \in \mathbb{R}^{|V| \times \zeta}$, with ζ being the size of the feature space for every attribute vector, and the set of labels Υ, where $Y \in \mathbb{R}^{|V| \times |\Upsilon|}$. Compared to a traditional machine learning classification task where we are trying to learn a hypothesis Ξ mapping elements of X into the label set Υ in this approach we can make use of the important information about the dependence of the examples placed in the structure of G.
This problem is related to as the relational classification or the collective classification. Previously, this problem was addressed as reasoning in an undirected Markov network with the use of iterative approximate reasoning algorithms that aimed to derive the succeeding distribution of labels, provided the network structure. Here, a different approach is taken to encapsulate the topology information of the network. The label space is not mixed as a part of the feature space. Instead, the unsupervised method learns features that encapsulate

the structure of the network *independent* of the distribution of the labels. Separating the structural characterization and labeling task allows us to avoid the consecutive errors that can happen in iterative techniques. Also, it is possible to reuse the same representation for other classification tasks in the network.
The goal is to learn low-dimensional representation X_e, where $X_e \in \mathbb{R}^{|V| \times d}$ and d represent a small number of latent dimensions. These representations are distributed in a way that each social event is indicated by a subset of the dimensions and each dimension adds to a subgroup of the social notions expressed by the space. Taking advantage of these structural features we will change the space of the attributes to support the classification decision. The use of these features is broad: they can be used with any classification method and they integrate easily with simple machine learning algorithms. They also have the property of scaling well in real-world networks.

5.2 Social Representation Characteristics

We expect our social representations to satisfy the following characteristics:

1. Adaptability
   Since the real-world social networks are always changing we expect our representations to be able to deal with new social relations in a way that does not require the learning process to be repeatedly performed.
2. Being aware of the community
   We want to ensure that similar members of the network are concentrated together. With that in mind, the distance among latent dimensions is a metric we use to evaluate similarity among the members of the network.
3. Low dimensionality
   Low dimensional techniques generalize better, converge, and make inferences faster when the labeled data is scarce.
4. Being continuous
   Latent representations should model the limited community membership in continuous space. A continuous representation allows for more robust classification because of its straightforward decision boundaries among the communities.

These characteristics are satisfied by learning the representation for nodes from a stream of random walks, utilizing the optimization methods initially introduced for the tasks of language modeling.

5.2.1 Random Walks

Let us consider a random walk with a root at the node v_i, $\wp_{v_i}$. It represents a stochastic process with random variables $\wp_{v_i}^1$, $\wp_{v_i}^2$, $\wp_{v_i}^3$, ..., $\wp_{v_i}^k$ where $\wp_{v_i}^{k+1}$ is a node chosen randomly from the neighborhood of a node v_k. Many tasks in the domain of community detection and content recommendation relied on random walks as similarity measures.
They are also beneficial in deriving a local community structure knowledge taking sublinear time in comparison to the size of the input network in the class of output sensitive algorithms. The random walks are the basis of the $DeepWalk$ algorithm for a few reasons. Firstly, they capture the local information. Secondly, it is easy to parallelize the local exploration. Meaning that more than one random walker can explore different parts of the same network at once. Finally, since we are using a stream of short random walks and the information they provide, we can adjust to small changes in a graph structure without having to compute the model from scratch. The learned model can be iteratively updated with added random walks in a changed part of the graph in time that is sublinear to the whole graph.
Once the random walks are chosen we need an appropriate technique to capture the information about graph structure that random walks provide. If the degrees of the connected graph is distributed following a power law i.e. the distribution of the degrees is a *scale-free* then the frequency in which nodes show up in the random walks will also have a power-law distribution.
The frequency of the words in a natural language shows similar distributional behavior. Hence, the methods for language modeling take into account this distribution property. For example, it can be shown that the frequency of node occurrence in the short random walks in a scale-free graph and on the other hand, the frequency of word occurrence in the text of $Wikipedia$ articles text follow similar power-law distributions. This justifies the idea that the methods applied to modeling the natural language in which the frequency of the symbols obeys the *Zipf's law* distribution can be reused to mold the community structure in networks. We continue examining the language modeling work and adapting it to learn the representations of the nodes that meet the desired criteria.

5.2.2 Natural Language Modeling

Language modeling aims to estimate the probability that a particular sequence of words will appear in a collection of words. Mathematically, we define a sequence of words as $W_i^n = (w_0, w_1, w_2, ..., w_n)$, where $w_i \in \nu$ and ν represents a vocabulary.
The goal is to maximize the probability $P(w_n | w_0, w_1, ..., w_{n-1})$ over the whole collection of words we train on. Here we try to generalize the language modeling to explore the

network using a stream of short random walks. We think of these walks as short sentences or phrases in a language. The analogy is the following: estimate the probability of noting the node v_i, given all the previous nodes visited until now in the random walk. Mathematically, that probability is denoted by $P(v_i|(v_1, v_2, v_3, ..., v_{i-1}))$.
We want to learn a latent representation so we define a mapping $\varphi : v \rightarrow \mathbb{R}^{|V| \times d}$, $v \in V$. The function φ represents the underlying social representation related to each node v in the network. Practically, we will represent φ as a matrix of free parameters of the dimensions $|V| \times d$ and we will later on use it as our X_e. So, now we are trying to estimate the probability:

$$P(v_i|\varphi(v_1), \varphi(v_2), \varphi(v_3), ..., \varphi(v_{i-1})) \tag{3}$$

With the walk length becoming larger deriving this objective function becomes impracticable.
Luckily, certain relaxations in language modeling can be used to modify the optimization problem. The first one is that we do not use the context to predict the word that is missing but we use one word to predict the context. Secondly, both the words that are to the right side and to the left side of the word we are considering are creating the context. Finally, the ordering constraint of the problem is taken out and we require that the model maximizes the likelihood of any word showing up in the context in the absence of information about its offset from the observed word. Related to node representation modeling, the optimization problem becomes:

$$\min_{\varphi} - logP((v_{i-k}, ..., v_{i-1}, v_{i+1}, ..., v_{i+k})|\varphi(v_i)) \tag{4}$$

The mentioned relaxations are very useful for social representation learning. Namely, assuming the order independence allows us to better capture the information about the nearness of the nodes that random walks provide. This relaxation also allows us to speed up the training process since we are building small models with one node at a time.
By solving the optimization problem represented by (4) we will get representations that preserve the common similarities among the nodes in a local graph structure. Nodes with similar neighborhoods will obtain alike representations.
Short random walks combined with neural language models create a method that meets all of the desired criteria. The representations of the social networks generated by this model are low-dimensional and in a continuous vector space. They encode underlying forms of community membership. The method computes the helpful intermediate representations and thus it is adaptable to dynamic network topology.

5.3 Algorithm

Following a theme of language modeling methods the only requirements for the $DeepWalk$ as an input are a corpus and a vocabulary ν. In the case of a $DeepWalk$ the corpus is a set of short truncated random walks and the vocabulary is a set of the graph nodes, meaning $\nu = V$. Although it is useful to have a knowledge of V and the frequency distribution of nodes in the random walks before the training process, it is not mandatory for the algorithm's efficiency.

Algorithm 3 DeepWalk

```
Our input is a graph $G(V, E)$, as well as window size $w$, the size of the embedding $d$,
walks per node $\upsilon$ and walk length $\tau$. The desired output is a matrix of node representa-
tions $\varphi \in \mathbb{R}^{|V|\times d}$.
**1.Initialization**
1 : Sample $\varphi$ from $U^{|V|\times d}$
2 : Construct a binary tree $T$ from $V$
**2.Main part**
3 : for $i = 0, ..., \upsilon$:
4 :     $o$ =Shuffle($V$)
5 :     for $v_i \in o$:
6 :         $\wp_{v_i}$ = $RandomWalk(G, v_i, \tau)$
7 :         SkipGram($\varphi$,$\wp_{v_i}$, w)                ▷ Updating the representations
```

The algorithm has two main parts: a random walk generator and an update procedure. The role of the random walk generator is to take a graph G and sample uniformly a random node v_i as the starting point of the random walk $\wp_{v_i}$. At each step, until the maximum length τ is obtained the walk chooses uniformly from the neighborhood of the last visited node. The random walks do not have to be of the same length. They can have restarts, i.e., returning to the root, but having them would not yield any improvements. This implementation fixes the number of random walks υ of length τ starting at each node.

In the main part of the algorithm, the outer for loop states the number of walks υ that we should start at each node. In each iteration, we pass over the data and choose one walk per node in this pass. To speed up the convergence of the stochastic gradient descent, at the beginning of each pass a random ordering to traverse the nodes is generated. In the inner for loop the algorithm iterates over all the nodes of the network. A random walk $\wp_{v_i}$ of the length τ is generated for each node v_i and then utilized to update the representations. The updating of the representations is done with the $SkipGram$ algorithm in correspondence with the objective function from (4) [10].

Instead of inferring the present word based on the context, the continuous Skip-gram model [11] aims to maximize the classification of a word on the basis of the other word in the same sentence. The model uses a log-linear classifier with a continuous projection layer and feeds each current word as input. Predictions are words within a certain scope before

and after the present word. Increasing the scope gives the resulting word vectors a higher quality with the increased computational cost. The words more apart from the current word are commonly less related compared to those close to it. This is taken into account by giving less weight to the far-apart words by taking fewer samples from those words in the training examples.
The complexity of the training process for this model is proportional to
$q = \rho \times (d + d \times \log_2(V))$, where ρ is the maximum distance of the words, V is the size of the vocabulary. For example, if we set ρ = 5 for each training word we choose a random number $r < \rho$. We proceed with using r words from before and r words after the current word as correct labels. With the current word as input and each of the $r + r$ words as output, we end up doing $r \times 2$ classifications. To summarize, the $SkipGram$ predicts surrounding words given the current word.

Algorithm 4 SkipGram

```
Our input is $\varphi$, $\wp_{v_i}$ as well as window size $w$.
1 : for $v_j \in \wp_{v_i}$:
2 :     for $u_k \in \wp_{v_i}[j-w, j+w]$ :
3 :         $J(\varphi) = -logP(u_k|\varphi(v_j))$
4 :         $\varphi = \varphi - \alpha\frac{dJ}{d\varphi}$
```

The main idea of the $SkipGram$ algorithm is that within a window of the size w we iterate over all possible collocations in a random walk. For each collocation, we map each node v_j to its present representation vector $\varphi(v_j) \in \mathbb{R}^d$. Having the representation of v_j in line 3, we aim to maximize the likelihood of neighbors of v_j in the walk. Learning such a posterior distribution can be achieved through different choices of classifiers. For instance, if we decide to use a logistic regression we would end up with a large number ($|V|$) of labels. Having millions or billions of labels would require a lot of computational resources. Several optimization tactics can be used to speed up the training process. The first one is using a Hierarchical Softmax to approximate the probability distribution.
Since $u_k \in V$, it is not attainable to calculate $P(u_k|\varphi(v_j))$ and deriving the partition function, i.e., normalization factor, is high-cost. By allocating nodes of the graph to the leaves of a binary tree, we are now solving the problem of maximizing the likelihood of a particular path in the tree. Let us identify the path to a node u_k by a series of tree nodes $(t_0, t_1, ..., t_{\lceil \log |V| \rceil})$, where t_0 is a root, and $t_{\lceil \log |V| \rceil}$ is u_k. Now, the probability becomes

$$P(u_k|\varphi(v_j)) = \prod_{l=1}^{\lceil \log |V| \rceil} P(t_l|\varphi(v_j))$$

One way to model $P(t_l|\varphi(v_j))$ is by using the binary classifier that is allocated to the node's t_l parent. This way the computational complexity of deriving $P(u_k|\varphi(v_j))$ is decreased

from $O(|V|)$ to $O(log|V|)$.
Another way to speed up the training time is by allocating shorter paths to the frequently occurring nodes in the random walks. One can use Huffman coding to decrease the access time of frequently occurring elements of the tree. The parameter set of the model is φ, T and the size of each one is $O(d|V|)$. In line 4, we use Stochastic gradient descent (SGD) to enhance these parameters. The back-propagation algorithm is used to estimate the derivatives. In terms of the learning rate, α for SGD at the start of the training has a value of 0.025. During the training, it decreases linearly with the number of nodes seen thus far. As we mentioned, the frequency distribution of nodes in random walks of the social graph and words in a language obey a power law. Consequently, we end up with a long tail of rare nodes and accordingly, the updates that concern φ will be sparse. Hence, we can use an asynchronous stochastic gradient descent (ASGD), when dealing with the multi-worker case. Since our updates are sparse one can access the shared parameters of the model ASGD attains an optimal convergence rate.
$DeepWalk$ is highly scalable and can be utilized in large scale deep learning. The parallelizing that we just presented allows us to speed up the processing of the networks as we increase the number of workers. There is also no loss of predictive performance in terms of running the algorithm serially.

5.4 Different Variants of the Algorithm

Here we present two variants of the *DeepWalk* method: *streaming* and *non-random* walks. *Streaming* approach is a variant of *DeepWalk* that can be implemented without knowing about the whole network. Short walks from the network are sent straight to the representation learning code and the algorithm is updated straight away. There are a few changes to be made to the learning process. Firstly, the learning rate α should be set to a small constant value instead of using a decaying learning rate. It will be harder to learn it but could be useful in some applications. Secondly, we are not able to build a tree of parameters now. In the case when the cardinality of a set of vertices is either familiar or can be bounded the Hierarchical Softmax tree can be constructed for that maximum value. The construction goes as follows: When we first see the node we assign it to one of the leftover leaves. If the node frequency can be estimated in advance, the Huffman coding can still be used to reduce the access times of the recurring element.
Some networks are made by a stream of *non-random walks*, for example, users navigating the pages on a website. In that case, we can feed the modeling phase straight away. Sampling the graph in this way allows it to capture the network structure information as well as the information about the frequency at which paths are traveled across. This approach also encloses language models. We can think of sentences as purposed walks through a language graph, and use language models such as *SkipGram* to encapsulate this

behavior. Combining this approach with the *Streaming* variant allows us to train features on an evolving network without having to construct the whole graph. This approach could allow web-scale classification without the complexity of having a web-scale graph.

5.5 Example

In this example, we use a famous $Zachary's\ Karate\ Club$ [12] dataset represented in Figure 6. Using $DeepWalk$ [13] each node is represented as a vector in $\mathbb{R}^d$ space. In our case, we end up with 64-dimensional data. These vectors should mirror the community structure of the graph and can further be utilized by conventional classification algorithms. We reduce the embedding dimensionality using the PCA algorithm [6] to 2-dimensional space. Additionally, we divided the graph using the $KMeans$ [6] algorithm into 3 clusters and we can see the corresponding community structure in Figure 7.

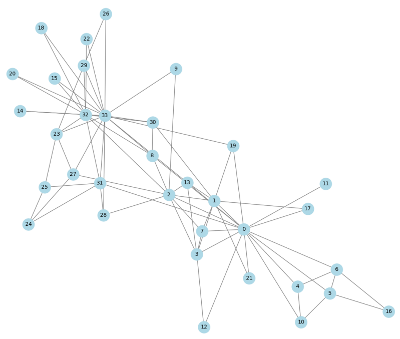

Figure 6. *Zachary's Karate Club graph*

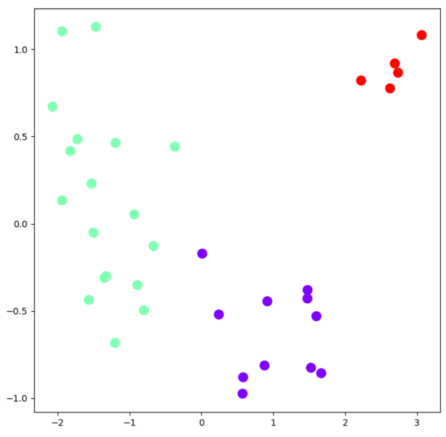

Figure 7. *The Graph* $\mathbb{R}^2$ *Representation*

Conclusion

In this thesis, we presented the topic of selected graph embeddings. An embedding maps every node to a low-dimensional vector of features while trying to preserve the structure properties of the graph and the strength of the links among the nodes. The graph embedding methods can be categorized into Factorization based, Random Walk based and Deep Learning based. We give a short overview of each category with information about time complexity and which proximity measure each method preserves. The focus is set on three approaches - *Locally Linear Embedding (LLE)*, *Laplacian Eigenmaps*, and *DeepWalk*.
Locally Linear Embedding (LLE) tries to discover nonlinear structures in high-dimensional data by using the local symmetries of linear embeddings. It addresses the problem of mapping high-dimensional data into a sole global coordinate system of fewer dimensions. An example that showcases the nonlinear dimensionality reduction of three-dimensional data to a two-dimensional embedding space is provided.
Laplacian Eigenmaps is a geometrically inspired dimensionality reduction algorithm that preserves local properties and has a natural tendency towards clustering. The basis of the algorithm is simple, with a few local calculations and an eigenvalue problem, while its solution considers the geometric shape of the manifold to which the data belongs. This is made possible through the observation that the Laplace matrix of the graph formed over the data points can be viewed as an estimate of the Laplace-Beltrami operator established on the manifold. We provide an example in which we obtain a two-dimensional representation that preserves the locality of a manifold embedded in $\mathbb{R}^3$.
DeepWalk bridges the gap in language modeling and unsupervised feature learning from word sequences to graphs. With *DeepWalk* the successful deep learning techniques from natural language processing are introduced to network analysis. An algorithm *learns* latent features of the nodes that describe neighborhood properties and similarities by representing a stream of short random walks. These features are encoded into a continuous vector space with a small number of dimensions. Once the representation is outputted simple linear classifiers can be used to obtain good results. As an example, we use a *Zachary's Karate Club* dataset and present each node as a 64-dimensional vector. These vectors capture the community structure of the graph and can further be manipulated by conventional classification algorithms.

Appendix Python Code

LLE

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_swiss_roll
from sklearn.manifold import LocallyLinearEmbedding
from sklearn.preprocessing import MinMaxScaler
from mpl_toolkits.mplot3d import Axes3D
from google.colab import files

# Setting the random seed for reproducibility
np.random.seed(42)

# Generating Swiss Roll data
data, color = make_swiss_roll(n_samples=1000, noise=0.3)

# Normalizing the data
scaler = MinMaxScaler()
data_normalized = scaler.fit_transform(data)

# Applying LLE
n_neighbors = 15
lle = LocallyLinearEmbedding(n_components=2, n_neighbors=n_neighbors)
reduced_data = lle.fit_transform(data_normalized)

# Plotting the original Swiss Roll data
fig1 = plt.figure(figsize=(7, 7))
ax1 = fig1.add_subplot(111, projection='3d')
ax1.scatter(data[:, 0], data[:, 1], data[:, 2], c=color, cmap='viridis'
    )

ax1.set_xlabel("X")
ax1.set_ylabel("Y")
ax1.set_zlabel("Z")
ax1.grid(False)
ax1.view_init(10, -70)  # viewing angle

# Saving the original Swiss roll data figure
fig1.savefig('swiss_roll_original.png', dpi=300)
plt.show()

files.download('swiss_roll_original.png')
```

```
# Plotting the LLE reduced data
fig2 = plt.figure(figsize=(7, 6))
ax2 = fig2.add_subplot(111)
ax2.scatter(reduced_data[:, 0], reduced_data[:, 1], c=color,
cmap='viridis'            )

ax2.set_xlabel("X")
ax2.set_ylabel("Y")
ax2.grid(False)

# Saving the LLE reduced data figure
fig2.savefig('swiss_roll_LLE.png', dpi=300)
plt.show()

files.download('swiss_roll_LLE.png')
```

LAPLACIAN EIGENMAPS

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_swiss_roll
from sklearn.manifold import SpectralEmbedding
from sklearn.preprocessing import MinMaxScaler
from mpl_toolkits.mplot3d import Axes3D
from google.colab import files

# Setting the random seed for reproducibility
np.random.seed(42)

# Generating the Swiss Roll data
data, color = make_swiss_roll(n_samples=2000, noise=0.3)

# Normalizing the data
scaler = MinMaxScaler()
data_normalized = scaler.fit_transform(data)

# Applying Laplacian Eigenmaps (Spectral Embedding)
n_neighbors = 15
laplacian_eigenmaps = SpectralEmbedding(n_components=2, n_neighbors=
    n_neighbors)
reduced_data = laplacian_eigenmaps.fit_transform(data_normalized)

# Plotting the original Swiss Roll data
fig1 = plt.figure(figsize=(7, 7))
ax1 = fig1.add_subplot(111, projection='3d')
ax1.scatter(data[:, 0], data[:, 1], data[:, 2], c=color, cmap='plasma')
```

```
ax1.set_xlabel("X")
ax1.set_ylabel("Y")
ax1.set_zlabel("Z")
ax1.grid(False)
ax1.view_init(10, -70)  # the viewing angle

# Saving the original Swiss Roll data figure
fig1.savefig('swiss_roll_original.png', dpi=300)
plt.show()

files.download('swiss_roll_original.png')

# Plotting the Laplacian Eigenmaps reduced data
fig2 = plt.figure(figsize=(7, 6))
ax2 = fig2.add_subplot(111)
ax2.scatter(reduced_data[:, 0], reduced_data[:, 1], c=color, cmap='
    plasma')
ax2.set_xlabel("X")
ax2.set_ylabel("Y")
ax2.grid(False)

# Saving the Laplacian Eigenmaps figure
fig2.savefig('swiss_roll_laplacian_eigenmaps.png', dpi=300)
plt.show()

files.download('swiss_roll_laplacian_eigenmaps.png')
```

DEEPWALK

```
!pip install karateclub

import networkx as nx
from karateclub import DeepWalk
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans

# Reading in the graph
G = nx.karate_club_graph()

# Removing the loops from the graph
G.remove_edges_from(nx.selfloop_edges(G))

# Applying the DeepWalk algorithm
dw = DeepWalk(dimensions=64)
dw.fit(G)
```

```
embedding = dw.get_embedding()

# Dimensionality reduction to 2D using PCA
pca = PCA(n_components=2)
embedding_2d = pca.fit_transform(embedding)

# Clastering with KMeans
kmeans = KMeans(n_clusters=3, n_init=10, random_state=42)
labels = kmeans.fit_predict(embedding_2d)

# Plotting the embedding in 2D
def plot_embedding(embedding_2d, labels):
    plt.figure(figsize=(10, 8))
    scatter = plt.scatter(embedding_2d[:, 0], embedding_2d[:, 1], c=
        labels, cmap=plt.cm.rainbow, s=100)
    plt.colorbar(scatter)
    plt.savefig("embedding_visualization.png")
    plt.show()

# Calling in function for plotting
pos = nx.spring_layout(G, iterations=1000, seed=42)
plot_embedding(embedding_2d, labels)
```

Biography

Jelena Petković was born on the 17th of November in 1998 in Bajina Bašta. She finished elementary school "Sveti Sava" in Bajina Bašta and then moved to Novi Sad to enroll in grammar school "Jovan Jovanovic Zmaj" in Novi Sad. In 2017. Jelena started her education at the Faculty of Sciences, University of Novi Sad. In 2020. she received a Bachelor's degree in Pure Mathematics and enrolled in a Data Science program at the same faculty. Her sphere of interest is Machine Learning and she plans to continue her academic and professional career as a researcher at the Institute for Artificial Intelligence of Serbia in Novi Sad.

Bibliography

[1] Mengjia Xu. *Understanding graph embedding methods and their applications*. 2020. arXiv: 2012.08019 [cs.LG]. URL: https://arxiv.org/abs/2012.08019.

[2] Palash Goyal and Emilio Ferrara. "Graph embedding techniques, applications, and performance: A survey". In: *Knowledge-Based Systems* 151 (July 2018), pp. 78–94. ISSN: 0950-7051. DOI: 10.1016/j.knosys.2018.03.022. URL: http://dx.doi.org/10.1016/j.knosys.2018.03.022.

[3] Sam T. Roweis and Lawrence K. Saul. "Nonlinear Dimensionality Reduction by Locally Linear Embedding". In: *Science* 290.5500 (2000), pp. 2323–2326. DOI: 10.1126/science.290.5500.2323. eprint: http://www.sciencemag.org/cgi/reprint/290/5500/2323.pdf. URL: http://www.sciencemag.org/cgi/content/abstract/290/5500/2323.

[4] Lawrence K. Saul and Sam T. Roweis. "An Introduction to Locally Linear Embedding". In: (Jan. 2001). URL: https://cs.nyu.edu/~roweis/lle/papers/lleintro.pdf.

[5] Brian Keng. "Manifolds: A Gentle Introduction". In: (Aug. 2024). URL: https://bjlkeng.io/posts/manifolds/.

[6] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[7] Mikhail Belkin and Partha Niyogi. "Laplacian eigenmaps and spectral techniques for embedding and clustering". In: *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*. NIPS'01. Vancouver, British Columbia, Canada: MIT Press, 2001, pp. 585–591.

[8] Mikhail Belkin and Partha Niyogi. "Laplacian Eigenmaps for Dimensionality Reduction and Data Representation." In: *Neural Comput.* 15.6 (2003), pp. 1373–1396. URL: http://dblp.uni-trier.de/db/journals/neco/neco15.html#BelkinN03.

[9] Steven Rosenberg. *The Laplacian on a Riemannian manifold: an introduction to analysis on manifolds*. 31. Cambridge University Press, 1997.

[10] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. "DeepWalk: online learning of social representations". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD '14. ACM, Aug. 2014. DOI: `10.1145/2623330.2623732`. URL: `http://dx.doi.org/10.1145/2623330.2623732`.

[11] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: `1301.3781 [cs.CL]`. URL: `https://arxiv.org/abs/1301.3781`.

[12] Wayne W Zachary. "An information flow model for conflict and fission in small groups". In: *Journal of anthropological research* (1977), pp. 452–473.

[13] Benedek Rozemberczki, Oliver Kiss, and Rik Sarkar. *Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs*. 2020. arXiv: `2003.04819 [cs.LG]`. URL: `https://arxiv.org/abs/2003.04819`.

UNIVERZITET U NOVOM SADU
PRIRODNO-MATEMATIČKI FAKULTET
KLJUCNA DOKUMENTACIJSKA INFORMACIJA

Redni broj:
RBR

Identifikacioni broj:
IBR

Tip dokumentacije: Monografska dokumentacija
TD

Tip zapisa: Tekstualni štampani materijal
TZ

Vrsta rada: Master rad
VR

Autor: Jelena Petković
AU

Mentor: dr Dušan Jakovetić
MN

Naslov rada: Izabrane metode ugradnje grafova
NR

Jezik publikacije: Engleski
JP

Jezik izvoda: Engleski
JI

Zemlja publikovanja: Republika Srbija
ZP

Uže geografsko područje: Vojvodina
UGP

Godina: 2024.
GO

Izdavač: Autorski reprint
IZ

Mesto i adresa: Novi Sad, Prirodno-matematički fakultet, Departman za matematiku i informatiku, Trg Dositeja Obradovića 4
MA

Fizički opis rada: 5 poglavlja, 46 strana, 13 lit. citata, 7 slika
FO

Naučna oblast: Matematika
NO

Naučna disciplina: Primenjena matematika
ND

Ključne reči: Grafovi, redukcija dimenzije, ugradnje grafova, mnogostrukosti, obeležja, struktura zajednica, Lokalno linearno ugrađivanje, Laplasovi sopstveni prikazi, Duboka šetnja
KR

Univerzalna decimalna klasifikacija:
UDK

Čuva se: Biblioteka Departmana za matematiku i informatiku, Prirodno-matematičkog fakulteta, u Novom Sadu
ČU

Važna napomena:
VN

Izvod: U ovom radu pružamo pregled izabranih metoda ugradnje grafova, sa posebnim osvrtom na njihove primene u redukciji dimenzije. Motivacija za ovu temu leži u činjenici da mnogi realni problemi koriste podatke u obliku grafova. Iako postoje uspešni algoritmi za analizu mreža, potreba za metodama koje pojednostavljuju strukturu grafova i dalje postoji. Rad prvo uvodi matematičke definicije grafova i metode njihove ugradnje, zatim sistematizuje postojeće metode u tri kategorije. Zatim obrađuje tri algoritma. Prvi algoritam, Lokalno linearno ugrađivanje, koristi lokalne simetrije za redukciju dimenzije i otkrivanje nelinearnih struktura. Laplasovi sopstveni prikazi, drugi algoritam koji obrađujemo, čuvaju lokalna svojstva grafova i imaju prirodnu sklonost ka grupisanju. Treći algoritam je Duboka šetnja, koji primenjuje tehnike dubokog učenja za reprezentaciju grafova putem nasumičnih šetnji. Ove metode su testirane na poznatim skupovima podataka, gde su čvorovi uspešno predstavljeni u vektorskom obliku sa smanjenom dimenzionalnošću.
IZ

Datum prihvatanja teme od strane NN veca: 6.9.2024.
DP

Datum odbrane:
DO

Članovi komisije:
KO

Predsednik: dr Maja Jolić, docent, Prirodno-matematički fakultet, Univerzitet u Novom Sadu

Mentor: dr Dušan Jakovetić, vanredni profesor, Prirodno-matematički fakultet, Univerzitet u Novom Sadu

Član: dr Nataša Krklec Jerinkić, redovni profesor, Prirodno-matematički fakultet, Univerzitet u Novom Sadu

UNIVERSITY OF NOVI SAD
FACULTY OF SCIENCES
KEY WORDS DOCUMENTATION

Accession number:
ANO

Identification number:
INO

Document type: Monograph type
DT

Type of record: Printed text
TR

Contents code: Master's thesis
CC

Author: Jelena Petković
AU

Mentor: Dr Dušan Jakovetić
MN

Title: Selected graph embedding methods
TL

Language of text: English
LT

Language of abstract: English
LA

Country of publication: Republic of Serbia
CP

Locality of publication: Vojvodina
LP

Publication year: 2024.
PY

Publisher: Author's reprint
PU

Publ. place: Novi Sad, Department of Mathematics and Informatics, Faculty of Science and Mathematics, University of Novi Sad, Trg Dositeja Obradovića 4
PP

Physical description: 5 chapters, 46 pages, 13 references, 7 figures
PD

Scientific field: Mathematics
SF

Scientific discipline: Applied mathematics
SD

Key words: Graphs, dimensionality reduction, graph embeddings, manifolds, features, community structure, Locally linear embedding, Laplacian eigenmaps, Deepwalk
KW

Universal decimal classification:
UDC

Holding data: The Library of the Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad
HD

Note:
N

Abstract: In this paper, we provide an overview of selected graph embedding methods, focusing on their applications in dimensionality reduction. The motivation for this topic lies in the fact that many real-world problems use graph-structured data. Although there are successful algorithms for network analysis, the need for methods that simplify graph structures still exists. The paper first introduces the mathematical definitions of graphs and their embedding methods, then classifies existing methods into three categories. It then covers three algorithms. The first algorithm, Locally Linear Embedding, uses local symmetries for dimensionality reduction and the discovery of nonlinear structures. Laplacian Eigenmaps, the second algorithm, preserves the local properties of graphs and has a natural tendency for clustering. The third algorithm is DeepWalk, which applies deep learning techniques to represent graphs through random walks. These methods were tested on well-known datasets, where the nodes were successfully represented in a reduced-dimensional vector form.
AB

Accepted by the Scientific Board on: 6.9.2024.

ASB

Defended on:
DE

Thesis defend board:
DB