1 Methods and Materials

This chapter aims to equip the reader with knowledge of the data and methods later implemented and explain why skeletonization is interesting in a Graph-Based Semi-Supervised Learning setting.

First, the data set is explored, as well means to compare two data points in the data set. Then the purpose and use case of Semi-Supervised Learning (SSL) and the general method for SSL. Then Graph-Based Semi-Supervised Learning (GSSL) is discussed, including the reason for using GSSL, the general method of GSSL, and the specific process of label propagation implemented later in the implementation. Lastly, the skeletonization notion and local- and front-separator algorithms are explained, and the coupling between GSSL and skeletonization is lastly explored.

1.1 MNIST Numbers

The MNIST data set consists of 70.000 handwritten numbers divided into a 60.000 sized training set and a 10.000 sized test set, all fully labeled. The data set was in this thesis chosen as a means to compare the different methods used in **??**, partly for the visual nature of it, and partly because it is a common data set used in Machine Learning, providing plenty of other results to compare the methods to such as [Yan21].
Each number is a fixed size of 28×28 pixels and is encoded in the greyscale color code, where each pixel value ranges from 0 to 255. Figure 1.1 shows an example of each digit in the data set. The numbers are size-normalized and centered.

1.1.1 Convention of conversion

The digits are represented in one of two states:

1. 28×28-matrix: A matrix that translates to the viewed image; the pixel in row 0 and column 0 corresponds to row 0 column 0 in the matrix.

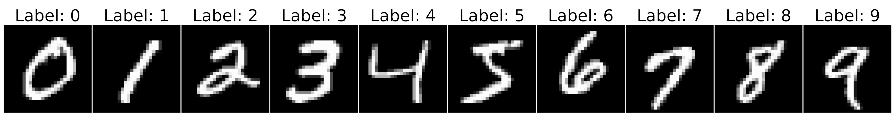


Figure 1.1: A subset of the MNSIT data set with an example of each label

2. 764-vector: A vector, the flattened matrix, where each 29'th index corresponds to an increment in the row and a reset in the column to index 0.

To not express every time a digit changes shape between vector and matrix, one can assume that it is in vector shape unless it is visualized.

1.1.2 Measure of similarity

As becomes apparent in the following chapters, finding a good measure of similarity between two digits is crucial. The requirements are a low computation time and a reasonable estimate of similarity where digits of the same label have a low value and digits of different labels have a high value. The Euclidean distance shown in Equation (1.1) provides both of the requirements but had a hurdle in that the comparison is pixel-to-pixel based.

$$\|u-v\|_2 = \left(\sum \left(w_i \left|(u_i - v_i)\right|^2\right)\right)^{1/2} \tag{1.1}$$

This means that if the shape of two digits is the same, but they are placed differently, the equation would rate them as being farther apart than is true, as is displayed in Figure 1.2, where the shape of the two images are the same, but they have been placed one square apart. What was not immediately obvious was the fact that the MNIST numbers are both "size-normalized and centered in a fixed-size image"[Yan21], so any positional problems were naught, and it can be assumed that the Euclidean distance gives a reasonable estimate of similarity between the shapes of digits.

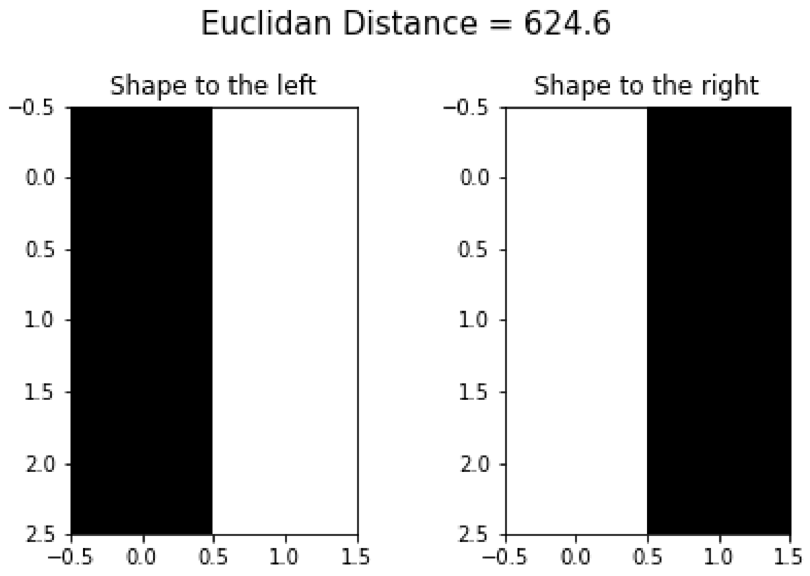


Figure 1.2: Same shape in different locations

1.2 Graph-based Semi-Supervised Learning

This section will first introduce the reader to semi-supervised learning, a cornerstone in graph-based semi-supervised learning. After that, graph-based semi-supervised learning is introduced, its taxonomy, and the particular method of constructing a graph with KNN and label inferring with Label Propagation.

Graph construction and label propagation lean on the works of Zixing Song, Xiangli Yang, Zenglin Xu in [Son+21] providing a taxonomy for defining a graph-based semi-supervised learning problem as well as a general survey of the different gssl algorithms, and the works of Xiaojin Zhu and Zoubin Ghahramani in [ZG02] clearly defining the label propagation algorithm and implementation.

1.2.1 Semi-supervised learning introduction

Semi-supervised learning is an answer to a gap in data science. One would usually work with a feature-label pair in supervised learning or a fully unlabelled data set in unsupervised learning. Semi-supervised learning works with the spectrum in between with a data set partly labeled, often sparsely. Semi-supervised learning can be divided into two types: transductive and inductive learning.
Transductive learning aims to predict labels within a given data set and not any unknown data points. Informally it can be described as reconstructing the data set. Formally transductive learning is defined as: Given a training set consisting of labelled and unlabelled data $D = \{\{x_i, y_i\}_{i=1}^{n_i}, x_{i}{}_{i=1}^{n_u}\}$ the goal of a transductive learning algorithm is to learn a function $f : X \rightarrow Y$ so that f is only able to predict the labels for the unlabelled data $\{x_i\}_{i=1}^{n_u}$[Son+21].

1.2.2 Overview

Graph-based semi-supervised learning works on the same principles and assumptions as semi-supervised learning. The domain is still sparsely labeled data sets, but the setting is different. Where graph-based semi-supervised learning differs is in the structure of the algorithms. GSSL uses the graph structure to represent data point and their similarity. Each node in a graph represents a data point in the data set, and each edge in the graph represents that two nodes are similar to each other.

Based on the taxonomy proposed in [Son+21] the process of constructing a gssl-problem involves two steps: 1) graph construction, 2) label inferring on the graph. There are multiple ways to achieve both steps. Because this thesis is rather broad, a decision was made early on to find algorithms for both steps without too much complexity. Constructing the graph with the k-nearest-neighbors algorithm and inferring the labels with label propagation met these criteria.

Definition of graph

Given a data set D, the graph G consists of a set of nodes V where each node represents a data point in the data set such that $|V| = |D|$, a set of edges E such that $E \in V \times V$ and a weight matrix $W \in R^{|V| \times |V|}$ to represent the weight on the edges if the graph is weighted. If the graph is not weighted the weight matrix will represent edges such that $W_{i,j} = 0$ if i, j does not share en edge and $W_{i,j} = 1$ if i, j does share an edge. This is akin to an adjacency matrix if it is unweighted. The graph is then formally described as $G = (V, E, W)$.

1.2.3 Graph construction

To perform graph-based semi-supervised learning, a graph is necessary. The nodes in the graph represent data points in the data set, and edges between nodes represent a similarity between the two nodes. Edges might also be weighted to reflect the similarity between two nodes. Edges in the graph can either be directed or undirected, which does not limit the construction of the graphs. In some cases, a graph network might already exist, such as a social network, and might not need the graph will practically already be constructed. If that is not the case, a similarity function is used to identify similar nodes and connect them, such as the Euclidean distance.

KNN-based graph construction

The premise of KNN-based graph construction is to choose a number of neighbors K and construct a graph where each node is connected to the K closest nodes based on a similarity function $sim(x_i, x_j)$, that can quantify similarity between node pairs. These values will then be added to the weight matrix such that

$$W_{i,j} = \begin{cases} sim(x_i, x_j) & i \in neighbors(j) \\ 0 & otherwise \end{cases}$$

In the case of an unweighted graph, the $sim(x_i, x_j)$ can be exchanged with 1, which is the case of this thesis in the implementation of KNN.

The $neighbors(j)$-function is the cornerstone of the KNN algorithm. The function requires to choose the K lowest values of similarity between given digits x_i and all other digits $x_j \mid j \neq i$, which also translates to a graph with no loops. The no loop rule is essential as using the Euclidean distance on the same digit always yields 0, and if loops were allowed, every node would have a loop, as 0 will always be the lowest value.

The runtime of KNN is $O(n^2)$ as we have to check each index in the weight matrix, and the space complexity is $O(n^2)$ as the row and column have to exist for every data point in the data set.

Visualizing KNN-based graph construction

To aid the reader, a small example is displayed in Figure 1.3. Unlike the MNIST numbers, the data set in Figure 1.3 is located in 2-dimensional space; the same rules, however, exist for MNIST. The Euclidean distance is used to measure similarity, and the only difference is that it is impossible to visualize all 764 dimensions of an MNIST data point in a single graph.

1.2.4 Label inferring with label propagation

Label inferring is the process of spreading known labels throughout the graph. Using label propagation to infer requires three steps:

1. Spread the known labels one time-step throughout the graph

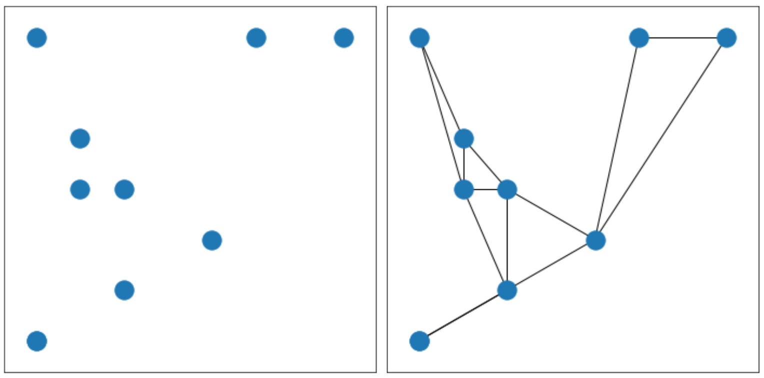


Figure 1.3: Example of KNN graph construction in 2 dimensions. Left is data points, right is graph constructed with 2 neighbors

2. Normalise the populated unlabelled nodes
3. Clamp known labels and repeat until the algorithm converges

This process can be done through matrix multiplication alone, as this thesis does, by multiplying the weight matrix with a one-hot encoded matrix of labels. It will later show that this approach can vastly improve performance utilizing sparsity matrices.

Example of label propagation

To get an intuition for the procedure of label propagation, a small example has been created shown in Equation (1.2) and visualized in Figure 1.4. Initially, in the problem, we a weight matrix W representing the graph, and we have some known and unknown labels in a list of labels, where each index corresponds to a specific node in V.

$$W = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix} \quad L = \begin{bmatrix} -1 \\ 0 \\ 1 \\ 1 \\ -1 \end{bmatrix} \tag{1.2}$$

By transforming the label matrix such that it has as many columns as unique labels in the data set and as many rows as data points in the data set, $L_{i,0} = 1$ indicates that a given data point with index i has a label 0, and an empty row indicates that a given index does not have a label at all. This is also known as one-hot encoding. The transformed label matrix named OL (One-hot encoded Labels) can be seen in Equation (1.3)

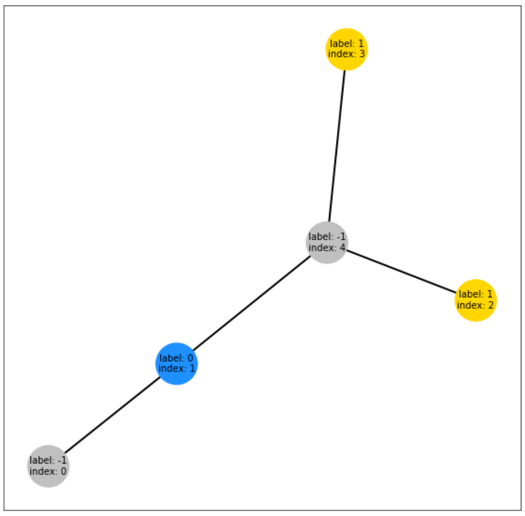


Figure 1.4: Visual representation of weight and label matrix in Equation (1.2)

$$L \xrightarrow{\texttt{one-hot encode}} OL = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \tag{1.3}$$

Multiplying the weight matrix and transformed label matrix, the result will effectively be the same as counting the number of labels the neighbors of each node have, as can be seen in Equation (1.4). It will perform the first step of label propagation in spreading the labels one-time step. Notice how all nodes with labeled nodes - specifically node 0 and node 4 - have counted each label's type and occurrence in their labeled neighbors.

$$W \cdot OL = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 2 \end{bmatrix} \tag{1.4}$$

Step 2 of label propagation is row normalizing the label matrix. These normalized values can be interpreted as the probability that a given unknown node i has label x. The normalized rows can be seen in Equation (1.5)

$$W \cdot OL \xrightarrow{\texttt{row-normalise}} \begin{bmatrix} 1.00 & 0 \\ 0.00 & 0.00 \\ 0.00 & 0.00 \\ 0.00 & 0.00 \\ 0.33 & 0.67 \end{bmatrix} \tag{1.5}$$

This completes the second step of label propagation. Before checking for convergence, the already known labels are clamped back to the known labels, as can be seen in Equation (1.6). Intuitively from this matrix, it can now be seen that the algorithm converges and node $L_0 = 0$ and $L_4 = 1$.

$$\begin{bmatrix} 1.00 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0.33 & 0.67 \end{bmatrix} \tag{1.6}$$

A small note on converging
Unfortunately, later in this project, the paper [ZG02] was discovered, showing that the label propagation algorithm converges to a simple solution not requiring this iterative process. As the run time of the final implementation of the algorithm was not very significant, this simpler solution was not implemented.

1.3 Skeletonization

This section will introduce the reader to the concept of skeletonization and introduce the two algorithms: front-separator and local-separator devised and implemented by Andreas Bærentzen, and Eva Rotenberg in [BR20]. The section will briefly touch on the process of packing and extracting the skeleton. Lastly, it will reflect on skeletonization in a graph-based semi-supervised learning setting.

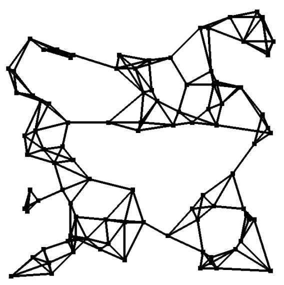
1.3.1 Overview

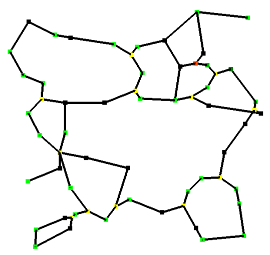
Skeletonization is the process of simplifying graphs with a focus on highlighting features and reducing noise. The premise of skeletonization is to take the nodes from the original graph and shrink them until only the overall shape of the graph is back. One can imagine using it on a 3D model of a human and only having the path of the skeleton left when the algorithm is done.

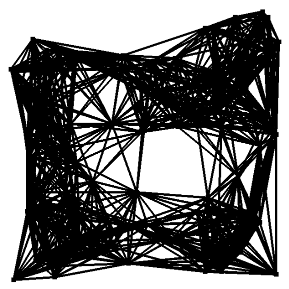
The algorithms work on the premise of separators. A separator is defined as a subset of vertices V where starting from one side of the separator and ending on the other is not possible without passing through. This has some interesting properties.

Suppose a graph is cyclical, and passing through all nodes can only be done by visiting each node. In this case, each node in the cycle is effectively a separator, and using either of the algorithms would yield the same output graph as the input graph. An opposite example of this is a graph G with nodes spread around, all connected to a single node in the middle. The valency of each node around the center node would be 1, and the center node would have a valency of $|V| - 1$, as it is connected to each node except itself. This center node would effectively be a single separator connecting all nodes, as one can not pass from an outside node to another outside node without passing through the center node. Applying either of the skeletonization algorithms would yield a graph with a single node placed where the center node is located. The last example to give a fuller picture is a graph where all nodes are connected. Here all nodes effectively create a separator together, and the final graph would again be a single node in the skeleton.

This can be extrapolated and shown that an increase in valency in the graph computes a simpler skeleton. This is shown in Figure 1.5, where we see that the graph with fewer neighbors provides a more complicated skeleton and vice versa.


(a) Graph with $k = 4$


(b) Skeleton from graph with $k = 4$


(c) Graph with $k = 20$

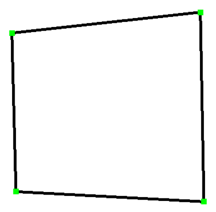

(d) Skeleton from graph with $k = 20$

Figure 1.5: Example of correlation between valency in graph and magnitude of simplification in skeleton. Dataset is a 100 random datapoints, and edges are created by finding k closest nodes for each node

1.3.2 Local separator algorithm

The informal approach of the local separator algorithm is as follows. From the input graph, choose a selection of random nodes. From each of these random nodes, now look at all neighbors and add them to a queue of new nodes to explore as a front (F).

From F, we now choose the closest neighbor using a bounding sphere[1]. At some point, the separator will have grown so much that it separates the graph, meaning there is no path from one side of F to the other without passing through the separator. This is shown in Figure 1.6 G. Lastly, the separator is shrunk, as can be seen in Figure 1.6 H. The shrinking works on a few principles. First, the separator is smoothed using Laplacian smoothing. Then the set of nodes in the separator is removed regarding decreasing distance (the furthest away from the center of the separator is removed first). If the separator breaks into two components, that is, two subgraphs of G not connected by an edge, in this stage, the separator is discarded. If not, the separator can now be added to a set of separators and be further handled in the packing stage of the algorithm.

A significant property of the local separator algorithm is the reliance on the bounding sphere, specifically because the implementation only allows for nodes embedded in 2 or 3 dimensions. This is a constraint as the MNIST numbers only have an embedding in their original 764 dimensional space. This will be addressed further in **??**

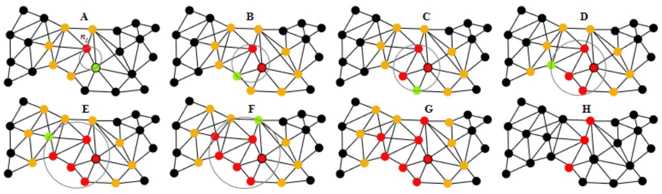


Figure 1.6: Picture from [**DBLP:journals/corr/abs-2007-03483**] showing the growing of a local separator

1.3.3 Front separator algorithm

The front separator algorithm also works with a front F. It is, however, defined a little differently. Instead of choosing the front based on a bounding sphere from random nodes, the front is devised as a breadth-first search. Informally the algorithm works by choosing a selection of random nodes from the graph. From these nodes, a distance is calculated to all other nodes. This could be the Euclidean distance as is in this thesis. A breadth-first search algorithm is used to find the shortest path from the seed node to all other nodes from each of these nodes. This is saved for later reference.

From here, the algorithm chooses random starting nodes within the graph. For each node in the graph in a random order, a node is hypothesized to start a new separator. If the node has already been explored and added to a separator, the attempt is terminated, and a new node is chosen. If not, the node finds all the closest neighbors and adds them to the front. For each node in the front, the algorithm checks if the neighbor has been explored. If another separator has explored it, the attempt is terminated, and a new node is chosen. Suppose the neighbor has not been explored, and the path from the current

[1]A bounding sphere works by finding the center of "mass" of the nodes already explored and uses this center to calculate the distance to neighbors. This is done to find the closest point to the collective of the explored nodes

node to the neighbor is the optimal route. In that case, the neighbor is added to the separator, the neighbors are added to the front, and the node is indicated to have been visited by the separator. This continues until the front is empty and adds a separator to the set of separators. If the separator at any time finds one of the reasons mentioned above to terminate, the indicated nodes visited by the separator are reset as if not visited yet.

The clear advantage of using the front-separator algorithm is that, unlike the local-separator algorithm, it does not need embedded positions as it does not rely on the bounding sphere but instead only needs a pre-calculated distance from one node to all other nodes.

1.3.4 Packing

With both separator algorithms, it is possible to get a node placed in multiple separators, as separators can overlap. Packing ensures that each vertex on the graph only is covered by a single separator. Vertices without a separator might also exist. These are assigned to the closest separator.

1.3.5 Skeleton extraction

From packing to extracting the skeleton includes two steps. First, the vertices of the skeleton are calculated. This is done by calculating the average position of each separator. However, this step might lead to vertices in the graph with a valency $V \geq 3$, which is undesirable. The final step consists of removing these vertices by merging them with neighboring vertices of valency 2.

1.3.6 Motivation for skeletonization in a graph-based semi-supervised learning setting

The initial reasoning for exploring skeletonization coupled with graph-based semi-supervised learning was its formerly described ability to shrink graphs and keep the features present. If the graph is constructed such that edges connect similar nodes and dissimilar nodes are not connected at all, compressing the graph could lead to a skeleton expressing the core structure of the data. This could be a valuable asset when label inferring on very sparsely labeled data, as the skeletonization process would guide the label inferring algorithm.