1 Implementation

The pipeline for the implementation of GSSL in this thesis is as follows:

1. Calculate distance matrix
2. Calculate weight matrix

   This doubles as effectively the graph
3. Propagate labels
4. Use model to test on the test dataset

In addition, small tweaks will be made to the framework to test the hypothesis of some improvement using skeletonization. The tweaks are as follows:

1. Skeletonize before propagating labels, then propagate through the skeleton, then test
2. Skeletonize after propagating labels, then test

1.1 Calculating distance matrix

To calculate the distance matrix, the L2-norm is used. The formula for the matrix is:

$$\sqrt{(x_1 - y_1)^2 \ + \ (x_2 - y_2)^2 \ + \ ... \ + \ (x_N - y_N)^2}$$

Where X_n corresponds to a pixel value in a minst number, this will estimate the similarity between two images, and the distance matrix can be thought of as a ”similarity matrix.” Because the dataset is standardized and centered, we can assume that numbers will not be placed in weird corners or positions that make similar numbers get a high value, as they do not fill the same space, even though they have the same shape. As the training set will be a size of 60.000×764, as there are 60.000 images each with 764 pixels, the size of the matrix will be 60.000^2. To counteract the size of this and make it reasonable to perform on local hardware, a sparsity matrix is used, with a given threshold to remove outliers but also edges between two points with too much distance. In addition, as the only operation is updating the sparsity matrix, concurrent operations can be used within a given range of indices without the problem of overwriting entries.

1.2 Construct graph

To construct the graph, a KNN-based method will be used. The K nearest neighbors will be chosen for each data point, and an edge is constructed between them. Because the chosen method for inferring labels (label propagation) does not consider the distance between points, the resulting graph will be weightless. It will have the size N^2, and a

1 on each index where a neighbor is present and a 0 otherwise. The indices become important in label propagation.

1.3 Propagate labels

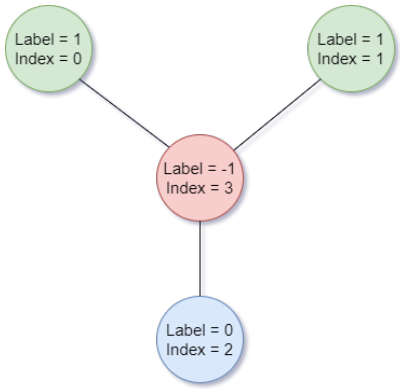
The general approach in propagating labels can be described as three steps:

Step 1: Propagate all labels one step, that is, calculate the probability of labels in unlabelled nodes for one iteration.

Step 2: Row normalize the result for labels

Step 3: Reinsert the already known labeled data and repeat until the labels converge.

To implement this method, matrix multiplication is used. The general approach in this regard is that we can model the edges in a graph in one N^2 matrix, where N is the number of data points and the labels in a N^L_{num} where L_{num} is the number of unique labels. In the case of MNIST, this number is 10 ($\{0, 1, ..., 9\}$. A simple illustration of this representation can be seen in Figure 1.1.



(a) Visual representation of graph

$$N_{\texttt{Neighbours}} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

$$L_{\texttt{Labels}} = \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 0 \end{bmatrix}$$

(b) Computational representation of graph

Figure 1.1: Representations of graph. For simplicity only the labels ”0” and ”1” are included.

1.3.1 Step 1: Propagate labels 1 step

To do this the neighbours matrix is multiplied with the label matrix e.g. $N \cdot L$. The resulting matrix will have the same shape as the labels, and will effectively be a count of every label a given nodes neighbour has. Calculating the example in fig. 1.1b yields:

$$N \cdot L = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 2 \end{bmatrix} \tag{1.1}$$

Checking the graphical representation, we can see that only the node with index 3 has neighbors with labels and that as calculated in eq. (1.1) it has one neighbor with label 0 and two with label 1.

1.3.2 Row normalising

The next step is to calculate the probability of each label. It makes sense that if more neighboring nodes have label 1 than label 0, the probability that the neighbored node is of label 1 is higher. To do this, each row is normalised according to the total of counted labels.

1.3.3 Clamping

The final step is to clamp the values. This is because we form the training data has a ground truth in the already known labels, and we do not want to change the value of these. To counter this, the true labels are reinserted into the matrix, but the probabilities seen before are kept.

1.3.4 On converge

When the algorithm converges, the algorithm is finished. The algorithm converges when there is either no change at all or the change in probabilities is so minimal that effectively there is no change in labels. When this is done, all unlabelled nodes are assigned the label with the highest probability.

1.4 Test function / graph-model