DTU Compute
Department of Applied Mathematics and Computer Science

Datanalysis using skeletons derived from graphs

Bachelor Thesis

Abstract

Hello

Approval

This thesis has been prepared over six months at the Section for Indoor Climate, Department of Civil Engineering, at the Technical University of Denmark, DTU, in partial fulfilment for the degree Master of Science in Engineering, MSc Eng.

It is assumed that the reader has a basic knowledge in the areas of statistics.

Gustav Lang Moesmand - s174169

.....

Signature

.....

Date

Acknowledgements

Aasa Feragen, Lektor, Institut for Matematik og Computer Science
[text]

Jakob Andreas Bærentzen, Professor, Institut for Matematik og Computer Science
[text]

Eva Rotenberg, Lektor, Professor, Institut for Matematik og Computer Science
[text]

List of Symbols

c	Number of labels
l_i	i-th label
md	Maximum distance between neighbours
N	Size of dataset
N_{test}	Size of dataset for testing
N_{train}	Size of training dataset
nn	Number of Neighbours
x_i	i-th data point
MDE	Multi Dimensional Embedding
MDS	Multi Dimensional Scaling
SM	Skeleton Model

Contents

1 Introduction

Data analysis has been on the rise for the last 50 years. Its function is crucial in many places, from where to place certain items in a grocery store, managing the flight plans in airports, or predicting the weather. These problems could at some point in time be solved to a satisfying degree by humans. However, as the complexity and immensity of the data rose, computers became necessary to traverse these enormous amounts of data.

With this ever-increasing size of data, maintaining its quality has become more complex. The sheer size alone is not feasible to make a human decipher or is too expensive. This means that some data could be a lot more usable, for instance, data analysis or model training, if targets for the observations were available.

Spring i tekst

A solution to this problem was introduced with semi-supervised learning. The main idea behind semi-supervised learning is observation-result pairs in a dataset that is not covered and makes it predict new labels for unlabelled data.

One of the methods within semi-supervised learning works by constructing graphs. One upside of graphs is that whether a connection between two nodes is to make has nothing to do with whether the data is labeled or not. This makes it ideal, as this structure of edges can propagate labels from labeled nodes to unlabelled neighbors under the assumption that the two nodes are neighbors because they have enough observations in common.

Spring

In an article by Andreas Bærentzen, Eva Rotenberg called ”Skeletonization via Local Separators” [BR20]. The premise of these skeletons is to highlight features of the input graph, and among other things, remove noise and unnecessary complexity.

Vi vil undersøge,...

This feature set of skeletonization may be ideal for boosting the accuracy of graph-based supervised and semi-supervised learning, as the skeleton is optimized for highlighting features. The MNIST database will be used to test this hypothesis.

–MORE–

This ultimately leads to the following research questions:

1. Does skeletonizing graphs lead to better accuracy?
2. How does a skeletonized graph compare to more conventional classification methods such as KNN in supervised learning.
3. How does a skeletonized graph compare to KNN in semi-supervised learning?

2 Foundations

This chapter will provide the reader with some basic knowledge on the different foundations on which the later methods are based. It will give a brief overview of the different methods within machine learning and expand on the methods used for this project. It will also provide some insight into graph theory and linear algebra that will help understand especially skeletonization. Lastly, it will introduce Multi-Dimensional Embedding used to embed the MNIST images later in the project.

2.1 Machine Learning

Information in this section is based on [TMll]. As machine learning is a broad subject, this section has been narrowed down to the specific parts used in this thesis. This includes supervised learning, semi-supervised learning, and hyperparameters.

- basic ML antages kendt?

General notion of machine learning

In general, machine learning works by giving a machine some data to be processed and turned into a model. The purpose of the model is to make a prediction given new data in the same format as the last. Often machine learning is compared to the process of us humans think. When hearing that it is more of a prediction than an absolute, one might therefore be confused, as humans knowing a given picture is of a number and a computer guessing that it is a given number seems fundamentally different.

The result of using machine learning is a model that, given some relevant data, will try to predict an answer. An intuitive example would be to give an algorithm the weather data from the last 50 years and ask it to predict the weather tomorrow.

2.1.1 Types of learning

Generally, Machine Learning can be divided into three different categories of learning: supervised, unsupervised, and reinforcement learning. As reinforcement learning is not used in this thesis, it will not be further mentioned, somewhere between supervised and Between supervised and unsupervised lies a fourth type of learning: semi-supervised learning.

2.1.2 Supervised Learning

Supervised learning consists of training a model based on a dataset of N observations $[x_1, ..., x_N]$ and N targets $[y_1, ..., y_N]$, to predict y given x. This is formulated in [TMll] as

$$y = f(x, w) + \epsilon$$

Where w is a set of tunable parameters and ϵ is a noise term. The task then consists of tuning the parameters w based on the training data. The model used in this thesis is based on classifying a label of an image which is called a classifying model.

2.1.3 Semi-Supervised Learning

Semi-Supervised learning is an answer to a problem where a dataset has lots of observations but very few labels. Introducing more labels to the dataset would be expensive as the task often requires a human agent to decipher the observations. It generally works by

2.1.4 Hyper parameters

In machine learning, there is a clear distinction between parameters and hyperparameters. Parameters are what the machine is training when creating the model, such as the slope in regression. On the other hand, hyperparameters are predetermined tunable parameters chosen before training the model, such as choosing k in KNN. Often hyperparameters are

determined differently given the job. A common approach to determining hyperparameter is to split the training set into a test and validation set x_{test}, x_{train}. Through this process, the optimal parameters can be estimated, and the assumption is that the parameters giving the best result in the optimization phase are prone to also give the best result in the actual training of the model.

2.1.5 Classification

Classification in machine learning works on the premise of working with a categorized set of data; this could be

2.2 MDS - Multi-Dimensional Scaling

The premise of MDS analysis is to find a spatial configuration of data points, with the only known relation between them being their general similarity or dissimilarity. There are lots of different ways to calculate the similarity between data, one of which is the euclidean distance, where the general formula can see in Equation (2.3)

$$\texttt{Given a point p and a point q} \tag{2.1}$$

$$\texttt{2 dimensional} : d(p,q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2} \tag{2.2}$$

$$\texttt{n dimensional} : d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} \tag{2.3}$$

Once the distances have been calculated, the data collection is concluded, and by using Eigen Decomposition positions in a graph can be calculated.

Within MDS, it common to make a distinction between classical (metric) MDS and nonmetric MDS. Classical MDS assumes that the data display metric properties like

Jeg forstår ikke det her afsnit. - Eva

3 Methods and Materials

This chapter aims to provide some insight into the data used in this thesis and the methods for modeling and predicting it.

3.1 MNIST

The MNIST database (Modified National Institute of Standards and Technology) is a collection of handwritten digits. The database consists of a training set of 60.000 images and a test set of 10.000 images. The images have been normalized and shaped into 28×28 pixels and are in grayscale ranging in value from $0 \rightarrow 255$. –MORE–
A subset of the database can be seen in Figure 3.1

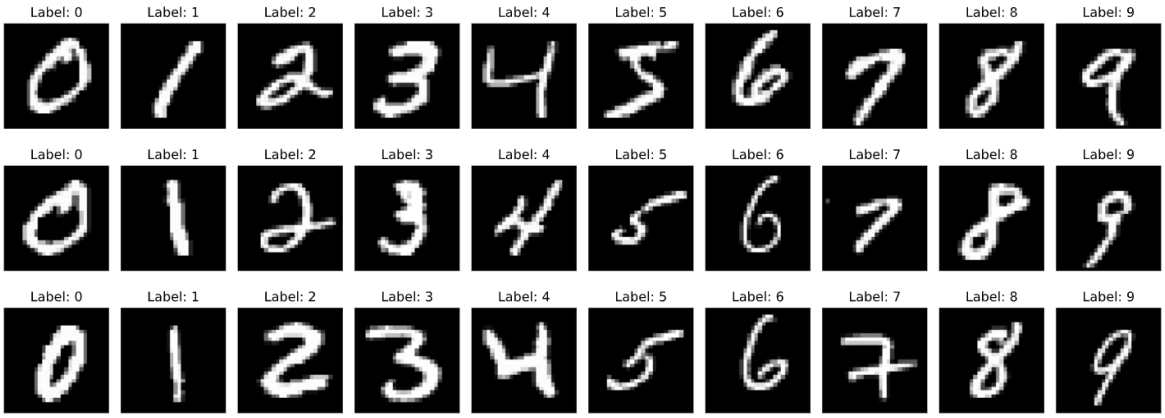


Figure 3.1: Subset of MNIST images

3.1.1 Pixel embedding of MNIST

When looking at images of the MNIST numbers, it can be pretty hard to imagine any embedding of the numbers.

The MNIST images have no apparent embedding; that is, a picture of a number is seen, but to place that number in some coordinate system might mean trouble. To then propose that these images in their inherent values actually have some numerical distance between each other might be confusing, but it is actually true. Impossible to really imagine, but by creating a coordinate system, where each cell of the image is a new axis, the images can actually be plotted in a $28 \cdot 28 = 784$ dimensional space. This, of course, is impossible to imagine, so to ease the understanding, follow this small example. –Det her skal skrive om–

Given these same conditions as the MNIST images, where pixel values are grayscaled, we can produce images that are plotable in observable space. As seen in Figure 3.2

3.1.2 Distance between MNIST

3.2 Graphs

- antages kendt?

3.2.1 Valency

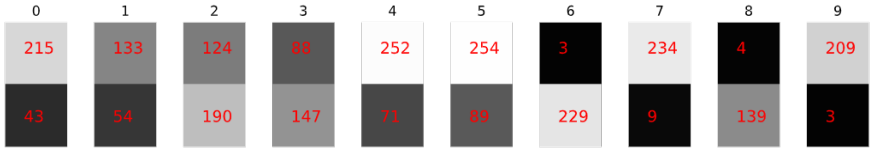
?

3.3 Skeletonization

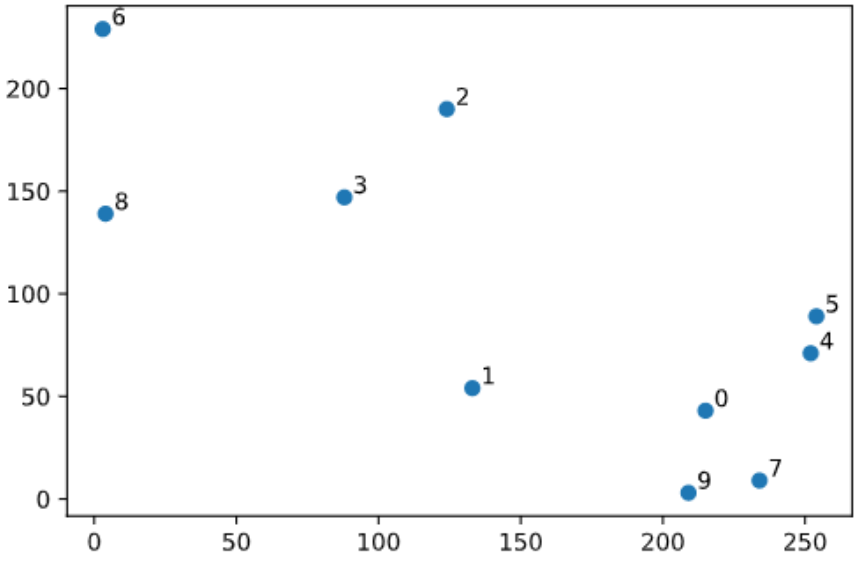
Et skelet er ...

via

This section will talk in general about skeletons and about the implementation of local separators developed in [BR20], as the algorithm developed in that article is what is used and experimented within this thesis, and as some of the attributes to that solution contributes to how the algorithm developed in this thesis was developed.



(a) Images with two pixels and values ranging from 0 to 255



(b)

Figure 3.2

3.3.1 Skeletons

Skeletonization is the process of simplifying geometry with a focus on highlighting features and reducing noise. The premise of skeletonizing 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.

3.3.2 Brief overview of algorithm

The general approach of the local separator algorithm is as follows. From the input graph, choose a bunch of random nodes. From each of these random nodes, now look at all connected nodes (neighbors) and add them to a queue of new nodes to explored called the front (F). From F, 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 way path from one side to the other of the graph without passing through the separator. This is shown in G in Figure 3.3

3.3.3 Feature of local separator skeletonization

Because the local separators algorithm needs to split a graph G g into two separate components, that is, there is no way to get from the first component to the second component without passing through the separator, the overall valency of the graph impacts the reduction done by skeletonization, as the number of nodes needed to separate a graph increases, when there is an increased opportunity to find a path without going through the separator, because of a higher number of edges. This can also be seen in Figure 3.4 where it is appar-

[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

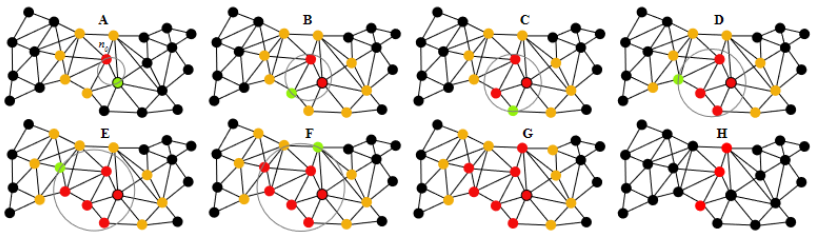
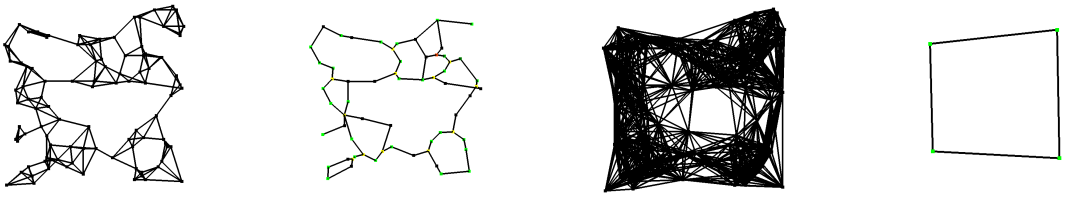


Figure 3.3: Picture from [BR20] showing the growing of a local separator

ent that the higher number of neighbors results in a much more simplified skeleton. This is important because the primary purpose of this paper is to experiment with the effect of accuracy when skeletonizing a graph model, and thus we can draw a clear line and say, number of neighbors is directly related to the amount of change skeletonization brings and for the hypothesis of skeletonization improving precision to be accurate, precision should rise with the number of neighbors in the input graph.



(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 3.4: 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

4 Algorithms

pro tip: man kan forklare algoritmerne først, med ord, og så bag

This chapter will provide the reader with an insight into the implementation of producing a model skeletonized from a graph to predict labels of MNIST images. The acronym of the model SM (Skeleton Model) will be used to refer to it. It will also shortly describe how KNN is implemented to give the reader some insight into the workings of the compared model.

4.1 A comment about embeddings

Part of the original scope of this project was to extend the library GEL and make the graphs capable of having n-dimensional nodes; this, however, turned out to be too big a task and was abandoned. This limited the project to only create the graph structure needed in 3 dimensions, and as there is no precomputed embedding for the MNIST numbers, an alternative was found in MDS discussed in section 2.2. Due to the nature of scaling dimensions, some loss of information is likely to occur. This is discussed in further detail in [Wic03] about the goodness of fit. To gain as much information as possible, the model uses two different embeddings: 3-dimensional embedding and the original 784-dimensional embedding.

The role of 3 dimensional embedded images

The embedded images are used in the skeletonization phase (section 4.2.4). The skeletonization algorithm uses a bounding sphere to expand and stops when the separator separates two sub-graphs of g with no common edges. This bounding sphere based on 3D coordinates is a downside of this implementation, as information is lost in the multidimensional downscaling. In a more optimized algorithm, this whole graph would be in the original dimensional space of the images, which might lead to worse results that cannot be tested in this thesis. An upside to this, however, is that it is computationally easier to calculate the bounding sphere for 3-dimensional coordinates **??**, so the algorithm might yield a faster runtime skeletonizing.

The role of 784 embedded images

The non-embedded images are used to calculate the distance between points which are the values the edges between vertices in the graph are based on. As mentioned in 4.1 information is lost when downscaling spaces, so using the original space of 784 dimensions should yield better results.

4.2 SM - Skeleton Model

The algorithm developed in this thesis takes a custom amount of flattened MNIST images with the shape $[784]$ as input and produces a Skeleton Model as output. The SM can then be used to predict the labels of new MNIST images. The general approach is to first create a graph with all training images as nodes and no edges. The algorithm first needs the distance from all nodes to all other nodes to connect a pair of nodes with an edge. A distance matrix is created that stores all distances between all nodes. Next, based on these distances, the graph can be connected. Here the number of neighbors and the maximum distance for a node to be considered a candidate as a neighbor can be set. With this connected graph, skeletonization is possible. As discussed in Section 3.3 when creating separators, multiple nodes are combined into one, and new nodes are not based on any nodes from the skeleton, which gives nodes without labels and nodes with more than one label. Each node needs to have precisely one label. This is solved in the relabeling and propagation phase.

The pipeline developed can be seen in Algorithm 4.1. The sections below are made to explain the different parts of the algorithm, and the corresponding section can be seen as refereed as a comment in the algorithm.

```
input: float forget_labels, float[][] mnist_images
output: GraphAnalysis

begin
    g := createGraph() % See section 4.2.1
    d := createDistMatrix() % See section 4.2.2
    connectGraph(g, d) % See section 4.2.3

    s := skeletonizeGraph(g) % See section 4.2.4
    propogateLabels(s) % See section 4.2.5

    return s
end
```

Algorithm 4.1: Outline of Graph Analysis Algorithm

Please note that illustrations work on the principles of the algorithm, but positions and values are not based on any actual data.

4.2.1 createGraph() - Creation of the graph

The Graph Analysis model is created by first embedding the MNIST numbers into three dimensions using the framework PyMDE developed in the article Minimum-Distortion Embedding [AAB21]. By using this method, the images can be embedded in a 3D space applicable for the implementation of Graphs and Skeletonization in GEL. The result of embedding all 70.000 coordinates can be seen in figure **??**.

formattering

Embedding MNIST using MDE

Not only does this allow for the use of pygel, which will enable skeletonization using local separators, but it also helps visualize the data .

An important point about this transformation is that only the training images are used to embed; that is, if only 10.000 images are used in the graph, the embedding is also only based on 10.000 images. This is to avoid an unfair advantage and to avoid potential data snooping[1].

To help visualize the current state of the data, look at figure Figure 4.1

(a) Graph state in supervised case, where all labels are known

(b) Graph state in semi-supervised case, some amount of labels are unknown

Figure 4.1: Illustration of state of graph after running createGraph(). Position of nodes and label of each node is initialised.

In the graph of SM, the nodes are placed. For each node, the label of that node is tied, as well as its pixel values. Labels are used later in propagation, and pixel-values are used to calculate distances between nodes in the distance matrix phase.

4.2.2 createDistMatrix() - The Distance Matrix

The distance matrix stores the distance between all images and given n images have a size of n^2. The distances are calculated using the heuristic discussed in Section 3.1.2. The

[1] In this context, because the skeletonization algorithm uses a bounding sphere to find vertices close to each other, using more images to create the embedding would yield a better embedding, and thus might create better results, due to the positions of vertices in the graph being embedded relatively to also the training points.

function of the dist matrix is to give context to the relation between the different nodes, as the three-dimensional distance between them does not match their actual distance in their original 784-dimensional space. The dist-matrix is by far the most time-consuming part of the algorithm. An alternative to the dist-matrix could have been a KDTree described by Maneewongvatana and Mount in [MM14], for which there are many implementations ready to use even with the pure pixel values of the images. After looking at the documentation[2] however, it became clear that this implementation would see no real performance benefit, as the documentation states: "For large dimensions (20 is already large) do not expect this to run significantly faster than brute force. High-dimensional nearest-neighbor queries are a substantial open problem in computer science," and the idea was therefore abandoned. An illustration of this can be seen in figure Figure 4.2

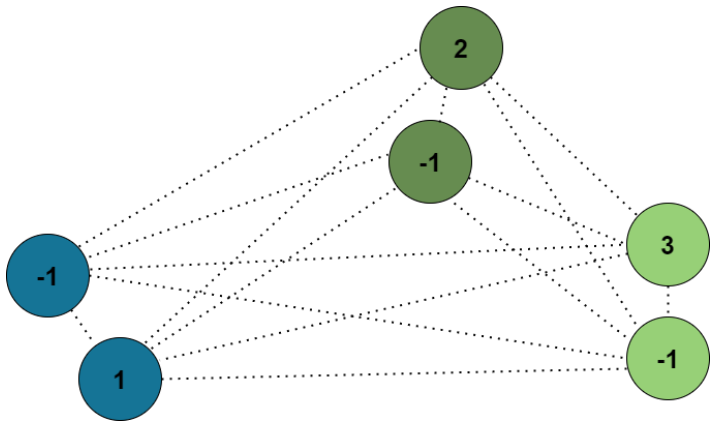


Figure 4.2: Illustration of the distances found between nodes. These can be seen as potential edges, and in the case of the below figure, as all nodes have a valency of 5, if number of neighbours is set to 5, all 5 edges from each node would become an edge in the graph

4.2.3 connectGraph() - Connecting vertices

Given all distances between nodes, the nodes can now be paired with edges. Two parameters are essential in this choice of connections: the number of neighbors nn and the maximum distance between neighbors md. These two parameters are likely to change based on the size of the dataset used and are therefore left as hyperparameters. Skeletonization creates simpler skeletons given more connections. An example of that can be seen in Figure 3.4. Following the illustration of Figure 4.2, Figure 4.3 shows the case of connecting the graph with $nn = 2$.

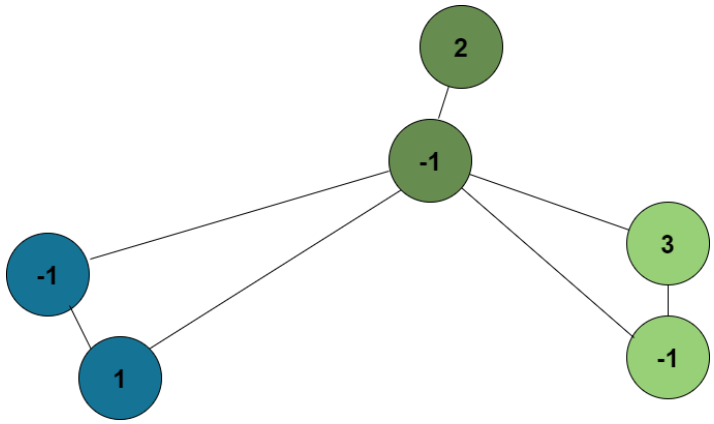


Figure 4.3: Illustration of connecting graph with $nn = 2$ following from Figure 4.2

[2] https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.KDTree.html

An important note is that the distance for clarity is based on the geometric position of the nodes in the 2D space they operate in. In the algorithm, the distances are not based on this as the position of nodes is only an estimate based on the embedding, and the accurate distance between them are calculated based on all pixel values as discussed in Section 3.1.2.

4.2.4 skeletonizeGraph() - Skeletonization of the graph

This part of the pipeline is entirely done by a method already developed in [Son+21], which is discussed in Section 3.3. The skeletonization algorithm both creates this skeleton and a map between original graph nodes and skeleton nodes, `skeletonize` $\rightarrow$ `skeleton, m`. This map is essential as the data structure of the nodes is only a vector and does not hold the labels. To what label the different skeleton nodes represent, it is, therefore, necessary to map between graph and skeleton nodes, as the labels of the graph then can be assigned to the skeleton. If the original graph has an overall low valency, that the number of edges is low in relation to the nodes, some graph nodes may become a separator in themselves. In this case, the label of the graph can be directly assigned to the skeleton. There exists, however, two other cases whose frequency rises in proportion to the valency of the graph: shrunk nodes and helper nodes. These are discussed in **??**

4.2.5 propogateLabels() - Propogating and relabelling skeleton

Er dette ikke bare nearest neighbour

Propagation is, in this context, the notion of spreading information along the edges in the graph. After skeletonization the labels in the output graph can be in 1 of three states: the node has exactly one label, the node has no labels, or the node has more than 1 label, as seen in **??**. The general approach to this is as follows. Search through all nodes in the new skeleton. For each node check the number of labels. If the node has exactly one label continue, if the node has more than 1 label, but there is a majority of one particular assign that majority label to the node. If this is not the case, either because two labels have the same amount or there are no labels at all we need to propagate. This is done by searching neighbours of the nodes.

4.2.6 Predicting

Prediction in SM is done by checking the distance between all points in the model with the new prediction point using the heuristic discussed in Section 3.1.2.

This area of prediction has some methods that might improve accuracy, that was outside the scope of the project. Firstly it might be interesting to look at propogating the answer throughout edges, so the resulting prediction becomes a weighted answer, which might remove some noise and/or randomness. Another interesting possibility that just was not possible because the graph is in 3 dimensions, would be to look at closest edge instead of closest point. This might make the accuracy even better, as the edges should represent some feature of the numbers, that might be interesting.

4.3 K Nearest Neighbours

K Nearest Neighbours is a machine learning method that can be used for both clustering and classification and will in this project be used for classification. KNN can only work with observation and target pairs. It works looking at

Det ville hjælpe med et afsnit med overblik før de mange detaljer. Skriv, med egne ord, hv

5 Experiments and Results

This section aims to test the skeletonization model with KNN on labeled and semi-labeled MNIST data and quantitatively compare the results of the two models. The results will also take a general comparison with the results from Yann LeCun [**yanlecun**] to compare to general optimized models. The small analytical experiments used to optimize the models will also be shown.

As discussed in **??** the main point of this thesis is to test the hypothesis that skeletonization is a viable option in Machine Learning, primarily in terms of accuracy. The focus of this project has thus been on optimizing predictions and not runtime. Therefore prediction results, and specifically accuracy when using a large number of neighbours, will be the primary success criteria, and runtime will not. Nonetheless, runtimes for both building and predicting will be interesting to compare.

5.1 Hardware and software specifications

Both models have been trained and tested on a Windows 10 machine, with a 3.7ghz 12 core ryzen 5900x and 32GB of 3600mhz ddr4 memory. All training and testing have been done in python 3.8.

Primarily because of the size of the distance matrix, training on the full MNIST dataset has not been possible, and the highest amount of images for training data has been 20.000 images. This limitation is accounted for in the KNN model, and both models have had the same amount of data available.

5.2 A note on optimizing hyperparameters

The general approach of optimizing parameters for both SM and KNN is to look at a subset of the training set and split it into training and validation. This smaller model can be made and tested, which does not represent the entire dataset but indicates reasonable parameters. These parameters will then be used with the entire training dataset to find the expected best accuracy.

5.3 Supervised experiment and results

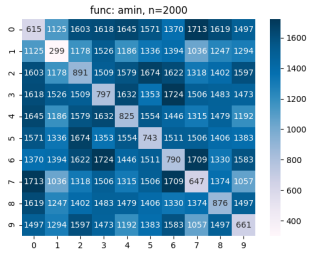
5.3.1 Formulation of goal

This experiment aims to find the best parameters for both SM and KNN, using the test set of 20.000 images, and find their accuracy and runtime using the test set consisting of 10.000 images omitting no labels. Furthermore it will experiment on a varying size of the dataset to see, how size of dataset affects precision. The expectation is that the accuracy of both SM and KNN is around 95% for $N = 20.000$ based on results from [Yan], but as the dataset is $/frac13$ the size of the dataset used by Yan LeCun, slightly worse accuracy might be expected. It is expected that the build time and prediction time of SM will be significantly higher than KNN.

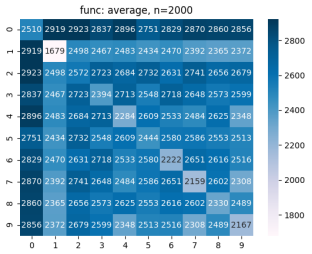
5.3.2 Optimizing hyperparameters of SM

As discussed in Section 4.2 the hyperparameters of SM are the number of neighbors nn and maximum distance between nodes md. The approach is first to find a range that is expected to contain the best distance between nodes. Intuitively the distance should heighten the possibility of nodes with the same labels being connected and nodes without labels not be

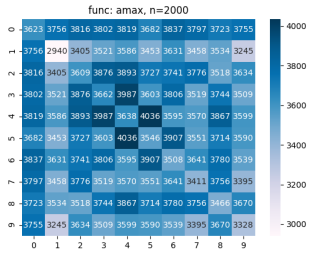
Fine figurer



(a) Minimum distance between images, omitting distances between same image, as min distance for same image is 0



(b) Average distances between images



(c) Max distance between images

Figure 5.1: Confussion matrices showing the minimum, average and maximum between images. Axis are different labels for images. Row 1, Col 1 thus shows the distance between images of zeroes. Row 1, Col 2 shows distance between zeroes and ones, etc.

connected. Looking at the minimum, average, and maximum distance between labels yields the confusion matrices seen in Figure 5.1

To define a suitable range, two values are taken note of: max value of min and max value of average $(max(min), max(avg))$. The minimum value for including all images of the same label. This number can be found in Figure 5.1a and is, in this instance, the minimum distance between two images of label two at value $891 \approx 900$. The number is rounded for practical reasons in the following iteration. The other value is read as the average distance between zeros at $2510\ 2600$. Interestingly looking at the average distances, it is seen that the distance average distance between zeros is higher than the average distance between 1 and all other labels than zero. Whether this is an issue will be commented on in Section 5.3.4. The range to iterate through is now defined as:

$$max(min) = 900, max(avg) = 2600$$

Deciding a range of a number of neighbors is a bit more tricky, as there is no precomputed metric to hint at a good result. The purpose of this paper is to test the effectiveness of skeletonizing a graph, and as already established in Section 4.2.3 increasing the number of neighbors increases the simplicity of the resulting skeleton. Thus it is hoped that a higher number of neighbors yields an increased accuracy. To test this, the number of neighbors is set to $nn = [1; 39]$. To test the whole algorithm of SM shown in listing 4.1 for

$$x_{train} = 2000,\ Y_{train} = 2000, x_{validation} = 1000, y_{validation} = 1000$$

The reason for the comparably tiny subset of the original dataset of 20.000 points is the runtime of the whole algorithm to test and the assumption that the general trend of accuracy will follow from the much smaller subset. If the accuracy of multiple configurations is close, these will be tested separately.

The results shown in Figure 5.2 indicate that a smaller number of neighbors gives better results in this supervised environment. It also suggests a tendency for better accuracy when increasing md. It is observed, that the accuracy does not change for $nn \in [1, 3, 5]$ when $md \leq 1700$. This can be explained by all nodes reaching the maximum number of neighbors nn and the maximum distance, therefore not changing the accuracy. This saturation is an product of the limitation of the dataset because the model has a small selection of nodes. To

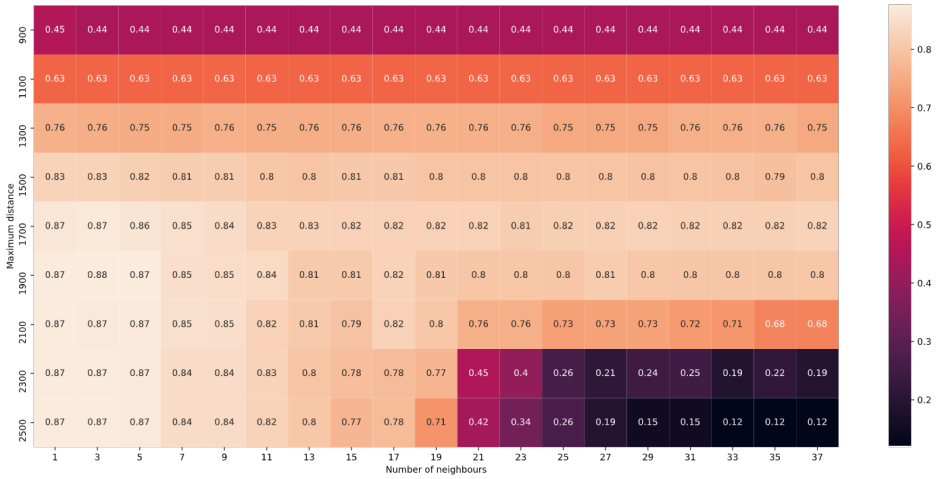


Figure 5.2: Confussion matrix of validation results. $n = 2000$

further investigate the parameters, a second experiment is made on a narrowed down search for $md = [1900, ...2900]$ and $nn = [1, ..5]$ for $n_{test} = 2000$, results are shown in Figure 5.3

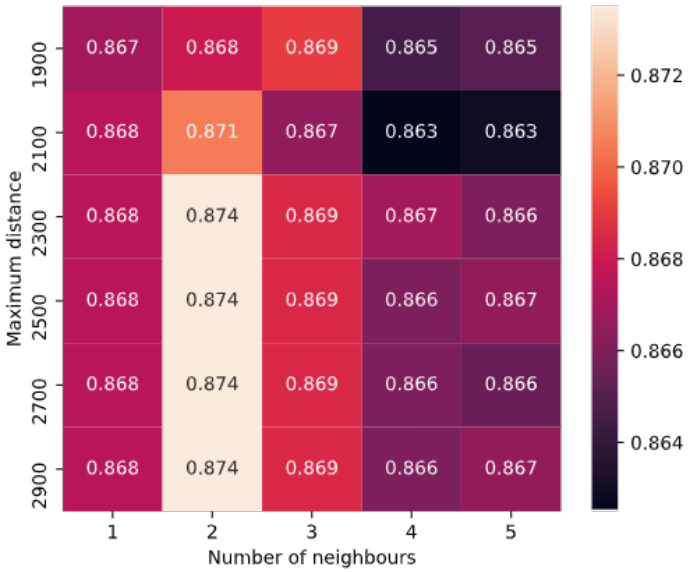


Figure 5.3: Expanded confussion matrix of close to identical results in Figure 5.2

It is noticed, that there is a small variance in the accuracy, but that the best accuracy by 0.05 is located with $nn = 2$. Again a narrowing is done with parameters set to:

$$nn = 2, md \in 2300, ..4300, N_{train} = 3000, N_{test} = 3000$$

for which the result can be seen in Figure A.1. As the precision is not changing in releation to the maximum distance, a max distance is chosen at $md = 2300$, leaving the final hyperparameters at:

$$md = 2300, nn = 2$$

5.3.3 Optimizing hyperparameters of KNN

"K" nearest neighbors has one hyperparameter namely the "K" refering to number of neighbours. The tuning is done by taking the training data set, and splitting it into a training and validation set. Then by testing this smaller experiment the precision is measured. The value of nn that yields the highest accuracy is chosen. Looking at Figure 5.4 the optimal number of neighbours is read as 1.

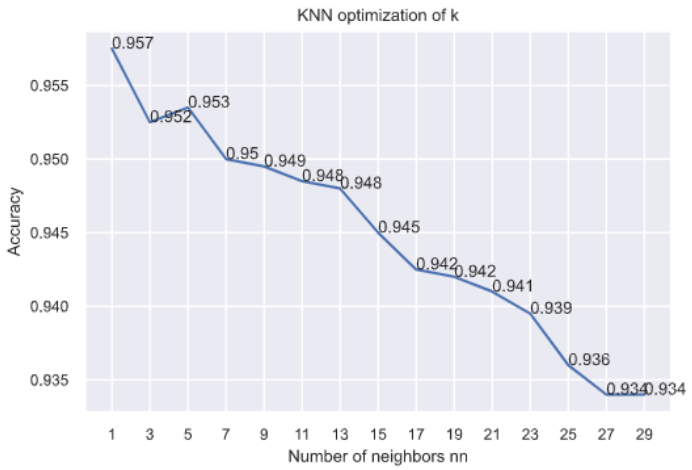
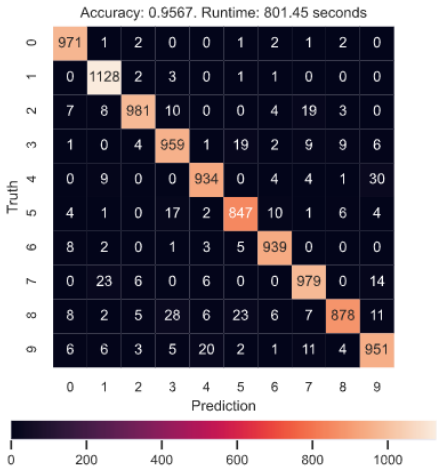


Figure 5.4: Precision running KNN on $N_{train} = 18000$, $N_{validation} = 2000$

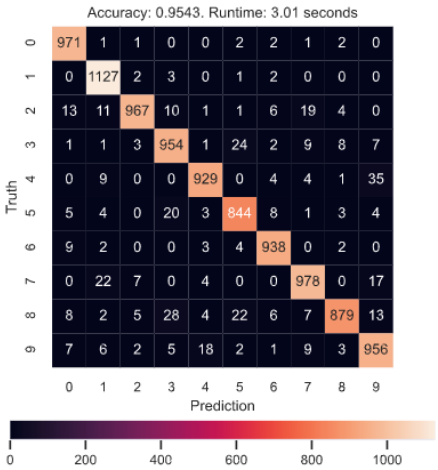
5.3.4 Results and discussion

This section will show the precision and runtime of using full dataset with optimized hyperparameters in SM and KNN, as well as the result of varying the size of the dataset.

Full dataset



(a) Confussion matrix of supervised SM. nn=2,$md = 2300$, $N_{train} = 20.000$, $N_{test} = 10.000$



(b) Confussion matrix of supervised KNN. $k = 1$, $N_{train} = 20.000$, $N_{test} = 10.000$

The accuracy between SM and KNN is close to identical, the runtime of the algorithms however are not, where the difference between is over a 200th fold. That does not incentives to use SM over KNN. The number of neighbours used is also very low meaning the graph is not getting reduced by a lot as established in **??**. Table 5.1 shows the number of nodes in input graph and output skeleton, and as seen in Figure 5.3 there was some advantage in precision in increasing the number of neighbors. Because the method of prediction in SM is to find the closest node in the skeleton, and assign that nodes label to the prediction node,

we can be sure, that the increase in accuracy is an effect of skeletonization and not because of a better connected graph, simply because changing the number of neighbours in the graph would not change the predicted values, as the position of nodes in the input graph does not change, and thus the only reason for an increase in performance is because skeletonization shrinks and thereby moves the position of the nodes along the trends of the data by taking the average pixelvalues of the shrunk nodes. –This should be moved to skeletonization or algorithm–.

	Graph	Skeleton	Reduction
Nodes	20.000	18826	5.87%

Table 5.1: Table showing number of nodes in graph and skeleton of SM using $nn = 2$ and $md = 2300$

Varying size of dataset
Because of the increasing running time of SM, the stepping size of increasing N is relatively large.

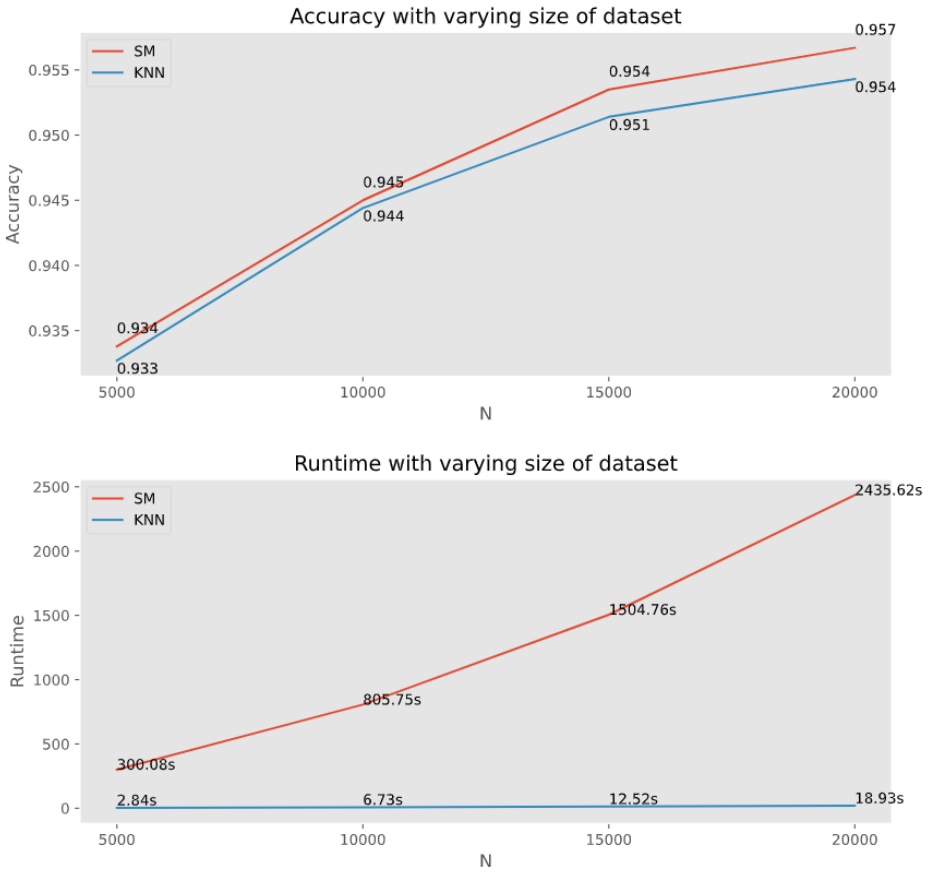


Figure 5.6: Accuracy and precision of SM and KNN for $N_{train} \in [5000, 10000, 15000, 20000]$

5.4 Semi-Supervised experiments and results

5.4.1 Formulation of goal

The following experiments aim to find how good SM is at semi-supervised learning compared to more conventional method in KNN. First experiment 90% of labels will be forgotten for both SM and KNN for $N = 20.000$. Thereafter experiments will be conducted to find the point at which SM drops below a certain accuracy determined from the result of the first experiment. KNN will be tested alongside for the same percentage of forgotten labels to see whether SM at some point significantly outperforms KNN.

5.4.2 Optimizing hyperparameters of SM and KNN

The procedure for finding optimal hyperparameters will be the same as Section 5.3.2 and Section 5.3.3, and the results of those optimizations can be seen in Appendix B. Very important here is, that this semi supervised environment is artificial, as all labels are actually known, and is an emulation of a real scenario where labels are actually unknown. The point of this is, that previous configuration of hyper parameters cannot be used as inspiration for this implementation.

5.4.3 Results and discussion

This

Ved hvert forsøg kan der være samme historie: Her er hv

Til sidst kan man diskutere de store træk og man kan se fremad: hvad kunne være int

6 Conclusion

Bibliography

[Tur50] A. M. Turing. “Computing Machinery and Intelligence”. English. In: *Mind*. New Series 59.236 (1950), pp. 433–460. ISSN: 00264423. URL: http://www.jstor.org/stable/2251299.

[Wic03] Florian Wickelmaier. “An Introduction to MDS”. In: (2003). URL: https://www.mathpsy.uni-tuebingen.de/wickelmaier/pubs/Wickelmaier2003SQRU.pdf.

[Jer05] Xiaojin Zhu (Jerry). “Semi-Supervised Learning Literature Survey”. In: (2005). URL: http://digital.library.wisc.edu/1793/60444.

[BM08] Zalan Bodo and Zsolt Minier. “ON SUPERVISED AND SEMI-SUPERVISED k-NEAREST NEIGHBOR ALGORITHMS”. In: *Studia Universitatis Babeş-Bolyai. Informatica* 53 (Jan. 2008).

[Pet09] L. E. Peterson. “K-nearest neighbor”. In: *Scholarpedia* 4.2 (2009). revision #137311, p. 1883. DOI: 10.4249/scholarpedia.1883.

[FI14] Yasuhiro Fujiwara and Go Irie. “Efficient Label Propagation”. In: Proceedings of Machine Learning Research 32.2 (22–24 Jun 2014). Ed. by Eric P. Xing and Tony Jebara. URL: http://proceedings.mlr.press/v32/fujiwara14.html.

[MM14] Songrit Maneewongvatana and David M. Mount. “It’s okay to be skinny, if your friends are fat*”. In: (2014). URL: https://www.cs.umd.edu/~mount/Papers/cgc99-smpack.pdf.

[Sil17] Thalles Silva. “Semi-supervised learning with Generative Adversarial Networks (GANs)”. In: (2017). URL: https://towardsdatascience.com/semi-supervised-learning-with-gans-9f3cb128c5e.

[BR20] Andreas Bærentzen and Eva Rotenberg. “Skeletonization via Local Separators”. In: *CoRR* abs/2007.03483 (2020). arXiv: 2007.03483. URL: https://arxiv.org/abs/2007.03483.

[pyi20] pyim59. “Starter: MNIST in CSV 5b700bc9-3”. In: (2020). URL: https://www.kaggle.com/pyim59/starter-mnist-in-csv-5b700bc9-3.

[AAB21] Akshay Agrawal, Alnur Ali, and Stephen Boyd. “Minimum-Distortion Embedding”. In: *arXiv* (2021).

[Son+21] Zixing Song et al. “Graph-based Semi-supervised Learning: A Comprehensive Review”. In: *CoRR* abs/2102.13303 (2021). arXiv: 2102.13303. URL: https://arxiv.org/abs/2102.13303.

[Min] Zhen Zhang Ming Wu. “Handwritten Digit Classifications the MNIST Data Set”. In: (). URL: https://www.academia.edu/6509648/Handwritten_Digit_Classification_using_the_MNIST_Data_Set.

[Yan] Christpoher J.C. Burges Yann LeCun Corinna Cortes. In: (). URL: http://yann.lecun.com/exdb/mnist/index.html.

[Bha17] Rahul Bhalley. “Digit Recognition”. In: (Mar 12, 2017). URL: https://towardsdatascience.com/mnist-with-k-nearest-neighbors-8f6e7003fab7.

[TMll] Mikkel N. Schmidt Tue Herlau and Morten Mørup. *Introduction to Machine Learning and Data Mining*. Technical University of Denmark, 2019 Fall.

A Plots

A.1 Expanded heatmap for $nn = 2,\ md \in [2300, ..., 4300]$

Figure A.1: Expanded heatmap of close to identical results in Figure 5.3

B Semi-supervised hyperparameter optimization results

B.1 SM results

B.2 KNN results

C Links

Link to repository: https://github.com/Moesen/BachelorProject