New Systems for Graph Classification

Using learned graph embeddings to obtain support vector machine kernels

Master Thesis

Matteo Raphael Biner

University of Bern
Faculty of Science, Pattern Recognition Group
PD Dr. Kaspar Riesen, Kalvin Dobler

May 2025

Abstract

This thesis investigates new methods of graph classification with the help of graph neural networks. Based on earlier solutions, it presents a broad spectrum of possible designs and compares them between each other. This helps with finding directions in which to start deeper investigations. The focus of this thesis lies on the feasability and the accuracy of the presented solutions.

In a first step, this research evaluates different designs of graph neural networks by comparing them to a reference system. Then it describes the development of a kernel for a support vector machine. Different similarity measures for the support vector machine are explored and compared to the previously developed systems.

The results obtained by this research indicate that all approaches presented in this thesis lead to working solutions. Algorithms with significantly worse performance are not discussed in detail. Given the scope of implemented solutions, there is a solution for every dataset that outperforms the reference system. However, there is no version of the support vector machine classification that outperformes the reference system for all datasets, indicating that better solutions could still be developed with more research.

Overall, this thesis gives a deep insight into the possibilities that arise from using graph neural networks for classification tasks. Thus providing valuable information about the direction in which future work can go.

Contents

1 Introduction

The field of *artificial intelligence* has developed over a long period of time and therefore encompasses a broad spectrum of technologies. Especially with recent advancements through the use of *neural networks*, more research is conducted in this area, thus further broadening the field. While there are many ways to group algorithms into subfields of artificial intelligence, avoiding any overlap between the fields is difficult. This thesis remains within the scope of *machine learning*.

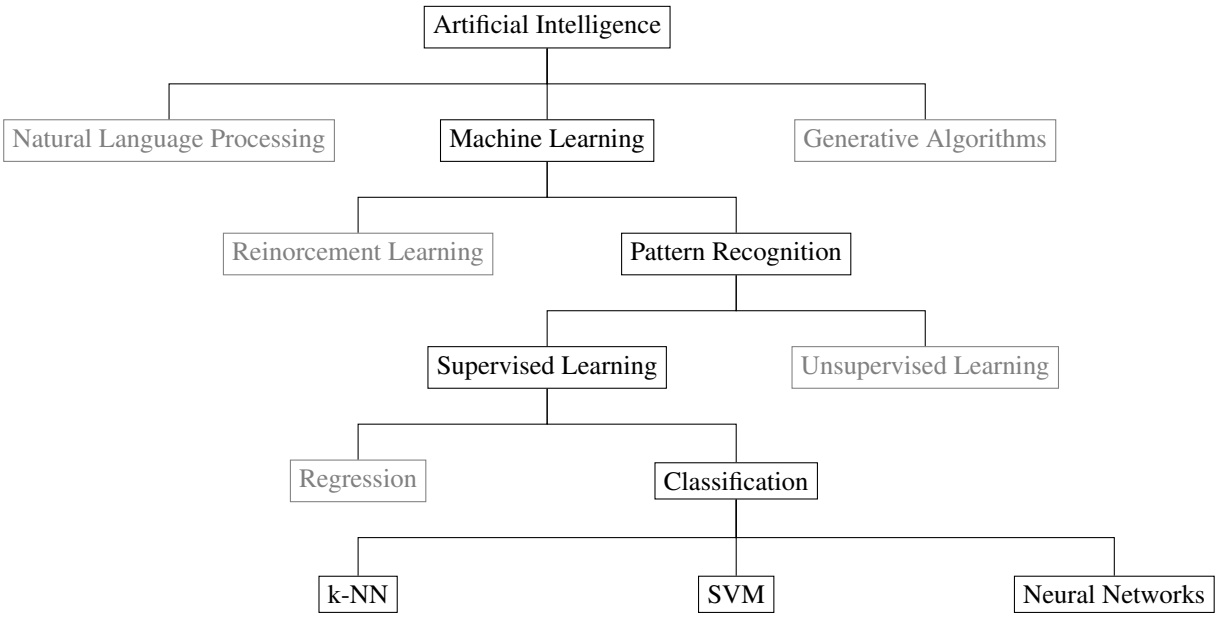


Figure 1.1: Overviev of the used technology in the context of artificial intelligence.

Figure 1.1 indicates where the topics of this thesis lie within the larger scope of artificial intelligence. Each level of the tree could be expanded on more, but only some areas are shown as examples. The topics relevant for this thesis are presented in black, while the other topics are in light gray. There are multiple ways in which all the subcategories could be arranged and divided, with no objectively correct answer. While machine learning is in itself a subcategory of artificial intelligence, machine learning techniques are also used in other fields of artificial intelligence. The field of *pattern recognition* is here divided based on the amount of information that the algorithm is given during training. This thesis uses *supervised learning* to do *classification*, meaning that each entry in the training set is labeled with its corresponding class. What Figure 1.1 does not fully cover is the form in which the data is presented to the algorithm. Including that context into the division of pattern recognition provides jet another view.

When focusing on the context of the data, pattern recognition can be divided into *statistical pattern recognition* and *structural pattern recognition*. The former relies on the representation of data in the form of feature-vectors, while the latter keeps more information about the structure of the data through the use of graphs. The field of *graph-based pattern recognition* has itself a long history and can not only be divided into different categories, but also into different eras. In the first era, graph matching and graph clustering algorithms were explored for different purposes. Graph kernels were developed in a second era to facilitate the use of a broader spectrum of known algorithms for graph-based data. In the third era, algorithms using neural networks were adapted to structural pattern recognition.

Altough this thesis mainly focuses on structural pattern recognition, some algorithms can be explained more easily in the domain of statistical pattern recognition. In fact, most algorithms used in this thesis have first been developed in the domain of statistical pattern recognition. In recent years advancements in neural networks have opened up new possibilities for the field of pattern recognition. There is also a large potential benefit in combining neural networks with more traditional algorithms, wich this thesis attempts. A previous attempt at this combination, done by Dobler and Riesen [7] on this topic, explores the use of a *graph isomorphism network* (GIN) to estimate the *graph edit distance* (GED) and using the *k-nearest-neighbor* (k-NN) algorithm for classification.

The primary goal of this thesis is to broaden the spectrum of algorithms used for graph classification. On the side of *graph neural networks* (GNNs), there are other options that can be explored besides GIN. In this thesis, *graph attention networks* (GATs) and *graph convolutional networks* (GCNs) are investigated and compared to GIN. What all these network designs have in common is that they pass information across a graph. The way their weights are trained and mapped to the graphs is where the main difference lies. Concerning classification algorithms, *support vector machines* (SVMs) are the only candidates besides k-NN that are considered in this thesis. The connection between GNNs and SVMs involves enough decisions that a wide range of different solutions can be developed. One benefit of the use of SVMs is that it provides a solution without the calculation of the GED. While the computational complexity of the system is still dominated by the matching of nodes between graphs, this approach could yield more performant solutions in practice. The conclusions drawn from the comparisons in this thesis may be helpful for future work, where these algorithms could be refined or even combined.

There are considerable challenges in combinig methods such as GNNs and SVMs. Given that multiple GNN architectures are tested, a well-performing version of each GNN has to be obtained, which requires computationally expensive training. To ensure generalization, the final solutions have to be compared based on multiple datasets. Testing solutions on different datasets can lead to inconclusive results, if one GNN architecture performs best on a dataset A, while another performes best on a dataset B. This statement can also be extended to the exact design of each architecture. Small design choices in the GNNs may vary, if they are optimized based on one dataset instead of another. Given the existing solution

using GIN and k-NN, some statement can be made for the performance of GAT and GCN. This makes it easier to see if a change in the model improved it or not. For the classification using a SVM, there is an inherent lack of knowledge concerning its potential. In addition to that, the space of design choices that can be made is very large. Datasets need to be picked carefully because, due to random variance in the results, good designs may be overlooked in some cases. It is also possible to assume a design to be good because it randomly performs well under certain conditions. Therefore, it can help to make the choice of datasets based on the performance of the reference systems. While testing datasets where the reference system performes badly can give valuable insight into a new solutions potential, testing datasets where the reference system performs well can help eliminate bad solutions early.

Testing different designs of GNNs helps future investigations in this field. Even if all designs had similar performance, that would be relevant information, indicating that the first choice of a specific GNN is not as critical. One idea behind using a SVM, rather than k-NN-Classification, is to use the kernel trick to obtain a different separation of datapoints, which could make badly separable datasets easier to separate. While some temporal efficiency gain may be expected from the SVM-Classification, finding solutions that worked proved to provide a large enough scope for this thesis without the requirement of filtering them for time efficiency. Instead, a deeper investigation into the nature of the datasets is presented. This could help with deciding which kinds of datasets show the highest potential for an accurate classification.

The next chapter introduces the theoretical background required for the solution. It begins with the description of the classification problem in the domain of statistical pattern recognition, before moving on to its application in the domain of structural pattern recognition which includes the description of the reference system. In Chapter 3 some ideas behind the existing solution developed by Dobler and Riesen [7] are presented. That solution is based on a GIN and classifies graphs using the k-NN algorithm with the GED as a distance metric. After that, the developed GNNs are presented in detail, which includes the GED estimation process by which the different GNN designs are first evaluated. All the SVM designs with different similarity measures are presented at the end of the chapter. Chapter 4 beginns with an introduction to the datasets used in this thesis and the reasoning behind those choices. It then presents the results obtained by the GNN-GED estimations and compares them between each other, as well as to the reference system. That chapter also includes some intermediate results that are necessary for the validation of the solutions metaparameters. The comparison of the accuracy of all the developed systems is presented, before some visual investigations into the reasons for misclassifications conclude that chapter. Finally, in Chapter 5 some conclusions about the performance of the solutions are drawn, before laying out possible paths for future work.

2
Theory and Related Work

This chapter provides an overview of all the theory required to develop the new solution, as well as some insights into the related work which this thesis is based on. The chapter is divided into two sections. Section 2.1 introduces the general concept of classification with some classical algorithms in the domain of statistical pattern recognition. In Section 2.2 these concepts are adapted to structural pattern recognition with algorithms that are adapted to graphs.

2.1 Classification of Feature Vectors

This section introduces the task of classification, together with some algorithms that solve this task. To obtain a good basic understanding, the focus of this section remains within the realm of statistical pattern recognition [7]. This means that the data is represented using n-dimensional feature vectors instead of graphs.

2.1.1 General Procedure

Classification is only one of many tasks that are typical in machine learning. The problem, as described by Patrick and Fischer [16], is a special case of supervised discrimination. Given a feature vector x from a set of M classes denoted $\omega_1, ..., \omega_M$, the goal is to assign x to the correct class ω_i. If x belongs to the class ω_i, then there is a probability density function $f(x, \omega_i)$ that is assumed fixed, continuous and unknown, while the probability P_i that x belongs to class ω_i is known. The model that solves this problem can be described as a learner that produces a hypothesis h, as is done by de Lacerda et al. [5]. This hypothesis is generated by adjusting a set of parameters λ based on a training set D. This set consists of l samples from the original dataset of size n. For each sample vector $x \in D$, there is also a label y which corresponds to the class ω_i of x. The parameters λ are then tuned to associate that vector x with its label y. To evaluate the model's performace, the test set consisting of $n - l$ sampels, that are not in the training set, is given to the model. After predicting the labels of all vectors in the test set, the simple accuracy measure can be calculated by dividing the number of correct labels by $n - l$, the number of samples in the test set. In practice, more elaborate techniques to assess performance are used, that will be described later. One problem of simply optimizing for one specific training set is that the model can be overfitted to the training

set, thus not adapting well to new data. Another problem is the initialization of parameters λ_0, which could rely on an extensive manual search.

There are many approaches to solving these problems, but for this thesis one approach suffices. As suggested by Kohavi [13], some improvements on performance can be made using *crossvalidation*. In this procedure, the training set is split again into k mutually exclusive sets denoted $D_1, ..., D_k$. The model is then trained on $D \setminus D_i$ and its performance is evaluated on D_i. This process is then repeated k times, leaving out each D_i at some point. To evaluate the performance one calculates the average accuracy over all repetitions. Using this technique for a set of predefined initial parameter values yields values that work well for new data. Finally, the model is tested on the test set with the best parameters, giving a result that allows it to be compared to other models.

A lot of different algorithms can be used to do classification. Giving an introduction into all popular algorithms would be out of the scope for this thesis. Hence, only the algorithms that are relevant for this thesis are presented in the next sections.

2.1.2 Traditional Algorithms

The first algorithm to be introduced is the *k-nearest-neighbor classification* (k-NN-Classification). Patrick and Fischer [16] have a good description for it. The requirements for this algorithm are a parameter $k \in \mathbb{N}$ and a distance metric, that is defined on the vectorspace of the samples. Given a vector x in the test set, k-NN-Classificaiton considers the k vectors $x_0, ..., x_k$ in the training set that are nearest to x denoted $N(x)$. The decision is then based on the labels of the majority of vectors in $N(x)$. Or more precisely, the decision $h_k(x)$ can be described as follows:

$$h_k(x) = \max_{j \in M} \sum_{x_i \in N(x)} \delta_j(y_i) \tag{2.1}$$

where M is the set of classes (e.g. $\{0, 1\}$), δ_j is the kronecker delta symbol corresponding to the class j and y_i is the label of x_i. To break ties for datasets with two classes, k can be chosen as an odd number. If there are more classes, the assigned class can be arbitrary. In a practical implementation, this means that the decision is based on the ordering of the data.

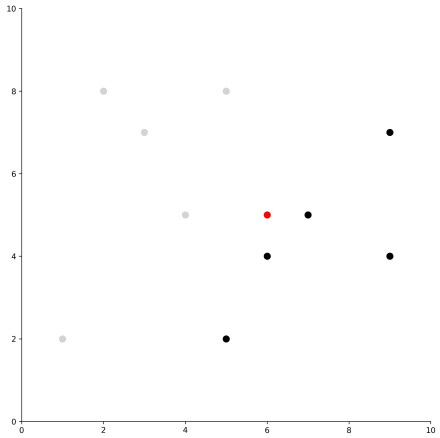


Figure 2.1: Example of a classification problem that can be solved with k-NN-Classification.

Figure 2.1 shows an example of k-NN-Classification. Here, the two classes in the training set are represented in black and gray respectively, and the point in the test set is represented in red.

Applying k-NN-Classification for $k = 3$ with the euclidian distance as the distance metric yields the points at (4, 5, gray), (6, 4, black), and (7, 5, black) as the nearest neighbors. Because two out of three neighbors are labeled black, the tested point is classified as part of the black set. Intuitively this looks like a reasonable classification, but the actual class of the red point could still be gray.

One can use cross validation to optimize the parameter k by trying out a set of integers, for example $k \in \{1, 3, 5, 7, 9, 11\}$. The k-NN-Classification algorithm also has some weaknesses compared to other algorithms. For the problem investigated in this thesis, obtaining a meaningful distance metric is an expensive computation. The next algorithm described does not require a distance metric, but rather a similarity measure.

Support vector machines (SVMs) can be used to solve a variaty of problems including classification, regression and distribution estimation. The SVM for this thesis is implemented using the sklearn library, which is based on the LIBSVM library as described by Chang and Lin [4]. The focus of this thesis lies on classification. Therefore, one uses the *C-support vector classification* (C-SVC) version of the LIBSVM library. Specifically, the implementation uses C-SVC as described by Chang and Lin[4] and Boser et al.[3] to solve the following optimization problem:

Given a set of training vectors $x_i \in \mathbb{R}^n, i = 1, ..., l$ and a label vector $y \in \mathbb{R}^l$ for two classes, such that $y_i \in \{1, -1\}$ solve:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2} w^T w + C \sum_{i=1}^{l} \xi_i \\ \text{subject to} \quad & y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \quad i = 1, ..., l, \end{aligned} \tag{2.2}$$

where $\phi(x_i)$ maps x_i into a higher dimension and $C > 0$ is the regularization parameter. To solve this primal optimization problem, it is transformed into the following dual problem:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2} \alpha^T Q \alpha - e^T \alpha \\ \text{subject to} \quad & y^T \alpha = 0, \\ & 0 \leq \alpha_i \leq C, \quad i = 1, ..., l, \end{aligned} \tag{2.3}$$

where $e^T = [1, ..., 1]^T$ is the vector of all ones, $Q_{i,j} = y_i y_j K(x_i, x_j)$ with a custom kernel matrix $\kappa = K(x_i, x_j)$. The regularization parameter is optimized manually with the constraints: $C = 10^k$ for $k \in \{-10, -9, ..., -1, 0, 1, ..., 9, 10\}$. The implementation for this thesis uses different versions of kernel matrices, which will be elaborated on later. After solving the problem 2.3, one obtains an optimal w by using the primal-dual relationship. This w satisfies:

$$w = \sum_{i=0}^{l} y_i \alpha_i \phi(x_i) \tag{2.4}$$

The decision function for the classification is then:

$$sgn(w^T\phi(x) + b) = sgn(\sum_{i=0}^{l} y_i\alpha_i K(x_i, x) + b). \tag{2.5}$$

This decision function yields a separation boundary, that aims to fully separate the classes. The shape of said boundry depends on the kernel matrix that was used. All parameters are then stored in the model for the prediction. For the case where the labels are for more than two classes, the ”one-versus-one” approach is used. This means that for n classes, $\frac{n(n-1)}{2}$ classifiers are constructed. Each classifier is then trained to destinguish one class form another class [2].

Figure 2.2 shows an example of SVM-Classification. Again, the two classes in the training set are represented in gray and black respectively, while the point from the test set is represented in red. The blue line represents the decision boundary with a linear kernel, that was optimized on the training set.

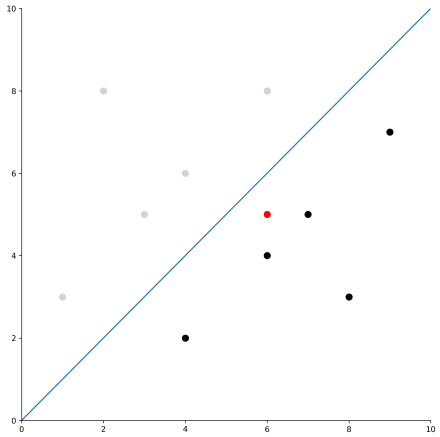


Figure 2.2: Example of a classification problem that can be solved with SVM-Classification.

The red point will be classified as part of the black class, because it lies on the same side of the separation boundary. This could still possibly be a wrong classification, because there could be an actual boundary between the classes that is shifted to the bottom right. In some cases the classes could also be inseparable by a linear boundary. That is why the choice of kernel matrix can matter a lot for SVM-Classification.

2.1.3 Modern Classification Algorithms

Another approach to classification is the use of *neural networks*, which are loosely inspired by neuroscience. Goodfellow et al. [12] explain that a neural network tries to estimate a function f^*. For classification, this means that the function f^* maps the input x to a label y. The neural network now tries to estimate that functioin with a parameter θ, such that $f(x, \theta) = y$. This is typically done by combining multiple functions in *layers*, meaning $f(x) = f_n(f_{n-1}(...(f_2(f_1(x)))))$ where n is called the *depth* of the neural network. The function f_1 is called the *input layer*, f_n is called the *output layer*, and all other layers are called *hidden layers*. Each layer is represented by a vector, whose elements can be interpreted as neurons, because they play a similar role as neurons do in neuroscience. Each element of the layer i calculates an *activation value*, based on weights that are assigned to all values of the layer $i - 1$. The layers may

also vary in size. Only the input- and outputlayer have to be adapted to the sizes of the input and output respectively. For classification specifically, the number of output nodes has to match the number of classes available. Figure 2.3 shows a small example of such a network. A network like this could be used for a dataset where each vector has three features, and is part of one of two classes. The final two nodes would then give the probabilities for each class, that the vector belongs to said class.

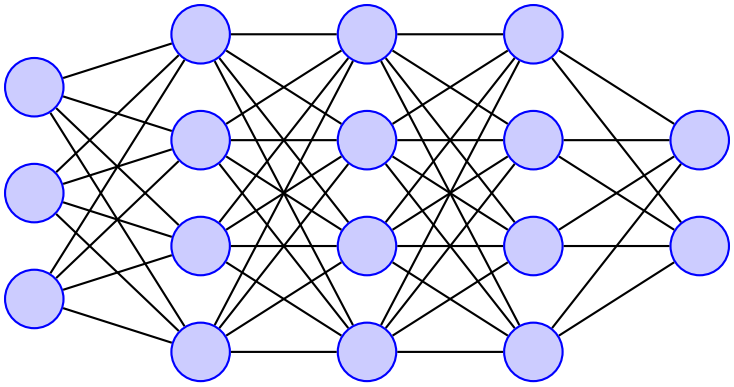


Figure 2.3: Example of a neural network with three nodes in the input layer, four nodes in each of the three hidden layers, and two nodes in the output layer.

A more detailed description of the calculations actually used for this thesis, will be given in Chapter 3. The general calculation of a neural network is based on the following formula for the first layer [12]:

$$h^{(1)} = g^{(1)}(W^{(1)T}x + b^{(1)}) \tag{2.6}$$

where g represents the activation function, W is a weight matrix, b is a weight vector, and x is the input vector. For subsequent layers $h^{(i)}$, x is replaced by $h^{(i-1)}$.

2.2 Graph-Based Classification

This section provides the graph-related definitions used in this thesis, as well as the explanation of the reference system. To understand the latter, the notion of the *graph edit distance* (GED) has to be introduced as well. The basic idea used by Dobler and Riesen [7] for the new system is introduced in Section 2.2.2. Also, the methods described in Sections 2.1.2 and 2.1.3 are now applied to graphs. Thereby showing how they are applicable to structural pattern recognition [7].

2.2.1 Graph Edit Distance

The GED, as described by Riesen and Bunke [19] is a measure of similarity between two graphs and has many applications. In this case, it is used in the classification of graphs. The main issue with the GED is that its computational complexity is very high. Therefore, it is often estimated, which is also a goal of this thesis.

Given two graphs G_1, and G_2, one calculates the GED by transforming one graph into the other. There are many operations that can be performed to achieve this goal. Here, only some of the node edit operations and edge edit operations are considered. There are substitution, deletion and insertion for nodes, and deletion and insertion for edges. The lack of a substitution operation for edges lies in the fact that the edges are not always labled in the investigated datasets. For nodes u and v, one denotes substitution

with $(u \to v)$, the deletion with $(u \to \epsilon)$ and the insertion with $(\epsilon \to v)$. For edges, usually denoted by e, a similar notation is used. Each of these operations has a cost assigned to it, that will be used in the calculation of the GED. For the purposes of this thesis, the following costs are used for node operations:

$$cost(u \to v) = \begin{cases} 1 & if\ label[u] \neq label[v] \\ 0 & if\ label[u] = label[v] \end{cases} \tag{2.7}$$

$$cost(\epsilon \to v) = 1 \tag{2.8}$$

$$cost(u \to \epsilon) = 1 \tag{2.9}$$

where u, and v are nodes. For edge operations, the costs look as follows:

$$cost(e \to \epsilon) = 1 \tag{2.10}$$

$$cost(\epsilon \to e) = 1 \tag{2.11}$$

The sequence of operations used in the transformation of one graph into another is called the edit path, denoted $o_1, ..., o_k$. The GED is defined on the optimal edit path, that leads to the lowest total cost. Hence, the GED is defined as:

$$ged(G_1, G_2) = \min_{(o_1,...,o_k)\in\Gamma} \sum_{i=0}^{k} cost(o_i) \tag{2.12}$$

where Γ is the set of all possible edit paths. The number of possibilities for such edit paths grows exponentially with the size of the graphs. This is because the nodes of a graph G_1 need to be matched to the nodes of graph G_2. For two graphs of the same size $|G_1| = |G_2| = n$, there are $n!$ ways to match their nodes. Hence explaining the high complexity and the need for estimation mentioned above.

As Riesen and Bunke [19] show, one can calculate an estimation of the GED more efficiently with their *bypartite graph edit eistance* (BP-GED) algorithm. Their system also serves as a good reference for the solutions developed in this thesis. The difficult part of computing the GED is to find a matching between the nodes in both graphs. This can be formulated as an *assignment problem*, where elements of a set A need to be assigned to elements of a set B of the same cardinality. This problem can be reformulated as finding an optimal matching in a complete bipartite graph. To solve this problem, Riesen and Bunke use Munkres' algorithm. Explaining this algorithm in detail is out of the scope of this thesis, but most importantly, it gives an optimal solution to the assignment problem in $O(n^3)$ time. This result is achieved by doing smart operations on a cost matrix C, where $C_{i,j}$ represents the cost of assigning node $n_i \in G_A$ to node $n_j \in G_B$. Here G_A and G_B refer to the parts in the bipartite graph that represent the sets A and B.

The adaption of Munkres' algorithm to graphs is done by defining a cost matrix C that represents node assignments in a meaningful way. It is defined on the graphs G_1 and G_2 from the original problem, that represent G_A and G_B in the bipartite graph:

$$C = \begin{bmatrix} X & Y \\ Z & 0 \end{bmatrix} \tag{2.13}$$

where, with $n = |G_1|$ and $m = |G_2|$, X is a $n \times m$ matrtix representing node substitutions, Y is a $m \times m$ matrix representing node deletions, Z is a $n \times n$ matrix representing node insertions, and 0 is the $m \times n$ matrix with all zeros. Because nodes can only be inserted or deleted once, all the non-diagonal

entries in Y and Z are set to ∞. It is also important to mention, that the node operations imply edge operations that are added to the total cost.

Applying Munkers' algorithm to this matrix yields an optimal node assignment, that minimizes the cost of node operations. However, it is only a suboptimal solution to the GED problem. The reason for this is that for each node operation only the local structure is considered, thus being unable to infer edge operations dynamically. Hence, the BP-GED algorithm only gives an upper bound for the true GED of the two graphs. While a lower bound can also be calculated based on the same system, the reference system is based on the upper bound.

2.2.2 Node Embeddings

To estimate the edit distance between two graphs G_1 and G_2, one tries to match each graph's nodes in a sensible manner, thereby minimizing the search space of possible matchings. Another way to do this is by first obtaining node embeddings, that can later be used to give a distance metric for the matching process. For that, each GNN is trained for the classification of the graphs. Taking the values of the last GNN-layer yields a 64-dimensional vector for each node in the graph. Having obtained these node embeddings, the pairwise distance between all nodes in the graph G_1 and the graph G_2 is calculated using a distance metric. The original version developed by Dobler and Riesen [7] used the euclidean distance as a metric. To evaluate the effects of the choice of metric, the cosine distance is being used for this thesis. It is defined as:

$$d(u, v) = 1 - \frac{u \cdot v}{\|u\|_2 \|v\|_2} \tag{2.14}$$

where $\|u\|_2$ and $\|v\|_2$ are the respective two norms of the vectors u and v, and $u \cdot v$ is the dot product of the vectors u and v. This yields a $n \times m$ matrix, where n is the number of nodes in G_1 and m is the number of nodes in G_2. With that matrix, a linear sum assignment problem can be solved to assign the nodes. The linear sum assignment problem can be described by the formula:

$$min \sum_{i \in U} \sum_{j \in V} C_{i,j} X_{i,j} \tag{2.15}$$

where $C_{i,j}$ is the cost calculated by the cosine distance $d(u, v)$, and $X_{i,j} = 1$ if and only if the row i is assigned to the column j. The nodes are then assigned according to $X_{i,j}$, yielding a matching M. In the next step the edit distance between the two graphs G_1 and G_2 is calculated based on the definition in Section 2.2.1. Calculating this graph edit distance for each pair of graphs yields a distance matrix D. This distance matrix can now be used to classify the graphs with the k-NN method described in Section 2.2.3.

2.2.3 Traditional Classification Algorithms for Graphs

To adapt k-NN-Classification to graphs, one only needs a distance metric. One promising candidate for this metric is the GED, wich can be estimated in different ways. Riesen and Bunke [19] show the BP-GED estimation discussed in Section 2.2.1. Another estimation of the GED, as used by Dobler and Riesen [7], is based on graph matching through node embeddings. These node embeddings need to be obtained by a system that conveys information about the graphs' structures, as well as their class. This solution will be discussed in more detail in Section 3.1.

The SVM-Classification described in Section 2.1.2 can most easily be adapted to graphs with a precomputed kernel matrix. Each entry in that matrix corresponds to a measure of similarity between two graphs. Since the graph edit distance is a dissimilarity measure, one approach to obtaining said matrix is to take a kernel function $k(d_{ij})$ of the graph edit distance between all pairs of graphs G_i, G_j. This needs to

be done in a way, that leads to a similarity measure. Some trivial examples from Riesens lecture [20] of functions that achieve this are as follows:

$$\begin{aligned} k_1(d_{ij}) &= -d_{ij} \\ k_2(d_{ij}) &= -d_{ij}^2 \\ k_3(d_{ij}) &= tanh(-d_{ij}) \\ k_4(d_{ij}) &= exp(-d_{ij}) \end{aligned} \tag{2.16}$$

A solution with k_1 as a kernel was tested as a proof of concept. While that solution did work, it has been outperformed by at least one version of the actual solution for each of the tested datasets.

More involved kernels have also been suggested. One example of a kernel for graphs is the *direct product kernel*, as described by Gärtner [10]. This kernel considers *walks* on graphs (i.e. sequences of nodes) that have equal labels and may possibly contain gaps. Computing the product graph allows for efficient computation of these walks. The direct product graph contatins a node for every pair of nodes in the original graphs that have the same labels. It contains an edge between two nodes whenever an edge between the corresponding original nodes is also present and has the same label. The kernel computation is then based on the adjacency matrix $E_\times$ of this direct product graph. To be more precise, the calculation depends on the limit of matrix power series involving $E_\times$. If this limit exists, the formula for the graph kernel looks as follows:

$$k_\times(G_1, G_2) = \sum_{i,j=1}^{|V_\times|} \left[\sum_{n=0}^{\infty} \lambda_n E_\times^n \right]_{ij} \tag{2.17}$$

with $|V_\times|$ being the size of the direct product graph and $\lambda_n \in \mathbb{R}$. An appropriate choice of λ allows for the calculation of this kernel in cubic time. Section 3.2 shows the actual solution, another appoach to obtaining a kernel for SVM-Classification based on GNNs.

2.2.4 Modern Classification Algorithms for Graphs

Graph neural networks (GNNs) extend deep learning approaches for graph data. As Wu et al. [22] describe, the irregular nature of graphs makes some operations difficult. Convolutions for example, can be calculated in many ways. The basic idea of taking neighborhood information of each node is similar to convolution in the image domain. This approach results in message passing algorithms that spread information across the graph. The first category of GNNs that use message passing are *recurrent graph neural networks* (RecGNNs). They assume a constant exchange of informatioin between nodes in a graph until a stable equilibrium is reached. RecGNNs inspired *convolutional graph neural networks* (ConvGNNs). The main idea behind ConvGNNs is to aggregate a node's features with the features of its neighbors. The difference is that ConvGNNs stack multiple graph convolutional layers, resulting in a high-level node representation. They can be used to build many more complex GNN models (e.g. for node classification or graph classification). Other categories investigated by Wu et al. [22] include *graph autoencoders* and *spacial-temporal graph neural networks*. They are not relevant for this thesis and are hence not explained further.

Graph classification with GNNs is based on a combination of graph convolutional layers, graph pooling layers, and readout layers. The graph convolutional layers extract node representations that are then downsampled by a graph pooling layer. The readout layer yields a graph representation that can then be sent into a multilayer perceptron. Finally, the classification is obtained by adding a softmax layer to the end of the network.

3 The Classification Systems

This chapter discusses the implementation of the systems that are being tested. That includes the system which serves as a starting point for this thesis. Said system, as well as other approaches to GED estimation, will be described in detail in Section 3.1. Section 3.2 presents the new solution for classification based on different implementations of SVMs.

3.1 K-NN-Classification with Graph Neural Networks

The k-NN-Classification developed by Dobler and Riesen [7] serves as a starting point for this thesis. The general procedure can be summed up briefly. In a first step, a GNN is trained for the direct classification. Then, from the last layer of the network, node embeddings are extracted. This results in node embeddings, which can help with the estimation of the GED. Finally, that estimation of the GED is used as a distance metric for the k-NN-Classification, as described in Section 2.2.3.

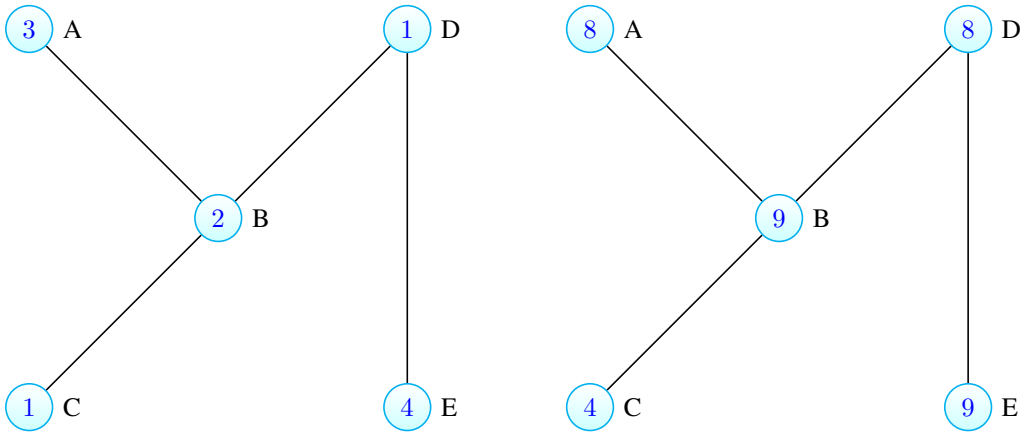
The GNN used in the first solution by Dobler and Riesen [7] is a *graph isomorphism network* (GIN). Because the other GNNs used in this thesis are conceptually similar, the GIN will be explained in some more detail before presenting the crucial differences. This explanation is based on a lecture from Riesen [20]. The GIN follows the description in Section 2.2.4. For each node $v \in V$, the initial embedding $h_v^{(0)}$ is just the node's label x_v. Then for each node, the node embedding is computed in K steps, with the following formula for a node $v \in V$ at step k:

$$h_v^{(k)} = g^{(k)} \left(\sum_{u \in N(v)} h_u^{(k-1)} + (1 + \epsilon^{(k)}) \cdot h_v^{(k-1)} \right) \tag{3.1}$$

where $g^{(k)}$ is the aggregation function, $N(v)$ is the set of neighbors of v, and $\epsilon \in \mathbb{R}$ is a parameter. Here, " $\cdot$ " refers to scalar vector multiplication. With this process, each node gradually receives information about its environment. In the first step, this information is limited to the node's neighbors. In every step, a node indirectly receives information about all nodes that are two steps away from it. Thus information is passed along the graph, hence the name message passing algorithm. This model has good

scaling capabilities, because the aggregation function $g^{(k)}$ and the parameter ϵ are shared between all nodes.

Figure 3.1 shows an example of one step of the GIN algorithm. The numbers represent the actual labels that are relevant to the algorithm. The letters are there to referr to the nodes in the explanation. In this simplified example, all nodes only have one label. Because the operation can be done in the same way for each label, this illustration suffices.



(a) Starting graph at step $k = 0$

(b) The same graph after step $k = 1$

Figure 3.1: One step of the GIN algorithm applied to a graph.

Since the calculations are the same for each node, they will be demonstrated on node D. Using the *rectified linear unit* (ReLU) as an activation function and $\epsilon = 1$ yields the following calculation:

$$\begin{aligned} h_1(D) &= ReLU(h_0(B) + h_0(E) + (1+\epsilon) \cdot h_0(D)) \\ h_1(D) &= ReLU(2 + 4 + (1+1) \cdot 1) \\ h_1(D) &= ReLU(8) \\ h_1(D) &= 8 \end{aligned} \tag{3.2}$$

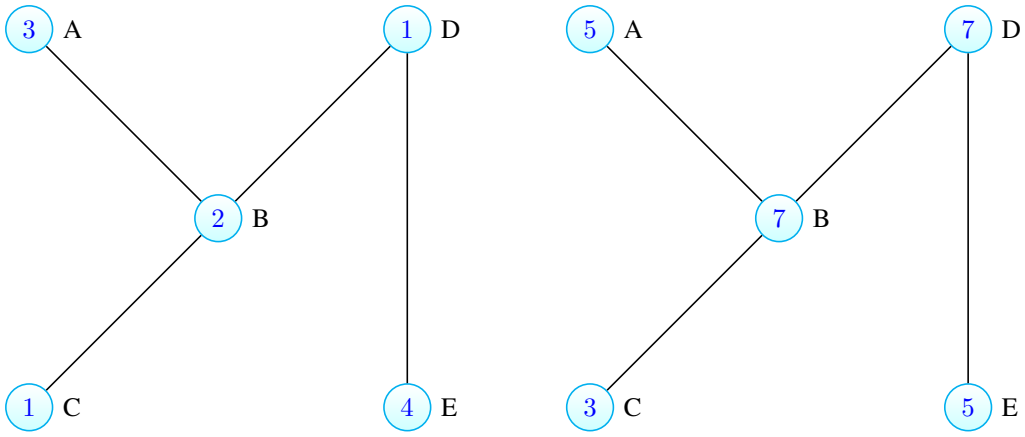
When applying the same calculation for each node, one obtains the graph in Figure 3.1b. Each layer in the network applies one of these operations, before further layers do the classification. With this setup of g and ϵ, the labels of the nodes will only grow with each step. This is not an issue, because only a limited amount of layers are used. For this thesis in particular, three layers are being used. Using too many layers would also not help, because nodes with higher degrees will grow more quickliy. This would mean that the labels approach a relative distribution which is similar to just the degrees of nodes.

The first goal of this thesis is to obtain comparable solutions to the one developed by Dobler and Riesen [7]. For that, two other GNNs are implemented, namely a *graph attention network* (GAT) and a *graph convolutional network* (GCN). Because they work in a very similar way as GIN, only the different formulas for the message passing calculations, as well as an example graph are shown. The formulas are based on lecture notes from Riesen [20]. For the GAT, the formula is:

$$h_v^{(k)} = g^{(k)} \left(W^{(k)} \left[\sum_{u \in N(v)} \alpha_{vu}^{(k-1)} h_u^{(k-1)} + \alpha_{vv}^{(k-1)} h_v^{(k-1)} \right] \right) \tag{3.3}$$

where $W^{(k)}$ is the shared weight matrix and the attention weights α are calculated between two nodes by some attention mechanism.

Figure 3.2 shows an example of what the weights could look like after one layer has been applied. For this, the starting weights are chosen as $W^{(1)} = I$, where I refers to the identity matrix, and $\alpha_{uv}^{(1)} = 1$ for all nodes u, v. Because there is only one label in this example, the matrix $W^{(k)}$ has only one dimension and is equal to 1. The ReLU function is again used as the activation function $g^{(1)}$.



(a) Starting graph at step $k = 0$

(b) The same graph after step $k = 1$

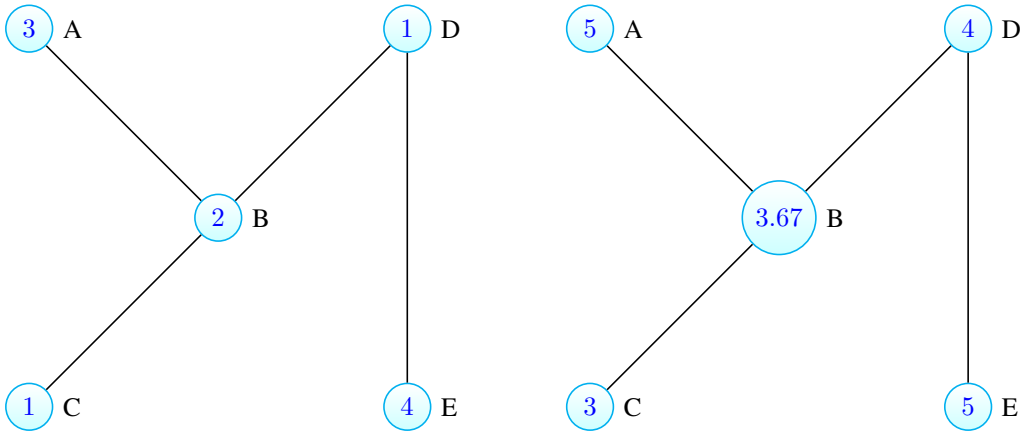
Figure 3.2: One step of the GAT algorithm applied to a graph.

The GAT introduces a way to weigh the inpact of each node to another node separately, making the process very customizable, while sacrificing some scalability. The GCN uses another approach that normalizes the weights from a node's neighbors based on the node's degree. The formula for the GCN looks as follows:

$$h_v^{(k)} = g^{(k)} \left(W^{(k)} \cdot \frac{\sum_{u \in N(v)} h_u^{(k-1)}}{deg(v)} + B^{(k)} \cdot h_v^{(k-1)} \right) \tag{3.4}$$

where $W^{(k)}$ and $B^{(k)}$ are weight matrices and $deg(v)$ is the degree of the node v. In this context, " $\cdot$ " refers to matrix multiplication. Similarily to GIN, GCN scales also well because the weight matrices are shared between all nodes.

Figure 3.3 shows an example of one step of the GCN applied to the example graph. The weights for the initial step are all set to one, with ReLU as the activation function.



(a) Starting graph at step $k = 0$

(b) The same graph after step $k = 1$

Figure 3.3: One step of the GCN algorithm applied to a graph.

It is immediately appearant that the nodes with higher degrees do not grow as quickly as in the other examples. In the case of GAT, this could be adapted through a different choice of parameters.

The calculation of the GED estimation works in the same way as described in Section 2.2.2. The only difference lies in the exact numbers, based on which the nodes are assigned. A first evaluation of the different GNNs can be made simply based on the GNN-GED estimation compared to the BP-GED, which is used as a reference. Using this evaluation, the GNN-GED estimations can also be compared between each other.

Figure 3.4 shows an example of a plot that presents the comparison between GNN-GED estimations and BP-GED estimations. The 45°-axis represents the estimation that stems from the BP-GED. The four points each represent the GNN-GED-based estimation of a particular graph. The graph is also normalized, such that the largest GED corresponds to one. The point on the bottom left represents the case where both systems estimated an edit distance of zero, while the point on the top right represents the case where both systems estimated the maximum edit distance. The point on the top left represents an overestimation of the edit distance by the GNN-based system. To be precise, the GNN-GED estimated an edit distance equal to 80% of the maximum distance, while the BP-GED estimated 20% of the maximum distance. For the point on the bottom right it is the other way around, meaning that the GNN-GED estimated 20% of the maximum distance, while the BP-GED estimated 80% of the maximum distance.

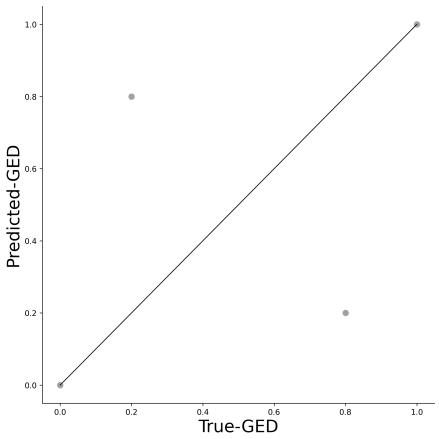


Figure 3.4: Example of a GNN-GED estimation relative to the BP-GED estimation.

The estimation performance of a system can be evaluated by how close the points are to the 45°-axis. It is still important to remember that the BP-GED is also an estimation. In fact, it gives an upper bound to the GED. Hence, the actual GED would be scattered below the 45°-axis.

3.2 SVM Classification

As already mentioned in Section 2.2.3, SVM-Classification requires a precomputed kernel matrix to work well. The approach taken for this work is to calculate this similarity measure directly on the node embeddings. For that, the node embeddings are obtained from the last layer of the trained GNNs, yielding three different starting points for the kernel calculation. This means that the following calculations need to be tested for each of the GNN designs. Given two graphs G_1 and G_2, the cosine distances between all pairs of nodes $n_i \in G_1$ and $n_j \in G_2$ are calculated with the same formula as in equation 2.14. The result of this procedure is a $(|G_1| \times |G_2|)$-matrix with all the similarities between the graphs' nodes. This matrix is then used to obtain a new matching M, by applying the linear sum assignment problem described in Section 2.2.2. As with the other systems, this matching represents an estimation of the optimal matching for the calculation of the GED. While the GED is not required for this solution, the matching still contains the necessary information of the graphs' similarities. Instead of calculating the GED, the *cosine similarity* between two graphs G_1 and G_2 can be defined as follows:

$$cosine\_similarity(G_1, G_2) = \sum_{(u,v)\in M} (1 - d(u, v)) \tag{3.5}$$

where $d(u, v)$ is the distance in equation 2.14. Calculating this similarity for each pair of graphs yields the new kernel matrix, that can be used for the SVM-Classification. Analyzing the information that is lost with this procedure presents possible weaknesses of the solution.

Figure 3.5 shows a simplified example of this with two-dimensional node embeddings. In the table, the node embeddings correspond to the nodes that are matched between the graphs. One could expect the similarity of a graph to itself to be a maximum value that depends on the context. In the example, the graph G_1 compared to itself has a cosine similarity of 2.0. But the same graph compared to the larger graph G_2 reaches the same similarity. Even more counterintuitively, comparing G_2 to the graph G_3 yields a larger similarity than comparing G_1 to itself.

Node Embedding	G_1	G_1	G_1	G_2	G_2	G_3
n_0	(2, 0)	(2, 0)	(2, 0)	(2, 0)	(2, 0)	(2, 0)
n_1	(1, 0)	(1, 0)	(1, 0)	(1, 0)	(1, 0)	(1, 0)
n_2	(2, 0)	(2, 0)	(2, 0)	(2, 0)	(2, 0)	(2, 0)
n_3	-	-	-	(9, 1)	(9, 1)	(0, 1)
n_4	-	-	-	(9, 1)	(9, 1)	(0, 1)
Cosine Similarity	3.0		3.0		3.22	

Figure 3.5: Example of cosine similarities between graphs.

A similar problem occurs if the two larger graphs are also very similar. In general, two small graphs that are very similar, may have a low similarity compared to two large graphs that are less similar, simply because more similarities are added up. One approach to normalization is to divide the cosine similarity by the size of the matching. Some experiments have shown that this approach leads to very bad performance, because too much information gets lost. With that method a graph and its subgraph would still have the same similarity as the subgraph compared with itself, if matched correctly. Instead, the cosine similarity is divided by the size of the larger graph, giving the following formula:

$$normalized\_cosine\_similarity(G_1, G_2) = \frac{\sum_{(u,v)\in M} (1 - d(u, v))}{\|G_2\|} \tag{3.6}$$

where $\|G_2\|$ is the number of nodes in the graph G_2 and $\|G_2\| \geq \|G_1\|$. That way, the information about the different graph sizes is conserved. Applying this similarity measure to the example graphs gives the results in Figure 3.6.

Node Embedding	G_1	G_1	G_1	G_2	G_2	G_3
n_0	(2, 0)	(2, 0)	(2, 0)	(2, 0)	(2, 0)	(2, 0)
n_1	(1, 0)	(1, 0)	(1, 0)	(1, 0)	(1, 0)	(1, 0)
n_2	(2, 0)	(2, 0)	(2, 0)	(2, 0)	(2, 0)	(2, 0)
n_3	-	-	-	(9, 1)	(9, 1)	(0, 1)
n_4	-	-	-	(9, 1)	(9, 1)	(0, 1)
Cosine Similarity	1.0		0.6		0.64	

Figure 3.6: Example of normalized cosine similarities between graphs.

There are also other similarity measures that could be used instead of the cosine similarity. To investigate the impact of the similarity measure, the cosine similarity can be compared to the *correlation*. The implementation follows the same steps as explained above, except that for each node pair, one calculates the correlation distance given by:

$$\bar{d}(u, v) = 1 - \frac{(u - \bar{u}) \cdot (v - \bar{v})}{\|(u - \bar{u})\|_2 \, \|(v - \bar{v})\|_2} \tag{3.7}$$

where $\bar{u}$ and $\bar{v}$ are the respective means of the elements in u and v, and " $\cdot$ " represents the dot product. After obtaining a new matching M by solving the linear sum assignment problem with the new distances, the correlation between graphs G_1 and G_2 is calculated as:

$$correlation(G_1, G_2) = \sum_{(u,v)\in M} \big(1 - \bar{d}(u, v)\big) \tag{3.8}$$

This yields another kernel matrix which can be used for the SVM-Classification. Applying this similarity measure without normalization to the example from before gives the results in Figure 3.7

Node Embedding	G_1	G_1	G_1	G_2	G_2	G_3
n_0	(2, 0)	(2, 0)	(2, 0)	(2, 0)	(2, 0)	(2, 0)
n_1	(1, 0)	(1, 0)	(1, 0)	(1, 0)	(1, 0)	(1, 0)
n_2	(2, 0)	(2, 0)	(2, 0)	(2, 0)	(2, 0)	(2, 0)
n_3	-	-	-	(9, 1)	(9, 1)	(0, 1)
n_4	-	-	-	(9, 1)	(9, 1)	(0, 1)
Cosine Similarity	3.0		3.0		1.0	

Figure 3.7: Example of correlations between graphs.

This result already looks different from the one in Figure 3.5. Here the similarity between G_2 and G_3 is not automatically larger because the graphs have more nodes. Multiple nodes with similar embeddings though, would still lead to larger similarities. Therefore, a normalized version is still worth investigating. Similar to the normalized cosine similarity, one defines the normalized correlation as:

$$normalized\_correlation(G_1, G_2) = \frac{\sum_{(u,v)\in M} \left(1 - \bar{d}(u,v)\right)}{\|G_2\|} \tag{3.9}$$

With this calculation for the example graphs, one obtains the results shown in Figure 3.8, this time yielding the smallest similarity for the graphs with the same size but different embeddings for two nodes.

Node Embedding	G_1	G_1	G_1	G_2	G_2	G_3
n_0	(2, 0)	(2, 0)	(2, 0)	(2, 0)	(2, 0)	(2, 0)
n_1	(1, 0)	(1, 0)	(1, 0)	(1, 0)	(1, 0)	(1, 0)
n_2	(2, 0)	(2, 0)	(2, 0)	(2, 0)	(2, 0)	(2, 0)
n_3	-	-	-	(9, 1)	(9, 1)	(0, 1)
n_4	-	-	-	(9, 1)	(9, 1)	(0, 1)
Cosine Similarity	1.0		0.6		0.2	

Figure 3.8: Example of normalized correlations between graphs.

Deciding which similarity is the most reasonable to use in a given situation, depends on the context of the data. With the abstraction of embeddings, which stem from GNNs, it is hard to know in advance which measure will lead to the best performance. Therefore, it can be useful to test multiple similarity measures for each new dataset. While there are many ways to calculate such similarity measures, investigating four measures provides enough basic understanding of the effects that these measures have. This also includes information about the effects of normalization, as will be shown in Chapter 4. With the kernel matrix, the goal that was set in Section 2.2.3 is achieved. The SVM-Classification can therefore be conducted as described in Section 2.1.2. To obtain the best performance, the C-parameter in the SVM-Classification is optimized using cross validation. The results from this cross validation will be shown in Chapter 4.

To summarize, there are 14 new systems that can be compared to the k-NN-Classifications based on the BP-GED [19], as well as the GIN-GED [7]. While two of these new systems are also based on k-NN-Classification, 12 systems give a deeper insight into the performance of SVM-Classification.

4 Experimental Results

The first section in this chapter introduces the datasets used for this thesis and provides some reasoning behind the choices. In the second section, this chapter also provides an insight into the estimation process and the results it achieved. Section 4.3 focuses on the C-Optimization that is used to tune the SVM for maximum performance. Some essential graphs of the C-Optimization are presented there, but including all 120 graphs would add little value. That sectioin does not discuss the optimization of the k-NN based classifiers, because that process is already described in Section 2.1.2. Section 4.4 then shows the obtained accuracy of the tested solutions, evaluated by the F1-score, before comparing some examples of correctly and incorrectly classified graphs.

4.1 Datasets

Five datasets from the TUDataset library [14] are investigated to evaluate the performance of the developed solutions and compare them to the existing systems. Both the reference BP-GED [19] and the GIN-GED [7] are evaluated based on the k-NN classifier developed by Dobler and Riesen [7]. The calculation of the BP-GED is based on an implementation by Gillioz and Riesen [11]. The *MUTAG* dataset serves used as a good starting point, given that the reference solutions work fairly well with some room for improvement. As a second dataset *OHSU* is investigated, to see if any solution could result in large performance increases, because the reference systems leave more room for improvement. The *AIDS* dataset can help because the reference systems perform very well. With the evaluation of the performace on this dataset, poorly performing solution attempts can easily be discarted. The *ENZYMES* dataset is labeled based on six classes instead of two, thus enabeling investigations of the generalization potential the developped solutions have compared to the reference systems. Finally, the investigtions conducted by Dobler and Riesen [7] into the *DD* dataset suggest a very hard task for GNN-based classification. To give a deeper insight into these datasets, they are described below.

The *MUTAG* dataset, taken from Debnath et al. [6], contains 188 graphs sorted into two classes. The average number of nodes per graph is 17.93, while the average number of edges is 19.79. This is a relatively small dataset that is not too complex. The dataset is from the domain of drug development and investigates the toxicity of certain compounds. It is used by Debnath et al. [6] to analyze the structure-activity

relationship of mutagenic aromatic and heteroaromatic nitro compounds. For this dataset, a graph's nodes represent the atoms in the compounds and the edges represent the covalent bonds.

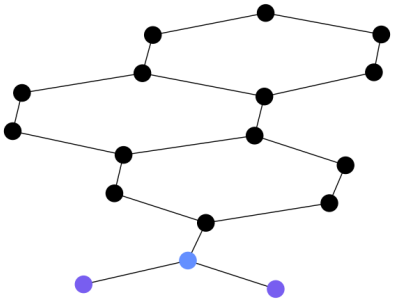


Figure 4.1: Representative graph form the *MUTAG* dataset.

The *OHSU* dataset, based on brain mappings by Craddock et al. [17] and investigated by Pan et al. [15], contains 79 graphs sorted into two classes. The average number of nodes per graph is 82.01, while the average number of edges is 199.66. This is still a relatively small dataset, but it contains much more complex graphs. The contained graphs are connecting different regions of interest in the brain. The dataset labels correspond to the hyper/impulsive behavior of the tested individuals. The graph's nodes correspond to the different regions of interest in the brain, while the edges represent their correlation.

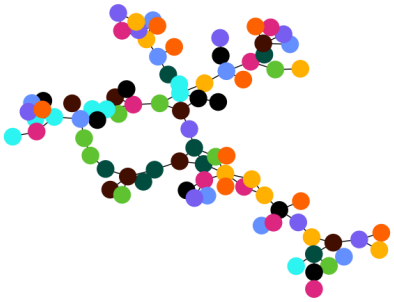


Figure 4.2: Representative graph form the *OHSU* dataset.

The *AIDS* dataset, aggregated by Riesen and Bunke [18] based on the AIDS Antiviral Screen Database of Active Compounds [9], contains 2000 graphs sorted into two classes. The average number of nodes per graph is 15.69, while the average number of edges is 16.20. This dataset is large and has fairly simple graphs. It containes graphs that represent molecules that are either active or inactive against HIV. The molecules are represented as graphs in the same way as for the *MUTAG* dataset.

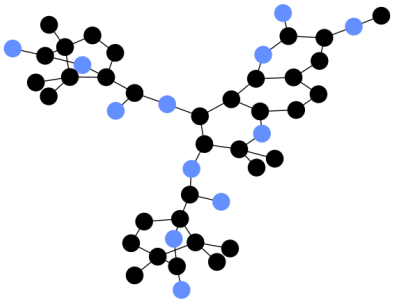


Figure 4.3: Representative graph form the *AIDS* dataset.

The *ENZYMES* dataset is part of the BRENDA database by Schomburg et al. [21] and contains 600 graphs sorted into six classes. The average number of nodes per graph is 32.63, while the average number of edges is 62.14. While this dataset is medium in size and complexity, it is the only dataset with six classes instead of two that is being tested. The graphs represent different enzymes that are labeled based on the reaction catalysed by the enzyme. Here nodes represent amino acids instead of single atoms and the edges represent their connections.

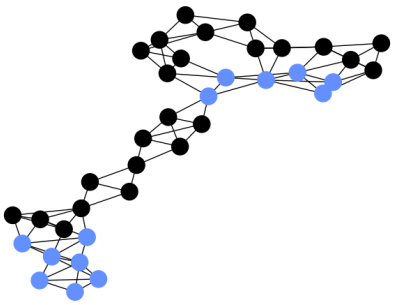


Figure 4.4: Representative graph form the *ENZYMES* dataset.

The *DD* dataset, compiled by Dobson and Doig [8], contains 1178 graphs sorted into two classes. The average number of nodes per graph is 284.32, while the average number of edges is 715.66. This dataset contains the most complex graphs out of the datasets investigated. The graphs represent proteins that are labeled as enzyme structures or non-enzymes. The representation of nodes and edges is similar to the representation in the *ENZYMES* dataset.

Figure 4.5: Representative graph form the *DD* dataset.

4.2 GED Estimation Results

Figures 4.6 - 4.10 show the GNN-based estimations of the GED compared to the BP-GED-based estimations. The format of the plots is explained in Section 3.1. Each figure presents the results for one dataset, while in each figure the different GNN architectures are compared.

In Figure 4.6 there are some large differences between the GNN-based estimations and the estimations from the BP-GED system. Despite the differences, there is a clear correlation between both estimations for all GNN architectures. It is hard to find any meaningful differences between the GNNs. Only the bottom section in the middle seems to have fewer points in the GAT-based design, indicating that there are fewer underestimations with that architecture.

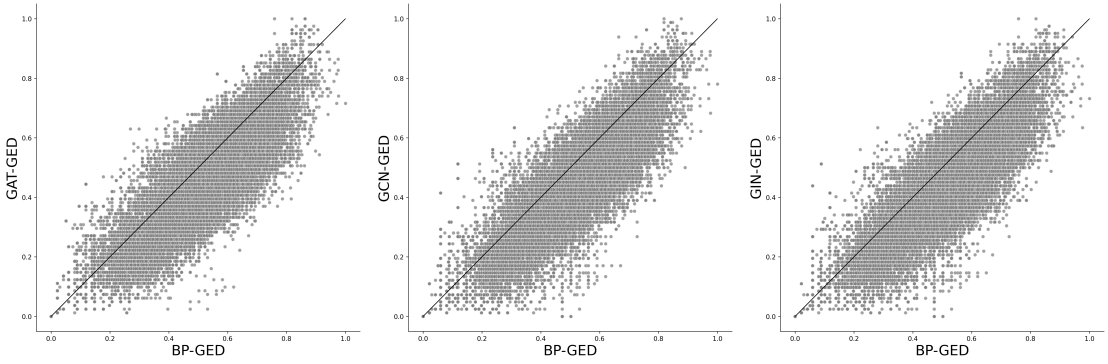


Figure 4.6: The GNN-GED estimates relative to the BP-GED estimate for the MUTAG dataset.

Figure 4.7 shows the same comparison for the *OHSU* dataset. Here the pattern looks a bit different compared to Figure 4.6. For small distances, there are much smaller discrepancies between the GNN estimation and the reference system. Again, there are barely any differences between the GNN architectures. For the *AIDS* and *ENZYMES* datasets, represented in figures 4.8 and 4.9, the errors appear to be smaller than those of the *MUTAG* dataset. They are also more evenly distributed than the errors of the *OHSU* dataset.

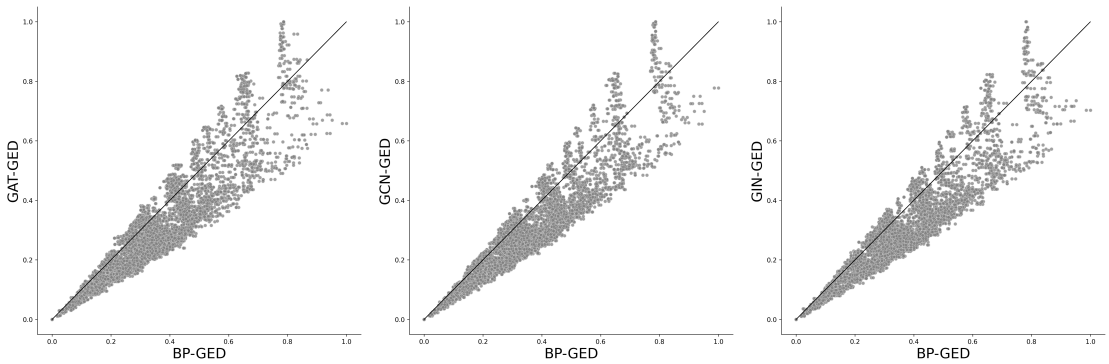


Figure 4.7: The GNN-GED estimates relative to the BP-GED estimate for the OHSU dataset.

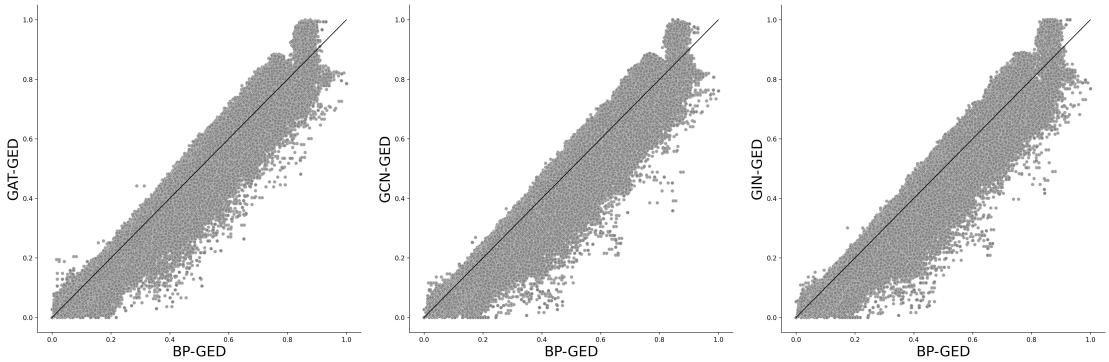


Figure 4.8: The GNN-GED estimates relative to the BP-GED estimate for the AIDS dataset.

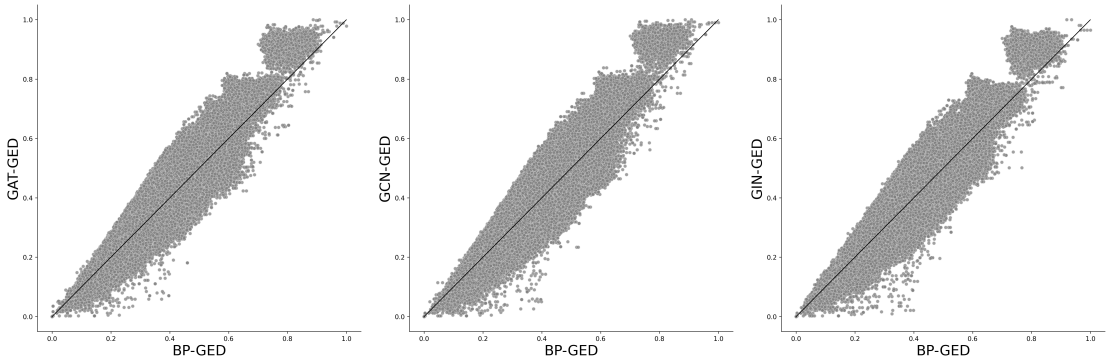


Figure 4.9: The GNN-GED estimates relative to the BP-GED estimate for the ENZYMES dataset.

The smallest differences between the GNN estimations and the reference system appeared in Figure 4.10. A clear division into clusters can be observed, which has to do with the nature of the dataset. While finding the best GNN architecture is still not trivial, the GIN-based estimation marginally stands out. Its estimated distances are not necessarily closer to the BP-GED-based estimations, but there is less variance

in each cluster.

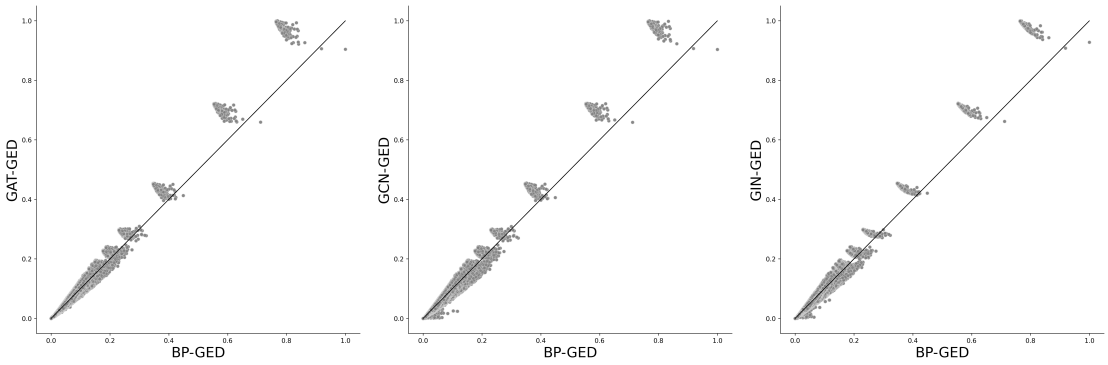


Figure 4.10: The GNN-GED estimates relative to the BP-GED estimate for the DD dataset.

One summarizing result is that, for a given dataset, all GNN-GED estimations are very similar to each other. That fact also holds for every investigated dataset. Although this suggests very similar performance for the classification, there are some outliers in the resulting performances which are hard to explain. The results are also promising in the sense that there is a clear correlation between the GNN-GED estimations and the BP-GED estimation. One might expect a smaller spread from the 45°-axis to result in better performance for classification. Section 4.4 will show how that is not necessarily the case. In fact, drawing conclusions about performance, based on figures 4.6 - 4.10, proves to be difficult.

4.3 Validation of Metaparameters

To start the C-Optimization, one first needs to define a performance measure. For this thesis, the *F1-score* [1] is a reasonable approach. The formula for the F1-score is given by:

$$F1 - score = \frac{(2 \cdot TP)}{(2 \cdot TP) + FP + FN} \tag{4.1}$$

where TP is the number of true positives, FP is the number of false positives, and FN is the number of false negatives. Since this calculation only makes sense for datasets with two classes, another approach needs to be taken for multiclass datasets. The simple solution is to calculate TP, FP, and FN globally. This is done by comparing each class separately to all other classes.

The parameter C is the only parameter that needed to be tuned for the SVM-Classification. This parameter was already discussed in Section 2.1.2 and is only restricted by $C > 0$. As briefly mentioned in Section 2.1.2, C is first optimized with $C = 10^k$ for $k \in \{-10, -9, ..., 0, 1, ..., 9, 10\}$. After that, a finer C-Optimization is conducted for $C = kC^*$ for $k \in \{0.5, 0.6, ..., 1.4, 1.5\}$ and with C^* as the optimal value from the coarse C-Optimization. This finer optimization did not have a big impact due to random fluctuations specific to the training set. For the *AIDS* dataset, all of the graphs, independent of similarity measure or GNN architecture, appear similar for the coarse optimization. For the fine optimization, the results look random within a small band of difference in performance. Figure 4.11 shows what the graphs look like for the example of the normalized correlation with the GCN architecture.

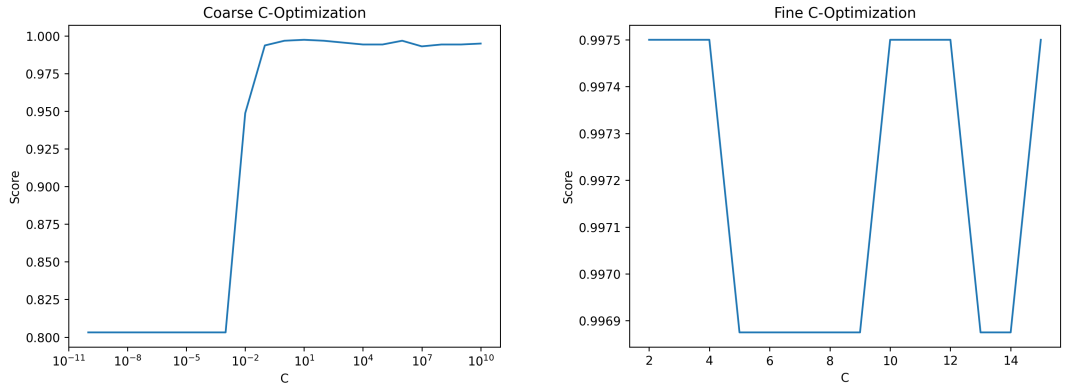
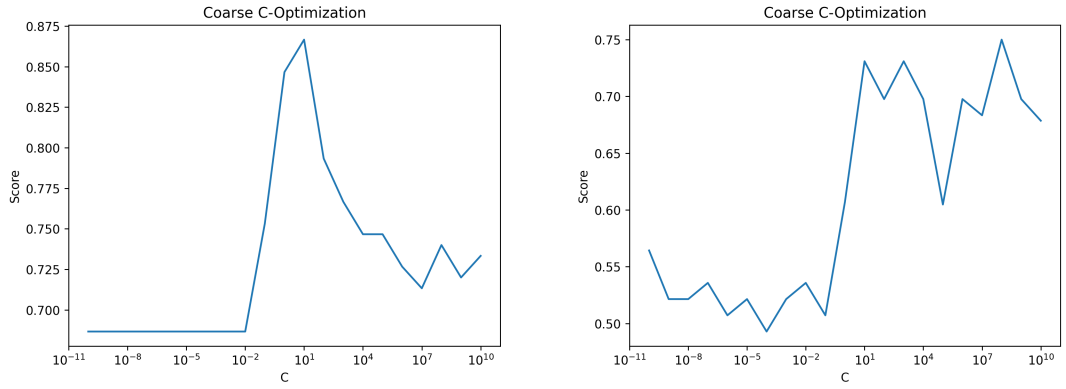


Figure 4.11: The coarse and fine C-Optimization for the *AIDS* dataset using the normalized correlation based on the GIN architecture. The score axis represents the F1-score.

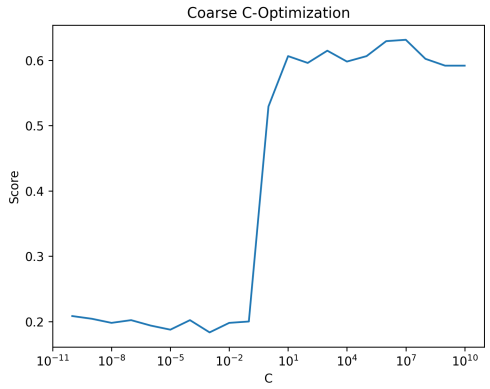
For the other datasets, there are more differences between the implementations. Still, the fine optimization remains mostly random, and will therefore not be shown. Figures 4.12 and 4.13 show the optimizations of the SVM-Classification with the GIN-based normalized correlation for the other datasets. Note that for the *DD* dataset, the scaling of the C-axis is different. This is because for very small values of C, the SVM does not converge. The main takeaway out of figures 4.11 - 4.13 is that the performance is relatively low for small values of C, before increasing at a certain threshold. There are also examples where the F1-score decreases more after a peak in the middle. Another interesting observation is a correlation between the size of the dataset and the random noise in the graph. As will be discussed in the next section, this correlation does not imply a correlation between the size of a dataset and the classification performance.



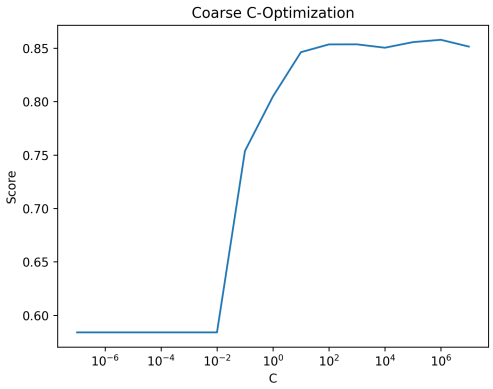
(a) C-Optimization with GIN-based normalized correlation for the *MUTAG* dataset.

(b) C-Optimization with GIN-based normalized correlation for the *OHSU* dataset.

Figure 4.12: The score axis represents the F1-score.



(a) C-Optimization with GIN-based normalized correlation for the *ENZYMES* dataset.



(b) C-Optimization with GIN-based normalized correlation for the *DD* dataset.

Figure 4.13: The score axis represents the F1-score.

4.4 Classification Results

Figure 4.14 shows the F1-scores of all the relevant experiments. The results for the SVM-Classification with the negative GED, as well as other normalization attempts, are omitted because they are less accurate than the best shown solutions. The former design also defeats one purpose of using SVM-Classification, which is to be independant of the GED calculation. The results shown are all based on the same random train-test-split of the datasets. Although the best results for all datasets are from the SVM-Classification, a consistent improvement compared to the reference system cannot be observed across all versions of the solution. For the *MUTAG* dataset, only one approach outperformes the reference system. Still, the scores of the SVM-Classification is on average very similar to the k-NN-Classification, which indicates reasonable performance. With the *OHSU* dataset one should be careful, because there can be a high variance in performance due to its small size. The average performance though, is once again similar to the reference system. There is not much to say about the *AIDS* dataset, since it appears to be easy for all solutions. One interesting result is that only the *ENZYMES* dataset shows much better results for the GIN-based architectures compared to the other SVM-based solutions. The GIN-based arcitecture's performance is more similar to the implementations that used the k-NN classifier, while the other SVM-based solutions perform about as well, or worse than the reference system. The only dataset for which there is a consistent improvement is the *DD* dataset. Not only do all approaches outperform the reference system, but all the SVM-based solutions outperform k-NN-based solutions .

Classification System	Embedding	MUTAG	OHSU	AIDS	ENZYMES	DD
Reference System (k-NN)	BP-GED	90.9	63.2	98.9	29.2	57.4
GNN-GED with k-NN-Classification	GAT	85.7	63.6	99.2	42.5	67.8
	GCN	85.7	63.6	99.5	41.7	67.8
	GIN	80.0	81.5	99.4	44.2	71.6
Cosine Similarity with SVM-Classification	GAT	82.6	**85.7**	99.7	27.5	72.7
	GCN	80.0	42.1	**99.8**	28.3	74.4
	GIN	87.5	57.1	**99.8**	38.3	72.0
Normalized Cosine Similarity with SVM-Classification	GAT	82.6	57.1	**99.8**	29.2	73.7
	GCN	89.1	63.6	**99.8**	31.7	72.0
	GIN	82.6	66.7	**99.8**	49.2	75.4
Correlation with SVM-Classification	GAT	82.6	**85.7**	98.0	21.7	73.1
	GCN	89.8	42.1	**99.8**	27.5	73.0
	GIN	**91.7**	57.1	99.5	42.5	**76.2**
Normalized Correlation with SVM-Classification	GAT	82.6	63.6	99.2	25.0	73.1
	GCN	82.6	69.6	**99.8**	28.3	71.8
	GIN	83.7	66.7	99.7	**53.3**	75.4

Figure 4.14: F1-scores(%) based on random train-test splits.

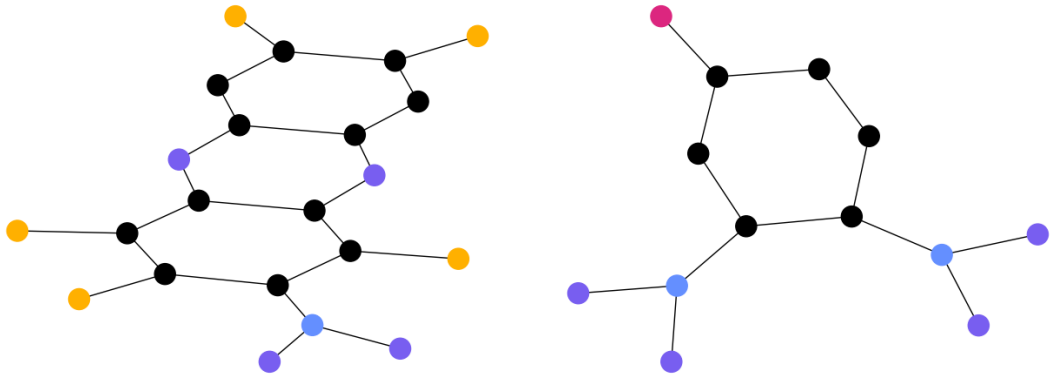
Having established that there is no single best solution that outperforms the reference system for all datasets, it also makes sense to compare the different approaches on average. Figure 4.15 shows the average performance of the SVM-Classification across all datasets.

	k-NN	Cosine	Normalized Cosine	Correlation	Normalized Correlation	Total SVM
GIN	75.3	70.9	74.7	73.4	**75.8**	73.7
GAT	71.6	73.6	68.5	72.2	68.7	70.8
GCN	71.7	64.9	69.2	66.4	70.4	67.8
Total	72.9	69.8	70.8	70.7	71.6	70.8

Figure 4.15: Average F1-scores(%) over all datasets.

As can be seen in Figure 4.15, the GIN architecture outperformes the others by a fair margin. This holds for most SVM-based classifications, as well as for the k-NN-based classification. A slightly smaller margin can be observed with the different similarity measures for the SVM-Classification. The best performance overall is obtained by the normalized correlation based on the GIN architecture. Compared to the average performance of the reference system, which is 67.9%, the best architecture shows a relative improvement of 11.6%. The fact that the k-NN-Classification, based on the GIN architecture, is 75.3% suggests that there is no significant difference to the best SVM-Classification with a relative performance increase of only 0.664%.

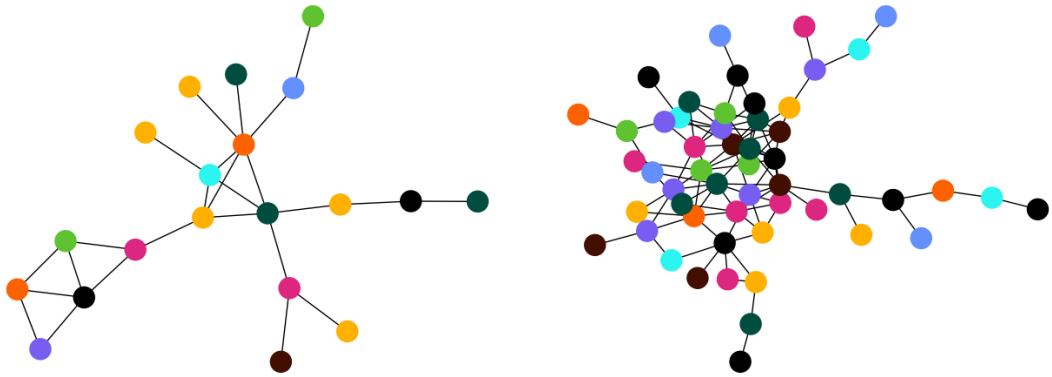
Figures 4.16 - 4.20 show some examples of correct and wrong classifications for each dataset. The examples are based on the SVM-Classification with the kernel matrix from the normalized correlation with the embeddings from the GIN. For the *MUTAG* dataset, there is no clear increase in complexity that would explain a wrong classification.



(a) Correct SVM-Classification (b) Wrong SVM-Classification

Figure 4.16: Example of classified graphs from the *MUTAG* testset.

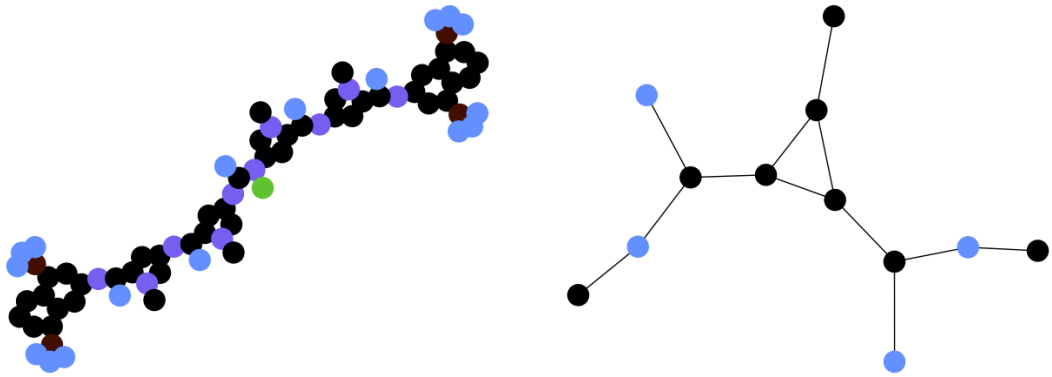
In the *OHSU* dataset, there are clear differences in complexity between the graphs in Figure 4.17. Given the information from the *MUTAG* dataset, it is not clear if that is the reason for the wrong classification. This could also suggest that there is an inherent difficulty in classifying graphs from this dataset.



(a) Correct SVM-Classification (b) Wrong SVM-Classification

Figure 4.17: Example of classified graphs from the *OHSU* testset.

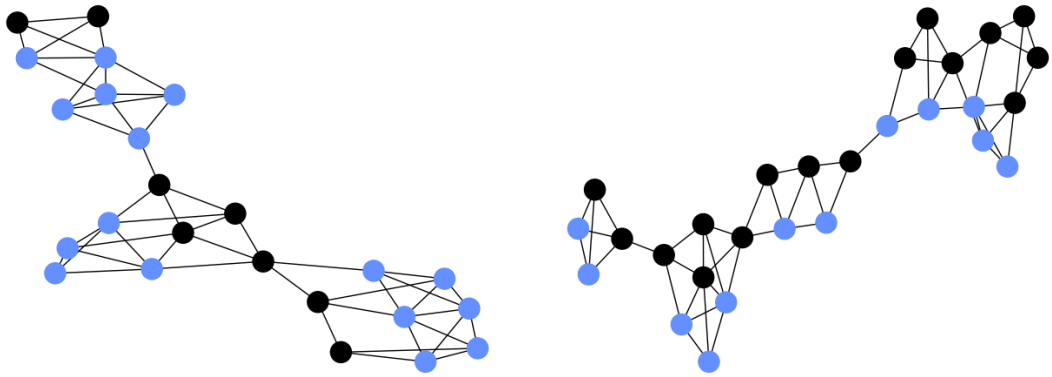
For the *AIDS* dataset in Figure 4.18, it looks like the correctly classified graph is more complex. There is still rotational symmetry in both cases though, that comes from the nature of the dataset. This might suggest that larger graphs with more information are sometimes easier to classify.



(a) Correct SVM-Classification (b) Wrong SVM-Classification

Figure 4.18: Example of classified graphs from the *AIDS* testset.

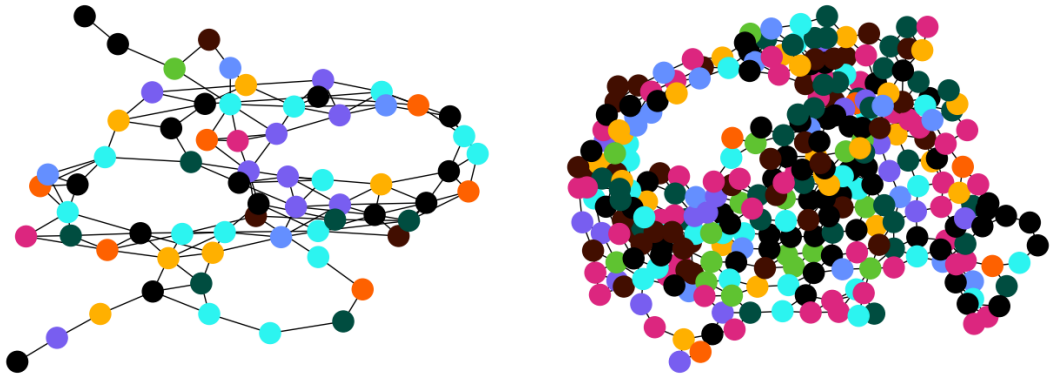
For the *ENZYMES* dataset, the graphs look similar in complexity. Jet the F1-score this system achieves is only 53.3%. Some of the difficulty in classification stems from the fact that there are six classes for this dataset. Taking that into account, the F1-score is still a lot better than a guess.



(a) Correct SVM-Classification (b) Wrong SVM-Classification

Figure 4.19: Example of classified graphs from the *ENZYMES* testset.

For the DD dataset in Figure 4.20, there is a similar increase in complexity as observed in figure 4.17. The increase in performance compared to the *OHSU* dataset may come from the nature of the dataset. The *DD* dataset could have fewer *complex* graphs. To test this one would need an more objective measure of complexity.



(a) Correct SVM-Classification (b) Wrong SVM-Classification

Figure 4.20: Example of classified graphs from the *DD* testset.

The real reasons for misclassifications would have to be analyzed with more expert knowledge in the field from which each dataset stems. Although improvements compared to the reference system were made in many cases, limitations were encountered for some datasets. Whether these limitations are due to the systems used, or due to the nature of the datasets remains to be discovered.

5 Conclusion and Future Work

5.1 Conclusion

The goal of this thesis was to expand on the system built by Dobler and Riesen [7]. Therefore, the reasearch question was to find oher classification systems that were also based on GNNs. The first step in obtaining a solution was to investigate the impact of the GNN designs, namely GAT and GCN compared to GIN. This involved building a GAT and a GCN, and finding a way to evaluate their effectiveness. In part, this was done by comparing the GED estimations to the BP-GED. Given the promising results in the comparison with GIN, the implemented systems were compared based on classification performance. To get a direct comparison, the classification was performed with k-NN-Classification and the same GED estimation algorithm as was used by Dobler and Riesen [7]. Since all GNN designs resulted in solutions performing with similar F1-scores as the reference system, all designs were kept for further investigations.

The second step in obtaining a solution was to investigate other machine learning algorithms. Specifically, the hope was to achieve better results with SVM-Classification. After a proof of concept with the negative GED as a kernel matrix, a better solution had to be found. This required a more direct translation of the graph embeddings into a kernel matrix. It was interesting to investigate what impact on performance the similarity measure had, even though there were no big differences that were consistent across all datasets. One of the challanges was to deal with the different sizes of graphs without loosing too much information from the embeddings. Normalizing the similarity measures resulted in minor improvements when averaging over all datasets and GNN designs. Still, these improvements were not consistent across different GNN designs. For the classification into two classes, the best performing system for each dataset was always a version without normalization, while the same is true for the worst performing system in three out of four cases.

With some datasets, the results show high variability despite using seemingly very similar classification methods. Therefore it is hard to draw a meaningful conclusion. The performance for the *AIDS* dataset suggests that all designs work as intended. Still, there is no single system that outperformes all the others. The averages presented in Figure 4.15, obtained by the SVM-Classification, give some insight. As intuition suggests when looking at Figure 4.14, the GIN implementation performes the best on average across all

similarity measures. The different SVM implementations are very close when averaging over all GNN designs. The best average performance is obtained by the normalized correlation, with both normalized versions outperforming the others on average. In the end, any of the GIN-based SVM-Classifications outperform the reference system for four out of five datasets. The normalized versions also outperform the GIN-based k-NN-Classification for four out of five datasets. Hence, it is apparent that the this method works well for classification, even though the improvements were only marginal.

5.2 Future Work

The solutions presented in this thesis show that good approaches can be made with different designs. Jet, for some datasets, obtaining a reliable solution was not possible. Tuning the system even more by changing the GNN designs based on the final classification would certainly be an option. A better approach would probably be to define a different GNN that is more specialized in giving fitting embeddings for SVM-Classification.

What this thesis does not discuss is the computation time of the different systems. The main reason for this is that comparing one value per design for each dataset already results in the evaluation of tradeoffs. Including time in the evaluation could possibly result in more uncertainty of which design to persue. Also, because the embedding data is only written to once, all algorithms are highly parallelizable, which makes temporal performance gains less significant. Still, a thorough analysis of the computation times of all systems could be interesting. In the unlikely event that a certain solution was much quicker than all the others, one would have a best solution, that could serve as a future reference point.

Another way to find out more about the presented solutions would be to test them on more datasets. This could lead to some insight into why certain datasets are harder to classify. Focusing on large datasets could be useful to avoid big fluctiuations in the F1-scores. As shown in Section 4.3, larger datasets are much more consistend for different C-values. This can also be extended to different classification algorithms. Having established that GIN-based architectures give the best average result across five datasets, using only GIN-based implementations of the different machine learning approaches would be a reasonable choice. Once a better classification algorithm is found, the design of the GNN could be revisited again.

Finally, given that there are so many working solutions, one could implement a majority voting algorithm, that combines all systems. Some accuracy gain could be expected of this approach. Even though a lot of computations could be shared between the systems, the increase in computational cost could be too large.

Bibliography

[1] scikit-learn f1_score. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html. Accessed: 2025-03-11.

[2] scikit-learn support vector machines. https://scikit-learn.org/stable/modules/svm.html. Accessed: 2025-03-25.

[3] Bernhard E. Boser, Isabelle Guyon, and Vladimir Vapnik. A training algorithm for optimal margin classifiers. In David Haussler, editor, *Proceedings of the Fifth Annual ACM Conference on Computational Learning Theory, COLT 1992, Pittsburgh, PA, USA, July 27-29, 1992*, pages 144–152. ACM, 1992.

[4] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27, 2011.

[5] Estefane G. M. de Lacerda, André Carlos Ponce de Leon Ferreira de Carvalho, and Teresa Bernarda Ludermir. A study of crossvalidation and bootstrap as objective functions for genetic algorithms. In Teresa Bernarda Ludermir and Marcílio Carlos Pereira de Souto, editors, *7th Brazilian Symposium on Neural Networks (SBRN 2002), 11-14 November 2002, Recife, Brazil*, pages 118–123. IEEE Computer Society, 2002.

[6] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34(2):786–797, 1991.

[7] Kalvin Dobler and Kaspar Riesen. Learning graph matching with graph neural networks. In Ching Yee Suen, Adam Krzyzak, Mirco Ravanelli, Edmondo Trentin, Cem Subakan, and Nicola Nobile, editors, *Artificial Neural Networks in Pattern Recognition - 11th IAPR TC3 Workshop, ANNPR 2024, Montreal, QC, Canada, October 10-12, 2024, Proceedings*, volume 15154 of *Lecture Notes in Computer Science*, pages 3–12. Springer, 2024.

[8] P. D Dobson and A. J. Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology*, 330(4):771 – 783, 2003.

[9] DTP. Aids antiviral screen. 2004.

[10] Thomas Gärtner. A survey of kernels for structured data. *SIGKDD Explor.*, 5(1):49–58, 2003.

[11] Anthony Gillioz and Kaspar Riesen. Improving graph classification by means of linear combinations of reduced graphs. In Maria De Marsico, Gabriella Sanniti di Baja, and Ana L. N. Fred, editors, *Proceedings of the 11th International Conference on Pattern Recognition Applications and Methods, ICPRAM 2022, Online Streaming, February 3-5, 2022*, pages 17–23. SCITEPRESS, 2022.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[13] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, pages 1137–1145. Morgan Kaufmann, 1995.

[14] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020.

[15] Shirui Pan, Jia Wu, Xingquan Zhu, Guodong Long, and Chengqi Zhang. Task sensitive feature exploration and learning for multitask graph classification. *IEEE Trans. Cybern.*, 47(3):744–758, 2017.

[16] Edward A. Patrick and Frederic P. Fischer III. A generalized k-nearest neighbor rule. *Inf. Control.*, 16(2):128–152, 1970.

[17] Craddock RC, James GA, Holtzheimer PE 3rd, and Mayberg HS Hu XP. A whole brain fmri atlas generated via spatially constrained spectral clustering. In *Human Brain Mapping Vol. 33*, pages 1914–1928, 2012.

[18] K. Riesen and H. Bunke. Iam graph database repository for graph based pattern recognition and machine learning. In *Structural, Syntactic, and Statistical Pattern Recognition: Joint IAPR International Workshop*, pages 287–297, 2008.

[19] Kaspar Riesen and Horst Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image Vis. Comput.*, 27(7):950–959, 2009.

[20] Kasper Riesen. Graph-based pattern recognition, lecture notes. 2024.

[21] I Schomburg, A. Chang, C. Ebeling, M. Gremse, C. Heldt, G. Huhn, and D. Schomburg. Brenda, the enzyme database: updates and major new developments. *Nucleic Acids Research*, 32(Database-Issue):431–433, 2004.

[22] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Networks Learn. Syst.*, 32(1):4–24, 2021.