



University of Stuttgart
Germany

Faculty 5: Computer Science, Electrical Engineering
and Information Technology



Department of
Analytic Computing

Learning Quantitative Argumentation Frameworks Using Sparse Neural Networks and Swarm Intelligence Algorithms

Bachelor's Thesis

in partial fulfillment of the requirements for
the degree of Bachelor of Science (B.Sc.)
in Informatik

submitted by
Mohamad Wahed Bazo

First supervisor: Prof. Dr. Steffen Staab
Department of Analytic Computing

Second supervisor: Dr. Nico Potyka
Department of Analytic Computing

Stuttgart, September 2021

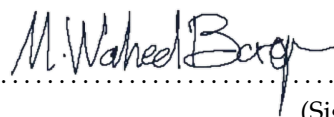
Statement

I hereby certify that this thesis has been composed by me and is based on my own work, that I did not use any further resources than specified – in particular no references unmentioned in the reference section – and that I did not submit this thesis to another examination before. The paper submission is identical to the submitted electronic version.

	Yes	No
I agree to have this thesis published in the library.	<input type="checkbox"/>	<input type="checkbox"/>
I agree to have this thesis published on the Web.	<input type="checkbox"/>	<input type="checkbox"/>
The thesis text is available under a Creative Commons License (CC BY-SA 4.0).	<input type="checkbox"/>	<input type="checkbox"/>
The source code is available under a GNU General Public License (GPLv3).	<input type="checkbox"/>	<input type="checkbox"/>
The collected data is available under a Creative Commons License (CC BY-SA 4.0).	<input type="checkbox"/>	<input type="checkbox"/>

Stuttgart, den 30.09.2021

.....
(Place, Date)



.....
(Signature)

Note

- If you would like us to contact you for the graduation ceremony,
please provide your personal E-mail address:
- If you would like us to send you an invite to join the AC Alumni
and Members group on LinkedIn, please provide your LinkedIn ID :

Zusammenfassung

Argumentation-Frameworks repräsentieren einen Ansatz der Formalisierung von Argumenten und derer Beziehungen in einer Graphenstruktur. Sie dienen dazu, Schlussfolgerungen aus dieser Wissensmodellierung zu ziehen. Da Argumentation ein wichtiger Bestandteil der menschlichen Intelligenz ist, kann man diese Technologie als eine interessante Methode der *erklärbaren künstlichen Intelligenz* betrachten. In dieser Bachelorarbeit werden Argumentation-Frameworks erstellt, um Klassifikationsprobleme zu lösen, indem man sie aus *Multilayer-Perceptrons* (MLPs) erstellt. Die argumentationale Klassifizierer werden implementiert, getestet, und evaluiert auf drei unterschiedlichen Datensätzen. Da wir für ihre Interpretierbarkeit interessiert sind, verwenden wir *dünnbesetzte MLPs* in der Implementierung. Das trägt dazu bei, dass die Graphen simplere Strukturen haben. Das Problem des Strukturlernens der dünnbesetzten MLPs wird als ein Teilmengensuchproblem modelliert und mit einem Schwarmintelligenz-Algorithmus, nämlich der *Partikelschwarmoptimierung* (PSO), behandelt. Die Ergebnisse der Studie zeigen, dass Modelle der Argumentation Frameworks gute Klassifikationsleistung mit minimalen Strukturen erbringen können. Außerdem hat der PSO-Algorithmus seine Effizienz bei der Lösung des Problems gezeigt.

Abstract

Argumentation Frameworks are an approach of formalizing arguments and their interrelations in a graph structure. They can be used to draw conclusions from this modelling of knowledge. Since argumentation is an important part of human reasoning, these graph structures can be considered easily interpretable, what makes this technology an interesting explainable artificial intelligence method. Although this is not their main purpose, *Quantitative Argumentation Frameworks* can be used to solve classification problems by following a new approach. This approach is based on constructing them out of *multilayer perceptrons* (MLP), based on the work of Potyka.

In this thesis we were motivated to construct Quantitative Argumentation Frameworks out of sparse MLPs. A *swarm intelligence* algorithm, namely *Particle Swarm Optimization* (PSO), was developed to search for *sparse MLP* models with specific characteristics that relate to performance and topology of the graphical structures. Models were implemented, tested, and evaluated on three different datasets.

The implementation includes preprocessing of the datasets, parameter learning of MLPs based on backpropagation, and structure learning of the MLP graphical structures. The evaluation involves constructing fully connected MLPs and decision trees for comparison purposes.

The resulting models achieved high performance and low complexity in their structure. The PSO algorithm also proved its efficiency in solving the structure learning

problem. Additionally, the study showed that this technology can be considered an important explainable machine learning approach in terms of performance and interpretability, based on comparing it to other machine learning technologies.

Keywords: Argumentation Frameworks, Sparse Neural Networks, Swarm Intelligence, and Particle Swarm Optimization.

Contents

1. Introduction	1
2. Background	3
2.1. Argumentation Frameworks and QBAFs	3
2.1.1. Abstract Argumentation Frameworks and Bipolar Argumentation Graphs	3
2.1.2. Quantitative Bipolar Argumentation Frameworks (QBAFs)	4
2.2. MLPs and MLP-based Semantics for QBAFs	4
2.3. Discrete Binary PSO for Subset Problems	7
2.3.1. Swarm Intelligence	7
2.3.2. Particle Swarm Optimization Algorithm	7
2.3.3. Discrete Binary PSO for Subset Problems	9
3. Related Work	10
3.1. Argumentation Frameworks and Machine Learning	10
3.2. Sparse MLPs and MLP Structure learning	10
4. Methodology	11
4.1. BAGs as classifiers	11
4.2. Dataset Preprocessing	13
4.3. Parameter Learning	14
4.4. Structure Learning	16
4.4.1. Particles and Search Space	16
4.4.2. Objective Function	17
4.4.3. Initialization	17
4.4.4. Stopping Condition and Dynamic Execution	18
4.4.5. Fine-Tuning	19
4.4.6. Time Efficiency	20
5. Experiments and Results	22
5.1. The Iris Dataset	22
5.2. The Mushroom Dataset	24
5.3. The Adult Income Dataset	26
6. Conclusion	28
7. Limitations and Future Work	29
A. Appendix	32
A.1. The Iris Dataset	32
A.2. The Mushroom Dataset	35
A.3. The Census Income Dataset	41
A.4. General Results	44

List of Figures

1.	Example of a BAG of a decision problem presented in [20]	3
2.	Illustration of the strength update procedure [20]	5
3.	Layered graph structure of an MLP (left) and illustration of the local feed-forward mechanism at a node in an MLP (right) [20]	5
4.	High-level Architecture of Argumentation Classifiers as illustrated in [18]	12
5.	Illustration of transforming a binary connection matrix of a sparse MLP layer into a vector, in order to construct the search space position vector.	17
6.	An example graph of a sparse MLP model (interpreted as a BAG) for the iris dataset. The model achieved an accuracy of 100% on the test set. The model was found using the PSO configuration in table 5, and it is the model with the best accuracy score.	33
7.	Graph of a sparse MLP model (interpreted as a BAG) for the iris dataset that was found using the PSO configuration in table 5. This model had the highest sparsity and achieved an accuracy of 93.33% on the test set.	33
8.	Example of a decision tree for the iris dataset with a depth of three (random seed 42). The decision tree reached an accuracy of 95.83% on the training set and 100% on the test set.	34
9.	A plot of the development of the particles' personal best scores. The personal best score is plotted (vertical axis) for each particle in the population (horizontal axis) after each phase. This plot was taken from one of the runs of the PSO algorithm with configurations in table 8. One can notice the effect of the high value of the social coefficient in the third phase, where particles exploit the area of the global best position.	36
10.	A BAG of the mushroom dataset. This model achieved the highest sparsity among the ten models that were found with configuration 8. It achieved a test accuracy of 98.33%. Noticeable is the dependency on only three values of the feature "oder".	37
11.	Graph of a sparse MLP model (interpreted as a BAG) for the mushroom dataset that was found using the PSO algorithm with configurations in table 8. This model combined high sparsity and performance (98.52% accuracy on the test subset).	37
12.	A plot of distribution of classes in the values of the feature "oder" in the mushroom dataset. The count plot illustrates the importance of the value "n" where a relatively big portion of the samples is concentrated with a preponderance of one class "edible".	38

13.	A violin plot of the mushroom dataset after transforming the categorical features into ordinal ones. The plot illustrates the distribution of the two class labels over the features. Here is also noticeable how the "oder" feature peaks at a certain value.	38
14.	Example of a decision tree for the mushroom dataset with a depth of three (random seed 42). The decision tree reached an accuracy of 98.6% on the training set and 98.21% on the test set. This illustration with this particular depth is chosen due to its comparable simplicity to the found sparse MLP models.	40
15.	Graph of a sparse MLP model (interpreted as a BAG) for the income dataset that was found using the PSO algorithm with configurations in table 11. The model achieved a test accuracy of 82.67%.	42
16.	A personal best score plot of a particular run of the PSO algorithm on the income dataset with configurations 11.	43
17.	Summery of performance scores of BAG classifiers averaged over ten runs, including standard deviation, from the genetic algorithm implementation in [24]. This table was taken directly from the paper. Note that the performance values in the table are in percent, where performance values in this study are between zero and one.	44

1. Introduction

Argumentational reasoning constitutes an essential part of human intelligence [4]. We form our thoughts and opinions in arguments and engage in argumentation in almost every aspect of human interaction. We express these arguments, sometimes attempt to defend or attack them by using other arguments. Interestingly, we can perceive this study as an argumentational reasoning work. We can consider each statement in it as an argument that is a subject of correctness, including this very one.

Studying the nature and mechanism of argumentation has drawn many researchers' interest in different fields, such as philosophy, logic, and artificial intelligence [4].

Argumentation frameworks as introduced by Dung [4] represent an approach for formalizing argumentation. His work was focused on constructing graph structures out of arguments and their interrelations in order to have a computational framework that mimics the process of drawing conclusions and determining acceptable arguments in an argumentational problem. In an argumentation graph, arguments are represented by nodes and the relations are represented by edges. In his original work, the relations are considered *attack* relations. Dung claims that the principle of argumentational reasoning is that an argument is believable (accepted) if it is able to successfully stand against its attacking arguments [4]. However, the relations can be extended to include *support* relations (*bipolar*) [18]. Frameworks with this extension are called *bipolar*. The intuitiveness of this technology and its closeness to human thinking makes its models easy to interpret and understand. This makes it an exciting method of explainable artificial intelligence that is worthy of exploring.

In a more recent work, Potyka explored the relationship between argumentation frameworks and neural networks [20]. His work introduced an approach to interpret *multilayer perceptrons* (MLPs) as *quantitative argumentation frameworks*. In a quantitative argumentation framework, the acceptance of an argument is numerically measured. This interpretation opened the door to constructing argumentation frameworks out of MLPs, which is one of the goals of this thesis. In this thesis, argumentation frameworks are constructed, tested, and evaluated for classification purposes and out of sparse MLPs. The sparsity of MLPs is crucial here, since we are interested in simple structures that can yield good interpretability. The argumentational classifiers are deployed on three different datasets. Classification models of other types, namely decision trees and fully connected MLPs are also built to establish a comparison.

The work on this thesis was done parallel to the study in [24]. The authors also aimed to explore argumentational classifiers following the approach that Potyka in [20] proposed. However, structure learning of sparse MLPs was approached by using a genetic algorithm. Therefore, one of the goals of this thesis is to present an additional implementation based on the deployment of a swarm intelligence

algorithm, namely the *particle swarm optimization algorithm* (PSO). The problem of finding sparse models with specific properties (structure learning) is considered a subset search problem, and a version of the PSO algorithm for discrete spaces was deployed for this purpose. This different method can deliver us a comparison of the performance of the two algorithms in approaching the problem.

Finally, the Combination of the two topics of argumentation, which is a technology inspired by human intelligence, and swarm intelligence, which is an algorithm family inspired by nature and complex systems, was found very intriguing and was personally the main incitement behind the work on this thesis.

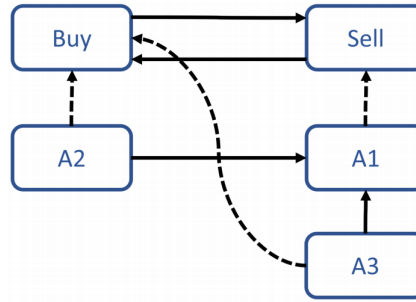


Figure 1: Example of a BAG of a decision problem presented in [20]

2. Background

2.1. Argumentation Frameworks and QBAFs

This chapter presents background on argumentation frameworks and their variants that were used in this thesis.

2.1.1. Abstract Argumentation Frameworks and Bipolar Argumentation Graphs

Abstract argumentation frameworks are an approach for representing arguments and the relations among them in a graph structure. Arguments are modelled as abstract entities. These entities are represented as nodes in the graph, while the relations among the arguments are modelled as edges among the entities in the graph [18].

This thesis considers only *bipolar argumentation frameworks*, where a relation between two entities can be either a *support* or an *attack* relation. In an argumentation framework, arguments are quantified by their acceptability, where the acceptability of an argument depends on the acceptability of its attackers and supporters.

The resulting graph is called a *bipolar argumentation graph (BAG)*. In a BAG, attack and support relations are usually denoted by solid and dashed edges, respectively. In figure 1, we see an example of a BAG that models a part of a decision problem that is modelled as an argumentation framework. This example was presented in [18]. The interest here is to determine whether to buy or sell stocks in a company. A1, A2, and A3 represent statements from experts that could relate to each other or the decision arguments, e.g. to buy or to sell.

Our focus in this study is on a variant of these frameworks, namely the quantitative bipolar argumentation frameworks, which we introduce in the next chapter 2.1.2.

2.1.2. Quantitative Bipolar Argumentation Frameworks (QBAFs)

Definition 1: Quantitative bipolar argumentation frameworks (QBAFs). A QBAF over the interval ($\mathcal{D} = [0, 1]$) is a quadruple $(\mathcal{A}, Att, Sup, \beta)$ where \mathcal{A} is a set of arguments, Att and Sup are two sets of binary relations that are called attack and support relations, and β is a function that assigns a *base score* to each argument $a \in \mathcal{A}$ [20].

In quantitative argumentation frameworks, acceptability is a numerical value. The semantics of a QBAF is defined by using *interpretations*. An interpretation represents a function that assigns a strength value to each argument of the framework [20]. Formally defined:

Definition 2: QBAF interpretation. Let Q be a $QBAF = (\mathcal{A}, Att, Sup, \beta)$ with \mathcal{A} is over $\mathcal{D} = [0, 1]$. An interpretation of Q is a function $\sigma : \mathcal{A} \rightarrow [0, 1] \cup \{\perp\}$. For an argument $a \in \mathcal{A}$, we call $\sigma(a)$ the strength of a . The strength of an argument a is called *partially defined* if $\sigma(a) = \perp$. Otherwise, it is called *fully defined* [20].

Usually, interpretations are defined in iterative procedures based on updating the strength values iteratively. The Semantics defined in this way are called *modular semantics*. In [20], a modular semantics procedure is described as follows: Initially, the strength of each argument is initialized with its base score. Then, we do the following in each update iteration: For each argument, its strength value is updated in two steps. In the first step, the strength values of the attackers and supporters of the argument are aggregated by an *aggregation function* α to one single value called *the aggregate*. Aggregation functions can be based on different types of functions, such as addition, multiplication, and the maximum function [20]. In the second step, an *influence function* ι is applied to the resulting aggregate. The influence function adapts the strength value with consideration of the base score of the argument [20]. This update procedure is illustrated in figure 2 .

With this procedure, we anticipate the convergence of the strength values after some iterations. However, convergence is not always guaranteed, as shown in [16], in the case of cyclic graphs. In this thesis, we are interested in another type of semantics, namely the MLP-based semantics, and convergence is not discussed further, since it is guaranteed for acyclic graphs and all BAGs that we implement are acyclic. In the next chapter, we introduce the relationship between multilayer perceptrons (MLPs) and QBAFs, and the MLP-based semantics of QBAFs. This topic represents the base of constructing BAG classifiers in this thesis.

2.2. MLPs and MLP-based Semantics for QBAFs

Multilayer perceptrons (MLPs) are a popular class of feed-forward neural networks that are usually used for classification and regression problems. MLPs have a layered graph structure with three or more layers [20]. An MLP consists of an input

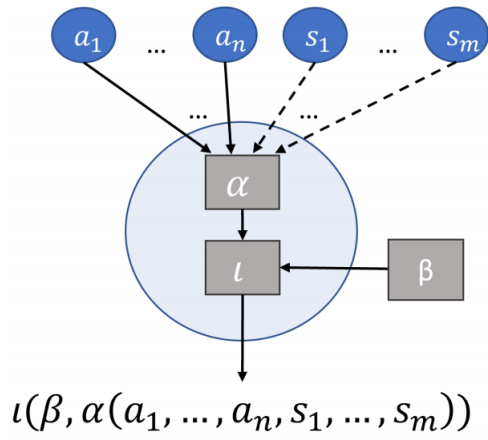


Figure 2: Illustration of the strength update procedure [20]

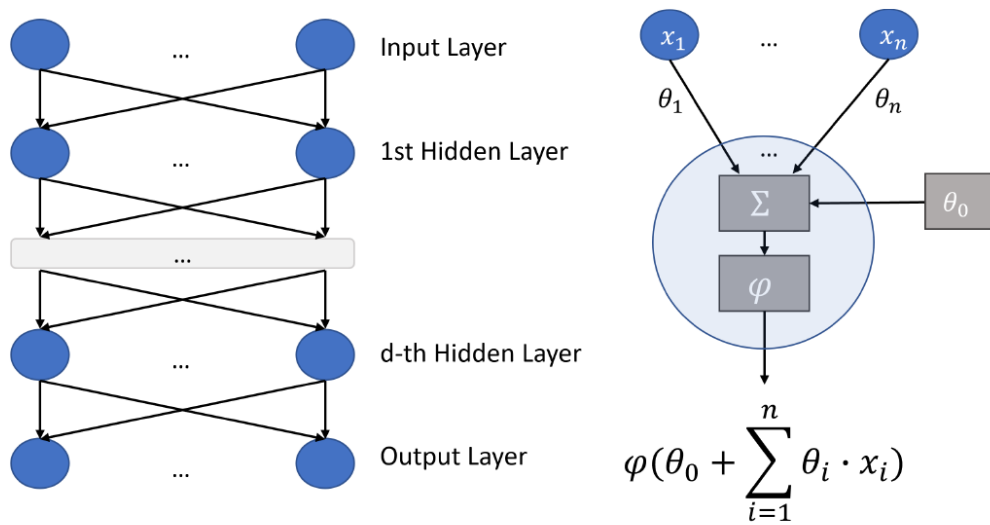


Figure 3: Layered graph structure of an MLP (left) and illustration of the local feed-forward mechanism at a node in an MLP (right) [20]

layer, an output layer, and one or more hidden layers. Each layer consists of nodes or *neurons* that are (fully) connected to the nodes of the previous and the next layer. Nodes are associated with activation values. These values at the hidden nodes and the output nodes are computed by feed-forward-propagating the values of the the input nodes, which are initialized with input feature values, through the network [20]. Figure 3 illustrates the structure and the feed-forward propagation mechanism of MLP.

Each node in an MLP is optionally associated with a *bias* θ_0 and the incoming edges are associated weights $(\theta_1, \dots, \theta_n)$. In a local feed-forward propagation at some node, each of the weights θ_i is multiplied by the value of its source node x_i , and the resulting values are summed up. The sum is then added to the bias of the node θ_0 . Afterwards, the result is fed to an activation function ϕ . There are various types of activation functions. However, we only use the logistic function $\varphi(z) = \frac{1}{1+\exp(-z)}$ for this purpose due to its property of mapping its input to the interval $[0, 1]$, which yields a simple interpretation of the activation values [20].

In [20], Potyka discussed the high similarity between the local update mechanics in QBAFs and the local feed-forward propagation mechanics in MLPs. One can visually identify this similarity by comparing figure 2 and 3 (right). The author also mentioned that we can view an MLP as a QBAF, of which the aggregation function α is based on addition, and the influence function ι is based on the activation function of the MLP. For simplifying the connection between MLPs and QBAFs, we introduce the edge-weighted QBAF as in [16]:

Definition 3: Edge-weighted QBAF. An Edge-weighted QBAF (over $\mathcal{D} = [0, 1]$) is a quadruple $QBAF = (\mathcal{A}, E, \beta, \omega)$ where \mathcal{A} is a set of arguments, E is a set of binary edges between arguments of \mathcal{A} , $\beta : \mathcal{A} \rightarrow [0, 1]$ is a function that assigns a *base score* $\beta(a)$ to each argument $a \in \mathcal{A}$, and $\omega : E \rightarrow \mathbb{R}$ is a function that assigns a weight to each edge in E [20].

Edge-weighted QBAFs have only one set of edges. We consider edges with negative weights as the attack relations and those with positive weights as the support relations. In order to interpret the arguments in an edge-weighted QBAF, we use a modular semantics based on the relationship between QBAFs and MLPs that we introduced earlier. Similar to the modular semantics that are presented in chapter 2.1.2, the strength values are computed in an iterative approach. For each argument a , we let the base score $\beta(a)$ be the initial strength value. The strength values are then updated by repeating the two following steps for all $a \in \mathcal{A}$:

- **Aggregation:** $\alpha_a^{(i+1)} := \sum_{(b,a) \in E} w(b,a) \cdot s_b^{(i)}$
- **Influence:** $s_a^{(i+1)} := \varphi_l \left(\ln \left(\frac{\beta(a)}{1-\beta(a)} \right) + \alpha_a^{(i+1)} \right)$

where $\alpha_a^{(i)}$ and $s_a^{(i)}$ are respectively the aggregate and the strength value of argument a in the i -th iteration, and $\varphi_l(z) = \frac{1}{1+\exp(-z)}$ is the logistic function [20].

Based on this introduction, we established the connection between MLPs as QBAFs and can now conceive MLPs as QBAFs.

2.3. Discrete Binary PSO for Subset Problems

2.3.1. Swarm Intelligence

Swarm Intelligence is a discipline for designing and implementing computational methods that are aimed to solve problems in a way that is inspired by the behaviour of biological swarms and colonies. These biological entities represent systems with complex collective behaviour that emerges from specific features, such as self-organization and local or indirect communication [15].

The early stages of swarm intelligence research included attempts from scientists and biologists to understand and simulate complex social behaviours of natural swarms, such as bird flocks and fish schools [9]. For example, researchers were intrigued by the organized movement of bird flocks. They tried to find the underlying rules that enabled large numbers of birds to flock synchronously, like suddenly changing direction, scattering, and regrouping. The simulations suggested that the mechanism behind these phenomena was the birds' attempt to maintain an optimum distance between themselves and their neighbours [9].

Swarm intelligence depends on several principles: The problem-solving ability should emerge in the interactions of simple information-processing units that compose the swarm. The concept of swarm proposes multiplicity and stochasticity. These simple information-processing units represent an abstract modelling of simulation units or problem variables, i.e., human beings, birds, bees, or array elements [8]. The interaction among the units can have different characteristics, but it is an essential principle. This flexibility of rules gives us a diversity of paradigms and approaches. Therefore there exists numerous algorithms and methods that specify as swarm intelligence methods, such as *Ant Colony Optimization* and *Particle Swarm Optimization* [8].

In the following chapters, we present the particle swarm optimization algorithm and elaborate its deployment in this study.

2.3.2. Particle Swarm Optimization Algorithm

The *particle swarm optimization* algorithm (PSO) was introduced by Kennedy and Eberhart as an optimization method for non-linear functions [10]. A series of studies on bird flocking and fish schooling inspired them to develop the algorithm [15]. PSO is also considered a *metaheuristic*. Metaheuristics are algorithmic frameworks that are usually nature-inspired and designed to solve complex optimization problems. Metaheuristics generate or search for heuristics in order to find a sufficiently good solution depending on a balance between exploration and exploitation of the

search space [1]. PSO has been found a powerful method for solving problems with aspects of non-linearity, non-differentiability, multiple optima, and high dimensionality [11].

According to [15], the optimization problem is initially modelled in a D-dimensional search space. The algorithm maintains a set of m particles that represent points in the search space, each with a coordinates vector $x_i = (x_{i1}, \dots, x_{iD})$. The algorithm maintains also for each particle a *velocity* vector $v_i = (v_{i1}, \dots, v_{iD})$ that represents the rate of position change in each dimension. The position of a particle at some point is evaluated (score) with an *objective function* f that has to be defined. Each particle $i \in 1, \dots, m$ is initialized with a random position in the search space. A Particle x_i keeps and updates a record of the position of its personal so-far-best found position in a vector p_i and the associated score. The algorithm also keeps and updates the overall (global) so-far-best found position and its associated score. In each iteration of a PSO algorithm, the particles evaluate their scores with the evaluation function f and update there personal best positions p_i , i.e., if $f(x_i) > f(p_i)$ then assign $p_i := x_i$. Afterwards, the algorithm updates the global best position and score, i.e., if $f(p_i) > f(p_g)$ then assign $p_g := p_i$. Then, the velocity of each vector is updated for each dimension $d \in 1, \dots, D$. The newly calculated velocity v_{id} depends on three factors: the previous velocity v_i , the current position of the particle x_i , the current personal best position of the particle p_i , and the current global best position p_g . v_{id} is calculated as follows:

$$v_{id} := w \cdot v_{id} + c_1 \cdot r_1 \cdot (p_{id} - x_{id}) + c_2 \cdot r_2 \cdot (p_{gd} - x_{id}) \quad (1)$$

where:

- w is a parameter that is called the *inertia weight*. It expresses the influence of the previous velocity in a particular direction. The higher the value of w , the more the particles tend to search in new areas. Usually, w takes values that are slightly smaller than 1.
- c_1 and c_2 are acceleration coefficients. c_1 is called the cognitive parameter. It controls the influence of the personal best position. c_2 is called the social parameter. It determines the influence of the global best position. The higher these coefficients are, the more the particles tend to search near their personal best positions and the global best position, respectively. [15] states that these two parameters usually take the value of 2.
- r_1 and r_2 are random numbers that are drawn uniformly from the interval $[0, 1]$.

To calculate the new position of the particle, its newly calculated velocity is added to its current position as a rate of change:

$$x_{id} := x_{id} + v_{id} \quad (2)$$

Algorithm 1 Particle Swarm Optimization [15]

```
1: for particle  $i \in 1, \dots, m$  do:
2:   initialize  $x_i$  and  $v_i$ 
3: end for
4: repeat
5:   for particle  $i \in 1, \dots, m$  do:
6:     evaluate  $f(x_i)$ 
7:     update personal best position
8:   end for
9:   update the global best position
10:  for particle  $i \in 1, \dots, m$  do:
11:    update velocity  $v_i$ 
12:    update the new position  $x_i$ 
13:  end for
14: until Stopping condition is reached
```

2.3.3. Discrete Binary PSO for Subset Problems

In order to use the PSO for our search problem, we need a specific variant of the algorithm for discrete spaces and subset search problems, since our goal is to find a suitable subset of connections of the set of all possible connections in a sparse MLP. In *discrete binary PSO*, the search space is represented by a D -dimensional binary vector where $D = |M|$ is the size of the given set M . Each bit element in the vector corresponds to the existence of a corresponding element of M in the subset. Thus the position of a particle corresponds to a candidate subset or a possible solution [15] [11]. The discrete binary PSO algorithm is similar to the continuous PSO. However, particles' positions are updated differently. Since we deal with binary components, velocity is translated into probability. A high-velocity value in one dimension corresponds to a high probability that the corresponding bit takes the value of one [15].

A velocity component is computed exactly as in equation 1. However, in order to interpret velocities as probabilities, we need to map the continuous velocity values to the interval $[0, 1]$. We achieve this by applying the sigmoid function:

$$\text{sig}(v_{id}) = 1 / (1 + \exp(-v_{id})) \quad (3)$$

To determine the state of the d th bit of the position vector of particle i , we compare the sigmoid value of the corresponding velocity component v_{id} as in equation 3 with a random component r_{id} that is uniformly drawn from $[0, 1]$. This step guarantees the constant movement of particles in the search space. The bit state is determined as:

$$x_{id} = \begin{cases} 1 & : r_{id} < \text{sig}(v_{id}) \\ 0 & : \text{otherwise} \end{cases} \quad (4)$$

With equations 3 and 4 we connect the continuous movement of particles in the PSO algorithm for continuous spaces to the discrete search space [15] [11].

3. Related Work

In this chapter, studies that relate to the followed methods in this thesis are presented, such as using argumentation frameworks for machine learning problems and structure learning of multilayer perceptrons.

3.1. Argumentation Frameworks and Machine Learning

Combining argumentative reasoning and machine learning has been gaining a growing interest in recent years [20]. An example is a work of Garcez, Gabbay, and Lamb in 2005 on using neural networks for argumentation [2]. The authors indicated how *value-based argumentation frameworks* can be translated into MLPs. Each argument in a value-based argumentation framework is related to a *value*. The arguments are then either *subjectively accepted* by one member of the set of audiences that have different preferences over the values or *objectively accepted* by all these audiences. The authors also showed that the prevailing arguments in these frameworks can be computed with an MLP with a single hidden layer and a semi-linear activation function [2].

In more recent works [13] [23], authors presented attempts to apply neural networks for the approximate computation of labellings of classical argumentation frameworks.

However, the most significant and recent work for this thesis is the paper by Potyka N. in 2021 [20]. The author presented in this paper a detailed study of relationships between quantitative bipolar argumentation frameworks (QBAFs) and multilayer perceptrons (MLPs).

3.2. Sparse MLPs and MLP Structure learning

Sparse neural networks are also a topic with a recently growing research interest. The aim of sparse neural networks is not limited to obtain better interpretability, but also to reduce learning complexity and storage requirements [20].

Frankle and Carbin presented a technique for eliminating unnecessary weights from neural networks (pruning) [5]. The technique depends on *the lottery ticket hypothesis*: dense randomly-initialized neural networks contain subnetworks that can achieve an accuracy that is comparable to the accuracy that the original network has reached after training the networks for the same number of iterations.

Another related work to this topic is from Louizos, Welling, and Kingma [14]. Their proposed method for pruning is based on using L_0 regularization to encourage weights to become exactly zero during training.

In this thesis, we consider the structure learning problem as a *subset search problem*,

where we search for a subset of connections that hopefully delivers optimal performance and interpretability.

4. Methodology

4.1. BAGs as classifiers

One of the main goals of this thesis is to construct argumentation frameworks to solve classification problems. The aim of classification is to map instances of input features $x = (x_1, \dots, x_m)$ to class labels $y \in L$, where each x_i belongs to a specific domain D_i and $L = \{y_1, \dots, y_n\}$ is a finite set of class labels [19]. Classification is a widely explored problem in machine learning. The task of a classifier is to predict the class labels of instances of x .

A numerical classifier is a function $c : (X_{i=1}^k D_i) \times L \rightarrow \mathbb{R}$ that assigns a numerical value to every pair (x, y) . An important special case of numerical classifiers is the probabilistic classifier $p : (X_{i=1}^k D_i) \times L \rightarrow [0, 1]$. In this type of classifiers, the numerical value is restricted to the interval $[0, 1]$ and is comprehended as a probability where $\sum_{j=1}^{|L|} p(\mathbf{x}, y_j) = 1$. This probability is associated with the confidence of the classifier that the corresponding class label of x is y . Practically, every numerical classifier can be transformed into a probabilistic one by normalizing the numerical values for each label y_i with the softmax function [19]:

$$p_c(\mathbf{x}, y) = \frac{\exp(c(\mathbf{x}, y_i))}{\sum_{j=1}^{|L|} \exp(c(\mathbf{x}, y_j))} \quad (5)$$

As already introduced in section 2.1, argumentation frameworks can be used to solve decision problems. Intuitively, we can perceive classification also as a decision problem. Potyka describes in [19] the approach of using argumentation frameworks for classification purposes. To achieve this, several measures have to be taken. For instance, all input domains D_i have to be transformed into arguments. Here we have to distinguish between categorical and continuous features. For a categorical feature with domain $D = \{d_1, \dots, d_l\}$, we construct l corresponding arguments $A_{D,1}, \dots, A_{D,l}$. If the value of x for this feature has the value d_i , then we accept the corresponding argument $A_{D,i}$ and reject all remaining arguments $A_{D,j}, j \neq i$. For a continuous feature with domain $D \subseteq \mathbb{R}$, we need to discretize D by partitioning it into l intervals. Then, we construct l arguments that correspond to these intervals analogously to categorical features. The resulting input arguments of this step are denoted by \mathcal{A}_{in} [18].

The second measure is to convert class labels into arguments. For multiclass problems, we construct an argument for each class label analogously to the process for categorical input features. For binary classification, we have the option to construct one single argument for one of the two class labels. Rejecting this argument would

then correspond to accepting the other class label. The resulting class arguments of this step are denoted by \mathcal{A}_{out} [18].

Figure 4 illustrates a visualization of the end result of these measures.

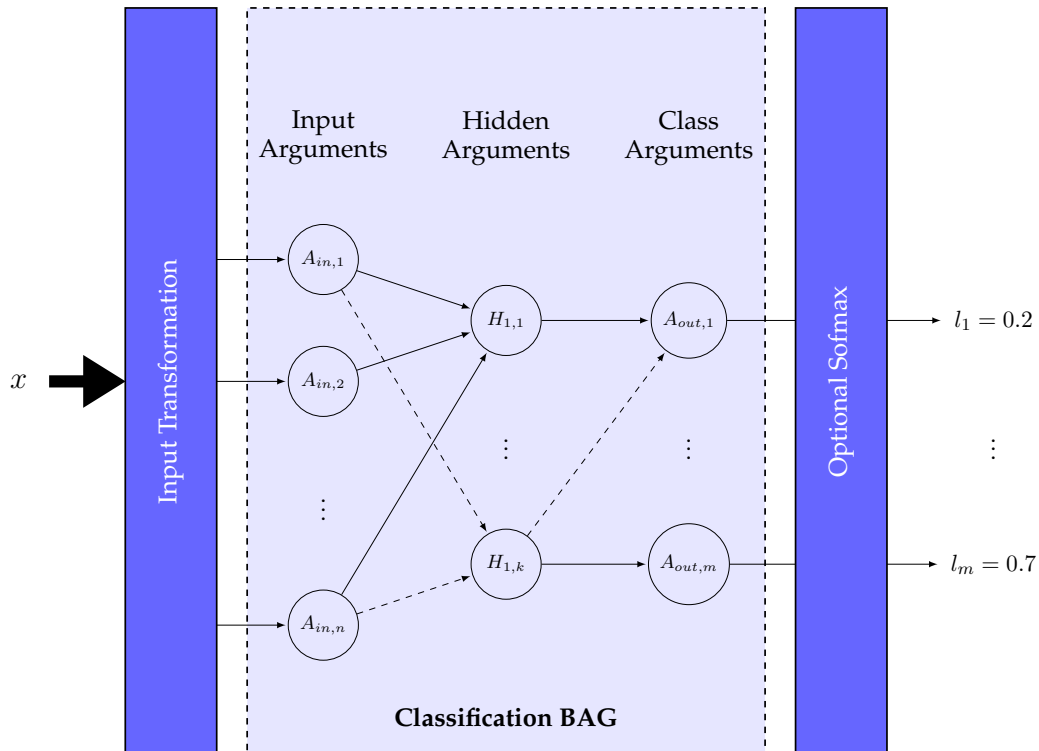


Figure 4: High-level Architecture of Argumentation Classifiers as illustrated in [18]

One of the incentives behind exploring argumentation frameworks is the interpretability of their graphical structures. Dung discusses this in his original paper of argumentation frameworks [4]. The inspiration of argumentation frameworks originates from studying the mechanisms humans use in argumentation and reasoning and trying to create similar computational models. Elements such as arguments, relations, and acceptance are also present in human argumentation, which is a major component of human intelligence [4]. Because these principles are also points of interest in this thesis, we are interested also in the topographical features of the constructed BAGs, such as sparsity.

So far, we have constructed input and output arguments out of the components of the classification problem. Constructing relation edges only between these two layers of arguments results in a BAG that has a simple structure and is easy to interpret. However, it lacks the ability to express complicated relations among the

input arguments [18]. Therefore, we construct hidden layers H_1, \dots, H_k composed of hidden arguments $H_{i,1}, \dots, H_{i,o}$ for a layer H_i in order to overcome this problem. The author in [18] defined classification BAGs with hidden layers as *deep classification BAGs*. Their principle of hidden layers was inspired by the architecture of deep feedforward neural networks, with the hope of gaining more sophisticated patterns from the hidden layers [18].

Potyka also showed in [18], that similar to the idea of the approximation theorem for neural networks [7], a classification BAG with a single hidden layer can approximate any discrete function. This can be achieved by generating a hidden argument for each possible input value that is fully connected with the input and the class arguments. This type of construction has several disadvantages, such as the possibility of overfitting the training data samples and high complexity in the graphical structure, which leads to a difficulty in interpreting such models [18].

Since interpretability is an essential motivation behind using classification BAGs, we are interested in creating *sparse classification BAGs*. In a sparse classification BAG, arguments in some two layers are not fully connected among each other. A sparsely connected network with a high sparsity degree has a low number of connections among its nodes. Therefore, we consider this structure learning problem as an optimization and a search problem and try to solve it using the particle swarm optimization algorithm with the hope of reaching models with interesting features.

4.2. Dataset Preprocessing

As already mentioned in the previous chapter, implementing classification BAGs requires preprocessing of the data in order to create arguments of input features and class labels. Because we build argumentation frameworks as MLPs (MLP-based semantics), we prepared the datasets for MLP implementation. In the following, we describe the techniques that were used on the experiments' datasets.

For each categorical feature with a finite number of values k , an argument was created for each value that corresponds to it. This was achieved by transforming these features to a *one-hot encoding*. A one-hot encoder translates the values into a binary vector with k corresponding bits. When a sample takes a value, its corresponding bit will be set to one and all other bits will be set to zero. This technique actually fulfils our requirements, since we need inputs in $[0, 1]$, and extreme values, i.e., from $\{0, 1\}$ denote total acceptability (1) or total rejection (0) of the respective argument.

For continuous numerical features, discretization is necessary to convert them into categorical features. Afterwards, they can analogously be one-hot encoded. The discretization method that was used is *binning*. A *k-Bin discretizer* divides the value range of the numerical feature into k interval [17]. Each bin represents a single new

value that represents all original values that reside in the corresponding interval. This results in k categorical values that can be one-hot encoded. However, one task remains to be done, which is defining the widths of the bins. There are different strategies to choose from, such as *equal width binning (uniform)* and *equal frequency binning (quantile)*. In the "uniform" strategy, all the bins share the same width, i.e., the range of values is divided into k equally wide intervals that correspond to the bins. The advantage of this strategy is that it will preserve the probability distribution of the values. In the "quantile" strategy, however, bin widths or interval edges are defined in a way that guarantees that each bin has an approximately equal number of samples. This is done by splitting the range of values into k intervals that have equal sample frequencies depending on calculating distribution quantiles. More information can be found in the documentation of the scikit-learn library [17].

In all the experiments, the quantile strategy was chosen. The reasoning behind this choice is its guarantee of an approximately equal number of samples in each bin. Other strategies can also be used. However, their exploration was not part of this thesis. The number of bins was also experimentally defined after testing different values. Regarding target features, multiclass labels were handled as categorical features and were translated into a one-hot encoding. Binary classes were represented by single bits.

The end result of the dataset preprocessing procedures is a single two-dimensional binary matrix (dataframe), whose dimensions are the number of samples and the number of features. The dataset matrix can be then used for the parameter learning process.

4.3. Parameter Learning

MLP models are implemented in the thesis for two purposes. The first one is to implement argumentation classifiers under MLP-based semantics as explained in section 2.2. The models for this purpose are sparse MLPs, i.e., the nodes (neurons) in some two neighbouring layers are not fully connected with each other. This kind of models is also used for structure learning, as explained in the next section. The second purpose is to implement fully connected classical MLP models as a comparison reference for high performance since we know that removing connections in an MLP could compromise its performance [18]. In the following, we present the architecture and techniques that were applied in this thesis in order to implement, train, and test MLP models, i.e., parameter learning.

Each MLP model consists of an input layer, an output (class) layer, and a single hidden layer. The input and output layers are preprocessed as shown in section 4.2. We were confined to one hidden layer to maintain the simplicity of the graphical structure. The hidden layer has the sigmoid function $\text{sig}(x) = 1 / (1 + \exp(-x))$ as an activation function. The softmax function is also used as an activation function on class layers.

Parameter learning can be based on minimizing a loss function. A loss function that is commonly used for classification problems is the *cross entropy loss*. Cross entropy loss for non-binary classification is defined as:

$$\mathcal{L}(\theta) = -\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^{|L|} y_{ij} \cdot \log(\hat{y}_{ij})$$

The loss value is associated with the current parameter tensor θ . For each sample x_i , we calculate its prediction loss by iterating over all class labels $l_j \in L$ and multiplying log of prediction probability of the label $\log(\hat{y}_{ij})$ with the term y_{ij} . y_{ij} is a binary value that takes the value one when the correct class label is l_j . We apply this to all N training samples and sum their loss values and finally invert the result [18].

A commonly used method for minimizing the loss function is *gradient descent* which is an iterative optimization algorithm for finding a local minimum of an objective function [18]. It is based on moving in the search space in iterative steps in the opposite direction of the gradient of the objective function. At the early stage of implementation, the standard algorithm for stochastic gradient descent (SGD) in the `pytorch` library was used in experimentation. Later, however, *Adam* (Adaptive Moment Estimation), which is an optimized variant of the algorithm, was adopted for optimizing the loss function minimization. This adoption was based on the knowledge of Adam's advantage of increasing the training speed [12]. The usage of this optimization variant showed a substantial acceleration on the learning process. More details on this method can be found in [12].

The functionality of MLPs is based on two key mechanisms. The first one is feed-forward propagation which is used for calculating activation values on each node. This is done by propagating them through the network, starting on the input level and ending on the class level. The second mechanism is back-propagation which functions similarly to feed-forward propagation and is used for learning the weights and biases of the network. This is achieved by calculating the gradients of the loss function and propagating them in the opposite direction of feed-forward propagation. The gradients are then used by the gradient descent algorithm to update the parameters [6]. The number of training iterations (*epochs*) and the *learning rate* of gradient descent are considered and explored as hyperparameters.

Training a model requires splitting the dataset into two subsets, the training set and the test set. As its name reveals, the training set is used for the actual training of the parameters. The test set is then used after the training process finishes to test it. Several measurements (scores) are used to evaluate parameter learning, such as accuracy, precision, recall, and the F_1 -score. The Accuracy score is the percentage of correctly classified instances. The other scores are similar. However, they take *false (true) positives* and *false (true) negatives* into account. More details on these measurements are found in [21].

All implementations of the previously presented methods, such as data preprocessing, MLP models, parameter learning, and the evaluation scores were implemented using several external python libraries such as `pandas`¹, `pytorch`², `sklearn`³, and `sparselinear`⁴. More information and documentation can be found in the URLs in the footnote section.

4.4. Structure Learning

In order to find a BAG with a desired graphical structure, the discrete binary PSO algorithm was used. We use the PSO algorithm to search for sparse MLP models with particular features such as high sparsity and performance. The found MLP models would then be interpreted as QBAFs or BAGs under the MLP-based semantics that were introduced in section 2.2. The quality of a sparse MLP model depends on its classification performance after learning its weights and biases as presented in section 4.3 and the number of connections the model has.

In order to use the PSO algorithm properly, we need to do some modelling, such as defining particles, the search space, and the objective function.

4.4.1. Particles and Search Space

A *particle* is defined as a sparse MLP model with a graphical structure that has a specific configuration of edges (connections).

The search problem can be viewed as a subset search problem where the given set M is the set of all possible connections among the three layers of the MLP. A subset represents a position point in the discrete space. As shown in chapter 2.3.3, the position of a particle at a certain point in time is represented as a D -dimensional binary vector with each element denoting the existence of a corresponding element in the given set, and D is the cardinality of M . The position vector was modelled by concatenating two vectors; each is the result of vectorizing (flattening) the corresponding connection matrix described in section 4.3. Vectorization is a linear transformation that converts an $m \times n$ matrix into an $m \cdot n \times 1$ vector as illustrated in figure 5. In the python implementation, a helper script with functions was implemented to transform *connectivity matrices* of the `sparselinear` library into our modelling of connection matrices that was introduced in chapter 4.3. More information can be found in the documentations of `sparselinear`.

For the velocity vectors, there is no need for modelling since they are only a functional part of the algorithm.

¹<https://pandas.pydata.org/>

²<https://pytorch.org/>

³<https://scikit-learn.org/>

⁴<https://pypi.org/project/sparselinear/>

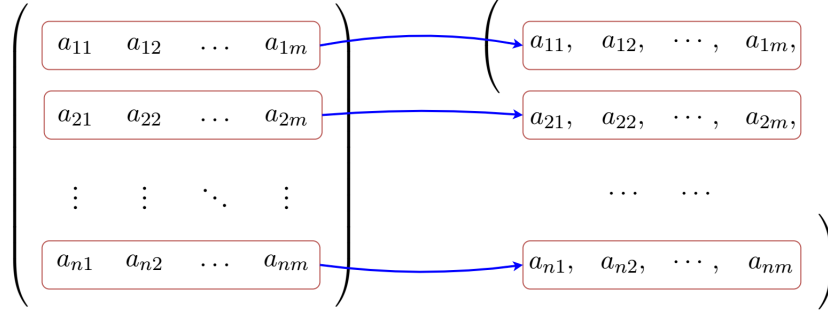


Figure 5: Illustration of transforming a binary connection matrix of a sparse MLP layer into a vector, in order to construct the search space position vector.

4.4.2. Objective Function

To evaluate a particle's position x , we need to define an objective function f that assigns a score to the particle. In our objective function, two elements are considered. The first element is the classification performance. For this element, we used the accuracy score (see section 4.3) $Acc(x) \in [0, 1]$, where $Acc(x) = 1$ if the model was able to successfully predict all instances.

The second element is the sparsity of the model's graphical structure. Sparsity is based on the number of existing connections in the network $n(x)$. This number is subtracted from and then normalized by the number of all possible connections in the network N_{total} . The resulting term takes values in $[0, 1]$, where 0 denotes a fully connected graph and 1 denotes a graph with no connections.

The accuracy and sparsity terms are regularized by a parameter $\alpha \in [0, 1]$. This regularization parameter conveys the influence of each term in f . Thus, the higher the values α takes, the more significant is the accuracy term in the function and vice versa:

$$f(x) = \alpha \cdot Acc(x) + (1 - \alpha) \cdot \frac{N_{total} - n(x)}{N_{total}}$$

α is also considered and explored as a hyperparameter in the implementation.

The objective function f takes values in $[0, 1]$. The PSO algorithm aims to maximize f since high values indicate high performance and high sparsity. However, a score of 1 is practically unreachable because it necessitates a graphical structure with zero connections. This function was also deployed for a similar purpose in [24].

4.4.3. Initialization

All position vectors are initialized by randomly distributing a predefined number of initial connections in each connection matrix. The intuition behind this partially ran-

dom initialization is that we have some prior knowledge of the problem. We know that the desired and expected solution models should have a very low number of connections. Therefore, we can initialize them with a low number of connections and depend on the fine-tuning methods, introduced in chapter 4.4.5, to keep the number of connections relatively low.

The two initial numbers of connections are predefined and considered hyperparameters. In the initialization, we randomly chose a subset of n bits from a connection matrix, where n is the number of initial connections in the matrix, and set these bits to 1. All other bits will remain 0.

For velocity vectors, all the velocity values are initialized with zero. The velocity values will then start to develop in the first iteration of the optimization.

4.4.4. Stopping Condition and Dynamic Execution

The stopping condition is dropped in the implementation. Instead, the optimization loop was executed for a predefined number of iterations. The motivation behind this was to obtain the freedom of monitoring the optimization process, and the ability of dynamically changing certain parameters during runtime, such as the acceleration coefficients and the number of training epochs. The number of iterations was experimentally explored and is considered a hyperparameter. As a result, the PSO algorithm was split into two main sub-routines: *initialize* and *optimize*. In "initialize", particles and position/velocity vectors are initialized. The "optimize" sub-routine takes an argument n (the number of iteration) and runs the optimization loop for n iterations. Figure 2 illustrates an algorithm scheme with both sub-routines.

In all experiments, the optimization procedure was executed in three phases. This approach aims to allow particles to explore more in the first phase and to exploit the advanced search state more in the last phase. This can be achieved by modifying the social and cognitive parameters (acceleration coefficients) c_1 and c_2 during runtime. The following python code snippet illustrates a simplified example for running the optimization with this approach:

```
# initialization
pso.initialize()

# first phase
pso.c_1 = 2
pso.c_2 = 2
pso.optimize(num_of_iterations=20)

# second phase
pso.c_1 = 4
pso.c_2 = 4
pso.optimize(num_of_iterations=50)

# third phase
```

```
pso.c_1 = 1
pso.c_2 = 10
pso.optimize(num_of_iterations=30)
```

4.4.5. Fine-Tuning

During experimentation with the algorithm, it was noticed that the number of connections in each model rapidly increases in the early stage, which led to high inefficiency.

Despite initialization with a very small number of connections, the MLP models in the second iteration would have a number of connections that distributes around $\frac{N_{total}}{2}$, where N_{total} is the number of overall possible connections. This happens due to randomness and the lack of information the algorithm has in the early stages of optimization.

In an attempt to solve this problem, two measures were introduced:

Upper and lower boundaries were implemented to ensure that the numbers of connections do not exceed certain values. Bits in connection matrices are prevented from changing to one (or to zero) if the associated number of connections does not lie within these boundaries. This mechanism was inspired by defining search space boundaries and particle reflection in PSO for continuous spaces. The upper and lower boundaries are also considered as hyperparameters in the implementation.

However, low upper boundaries were not enough to restrain the high number of connections. Therefore, a **velocity bias** parameter, which is the second measure, was introduced. The velocity bias ϵ serves as a suppression factor for randomly high velocity values. As an outcome of this bias term, unless velocity values are high enough because of the acceleration coefficients and the weighted previous velocity value, it is more unlikely for their associated bits to be set to one. It is also useful to note that all differential terms in the velocity equation 1 can have only values in $\{-1, 0, 1\}$. Therefore, the parameters and the random terms in the velocity equation play the main role in determining and scaling the velocity value.

The bias ϵ is subtracted from the sigmoid of the velocity before comparison with the random factor r_{id} :

$$x_{id} = \begin{cases} 1 & : r_{id} < \text{sig}(v_{id}) - \epsilon \\ 0 & : \text{otherwise} \end{cases} \quad (6)$$

Although these measures were experimentally implemented and were not studied or analyzed, they contributed to the optimization results and efficiency, as one can see in section 5. Besides, one can argue that their mechanism is analogous to regularization approaches in training of neural networks. Therefore, they were kept in the final implementation.

4.4.6. Time Efficiency

Evaluating a particle's position in this implementation includes constructing an MLP, training it, and evaluating its training. This leads to a very time-consuming search problem. Therefore, two measures were taken to accelerate the optimization. The first measure is *early stopping* of parameter learning. Early stopping methods are usually used as regularization forms to tune the training process and avoid overfitting [22]. However, we use it here to accelerate the PSO algorithm.

During training, if the current loss value does not decrease with a predefined *early stopping threshold*, the model is given a chance of a certain number of epochs (*patience*) to achieve this decrease. If not reached, the training process will be stopped. This approach can be justified by arguing that models with only "bad" connections will not be able to improve in training from the start, what makes it efficient to not fully train them. This technique was explored initially on fully connected MLP models, where it was experimented with different values of patience and threshold. The patience and the early stopping threshold are also considered hyperparameters of the PSO algorithm.

The second measure is gradually and dynamically increasing the number of training epochs between optimization phases. This can be justified by arguing that in the early and middle stages of the optimization, the performance scores (accuracy) do not have to be very accurate to judge the contribution of existing connections to the performance. In the last stage, the greatest number of epochs is used for training all models.

Algorithm 3 Scheme of the two procedures of the PSO algorithm as used in the implementation

```
1: procedure INIT_PARTICLES()
2:   for particle  $i \in 1, \dots, m$  do:
3:     initialize  $x_i$  randomly with a specific number of connections
4:     initialize  $v_i$  as  $(0, \dots, 0)$ 
5:   end for
6: end procedure
7:
8: procedure OPTIMIZE(iterations)
9:    $i \leftarrow 0$ 
10:  while  $i < iterations$  do
11:    for particle  $P_i, i \in 1, \dots, m$  do:
12:      evaluate  $P_i$  and update its score
13:      update the personal best score of  $P_i$ 
14:      update  $p_i$  (the personal best position of  $P_i$ )
15:    end for
16:    update the global best position  $g$ 
17:    for particle  $P_i, i \in 1, \dots, m$  do:
18:      update velocity of  $P_i$ 
19:      update position of  $P_i$ 
20:    end for
21:  end while
22: end procedure
23:
```

5. Experiments and Results

The previously presented methodology was applied on three different datasets from the *UCI machine learning repository*⁵ [3]. These datasets differ in size and complexity and had to be approached differently.

For each dataset, the data were preprocessed before deploying the PSO algorithm for structure learning. Parameter learning was explored separately before, in order to specify its parameters, namely the number of epochs and the learning rate. The preprocessed dataset was split into train and test subsets with a ratio of 80/20%. The split was done randomly with the predefined random seed for each experiment run to maintain reproducibility. The progress of the training loss function decrease was observed to roughly specify early stopping parameters, namely the patience number and the early stopping threshold. These hyperparameters were deployed afterwards in the structure learning process. For each dataset, the PSO was executed ten times with different random seeds. The random seeds are identical to those used for train/test splitting. The results of these runs were then averaged and summarized.

To evaluate the results, two different types of classifiers were implemented on each dataset as comparison benchmarks:

- fully-connected MLP models to compare their performance with the performance of the resulting BAG classifiers
- decision trees with different depths to observe the effect of limited depth on tree performance. Also, looking for tree models with a performance that is comparable to the resulting BAG classifiers and compare both models in terms of structural simplicity.

Extensive details on each dataset and the experiments' results are summarized in the appendix A. The appendix also includes tables with all hyperparameter values and run configurations for each experiment. Additionally, all experiments are reproducible, since they were executed with fixed random seeds for the pseudo-random number generators of the used python libraries. The implementation with the result reports can be found in the git repository⁶.

5.1. The Iris Dataset

The *Iris dataset* is the first dataset that was explored. It is well known for its simplicity, small size, and feasibility of yielding good classification results. It is also considered a typical test case for many machine learning classification methods [3]. The dataset consists of 50 samples from each of the three species of the Iris plant,

⁵<https://archive.ics.uci.edu/>

⁶<https://gitlab-ac.informatik.uni-stuttgart.de/potykanostudy-project-mohamad-wahed-bazo/>

where one of the classes is linearly separable from the other two classes. There are four continuous numerical features that relate to certain properties of the iris flower. The target attribute is the species of the flower.

In the preprocessing phase, all four input features were discretized with k-bin-discretizers. The bin size was set to 10 samples for all four discretizers. As explained in section 4.2, the quantile strategy was used, and a reasonable bin size was experimentally chosen. Preprocessing resulted in 12 one-hot-encoded input nodes and three class nodes, with each node corresponding to each species.

MLP models built for parameter learning and then structure learning consist of one hidden layer with six hidden nodes. The number of hidden nodes was experimentally chosen. Nevertheless, six hidden nodes were sufficient since all resulting BAG models had at least two hidden nodes that were not connected to any feature. After some exploration with parameter learning, a learning rate of 0.01 was found suitable. Also, about 450 epochs of training were sufficient to reach a test accuracy of 100% in a fully connected model. In structure learning, a population of 30 particles were deployed in the PSO algorithm. After experimentation with different values for parameter α and observing their effect on particles' scores, $\alpha = 0.7$ was found a balanced value for both good performance and high sparsity. The appendix section for this dataset A.1 contains extensive details on the optimization's parameters and configurations. Also, two models were illustrated as BAGs; they highlight the model with the highest performance and the model of highest sparsity (figures 6 and 7 respectively).

Averaging ten runs of the PSO algorithm with the previously presented configurations, we obtain the results in table 1.

Training subset accuracy	Test subset				number of connections	
	accuracy	recall	precision	f_1 score	1. layer	2. layer
0.8933	0.9433	0.9330	0.9471	0.9346	4	2.4
std	std				std	std
0.0359	0.0299				1.5491	0.6633

Table 1: Average of results after running the PSO algorithm ten times with different random seeds on the iris dataset and with the parameters and configurations in table 5. The data describe performance scores on the train and test subsets and the average number of connections in both layers of the found models.

According to these results, we can say that we were able to reach relatively good performance with very few connections. The relatively low standard deviation values of accuracy scores and numbers of connections also indicate the stability of these results. However, the small size of this dataset prevents us from drawing solid conclusions over the implementation. In fact, the test subset on this dataset contains only 30 samples when splitting with an 80/20% train-test ratio.

The experimentation also showed that we can reach very good optimization results with a relatively small particle population and few PSO iterations, in comparison with the other two datasets.

An also interesting observation is that the feature " $pl \in [5.1, 6.9]$ " appeared in nine of the ten resulting models as a connected feature.

5.2. The Mushroom Dataset

The next and second-largest dataset that was deployed in the experiments is the *mushroom dataset*. The dataset contains descriptions of hypothetical samples that correspond to 23 species of gilled mushrooms [3]. Each sample is identified as "edible" or "poisonous", which represents the target class for classification. There are 22 categorical input features that describe certain characteristics such as *odor*, *gill color*, and *cap shape*. The dataset has 8124 samples that are relatively balanced regarding the target feature (52% *edible* and 48% *poisonous*). Also worth mentioning is that there is no simple rule to determine the edibility of a mushroom in the dataset [3]. The mushroom dataset was explored the most in the thesis due to its medium size and complexity.

The preprocessing of the dataset included transforming the 22 input features into 111 one-hot-encoded columns. There was no need for discretizations since all features are categorical. Because some samples have missing values of the feature "stalk root", the one-hot column that corresponds to the missing value (encoded as "?") was dropped. The target feature was encoded in one binary column where the value of 1 corresponds to the label "poisonous".

For this dataset, a population of 100 particles was used in the PSO algorithm. A 0.7 value for parameter α was found balanced after some exploration. This experimentally found value achieved a relative balance between performance and sparsity. The used run configurations with the corresponding PSO parameter values can be found in the appendix section A.2. Particles were initialized with a hidden layer with five nodes, and with (15, 5) initial connections for the two connection matrices, respectively. For parameter training, a learning rate of 0.01 and 300 epochs were found suitable for the training process. These values were also experimentally found after exploring parameter training on a fully connected MLP.

Averaging ten runs with the previous configuration and with different random seeds, we received the results in table 2. Noticeable here is that the numbers of

all performance metrics are relatively high, and the number of connections are relatively low. The low standard deviation values also indicate the stability of the results.

Training subset accuracy	Test subset				number of connections	
	accuracy	recall	precision	f_1 score	1. layer	2. layer
0.9765	0.9761	0.9759	0.9764	0.9761	4.2	2.2
std	std				std	std
0.0070	0.0059				0.8717	0.3999

Table 2: Average of results after running the PSO algorithm ten times with different random seeds on the mushroom dataset and with the parameters and configurations in table 8. The data describe performance scores on the train and test subsets and the average number of connections in both layers of the found models, in addition to the standard deviation of accuracy and number of connections.

Table 3 summarizes the results of ten runs of the PSO with the same configurations as before, however, without the usage of the velocity bias ϵ and upper boundaries of connections. We notice here the higher values in performance, however drastically higher numbers of connections. This can be considered as a validation of the usage of these approaches in the implementation.

Training subset accuracy	Test subset				number of connections	
	accuracy	recall	precision	f_1 score	1. layer	2. layer
0.9906	0.9904	0.9902	0.9907	0.9904	66.2	4.6
std	std				std	std
0.0056	0.0058				14.8512	0.4898

Table 3: Results of running the PSO algorithm with the same configurations in table 8, however with $\epsilon = 0$ and without the usage of upper boundaries in all optimization phases.

Two BAG models were illustrated in figures 10 and 11. Figure 10 represents a BAG that depended on only three values of a single input feature "*odor*". In fact,

the argument "*odor = n*" appeared in all ten found models as a connected feature. This motivated the analyses of this feature. Figure 12 shows a count plot for the feature "odor". The plot illustrates how this feature can divide the dataset based on the target feature to a good extent, especially the value "n", where a relatively big portion of the samples is concentrated with a predominance of the class "edible". This analysis contributed to the validation of this finding in the PSO results.

Finally, comparing the PSO results in table 2 with the results of fully connected MLPs in table 9 and of decision trees in table 10, we can be convinced that the found BAG models are a good combination of both high performance and good simplicity.

5.3. The Adult Income Dataset

The *adult income dataset* is the third and largest dataset that was explored in the experiments. The classification aim of this dataset is to predict whether an individual's annual salary exceeds \$50,000 based on census data, such as age, gender, and education level. The dataset consists of 48842 samples with 13 attributes that comprise personal information of individuals, in addition to the target attribute (income). The features vary between categorical features and continuous (integer) features [3]. The two classes are imbalanced, with a skew towards the " ≤ 50 " class, approximately 75/25%.

In the preprocessing phase, all samples with missing values were dropped. One-hot columns were created out of the categorical features. For continuous features, k-bin-discretizers were deployed with the quantile strategy. The bin size for each feature was chosen experimentally with regard to the particularity of each feature and with the attempt of reducing the overall number of one-hot encoded columns. For example, the feature 'age' has values that range between 17 and 90 years. A number of bins of 5 was found reasonable to divide this feature into five age groups. The target feature was also encoded in a single binary column, where the value of 1 corresponds to a salary that is greater than \$50,000 a year. The outcome of this process is 119 binary input features and one binary output feature.

This dataset was not widely explored as the mushroom dataset due to time limitations. The most significant issue that was not addressed is the imbalance of class labels. Despite this, the experiment was done to have an extra reference on the implementation. In the PSO implementation, the parameter α had to be slightly increased to $\alpha = 0.8$ in order to maintain a balance. Table 4 shows the results of the PSO optimization. Comparing accuracy scores with the results of fully connected MLPs in table 12, we find some progress in achieving performance in the BAG models. Worth mentioning here is that the accuracy of 75% can be considered as random classification since this complies with the imbalance ratio of the dataset. We also notice higher numbers of connections and higher standard deviation values in comparison with the other datasets.

Regarding connected input features, it was noticed that certain features reoccurred in the resulting BAGs. Values of the feature "education", for instance, existed in all models as connected features. The feature "marital status: married civ spouse" is also present in eight of the ten found models. Although these features were not analyzed due to time limitations, we can consider these findings interesting for interpretability discussions.

Training subset	Test subset				number of connections	
accuracy	accuracy	recall	precision	f_1 score	1. layer	2. layer
0.8103	0.8129	0.6870	0.7640	0.7086	6.7	2.4
std	std				std	std
0.0082	0.007				2.14	0.68

Table 4: Average of results after running the PSO algorithm ten times with different random seeds on the income dataset and with the parameters and configurations in table 11. The data describe performance scores on the train and test subsets and the average number of connections in both layers of the found models, in addition to the standard deviation of accuracy and number of connections.

6. Conclusion

In this thesis, we explored argumentation frameworks as argumentative classifiers. Quantitative bipolar argumentation frameworks, represented as bipolar argumentation graphs, were implemented to solve classification problems. This implementation was based on the ability of interpreting multilayer perceptrons as QBAFs. Swarm intelligence was also deployed in the thesis. The problem of finding BAGs with specific graphical features related to performance and graph-structural characteristics was considered a subset search problem. The particle swarm optimization algorithm was used for this purpose.

The two fundamental purposes of this study were to implement, test, and evaluate BAG classifiers on real classification problems and to implement and test the particle swarm optimization and explore its efficiency in solving such a problem of BAG structure learning.

After viewing and discussing the results and findings, we can assert that argumentation frameworks can be considered an important and versatile explainable machine learning technology. We were able to construct efficient BAG models with very simple graph structures that can be a subject of interpretability discussion. Comparing these BAG models with the comparison references, we can also claim that the BAGs achieved very good results in terms of performance in fully connected MLPs, and in terms of graphical complexity to performance relationship in decision trees.

The particle swarm optimization algorithm also proved its efficiency in solving the search problem. The algorithm was able to deliver consistent results when executed with different random states. Plotting the progress of the optimization has also shown the advantage of the ability to monitor and dynamically modify the optimization process. As also illustrated in the results, we can conclude that the deployment of the fine-tuning measures, like the velocity bias parameter ϵ , is clearly advantageous for the efficiency and quality of results. Being inspired by regularization approaches in training of neural networks, when combined with the suitable acceleration parameter values, the velocity bias parameter was able to suppress volatile connections without preventing contributing connections from remaining in the structure.

Finally, argumentation plays an essential role in human intelligence and reasoning. This makes argumentation frameworks an interesting explainable machine learning technology. As this argument was part of the motivation behind the original work of Dung on argumentation, it has also been a part of motivation behind working on this thesis.

7. Limitations and Future Work

Due to time limitations, there were some elements in the implementation that could be further explored. For instance, some aspects that relate to the hyperparameters of the PSO algorithm can be further explored, such as discretization strategies and the numbers of bins. Another example is the hyperparameter of the number of training epochs. In the implementation, we used a static number of epochs for parameter training of all particles. This could lead to underfitting or overfitting in some models. This issue can be avoided by using a validation subset of the training subset to tune the training process.

Additionally, the initialization strategy can be further explored. Despite the good results, one cannot conclude that the partially random initialization method that was applied does not have disadvantages on the results, compared with a completely random initialization. Therefore, the further exploration and analysis of the two strategies can be a subject of future work.

Another idea for future work is to modify the objective function of the PSO algorithm to include additional terms that relate to the topological features of the BAG beside sparsity. For example, we can encourage the formation of incoming edges to a certain node that share similar signs and discourage the formation of double attack relations among three nodes, as in figure 7. These additional measures could lead to simpler structures of the BAGs.

References

- [1] Leonora Bianchi et al. “A survey on metaheuristics for stochastic combinatorial optimization”. In: *Natural Computing* 8.2 (2009), pp. 239–287.
- [2] Artur S D’Avila Garcez, Dov M Gabbay, and Luis C Lamb. “Value-based argumentation frameworks as neural-symbolic learning systems”. In: *Journal of Logic and Computation* 15.6 (2005), pp. 1041–1058.
- [3] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: <http://archive.ics.uci.edu/ml>.
- [4] Phan Minh Dung. “On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games”. In: *Artificial intelligence* 77.2 (1995), pp. 321–357.
- [5] Jonathan Frankle and Michael Carbin. “The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks”. In: *International Conference on Learning Representations*. 2018.
- [6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.
- [8] James Kennedy. “Swarm intelligence”. In: *Handbook of nature-inspired and innovative computing*. Springer, 2006, pp. 187–219.
- [9] James Kennedy and Russell Eberhart. “Particle swarm optimization”. In: *Proceedings of ICNN’95-international conference on neural networks*. Vol. 4. IEEE. 1995, pp. 1942–1948.
- [10] James Kennedy and Russell Eberhart. “Particle swarm optimization”. In: *Proceedings of ICNN’95-international conference on neural networks*. Vol. 4. IEEE. 1995, pp. 1942–1948.
- [11] James Kennedy and Russell C Eberhart. “A discrete binary version of the particle swarm algorithm”. In: *1997 IEEE International conference on systems, man, and cybernetics. Computational cybernetics and simulation*. Vol. 5. IEEE. 1997, pp. 4104–4108.
- [12] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).
- [13] Isabelle Kuhlmann and Matthias Thimm. “Using graph convolutional networks for approximate reasoning with abstract argumentation frameworks: A feasibility study”. In: *International Conference on Scalable Uncertainty Management*. Springer. 2019, pp. 24–37.
- [14] Christos Louizos, Max Welling, and Diederik P Kingma. “Learning Sparse Neural Networks through L₀ Regularization”. In: *International Conference on Learning Representations*. 2018.

- [15] Daniel Merkle and Martin Middendorf. “Swarm Intelligence”. In: *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Ed. by Edmund K. Burke and Graham Kendall. Boston, MA: Springer US, 2005, pp. 401–435. ISBN: 978-0-387-28356-2. DOI: 10.1007/0-387-28356-0_14. URL: https://doi.org/10.1007/0-387-28356-0_14.
- [16] Till Mossakowski and Fabian Neuhaus. “Modular semantics and characteristics for bipolar weighted argumentation graphs”. In: *arXiv preprint arXiv:1807.06685* (2018).
- [17] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.
- [18] Nico Potyka. “A tutorial for weighted bipolar argumentation with continuous dynamical systems and the java library attractor”. In: *arXiv preprint arXiv:1811.12787* (2018).
- [19] Nico Potyka. “Foundations for Solving Classification Problems with Quantitative Abstract Argumentation.” In: *XI-ML@ KI*. 2020.
- [20] Nico Potyka. “Interpreting Neural Networks as Gradual Argumentation Frameworks (Including Proof Appendix)”. In: *arXiv e-prints* (2020), arXiv–2012.
- [21] David MW Powers. “Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation”. In: *arXiv preprint arXiv:2010.16061* (2020).
- [22] Garvesh Raskutti, Martin J Wainwright, and Bin Yu. “Early stopping and non-parametric regression: an optimal data-dependent stopping rule”. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 335–366.
- [23] Régis Riveret et al. “Neuro-Symbolic Agents: Boltzmann Machines and Probabilistic Abstract Argumentation with Sub-Arguments.” In: *AAMAS*. 2015, pp. 1481–1489.
- [24] Jonathan Spieler, Nico Potyka, and Steffen Staab. “Learning Gradual Argumentation Frameworks using Genetic Algorithms”. In: *arXiv preprint arXiv:2106.13585* (2021).

A. Appendix

A.1. The Iris Dataset

Attribute Information

Input features:

1. Sepal length in cm
2. Sepal width in cm
3. Petal length in cm
4. Petal width in cm

Classes:

1. Iris Setosa
2. Iris Versicolour
3. Iris Virginica

Number of input nodes after preprocessing: 12

population	hidden nodes	α	num. of initial connection	upper boundaries	patience	threshold
30	6	0.7	7, 3	None	None	None
	phase	iterations	w	c_1	c_2	ϵ
	first:	30	0.9	2	2	0.4
	second:	30	0.9	4	4	0.4
	third:	20	0.9	1	10	0.4

Table 5: Hyperparameter values and phase configurations for the iris dataset.

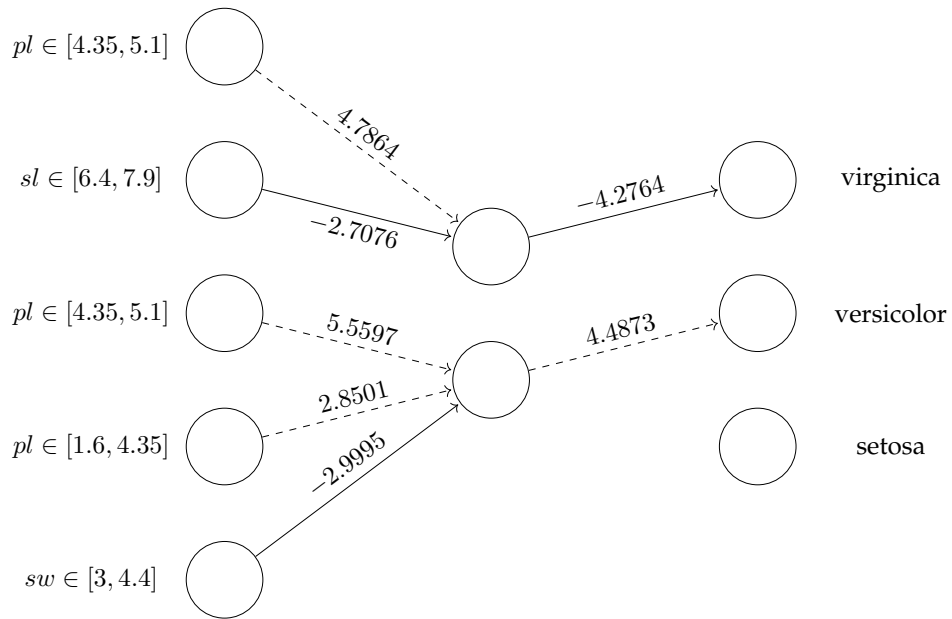


Figure 6: An example graph of a sparse MLP model (interpreted as a BAG) for the iris dataset. The model achieved an accuracy of 100% on the test set. The model was found using the PSO configuration in table 5, and it is the model with the best accuracy score.

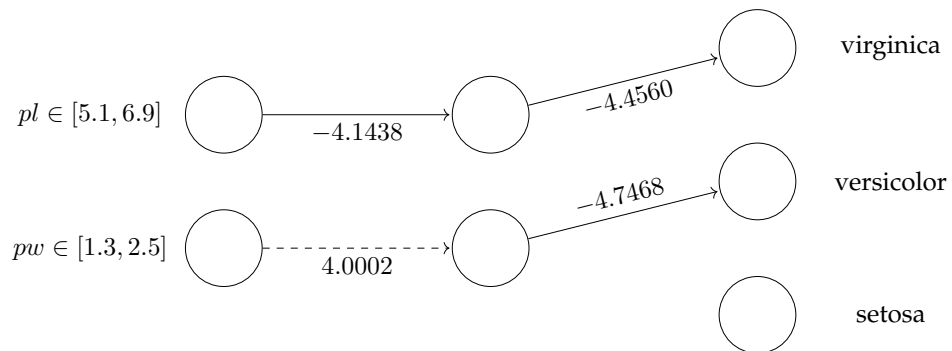


Figure 7: Graph of a sparse MLP model (interpreted as a BAG) for the iris dataset that was found using the PSO configuration in table 5. This model had the highest sparsity and achieved an accuracy of 93.33% on the test set.

Training subset accuracy	Test subset			
	accuracy	recall	precision	f_1 score
0.9783	0.98	0.9771	0.9825	0.9786

Table 6: Performance results of training a fully connected MLP with six hidden nodes on the iris dataset, averaged over ten runs with different random seeds. Each of the models was trained over 450 epochs and with a learning rate of 0.01.

maximum depth	avg. accuracy on training subset	avg. accuracy on test subset
1	67.25%	64.33%
2	96.58%	93.33%
3	97.83%	95.33%
4	99.33%	94.33%
5	99.83%	94.66%
6	100%	95%
7	100%	95%

Table 7: Averaged accuracy scores of decision tree models with various depths on the iris dataset. The values for each depth were averaged over ten different runs, each with a unique random seed.

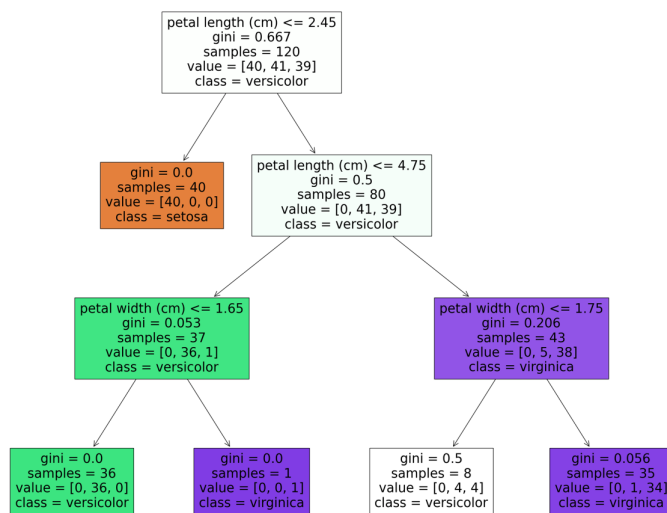


Figure 8: Example of a decision tree for the iris dataset with a depth of three (random seed 42). The decision tree reached an accuracy of 95.83% on the training set and 100% on the test set.

A.2. The Mushroom Dataset

Attribute Information

Input features:

1. cap-shape: bell=b, conical=c, convex=x, flat=f, knobbed=k, sunken=s
2. cap-surface: fibrous=f, grooves=g, scaly=y, smooth=s
3. cap-color: brown=n, buff=b, cinnamon=c, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y
4. bruises?: bruises=t, no=f
5. odor: almond=a, anise=l, creosote=c, fishy=y, foul=f, musty=m, none=n, pungent=p, spicy=s
6. gill-attachment: attached=a, descending=d, free=f, notched=n
7. gill-spacing: close=c, crowded=w, distant=d
8. gill-size: broad=b, narrow=n
9. gill-color: black=k, brown=n, buff=b, chocolate=h, gray=g, green=r, orange=o, pink=p, purple=u, red=e, white=w, yellow=y
10. stalk-shape: enlarging=e, tapering=t
11. stalk-root: bulbous=b, club=c, cup=u, equal=e, rhizomorphs=z, rooted=r, missing=?
12. stalk-surface-above-ring: fibrous=f, scaly=y, silky=k, smooth=s
13. stalk-surface-below-ring: fibrous=f, scaly=y, silky=k, smooth=s
14. stalk-color-above-ring: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y
15. stalk-color-below-ring: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y
16. veil-type: partial=p, universal=u
17. veil-color: brown=n, orange=o, white=w, yellow=y
18. ring-number: none=n, one=o, two=t
19. ring-type: cobwebby=c, evanescent=e, flaring=f, large=l, none=n, pendant=p, sheathing=s, zone=z
20. spore-print-color: black=k, brown=n, buff=b, chocolate=h, green=r, orange=o, purple=u, white=w, yellow=y
21. population: abundant=a, clustered=c, numerous=n, scattered=s, several=v, solitary=y
22. habitat: grasses=g, leaves=l, meadows=m, paths=p, urban=u, waste=w, woods=d

Target feature:

- poisonous?: poisonous=p, edible=e

Number of input nodes after preprocessing: 111

population	hidden nodes	α	num. of initial connection				upper boundaries	patience	threshold
100	5	0.7	15, 5				15, 5	10	0.001
		phase	iterations	w	c_1	c_2	ϵ	epochs	
		first:	20	0.9	2	2	0.5	100	
		second:	50	0.9	4	4	0.5	200	
		third:	30	0.9	0.5	10	0.5	300	

Table 8: Hyperparameter values and phase configurations for the mushroom dataset.

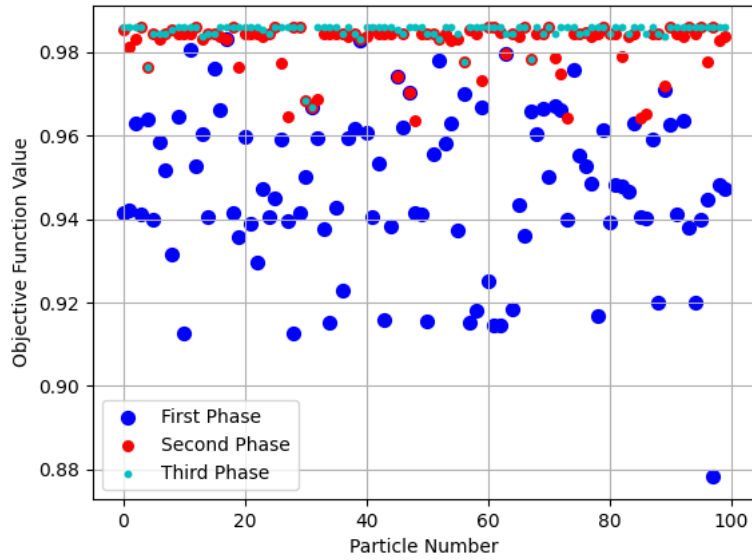


Figure 9: A plot of the development of the particles' personal best scores. The personal best score is plotted (vertical axis) for each particle in the population (horizontal axis) after each phase. This plot was taken from one of the runs of the PSO algorithm with configurations in table 8. One can notice the effect of the high value of the social coefficient in the third phase, where particles exploit the area of the global best position.

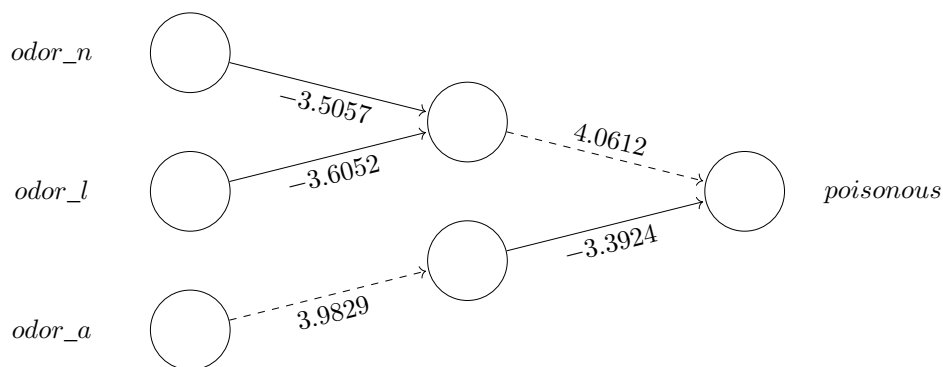


Figure 10: A BAG of the mushroom dataset. This model achieved the highest sparsity among the ten models that were found with configuration 8. It achieved a test accuracy of 98.33%. Noticeable is the dependency on only three values of the feature "oder".

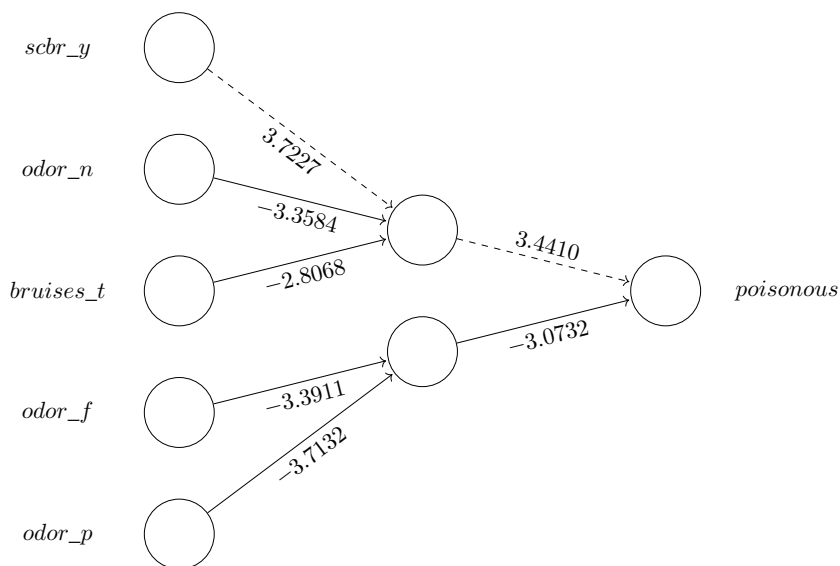


Figure 11: Graph of a sparse MLP model (interpreted as a BAG) for the mushroom dataset that was found using the PSO algorithm with configurations in table 8. This model combined high sparsity and performance (98.52% accuracy on the test subset).

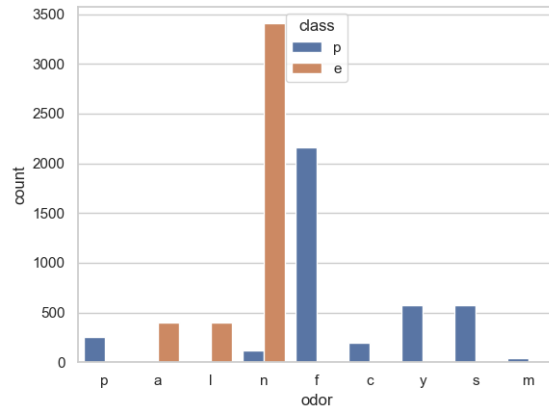


Figure 12: A plot of distribution of classes in the values of the feature "oder" in the mushroom dataset. The count plot illustrates the importance of the value "n" where a relatively big portion of the samples is concentrated with a preponderance of one class "edible".

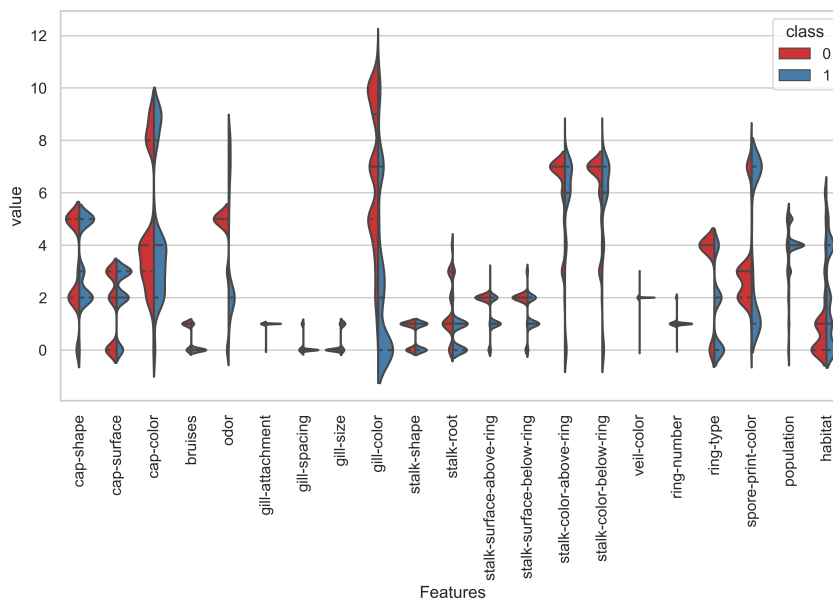


Figure 13: A violin plot of the mushroom dataset after transforming the categorical features into ordinal ones. The plot illustrates the distribution of the two class labels over the features. Here is also noticeable how the "oder" feature peaks at a certain value.

Training subset accuracy	Test subset			
	accuracy	recall	precision	f_1 score
0.9999	0.9994	0.9988	1.0	0.9994

Table 9: Performance results of training a fully-connected MLP on the mushroom dataset with 5 nodes in the hidden layer. The results are averaged over ten runs with different random seeds. Each of the models was trained over 300 epochs and with a learning rate of 0.01.

maximum depth	avg. accuracy on training subset	avg. accuracy on test subset
1	88.71%	88.46%
2	95.44%	95.34%
3	98.53%	98.49%
4	99.39%	99.20%
5	99.86%	99.75%
6	99.97%	99.94%
7	100%	99.97%
8	100%	99.97%

Table 10: Averaged accuracy scores of decision tree models with various depths on the mushroom dataset. The values for each depth were averaged over 10 different runs, each with a unique random seed.

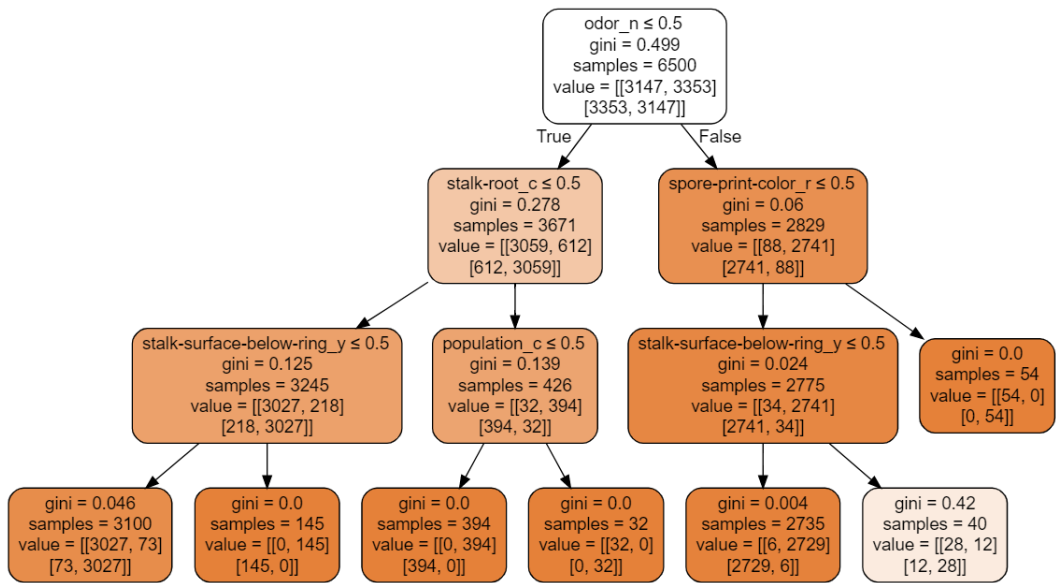


Figure 14: Example of a decision tree for the mushroom dataset with a depth of three (random seed 42). The decision tree reached an accuracy of 98.6% on the training set and 98.21% on the test set. This illustration with this particular depth is chosen due to its comparable simplicity to the found sparse MLP models.

A.3. The Census Income Dataset

Attribute Information

Input features:

1. age: continuous
2. workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked
3. fnlwgt: continuous
4. education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool
5. education-num: continuous
6. marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse
7. occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces
8. relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried
9. race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black
10. sex: Female, Male
11. capital-gain: continuous
12. capital-loss: continuous
13. hours-per-week: continuous
14. native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinidad& Tobago, Peru, Hong, Holand-Netherlands

Target feature:

- Income: >50K, <=50K.

Number of input nodes after preprocessing: 119

population	hidden nodes	α	num. of initial connection				upper boundaries	patience	threshold
50	5	0.8	15, 5				30, 10	25	1e-06
		phase	iterations	w	c_1	c_2	ϵ	epochs	
		first:	25	1	10	10	0.5	500	
		second:	25	0.9	10	10	0.5	750	
		third:	10	0.9	1	20	0.5	1000	

Table 11: Hyperparameter values and phase configurations for the income dataset.

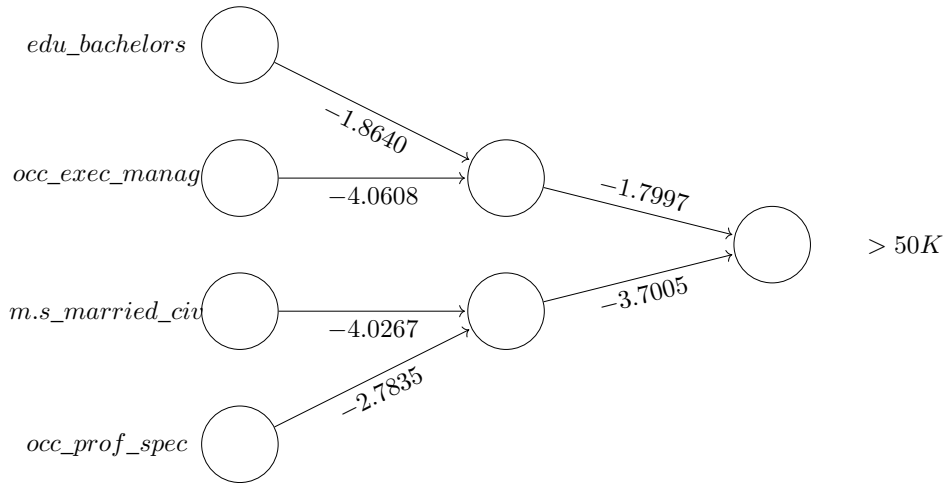


Figure 15: Graph of a sparse MLP model (interpreted as a BAG) for the income dataset that was found using the PSO algorithm with configurations in table 11. The model achieved a test accuracy of 82.67%.

Training subset accuracy	Test subset			
	accuracy	recall	precision	f_1 score
0.8509	0.8410	0.7601	0.7933	0.7740

Table 12: Performance results of training a fully connected MLP on the income dataset with five nodes in the hidden layer. The results are averaged over ten runs with different random seeds. Each of the models was trained over 1000 epochs and with a learning rate of 0.01.

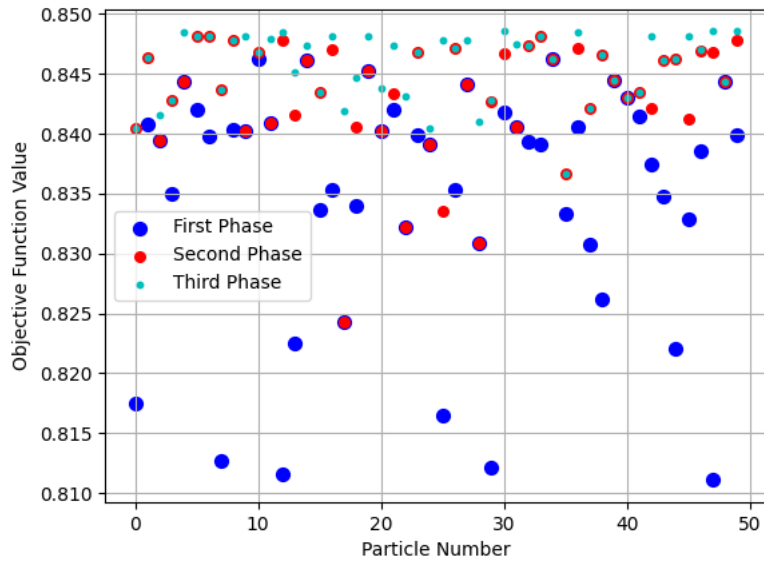


Figure 16: A personal best score plot of a particular run of the PSO algorithm on the income dataset with configurations 11.

maximum depth	avg. accuracy on training subset	avg. accuracy on test subset
1	75.05%	75.30%
2	79.68%	79.97%
3	81.46%	81.62%
4	82.99%	82.99%
5	84.36%	84.28%
6	84.99%	85.03%
7	85.27%	85.14%
8	85.73%	85.2%
9	86.16%	85.39%
10	86.66%	86.66%
11	87.15%	85.3%
12	87.69%	85.23%
30	97.98%	81.50%
50	99.97%	81.25%

Table 13: Averaged accuracy scores of decision tree models with various depths on the income dataset. The values for each depth were averaged over 10 different runs, each with a unique random seed. One can notice the effect of overfitting in trees with a depth that is greater than 10.

A.4. General Results

Dataset	Training	Test				Connections	
	accuracy (std)	accuracy (std)	recall	precision	f_1 score	1.layer (std)	2. layer (std)
Iris	0.8933(0.0359)	0.9433(0.0299)	0.9330	0.9471	0.9346	4(1.54)	2.4(0.66)
Mushroom	0.9765(0.007)	0.9761(0.0059)	0.9759	0.9764	0.9761	4.2(0.87)	2.2(0.39)
Income	0.8103(0.0082)	0.8129(0.007)	0.6870	0.7640	0.7086	6.7(2.14)	2.4(0.68)

Table 14: Summary of performance scores of BAG classifiers implemented on the three datasets. "std" denotes the standard deviation of the variable.

Data set	Training	Test			
	Accuracy (std.) %	Accuracy (std.) %	Precision (std.) %	Recall (std.) %	Number of connections (std.)
Iris data set	97.53 (0.87)	97.34 (2.00)	97.34 (2.00)	97.63 (1.72)	8.20 (1.72)
Adult income	81.31 (0.29)	81.72 (0.29)	76.55 (0.30)	70.66 (1.34)	13.60 (4.22)
Mushroom	97.32 (1.33)	97.12 (1.63)	97.16 (1.63)	97.12 (1.60)	16.50 (5.02)

Figure 17: Summary of performance scores of BAG classifiers averaged over ten runs, including standard deviation, from the genetic algorithm implementation in [24]. This table was taken directly from the paper. Note that the performance values in the table are in percent, where performance values in this study are between zero and one.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Druck-Exemplaren überein.

Datum und Unterschrift:

30.09.2021

A handwritten signature in black ink, appearing to read "M. Waheed Bary". The signature is written in a cursive style with a long horizontal stroke at the end.

Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted hard copies.

Date and Signature: