Imperial College
London

BEng JMC Individual Project Report

Department of Computing

Imperial College of Science, Technology and Medicine

Learning Argumentative Classifiers from Tabular Data

Supervisor:
Dr. Nico Potyka
Prof. Francesca Toni

Author:
Taowen Liu

Second Marker:
Dr. Yingzhen Li

June 21, 2023

Abstract

This research is dedicated to exploring a new category of explainable artificial intelligence algorithms, the argumentation framework classifier, which has the same expressiveness as a multi-layer perception (MLP) while being explainable. On top of previous work by Potyka and Spieler, this paper introduces enhancements using four concepts and adapts a genetic algorithm to facilitate the structure learning problem. The extensions apply to the model include fuzzy input transformation, multiple hidden argument layers, direct attack and support, and joint attack and support. Experiments are concluded on three datasets to assess the classification performance and interpretability. For simple datasets, direct attack and support improves the model's complexity. For complicated datasets, applying multiple extensions is required. An initial feasibility study is also conducted on iterative pruning performing structure learning tasks. Experiments show that structure learned by iterative pruning significantly improves classification accuracy with similar classifier complexity.

Acknowledgments

I would like to express my gratitude to my supervisors, Dr. Potyka and Prof. Toni, for their invaluable support and guidance throughout the project.

I would also like to thank my parents, friends and teachers for their care and support.

Contents

Chapter 1

Introduction

1.1 Background

The utilisation of machine learning algorithms is becoming more involved in decision-making. However, these algorithms' internal mechanism remains a black box for human beings. The need for interpretability and transparency raises concerns about their fairness, reliability and safety. This project aims to explore the field of Explainable Artificial Intelligence (XAI) as emphasised by Molnar et al. [15]. XAI designs algorithms that are explainable and transparent. This includes explaining both the decision made by the algorithm and also the reasoning process by the algorithm[8, 30]. In areas like medical care, self-driving cars and finance, the decision made by models significantly impacts people's life. Thus, it is important to use models that can be ultimately understood. XAI can build people's trust in the machine learning models and their decisions. Moreover, people identify the bias and errors in machine learning models and finally solve them.

While shallow decision trees and logistic regression models are interpretable, they perform less powerfully than ensemble classifiers or neural networks. On the other hand, although a few approaches are employed to explain the prediction process in neural networks [25, 2, 29], their faithfulness in explaining these models still needs to be made.

Argumentation frameworks provide an abstraction of the process of argumentation [9]. Recent work showed a connection between Multi-layer perceptrons and Quantitative Bipolar Argumentation Framework (QBAF) [19, 20]. Unlike other algorithms, utilising MLP-based semantics of argumentation framework allows for faithfully explaining the internal work of MLP models. It is deemed that argumentation frameworks can be built to create human interpretable classifiers through this spoken connection. A QBAF-classifier offers the same expressiveness as an MLP, leading to more robust performance.

MLPs usually consist of densely connected layers involving a huge number of neurons. Explaining the corresponding QBAF of a densely connected MLP becomes challenging because too many arguments are involved. A meta-heuristic algorithm must be employed to learn a sparse MLP model to achieve an interpretable model. Spieler et al. [27] developed a genetic algorithm that was used to fit a two-layer QBAF classifier. A QBAF classifier comprises an input transformation module, a QBAF module for reasoning and classification, and a softmax module. We use Spieler et al.'s work as a baseline model and extend upon that. In a separate study by Bazo [3], particle swarm optimisation was employed.

1.2 Contributions

Our research improves the baseline model using four ideas and adapts the genetic and iterative pruning algorithms to train the classifier structure.

Design extended QBAF classifer

- Improving the input transformation module using fuzzy sets
- Allowing input arguments to attack or support output arguments directly
- Exploring the impact of incorporating multiple layers of hidden arguments
- Enabling the joint attack or support semantics[4, 17]

In the baseline model, the input transformation converts the data into binary values. We employ fuzzy sets in the input transformation to convert values in the range [0, 1]. One idea for improving the classification performance is to increase the number of layers in the QBAF classifier. More layers of hidden arguments might capture argumentation involving more steps. We investigate the impact of increasing the depth on the model's interpretability and classification performance. In the baseline model, the input arguments only affect the output arguments through hidden arguments, increasing the model's complexity and reducing the explainability. To solve this problem, we extend the baseline model by allowing input arguments to attack or support output arguments directly. Joint attack and support[4, 17] can capture the idea that a single argument is insufficient for defeating another argument, but multiple arguments can. Joint attack and support generalise the graphical structure of the model into a hyper-graph, making it more expressive. Experiments show that fuzzy input, direct attack and support, and joint attack and support improve the model's interpretability.

Extend structure learning genetic algorithm

Learning a QBAF classifier incorporates a structure learning problem and a parameter learning problem. Previous works[27, 3] explored using the genetic algorithm and PSO algorithm to perform structure learning. We adapt the genetic algorithm to accommodate the extended QBAF classifier structure learning problem.

Explore iterative pruning for structure learning

Iterative pruning is commonly used to decrease memory usage or improve inference time. This research explores the possibility of using iterative pruning to learn the sparse QBAF classifier structure. Experiments show that iterative pruning could achieve significantly higher accuracy on the mushroom dataset and faster runtime than the genetic algorithm.

Experiment and evaluation

We experiment on three datasets and evaluate our model's precision, accuracy, recall, and f1-score. We also test the combination of different extensions. We use the complexity of the model and graphical illustration to assess the interpretability of the model.

1.3 Overview

We explain the necessary basics of our research in Chapter 3. Chapter 4 discusses the design for the four enhancements we applied to QBAF classifier. Chapter 5 shows the genetic and iterative pruning algorithms' implementation. Chapter 6 shows our experimental results and some graphical visualisation of learned QBAF.

Chapter 2

Ethical Issue

The data used for this research include the adult income dataset from UCI [1], which contains public information. However, this is a long-established open dataset with no risk of identifying individuals.

Recently, machine learning algorithms are more and more complicated these days. Decisions made by these algorithms might have bias and can lead to safety issues. Moreover, these algorithms are intricate to be understood by humans.

For our research topic, explainable AI (XAI) improves the transparency and interpretability of machine learning algorithms. XAI algorithms can be helpful when addressing bias and discrimination, helping decision-making using machine learning, fair and just. However, XAI could mislead people into thinking the AI algorithm is more transparent and trustworthy than they are.

Also, XAI cannot alleviate the problem of privacy and personal information leaking. Thus, these factors need to be considered when applying XAI algorithms.

Chapter 3

Previous work

We explain the basics of quantitative bipolar argumentation frameworks (QABF), QBAF classifier, genetic algorithm to learn QBAF classifier by Spieler et al., fuzzy sets, t-norm and joint attack and support. QBAF and MLP-based semantics are the theoretical bases for our project. The previous work by Spieler et al. [26], a genetic algorithm to learn the QBAF classifier, is the baseline algorithm we compared with. Fuzzy sets, t-norm, and joint attack and support are the theoretical background for our extension.

3.1 Quantitative Bipolar Argumentation Framework (QBAF)

The QBAF Classifier[19, 7, 20, 22] is the primary method we are interested in for this project. We use MLP-based semantics to build a QBAF classifier.

The Argumentation Framework(AF) family[9, 19] abstracts the process of argument. A QBAF is a special kind of AF. A QBAF has a graphical structure, such that the nodes represent arguments, and edges represent the attack or support relations between the arguments. An argument can be accepted to a certain degree. Each argument has a base score. The strength value is determined through argumentation, which is specified by the attack or support relation of the graph.

Example 3.1.1 *The left figure 3.1 shows a graphical structure of QBAF that abstracts the argumentation in choosing to buy or sell a company's stock. A1, A2, and A3 are arguments from experts. A dashed line means support, and a solid line denotes attack. Buying and selling are conflicting arguments. A1 supports selling. A2 and A3 support buying and attack A1.*

In our research, we let the acceptance value of QBAF be in the range $[0, 1]$.

Definition 1 *QBAF: we define a QBAF as a tuple* (A, Att, Sup, β).

- *A is the set of all arguments.*

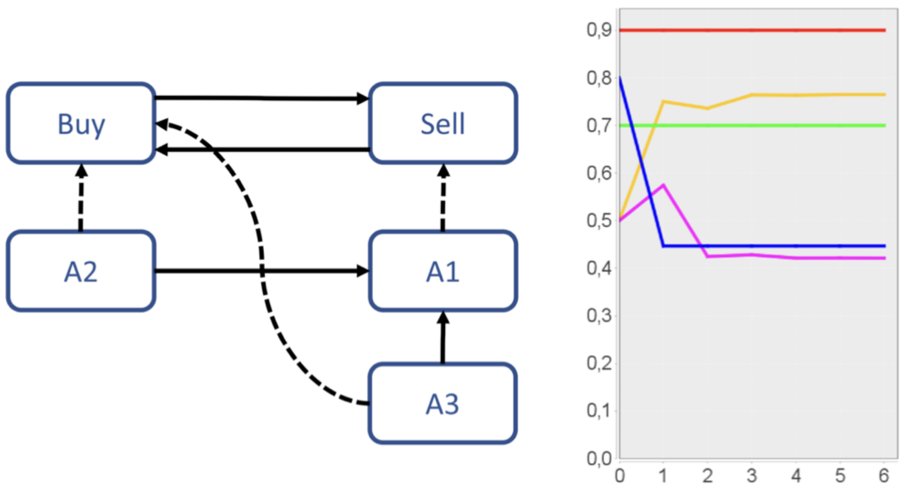


Figure 3.1: example show how strength value of argument being calculated [27]

- *Att is a set of binary-relation over set A, representing the attacking relations between the arguments.*
- *Sup is also a set of binary-relation over set A, representing the support relations between arguments.*
- *β is a function over A, $\beta : A \to [0, 1]$, this function assign base score to the arguments.*

We evaluate the semantics of a QBAF using the interpretation of QBAF[19], which assigns an acceptance score to each argument.

Definition 2 *Interpretation of QBAF: suppose we have a QBAF (A, Att, Sup, β), an interpretation is a function $\sigma : A \to [0, 1] \cup \{\perp\}$. And if $\sigma(a) \in [0, 1]$, $\sigma(a)$ is just the acceptance of the argument a. If $\sigma(a) = \perp$ for some $a \in A$, σ is called partial. Otherwise, what we call the QBAF is fully defined.*

For a cyclic QBAF, we often follow an iterative process to assign acceptance value using the influence ι and aggregation functions α, as suggested in [16]. The process of iterative evaluation typically involves three steps:

- Initialise: The arguments are assigned with base score.
- Update: The strengths of the arguments are adjusted based on the argumentation outcomes. The aggregation function α decides how to summarise the information from the neighbouring argument. The influence function ι defines how the summarised information affects the argument's strength value considering the base score involved. If another argument attacks an argument, the strength of this argument decreases, and if other arguments support it, the strength of this argument increases.

- Termination: The iterative evaluation process is terminated when the strength of the argument converges. If the strength value fails to converge, the interpretation of the QBAF is said to be partial.

For a cyclic QBAF, convergence is not always guaranteed. However, we focus on acyclic QBAF structures and the MLP-based semantics of these structures.

Example 3.1.2 *The right figure 3.1 shows the iterative updating of the strength of these arguments. The colour is represented as Buy (yellow), Sell (violet), A1 (blue), A2 (green) and A3(red)[16].*

3.2 Multi-layer Perceptron

A Multi-layer Perceptron (MLP) is a machine learning algorithm. A previous work[20] showed that an MLP could be regarded as a QBAF, which can be explained and understood by human beings. An MLP has an input layer, multiple hidden layers, and an output layer. One layer in MLP can be represented as the following.

$$h_k = \sigma(\sum_{i=0}^{N} w_{ki} r_i + b_k)$$

where $[r_1, r_2 \cdots r_n]^T$ are activation value from the previous layer. $\{w_{k1}, w_{k2} \cdots w_{kn}\}$ are the weight for k-th neuron in this layer. b_k is the bias of this neuron. σ is the activation function from $\mathbb{R}$ to $[0, 1]$.

An MLP is a composition of all the layers,

$$f(\boldsymbol{x}) = (l_n \circ \cdots \circ l_2 \circ l_1)(\boldsymbol{x}) = \boldsymbol{y}$$

where it is a composition of layer function $l_1, l_2 \cdots l_n$. On classification tasks specifically, the output layer's activation function is usually the softmax function.

$$\text{softmax}(\mathbf{x}) = \frac{\exp(x_i)}{\sum_{j=1}^{N} \exp(x_j)}$$

3.3 MLP-based semantics for QBAFs

Potyka [20] showed that an MLP could be regarded as a special kind of QBAF, which can be explained using the argumentation framework. On the other hand, by choosing appropriate aggregation and influence functions, interpreting a layered acyclic QBAF can be regarded as evaluating MLP. A positive weight w_{ki} denotes that the input argument supports the output argument, whereas a negative weight w_{ki} indicates that the input argument attacks the output argument.

The initial strength value of an argument is translated into the bias in MLP. The activation function and the bias term define the aggregation and influence functions. The strength value of arguments can be regarded as the output of a neuron applied with activation [20, 16].

Definition 3 *edge-weighted QBAF. An edge-weighted QBAF is a tuple (A, E, β, ω) where*

- *A is the set of arguments.*
- *E is the set of all edges in the graph, which is a binary relation over the set A.*
- *β is a function $\beta : A \to D$, which assign a base score for each node.*
- *ω is a function $\omega : E \to [0, 1]$ which assign a weight for each edge.*

When we want to interpret an MLP using edge-weighted QBAF, we choose the aggregation function and influence function as the following [20].

- Aggregation $\alpha^{(i+1)}(a) = \sum_{(b,a)\in E} \omega(b, a) \cdot s^{(i)}(b)$
- Influence $s^{(i+1)}(a) = \phi(ln(\frac{\beta(a)}{1-\beta(a)} + \alpha^{(i+1)}(a)$, where $\phi(x) = \frac{1}{1+exp(-x)}$ is the logistic function.

3.4 QBAF classifier

In [24, 22], Potyka et al. used layered QBAF to perform classification tasks because it has the same expressiveness as MLP while remaining to be explainable. In a QBAF classifier, the arguments in the first layer are input arguments, and the arguments in the last layer are output arguments. Any arguments in layers in between are hidden arguments (or meta arguments). The meta-arguments represent high-level argumentation results of the input arguments.

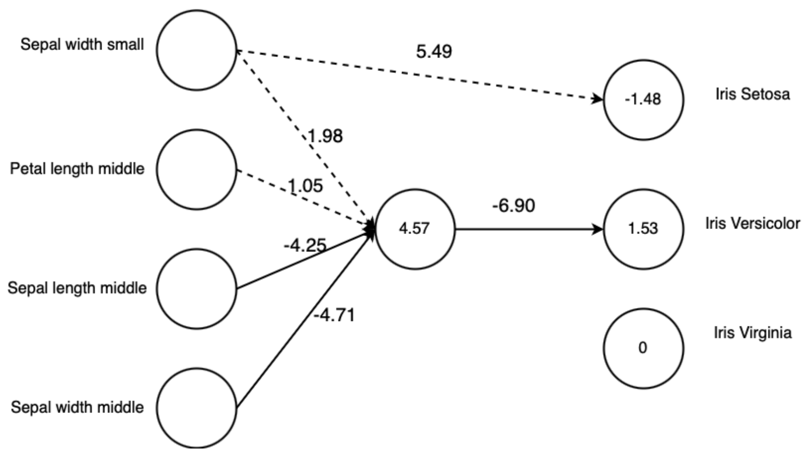


Figure 3.2: Example of QBAF classifier, dashed line denotes support and solid line denotes attack. At the inference stage, the input arguments are initialised with 0 and 1 accordingly.

For a QBAF classifier, the structure of QBAF, the base score for meta-arguments and output arguments, and the edge weights are determined at the training stage. At the inference stage, the base score for the input arguments is initialised with an input feature vector, a vector that comprises values in the range $[0, 1]$. Under the MLP-based semantics, the strength values of the arguments are evaluated layer by layer. The strength values of output arguments are the classification results.

However, for a classification task, the inputs are either numerical or categorical. To employ the QBAF for reasoning, we must transform categorical or numerical data into values in the range $[0, 1]$. In addition, we compose the QBAF with a softmax function to make our QBAF classifier a probabilistic classifier.

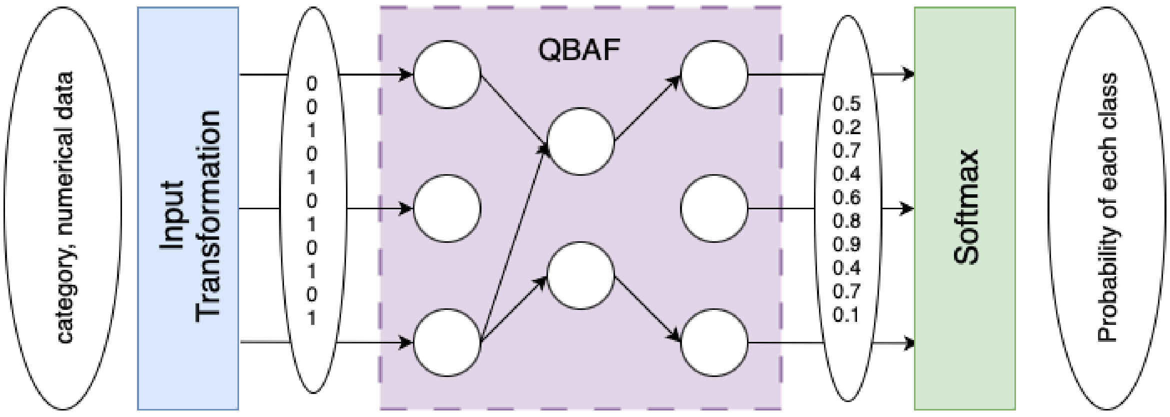


Figure 3.3: QBAF classifier structure

3.5 Learn a QBAF classifier

Learning the QBAF classifier[22] involves two problems: the structural learning problem and the parameter learning problem. The edge weight and base score are learned through gradient descent base algorithms. The parameter learning problem is well-studied. We are more interested in the structure learning problem because a densely connected QBAF is hard to be understood. Learning a sparse structure with fewer connections is essential for interpretability. In [27], Spieler et al., a genetic algorithm is used to learn the structure of QBAF. In a separate study by Bazo [3], a particle swarm optimisation (PSO) algorithm is used to perform the structure learning problem.

The genetic algorithm utilises a pool of candidates to learn the structure of the QBAF classifier. Every QBAF classifier in the population (candidate pool) is encoded into a chromosome. A chromosome is a tuple of adjacency matrices which represents the structure of QBAF. In each generation, the algorithm evaluation the fitness of each individual. Then it selects individuals based on their fitness. The algorithm crossover and mutate the candidate to generate a new population. In the next generation, the new population is used. The algorithm terminates when either a satisfactory solution is found or the maximum number of generations have passed.

The fitness function comprises an accuracy term and a sparsity term. The sparsity of the QBAF classifier corresponds to its explainability. A sparser structure is easier to be explained.

$$\text{fitness} = (1 - \lambda)\text{accuary} + \lambda\text{sparsity}$$

and the sparsity is determined by

$$1 - \frac{\text{number of connections}}{\text{number of all possible connections}}$$

Algorithm 1 Genetic Algorithm

```
1: population ← initialise(N)
2: while The termination condition is not reached: do
3:     mating pool ← select(population)
4:     offspring ← recombine(mating pool)
5:     offspring ← mutate(offspring)
6:     population ← replace(population, offspring)
7: end while
```

3.6 Fuzzy Set

A fuzzy set [28, 12] is a generalisation of a traditional set. A fuzzy set is defined by its membership function $f : A \to [0, 1]$. The membership function assigns membership value in $[0, 1]$ to objects. Logical operations can also be generalised, such as inclusion, union, and intersections. A fuzzy set with a membership function defined as $f : A \to \{0, 1\}$ degrades to a classical set.

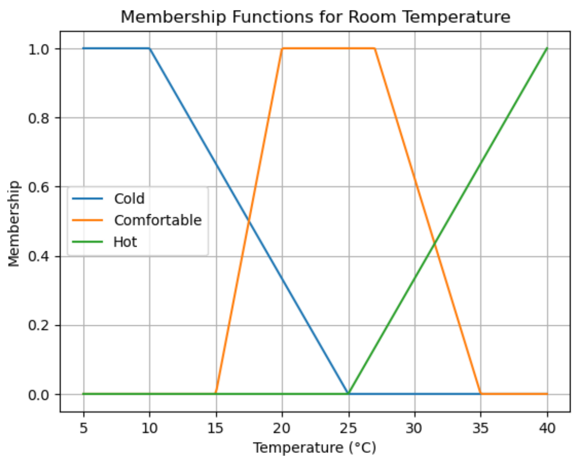


Figure 3.4: Fuzzy sets represents room temperature. This graph shows the membership function of the fuzzy set. There can be a temperature that some people think is cold, but others are comfortable.

Example 3.6.1 *In figure 3.4, we can apply fuzzy logic to room temperature in an AC system. We can define three fuzzy sets, cold, comfortable, and hot. For cold, membership function* f_{cold} *is 1 on* $(-\inf, 10]$ *celeuis, and decrease linearly to 0 on* $(10, 25]$ *celeuis. It remains to be 0 on* $(25, +\inf)$*. Similarly, the comfortable membership function has a trapezoid shape that peaks at* $[20, 27]$.

We can generalise logical operations on a fuzzy set like a classical set. We can define intersection, union, and complement for the fuzzy set. One possible definition for union operation on fuzzy sets is the maximum operation of their membership functions. One possible definition of the union of the two sets $A \cup B$ as a set with membership function $h(x) = max(f(x), g(x))$. f and g are membership functions of A and B.

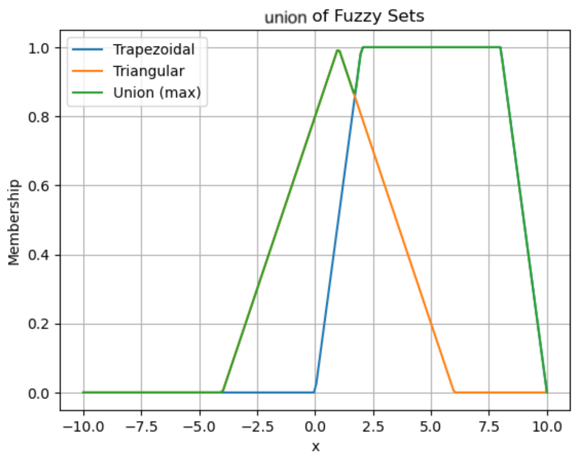


Figure 3.5: Union of the fuzzy sets. The green line is the maximum of the yellow and blue lines, and this membership function can represent the union of the two sets.

3.7 T-norm

T-norm, or the triangular norm, is a binary operation used in fuzzy set theory[12]. It is a generalisation of the notion of intersection in classical set theory. Common t-norm examples include minimum t-norm and product t-norm.

Our research represents joint attack and support[4, 17] using the product t-norm. Let fuzzy set A and B have membership function f and g again. Product t-norm can be expressed as $h(x) = f(x) \times g(x)$. For multiple fuzzy sets $A_1, A_2 \cdots A_n$ with membership function $f_1, f_2 \cdots f_n$ we can further extend the definition of t-norm to $h(x) = \prod_{i=1}^{n} f_i(x)$.

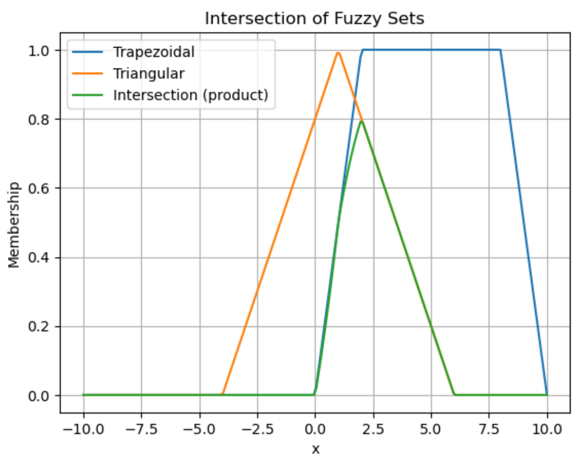


Figure 3.6: Intersection of Fuzzy Sets. The green line is the maximum of the yellow and blue lines, and this membership function can represent the intersection of the two sets.

3.8 Hypergraph and collective attack & support

Joint (collective) attack and support[4, 17] is a generalisation of the argumentation framework. Joint attack and support can capture the idea that a single argument is insufficient for defeating another argument, but multiple arguments can. Joint attack and support allow a set of argument attack or support another argument. This generalises the graphical structure of QBAF to a hyper-graph. In classic graph theory, an edge can only join two vertices. Whereas in a hypergraph, an edge can link any number of vertices.

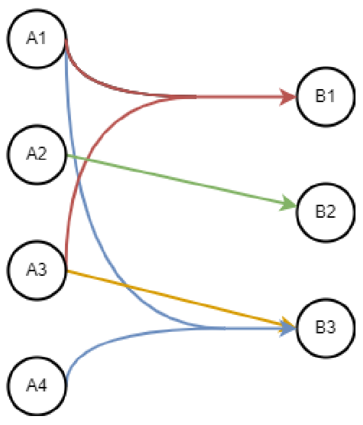


Figure 3.7: Joint attack and support, each colour represents a joint attack or support relation

Example 3.8.1 *Consider a scenario involving bank credit evaluation. Debates related to income status and savings amounts influence loan-granting decisions. A single argument of income status or having savings is not enough to support the argument for a loan. However, if there is joint support between income status and the amount of savings, it is sufficient for granting a loan.*

3.9 Regularisation

Regularisation is used in training neural networks to prevent overfitting and encourage sparsity. This is done by adding a penalty term in the loss function about the parameter.

L1 regularisation[18] adds the absolute value of the weights to the loss.

$$L_{\text{l1}} = L + \lambda \sum_{i=1}^{n} |w_i|$$

L2 regularisation [6] adds the squared weight to the loss.

$$L_{\text{l2}} = L + \lambda \sum_{i=1}^{n} w_i^2$$

3.10 Iterative pruning

As discussed before, learning a QBAF classifier, there are two problems to solve. One is to learn the parameters, which is achieved using gradient-descent-based algorithms. The other is finding the structure of the QBAF classifier. Spieler et al. [27] and Bazo [3] implemented genetic algorithm and PSO algorithm to solve. This research experiments with a new iterative pruning to learn the structure of the QBAF classifier.

Iterative pruning is commonly used to reduce the complexity of neural networks, reducing inference time and memory consumption without compromising the model's performance. This algorithm is deployed in resource-constrained fields like mobile devices and embedded systems.

”Optimal Brain Damage” [13] LeCun et al. introduces the idea that ”removing unimportant components from neural networks may not harm the performance while producing a smaller model”. They used second-order derivatives as a criterion to remove components from the neural network. The smaller the second-order derivative is, the less critical the parameter is.

The ”Lottery Ticket Hypothesis”[10] shows that a dense randomly-initialised network contains smaller subnetworks. When these smaller subnets are independently trained, they can achieve almost the same performance as the larger network. In other words, the smaller subnets made the most contribution to the larger network. In addition, it suggested that compared to the original net, the smaller subnet usually only has 10-20% of the size.

Li et al. [14] used the absolute value of the weights as the criterion to prune neural networks, rather than first-order or second-order derivative. Their research shows that the magnitude of the weights can indicate the importance of the parameter. This approach can reduce the training time in iterative pruning.

Another study by Castellano et al. [5] use iterative pruning to determine the optimal size of MLP. Their method trains a larger-than-necessary network. Then they apply the iterative pruning algorithm to remove the redundant structure in this large MLP. Finally, their method can find a smaller network which has similar performance as the large model.

However, in previous studies, their goals are reducing memory usage and improving inference speed. The pruning ratio is usually below 90%. We would like to apply iterative pruning at an extreme level, to learn a MLP (or QBAF) that can be understood.

Chapter 4

The extended QBAF classifier Design

The QBAF classifier comprises three total modules: input transformation, QBAF, and output transformation (softmax). We extended the input transformation and also the QBAF structure with 4 ideas.

For the input transformation, we used fuzzy sets and membership functions to convert the numerical data into fuzzy argument vectors, which comprise value in $[0, 1]$.

For the QBAF structure, we designed three extensions, involving multiple layers of hidden arguments, direct attack and support, and joint attack and support. The ideas we extend the QBAF with are listed below.

- Improving the input transformation module using fuzzy sets
- Allowing input arguments to attack or support output arguments directly
- Exploring the impact of incorporating multiple layers of hidden arguments
- Enabling the QBAF classifier to express the joint attack and support semantics

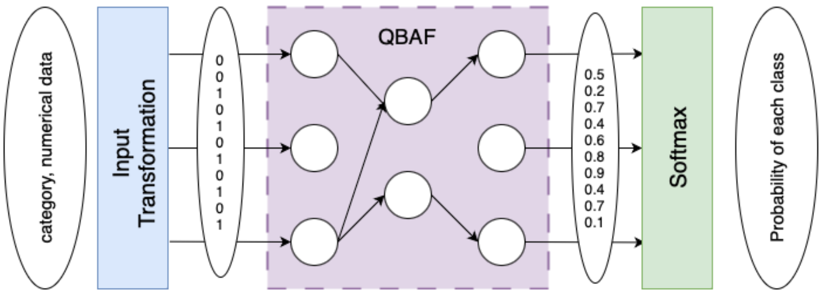


Figure 4.1: QBAF classifier structure

4.1 Input Transformation

In the baseline model[27], the input transformation converts categorical and numerical data into $\{0,1\}^M$ vector. The baseline approach to transforming the numerical data is splitting numerical features into non-overlapping intervals and generating a one-hot vector based on which bin the values fall into. However, the input arguments for the QBAF classifier can be any value in the range $[0,1]$. (as shown in figure 4.1), This transformation converts the input to $\{0,1\}$. Therefore, it may potentially reduce the performance of the QBAF classifier.

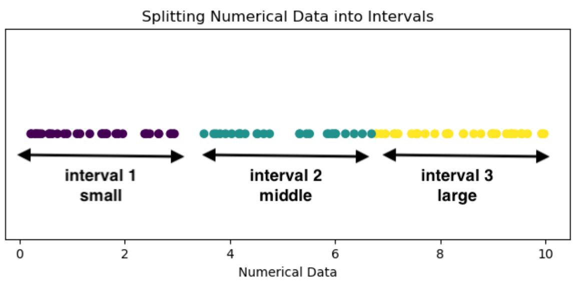


Figure 4.2: Baseline method of numerical feature conversion

We would like better to utilise the QBAF classifier's expressiveness, feeding the QBAF classifier with an input value in the range $[0,1]$. However, this is not interpretable if we naively linearly map the input value into $[0,1]$. The membership function is a concept in fuzzy sets and fuzzy logic, which can capture the vagueness and ambiguity inside data while being interpretable. Our extension defines fuzzy sets for each numerical feature. For each fuzzy set, there is a membership function associated with it. Each numerical value is transformed into a fuzzy argument vector. For categorical data, the input transformation remains to be the same as the baseline model.

Example 4.1.1 *In the iris dataset, a petal length of 4.3 cm is transformed into a vector* $[0,0,0.36,0.64]^T$. *This vector represents it doesn't belong to the "small" or "smaller middle" fuzzy set. The strength belongs the "larger middle" fuzzy set is 0.36. And the strength belongs the "large" fuzzy set is 0.64.*

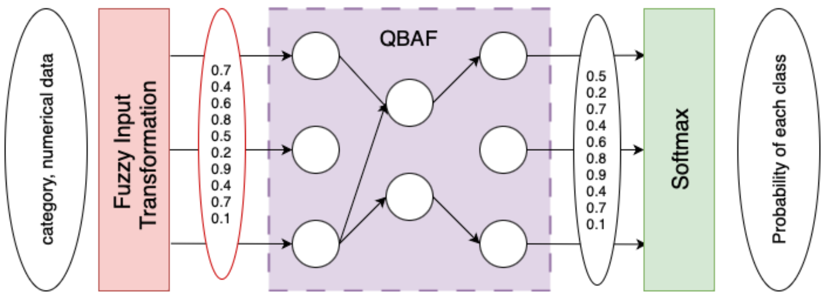


Figure 4.3: The extended input transformation, the inputs to QBAF are in range $[0,1]$ which are labeled in red

The membership function we used is a "triangular" membership function. Figure 4.4 shows four membership functions: "small," "smaller middle," "larger middle," and "large."

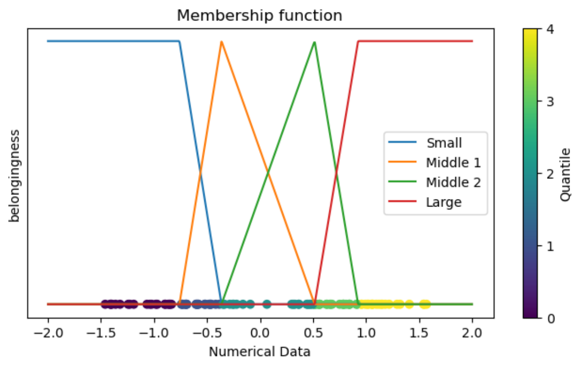


Figure 4.4: Proposed method converts of numerical feature conversion

The membership function can be defined using three parameters, a, b, and c. a and b denotes the boundaries, and c denotes the peak. a, b, and c are determined by quantiles of the underlying data, which we will further discuss in the next section.

$$\mu_{[a,b,c]}(x) = \begin{cases} 0 & x \leq a \\ \frac{x-a}{b-a} & a \leq x \leq b \\ \frac{c-x}{c-b} & b \leq x \leq c \\ 0 & x \geq c \end{cases}$$

We define the membership function slightly differently on the leftmost and the rightmost interval. We define the membership function on the leftmost interval to be

$$\mu_{[b,c]}(x) = \begin{cases} 1 & x \leq b \\ \frac{c-x}{c-b} & b \leq x \leq c \\ 0 & x \geq c \end{cases}$$

This can be understood as the smallest values are assigned with a belongingness of 1 to the fuzzy set "small". A similar expression is used for the rightmost membership function.

4.2 Multiple layers of hidden arguments

In the baseline model, the QBAF only has one layer of hidden arguments. However, in the real world, argumentation can involve a deeper chain of logical structure.

Example 4.2.1 *In an argumentation about climate change, the baseline algorithm may generate a simple QBAF focusing on shallow arguments, such as factory emissions* *leading to global warming. However, in reality, global warming involves multiple layers of logic. For instance, factory emissions contribute to greenhouse gas levels, thawing ice caps. This lowers the reflection of solar energy, subsequently affecting the climate system.*

One natural approach is adding more layers to the baseline model, which can capture deeper levels of argumentation. This kind of structure is further discussed by Potyka [21], denoted as Deep Classification BAG. Moreover, deeper neural networks perform better in practice than shallow ones because they have more non-linearities. However, a deeper argumentation framework might not be explainable. Therefore, we will evaluate these two factors' trade-offs in the experiment section.

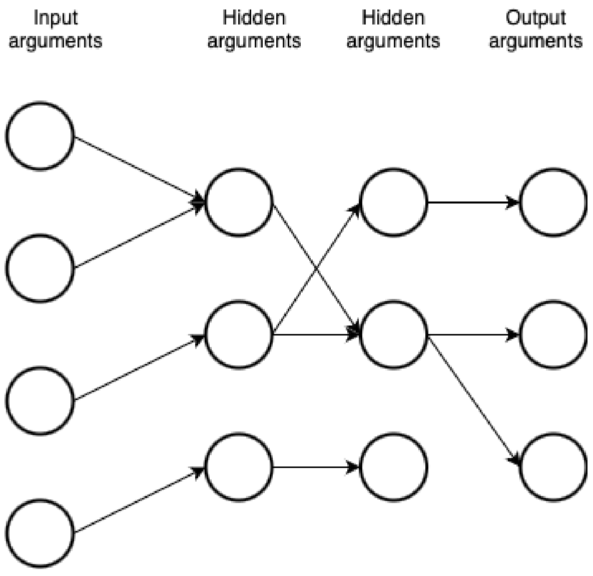


Figure 4.5: Argumentation framework with multi-layer of hidden layers

4.3 Direct attack and support

The baseline model uses a layered QBAF. The relationship between input and output arguments is indirect. Input arguments affect output arguments through hidden arguments. However, while analyzing the graphical structure of the baseline QBAF classifier, it is often observed that an input argument solely influences a single hidden argument, subsequently affecting only one output argument.

Example 4.3.1 *This can be seen in figure 4.6. The connection labelled in red can represent the scenario where the argument "this sample belongs to the class iris versicolor" is supported by an intermediate argument, which is attacked by the input argument "sepal length ≥ 4.9 cm". In other words, this can be intuitively translated to the input argument "sepal length ≥ 4.9 cm attacks the output argument "this sample belongs to class iris versicolor". The hidden middle argument primarily functions as an intermediate node, transmitting the attack or support from the input argument.*

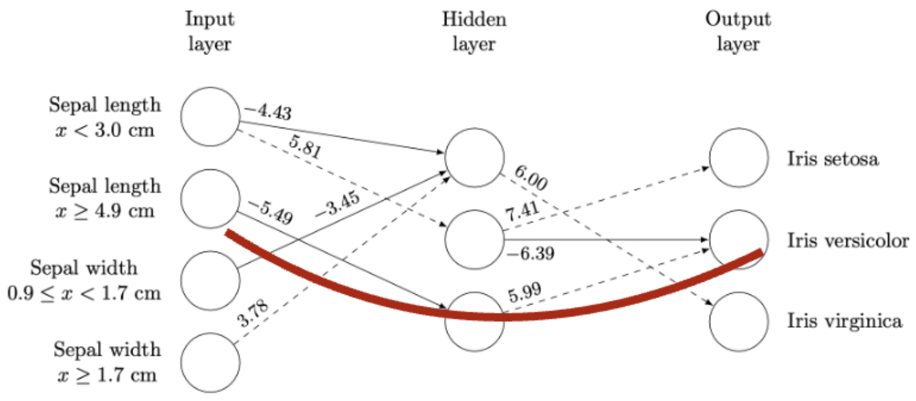


Figure 4.6: Input argument solely affects one hidden argument, subsequently affecting one output argument [27]

A more general acylic QBAF could capture the direct relation between the input and output arguments. This generalisation can be linked to adding skip connections in neural networks, as described by He et al. [11]. However, the mathematical expression for direct attack and support and skip connections have subtle differences. A direct attack or support in our framework has a weight that indicates the strength of the relation. In contrast, a skip connection is an identity map, represented by the equation $g(x) = x + f(x)$.

Example 4.3.2 *In figure 4.7, the input argument "Sepal width small" directly supports the output argument "this sample belongs to class iris setosa".*

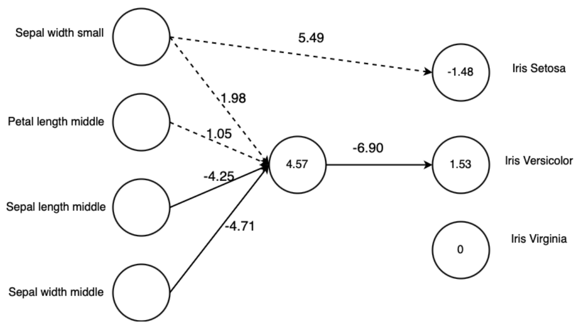


Figure 4.7: An input argument directly attacks an output argument.

The baseline model has three layers of arguments, the input layer, the hidden layer, and the output layer. Our extension mainly affects the interpretation of the output layer arguments. The semantics for the input and hidden layer arguments are the same as the baseline model. Using the MLP-based semantics, the output argument strength value in the QBAF structure in the baseline model can be represented as

$$o_k = \sigma(\sum_{i=0}^{M} w_{ki}h_i + b_k)$$

where o_k represents the strength value of k-th output argument, and h_i represent the strength value of i-th hidden argument. The hidden arguments solely determine the strength value of the output argument.

Our extension adds a direct attack and support term

$$o_k = \sigma(\sum_{i=0}^{M} w_{ki}^{\text{hidden}} h_i + \sum_{j=0}^{N} w_{kj}^{\text{direct}} x_j + b_k)$$

where x_j represents the strength value of j-th input argument. Now both input and hidden arguments can affect output arguments' strength value. This improves the expressiveness of the baseline model and removes the restriction of a "layered" structure. The semantics of direct attack and support is compatible with edge-weighted QBAF, which is discussed by Potyka [23].

4.4 Joint attack and support

The baseline model cannot capture the semantic that a single argument is sufficient to defeat another argument, but multiple arguments are required to defeat another jointly. To represent the joint attack and support semantics, we use t-norms from fuzzy logic. We denote the t-norm operator as $T : [0,1]^n \to [0,1]$, where n represents the number of input arguments to the t-norm. For instance, $T(a_1, a_2, a_3)$ is accepted only when all of a_1, a_2, and a_3 are accepted.

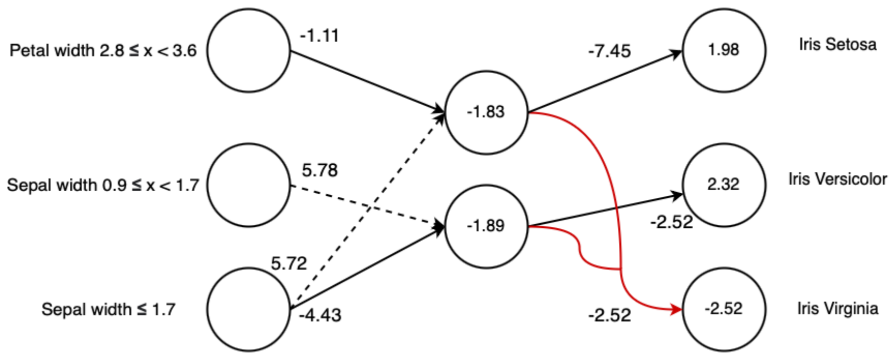


Figure 4.8: Joint attack and support, labelled in red.

Example 4.4.1 *In figure 4.8, the two hidden arguments jointly attack the argument "the sample belongs to the species iris virginia". They can only defeat this argument when both have high strength values.*

Moreover, the introduction of joint attack and support generalises the graphical structure of QBAF into a hyper-graph structure. When multiple argument attacks or support another argument, a hyper-edge connects these arguments. A hyper-edge is an edge that can connect more than two nodes. If the t-norm operator only has one input, this denotes that the hyper-edge degrades to a normal edge. $T(a_1) = a_1$.

We formally define the QBAF with joint attack and support as the following.

Definition 4 *Edge-weight QBAF with joint attack and support: similar to edge-weight QBAF, we can formally define our QBAF with joint attack and support semantic as a tuple* (A, H, β, ω)

- A *is the set of all arguments.*
- H *is an n-ary relation defined over the set* A*, which can be regarded as the directed hyperedge connection of the graphical structure of our model.*
- β *is the function to assign a base score to each argument.*
- ω *is the function to assign weights to each hyper-edge.*

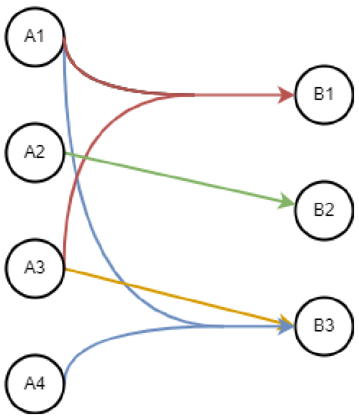


Figure 4.9: Joint attack and support

Example 4.4.2 *In figure 4.9, each colour represents a joint attack or support relation between arguments. The four joint connections are denoted as* $\{A1, A2\} \to B1$*,* $\{A2\} \to B2$*,* $\{A3\} \to B3$*, and* $\{A1, A4\} \to B3$*. .* $\{A2\} \to B2$ *and* $\{A3\} \to B3$ *are degraded to a normal attack or support between arguments.*

Different from the MLP-based semantics of QBAF, where the strength value of h_k is calculated as

$$h_k = \sigma(\sum_{i=0}^{N} w_{ki} r_i + b_k)$$

Extending a normal connection to a joint connection, r_i is replaced with $T(R_i)$. $T(R_i)$ is a t-norm applied to a non-empty subset of the arguments from the previous layer. It is worth noting that R_i can have only one item inside, which degrades to a standard connection $T(R_i) = r$. We can express R_i as follows. Let $I = \{1 \cdots N\}$ be an index set, where N is the number of input arguments. Let $I_1, I_2, \cdots, I_M$ be non-empty distinct subsets of the index set$I$. M is the number of hyper-edge. Then, R_i can be defined as

$$R_i = \{r_{j_1}, r_{j_2} \cdots r_{j_n}\}\text{where}\{j_1, j_2 \cdots j_n\} = I_i$$

We provide the following definition to interpret joint attack and support in the edge-weighted QBAF:

Definition 5 *Interpretation of joint attack support QBAF: The interpretation of joint attack and layered support edge-weighted QBAF can be determined layer by layer.*

$$h_k = \sigma(\sum_{i=0}^{M} w_{ki} T(R_i) + b_k)$$

- $T(R_i)$ *is t-norm applied to a non-empty subset of the strength value of the arguments in the previous layer*
- w_{ki} *is the weight of this* i*-th joint attack and support*
- b_k *is the base score*
- $\sigma : \mathbb{R} \to [0, 1]$ *is the activation function*

In addition, we check that the joint attack and support extension is a generalisation of the edge-weighted QBAF. When each joint connection only connects two arguments, the structure degrades to edge-weighted QBAF. The interpretation of a degraded structure is the same as the edge-weighted QBAF. Also, we check the acceptance value of all the arguments in domain $\mathbb{D} = [0, 1]$. Because we apply the sigmoid activation function $\sigma : \mathbb{R} \to [0, 1]$, this guarantee that all the acceptance value is in the range $[0, 1]$.

Joint attack and support are more complicated than a normal attack and support due to the increased number of inputs. Therefore it is harder to interpret. We explicitly separate the normal connection term and the joint connection term.

$$h_k = \sigma(\sum_{i=0}^{M} w_{ki}^{\text{joint}} T(R_i) + \sum_{i=0}^{N} w_{ki}^{\text{normal}} h_i + b_k)$$

We can use M to specify the desired number of joint attack and support. $|R_i|$ the number of input arguments can also be specified.

4.5 Summary

By integrating these extensions into the QBAF classifier, we made the QBAF supports fuzzy argument inputs with multiple layers of hidden arguments, direct attack and support, and joint attack and support mechanisms.

The enhanced QBAF classifier is comprised of

- Input Transformation (Discrete/Fuzzy)
- QBAF with Hyper-edges (without the restriction to be layered)
- Output Transformation (Softmax)

The input transformation handles the transformation of input arguments, supporting both binary and fuzzy arguments. The QBAF itself is an acyclic QBAF that utilises hyper-edge. The output transformation remains the same as the baseline.

Chapter 5

The extended QBAF classifier implementation

In this chapter, we discuss the implementation of the extended QBAF classifier. Implementing the extended QBAF classifier involves three items.

- Input transformation implementation
- Genetic algorithm implementation
- Iterative pruning implementation

Learning a QBAF classifier involves two problems. A gradient descent algorithm solves the parameter learning problem. Another problem is the structure learning problem. We implemented two algorithms to solve, the genetic algorithm and the iterative pruning algorithm.

For the genetic algorithm, multiple layers of hidden argument and direct attack and support change the chromosome encoding of the genetic algorithm. This leads to further adaption in initialisation, recombination, and mutation in the genetic algorithm. Joint attack and support generalise the graph structure of QBAF to a hyper-graph; no existing library can directly support this operation. We implemented this operator using Pytorch and modified the genetic algorithm to adapt the newly introduced structure.

The iterative pruning algorithm can learn QBAF structure with multiple layers of hidden argument and direct attack and support. However, it cannot be directly applied to the joint attack and support extension. This might be a future work to perform.

5.1 Fuzzy argument input implementation

In the previous section, we discussed using membership functions and fuzzy sets to extend the input transformation. The membership function is defined by

$$\mu_{[a,b,c]}(x) = \begin{cases} 0 & x \leq a \\ \frac{x-a}{b-a} & a \leq x \leq b \\ \frac{c-x}{c-b} & b \leq x \leq c \\ 0 & x \geq c \end{cases}$$

Parameters a, b, and c need to be determined. In this section, we discuss how we determine these parameters. In our implementation, there are three steps in performing fuzzy input transformation.

- find quantile bounds of the data distribution
- determines parameters of membership function
- apply membership function on datapoints, yielding fuzzy arguments vectors

Algorithm 2 Fuzzy input transformation algorithm

```
1: quantiles ← find quantiles(data points)
2: membership functions ← make membership(intervals)
3: for each datapoint in datapoints do
4:     $v$ ← make empty vector()
5:     for each membership function in membership functions do
6:         $v_i$ ← membership function(datapoint)
7:     end for
8: end for
```

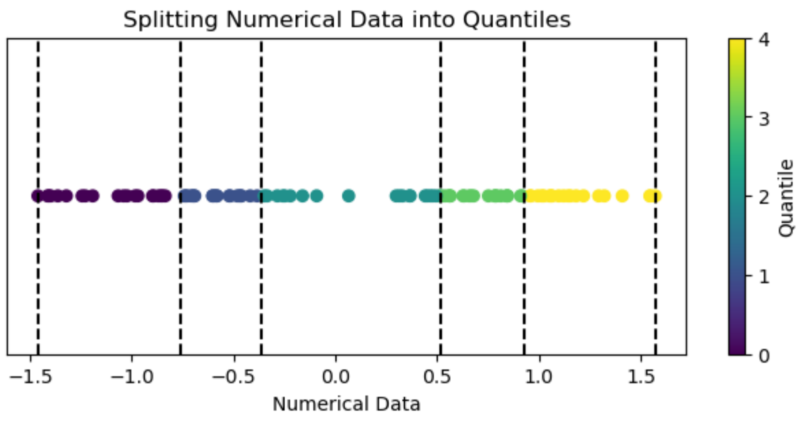


Figure 5.1: Quantile bounds denoted in dashed lines

The first step is to find the quantile bounds for the numerical features. To illustrate this process, figure 5.1 visualises the obtained quantile bounds $q_1, q_2 \cdots q_n$, denoted in verticle lines. We group quantiles and hence define membership functions. Each group consists of three consecutive quantiles. These three values serve as parameters for defining a single membership function. For example, the first group is $\{q_1, q_2, q_3\}$, while the second group is $\{q_2, q_3, q_4\}$. $\{q_i, q_{i+1}, q_{i+2}\}$ defines i-th membership function $f_i(x) = \mu_{[q_i, q_{i+1}, q_{i+2}]}(x)$, as shown in figure 5.2.

$$\mu_{[a,b,c]}(x) = \begin{cases} 0 & x \leq a \\ \frac{x-a}{b-a} & a \leq x \leq b \\ \frac{c-x}{c-b} & b \leq x \leq c \\ 0 & x \geq c \end{cases}$$

However, the leftmost and rightmost membership functions are defined differently. For the leftmost membership function, all the smallest numerical value is assigned with one belongingness of the ”small” fuzzy set.

$$\mu_{\text{smallest}}(x) = \begin{cases} 1 & x \leq q_1 \\ \frac{q_2-x}{q_2-q_1} & q_1 \leq x \leq q_2 \\ 0 & x \geq q_2 \end{cases}$$

A similar situation applies to the ”largest” fuzzy set.

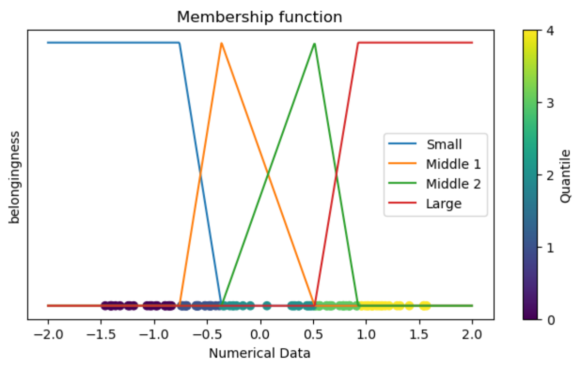


Figure 5.2: Membership functions defined using quantile bounds.

Applying the family of membership functions onto a datapoint would yield a n dimension fuzzy argument vector. We repeat this procedure on each numerical feature in a dataset and concatenate it into a fuzzy arguments vector.

In programming implementation, however, these membership functions are temporary variables which can be optimised. It is possible to directly use the grouped quantiles to find the belongingness of each fuzzy set. The definition of these membership functions can be inlined. Consequently, the code is vectorised hence being more efficient.

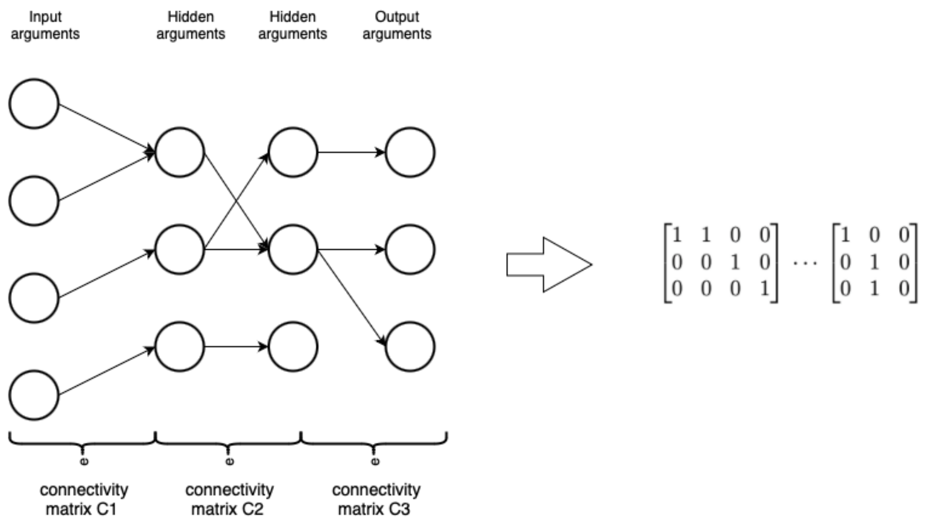
Algorithm 3 Input transformation

```
1: $v \leftarrow$ empty vector
2: for each pair of (left bound, right bound) in quantiles do
3:     Iterate over each datapoint $x$ within [left bound, right bound]:
4:     $v_i \leftarrow$ ($x$ - left bound) / (right bound - left bound)
5:     $v_{i-1} \leftarrow$ (right bound - $x$) / (right bound - left bound)
6: end for
```

5.2 The extended genetic algorithm implementation

5.2.1 Multiple layers of hidden arguments

Implementing multiple layers of hidden arguments mainly changes the chromosome encoding of the genetic algorithm. In the baseline algorithm, the chromosome encoding for a QBAF is two adjacent matrices, denoting the connectivity of the first and the second layer. We adopt a similar approach to accommodate multiple layers of hidden arguments by adding more adjacency matrices. The initialisation, recombination, and mutation functions must be adapted accordingly. Apart from that, the way to calculate sparsity is updated. Similar to the baseline algorithm, for the parameter learning problem, we utilised Pytorch and the sparse linear library to efficiently handle the training and inference of the sparse structure of the QBAF.



$$\begin{bmatrix}1&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}\cdots\begin{bmatrix}1&0&0\\0&1&0\\0&1&0\end{bmatrix}$$

Figure 5.3: Encoding of multiple hidden arguments layers

5.2.2 Direct attack and support

The direct attack and support QBAF can be expressed in the following expression.

Algorithm 4 Direct attack and support implementation

```
1: Compute the hidden layer values:
2: $\mathbf{h} \leftarrow \sigma(W_h\mathbf{x} + \mathbf{b})$
3: Compute the output layer values
4: $\mathbf{out} \leftarrow \text{softmax}(W_{h2}\mathbf{h} + W_d\mathbf{x} + \mathbf{b})$
```

IW_h, W_{h2}, and W_d are sparse weight matrices. We follow a similar approach to the baseline model, using adjacency matrices to encode the QBAF classifier with direct attack and support. The initialisation function, recombination function, and mutation function are updated. Like the baseline algorithm, the direct attack and support can also be efficiently implemented using PyTorch and the sparse linear library.

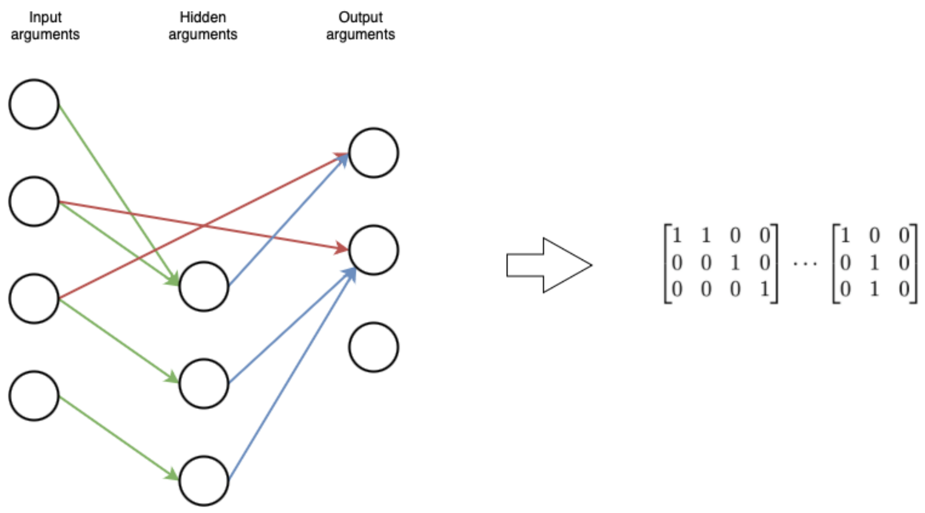

Input arguments
Hidden arguments
Output arguments
$\begin{bmatrix}1&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}\cdots\begin{bmatrix}1&0&0\\0&1&0\\0&1&0\end{bmatrix}$

Figure 5.4: Encoding of direct attack and support

Example 5.2.1 *In figure 5.4, the connections represented by the red, green, and blue lines are encoded using three different adjacency matrices.*

5.2.3 Joint attack and support

Implementing QBAF with joint attack and support is more complicated. This increased complexity is from the hyper-graph structure. To handle the parameter learning problem, we implement a new joint attack and support operator in Pytorch. Additionally, we make necessary modifications to the genetic algorithm to enable learning the joint attack and support structure. These modifications ensure that the learned classifier doesn't degrade to a normal QBAF structure.

To control the number of joint attack and support and prevent degradation, we distinguish the normal connection term and the joint attack and support term. In figure

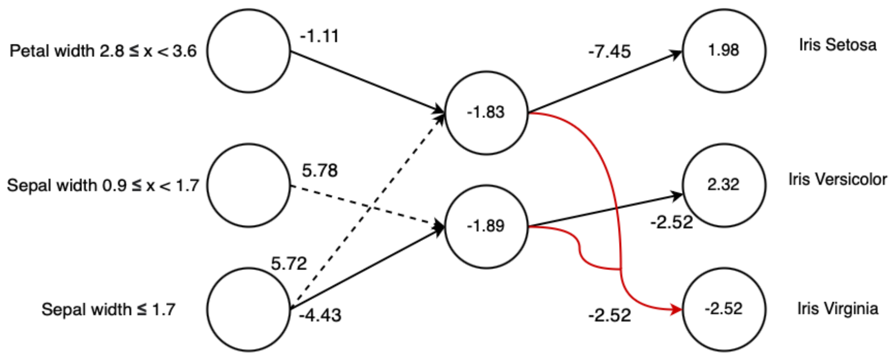


Figure 5.5: An example of joint attack and support. A dashed line represents support, and a solid line represents attack; Joint attack and support are labelled in red, while normal attack and support are labelled in black

5.5, the black connections are standard connections, and the red connection is a joint connection.

$$h_k = \sigma(\sum_{i=0}^{M} w_{ki}^{\text{joint}} T(R_i) + \sum_{i=0}^{N} w_{ki}^{\text{normal}} h_i + b_k)$$

Because the two components are combined using addition, in Pytorch implementation, we can compute each part individually and sum them together. The standard connection term reuses the sparsely connected layer, denoted as f. However, no library can directly support the operation for the joint attack and support term, denoted as j.

Algorithm 5 Joint attack and support implementation

```
1: $\mathbf{z}_{normal} \leftarrow f(\mathbf{x})$
2: $\mathbf{z}_{joint} \leftarrow j(\mathbf{x})$
3: $\mathbf{z} \leftarrow \mathbf{z}_{normal} + \mathbf{z}_{joint} + \mathbf{b}$
4: $\mathbf{a} \leftarrow \sigma(\mathbf{z})$
```

Let's focus on implementing the joint attack and support operator j. In our implementation, we employ the product t-norm operator.

$$T(R_i) = \prod_{r \in R_i} r$$

Pytorch implementation 1

For a joint connection $T(R_i)$ that combines n input argument to attack or support an output argument, we needed to compute the product of the elements of the input vector with indices $i_1, i_2 \cdots i_n$. In our initial implementation of this operator, we directly implemented it using a mask vector to select the arguments in the previous layer. We define a mask vector $mask_a$, the vector with elements at position $i_1, i_2 \cdots i_k$ were set to 1, and the remaining elements were set to 0. Then, we used the Pytorch subscription operator to select the values at these positions. Finally, we applied the Pytorch product operator to the selected item.

Algorithm 6 Joint attack and support Pytorch implementation 1

```
1: results ← zero_matrix
2: for each r in batch do
3:     for each $u_a$ do
4:         $\mathbf{masked\_r} \leftarrow \mathbf{r}[\mathbf{mask_a}]$
5:         $u_a \leftarrow \mathbf{masked\_r}.$ prod()
6:         result$[n_{batch}, a] = u_a$
7:     end for
8: end for
```

Because the number of elements to be multiplied is not fixed, this operation cannot be vectorised. This resuls in that the algorithm cannot complete in a reasonable amount of time.

Pytorch implementation 2

While implementing masked multiplication in Pytroch is challenging, we employ an alternative approach. The masked addition algorithm can be effectively implemented in Pytorch as a matrix multiplication operation. The exponential function exp and logarithm log function can convert multiplication into addition $x \cdot y = exp(log(x) + log(y))$. In PyTorch, the element-wise exponential and logarithm functions are vectorised and efficiently implemented. We transform the input vector into log space, perform masked addition using matrix multiplication, and then transform the result to the original scale by applying the exponential operation.

Algorithm 7 Joint attack and support Pytorch implementation 2

```
1: $\mathbf{x} \leftarrow log(\mathbf{x})$
2: $\mathbf{x} \leftarrow$ mask $\cdot \mathbf{x}$
3: $\mathbf{x} \leftarrow exp(\mathbf{x})$
```

Note that log and exp are element-wise operations.

Chromosome representation

To control the number of joint attack and support connections and prevent the degradation of joint attack and support. We enforce some restrictions on the structure that the genetic algorithm can learn. Users can specify the number of joint connections and a number of arguments that joint connections involve. Considering these restrictions, the chromosome encoding for connections between two layers is defined using two adjacency matrices. The first adjacency matrix represents the normal connection part. The second adjacency matrix represents the joint attack and support connection. The number of rows corresponds to the total possible joint attack and support between the two layers. Each row consists of N columns, where N is the number of arguments in the i-th layer. If the k-th argument in the i-th layer is involved in the j-th joint attack and support connection, C^i_{jk} is assigned a value of 1. Otherwise, it is set to 0.

5.2.4 Refined Fitness Function

Apart from that, we did another modification to the genetic algorithm. The following formula calculates each individual's fitness in the baseline model.

$$\text{fitness} = (1 - \lambda)\text{accuary} + \lambda\text{sparsity}$$

and sparsity is determined by

$$1 - \frac{\text{number of connections}}{\text{number of all possible connections}}$$

We make two discoveries regarding the fitness value. First, all individuals has similar numbers of connections. Secondly, due to the possible number of all connections being usually large (typically around 500-2000), the sparsity term of all individuals is consistently close to 1. Therefore, the contribution from the accuracy term dominated the fitness function, making the QBAF with different sparsity indistinguishable.

To solve this issue, we standardised the sparsity value in the fitness function. We introduced relative sparsity, defined as the following.

$$\text{relative sparsity} = \frac{s_i - min(S)}{max(S) - min(S)}$$

Here s_i denotes the sparsity for i-th individual in the population, and $max(S)$ and $min(S)$ represent the maximum and minimum sparsity in the population, respectively.

Example 5.2.2 *Let's consider a population of 3 individuals, with accuracy* $0.95, 0.96$*, and* 0.75 *and a respective number of connections* $20, 15$*, and* 14*. The maximum number of possible connections in a typical QBAF classifier with 100 input, ten hidden arguments, and two output arguments is 1020. Figure 5.6 shows that individual one and individual two are not distinguishable when varying* λ*. However, figure 5.7 shows that when using relative sparsity term in the fitness function, all the individuals are well distinguished.*

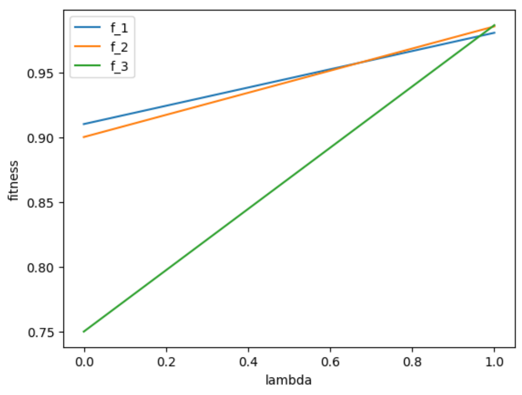


Figure 5.6: Baseline fitness of individuals

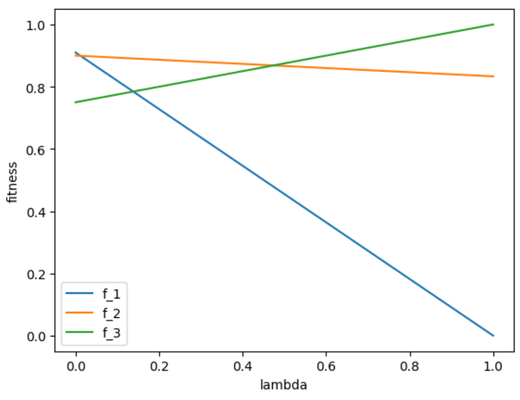


Figure 5.7: Fitness of individual when using relative sparsity

5.3 Iterative pruning implementation

We implement iterative pruning and employ it at an extreme level to learn a very sparse MLP (QBAF) structure. This algorithm can be effectively applied to the baseline QBAF classifier, the QBAF classifier with the direct attack and support, and the QBAF classifier with multi-layer hidden arguments.

Iterative pruning works by iteratively training the model, identifying less helpful components, removing them, and repeating this process.

This process can be summarised in the following steps:

- Initial Training: We begin by training a fully connected QBAF model using the training set, deploying a gradient descent-based algorithm to determine the weights and base score in the model.
- Importance Evaluation: We then evaluate the importance of arguments and relations. We compare the absolute values of edge weights (or the intensity of attack or support). A smaller absolute value of weight indicates lower importance.
- Pruning Phase: We remove the less significant structures identified in the previous step. In our implementation, the number of removed components is a particular proportion defined by a hyper-parameter.
- Model Retraining: Since removing specific structures may harm the model's classification performance. Retrain is necessary for the model to repair its original classification performance. We also apply l2 regularisation in this step to encourage the model to be more sparse. This phase also facilitates the evaluation of component importance in subsequent iterations.
- Iterative Process and Termination: We repeat this procedure until we meet the desired sparsity requirement.

However, this algorithm is unsuitable for learning the joint attack and support structure. We will address this limitation in the Evaluation and Future Work section.

Algorithm 8 Iterative Pruning for MLP

```
1: Input: Training dataset $\mathcal{D}$, MLP model $M$, initial learning rate $\eta$, pruning percentage $p$, desired sparsity $s$, l2 regularisation parameter $\alpha$
2: procedure IterativePruning($\mathcal{D}, M, \eta, p, s$)
3:     while Sparsity of $M < s$ do
4:         Train $M$ on $\mathcal{D}$ using learning rate $\eta$ with l2 regularisation $\alpha$
5:         Compute the absolute value of all weights $|w_i|$
6:         Sort the weights by $|w_i|$
7:         Set the lowest $p$ percent of weights to zero
8:         Set the pruned weight to be not trainable
9:     end while
10: end procedure
```

5.4 Summary

We implements extended QBAF classifier by extending the input transformation and adapting the genetic algorithm with new structures. We also employs iterative pruning algorithm to learn QBAF structure.

- Input transformation implementation
- Genetic algorithm implementation
- Iterative pruning implementation

Chapter 6

Experiments and evaluation

We conduct experiments to evaluate our extended model's classification performance and interpretability. We also provide graphical illustrations of the new extended QBAF structures. There are three groups of experiments.

- The baseline QBAF classifier with a single extension.
- The baseline QBAF classifier with multiple extensions.
- Initial investigation of the iterative pruning algorithm.

The experiments we conducted are listed here.

Experiment	Fuzzy	Multi	Direct	Joint	Algorithm
1	✓				Genetic
2		✓			Genetic
3			✓		Genetic
4				✓	Genetic
5	✓		✓		Genetic
6	✓			✓	Genetic
7			✓	✓	Genetic
8	✓		✓	✓	Genetic
9					Iterative Pruning
10	✓				Iterative Pruning
11		✓			Iterative Pruning
12			✓		Iterative Pruning

Table 6.1: Experiment Configurations

The experiment result shows that fuzzy input transformation, direct attack and support, and joint attack and support can improve the expressiveness of the model, hence improving model's interpretablity without compromising classification performance. The combination of these three extensions also showed promising results. Multi-layer of hidden arguments doesn't show improvement in the model's performance, thus we didn't incorporate this into multiple extensions experiments. The structure learned by iterative pruning has better accuracy than the genetic algorithm and the PSO algorithm. It is more time efficient than the genetic algorithm. However, iterative pruning reached a limit for complicated structures like joint attack and support and very sparse structures with less than eight connections.

6.1 Datasets

The choice of our dataset covers different classification problem scenarios.

- Iris dataset consists of 50 measurements for each class, taken from 3 species of iris flowers: setosa, versicolor, and virginca. It consists of four numerical features: sepal length, sepal width, petal length, and petal width. We conduct our experiment on this dataset to evaluate the classification performance of our model on a purely numerical feature dataset.
- Mushroom dataset describes 22 species of mushrooms using 22 different categorical features, such as cap shape, colour, and odour. Each item in this dataset is either edible (52%) or poisonous (48%). There are 8124 samples in total.
- Adult income dataset is the largest dataset in our experiment. This dataset contains 48842 samples, each with 13 attributes. The primary purpose of this dataset is to classify whether an individual's income is above or below 50k. However, this dataset shows an imbalance issue. 75% of the samples fall into the "≤ 50k" class.

6.2 Evaluation protocols

We evaluate our extended model on three datasets from the UCI Machine Learning Repository[1]. The dataset is divided into training set, validation set and test set with ratios of 70, 10 and 20. We measure the classification performance of the model using precision, accuracy, recall, and f1-score on the test set. Interpretability is assessed using model complexity and graph structure. Model complexity is measured by the of connections. Each experiment was repeated ten times. We report the mean and standard deviation of the metrics.

We first test the capability of single extensions. Then we examine the performance of the combination of the extensions. For each dataset, we also identify the most helpful extension combination. We also did some initial exploration on the feasibility of using iterative pruning to learn the structure of QBAF.

Genetic algorithm evaluation protocol

For the genetic algorithm, we proposed five configurations for the QBAF classifier. We presented the QBAF classifier configurations in the table below. The 'number of hidden arguments' is the number of hidden arguments. 'Number connection1' is the number of connections between input and hidden arguments. 'Number connection2' is the number of connections between hidden and output arguments.

Configuration	number of hidden argument	number connection1	number connections2
1	12	10	6
2	12	6	6
3	8	4	4
4	6	4	2
5	4	4	2

Table 6.2: QBAF classifier configuration

With computational time limitations, optimising hyper-parameters for each configuration was impossible. We decided to tune the hyper-parameter of the genetic algorithm and gradient descent algorithm for each dataset. This is because a smaller population and fewer generations might be enough for a simpler dataset like the iris dataset. In contrast, a more complex dataset needs more. A larger population and generations can yield better results but require longer training. For each dataset, we set a specific population size, sparsity regularisation parameter, the number of generations, etc. This can be found in the appendix.

We considered the following parameters:

- Population size: 20, 30, 40, 50, 100
- Number of generations: 5, 10, 20
- Learning rate: 0.003, 0.01, 0.03

Iterative pruning evaluation protocol

For iterative pruning, the hyper-parameters are the target number of connections (sparsity), the learning rate η, and the regularisation parameter α, the proportion of connections to be pruned in an iteration p. p is set to be fixed at 0.3. We first set a target number of connections. Then we run a grid search to determine the rest of the hyper-parameters.

We considered the following parameters:

- Learning rate: 0.001, 0.01, 0.1
- L2 regularisation parameter α: 0.0001, 0.0003, 0.001

6.3 Baseline

Our study utilised the baseline model by Spieler et al. [27] and Bazo [3]. [27] learn the structure of QBAF using a genetic algorithm, denoted as BaselineG. [3] learn the structure of QBAF using the PSO algorithm, denoted as BaselineP. Connections are the total number of attack and support relations in QBAF.

Dataset	Connections	Accuracy	Recall	Precision	F1
BaselineG(A)	13.60 (4.22)	0.81(0.00)	0.71(0.02)	0.76(0.00)	
BaselineG(M)	16.50 (4.22)	0.97 (0.02)	0.97(0.02)	0.97(0.02)	
BaselineG(I)	8.20 (1.72)	0.97 (0.02)	0.97(0.02)	0.97(0.02)	
BaselineP(A)	6.4 (1.27)	0.81(0.00)	0.68	0.76	0.71
BaselineP(M)	9.1 (2.28)	0.97 (0.02)	0.98	0.98	0.98
BaselineP(I)	6.4 (2.10)	0.94 (0.03)	0.93	0.95	0.93

Table 6.3: Result obtained from baseline

6.4 Single extension experiment result

Experiment	Connections	Accuracy	Recall	Precision	F1
BaselineG(I)	8.20 (1.72)	0.97 (0.02)	0.97(0.02)	0.97(0.02)	
BaselineP(I)	6.40 (2.10)	0.94 (0.03)	0.93	0.95	0.93
Fuzzy (I)	6.90 (2.96)	0.96 (0.04)	0.96 (0.04)	0.97 (0.03)	0.96 (0.04)
Direct (I)	**4.60** (0.97)	0.97 (0.02)	0.97 (0.02)	0.97 (0.02)	0.97 (0.02)
Multi (I)	8.00 (2.05)	0.91 (0.11)	0.91 (0.11)	0.90 (0.15)	0.90 (0.14)
Joint (I)	5.30 (1.06)	0.97 (0.02)	0.97 (0.02)	0.97 (0.02)	0.97 (0.02)
BaselineG(A)	13.60 (4.22)	0.81(0.00)	0.71(0.02)	0.76(0.00)	
BaselineP(A)	**6.40** (1.27)	0.81(0.00)	0.68	0.76	0.71
Fuzzy (A)	11.20 (2.66)	0.81 (0.01)	0.70 (0.03)	0.76 (0.01)	0.72 (0.02)
Direct (A)	8.10 (2.13)	0.81 (0.01)	0.69 (0.03)	0.76 (0.01)	0.71 (0.03)
Multi (A)	3.00 (3.42)	0.77 (0.03)	0.57 (0.09)	0.58 (0.22)	0.53 (0.14)
Joint (A)	9.30 (3.09)	0.81 (0.00)	0.70 (0.02)	0.76 (0.01)	0.72 (0.01)
BaselineG(M)	16.50 (4.22)	0.97 (0.02)	0.97(0.02)	0.97(0.02)	
BaselineP(M)	**9.10** (2.28)	0.97 (0.02)	0.98	0.98	0.98
Direct (M)	10.70 (4.32)	0.97 (0.02)	0.97 (0.02)	0.97 (0.02)	0.97 (0.02)
Multi (M)	14.80 (5.49)	0.97 (0.02)	0.97 (0.02)	0.97 (0.02)	0.97 (0.02)
Joint (M)	11.60 (3.03)	0.97 (0.01)	0.97 (0.01)	0.97 (0.01)	0.97 (0.01)

Table 6.4: Result obtained from a single extension. I, A, and M represent the iris, adult income, and mushroom datasets.

In table 6.4, *Fuzzy* represents fuzzy input argument, *Direct* represents direct attack and support, *Multi* represents multi-layer hidden arguments, and *Joint* represents joint attack and support. The dataset is indicated in parentheses. I, A, and M represent the iris, adult income, and mushroom datasets.

The PSO algorithm generally uses fewer connections to achieve similar accuracy than a genetic algorithm can achieve. The direct attack and support extension performs the best in reducing the number of connections while maintaining accuracy.

Fuzzy input transformation

As discussed earlier, the mushroom dataset only comprises categorical data, which does not apply to this experiment. Compared to the BaselineG, with similar test accuracy retained, the number of connections was reduced for the adult income dataset (13.60 to 11.20) and the iris dataset (8.20 to 6.90). However, with fuzzy input transformation, the genetic algorithm still cannot beat the PSO algorithm on all of three datasets.

Direct attack and support

Direct attack and support can capture a more straightforward relation between the input and output arguments. The result from direct attack and support is promising. With similar accuracy, compared to BaselineG, we achieved less amount of connections on the adult income dataset (13.60 to 8.10), the mushroom dataset (13.60 to 10.70) and the iris dataset (8.20 to 4.60).

This extension is especially helpful for the iris dataset because it is a very simple dataset. The classification result might be directly identified from the input without complex reasoning. Also, this extension beats the PSO algorithm in the iris dataset.

We also illustrate the graphical structure of QBAF with direct attack and support we obtained. The attack relation is represented using a solid line, and the support relation is represented using a dashed line.

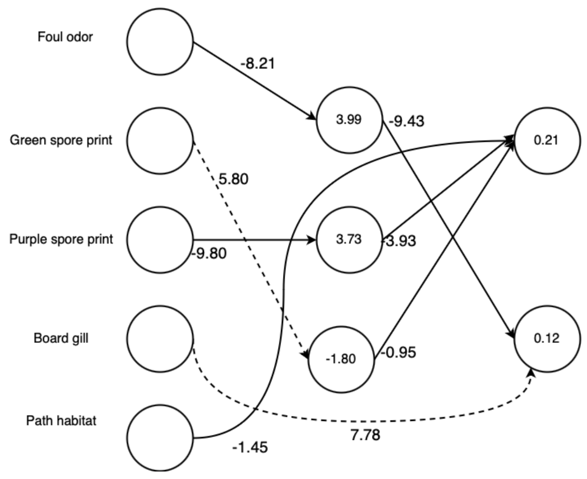


Figure 6.1: Learned QBAF with direct attack and support for the mushroom dataset

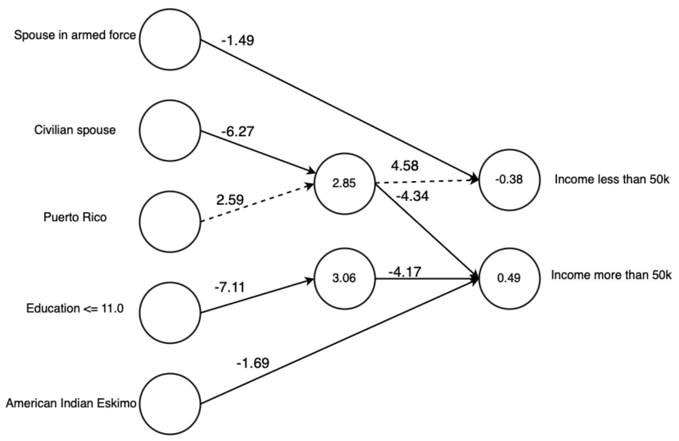


Figure 6.2: Learned QBAF with direct attack and support for the adult income dataset

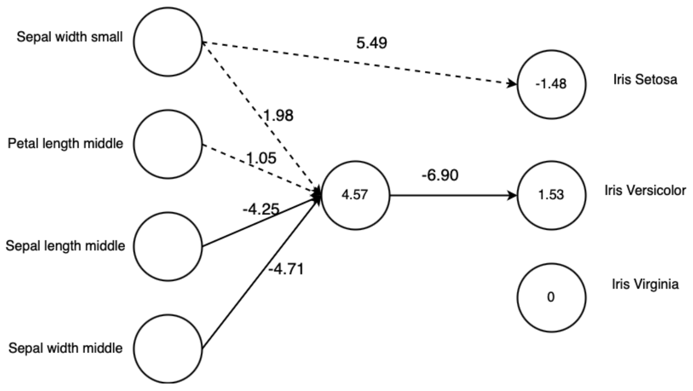


Figure 6.3: Learned QBAF with direct attack and support for the iris dataset

The QBAF classifier learned for the iris dataset, as shown in figure 6.3, can be explained as Iris Virginia is the default classification result. Small sepal width supports a sample to be classified as Iris Setosa. The classification result Iris Versicolor is attacked by a meta-argument. This meta-argument is directly supported by the small sepal width and middle petal length. And it is attacked by middle sepal length and middle sepal width.

Multi-layer of hidden arguments

The result of multiple layers of hidden argument extension is shown to be less effective. In the Adult dataset, which has 75% of data belonging to the class 'income less than 50k', the extended QBAF classifier didn't significantly outperform a baseline classifier that classifies all data as 'income less than 50k'. For the Iris dataset, the extended QBAF classifier achieves an accuracy of 0.91, lower than the baselines.

These results suggest adding extra layers of hidden arguments expands the hypothesis space. The genetic algorithm needs a larger population and more generations to explore the hypothesis space. Because this structure is less effective in our experiments, we didn't include it in multiple extension experiments.

Joint attack and support

The joint attack and support extension shows promising results. Compared to BaselineG, with similar accuracy, we achieved less amount of connections on the adult income dataset (13.60 to 9.30), the mushroom dataset (13.60 to 11.60) and the iris dataset (8.20 to 5.30). The genetic algorithm with joint attack and support wins the PSO algorithm on the iris dataset.

Compared to direct attack and support, it is less effective in reducing the number of connections. This can be explained by the joint attack and support generalising the graphical structure of QBAF into a hyper-graph, which makes the structure learning problem harder.

We also provide a graphical illustration of the learned structure. In figure 6.6, we obtained a degraded structure on the mushroom dataset. This means the joint attack and support relation we learned is degraded to normal attack and support.

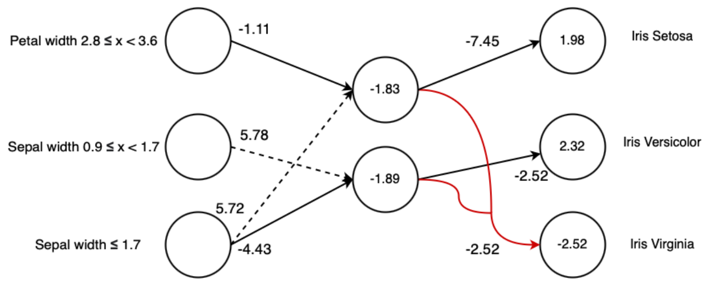


Figure 6.4: Learned QBAF with joint attack and support for the iris dataset. Joint attack and support are labelled in red.

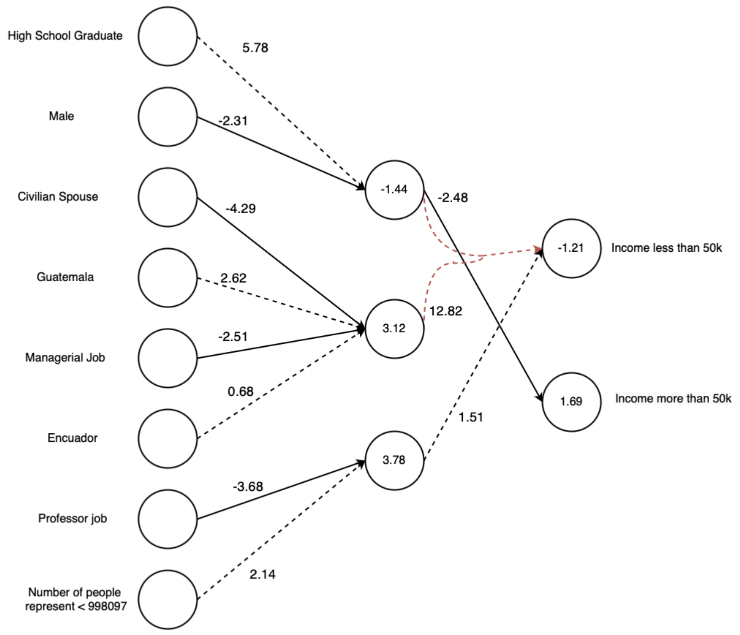


Figure 6.5: Learned QBAF with joint attack and support for the adult dataset. Joint attack and support are labelled in red.

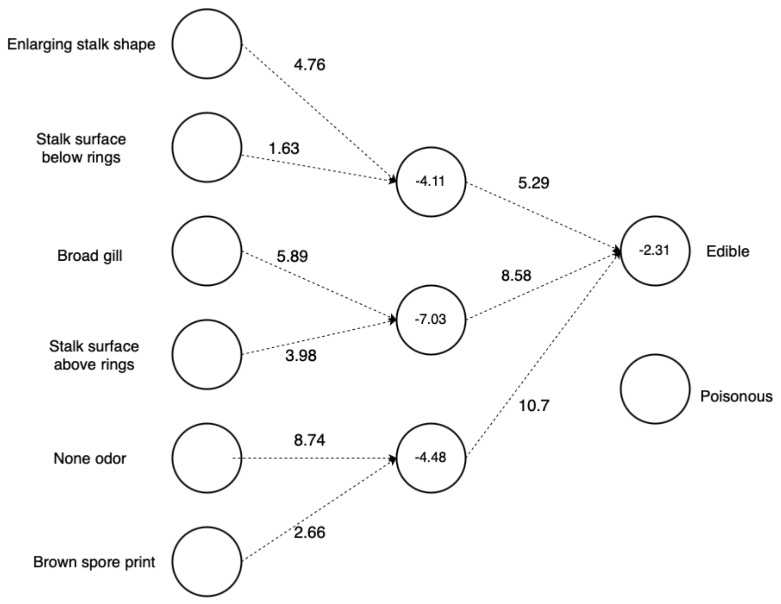


Figure 6.6: Learned QBAF with joint attack and support for the mushroom dataset. However, joint connections degrade to normal connections. Joint attack and support are labelled in red.

6.5 Multiple extensions experiments result

In the following, we use F, D, M, and J to indicate Fuzzy argument input, Direct attack and support, Multi-layer of hidden arguments, and Joint attack and support. Multiple letters represent the combination of different extensions. Datasets are indicated in a similar way as before.

On complicated datasets, fuzzy input can enhance the information in input arguments. Combining fuzzy argument input with structural extension can improve the expressiveness and reduce the complexity of the models.

Direct attack and support + Fuzzy input

Compared to only enabling direct attack and support (4.60 connections) on the iris dataset, enabling direct attack and support and fuzzy input requires more connections (6.10) to attain the same accuracy. However, the scenario with the Adult Income dataset is different. The direct attack and support extension uses 8.1 connections to achieve 0.81 accuracies. With both extensions enabled, the number of connections required is 6.70.

On a simple dataset, a single extension might be enough. However, fuzzy input can enhance the information in input arguments on a more complex dataset. Together with a structure extension, it improves the models' complexity.

Joint attack and support + Fuzzy input

Similar to direct attack and support and fuzzy input, for the iris dataset, the number of connections increases from 5.30 to 7.20, compared to solely applying for joint

Experiment	Connections	Accuracy	Recall	Precision	F1
BaselineG(I)	8.20 (1.72)	0.97 (0.02)	0.97(0.02)	0.97(0.02)	
BaselineP(I)	6.40 (2.10)	0.94 (0.03)	0.93	0.95	0.93
DF(I)	6.10 (2.02)	0.96 (0.04)	0.96 (0.04)	0.96 (0.04)	0.96 (0.04)
JF(I)	7.80 (2.44)	0.96 (0.04)	0.96 (0.04)	0.97 (0.04)	0.96 (0.04)
DJ(I)	6.30 (1.42)	0.96 (0.03)	0.96 (0.03)	0.96 (0.03)	0.96 (0.03)
DJF(I)	**5.30** (1.70)	0.97 (0.03)	0.97 (0.03)	0.97 (0.03)	0.97 (0.03)
BaselineG(A)	13.60 (4.22)	0.81(0.00)	0.71(0.02)	0.76(0.00)	
BaselineP(A)	**6.40** (1.27)	0.81(0.00)	0.68	0.76	0.71
DF(A)	**6.70** (2.26)	0.81 (0.01)	0.68 (0.01)	0.76 (0.02)	0.70 (0.01)
JF(A)	7.20 (2.74)	0.80 (0.02)	0.65 (0.06)	0.76 (0.02)	0.67 (0.08)
DJ(A)	10.30 (1.70)	0.81 (0.01)	0.70 (0.01)	0.76 (0.01)	0.72 (0.01)
DJF(A)	7.00 (2.31)	0.81 (0.01)	0.70 (0.02)	0.76 (0.02)	0.72 (0.01)
BaselineG(M)	16.50 (4.22)	0.97 (0.02)	0.97(0.02)	0.97(0.02)	
BaselineP(M)	**9.10** (2.28)	0.97 (0.02)	0.98	0.98	0.98
DJ(M)	9.40 (2.55)	0.97 (0.01)	0.97 (0.01)	0. 97 (0.01)	0.97 (0.01)

Table 6.5: Result obtained multiple extensions. We use F, D, M, and J to indicate Fuzzy argument input, Direct attack and support, Multi-layer of hidden arguments, and Joint attack and support. And I, A, and M represent the iris, adult income, and mushroom datasets, expressed in parenthesis.

attack and support. Also, for the adult income dataset, the number of connections required to attain an accuracy of 0.97 decreased from 9.30 (joint attack and support) to 7.20 (joint attack and support + fuzzy input).

For a more complicated dataset, using multiple extensions can improve the expressiveness of the QBAF classifier, improving the interpretability.

Joint attack and support and direct attack and support

When joint attack and support and direct attack and support are combined, we observed a reduction in the number of connections (9.40 vs 11.60 when joint attack and support only and 10.70 when direct attack and support only) on the mushroom dataset with similar accuracy. However, we didn't observe a similar effect on the other two datasets. However, because the improvement in the Mushroom dataset is small relative to the standard deviation, this improvement could be attributed to randomness within the learning algorithm. A visualisation is shown in figure 6.7.

Joint attack and support + direct attack and support and fuzzy input

When three extensions are all combined, we observed a reduction in the number of connections compared to only applying the joint attack and support extension and the direct attack and support extension. The reason might be that if only the joint attack and support extension and the direct attack and support extension are enabled, the discrete input might not have enough information to train a complicated structure. However, if fuzzy argument input is also activated, the input information

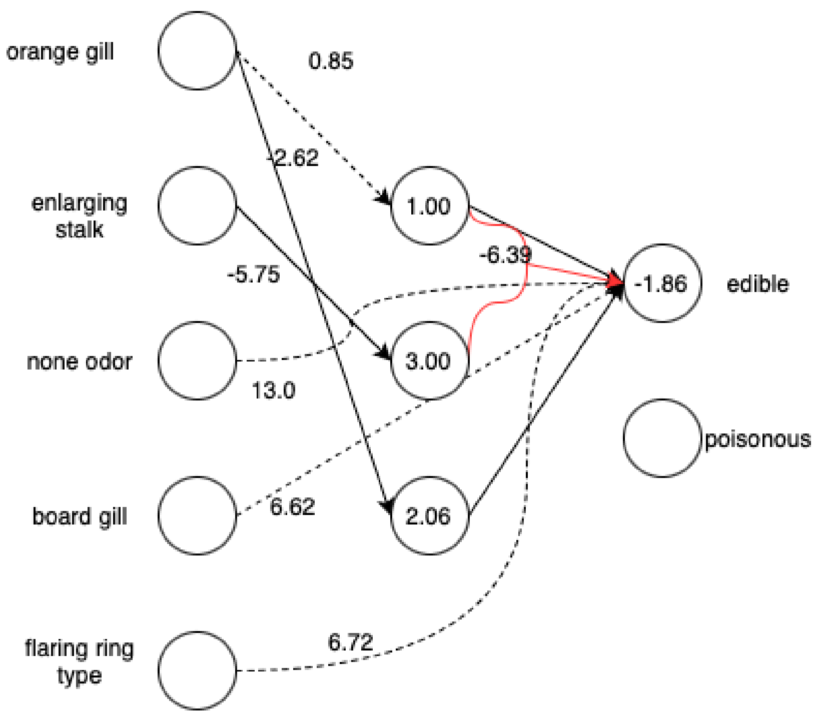


Figure 6.7: Learned QBAF with joint attack and support + direct attack and support for the mushroom dataset

might be expressive enough to support a more complicated structure.

6.6 Iterative pruning experiment result

Baseline

The iterative pruning algorithm applied to the baseline algorithm is denoted as BaselineI (baseline model trained with iterative pruning). The iterative pruning algorithm is very effective in learning the structure of the baseline QBAF classifier. It achieves the highest accuracy on all three datasets compared to both the genetic algorithm and the PSO algorithm. With 8 connections, it achieved 0.99 accuracy on the iris dataset. For the mushroom dataset, the iterative pruning algorithm achieved 1.00 accuracy with 8 connections. With fewer connections of 9, the adult dataset achieved higher accuracy of 0.82.

Iterative pruning utilises the importance of the weights when learning the structure of QBAF. Thus, it outperforms the PSO and the genetic algorithm.

Fuzzy argument input

We observed a slight drop in accuracy for all three datasets when the QBAF classifier was extended with fuzzy argument input. However, the difference in accuracy is minimal. This might be due to the randomness in the learning algorithm.

Direct attack and support

Direct attack and support are also promising. With 9 connections, the accuracy for the iris dataset is 1.00. And for the mushroom dataset, the accuracy is 0.9932.

Multi-layer of hidden arguments

Compared to the result obtained from the genetic algorithm, iterative pruning can learn structure with multiple hidden arguments better. A higher number of hidden arguments results in a larger hypothesis space for the genetic algorithm. The genetic algorithm needs a larger population and more generations to explore the hypothesis space. However, the iterative pruning algorithm learns the structure by removing less essential components. It can address this more complicated structure learning problem.

Experiment	Connections	Accuracy	Recall	Precision	F1
BaselineG(A)	13.60 (4.22)	0.81(0.00)	0.71(0.02)	0.76(0.00)	
BaselineG(M)	16.50 (4.22)	0.97 (0.02)	0.97(0.02)	0.97(0.02)	
BaselineG(I)	8.20 (1.72)	0.97 (0.02)	0.97(0.02)	0.97(0.02)	
BaselineP(A)	6.4 (1.27)	0.81(0.00)	0.68	0.76	0.71
BaselineP(M)	9.1 (2.28)	0.97 (0.02)	0.98	0.98	0.98
BaselineP(I)	6.4 (2.10)	0.94 (0.03)	0.93	0.95	0.93
BaselineI(A)	9	**0.82** (0.02)	0.69 (0.05)	0.78 (0.01)	0.71 (0.07)
BaselineI(M)	**8**	**1.00** (0.00)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)
BaselineI(I)	8	**0.99** (0.01)	0.99 (0.02)	1.00 (0.01)	0.99 (0.01)
Fuzzy(A)	9	0.80 (0.02)	0.65 (0.07)	0.77 (0.02)	0.66 (0.09)
Fuzzy(M)	8	**0.99** (0.02)	0.99 (0.02)	0.99 (0.02)	0.99 (0.02)
Fuzzy(I)	8	0.97 (0.00)	0.95 (0.00)	0.98 (0.00)	0.96 (0.00)
Direct(A)	9	0.80 (0.02)	0.64 (0.08)	0.77 (0.01)	0.64 (0.10)
Direct(M)	9	**1.00** (0.00)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)
Direct(I)	9	**1.00** (0.00)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)
Multi(A)	10	0.78 (0.02)	0.59 (0.07)	0.79 (0.03)	0.58 (0.10)
Multi(M)	10	**0.99** (0.01)	0.99 (0.01)	0.99 (0.01)	0.99 (0.01)
Multi(I)	10	**0.99** (0.02)	0.99 (0.02)	0.99 (0.02)	0.99 (0.02)

Table 6.6: Iterative pruning experiment result. I, A, and M represent the iris, adult income, and mushroom datasets.

6.7 Discussion and summary

On simpler datasets, direct attack and support can efficiently reduce the number of connections while attaining classification performance. However, on a more complicated dataset, more extensions are needed. The direct attack and support extension or joint attack and support made the QBAF classifier more expressive. A more expressive QBAF structure needs more information to facilitate training. The fuzzy argument input enriched input information.

Multi-layer of hidden argument expands the hypothesis space. The genetic algorithm needs a larger population and more generations to explore the hypothesis space. However, the iterative pruning algorithm learns the structure by removing less essential components. It can address this more complicated structure learning problem.

The iterative pruning algorithm attained better accuracy than the genetic and PSO algorithms. However, the genetic algorithm is more flexible. It can train a QBAF structure with joint attack and support, whereas iterative pruning cannot. In addition, the genetic algorithm can discover more sparse QBAF structures, for example, structures with four to five connections on the iris dataset. However, it is challenging for iterative pruning to learn a structure with few connections.

The run time for the genetic algorithm mainly depends on the population size, the number of generations, and the dataset size. One run of the genetic algorithm with a population size of 50 and 10 generations on the adult income dataset typically costs 21 minutes. On the other hand, the iterative pruning algorithm costs 5 minutes to train a QBAF classifier on the adult income dataset. The result was tested on a machine with an 8-core 5600X CPU and 16GB of RAM.

Chapter 7

Conclusion and future work

7.1 Conclusion

Neural networks' internal mechanism remains a black box for human beings. This project explores an interpretable and transparent algorithm that humans can understand. A previous work [24] utilises a QBAF to explain neural networks. Hence it builds interpretable classifiers using QBAF. The QBAF classifier has stronger classification performance than shallow decision trees or logistic regression.

Our research applies four key ideas beyond the baseline QBAF classifier [27]. We also implement an extended genetic algorithm and explores the feasibility of using iterative pruning to learn the structure of the QBAF classifier.

Extended QBAF classifer

Fuzzy input transformation, direct attack and support, and joint attack and support showed to be effective. Direct attack and support can capture a more straightforward relationship between the input and output argument, performing the best for a simpler dataset. However, for a more complicated dataset, combining fuzzy input argument, direct attack and support, and joint attack and support helped improve the model's expressiveness, resulting in fewer connections to achieve similar classification performance. The PSO algorithm generally performs better than the genetic algorithm. Multi-layer of hidden arguments enlarges the hypothesis space. The genetic algorithm needs a larger population and more generations to learn this structure.

- Fuzzy input transformation
- Multi-layer of hidden arguments
- Direct attack and support
- Joint attack and support

Iterative pruning

Iterative pruning is a method that improves memory usage and inference speed for neural networks. We experimented with the possibility of using iterative pruning at an extreme level, which can make the structure understood by humans. Results show that iterative pruning has a shorter runtime than the genetic algorithm, and the QBAF classifier learned using iterative pruning has better classification accuracy. Also, iterative pruning can learn multi-layer of hidden arguments because it learns the structure by removing less critical components. However, iterative pruning is hard to incorporate more complicated structures like the joint attack and support.

7.2 Future work

Further explore iterative pruning

We conducted an initial feasibility experiment on using iterative pruning to learn the structure of QBAF. However, the hyper-parameters can be further explored to achieve better results.

Experiment more datasets

We experimented on the iris, mushroom, and adult income datasets, despite these being classical tubular datasets that are commonly used. The iris dataset consists of only 150 samples, while the test set is only 30. The result from the iris dataset can have a significant standard deviation. A more concrete conclusion can be derived if we could experiment on more datasets.

Hyper-parameter for input transformation

Our experiment didn't explore the effect of having different numbers of fuzzy sets. Further investigations on the number of fuzzy sets might improve the classification performance.

Genetic algorithm fitness function

Relative sparsity showed a pleasing effect on the genetic algorithm. However, further experiments can be applied to understand the impact of relative sparsity fitness function. In addition, more designs of fitness functions can be explored.

Experiment cyclic QBAF structure

Our research focuses on acyclic QBAF structures. However, QBAF structures can be cyclic. Graph neural networks can potentially be interpreted as cyclic QBAF.

Remove restrictions for joint attack and support

To address the problem that joint attack and support degrades to standard attack and support, we enforced some restrictions on joint attack and support. However, this limits the expressiveness of the joint attack and support QBAF classifier. Further exploration can be applied to remove these restrictions.

Repeat parameter learning in genetic algorithm

Due to the randomness in the parameter learning algorithm (gradient descent), only using one run (multiple epochs) might not yield an accurate fitness. One mitigation can be to run the gradient descent more times and take the average to determine the fitness function's accuracy term.

Human experiment to evaluate explanability

A further human experiment can be conducted to assess the explainability of the QBAF classifier.

Specialised genetic algorithm operators

We can replace the general genetic operators, such as mutation, with more specialised operators that suit the QBAF structure learning. We can also experiment with other kinds of chromosome representations of QBAF.

Bibliography

[1] Arthur Asuncion and David Newman. Uci machine learning repository, 2007. pages 8, 39

[2] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7): e0130140, 2015. pages 5

[3] Mohamad Wahed Bazo. Learning quantitative argumentation frameworks using sparse neural networks and swarm intelligence algorithms. B.S. thesis, 2021. pages 6, 13, 17, 40

[4] Gustavo Bodanza, Fernando Tohmé, and Marcelo Auday. Collective argumentation: A survey of aggregation issues around argumentation frameworks. *Argument & Computation*, 8(1):1–34, 2017. pages 6, 15, 16

[5] G. Castellano, A.M. Fanelli, and M. Pelillo. An iterative pruning algorithm for feedforward neural networks. *IEEE Transactions on Neural Networks*, 8(3): 519–531, 1997. doi: 10.1109/72.572092. pages 18

[6] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. L2 regularization for learning kernels. *arXiv preprint arXiv:1205.2653*, 2012. pages 17

[7] Sylvie Coste-Marquis, Sébastien Konieczny, Pierre Marquis, and Mohand Akli Ouali. Weighted attacks in argumentation frameworks. 06 2012. pages 9

[8] Filip Karlo Došilović, Mario Brčić, and Nikica Hlupić. Explainable artificial intelligence: A survey. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 0210–0215, 2018. doi: 10.23919/MIPRO.2018.8400040. pages 5

[9] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial intelligence*, 77(2):321–357, 1995. pages 5, 9

[10] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018. pages 17

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. pages 23

[12] Cengiz Kahraman, Başar Öztayşi, and Sezi Çevik. A comprehensive literature review of 50 years of fuzzy set theory. *International Journal of Computational Intelligence Systems*, 9, 05 2016. doi: 10.1080/18756891.2016.1180817. pages 14, 15

[13] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989. pages 17

[14] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016. pages 17

[15] Christoph Molnar, Giuseppe Casalicchio, and Bernd Bischl. Interpretable machine learning – a brief history, state-of-the-art and challenges. In *ECML PKDD 2020 Workshops*, pages 417–431. Springer International Publishing, 2020. doi: 10.1007/978-3-030-65965-3_28. URL `https://doi.org/10.1007%2F978-3-030-65965-3_28`. pages 5

[16] Till Mossakowski and Fabian Neuhaus. Modular semantics and characteristics for bipolar weighted argumentation graphs. *arXiv preprint arXiv:1807.06685*, 2018. pages 10, 11, 12

[17] Søren Holbech Nielsen and Simon Parsons. A generalization of dung's abstract framework for argumentation: Arguing with sets of attacking arguments. In *International Workshop on Argumentation in Multi-Agent Systems*, pages 54–73. Springer, 2006. pages 6, 15, 16

[18] Mee Young Park and Trevor Hastie. L1-regularization path algorithm for generalized linear models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(4):659–677, 2007. pages 17

[19] Nico Potyka. A tutorial for weighted bipolar argumentation with continuous dynamical systems and the java library attractor. *arXiv preprint arXiv:1811.12787*, 2018. pages 5, 9, 10

[20] Nico Potyka. Interpreting neural networks as gradual argumentation frameworks (including proof appendix), 2020. URL `https://arxiv.org/abs/2012.05738`. pages 5, 9, 11, 12

[21] Nico Potyka. Foundations for solving classification problems with quantitative abstract argumentation. In *XI-ML@ KI*, 2020. pages 22

[22] Nico Potyka. Foundations for solving classification problems with quantitative abstract argumentation. In *XI-ML@ KI*, 2020. pages 9, 12, 13

[23] Nico Potyka. Interpreting neural networks as quantitative argumentation frameworks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 6463–6470, 2021. pages 24

[24] Nico Potyka, Mohamad Bazo, Jonathan Spieler, and Steffen Staab. Learning gradual argumentation frameworks using meta-heuristics. 2022. pages 12, 50

[25] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016. pages 5

[26] Benedek Rozemberczki, Lauren Watson, Péter Bayer, Hao-Tsung Yang, Olivér Kiss, Sebastian Nilsson, and Rik Sarkar. The shapley value in machine learning, 2022. URL `https://arxiv.org/abs/2202.05594`. pages 9

[27] Jonathan Spieler, Nico Potyka, and Steffen Staab. Learning gradual argumentation frameworks using genetic algorithms. *arXiv preprint arXiv:2106.13585*, 2021. pages 6, 9, 10, 13, 17, 20, 23, 40, 50

[28] H-J Zimmermann. Fuzzy set theory. *Wiley interdisciplinary reviews: computational statistics*, 2(3):317–332, 2010. pages 14

[29] Luisa M Zintgraf, Taco S Cohen, Tameem Adel, and Max Welling. Visualizing deep neural network decisions: Prediction difference analysis. *arXiv preprint arXiv:1702.04595*, 2017. pages 5

[30] Kristijonas Čyras, Antonio Rago, Emanuele Albini, Pietro Baroni, and Francesca Toni. Argumentative xai: A survey, 2021. URL `https://arxiv.org/abs/2105.11266`. pages 5

Appendix A

Configurations and Hyper-parameters

Fuzzy Baseline iris

- number-runs 10
- population-size 20
- number-generations 10
- learning-rate 0.01
- number-epochs 300
- hidden-size 4
- number-connections1 4
- number-connections2 2
- lambda 0.05
- crossover-rate 0.9
- mutation-rate 0.001
- patience-ES 5
- tolerance-ES 0.0001
- elitist-pct 0.1
- patience-GA 5
- tolerance-GA 0.0001
- is-fuzzy true
- model-name GBAG
- dataset iris
- input-size 12
- output-size 3

Fuzzy Baseline adult

- number-runs 10
- population-size 50
- number-generations 10
- learning-rate 0.01
- number-epochs 300
- hidden-size 12
- number-connections1 10
- number-connections2 6
- lambda 0.05
- crossover-rate 0.9
- mutation-rate 0.001
- patience-ES 5
- tolerance-ES 0.003
- elitist-pct 0.1
- patience-GA 5
- tolerance-GA 0.0001
- is-fuzzy true
- model-name GBAG
- dataset adult
- input-size 116
- output-size 2

Direct adult

- number-runs 10
- population-size 50
- number-generations 10
- learning-rate 0.01
- number-epochs 300
- hidden-size 12
- number-connections1 10
- number-connections2 6
- lambda 0.05
- crossover-rate 0.9
- mutation-rate 0.001
- patience-ES 5
- tolerance-ES 0.003
- elitist-pct 0.1
- patience-GA 5
- tolerance-GA 0.0001
- relative-sparsity true
- is-fuzzy false
- model-name DAGBAG
- dataset adult
- number-connections-skip 1
- input-size 116
- output-size 2

Direct mushroom

- number-runs 10
- population-size 30
- number-generations 10
- learning-rate 0.01
- number-epochs 300
- hidden-size 12
- number-connections1 6
- number-connections2 6
- lambda 0.05

- crossover-rate 0.9
- mutation-rate 0.001
- patience-ES 5
- tolerance-ES 0.003
- elitist-pct 0.1
- patience-GA 5
- tolerance-GA 0.0001
- relative-sparsity true
- is-fuzzy false
- model-name DAGBAG
- dataset mushroom
- number-connections-skip 1
- input-size 112
- output-size 2

Direct iris

- number-runs 10
- population-size 20
- number-generations 10
- learning-rate 0.01
- number-epochs 300
- hidden-size 4
- number-connections1 4
- number-connections2 2
- lambda 0.05
- crossover-rate 0.9
- mutation-rate 0.001
- patience-ES 5
- tolerance-ES 0.0001
- elitist-pct 0.1
- patience-GA 5
- tolerance-GA 0.0001
- is-fuzzy false
- model-name DAGBAG
- dataset iris
- number-connections-skip 1
- input-size 12
- output-size 3

Joint adult

- number-runs 10
- population-size 50
- number-generations 10
- learning-rate 0.01
- number-epochs 300
- hidden-size 12
- number-connections1 9
- number-connections2 6
- lambda 0.05
- crossover-rate 0.9
- mutation-rate 0.001
- patience-ES 5
- tolerance-ES 0.003
- elitist-pct 0.1
- patience-GA 5
- tolerance-GA 0.0001
- relative-sparsity true
- is-fuzzy false
- model-name JASGBAG
- dataset adult
- joint-connections-size1 1
- joint-connections-size2 1
- joint-connections-input-num1 4
- joint-connections-input-num2 2
- input-size 116
- output-size 2

Joint iris

- number-runs 10
- population-size 20
- number-generations 10
- learning-rate 0.01
- number-epochs 300
- hidden-size 4
- number-connections1 2
- number-connections2 2
- lambda 0.05
- crossover-rate 0.9
- mutation-rate 0.001
- patience-ES 5
- tolerance-ES 0.0001
- elitist-pct 0.1
- patience-GA 5
- tolerance-GA 0.0001
- relative-sparsity true
- is-fuzzy false
- model-name JASGBAG
- dataset iris
- joint-connections-size1 2
- joint-connections-size2 1
- joint-connections-input-num1 4
- joint-connections-input-num2 2
- input-size 12
- output-size 3

Joint mushroom

- number-runs 10
- population-size 30
- number-generations 10
- learning-rate 0.01
- number-epochs 300
- hidden-size 12
- number-connections1 5
- number-connections2 6
- lambda 0.05
- crossover-rate 0.9
- mutation-rate 0.001
- patience-ES 5
- tolerance-ES 0.003
- elitist-pct 0.1
- patience-GA 5
- tolerance-GA 0.0001
- relative-sparsity true
- is-fuzzy false
- model-name JASGBAG
- dataset mushroom
- joint-connections-size1 1
- joint-connections-size2 1
- joint-connections-input-num1 4
- joint-connections-input-num2 2
- input-size 112

- output-size 2

Fuzzy Direct adult

- number-runs 10
- population-size 50
- number-generations 10
- learning-rate 0.01
- number-epochs 300
- hidden-size 12
- number-connections1 6
- number-connections2 6
- lambda 0.05
- crossover-rate 0.9
- mutation-rate 0.001
- patience-ES 5
- tolerance-ES 0.003
- elitist-pct 0.1
- patience-GA 5
- tolerance-GA 0.0001
- relative-sparsity true
- is-fuzzy true
- model-name DAGBAG
- dataset adult
- number-connections-skip 1
- input-size 116
- output-size 2

Fuzzy Direct iris

- number-runs 10
- population-size 20
- number-generations 10
- learning-rate 0.01
- number-epochs 300
- hidden-size 4
- number-connections1 4
- number-connections2 2
- lambda 0.05
- crossover-rate 0.9
- mutation-rate 0.001
- patience-ES 5
- tolerance-ES 0.003
- elitist-pct 0.1
- patience-GA 5
- tolerance-GA 0.0001
- is-fuzzy true
- model-name DAGBAG
- dataset iris
- number-connections-skip 1
- input-size 12
- output-size 3

Fuzzy Direct mushroom

- number-runs 10
- population-size 30
- number-generations 10
- learning-rate 0.01
- number-epochs 300
- hidden-size 6
- number-connections1 4
- number-connections2 2
- lambda 0.05
- crossover-rate 0.9
- mutation-rate 0.001
- patience-ES 5
- tolerance-ES 0.003
- elitist-pct 0.1
- patience-GA 5
- tolerance-GA 0.0001
- relative-sparsity true
- is-fuzzy true
- model-name DAGBAG
- dataset mushroom
- number-connections-skip 1
- input-size 112
- output-size 2

Fuzzy Joint iris

- number-runs 10
- population-size 20
- number-generations 10
- learning-rate 0.01
- number-epochs 300
- hidden-size 12
- number-connections1 9
- number-connections2 6
- lambda 0.05
- crossover-rate 0.9
- mutation-rate 0.001
- patience-ES 5
- tolerance-ES 0.003
- elitist-pct 0.1
- patience-GA 5
- tolerance-GA 0.0001
- relative-sparsity true
- is-fuzzy true
- model-name JASGBAG
- dataset iris
- joint-connections-size1 1
- joint-connections-size2 1
- joint-connections-input-num1 4
- joint-connections-input-num2 2
- input-size 12
- output-size 3

Fuzzy Joint adult

- number-runs 10
- population-size 50
- number-generations 10
- learning-rate 0.01
- number-epochs 300
- hidden-size 12
- number-connections1 9
- number-connections2 6
- lambda 0.05
- crossover-rate 0.9
- mutation-rate 0.001
- patience-ES 5
- tolerance-ES 0.003
- elitist-pct 0.1
- patience-GA 5
- tolerance-GA 0.0001
- relative-sparsity true

- is-fuzzy true
- model-name JASGBAG
- dataset adult
- joint-connections-size1 1
- joint-connections-size2 1
- joint-connections-input-num1 4
- joint-connections-input-num2 2
- input-size 116
- output-size 2

Joint and Direct adult

- number-runs 10
- population-size 50
- number-generations 10
- learning-rate 0.01
- number-epochs 300
- hidden-size 12
- number-connections1 8
- number-connections2 6
- lambda 0.05
- crossover-rate 0.9
- mutation-rate 0.001
- patience-ES 5
- tolerance-ES 0.003
- elitist-pct 0.1
- patience-GA 5
- tolerance-GA 0.0001
- relative-sparsity true
- is-fuzzy false
- model-name JASDAGBAG
- dataset adult
- joint-connections-size1 2
- joint-connections-size2 1
- joint-connections-input-num1 4
- joint-connections-input-num2 2
- number-connections-skip 1
- input-size 116
- output-size 2

Joint and Direct mushroom

- number-runs 10
- population-size 30
- number-generations 10
- learning-rate 0.01
- number-epochs 300
- hidden-size 6
- number-connections1 2
- number-connections2 2
- lambda 0.05
- crossover-rate 0.9
- mutation-rate 0.001
- patience-ES 5
- tolerance-ES 0.003
- elitist-pct 0.1
- patience-GA 5
- tolerance-GA 0.0001
- relative-sparsity true
- is-fuzzy false
- model-name JASDAGBAG
- dataset mushroom
- joint-connections-size1 2
- joint-connections-size2 1
- joint-connections-input-num1 4
- joint-connections-input-num2 2
- number-connections-skip 1
- input-size 112
- output-size 2

Joint and Direct iris

- number-runs 10
- population-size 20
- number-generations 10
- learning-rate 0.01
- number-epochs 300
- hidden-size 4
- number-connections1 2
- number-connections2 2
- lambda 0.05
- crossover-rate 0.9
- mutation-rate 0.001
- patience-ES 5
- tolerance-ES 0.0001
- elitist-pct 0.1
- patience-GA 5
- tolerance-GA 0.0001
- relative-sparsity true
- is-fuzzy false
- model-name JASDAGBAG
- dataset iris
- joint-connections-size1 2
- joint-connections-size2 1
- joint-connections-input-num1 4
- joint-connections-input-num2 2
- number-connections-skip 1
- input-size 12
- output-size 3

Fuzzy Joint and Direct iris

- number-runs 10
- population-size 20
- number-generations 10
- learning-rate 0.01
- number-epochs 300
- hidden-size 8
- number-connections1 2
- number-connections2 4
- lambda 0.05
- crossover-rate 0.9
- mutation-rate 0.001
- patience-ES 5
- tolerance-ES 0.003
- elitist-pct 0.1
- patience-GA 5
- tolerance-GA 0.0001
- relative-sparsity true
- is-fuzzy true
- model-name JASDAGBAG
- dataset iris
- joint-connections-size1 2
- joint-connections-size2 1
- joint-connections-input-num1 4

- joint-connections-input-num2 2
- number-connections-skip 1
- input-size 12
- output-size 3

Fuzzy Joint and Direct adult

- number-runs 10
- population-size 50
- number-generations 10
- learning-rate 0.01
- number-epochs 300
- hidden-size 4
- number-connections1 3
- number-connections2 2
- lambda 0.05
- crossover-rate 0.9
- mutation-rate 0.001
- patience-ES 5
- tolerance-ES 0.003
- elitist-pct 0.1
- patience-GA 5
- tolerance-GA 0.0001
- relative-sparsity true
- is-fuzzy true
- model-name JASDAGBAG
- dataset adult
- joint-connections-size1 1
- joint-connections-size2 1
- joint-connections-input-num1 4
- joint-connections-input-num2 2
- number-connections-skip 1
- input-size 116
- output-size 2