Structural Pruning of Sum-Product Networks

Vereinfachen der Struktur von Sum-Product Networks
Bachelor thesis by Alexander Lind
Date of submission: February 21, 2021

1. Review: Eneldo Loza Mencía
2. Review: Moritz Kulessa
Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Computer Science Department

Knowledge Engineering Group

Erklärung zur Abschlussarbeit gemäß §22 Abs. 7 APB TU Darmstadt

Hiermit versichere ich, Alexander Lind, die vorliegende Bachelorarbeit gemäß §22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 21. Februar 2021

Alexander Lind

Abstract

We developed a basic algorithm, demonstrated the feasibility of post-pruning large Sum-Product Networks and improved their inference runtime. In particular, we show that the number of parameters for a larger model strongly reduces while maintaining the log-likelihood. Furthermore, the proposed pruning method merely operates on Sum-Product Network structures and does not require additional data. Especially applications with restricted memory requirements benefit from the lean structural representation of Sum-Product Networks. Moreover, it is observed that industries transition to smaller devices with limited computational resources. Our method merges highly correlating leaf nodes to a single leaf node with minimal log-likelihood decrease. In this context, this thesis evaluates two initial SPN structures created by the Learn-SPN algorithm on real-world data sets utilizing benchmark evaluation metrics. Furthermore, we highlight the connection between the initial SPN structures and post-pruned SPNs in relation to their model quality and quantity. Finally, we present the limitations while also giving an outlook on promising improvements to our approach.

Zusammenfassung

In dieser Arbeit wurde ein Algorithmus für das Kürzen von Sum-Product Networks entwickelt und demonstriert, dass es möglich ist, größere Modelle nachträglich zu verkleinern und somit deren Geschwindigkeit für die Entscheidungsfindung verbessert. Insbesondere zeigen wir, dass sich die Parameteranzahl besonders bei größeren Modellen stark vermindern lässt, während die Log-Likelihood beibehalten bleibt. Das vorgeschlagene Verfahren nutzt lediglich die Strukturen von Sum-Product Networks aus, sodass keine zusätzlichen Daten für das Kürzen benötigt werden. Gleichzeitig haben simplere Strukturen einen wirtschaftliche Vorteile, da diese im Gegensatz zu anderen Wissensrepräsentationen, weniger Speicherplatz benötigen. Außerdem zeigt sich, dass Industrieunternehmen zunehmend kleinere Endgeräte entwickeln, während die Größe von Modellen zur Entscheidungsfindung zunimmt. Die vorgestellte Methode kombiniert stark korrelierende Blattknoten zu einem einzelnen Blattknoten zusammen wobei die Log-Likelihood geringfügig abnimmt. In diesem Kontext werden in dieser Bachelorarbeit zwei SPN-Strukturen, die durch den Learn-SPN-Algorithmus erstellt wurden, mittels gängiger Bewertungsmethoden anhand praxisnaher Datensätze evaluiert. Des Weiteren soll das Zusammenspiel zwischen den genannten SPN-Strukturen und den gekürzten Strukturen hinsichtlich ihrer Güte und Quantität hervorgehoben werden. Zum Abschluss präsentieren wir die Einschränkungen sowie Vorschläge für erfolgversprechende Erweiterungen unseres Ansatzes.

Contents

List of Figures

List of Tables

List of Algorithms

Glossary

mis min instance slice. 22, 41–52, 54, 55, 58, 59, 65

rdc-threshold rdc-threshold. 22, 41–43, 45–47, 55

th threshold. 42, 45–49, 53–59

ACMN arithmetic circuit Markov networks. 23

avg. LL average Log-Likelihood. 38, 39, 43, 45–49, 52–56, 58, 59, 65

CMLL conditional marginal Log-Likelihood. 39

JSD Jensen–Shannon divergence. 31, 42, 45

KLD Kullback-Leibler divergence. 31

Learn-MSPN Learning Mixed Sum-Product Networks. 21, 22, 29, 41–45, 52, 57, 60, 65, 66

RDC randomized dependency coefficient. 21, 22

SPA-SPN Simple Post-Pruning Algorithm for Sum-Product Networks. 29, 31–33, 38, 39, 41, 42, 45, 47–52, 54–60

SPN Sum-Product Network. 11–13, 17–39, 41–60, 65, 66

TDIDT Top-Down Induction of Decision Trees. 15

1. Introduction

The principle of Occam's Razor implies that if there are two competing theories, one should focus on the simpler one. This concept often entails useful applications in the field of computer science and represents the main idea behind a variety of essential algorithms such as boosting, bagging, and pruning algorithms (Zhou, 2015; Patel and Upadhyay, 2012). For example, pruning methods for decision trees commonly incorporate the idea of Occam's razor as a heuristic to determine a decision tree that gives a simpler explanation (Patel and Upadhyay, 2012). Generally, pre-pruning methods induce an efficient and accurate decision tree from a collection of examples. However, decision trees generated by these methods often lead to overly complex models. Initial research by Quinlan (1987) focused on pruning algorithms for decision trees in order to discover simpler models. In the same way, previous research applied pruning to various tree-structured models, including graphs, e.g., Markov Networks, Sum-Product Networks (SPNs), or Bayesian Networks (Pedersen and Stork, 1996; Gogate, Webb, and Domingos, 2010).

The observed information in real-world problems is commonly limited and contains noise introducing uncertainty in data. In fact, the noise in data results from the error in measurement or subjective judgments. Generally, decision trees do not consider uncertainty in the observed information for reasoning and decision making (Costa, Verwer, and Blockeel, 2013; Patel and Upadhyay, 2012). In contrast, Probabilistic Graphical Models (PGMs) explain and reason uncertainty in data. PGMs utilize an intuitive representation of random variables to represent the underlying distribution of the given observations. Moreover, the graph-based representation of PGMs allows to model the interactions between an extensive collection of random variables by using probability distributions. As a result, PGMs model complex probabilistic models consisting of simple components to answer simple queries. The interaction of variables modeled by PGMs allows them to infer missing values for a given evident value by utilizing the joint probability. However, PGMs such as Bayesian Networks and Markov Networks often use an approximation to represent the joint probability. This approximation comes with an increased computational cost and can reach an exponential level in the worst case (Gogate, Webb, and Domingos, 2010).

Poon and Domingos (2011) introduced Sum-Product Networks (SPNs), uniting the ideas of deep learning and PGMs. An SPN's structure attempts to represent a probability distribution of a given data set compactly by sums and products of individual random variables. The benefit of using SPNs is that the computation of the joint distribution is linear in time depending on the size of its edges (Poon and Domingos, 2011). Initial research on SPNs focused on learning the structure of an SPN for a given data set. The learned structures showed increased accuracy and inference speed compared to PGMs (Gens and Domingos, 2013) and performed better than SPN structures designed by hand (Dennis and Ventura, 2012). Apart from this, SPNs incorporate structure learning methods by utilizing conditioning and decomposition steps to learn an SPN structure for a given data set. For example, SPNs recreate speech from noisy frequencies by artificially extending the bandwidth of signals (Lowd and Rooshenas, 2013). Nonetheless, the resulting Sum Product-Networks'

increased performance has the disadvantage of increased computational cost due to its large size in parameters (Lowd and Rooshenas, 2013). As a result, post-processing methods that remove redundancies in SPN structures while improving inference speed and memory requirements are a promising research topic. For example, lower memory requirements and faster inference enable real-time applications with low bandwidth and critical timing requirements. It should be noted that people working in the field of artificial intelligence have evolved a growing concern for the interpretability of machine learning models. As a result, there is a lot of discussion about finding better solutions to interpret a vast set of algorithms (Rudin, 2018). Consequently, removing redundant structures within an SPN leads to simpler models that are easier to interpret by experts.

In this context, this thesis introduces a post-pruning method for Sum-Product Networks (SPNs) aiming to reduce its number of parameters. Therefore, Chapter 2 gives a detailed description of decision trees and SPNs. While focusing on pruning methods, the individual models' similarities and differences are outlined, such as giving a detailed introduction into structure learning for SPNs. The Chapter 3 introduces the most important work for pre-pruning methods for SPNs just as highlighting crucial methods to enable post-pruning. Subsequently, the Chapter 4 revolves around the idea of post-pruning by finding similarities and redundancies within SPNs. Especially, we assume that similar structures within an SPN can be merged while the introduced performance loss is insignificant. In addition to this, we show a method for retaining a valid SPN structure such that the properties of a fast computation hold. To demonstrate the benefits of post-pruning SPNs, Chapter 5 compiles the most important results and steps to reproduce our findings. Moreover, the approach is empirically evaluated on five different data sets, including binary variables, just as utilizing various evaluation metrics. In order to prune an SPN, the Learn-MSPN (Molina, Vergari, Mauro, et al., 2018) algorithm creates distinct initial structures. Finally, we discuss the impact and limitations of using this approach and give an outlook for potential future work.

2. Background

This chapter starts with the general structure of decision trees limiting to the domain of classification, introduced in Section 2.1. The pruning Section 2.2 gives a general explanation of its approach and objectives. Furthermore, it outlines the differences in pre- and post-pruning methods in Subsections 2.2.1 and 2.2.2. The following Section 2.3 gives a comprehensive introduction to Sum Product-Networks covering its structure, evaluation and introducing structure learning. Additionally, the structure learning Sections 2.4, 2.4.1, and 2.4.2 describe novel frameworks to induce SPNs from data. The sections also highlight the relationships between pre-pruning methods for decision trees and structure learners for SPNs. Finally, this chapter provides the fundamental tools to implement post-pruning methods for decision trees to SPNs.

2.1. Decision Trees

A decision tree consists of edges, leaves, and nodes as well as a root node. The internal nodes represent rules that test the condition of a feature for its value. The edges are labeled by feature values and connect individual nodes, starting at the root node. Furthermore, the branches of a node represented by the edges to its child nodes indicate a rule's outcome. In the end, a leaf node labels a class and represents the prediction associated with the given input values. For example, Table 2.1 contains a collection of samples in each row. The table columns represent three features $\{\mathit{Weather}, \mathit{Temperature}, \mathit{Wind}\}$ and a class label **Hike**. Whether a family should go outside for a hike can be assessed by individual values of the features. The *Weather* can be **Sunny**, **Cloudy**, or **Rainy**, the *Temperature* can be **Hot**, **Cold**, or **Mild**, and the *Wind* blows either **Strong** or **Weak**.

Figure 2.1.: Sample data

Features			Class
Weather	Temperature	Wind	Hike
Sunny	Hot	Strong	Yes
Cloudy	Hot	Weak	Yes
Rainy	Cold	Weak	No
Sunny	Mild	Weak	Yes
Sunny	Cold	Weak	Yes
Cloudy	Mild	Strong	No
Cloudy	Hot	Weak	Yes
Rainy	Hot	Weak	Yes
Rainy	Mild	Weak	Yes
Rainy	Mild	Strong	No
Sunny	Mild	Strong	Yes

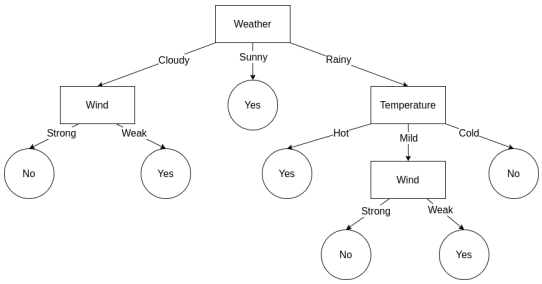

Figure 2.2.: This Figure shows a decision tree that predicts the sample data in Table 2.1 whether a family should go for a hike.

The decision tree in Figure 2.2 outlines the decision-making process about whether or not a family should go on a hike together. It evaluates the input values in a top-down fashion by checking the individual rules. For example, on a mild and rainy day, the wind is blowing strong, which means that the decision tree takes the values Rainy, Mild, Strong. The generation of the decision tree starts at the root node. The root node checks the Weather feature's value and propagates the remaining values (Mild, Strong) to its child node through the edge with the value Rainy. The node with the Temperature feature evaluates the value Mild and forwards the Strong value to the node with the label Wind. Finally, branching the Strong value to the leaf node and its class predicts that a family should not hike.

2.2. Pruning

Generally, pruning for decision trees aims to reduce the set of nodes, edges, and leaves in the tree while maintaining its accuracy in classifying examples. Pruning divides into two different categories, pre-, and post-pruning. Pre-pruning is applied during the construction of trees, while post-pruning resizes an existing tree to reduce parameters. Pre-pruning methods utilize a data set to generate the structure of a tree. It includes an early stopping criterion that controls the expansion of a tree. A branch stops to extend as soon as the decision tree only improves insignificantly.

Post-pruning operates on an existing tree and reduces the set of parameters while maintaining its performance. It utilizes heuristics to decide whether the replacement, insertion, or removal of a parameter is reasonable. Hence, post-pruning reduces the size of parameters in a tree structure. Both approaches try to find a simpler solution to a model while retaining its semantics and performance. Reducing the set of parameters offers faster computation and lower memory requirements while evaluating an example. Previous research applied

the concept of pruning methods to various other types of machine learning methods, e.g., Neural Networks, Boltzmann networks, and hidden Markov models, as shown in Augasta and Kathirvalavakumar (2011) and Pedersen and Stork (1996).

For example, there are 10^{120} possibilities to play a chess game, which a structured, directed tree can represent. The nodes model chess moves and the edges between the nodes represent valid moves. As the players pick moves, both players look for the optimal moves to win the chess game. Instead of evaluating all possible moves to win a game, heuristics can reduce the search space and find an optimal strategy for winning. One particular approach is introduced by Knuth and Moore (1975) called the alpha-beta pruning algorithm that tries to calculate an optimal strategy for winning the game. The idea is to discard a chess move strategy once it is proven to be inferior to alternative strategies. Thus, eliminating unfavorable strategies leads to reduced search space. The following sub-sections explain the approaches to obtain decision trees with a reduced set of parameters.

2.2.1. Pre-pruning

Pre-pruning algorithms limit the number of parameters during the construction of a decision tree. A prominent framework to induce a decision tree by a set of examples is introduced by Queyranne (1998). The Top-Down Induction of Decision Trees (TDIDT) algorithm is a top-down approach and falls into the pre-pruning methods category. TDIDT follows the divide and conquer paradigm by dividing the problem of generating a decision tree into smaller analytical units until each subproblem can be solved directly. It starts at the root node and inputs a data set consisting of variables and a collection of examples. The general framework incorporates two crucial steps for generating the decision tree. First, an attribute is selected to split the given data set by its attributes. The selected attribute represents a rule and generates a branch from the node for all possible outcomes. Then, the data instances are split based on the selected attribute outcomes, and each subset falls into the respective branch. These two steps are repeated for each branch until all instances have the same class.

The attribute selection is essential to growing simpler trees since it affects the attribute splits in subsequent nodes. Thus, the attribute selection to choose the optimal attribute split requires a measure of quality. Ideally, the selected attribute generates branches that split data instances with mostly a single class. But it should be noted that finding an optimal tree structure for a large number of values is challenging. For this reason, pre-pruning methods incorporate heuristics to limit the generation of branches. The heuristic functions as a test to measure the statistical significance between the attributes at a node. Its goal is to find attributes that are significant to the associated classes. The following paragraphs introduce various methods that select the appropriate attribute for a node.

ID3 developed by Queyranne (1998) is a top-down pre-pruning method. ID3 aims to select an attribute split that partitions the data instances into separate sets with similar classes. ID3 aims to select an attribute split that partitions the data instances into separate sets with similar classes. First, all possible attribute splits for a node are obtained, yielding pairs of subsets. A significance measure compares the individual subsets for each attribute split. ID3 uses entropy to calculate the significance within individual subsets of data instances.

Entropy is the amount of information represented by a collection of examples. In case that a collection of examples have the same class, the represented information is low. In contrast, when a collection of examples have different classes, the obtained information is high. Consequently, the objective is to split by attributes that minimize the entropy in each subset of data instances. For this, ID3 compares the average entropy by the subset of data instances from the resulting split with the original data instances' entropy. After comparing these values one receives the so-called information gain and the attribute split-step selects the attribute which maximizes it. Intuitively, the Information Gain reveals how much more classes a branch holds as opposed to its parent node. Furthermore, this algorithm splits unrelated attributes in each node and therefore leads to simpler decision trees (Queyranne, 1998).

However, one common problem of incorporating early stopping criterion is known as the horizon effect. The horizon effect terminates further induction of the tree too early, thereby avoiding promising attribute splits. There exist various methods to select the attribute to split. For example, exhaustive search, Gini coefficient, gain ratio, minimum description length, and chi-squared values, e.g., Pessimistic Error Pruning (PEP) and minimum-error pruning (Mansour, 1997; Frank, 2000).

2.2.2. Post-pruning

Post-pruning algorithms reduce the set of parameters for a given decision tree by inserting, removing, or replacing nodes and edges. The post-pruning framework first creates a tree that describes all possible attribute interactions. Then it tries to simplify the tree by removing redundant and non-critical sections of the decision tree. The replacement of the sub-tree that yields the best-estimated performance is selected to simplify the tree. This process repeats until a replacement leads to an insignificant performance increase. There exist two types of post-pruning methods called subtree replacement and subtree raising. Both methods consider a part of the decision tree given by a subtree. Subtree replacement selects a subtree and tries to replace it with a leaf node. Hence, this method declares a class label as a generalization of the respective subtree. In contrast, subtree raising selects a subtree and replaces it with the child node. The child node replaces the subtree, which indicates that the internal nodes represent unnecessary attribute interactions that a smaller set of variables can represent.

C4.5, introduced by Quinlan (1993), is a simple post-pruning algorithm for decision trees that uses the subtree replacement method. It is a bottom-up algorithm and an extension of the ID3 algorithm. Furthermore, C4.5 inputs an initial decision tree with every permutation of attribute interactions. Note that the post-pruning method can input arbitrary decision trees. The core idea of C4.5 is the assumption that a subset of examples from the given training set is misclassified by the given decision tree. This assumption gives a confidence interval as a pessimistic error rate that assists as a heuristic. Thus, a node replaces its subtree if the respective subtree's error estimate is lower than the node's estimated error. For a selected subtree, the error estimate is the weighted sum of error estimates in its descendant nodes. Intuitively, the pessimistic error rate has no confidence in the initial decision tree predictions since it estimates the error in the given data to be higher. A prominent extension of C4.5 is the Reduced error pruning (REP) algorithm (Quinlan,

1987). It utilizes a validation set that is a subset of the training data and tries to improve the decision tree's performance. Given a decision tree that classifies all examples in the training set as input parameters. REP replaces each node with a leaf node and chooses the class as a replacement that leads to an overall performance improvement of the decision tree (Elomaa and Kääriäinen, 2001). This process repeats until the decision tree's performance is decreasing according to evaluation on the validation data. The advantage of subtree replacement methods is the low complexity since each node is evaluated at most once (Elomaa and Kääriäinen, 2001).

Typically, subtree replacement methods for post-pruning work in a bottom-up fashion to replace a subtree by a single leaf node, e.g., Cost-Complexity Pruning (CCP), Critical Value Pruning (CVP), and Minimum Error Pruning (MEP) (Breiman et al., 1984; Mansour, 1997; Patel and Upadhyay, 2012, pg.66). The subtree raising methods for post-pruning are, for example, Error-Based Pruning (EBP), based on PEP, which is explained in detail in Patel and Upadhyay (2012).

2.3. Sum-Product Networks

SPNs are probabilistic graphical models first introduced by Poon and Domingos (2011) and combine deep learning with graphical models. SPNs represent structured directed acyclic graphs with sums and products as internal nodes connected by edges and leaf nodes modeling probability distributions over random variables. Furthermore, the nodes are connected by directed edges, and sum nodes connect weighted edges to their children's nodes. Like Bayesian and Markov Networks (Gogate, Webb, and Domingos, 2010), a SPN is modeling and reasoning uncertainty aiming to solve the unsupervised task of density estimation. SPNs model a compact representation of the distribution for a given data set by learning an estimator of the joint probability distribution over a set of variables (Gens and Domingos, 2013; Poon and Domingos, 2011). Compared to other probabilistic graphical models, an SPN structure allows computing the exact inference efficiently (Poon and Domingos, 2011). More precisely, the inference is the task of concluding the chance over a set of random variables. As a result, the complexity of calculating the joint probability is linear to the number of edges within a SPN (Poon and Domingos, 2011). Furthermore, the joint probability distribution is sufficient to calculate other statistical queries, e.g., the marginal and conditional probability. This allows SPNs to solve difficult tasks for various domains, e.g., speech recognition (Nicolson and Paliwal, 2020), language modeling (Cheng et al., 2014), or image classification (Sguerra and Cozman, 2016). For example, Poon and Domingos (2011) showed that SPNs are capable of predicting missing values by imputing missing pixels of an image. For example, SPNs complete tasks such as predicting missing values, e.g., performing image completion by imputing missing pixels of an image, as shown by Poon and Domingos (2011).

This chapter gives an introduction to Sum Product-Networks, focusing on the methods to learn the structure of an SPN from data. The following Subsection 2.3.1 presents the structure which gives a formal definition to Sum Product-Networks. Subsection 2.3.2 introduces the procedures for evaluating the probability of an SPN. Finally, the Subsections

2.4.1 and 2.4.2 introduce the framework to learn the structure of an SPN for a given data set.

2.3.1. Structure

An SPN structure is defined by sum, product, and indicator nodes as the leaf nodes. The leaf node of an SPN represents a random variable estimated by a probability function (Poon and Domingos, 2011). Moreover, the random variable allows to model distributions over different domains, e.g., continuous, categorical, or discrete variables. Additionally, a node's scope denotes the set of all random variables captured in all subsequent child nodes. As an example, the Figure 2.3 shows an SPN structure given by a sum node s_0 as the root node. An SPN can be defined recursively:

- A tractable univariate distribution is an SPN
- A product of SPNs with disjoint scopes is an SPN
- A weighted sum of SPNs with the same scope is an SPN, provided all weights are positive and sum to 1

Furthermore, this work adopts the definition by Poon and Domingos (2011) of valid SPNs. An SPN is defined as decomposable if the set of scopes for all children nodes of a product node are disjoint. An SPN is complete if all children nodes of a sum node have the same scope. An SPN is valid if it is decomposable and complete, which is assumed for the SPNs in this thesis.

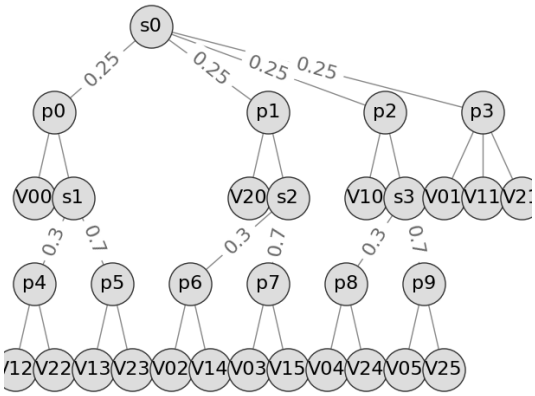

Figure 2.3.: This Figure shows SPN with a sum node s_0 as the root node. Each node's character prefix represents a node's type, either by p, s, or v representing product, sum, and leaf nodes, respectively. The suffix distinguishes different nodes from the same type by its scope. For leaf nodes, the suffix has two digits. The first digit indicates the scope of a leaf node. The second digit identifies individual leaf nodes. For example, the parent node of the leaf node v_{00} with scope 0 is p_0, and the parent node of the leaf node v_{03} with scope 0 is p_6. The leaf nodes v_{00} and v_{03} have the same scope, but their respective probability distributions differ.

2.3.2. Evaluation

The leaf nodes of an SPN model probability distributions and its scope corresponds to a random variable. For example, the leaf nodes v_{00} and v_{01} in Figure 4.4 define the same scope of 0 but differ in their probability distribution. Molina, Vergari, Mauro, et al. (2018) utilizes frequency distributions in the case of discrete random variables as leaves. Furthermore, the leaf nodes represent piecewise linear distributions to model continuous random variables (Molina, Vergari, Mauro, et al., 2018). In this thesis, we limit the leaf nodes of SPNs representing binary variables estimated with frequency distributions. To calculate the probability for a given configuration of input variables, an SPN evaluates the probability of its children nodes according to the operation given by its type. A leaf node evaluates its probability distribution by calculating the chance for a given event represented as an input value. Moreover, a sum node calculates the weighted sum of its children nodes, representing a mixture of dependent distributions. Furthermore, a product node evaluates the product of its children nodes, representing statistical independent distributions. Finally, an SPN evaluates a given set of input values for each random variable by evaluating the SPNs from the leaf nodes to the root node. The root node represents the joint probability distribution of the given input values in its descendant nodes. In this context, the joint probability is sufficient to calculate other statistical distributions, e.g., calculating the marginal distribution by imputing missing values. Thus, an SPN evaluates a given partial configuration of input values by marginalization, and the sum of partial configurations can compute the conditional probabilities. The marginal, conditional, and joint probability distributions evaluate in linear time dependent on the size of the SPN (Rooshenas and Lowd, 2014). Calculating the marginal and conditional probabilities in graphical models like Bayesian and Markov networks is usually intractable for larger treewidth (París, Sánchez-Cauce, and Díez, 2020; Gogate, Webb, and Domingos, 2010). Thus, SPNs have the advantage of a fast computation for inference tasks compared to other graphical models. The SPN structure allows to model a more complex distribution over many variables by utilizing a mixture of simpler distributions.

2.4. Structure Learning

The goal of structural learning for Sum-Product Networks is to find the optimal or near-optimal graph of an SPN. It incorporates methods from parameter learning for Sum-Product Networks, aiming to find the optimal parameters of an SPN for a given graph and data set (Poon and Domingos, 2011). There exist various approaches to learn the structure of an SPN, depicted in the chapter 3. In general, the structure of an SPN can be learned for a given data set compactly. The first structure learner is introduced by Poon and Domingos (2011). Its goal is to generate a valid SPN structure by utilizing a given data set. This algorithm starts with a valid and generic architecture of an SPN. Each instance of the data set is processed by running inference on it and updating the edges' weights. Standard methods to perform weight updates are Expectation Maximization or gradient descent (Dempster, Laird, and Rubin, 1977). This process terminates until a predefined convergence criterion is met. A common convergence criterion is to terminate weight updates if the improvement in likelihood concerning a validation set is insignificant. The resulting SPN is post-pruned

by removing zero-weight edges of all sum nodes. However, the initial generic structure of an SPN must be created by hand, which requires domain knowledge. There exist various algorithms that can learn the structure and weights of an SPN directly for a given data set introduced in the next subsections.

2.4.1. Learn-SPN

The first framework to learn the SPN structure for a given data set is introduced by Gens and Domingos (2013). Given a training data set D and a set of variables V, LearnSPN creates an SPN representing a distribution over V learned from D. The algorithm inputs a data matrix including rows and columns. The rows of the data matrix represent instances of D, and each column represents a variable of V. Previous work commonly used the LearnSPN framework, and various methods adapted the methods for splitting RVs and instances into subsets. Splitting instances assigns each instance to its most probable cluster. Each step of LearnSPN locally maximizes the posterior probability of the sub-SPN over the instances and variables. For example, the Expectation Maximization algorithm (Dempster, Laird, and Rubin, 1977) clusters data instances as shown in Gens and Domingos (2013). By which variables to split can be chosen by using a mutual information criterion to detect statistical dependence. Furthermore, a mutual information criterion measures the similarity between distributions since similar distributions are statistically independent. For example, Queyranne's algorithm (Queyranne, 1998) finds variable splits of two subsets with minimum empirical mutual information.

Algorithm 1: LearnSPN(D,V)

Data: set of instances D and set of variables V
Result: an SPN representing a distribution
over V learned from instances D
if $|V| = 1$ **then**
  return univariate distribution estimated
  from the variable's values in D;
else
  partition V into approximately
  independent subsets V_j;
  if *success* **then**
    return $\prod_j LearnSPN(D, V_j)$;
  else
    partition D into subsets of similar
    instances D_i;
    return $\sum_i \frac{|D_i|}{|D|} LearnSPN(D_i, V)$;
  end
end

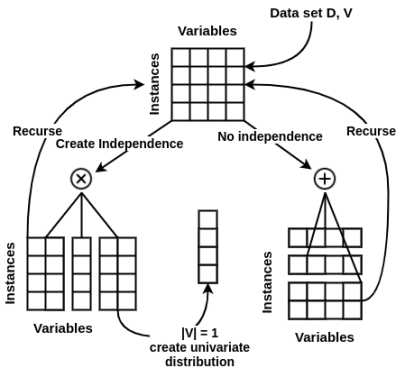

Figure 2.4.: The left side shows pseudo-code for the LearnSPN framework.
The right side shows the decomposition and conditioning steps for a data matrix.

The Figure 2.4 depicts the LearnSPN algorithm. Its goal is to grow a tree top-down by splitting the input data matrix by variables and instances. The Figure 2.4 visualizes the general framework of LearnSPN. The LearnSPN algorithm inputs a data matrix consisting of data instances D as rows and variables V as columns. In each iteration, the data matrix is either split among variables or data instances. Following, splitting a data matrix by variables is referred to as the decomposition step, depicted on the illustration's left side. Splitting a data matrix by data instances is called the conditioning step, shown on the illustration's right side. The LearnSPN algorithm starts at the root node with an initial data matrix. An independent statistical criterion determines whether to split the given data matrix by instances or variables. The decomposition step splits the variables V into a sub-set of variables tested for statistical independence. Note that the decomposition step is similar to the attribute split step for decision trees in the subsection 2.1, which tries to find the optimal attribute to split by. For the decomposition step, a mutual information criterion approximates the statistical independence between variables. The case of approximately independent subsets of variables results in generating a product node. Thus, the product node splits approximately independent variables. An edge connects the individual variable subsets to the generated product node, and LearnSPN recurses on each subset. In contrast, when the mutual information criterion is not satisfied, the conditioning step is initiated. The conditioning step clusters the data instances into subsets with similar instances. This case generates a sum node that connects each cluster by an edge, including a weight. The proportion of instances falling into a cluster indicates the weight of the associated edge. Hence, the sub-populations of a sum node represent dependent variables, and each sub-population contains similar instances. Furthermore, LearnSPN recurses on each sub-population of the sum node. In case that the data matrix contains a single variable V, a leaf node is created. The leaf node represents a univariate distribution estimated over the respective instances D. Following, creating a leaf node refers to the base case, depicted in Figure 2.4, illustrated as the case of a single column. During the conditioning and decomposition step, the LearnSPN framework recurses each subset of the data matrix. Hence, the algorithm is guaranteed to terminate, which ultimately leads to the base case.

2.4.2. Learn-MSPN

Molina, Vergari, Mauro, et al. (2018) introduced a pre-pruning algorithm for learning the structure of SPNs for hybrid domains. The Learning Mixed Sum-Product Networks (Learn-MSPN) algorithm enhanced the LearnSPN framework by modeling multivariate distributions over hybrid domains. Furthermore, Learn-MSPN automates selecting the parametric form of the random variables. This automation makes it easier for its users since they do not have to manually choose the distribution type. It can approximate any distribution from a mix of statistical types, e.g., continuous, discrete, or categorical distributions. As a result, Learn-MSPN follows the same paradigm as the LearnSPN framework by utilizing decomposition and conditioning steps, which lead to the base case. Moreover, the decomposition step of Learn-MSPN incorporates the independence measure randomized dependency coefficient (RDC), which applies to any statistical type of random variable. The RDC tests the independence between a subset of variables, which creates a product node when the variables are approximately independent. In contrast, if the sub-set of variables are not approximately independent, the conditioning step is used. The conditioning step

compares the similarity between subsets of data instances. This step is challenging for data instances with different statistical types. Hence, the conditioning step maps the data instances into a feature space by utilizing the value of the RDC. Finally, a similarity measure compares the mapped instances, e.g., K-Means, which splits the data matrix columns by a sum node (Molina, Vergari, Mauro, et al., 2018). Molina, Vergari, Mauro, et al. (2018) gives a detailed explanation for the decomposition and conditioning steps. In case that the variables are not statistically independent and the data is univariate, the base case is used. For the base case, frequency distributions or piecewise linear approximations estimate the data instances. The Learn-MSPN algorithm introduces two parameters to control the creation of a SPN by the min instance slice (*mis*) and rdc-threshold (*rdc-threshold*). The *rdc-threshold* is incorporated during the decomposition step when comparing two subsets of variables. It is zero if the variables are independent, whereas a higher *rdc-threshold* indicates more correlations between them. Statistical independence between variables for a given data set is rare. Thus the *rdc-threshold* can be used to loosen the constraint of absolute independence. By loosening this constraint, the variables should be split by a product node if the RDC value is smaller than the glsrdcth.

Therefore the next chapter introduces recent approaches for pre- and post-pruning SPNs, which utilize statistical independence tests to split variables using product nodes.

3. Related Work

We discussed structure learning and pruning algorithms for decision trees in the chapter 2. This chapter gives a historical introduction to structure learning and reducing redundancies in SPNs with pruning algorithms. We will look at each category from bottom-up to top-down approaches, pre- and post-pruning methods while focusing on the methods which create statistical independence of random variables for SPNs. Post-Pruning methods for SPNs are not well researched yet. Previously, research focused on learning its structure from a given data set. Structure learning is necessary since creating an SPN structure by hand requires domain knowledge, often not available. However, current structure learning techniques create SPNs with numerous parameters that increase computational cost (Lowd and Rooshenas, 2013). Thus, post-pruning methods are necessary to decrease the set of parameters for a given SPN while maintaining its performance.

LearnSPN introduced a framework, which for a given SPN structure, generates a tree by splitting random variables or data instances as described in Section 2.4.1. In the following, the decomposition step remains a crucial method for structure learners, which is implemented and enhanced in several forms, e.g., as shown in Gens and Domingos (2013). However, the authors found that splitting variables in two subsets with minimum empirical Mutual Information (Queyranne, 1998) was too slow. Instead, a pairwise statistical independence test aims to find appropriate variable splits. Moreover, using the statistical independence test to split variables has no negative influence on the likelihood for the resulting sub-SPNs (París, Sánchez-Cauce, and Díez, 2020). The following paragraphs introduce previous work that gradually improved the top-down LearnSPN framework (Chow and Liu, 1968; Vergari, Mauro, and Esposito, 2015; Breiman, 2001).

Rooshenas and Lowd (2014) criticizes that previous work captures variable interactions indirectly through implicit latent variables. A latent variable models specific information by inferring some information from the observed variables. For example, the probability of winning a chess game given the current game state is a latent variable. The observed factors of a chess game, such as the moves taken by each player and the possible future moves each player can take, infer the probability of winning a chess game. Thus, a latent variable depends on the context given by the observed variables. The authors proposed a state-of-the-art architecture ID-SPN (Rooshenas and Lowd, 2014) top-down algorithm to analyze dependencies between variables directly and indirectly. By adopting the LearnSPN framework, ID-SPN utilizes arithmetic circuit Markov networks (ACMN) nodes introduced by Rooshenas and Lowd (2013), which capture a mixture of distributions (París, Sánchez-Cauce, and Díez, 2020). The multivariate distribution represented by an ACMN node captures the dependencies between variables directly. Like the decomposition and conditioning steps in LearnSPN, ID-SPN replaces ACMN nodes with either sum or product nodes (Rooshenas and Lowd, 2013). In case that the likelihood with respect to a validation set improves, the ACMN node replaces a sum node. Else, a product node splits the sub set of variables. The decomposition step of ID-SPN captures dependencies between variables indirectly.

Dennis and Ventura (2012) introduced a top-down structure learning algorithm BuildSPN. BuildSPN adapts the structure learning framework's variable decomposition step by focusing on clustering subsets of variables instead of data instances. Initially, it builds a so-called regional graph representing a sum node separating similar clusters of data instances. To cluster data instances Dennis and Ventura (2012) used the K-Means algorithm. The scopes within a subpopulation of a regional graph are partitioned recursively into sub-scopes. Note that a scope represents a unique variable. Each sub-scope adds a region node to the region graph, which can be seen as sum nodes. Hence region nodes split the sub-scopes by data instances. For more information on expanding regional nodes, Dennis and Ventura (2012) gives a more detailed explanation. The K-Means algorithm is used again to split each region node's variables by creating a partition node. A partition node represents a product node that splits variables by similar variable clusters. Note that the variable clusters are compared pairwise to variable clusters with an equal set of scopes. Highly correlating variable cluster pairs are split by variables, thus introducing a latent variable. The latent variable represents the interactions between the two variable clusters by their respective distribution. Therefore, each latent variable reduces the set of parameters to represent a similar distribution depicted by the regional graph. A partition node is inserted above the regional node during the decomposition step, which splits the variables, including their scope, from the remaining subpopulations. This approach utilizes partition nodes to split variables by individual scopes, similar to the method used in this thesis. However, the initial regional graph does not consider the dependencies when splitting the data by its instances. Hence, this pre-pruning method eliminates dependencies in regional nodes as well. This local elimination means that the conditioning step may separate highly dependent variables among different regional nodes (Dennis and Ventura, 2012). Additionally, the SPN size and the time to learn the structure can become exponentially large as the number of variables of the data increases (París, Sánchez-Cauce, and Díez, 2020).

Peharz, Geiger, and Pernkopf (2013) introduced a bottom-up pre-pruning algorithm to learn the structure of a SPN. The General Merge Learning framework starts with simple models over small variable scopes and aims to grow larger, more complex over larger variable scopes. This framework adopts the notation of Dennis and Ventura (2012) by introducing region nodes, partition nodes, and regional graphs, which is explained in detail in Peharz, Geiger, and Pernkopf (2013) and Dennis and Ventura (2012). Starting from the bottom by the leaf nodes, the General Merge Learning framework introduces select candidates to merge by variables. Its goal is to select advantageous merge candidates. Consequently, a merging strategy that preserves a valid SPN structure is desired. K distinct region nodes initially separate the merge candidates with the same scope. This separation ensures that merge candidates are compared pairwise and have the same set of scopes. Moreover, the merge candidates are select by a Bayesian-Dirichlet independence test by utilizing a validation set to select winner variables (Peharz, Geiger, and Pernkopf, 2013). Winner variables represent highly correlating merge candidates. A parent node is inserted above the current region node when merging a set of merge candidates. As a result, the parent node splits variables with higher correlation from the remaining merge candidates. However, it is a greedy approach that only selects the best scoring likelihood among the features to reduce the learned structure's high complexity (Peharz, Geiger, and Pernkopf, 2013).

To date and our knowledge, Rahman and Gogate (2016) is the first and only post-pruning algorithm that introduced the idea to merge similar sub-SPNs. This method is post-pruning an initial SPN structure by merging similar sub-SPNs. For this, the LearnSPN framework generates the initial SPN structure for a given data set. The algorithm searches for leaf nodes with the same set of scopes by examining each node within a sub-SPNs. To clarify, each sub-SPN with the same set of scope represents a merging candidate. Furthermore, a node qualifies as a merging candidate if its respective sub-SPN constitutes the same structure as the sub-SPNs of the remaining merge candidates. Moreover, pairs of sub-SPNs have the same structure if the alignment of nodes and edges are identical. The distance between the probability distributions of the merging candidates guides as a similarity criterion for the respective sub-SPNs. In case that the sub-SPNs are similar, the merging step is initiated, which unites both sub-SPNs in a bottom-up fashion. First, the data sets used to generate the pairs of sub-SPNs merge. Then, a sub-SPN representing the pairs is generated by adopting the same structure and learning the weights by minimizing the Log-Likelihood utilizing the merged training set. As a consequence, the weights are chosen such that the accuracy over the validation set improves. In the same way, merging the candidates aims to replace redundancies of similar sub-SPNs. The steps on how the individual sub-SPNs are merged into the initial SPN structure are explained in detail in Rahman and Gogate (2016). However, this results in tree SPNs that have to be converted to graph SPNs again, which is prone to infinite loops (Rahman and Gogate, 2016). When using bagging of tree SPNs and graph SPNs, this algorithm outperforms other algorithms, e.g., ID-SPN (Rooshenas and Lowd, 2014), for high dimensional data sets (París, Sánchez-Cauce, and Díez, 2020).

Despite the advantages and disadvantages of pre-pruning algorithms for decision trees, previous research did not focus on investigating the benefits and pitfalls of post-pruning algorithms for SPNs yet. For this reason, this thesis introduces a simple post-pruning algorithm for valid SPN structures introduced in the next chapter.

4. Pruning for Sum Product Networks

Existing pruning methods for SPNs are primarily limited to pre-pruning methods (París, Sánchez-Cauce, and Díez, 2020). Structure learners create SPNs that perform better than SPN structures by hand (Dennis and Ventura, 2012). Additionally, the learned structures have an increased accuracy and inference speed compared to various Graphical Models (Gens and Domingos, 2013; Gogate, Webb, and Domingos, 2010; Ko et al., 2020). However, the resulting SPNs have the disadvantage of increased computational cost due to the large size in parameters (Lowd and Rooshenas, 2013). This work seeks to overcome this problem by focusing on post-pruning methods for valid SPNs.

The work of Rahman and Gogate (2016) introduced the idea of merging similar sub-SPNs for a given set of tree SPNs with identical structures. However, this post-pruning method is restricted to tree SPNs and assumes an identical structure among a set of sub-SPNs. Furthermore, tree SPNs have more parameters that need a translation to graph SPNs (Rahman and Gogate, 2016). This translation creates the overhead of transferring the tree SPNs to graph SPNs, which is computationally expensive (Rahman and Gogate, 2016; París, Sánchez-Cauce, and Díez, 2020). To overcome the problem of restricting the initial SPN to tree SPNs, we propose a simple bottom-up post-pruning method that directly applies to valid SPNs. In opposition, this thesis proposes a post-pruning method that forgoes the need for a validation set as a heuristic.

The proposed post-pruning algorithm's main idea is to loosen the constraint of dependent random variables represented by its respective leaf node by merging similar leaf nodes. Structure learners incorporate various statistical independence tests during the variable decomposition step (Molina, Vergari, Mauro, et al., 2018; Rooshenas and Lowd, 2014; Dennis and Ventura, 2012). In case that a statistical independence test fails to discover statistical dependence, the conditioning step is initiated. As a result, the leaf nodes estimated by a data set remain dependent within a sub-SPN when generating a sum node. With this in mind, loosening the constraint of statistical dependence among variables allows separating the aforementioned leaf nodes. Consequently, approximately independent leaf nodes can be merged by a single leaf node. As a consequence, similar leaf nodes become obsolete through the merged node, which reduces the set of parameters in a sub-SPN. To conclude, a sub-SPN can introduce a merged node as an independent variable. To clarify, the merged node is split from its remaining scopes by creating a product node above its parent node. Just as for post-pruning decision trees, the proposed approach is similar to subtree raising introduced in the Subsection 2.2.2.

In this context, the chapter focuses on two problems introduced by merging leaf nodes. First, addressing the problem of merging similar merge candidates with a similarity measure. Second, proposing an approach to retain a valid SPN structure while merging a set of potential merge candidates. The following sections divide into three parts to address these issues. The first part revolves around introducing the initial SPN structure for which the proposed algorithm's invariant performs successfully. Secondly, the post-pruning algorithm

for a given SPN structure will be described in detail. And finally, analyzing the relation between potential merge candidates and reducing the number of parameters to the proposed algorithm.

4.1. Sum Product-Network with alternating Structure

Section 2.3 introduced valid SPNs consisting of product nodes, sum nodes including their respective directed edges with weights, and univariate distributions represented by its leaf nodes. This definition for valid SPNs is guaranteed in most cases when learning an SPN structure for a given data set with the described pre-pruning methods in Section 2.4.1. Whereas removing one of the conditions from a sub-SPN within a given SPN leads to an invalid SPN structure.

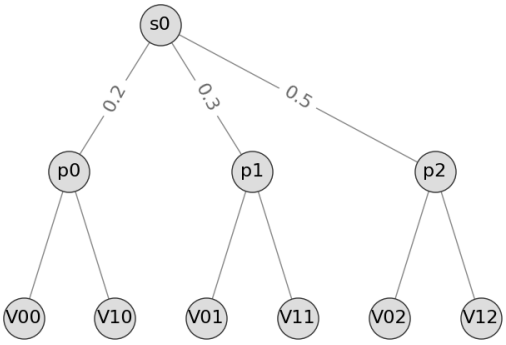

Figure 4.1.: This Figure shows the SPN structure with s_0 as its root node. It connects three product nodes, p_0, p_1, and p_2 with the respective weights at their edges. Each product node has two leaf nodes with two different scopes, $0, 1$ as its child nodes. The leaf nodes represent binary distributions as the random variables.

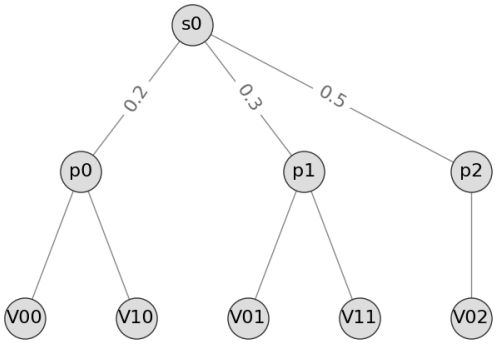

Figure 4.2.: Removing the leaf node v_{12} with scope 1 from the product node p_2 with the parent weight of 0.5 leads to an invalid SPN structure. The structure is invalid since it contradicts the rule of disjoint scopes in a product node's child nodes.

For example, the simple structured SPN depicted in Figure 4.1 shows a valid SPN structure. In fact, removing one of the leaf nodes v_{10}, v_{11}, v_{12} with scope 1 from one of the product nodes p_0, p_1, or p_2 invalidates the SPN structure. This operation contradicts the rule of disjoint scopes in the children nodes of a product node. The Figure 4.2 shows the SPN structure when removing the leaf node with scope v_{12} from its parent node, the product node p_2 with the weight of 0.5 at its edge. This structure for an SPN is invalid since the sum node s_0 has the scope $\{0, 1\}$, which means that the product node p_2 needs to incorporate the same scope as its parent node s_0. Particularly, the requirements for a valid SPN structure have to be respected to incorporate post-pruning methods for SPNs. In the case of replacing

the parameters of an SPN, such as insertion or deletion of nodes, it is necessary to retain a valid SPN structure.

With this in mind, the proposed post-pruning algorithm described in Section 4.2 requires the following conditions for a given SPN:

- The children nodes of a sum node are either product nodes or leaf nodes.
- The children nodes of a product node are either sum nodes or leaf nodes.

This requirement assures that the same type of nodes does not occur in its descendant or ascendant nodes. To clarify, this means that every layer in the SPN structure alternates between sum and product nodes with indicator nodes at its leaves. Following, an SPN with the above conditions for its structure refers to an SPN with an alternating structure or, in short, alternating SPN. In conclusion, these assumptions lead to a successful invariant of the proposed algorithm explained in Section 4.2.

Following, an alternating SPN is abbreviated as SPN and denoted as $\mathcal{G}$. The following paragraphs analyze all properties and regional information of an arbitrary leaf node v in the last layers of a given SPN $\mathcal{G}$. The parent node w_v of a leaf node v can have two different types. As a consequence, either the parent node w_v is a product node or a sum node of v.

In case that the parent node w_v is a product node, it follows that the product node p_w is its parent node or, in contrast, w_v refers as the root node of $\mathcal{G}$.

- In case that w_v is the root node, the SPN $\mathcal{G}$ is called a naive SPN. The naive SPN consists of one product node, which splits all leaf nodes by its scopes. This type of SPN is already minimal by the number of its parameters. Hence pruning is not necessary for $\mathcal{G}$.
- In case that w_v is not a root node, it has a parent node p_w. By the definition of an alternating SPN, the type of the parent node p_w is a sum node. The product node w_v splits the leaf nodes by the set of scopes given by its parent node p_w. Following, we refer to this structure as a sub-SPN. This sub-SPN is a valid SPN according to Section 2.3 when chosen as a root node. This case is the desired structure for the invariant of the proposed post-pruning algorithm. The algorithm aims to replace individual sub-SPNs with valid sub-SPNs with a reduced set of parameters and a similar semantic.

When the parent node w_v is a sum node it has either a parent node p_w or w_v s the $\mathcal{G}$'s root node.

- In case that w_v is the root node, the SPN $\mathcal{G}$ contains a single sum node, representing the weighted sum of all leaf nodes with scope v. Hence, the sum node w_v represents a sum of leaf nodes with a unique scope. A single leaf node can represent the leaf nodes within the sub-SPN since the leaf nodes' underlying probability distributions have the same set of parameters. The condition above allows merging a pair of leaf nodes into a single distribution. One approach is to merge the leaf nodes' underlying distribution parameters by the weighted sum for the individual subpopulations' edges (Clemen and Winkler, 1999; Rahman and Gogate, 2016).

- In case that w_v is not a root node, it has a parent node p_w. At first glance, this structure seems unnecessary since the product node p_w already splits the scope of v by the sum node w_v. However, in this case, the sub-populations of the sum node w_v can be summarized by a single leaf node, as explained in the previous paragraph. This summarization eliminates the sum node w_v by replacing the sum over the sub-populations by a single leaf node v, and v depicts the underlying distribution of w_v. Furthermore, this operation reduces the set of parameters of the SPN $\mathcal{G}$.

This section introduced the alternating SPN structure for the proposed post-pruning algorithm explained in the following sections. It should be noted that the internal nodes can represent sum nodes while pruning. Due to the bottom-up approach, a product node can be inserted between a leaf and sum node. Admittedly, the alternating structures are required to be strict for the last layers within a SPN. Although, this work assumes SPN structures generated with the Learn-MSPN algorithm introduced by Molina, Vergari, Mauro, et al. (2018). Moreover, any valid SPN structure generated by the Learn-MSPN can be transferred to an alternating SPN structure with the operations mentioned above.

4.2. Simple Post-Pruning Algorithm for Sum-Product Networks

This section proposes a simple post-pruning method for a given SPN called Simple Post-Pruning Algorithm for Sum-Product Networks (SPA-SPN). For the remaining sections, the SPN introduced in Section 4.1 is assumed. This chapter first explains the post-pruning algorithm's general scheme, then each step of the individual methods is explained in detail in the following subsections. Finally, the last section analyzes the reduction in parameters for the number of discovered merge candidates.

When pre-pruning methods utilize the decomposition step while learning an SPN structure, the data matrix reduces as the SPN layers increase. As the layers increase and data instances reduce, the number of samples in the data matrix vanishes. This reduction means that leaf nodes estimated by a distribution use a smaller set of data instances (Rahman and Gogate, 2016; Molina, Vergari, Mauro, et al., 2018; París, Sánchez-Cauce, and Díez, 2020). Furthermore, the estimated distributions may represent a lousy approximation of the observed data due to the limited amount of available data. Based on this assumption, this results in SPNs modeling potentially independent variables as statistically dependent. Thus, SPA-SPN's main idea is to generalize over similar leaf nodes by loosening the constraint of dependent leaf nodes. By loosening this constraint, similar leaf nodes can be generalized by a single merged leaf node or as a set of similar merged leaf nodes. The merged leaf node then depicts the combined probability distributions of the initial leaf nodes.

For a given SPN, SPA-SPN reduces its parameters by merging leaf nodes within a sub-SPN with equal scopes. The algorithm works in a bottom-up fashion by starting at the leaf nodes at the last layer of the SPN. 2 depicts the SPA-SPN framework, which inputs a scope k, a threshold t and an SPN denoted as $\mathcal{G}$ and output a pruned SPN $\mathcal{G}'$.

The algorithm's first step is to identify the possible merge candidates within the SPN $\mathcal{G}$. The $acquireSubSpns$ method in 2 indicates this step. Due to the structure defined in Section 4.1, the leaf nodes with scope k belong to unique sub-SPNs. Moreover, a leaf node belongs

Algorithm 2: SPA-SPN($\mathcal{G}$, k, t)

Data: a valid SPN $\mathcal{G}$, scope k and threshold t
Result: a valid SPN $\mathcal{G}'$
initialize $\mathcal{G}'$ as $\mathcal{G}$;
$\mathcal{G}_S = acquireSubSpns(\mathcal{G}')$;
for $g \in \mathcal{G}_S$ **do**
  $\mathcal{C}$:= leaf nodes in g with scope k;
  $\mathcal{M} = mergeLeafs(\mathcal{C}, t)$;
  $g' = IntegrateMergeCandidates(g, \mathcal{M})$;
  replace g with g' in $\mathcal{G}$;
  $\mathcal{G}_S = acquireSubSpns(\mathcal{G}')$;
end
return $\mathcal{G}'$;

to the first sum node in its ascendant nodes, representing a sub-SPN. For example, the leaf nodes v_{12} and v_{13} in Figure 4.2 belong to the sub-SPN with the sum node s_1 as the root node. Likewise, each sum node occurring in the last layers of the SPN represents a sub-SPN. Due to the definition of an SPN, every sub-SPN remains a valid SPN structure. Following $\mathcal{G}_S$ denotes the set of sub-SPNs that contain leaf nodes with equal scope k in its descendant nodes. Due to the definition in section 4.1, the sub-populations of a sub-SPN hold product nodes. Our definition SPNs implies that the children nodes of the product nodes, as mentioned earlier, are leaf nodes. Note that each product node defines the same set of scope as the sum node given by a sub-SPN. Furthermore, this indicates that the individual leaf nodes of all product nodes within a sub-SPN define the same set of scope. This regional information of unique sub-SPNs $\mathcal{G}_S$ allows SPA-SPN to merge leaf nodes belonging to a unique sub-SPN g to a single leaf node.

Sub-sections 4.2.1 and 4.2.2 give a detailed explanation of merging and integrating the leaf nodes within a sub-SPNs. The method $acquireSubSpns$ collects the sub-SPNs $\mathcal{G}_S$ for a given scope k. The SPN $\mathcal{G}$ searches for sub-SPNs $\mathcal{G}_S$, including a given scope k through breadth-first search. This method collects the sub-SPNs, representing the first sum node in the ascendant nodes of the leaf nodes with scope k. These leaf nodes signify potential merge candidates since the same set of parameters defines their respective distributions. $MergeLeafs$ inputs a threshold t and the potential merge candidates as leaf nodes denoted as $\mathcal{C}$ and returns the merged leaf nodes $\mathcal{M}$. Section 4.2.1 explains this method in detail. The threshold t specifies the tendency of merging the given leaf nodes. A higher threshold indicates to merge leaf nodes more dissimilar. In contrast, a lower threshold signifies to merge leaf nodes more similar to each other.

$IntegrateMergeCandidates$ inputs the resulting merge candidates $\mathcal{M}$ and the respective sub-SPN denoted as g and output a sub-SPN g'. Section 4.2.2 gives a detailed explanation of this method. The graph g' incorporates the merge candidates $\mathcal{M}$ while maintaining a valid SPN structure of the respective sub-SPN g'. Depending on the merged leaf nodes $\mathcal{M}$, g' either splits the scope k by a product node from the initial sum node or incorporates a sum node with a decreased size in parameters. The sub-SPN g' replaces the initial sub-SPN g within the SPN $\mathcal{G}$. Finally, this process repeats for all sub-SPNs $\mathcal{G}_S$, which yields the

pruned SPN $\mathcal{G}'$.

4.2.1. Merge similar Leaf Nodes

The statistical dependence gives insights into the similarity between two random variables (París, Sánchez-Cauce, and Díez, 2020; Rahman and Gogate, 2016). Statistical measures estimate the statistical dependence between random variables. In literature, frequently used statistical measures are divergence measures (Murphy, 2012, pg.57). For example, a prominent divergence measure is the Kullback-Leibler divergence (KLD) (Joyce, 2011; Rahman and Gogate, 2016). The KLD evaluates the statistical dependence of two arbitrary distributions, P, and Q. Let P and Q be arbitrary distributions with parameters p_i and q_i and $i \in N$ with N representing the number of outcomes. The following equation defines the Kullback-Leibler-Divergence given by:

$$\mathrm{KL}(P\|Q) = \sum_{i}^{N} p_i \log_2 \frac{p_i}{q_i} \tag{4.1}$$

From an information-theoretic perspective, the KLD represents the average number of extra bits needed to encode data from a distribution P instead of using Q (Murphy, 2012, pg.57).

In contrast to divergence measures, a metric incorporates the distance between arbitrary distributions. Divergence measures as the KLD are asymmetric. Hence divergence measures cannot directly be incorporated as distances. The Jensen–Shannon divergence (JSD) represents a symmetric version of the KLD (Lin, 1991; Murphy, 2012). For arbitrary probability functions P and Q, the JSD is defined by:

$$\begin{aligned} M &= \frac{1}{2}(P+Q) \\ JSD(P\|Q) &= \frac{1}{2}\mathrm{KL}(P\|M) + \frac{1}{2}\mathrm{KL}(Q\|M) \end{aligned} \tag{4.2}$$

The JSD is a valid metric, according to Lin (1991). Its bound constraints on $0 \leq JSD(P\|Q) \leq 1$ and $JSD(P\|Q) = JSD(Q\|P)$ denotes its symmetry. This definition of JSD is for univariate distributions, and Nguyen and Vreeken (2015) extends it to the multivariate case. The JSD provides insight into the similarity of two arbitrary probability distributions. The smaller the distance between two probability distributions, the more similar they are. If the value is 0, the parameters of the probability distributions are identical (Murphy, 2012). In contrast, the higher the distance between two probability distributions, the more dissimilar they are.

In the following, the similarity between two leaf nodes serves as an abbreviation for the similarity of two probability distributions represented by their respective leaf nodes. On the one hand, this work assumes that the insight given by a similarity distance between two distributions is sufficient to reduce the set of parameters for a given SPN. In the same way, the merged leaf node serves as an approximation for a set of highly similar leaf nodes. On the other hand, we assume that a similarity measure's introduced loss is insignificant for the entire SPN. For this reason, SPA-SPN aims to identify leaf node candidates that depict a similar distribution. For instance, the JSD allows identifying similar leaf nodes

from a set of merge candidates. To clarify, the more similar a pair of leaf nodes are, the closer the similarity distance is to zero. In contrast, the more dissimilar two leaf nodes are, the closer the similarity distance is to one. Therefore, the similarity distance determines whether a single leaf node can represent a pair of leaf nodes or not. For instance, a pair of leaf nodes merge in case that the similarity criterion is satisfied. As a consequence, the leaf nodes unite by taking the mean of their probability distributions (Clemen and Winkler, 1999; Rahman and Gogate, 2016).

The leaf nodes v_{ki} with scope k and the indexes $i \in \mathcal{S}$ denote the possible merge candidates, where $\mathcal{S}$ denotes the indexes of the leaf node within a sub-SPN. The children nodes of the sub-SPN represent $|\mathcal{S}| = m$ as the potential merge candidates. SPA-SPN selects the most similar leaf nodes from the potential merge candidates resulting in the merge candidates. Moreover, the similarity matrix contains the permutation of all leaf nodes within a sub-SPN. Furthermore, it evaluates the similarity between a set of leaf nodes. The similarity matrix abbreviates as $s : \{(v_{ki}, v_{kj})|(i,j) \in \mathcal{S}^2\} \rightarrow \{x|x \in \mathbb{R} : 0 \leq x \leq 1\}$ which compares the similarity between two arbitrary leaf nodes with scope k. It should be noted that the similarity measure compares the underlying probability distributions represented by their respective leaf node. The following similarity matrix can evaluate all permutations of similar leaf nodes for a given scope k:

$$S(\{(v_{kj}, v_{ki})|(j,i) \in \mathcal{S}^2\}) \coloneqq \begin{pmatrix} s(v_{k0}, v_{k0}) & \cdots & s(v_{k0}, v_{km}) \\ \vdots & \ddots & \vdots \\ s(v_{km}, v_{k0}) & \cdots & s(v_{km}, v_{km}) \end{pmatrix} \tag{4.3}$$

By utilizing this similarity matrix's properties, a pair of leaf nodes can expose as merge candidates. Since pruning aims to reduce the set of redundant parameters of a valid SPN, merging two identical leaf nodes is not desired. Hence, ignoring the matrix's trace is appropriate since it measures the similarity between two identical leaf nodes. Due to the chosen similarity measure's symmetry, the lower triangular matrix is sufficient to choose valid merging candidates. Moreover, this work assumes that merging two leaf nodes with the lowest similarity distance is the best option since they represent the most similar leaf nodes. Thus, the best candidates are chosen greedily by merging leaf nodes with minimum similarity distance. To clarify, the Figure 3 depicts the framework to find merge candidates utilizing a similarity matrix. The similarity matrix compares pairs of leaf nodes utilizing the equation 4.2. Then, the pair with the lowest distance is chosen as merge candidates. The selected merge candidates result in a new leaf node representing the initial pair's united probability distributions by taking their mean. Removing the initial pair from the set of potential merge candidates results to an updated set of candidates. As a consequence, the set of potential merge candidates contains the merged leaf node. Furthermore, the merged node represents the initial leaf nodes. Finally, this process repeats until either the similarity matrix is single-valued or a prespecified stopping criterion is satisfied. In case that the matrix is single-valued, there exist no merge candidates. In the same way, this work chooses a threshold t to obtain an early stopping criterion. As a result, the threshold gives an upper bound for the dissimilarity between leaf nodes concerning the similarity measure. To summarize, the process of evaluating the similarity and merging is repeated until either the possible merge candidates consist of a single value, e.g., $|\mathcal{P}_k| = 1$, or the lower triangular matrix does not satisfy the similarity constraint given by a threshold t.

In conclusion, the Chapter 5 analyzes the impact of the threshold for the SPA-SPN post-pruning method. The following Section 4.2.2 explains how to integrate the merged leaf nodes into the sub-SPN.

Algorithm 3: mergeLeafs($\mathcal{P}_{km}$, t)

Data: set of candidate $\mathcal{P}_{km}$ with scope k containing m leaf nodes and a threshold t
Result: a set of merged leaf nodes $\mathcal{P}_k$
if $|\mathcal{P}_{km}| = 1$ **then**
  return $\mathcal{P}_{km}$;
else
  $\mathcal{P}_{ks} := \{s(v, v')|v, v' \in \mathcal{P}_{km}\}$;
  if $min(\mathcal{P}_{ks}) > t$ **then**
    return $\mathcal{P}_{km}$;
  else
    $v, v' = \underset{(v,v')\in\mathcal{P}_{ks}}{arg\,min}(\mathcal{P}_{ks})$;
    $v_{new} =$ combination of v, v';
    $\mathcal{P}_{km} := \{v_{new}\} \cup \mathcal{P}_{km} \setminus \{v, v'\}$;
    return $mergeLeafs(\mathcal{P}_{km}, t)$;
  end
end

4.2.2. Integrate Merge Candidates

Subsection 4.2.1 introduced a method to merge leaf nodes by a similarity measure. This subsection focuses on integrating the merge candidates $\mathcal{M}$ into the given sub-SPN. Hence, assume a sub-SPN given by a sum node with n children nodes and m different scopes. Furthermore, $\mathcal{S}$ denotes the set of indexes for the n sub-populations of the sum node g and their respective scopes m. The indexes of the attributes of a leaf node denote as $\mathcal{A}$. Additionally, the edges connect the n children nodes of the sum nodes with weights w_i where $i \in \mathcal{S}$ and $|\mathcal{S}| = n$ holds. Consequently, each child node of the sum node is a product node with v_{ji} as child nodes, also the index j indicate the scope associated with the leaf node given by $j \in \mathcal{N}$ and $j < m$. It should be noted that the index i for a leaf node v_{ji} distinguishes individual leaf nodes with the same scope from each other since each node represents different probability distributions. Moreover, the sub-populations containing leaf nodes in its descendant nodes share a similar distribution for the attribute with the index k denoted as $\mathcal{P}_k$. As a result, this equation holds $v_{q,k} \sim v_{r,k} \forall q \in \mathcal{P}_k, r \in \mathcal{P}_k$. The following equation describes the sub-SPN mentioned above by:

$$\sum_{i\in\mathcal{S}} w_i \prod_{j\in\mathcal{A}} v_{j,i} \tag{4.4}$$

For simplicity, the following paragraphs assume the merged leaf nodes $\mathcal{M}$ contains a single merged leaf node. In addition to this, the equation $|\mathcal{M}| = 1$ holds, and the final paragraph of this sub-section covers the case when $|\mathcal{M}| > 1$.

First and foremost, we assume that the sum node's sub-populations are product nodes with child nodes v_{ij} as leaf nodes. In this case, the node v_{new} represents the combined leaf node for the attribute with index k. For this reason, the equation $v_{new} \in \mathcal{M}$ holds. Furthermore, $\mathcal{P}_k$ represents the indexes of the leaf nodes that are similar to the merged leaf node. As a result, the equation $v_{new} \sim v_{r,k} \forall r \in \mathcal{P}_k$ holds. To clarify, the merged leaf node v_{new} represents the leaf nodes $v_{r,k}$ with scope k and $r \in \mathcal{P}_k$ in case that the condition $|\mathcal{P}_k| = |\mathcal{S}|$ holds. Subsequently, these assumptions lead to the following equation starting from equation 4.4:

$$\begin{aligned} & \sum_{i \in \mathcal{S}} w_i \prod_{j \in \mathcal{A}} v_{j,i} \\ & = \sum_{i \in \mathcal{S}} w_i v_{k,i} \prod_{j \in \mathcal{A} \setminus \{k\}} v_{j,i} \\ & \overset{v_{new} \sim v_{ki}}{\Longrightarrow} \sum_{i \in \mathcal{S}} w_i v_{new} \prod_{j \in \mathcal{A} \setminus \{k\}} v_{j,i} \\ & = v_{new} \sum_{i \in \mathcal{S}} w_i \prod_{j \in \mathcal{A} \setminus \{k\}} v_{j,i} \end{aligned} \tag{4.5}$$

Importantly, equation 4.5 shows that a leaf node v_{new} can be isolated from its respective sum node. In fact, the probability distribution represented by the leaf node v_{new} is similar to the set of leaf nodes with the same scope k and indexes $\mathcal{P}_k$. Similarly, removing the leaf nodes with the same scope from their respective product nodes results in creating a new product node above the sum node. Moreover, the product node separates the merged leaf node v_{new} from the sum node and scopes $\mathcal{A} \setminus \{k\}$. For example, Figure 4.4 shows the isolation of the leaf node v_{new} from its sub-SPN.

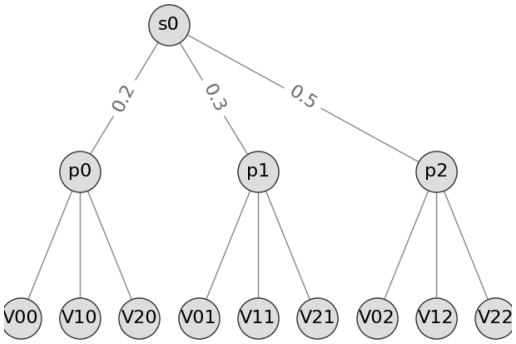

Figure 4.3.: A simple SPN structure with a sum node s_0 as the root node. The sub-populations of s_0 are given by the product nodes $p_0, p1$ and $p2$. Each product node splits the set of scopes $\mathcal{A}$ with leaf nodes.

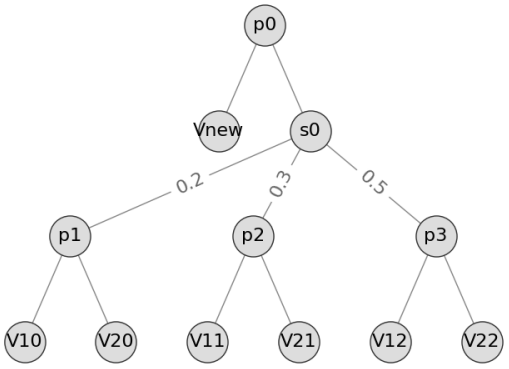

Figure 4.4.: Pruned SPN where all leaf nodes with scope 0 can be represented as the leaf node v_{new}.

In case that potential leaf nodes within a sub-SPN are merging candidates, the edge of the sum node is removed. As a consequence, a new product node is set above it in order to

maintain a valid SPN structure.

Generally, not all leaf nodes with the same scope are similar when incorporating different thresholds, as described in Subsection 4.2.1. As a result, a single leaf node cannot represent the leaf nodes with indexes $\mathcal{S}$. This assumption implies that similar leaf nodes with indexes $\mathcal{P}_k$ do not solely include viable merge candidates. Hence, the indexes of the sub-populations do not contain the merging candidates with $\mathcal{S} \setminus \mathcal{P}_k$. Furthermore, the leaf nodes for the sub-population indexes are dissimilar to the merge candidates, e.g. $|\mathcal{P}_k| < |\mathcal{S}|$ and $|\mathcal{S} \setminus \mathcal{P}_k| > 1$. Thus, v_{new} is similar to a subset of leaf nodes with scope k within a sub-SPN. However, v_{new} can represent the leaf nodes that qualified as merge candidates. This assumption consequently removes redundancies when integrating v_{new} into the sub-SPN with the following equation:

$$
\begin{aligned}
&\sum_{i \in \mathcal{S}} w_i \prod_{j \in \mathcal{A}} v_{j,i} \\
&= \sum_{i \in \mathcal{S} \setminus \mathcal{P}_k} w_i \prod_{j \in \mathcal{A}} v_{j,i} + \sum_{i \in \mathcal{P}_k} w_i \prod_{j \in \mathcal{A}} v_{j,i} \\
&\overset{v_{new}}{\Longrightarrow} \sum_{i \in \mathcal{S} \setminus \mathcal{P}_k}^{n} w_i \prod_{j \in \mathcal{A}} v_{j,i} + v_{new} \sum_{i \in \mathcal{P}_k} w_i \prod_{j \in \mathcal{A} \setminus \{k\}} v_{j,i} \\
&\overset{\sum w=1}{\Longrightarrow} \sum_{i \in \mathcal{S} \setminus \mathcal{P}_k}^{n} w_i \prod_{j \in \mathcal{A}} v_{j,i} + (1 - \sum_{i \in \mathcal{S} \setminus \mathcal{P}_k} w_i)(v_{new} \sum_{i \in \mathcal{P}_k} w_i \prod_{j \in \mathcal{A} \setminus \{k\}} v_{j,i}) \\
&\overset{\sum w=1}{\Longrightarrow} \sum_{i \in \mathcal{S} \setminus \mathcal{P}_k} w_i \prod_{j \in \mathcal{A}} v_{j,i} + (1 - \sum_{i \in \mathcal{S} \setminus \mathcal{P}_k} w_i)(v_{new} \sum_{i \in \mathcal{P}_k} (\frac{w_i}{\sum_{s=\mathcal{P}_k} w_s} \prod_{j \in \mathcal{A} \setminus \{k\}} v_{j,i})
\end{aligned}
\tag{4.6}
$$

For example, given a sub-SPN with 3 child nodes as product nodes, e.g., $\mathcal{S} := \{0, 1, 2\}$ and each product node has 2 leaf nodes with their respective scopes $0, 1$, e.g., $\mathcal{A} := \{0, 1\}$. Now assume that the leaf nodes v_{00} and v_{01} are similar, e.g., $\mathcal{P}_k := \{0, 1\}$. Thus, the leaf node v_{new} can represent the leaf nodes v_{00} and v_{01}, since $v_{00} \sim v_{01}$. The following equation incorporates the leaf node v_{new}:

$$
\begin{aligned}
&\sum_{i \in \mathcal{S}} w_i \prod_{j \in \mathcal{A}} v_{j,i} \\
&w_0 v_{00} v_{10} + w_1 v_{01} v_{11} + w_2 v_{02} v_{12} \\
&\overset{v_{new} \sim (v_{00}, v_{01})}{\Longrightarrow} w_0 v_{new} v_{10} + w_1 v_{new} v_{11} + w_2 v_{02} v_{12} \\
&= v_{new}(w_0 v_{10} + w_1 v_{11}) + w_2 v_{02} v_{12} \\
&\overset{\sum_{i=0} w_i = 1}{\Longrightarrow} (1 - w_2) v_{new}(w_0 v_{10} + w_1 v_{11}) + w_2 v_{02} v_{12} \\
&\overset{\sum_{i=0} w_i = 1}{\Longrightarrow} (1 - w_2) v_{new}(\frac{w_0}{w_0 + w_1} v_{10} + \frac{w_1}{w_0 + w_1} v_{11}) + w_2 v_{02} v_{12}
\end{aligned}
\tag{4.7}
$$

Clearly, this example did not reduce the number of parameters in the sub-SPN. The implication on parameters when integrating merge candidates into sub-SPNs is discussed in Subsection 4.3.

Finally, the next paragraphs discuss the case when there exists more than one merge candidate. Hence, the equation $|\mathcal{M}| > 1$ holds. In this case, different clusters of leaf nodes

in $\mathcal{M}$ are similar to each other. Each leaf node in $\mathcal{M}$ represents a similar node to a subset of merge candidates with indexes $\mathcal{P}_k$. Hence, the index of leaf nodes similar to a leaf node in $\mathcal{M}$ is defined as $\mathcal{P}_z$, where $z \in \mathcal{M}$. Furthermore, the family of sets $\mathcal{P}_P := \{\mathcal{P}_z | \forall z \in \mathcal{M}\}$ describes the indexes for leaf nodes similar to a unique merged node z. Hence, the equation $z \sim v_{i,k} \forall i \in r \forall r \in \mathcal{P}_z \forall z \in \mathcal{M}$ holds. Note that the leaf nodes similar within $\mathcal{M}$ have different indexes among all other leaf nodes in $\mathcal{M}$. This equation ensures that individual indexes of leaf nodes representing individual merge candidates which means they can be represented by a individual sum node irrespective the remaining leaf nodes. Hence the equation $z' \cap z'' = \emptyset, \forall z' \in \mathcal{P}_P \setminus z'', \forall z'' \in \mathcal{P}_P \setminus z'$ holds for their indexes. The following equation replaces individual leaf nodes within the sub-SPN by their respective leaf nodes in $\mathcal{M}$:

$$
\begin{aligned}
&\sum_{i \in \mathcal{S}} w_i \prod_{j \in \mathcal{A}} v_{j,i} \\
&\overset{def. \mathcal{P}_P}{\iff} \sum_{i \in \mathcal{S} \setminus \mathcal{P}_P} w_i \prod_{j \in \mathcal{A}} v_{j,i} + \sum_{z \in \mathcal{M}} \sum_{i \in \mathcal{P}_z} w_i \prod_{j \in \mathcal{A}} v_{j,i} \\
&\overset{def. \mathcal{M}}{\implies} \sum_{i \in \mathcal{S} \setminus \mathcal{P}_P} w_i \prod_{j \in \mathcal{A}} v_{j,i} + \sum_{z \in \mathcal{M}} z \sum_{i \in \mathcal{P}_z} w_i \prod_{j \in \mathcal{A} \setminus \{k\}} v_{j,i} \\
&\overset{\sum w = 1}{\implies} \sum_{i \in \mathcal{S} \setminus \mathcal{P}_P} w_i \prod_{j \in \mathcal{A}} v_{j,i} + \sum_{z \in \mathcal{M}} \sum_{i \in \mathcal{P}_z} w_i (z \sum_{i \in \mathcal{P}_z} w_i \prod_{j \in \mathcal{A} \setminus \{k\}} v_{j,i}) \\
&\overset{\sum w = 1}{\implies} \sum_{i \in \mathcal{S} \setminus \mathcal{P}_P} w_i \prod_{j \in \mathcal{A}} v_{j,i} + \sum_{z \in \mathcal{M}} \sum_{i \in \mathcal{P}_z} w_i (z \sum_{i \in \mathcal{P}_z} \frac{w_i}{\sum_{s \in \mathcal{P}_z} w_s} \prod_{j \in \mathcal{A} \setminus \{k\}} v_{j,i})
\end{aligned}
\tag{4.8}
$$

Section 4.3 analyzes the relationship between the number of merge candidates and the decrease in parameters of individual sub-SPNs. Chapter 5 covers the empirical evaluation of the methods introduced in this chapter.

4.3. Reduction in Parameter by merging similar leaf nodes

This section introduces the reduction in parameter size for a sub-SPN when merging similar leaf nodes. Equations 4.5 and 4.6 suggests two cases of integrating the merge candidate, e.g., $|\mathcal{M}| = 1$ into the respective sub-SPN. The first case relates when all leaf nodes are similar to each other, represented by a single leaf node. In contrast, the second case presents a merge strategy when a part of the leaf nodes are similar. Both cases can reduce the set of parameters for the given sub-SPN, thus reducing the size of parameters for the given SPN. The following paragraphs cover the reduction of parameters of the sub-SPN for both cases. In general, by merging $|\mathcal{P}_k|$ leaf nodes represented by one leaf node v_{new}, the leaf nodes, and their respective edges can be removed from each product node. Thus, the number of parameters for a given sub-SPN decrease by the number of leaf nodes and their respective edges $|\mathcal{P}_k| * 2$. The edge of a product node represents an edge with a weight of 1. If a single leaf node represents all leaf nodes within a sub-SPN, e.g., $\mathcal{P}_k = \mathcal{S}$, the leaf node generates independence by inserting a product node above the sub-SPN, which increases the number of parameters. Hence, the product node mentioned above adds one parameter.

Splitting the leaf node v_{new} and the initial sum node with a product node adds two edges. Finally, the leaf node v_{new} introduces one more parameter to the sub-SPN which increases the total number of parameters by 4. Thus, the decrease in parameter by integrating the merged leaf nodes can be described with $|\mathcal{P}_k| * 2 - 4$, which is linear by the number of merged leaf nodes $|\mathcal{S}|$.

If a leaf node v_{new} can represent only a subset of the leaf nodes, the parameters' decrease differs from the above approach. Removing $|\mathcal{P}_k|$ leaf nodes from the sub-SPN introduces a new product node as the sub-SPNs child node. The product node splits the leaf node v_{new} from the remaining leaf nodes with indexes $\mathcal{S} \setminus \mathcal{P}_k$. The product node, therefore, adds two new edges to split the leaf nodes. Furthermore, each remaining leaf node with indexes $\mathcal{S} \setminus \mathcal{P}_k$ needs to be separated by an additional sum node, which increases the number of parameters. The number of parameters of the given sub-SPN increases in total by 4. Thus, the decrease in parameter by integrating the merged leaf nodes can be described with $|\mathcal{P}_k| * 2 - 4$, which is linear by the number of merged leaf nodes $|\mathcal{P}_k|$. Finally, in case that more than one merge candidate exists, e.g., $|\mathcal{M}| > 1$ is discussed. Each merged leaf node $z \in \mathcal{M}$ represents the leaf nodes with indexes $\mathcal{P}_z \subseteq \mathcal{P}_P$ and all leaf nodes have different indexes. Moreover, equation 4.8 merges each z by integrating the individual leaf nodes with indexes $\mathcal{P}_z$ as in the previous paragraph. Hence each merge reduces the same parameters by $|\mathcal{P}_z| * 2 - 4$ for all merges, thus $\sum_{\mathcal{P}_z \in \mathcal{P}_P} |\mathcal{P}_z| * 2 - 4$ which is linear to the number of merge candidates in $\mathcal{M}$.

5. Evaluation and Experiments

This chapter covers the empirical evaluation of the post-pruning method presented in Chapter 4. For this, the metrics introduced in Section 5.1 evaluate the SPA-SPN algorithm. Five different benchmark data sets empirically evaluate the SPA-SPN, explained in detail in Section 5.2. Section 5.4 presents two different baselines that constitute the initial SPN structures. Furthermore, the Section 5.3 explains the steps to replicate the experiments and the results. Finally, Section 5.5 shows and discusses the experimental results. Finally, Section 6 highlights the most critical findings of this chapter and proposes future work ideas.

5.1. Evaluation Metrics

This section proposes three evaluation metrics to compare the results with related work on pruning algorithms for SPNs.

The model quantity metric utilizes the individual statistics of an SPN structure. This metric compares two SPN structures by their statistics in parameters. The number of edges, leaf nodes, sum nodes, and product nodes represents the individual statistics. Furthermore, the change in parameters when pruning an initial SPN structure depicts the impact of post-pruning methods. These statistics give valuable information for the general structure of an SPN. For example, many product nodes indicate more variable splits suggesting that an SPN depicts independent statistical variables. In contrast, a high amount of sum nodes suggests that more variables are statistically dependent. Finally, the evaluation speed of an SPN depends on the number of edges. Hence the change in parameters gives valuable information about differences in inference speed when comparing SPNs.

A prominent evaluation metric for the model quality is the likelihood which inputs a test data set T and a model $\mathcal{G}$ to evaluate. The likelihood of an SPN is calculated by observing the given data concerning the model parameters of an SPN. Hence, the likelihood compares unseen data points to the distribution estimated by an SPN. The log of the likelihood introduces an efficient calculation of the likelihood for a given model. Furthermore, to compare the log-likelihood for different domains, the log-likelihood is normalized by the number of available test data. Hence, the average Log-Likelihood (avg. LL) is defined by:

$$\begin{aligned} \mathcal{L}(T|\mathcal{G}) &= \frac{1}{|T|} \log \prod_{t \in T} \mathcal{G}(t) \\ &= \frac{1}{|T|} \sum_{t \in T} \log \mathcal{G}(t) \end{aligned} \tag{5.1}$$

The goal of modeling an SPN distribution is to find parameters that describe the observed data distribution well. Hence, the likelihood is a suited evaluation metric for this problem.

Maximizing the likelihood function's value indicates that the model parameters estimate data more likely to the observed data. Utilizing the avg. LL simplifies the calculation and encourages the comparison of experiments with different test data. An SPNmodels the probability for a given configuration. Hence its likelihood function is bound between 0 and 1. The log of the likelihoods will always be negative, and higher values indicate a model that fits the data well. It is negative because the log of the likelihood constrains to the interval $(-\infty, 0)$. In contrast, a log-likelihood value with smaller values indicates a model's parameters do not fit the data well. For example, the $\lim_{x \to 0} \log(x) = -\infty$ and $log(1) = 0$. To calculate the avg. LL, the probability distribution of an SPN evaluates each sample t from the given test set T. The log of the resulting values represents the intermediate probability values called logits. Additionally, the logits' sum represents log-likelihood. Finally, normalizing the log-likelihood by the number of instances yields the avg. LL. It is generally attempted to maximize a given model's log-likelihood, but there is a considerable trade-off between generalization and overfitting. This trade-off is a broadly discussed topic among the machine learning community (Bartoldson et al., 2020). A given model should generalize the observed data by giving reasonable estimates for unseen data. In contrast, a model should not overfit and remember each evident sample while giving a wrong estimate for unseen data.

Finally, introducing the conditional marginal Log-Likelihood (CMLL) as an evaluation metric. In contrast to the likelihood, the marginal likelihood measures the likelihood of data irrespective of other variables. For this, the test data set variables split Tinto a query set Q and an evidence set E. The following formula computes the CMLL:

$$CMLL(T|\mathcal{G}) = \sum_{t \in T, e \in E} \log \mathcal{G}(t|e) \tag{5.2}$$

The CMLL calculates the log of the likelihood for a subset of data given by the evident data. Imputing the missing values in Q by the sum over all possible events is called the marginal probability. The marginal probability utilizes the joint probability of a given evident sample and the remaining outcomes of events. Hence, the evaluation criterion is called the conditional marginal log-likelihood CMLL. Calculating the marginal probability of a model gives the probability of an event when ignoring other variables. Therefore, calculating the CMLL of a given test set model depicts how well the model's parameters can be estimated when imputing missing data.

5.2. Datasets

This section introduces the data sets[1] which evaluate the SPA-SPN algorithm from section 4.2. For this evaluation, the five different real-world data sets NLTCS, msnbc, plants, Netflix, and kdd are used. The data sets are mainly compiled by Haaren and Davis (2012), Rooshenas and Lowd (2013), Lowd and Davis (2010), and Bekker et al. (2015). Furthermore, Lowd and Davis (2010) post-processed the given data sets, mainly used to evaluate pre-pruning SPN algorithms (Chow and Liu, 1968; Poon and Domingos, 2011;

[1] https://github.com/arranger1044/DEBD

Molina, Vergari, Mauro, et al., 2018; Rooshenas and Lowd, 2014; Rahman and Gogate, 2016; Peharz, Geiger, and Pernkopf, 2013). Table 5.1 depicts the statistics of the different data sets in ascending order for the quantity in variables.

Karolien Geurts[2] donated the following data sets containing anonymized traffic accident data (Geurts et al., 2003). The data originates from traffic accidents with injured or deadly wounded casualties on a public road in Belgium. Accidents have been filled in a form by police officers for the period from 1991 until 2000. Furthermore, the data contains a rich context of information on the accident circumstances, the course of accidents, the traffic conditions, the environmental conditions, the road conditions, the human conditions, and the traffic data's geographical conditions. The dataset contains 111 different boolean variables, which are either true or false for the respective statement associated with the variable. An explanation of the individual variables can be found in Geurts et al. (2003).

The KDD Cup 2000 clickstream prediction data was obtained from an online retailer, consisting of web session data. It consists of 64 boolean variables where each variable indicates whether a session visited a web page matching a particular category (Lowd and Davis, 2010). A detailed description of the features can be found at the Washington computer science course[3].

The MSNBC data set was initially obtained from the UCI machine learning repository (Newman and Merz, 1998), and a compiled version is available at the kdd website[4]. The MSNBC anonymous web data contains information about whether a user visited a top-level MSNBC page[5] during a particular session for the entire day of 28th of September 1999. Each sequence in the data set represents a user's page view during a twenty-four hour period where each event represents a user request for a page. The pages are categorized by "frontpage", "news", "tech", "local", "opinion", "on-air", "misc", "weather", "health", "living", "business", "sports", "summary", "bbs" (bulletin board service), "travel, "MSN-news" and "MSN-sports" (Lowd and Davis, 2010). Furthermore, the entries of the instances in the data set contain boolean variables for each category visited. The variable for a category is true if the user visited a particular page associated with the category during a session. The average number of visits per user is 5.7, with a total number of 989818 users. Per category, the number of URLs ranges from 10 to 5000 with 17 categories. A detailed description of the categories can be found on the msnbc web page[5].

The plants data set consists of various plant types and the associated location where they are found can be found in Lowd and Davis (2010). Each variable in the data set represents a location, which is 1 if found there else 0.

The Netflix data set[6][7] is a random subset of the Netflix challenge 67 data and is compiled initially by Lowd and Davis (2010). The data set's instances represent user ratings, and the variables represent the hundred most rated movies. A particular user rates a movie represented by a boolean variable where 1 means it is rated and 0 means not rated.

[2] http://fimi.ua.ac.be/data/
[3] https://courses.cs.washington.edu/courses/csep546/12sp/psetwww/1/kddOrganizerReport.pdf
[4] http://kdd.ics.uci.edu/databases/msnbc/msnbc.data.html
[5] https://www.msnbc.com
[6] https://web.archive.org/web/20090925184737/
[7] http://archive.ics.uci.edu/ml/datasets/Netflix+Prize

NLTCS[8] (The National Long Term Care Survey) data set contains long time samples of disabled persons living in a community or institution above age 65 (Rooshenas and Lowd, 2014; Lowd and Davis, 2010). The dataset contains 16 binary variables of functional disability measures: 6 activities of daily living and 10 instrumental activities of daily living, pooled over 1982, 1984, 1989, and 1994 waves of the survey with 21574 data points.

Table 5.1.: Statistics of the individual data sets consisting of the number of training, test, and validation instances, including the number of variables.

dataset	no. of training instances	no. of test instances	no. of validation instances	no. of variables
nltcs	16181	3236	2157	16
msnbc	291326	58265	38843	17
kdd	180092	34955	19907	64
plants	17412	3482	2321	69
netflix	15000	3000	2000	100

5.3. Setup

This section covers the essential steps to evaluate the SPA-SPN algorithm from section 4.2 on the data sets introduced in section 5.2. The Learn-MSPN (Molina, Vergari, Mauro, et al., 2018) algorithm generates the initial SPNs. Additionally, the spflow[9] library implements the Learn-MSPN framework (Molina, Vergari, Stelzner, et al., 2019).

The Learn-MSPN algorithm introduces hyperparameters, which result in the initial SPN structures. Post-pruning the initial SPN structures with various configurations of hyperparameters encourages analyzing the effects of post-pruning algorithms for SPNs. Hence, the following paragraphs describe the hyperparameters and their configurations' effects on the initial SPN structures. Generally, the column parameter and *mis* values represent the hyperparameters for the Learn-MSPN framework. The column parameter defines the statistical independence method to use during the decomposition step. Furthermore, the column parameter in this work restricts to the **skips** and **rdc** methods. The skips parameter omits the decomposition step and instead initiates the conditioning step. Thus, the generation of the SPN structure solely depends on the *mis* parameter. The *mis* value creates a sum node if the minimum number of instances is satisfied, splitting the data instances. Violating this condition leads to a single product node that splits the given variables individually. The rdc column parameter introduces the *rdc-threshold* parameter used during the decomposition step. Learn-MSPN uses the rdc method to identify statistical independence among variables, explained in Section 2.4. Additionally, the rdc method utilizes the *mis* value during its conditioning step. Furthermore, a higher *mis* value indicates that more instances

[8] https://www.niagads.org/datasets/ng0007
[9] https://github.com/SPFlow/SPFlow

are required to create a sum node. This constraint leads to more children nodes as more instances are available to cluster the instances into subpopulations. In case that the number of instances does not satisfy the condition of the *mis* value, a product node is created, which splits the individual variables. A good rule of thumb is to use an *rdc-threshold* between 0.1 and 0.3 and 0.1 as the *mis* value.

The SPA-SPN algorithm has two parameters given by a threshold (*th*) and a scope. The scope parameter identifies the leaf node to prune for a given SPN. The *th* indicates the condition to merge similar leaf nodes. A similarity distance compares the similarity for a pair of lea nodes. In the following experiments, the JSD, introduced in Section 4.2.1, serves as a similarity measure. Two leaf nodes merge by combining the mean of their respective probability distributions, which results in the distribution of the merged leaf node (Clemen and Winkler, 1999). Finally, the SPA-SPN scope parameter selects the scope for which to prune an SPN. It defines the available leaf nodes to use as possible merging candidates. The sequence of pruning by different scopes is extensive since the permutations given by the number variables. For simplicity, a given SPN is pruned for all scopes individually in arbitrary order. For example, an SPN with 16 scopes 0, ..., 15 is pruned in ascending order of given scope 0 to 15 individually. This process is repeated for each scope until the structure of an SPN does not change.

5.4. Influence of the Parameters on the Learned Structures

This section examines the influence of the column parameter for creating the SPNs structures with the Learn-MSPN algorithm. Section 5.3 explains the configuration of methods for generating the SPNs structures with different column parameters. The resulting SPNs structures represent the initial SPNs for the SPA-SPN algorithm. The following paragraphs discuss the initial structures generated with two different configurations. The paragraphs discuss the general trends for all data sets on the example of the NLTCS and MSNBC data set. First, discussing the effects of the rdc method and then the skips method. Generally, the figures visualize the initial SPNs structures generated with the rdc method, e.g., Figure 5.1. In contrast, Table 5.2 depicts the statistics for the initial SPNs structures generated with the skips method for all data sets. The appendix provides figures for the remaining data sets in A.2 and A.1 gives a description on how to read figures by an example.

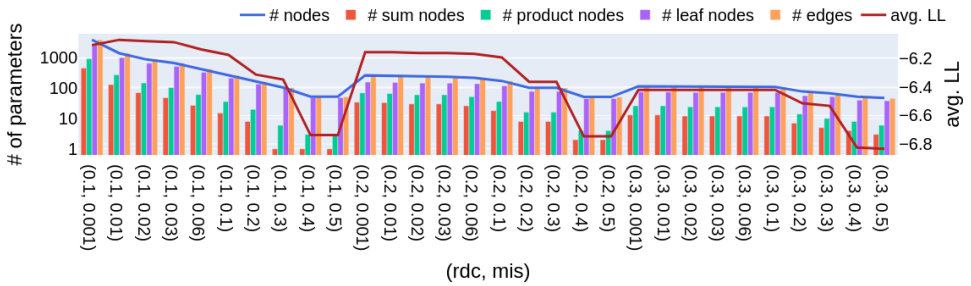

Figure 5.1.: This figure represents the initial SPNs structures learned on the NLTCS data set by the Learn-MSPN algorithm with the column parameter set to **rdc**.

This graph depicts the initial statistics of the SPNs learned with the column parameter set to **rdc**. The number of parameters decreases when the *mis* value increases and constant *rdc-threshold* values. Additionally, the avg. LL decreases as the number of parameters decrease. There exist one outlier for the avg. LL with an *rdc-threshold* value 0.01 and *mis* value 0.0001. For this tuple, the avg. LL is -6.06 and 1392 nodes. Furthermore, the avg. LL is significantly worse for SPNs with a higher share of sum nodes, e.g., the *rdc-threshold* and *mis* tuples for (0.1,0.4), (0.1,0.5), (0.2,0.4), and (0.2,0.5). The number of leaf nodes has the highest proportion in the number of nodes, followed by product nodes and sum nodes. The number of edges has the most significant fraction in the number of parameters. The number of edges holds the highest fraction because they connect each node within the SPNs. Generally, the number of parameters is more extensive for lower *rdc-threshold* and *mis* values. A lower *rdc-threshold* value restricts statistical independence between the variables. Furthermore, restricting the statistical independence test leads to more dependent variables since more conditioning steps initiate. Finally, initiating more conditioning steps leads to a higher number of sum nodes. Thus, a lower *rdc-threshold* value minimizes the chance to find statistical independence among variables leading to fewer product nodes. The *mis* value indicates the minimum number of instances that must be present to create a sum node. The conditioning step creates a leaf node when the number of instances contains fewer items than the *mis* value. Hence, a higher *mis* value limits the frequency of conditioning steps since the number of sum nodes decrease.

The NLTCS data set in Table 5.2 depicts the statistics for the SPNs structures generated by the Learn-MSPN algorithm with the column parameter set to skips. For this configuration, the increase of the *mis* value leads to an increase in avg. LL. Furthermore, the number of edges decreases from 9923 to 523. In contrast, the SPNs generated with the rdc column parameter show a higher proportion of edges than leaf nodes. This proportion difference occurs when a single sum node serves as a naive SPNs to split the data instance by the product nodes. In contrast to the rdc method, the skips method utilizes fewer parameters to reach the avg. LL of -6.165.

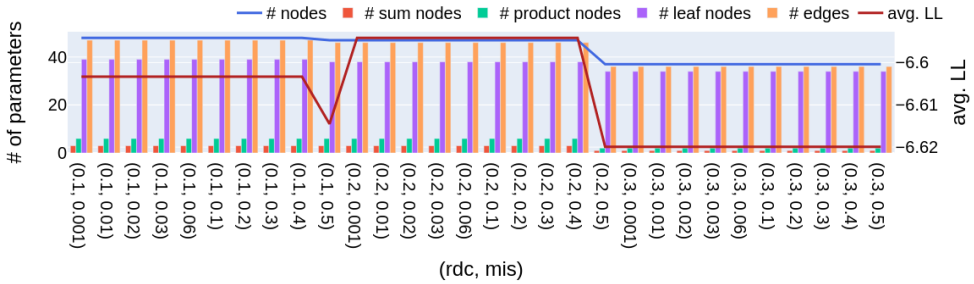

Figure 5.2.: This figure represents the initial SPN structures learned on the MSNBC data set by the Learn-MSPN algorithm with the column parameter set to **rdc**.

Figure 5.2 depicts the statistics for the initial SPNs learned with the rdc as a column parameters. For this configuration, the statistics do not change significantly for different configurations of *mis* values. Furthermore, an increasing *mis* value does not affect the number of sum nodes, ranging from 1 to 3 sum nodes for all the SPNs. A higher proportion of product nodes indicates that the decomposition step repeatedly discovers statistical independence among variables. However, an increasing rdc value has an insignificant impact on the SPN structure since a low rdc value is sufficient to split the variables.

In contrast, the SPN structure generated with the column parameter set to skips leads to different results. For this configuration, an increasing *mis* value leads to an average LL range from -6.339 and -6.556. Moreover, the available number of parameters for the SPNs depicted in Table 5.2 is much higher for lower *mis* values than for the configuration used in Figure 5.2.

Table 5.2.: The initial SPN structures generated with the Learn-MSPN and the column parameter is set to skips. Each block consists of five different SPN structures with the respective data set from the first column. On the vertical line's right-hand side, each structure's associated quantity and quality measures referring to the SPN generated by the mis value depicted in the second column.

data set	mis	avg. ll	# nodes	# sums	# products	# edges	# leafes
nltcs	0.001	-8.164	9924	991	2004	9923	6929
	0.01	-6.12	2473	167	351	2472	1955
	0.02	-6.087	1229	70	155	1228	1004
	0.03	-6.092	977	54	123	976	800
	0.06	-6.128	524	22	57	523	445
msnbc	0.001	-6.117	12335	1247	2542	12334	8546
	0.01	-6.109	2913	219	462	2912	2232
	0.02	-6.163	1743	120	257	1742	1366
	0.03	-6.202	1223	77	166	1222	980
	0.06	-6.286	531	25	56	530	450
kdd	0.001	-2.387	33830	1298	2613	33829	29919
	0.01	-2.209	7871	180	369	7870	7322
	0.02	-2.203	4391	119	240	4390	4032
	0.03	-2.205	4331	119	239	4330	3973
	0.06	-2.264	2164	60	120	2163	1984
plants	0.001	-15.548	44467	1535	3106	44466	39826
	0.01	-13.614	8939	188	402	8938	8349
	0.02	-14.127	5238	86	195	5237	4957
	0.03	-14.549	3510	38	96	3509	3376
	0.06	-15.491	1790	11	38	1789	1741
netflix	0.001	-60.847	152105	1123	2679	152104	148303
	0.01	-56.803	13943	5	142	13942	13796
	0.02	-57.084	6970	1	69	6969	6900
	0.03	-57.251	5253	1	52	5252	5200
	0.06	-57.829	2627	1	26	2626	2600

5.5. Evaluation of SPA-SPN

Section 5.4 introduced the initial structure of the SPNs trained on the different data sets with the Learn-MSPN algorithm. Each of the graphs in Figure 5.3 shows the relationship between *th*, *mis*, and avg. LL for a fixed *rdc-threshold* value. The SPA-SPN algorithm utilizes a threshold abbreviated as *th* to decide whether to merge a pair of leaf nodes by a similarity measure. Furthermore, the similarity measure utilized is the JSD explained in detail in Section 4.2.1. Section 5.2 introduced the data sets used to evaluate the given SPNs structures. The initial SPNs structures are created with the Learn-MSPN algorithm and its respective hyperparameters *rdc-threshold* and *mis* values with two different column parameters, skips, and rdc. Additionally, a *th* value of 0 indicates that no merge candidates are selected and refer to the initial SPNs structures. For example, Figure 5.3 with *mis* value 0.06, *rdc-threshold* value 0.1, and *th* value of 0 refer to the initial SPNs structure obtained by the Learn-MSPN algorithm with column parameter set to rdc. The SPNs with a *th* value of 0 refers to the initial SPNs structures introduced in Section 5.4. The evaluation metrics

from Section 5.1 relate to the figures in the following subsections. These figures show heat maps with the individual metric as the color concerning the parameters *th*, *mis*, and *rdc-threshold*. For example, a darker shade indicates a lower avg. LL value.

5.5.1. NLTCS data set

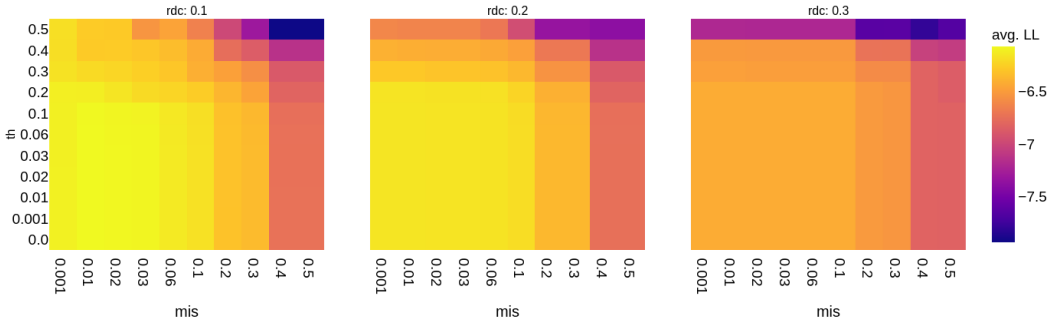

Figure 5.3.: This Figure depicts the **avg. LL** concerning the hyperparameters. Learn-MSPN generated the SPN structures for th 0 on the NLTCS data set and column parameter set to **rdc**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

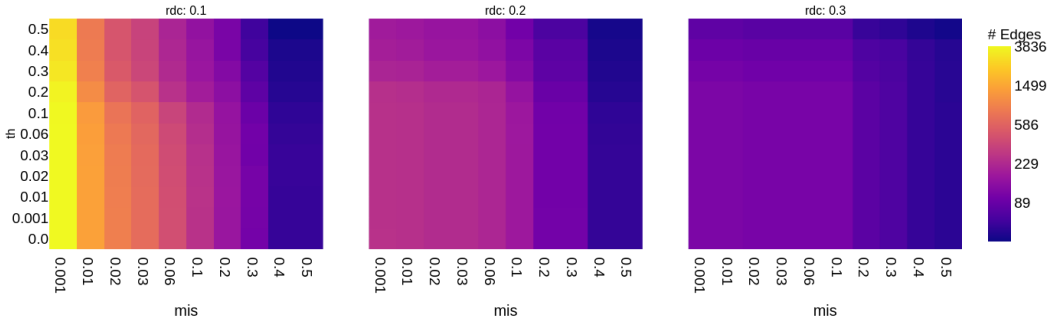

Figure 5.4.: This Figure depicts the **number of edges** concerning the hyperparameters. Learn-MSPN generated the SPN structures for th 0 on the NLTCS data set and column parameter set to **rdc**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

Figure 5.3 shows the avg. LL, and Figure 5.4 depicts the number of edges concerning the hyperparameters for the pruned initial structures with column parameter **rdc**. Each graph with a *th* value of 0 represents the avg. LL of the initial SPN structures ranging from -6.1 to -7. Generally, the *th* value in the range from 0 to 0.4 has no significant impact on the avg. LL of the resulting SPNs. For this configuration, a lighter transition in the shade indicates the change in avg. LL for increasing *th* values. Furthermore, each graph depicts this trend for different *rdc-threshold*. For the SPNs with an *rdc-threshold* of 0.1,

the number of edges decreases with an increasing *th* value. The linear transition of the shades in Figure 5.4 and *rdc-threshold* of 0.1. This increase in the number of edges is shown in 5.3, indicated by the linear transition in shades for increasing *th* values. This increase indicates that the initial SPN structures depict the same performance as the pruned SPNs while decreasing their parameters. For example, the configuration with *rdc-threshold* value 0.1 and *mis* values in range 0.03 to 0.1 initially depicts an avg. LL of approximately -6.1 with a *th* value of 0. For this configuration, the number of edges ranges from 900 to 400. The increasing *th* values ranging from 0.001 to 0.4 lead to SPNs with an avg. LL of -6.2. Additionally, the number of edges decreases in the range from 400 to 100. In contrast, the initial SPN structures with a small number of edges generally decrease rapidly in avg. LL through pruning. For example, the *rdc-threshold* values 0.2 and 0.3 in Figure 5.4 depict this decrease in likelihood. Furthermore, these configurations show a slight decrease in the number of edges represented by an abrupt change to a darker shade in Figure 5.3. Generally, the SPA-SPN algorithm decreases the number of parameters for SPNs with a higher initial number of edges which can be seen for smaller *mis* values and increasing *th*. The diagonal shades from the bottom-left to the top-right corner in Figure 5.4 become darker for increasing hyperparameters for *rdc-threshold* 0.1. Additionally, for these structures, the decrease in avg. LL is insignificant, which can be seen by the slight change in the shade in Figure 5.3 for RDC 0.1. In contrast, initial SPN structures with a small size in parameters lead to structures with a significantly worse performance in likelihood. At the same time, the decrease in the number of edges is insignificant. The best results for pruning are achieved for *mis* values between 0.01 and 0.06 because the change in avg. LL is insignificant and the number of parameters decrease strongly.

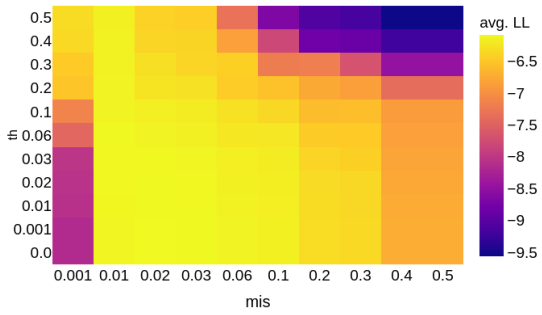

Figure 5.5.: This Figure depicts the **avg. LL** concerning the hyperparameters. Learn-MSPN generated the SPN structures for th 0 on the NLTCS data set and column parameter set to **skips**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

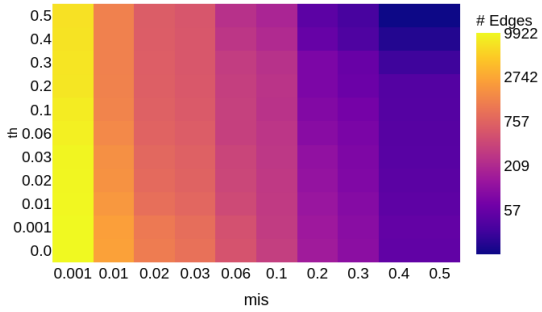

Figure 5.6.: This Figure depicts the **number of edges** concerning the hyperparameters. Learn-MSPN generated the SPN structures for th 0 on the NLTCS data set and column parameter set to **skips**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

Figure 5.5 shows the avg. LL, and Figure 5.6 depicts the number of edges concerning the hyperparameters for the pruned SPN structures with column parameter skips. For this configuration, the avg. LL of the initial SPN structures ranges from -6.08 to -8, represented by the *mis* values with *th* 0. Furthermore, the number of initial edges ranges from 16 to 9922, shown in Figure 5.6, and a *th* value of 0. Generally, increasing *mis* and *th* values lead to a decreasing avg. LL. The change in avg. LL is insignificant for initial SPN structures with a higher number of parameters. For example, the *mis* values in the range from 0.001 to 0.06 depict initial SPN structures with a higher number of edges in the range from 600 to 9900 edges. Furthermore, the change in likelihood is insignificant for these configurations, as shown in Figure 5.5 by a lighter transition in tone for increasing *th* values. The avg. LL increases rapidly for SPN structures with *mis* value 0.001 and increasing *th* values. This rapid increase means the SPA-SPN algorithm significantly improves the initial SPN structure's likelihood while reducing the number of edges. However, the avg. LL increases abruptly for SPN structures with *mis* values above 0.06, shown by the abrupt change to

a darker shade in Figure 5.5. The change in avg. LL does not change significantly for the *mis* values 0.01 and 0.02. Furthermore, a higher *mis* value leads to a significantly worse likelihood and increasing *th* values. This case suggests that SPA-SPN does not retain an SPN's performance while the reduction in parameter size is insignificant. Generally, SPA-SPN reduces the number of parameters shown in Figure 5.6 by the linear transition to a darker shade. Furthermore, SPA-SPN retains or improves the avg. LL for initial SPN structures with a larger parameter size. In contrast, SPA-SPN does not improve initial SPN structures' performance with a smaller parameter size.

5.5.2. MSNBC data set

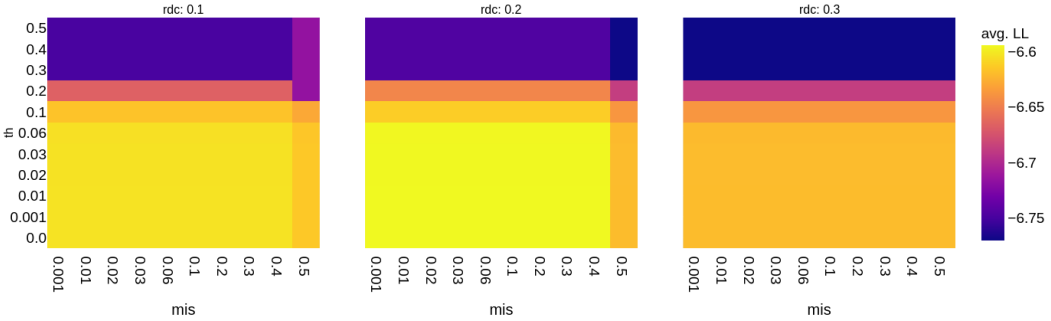

Figure 5.7.: This Figure depicts the **avg. LL** concerning the hyperparameters. Learn-MSPN generated the SPN structures for th 0 on the MSNBC data set and column parameter set to **rdc**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

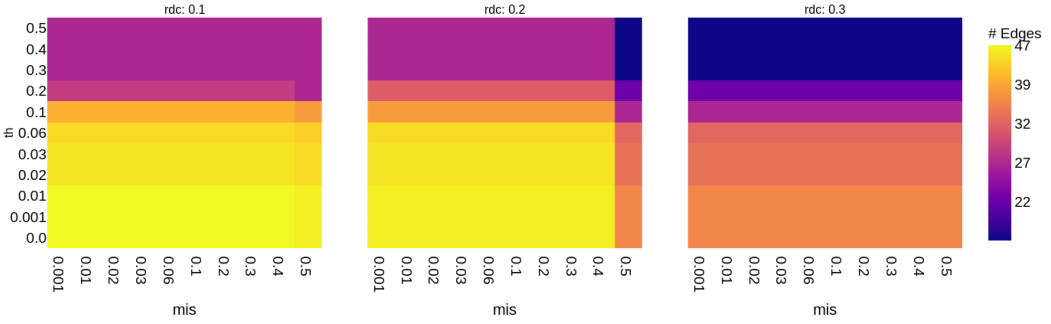

Figure 5.8.: This Figure depicts the **number of edges** concerning the hyperparameters. Learn-MSPN generated the SPN structures for th 0 on the MSNBC data set and column parameter set to **rdc**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

Figure 5.7 shows the average LL, and Figure 5.8 depicts the number of edges concerning the hyperparameters for the SPNs. For this configuration, the average LL does not change

for lower th values. However, the number of edges decreases for lower th values, e.g., th values in the range from 0.0 to 0.1. This decrease indicates that SPA-SPN retains the initial performance while reducing the number of parameters. The average LL decreases with a maximum value of -0.15, shown in Figure 5.7, indicated by a darker shade for an *mis* of 0.3. Furthermore, the number of edges decreases rapidly from 39 to 20 for this configuration. This trend indicates that an SPN maintains a similar performance by representing an SPN that has double its size in parameters. Generally, the Learn-MSPN algorithm generates similar initial structures for the given *mis* and *mis* values. The number of edges for these configurations is small compared to other data sets, e.g., the subsection mentioned earlier for the NLTCS data set. Hence, the Learn-MSPN already generates SPN structures representing independent leaf nodes. Furthermore, a smaller number of edges indicates that the decomposition step of the MSPN algorithm initiates more often. Additionally, the increase in *mis* values does not change the number of edges among different SPN structures. The insignificant change in the number of edges indicates that the decomposition step does not decrease the number of sum nodes when utilizing more instances during the conditioning step.

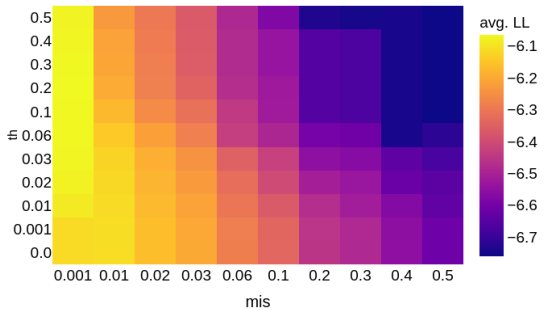

Figure 5.9.: This Figure depicts the **avg. LL** concerning the hyperparameters. Learn-MSPN generated the SPN structures for th 0 on the MSNBC data set and column parameter set to **skips**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

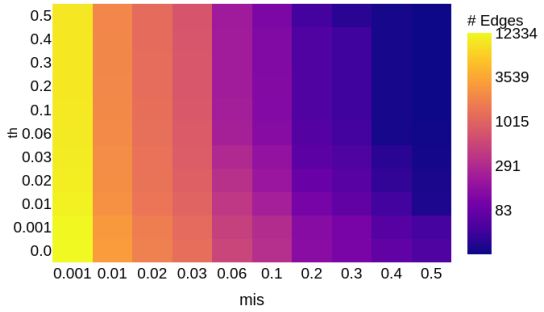

Figure 5.10.: This Figure depicts the **number of edges** concerning the hyperparameters. Learn-MSPN generated the SPN structures for th 0 on the MSNBC data set and column parameter set to **skips**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

Figure 5.9 shows the average LL, and Figure 5.10 depicts the number of edges concerning the hyperparameters for the SPNs. Generally, the number of edges decreases for increasing th values shown by the linear transition to a darker shade in Figure 5.10. Furthermore, the average LL decreases for initial SPN structures with *mis* values above 0.001. The decrease in average LL is insignificant for SPNs with an initially higher number of edges, e.g., SPNs with *mis* values range from 0.001 to 0.03. A lighter change in shade for increasing th values in Figure 5.9 shows the aforementioned slight decrease in likelihood. However, the likelihood decreases abruptly for initial SPNs with fewer edges. Furthermore, higher th values and *mis* values above 0.2 depict the decrease mentioned above in likelihood for these SPNs. Generally, SPA-SPN retains the performance of a given SPN structure for smaller th values. For example, in Figure 5.9, the th values below 0.2 show a slight change in likelihood for increasing th values. This slight decrease in performance means that SPA-SPN maintains the performance while reducing the set of parameters. However, the number of edges of the pruned SPNs is significantly higher than the SPNs generated with the rdc

column parameter. The number of edges for SPNs generated with rdc range from 50 to 22 edges. For the SPNs generated with the skips column parameter, the number of edges indicates more complex models. For example, the SPNs generated by a *mis* value up to 0.03 show SPNs with an average LL range from -6.1 to -6.2, which show a better performance than SPNs generated with column parameter rdc. However, the number of parameters for these SPNs ranges from 12000 to 1000, shown in Figure 5.10. This indifference in parameters indicates that Learn-MSPN with the column parameter rdc generates more compact SPNs with a smaller size in parameters and a good average LL performance. The SPA-SPN can reduce the number of parameters while maintaining a similar performance for the configuration above. One explanation for the statement above is that Learn-MSPN finds highly correlating variables during the decomposition step for the initial SPN structures. This comparison suggests that SPA-SPN does not replace the decomposition step of the Learn-MSPN algorithm by merging individual leaf nodes. However, SPA-SPN finds SPN structures with a smaller or similar size in parameters and significantly better likelihoods than Learn-MSPN for *mis* values between 0.3 and 0.5. For this configuration, the number of edges range from 140 to 23 and achieves a avg. LL in range from -6.7 to -6.5. This suggests that SPA-SPN creates SPN which are comparable or better relating Learn-MSPN with column parameter rdc.

5.5.3. KDD data set

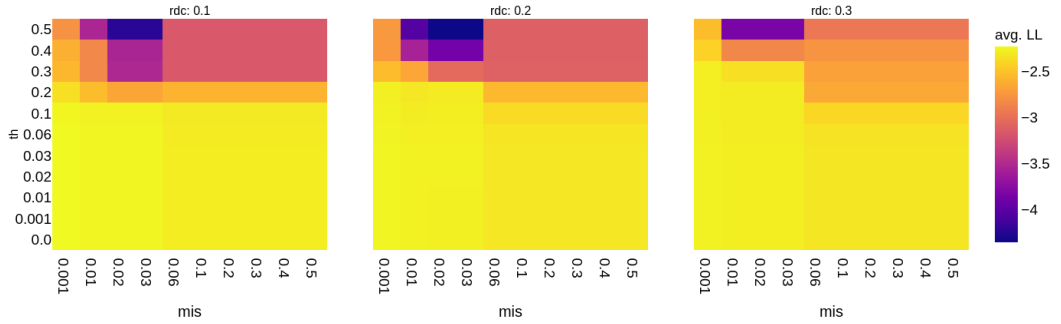

Figure 5.11.: This Figure depicts the **avg. LL** concerning the hyperparameters. Learn-MSPN generated the SPN structures for th 0 on the KDD data set and column parameter set to **rdc**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

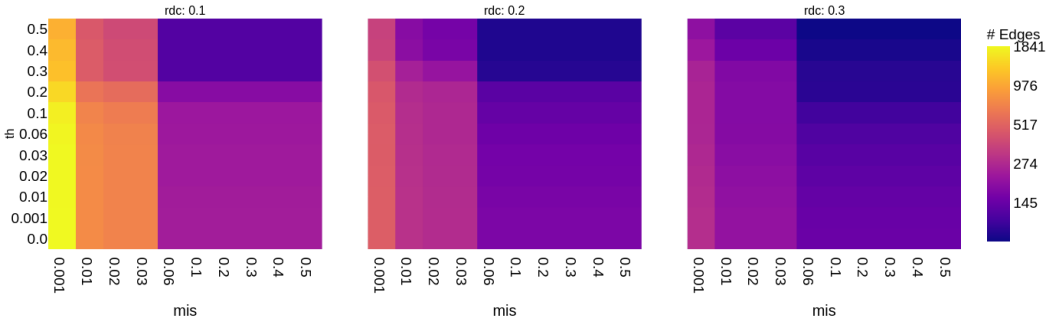

Figure 5.12.: This Figure depicts the **number of edges** concerning the hyperparameters. Learn-MSPN generated the SPN structures for th 0 on the KDD data set and column parameter set to **rdc**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

Figure 5.11 shows the avg. LL and Figure 5.12 depict the number of edges concerning the hyperparameters for the SPNs. Generally, the avg. LL change is insignificant for increasing *th* values. The avg. LL decreases at maximum by -1.5, reached at the highest *th* value of 0.5. This decrease indicates that merging more dissimilar leaf nodes results in SPNs with worse performance. However, the decrease in the number of edges is insignificant for increasing *th* values. This decrease is shown in Figure 5.12 since the transitions to a darker color are hardly present for increasing *th* values. Furthermore, this indicates that the initial SPN structures separate highly dependent variables. One explanation for the above statement is that a higher *th* value does not merge more dissimilar leaf nodes because a lower *th* value captured all possible merging candidates.

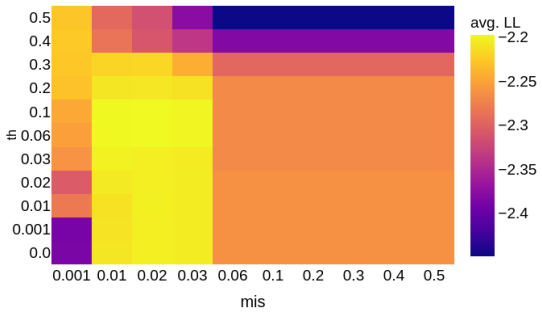

Figure 5.13.: This Figure depicts the **avg. LL** concerning the hyperparameters. Learn-MSPN generated the SPN structures for th 0 on the KDD data set and column parameter set to **skips**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

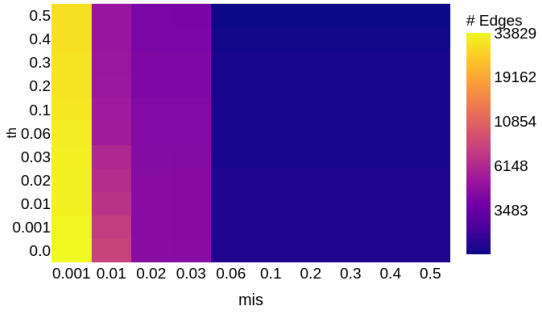

Figure 5.14.: This Figure depicts the **number of edges** concerning the hyperparameters. Learn-MSPN generated the SPN structures for th 0 on the KDD data set and column parameter set to **skips**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

Figure 5.13 shows the avg. LL and Figure 5.14 depict the number of edges concerning the hyperparameters for the SPNs. Generally, the decrease in avg. LL is insignificant for increasing *th* values. This decrease leads to a lighter change in the shade in Figure 5.14. Additionally, the SPNs depict a better performance in likelihood than SPNs generated with the column parameter set to rdc. However, the decrease in the number of edges is insignificant for increasing *th* values and all *mis* values except 0.01. For the *mis* value 0.01, the initial SPN structure is 1.5 times bigger than the pruned structure with a *th* value of 0.5. This decrease in parameters indicates that initial SPN structures generated with column parameter skips model highly dependent variables. Hence, an increasing *th* value does not change the number of parameters of the resulting SPNs. This suggests that SPA-SPN reduces the size in parameters for larger SPN structures while improving its likelihood. However, SPA-SPN does not find independencies for smaller SPN structures and fails to simplify these structures.

5.5.4. Plants data set

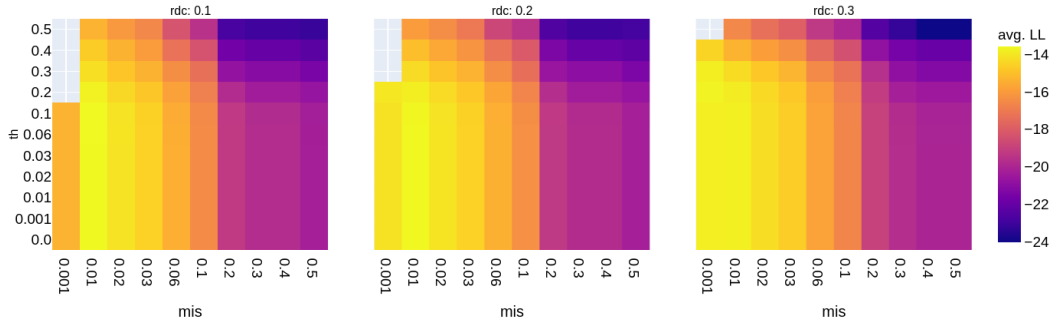

Figure 5.15.: This Figure depicts the **avg. LL** concerning the hyperparameters. Learn-MSPN generated the SPN structures for th 0 on the plants data set and column parameter set to **rdc**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

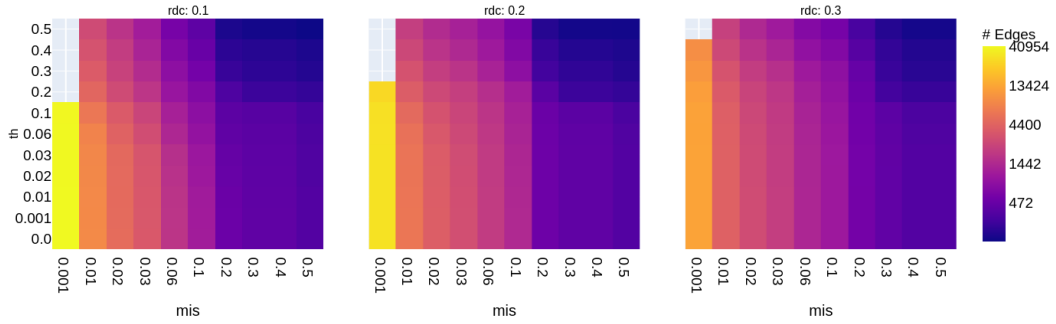

Figure 5.16.: This Figure depicts the **number of edges** concerning the hyperparameters. Learn-MSPN generated the SPN structures for th 0 on the plants data set and column parameter set to **rdc**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

Figure 5.11 shows the avg. LL, and Figure 5.12 depicts the number of edges with the hyperparameters for the SPNs. The transparent areas in the figures indicate missing data points. Generally, the avg. LL decreases significantly with increasing *th* values by a maximum of -10. However, the number of edges decreases for all configurations, which can be seen in Figure 5.18 by a lighter change in shade for increasing *th* values. The initial SPNs structures have up to 3 times more edges than the SPNs pruned with SPA-SPN, e.g., the *mis* value 0.03 for *rdc-threshold* 0.1 and *th* 0.5. Furthermore, the decrease in avg. LL is insignificant for smaller *th* values. This slight decrease indicates that SPA-SPN maintains the performance while reducing the set of parameters of a given SPNs.

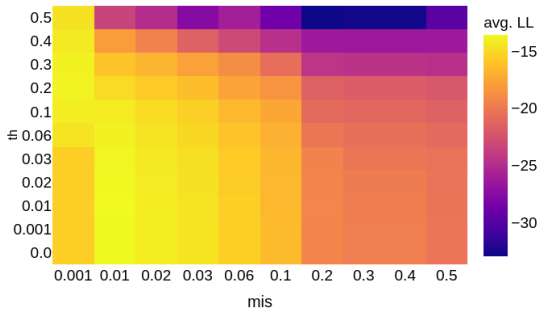

Figure 5.17.: This Figure depicts the **avg. LL** concerning the hyperparameters. Learn-MSPN generated the SPN structures for th 0 on the plants data set and column parameter set to **skips**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

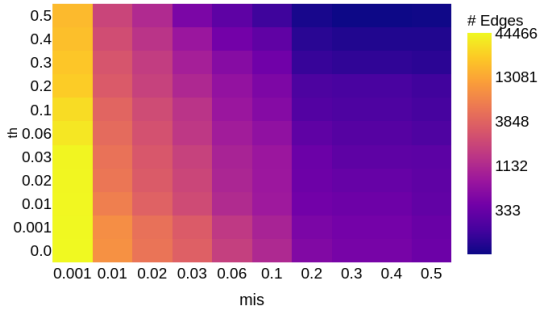

Figure 5.18.: This Figure depicts the **number of edges** concerning the hyperparameters. Learn-MSPN generated the SPN structures for th 0 on the plants data set and column parameter set to **skips**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

Figure 5.13 shows the avg. LL, and Figure 5.14 depicts the number of edges concerning the hyperparameters for the SPNs. Generally, the avg. LL decreases significantly for higher *th* values. Figure 5.17 shows the change in likelihood by the abrupt change in shade, e.g., *th* values above 0.2. In contrast, the avg. LL is similar to the initial SPNs structures for lower *th* values, e.g., *th* values below 0.3. Therefore, the SPA-SPN algorithm maintains the initially depicted likelihood when merging more similar leaf nodes. Furthermore, the number of edges decreases greatly for increasing *th* values. Figure 5.18 shows the rapid decrease in parameters indicated by a linear transition in the shade for increasing *th* values. The initial SPNs structures contain up to 3 times more edges than the pruned SPNs for *th* values up to 0.2. Additionally, the avg. LL for smaller *th* values do not differ from the initial structures generated with the column parameter set to rdc. However, the number of edges is significantly higher for the resulting SPNs with the column parameter set to skips. When post-pruning the initial structures with SPA-SPN, the avg. LL becomes significantly worse than the SPNs shown in Figure 5.16. Generally, SPA-SPN reduces the

number of parameters of a given initial SPNs structure while maintaining its performance. The Learn-MSPN with the column parameter rdc generated SPNs with highly correlating variables. Utilizing the SPA-SPN algorithm for these structures results in more compact SPNs that depict similar performances. Hence, SPA-SPN reduces the complexity for SPNs that depict a good distribution of the data set and larger structures. In contrast, SPA-SPN is not able to replace the decomposition methods used by this configuration. One argument for the above statement is that SPNs generated with the skips parameter result in SPNs with a significantly worse performance when utilizing SPA-SPN. Hence the given configurations show a trade-off between complexity and performance that deviates strongly with increasing *th* values.

5.5.5. Netflix data set

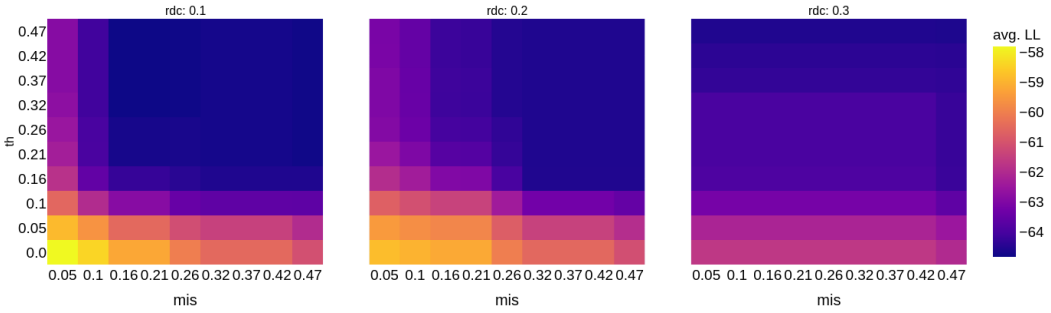

Figure 5.19.: This Figure depicts the **avg. LL** concerning the hyperparameters. Learn-MSPN generated the SPN structures for th 0 on the Netflix data set and column parameter set to **rdc**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

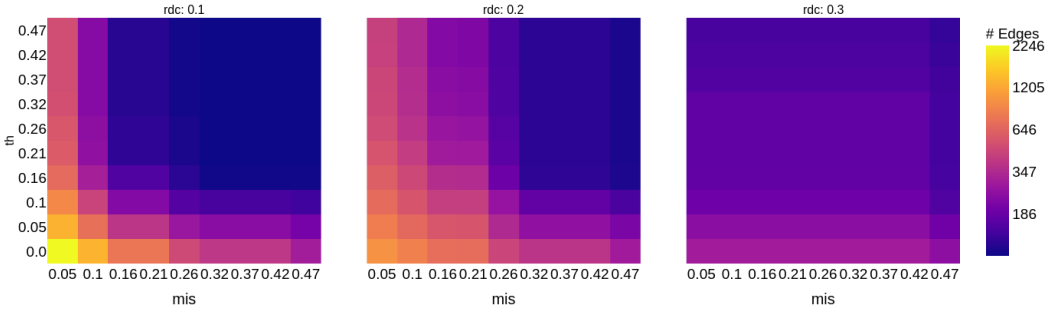

Figure 5.20.: This Figure depicts the **number of edges** concerning the hyperparameters. Learn-MSPN generated the SPN structures for th 0 on the Netflix data set and column parameter set to **rdc**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

Figure 5.19 shows the avg. LL, and Figure 5.20 depicts the number of edges concerning the hyperparameters for the SPNs. Generally, the avg. LL becomes significantly worse for increasing *th* values by a maximum of -6. The decrease in avg. LL does not change significantly after a *th* value of 0.1. Furthermore, the number of edges does not change significantly for *th* values above 0.1. As mentioned earlier, the conditions indicate that the SPNs satisfy a minimal structure, and further post-pruning leads to performance issues. Generating SPNs structures with smaller *mis* values below 0.05 lead to overly complicated models that did not fit into memory. However, the results show that SPA-SPN maintains the performance of SPNs with a larger number of variables while reducing their size in parameters when utilizing smaller *th* values. For example, the *th* values 0.05 and 0.1 show a lighter change in the shade in Figures 5.20 and 5.19.

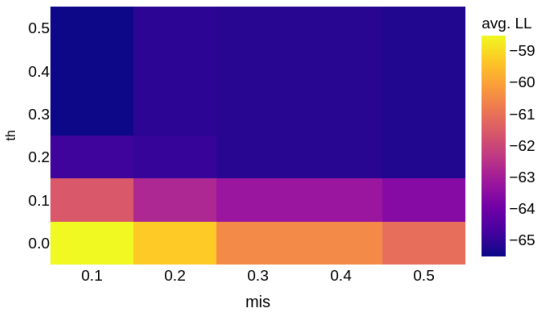

Figure 5.21.: This Figure depicts the **avg. LL** concerning the hyperparameters. Learn-MSPN generated the SPN structures for th 0 on the Netflix data set and column parameter set to **skips**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

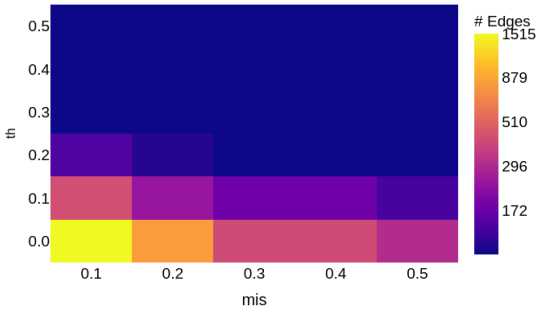

Figure 5.22.: This Figure depicts the **number of edges** concerning the hyperparameters. Learn-MSPN generated the SPN structures for th 0 on the Netflix data set and column parameter set to **skips**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

Due to the increased size in the Netflix data set variables, the generated SPNs structures lead to overly complicated models. Furthermore, most of the models did not fit into

memory. Therefore, the following paragraph discusses SPNs generated utilizing a sub-set of hyperparameters.

Figure 5.21 shows the avg. LL, and Figure 5.22 depicts the number of edges to the hyperparameters for the SPNs. The avg. LL becomes significantly worse for increasing *th* values. Additionally, the number of parameters does not decrease significantly for higher *th* values. The change in avg. LL deviates strongly for a small increase in *th* values, e.g., by 0.1. Furthermore, the respective figures show an abrupt change to a darker shade. Generally, SPA-SPN reduces the size in parameters immensly and creates up to 14 times smaller SPN structures. However, reducing the number of edges results in substantial performance losses. This loss can have two reasons: the small initial parameter size or a higher number of variables of the given data set. The small size of the initial parameters indicates that the given SPNs represent a minimal structure. This recurring pattern applies to all data sets in the previous subsections. SPNs generated with higher *mis* values lead to initial SPNs with a small number of parameters. For these configurations, the SPA-SPN algorithm did not reduce the number of parameters significantly and did not maintain the initial structures' performance. Furthermore, a higher number of variables may lead to more complex distributions depicted by an SPNs. When utilizing a higher number of variables, the learned distributions depicted by individual leaf nodes can be dependent on other variables, e.g., discussed in (Rahman and Gogate, 2016). Hence, separating the variables by utilizing SPA-SPN may destroy indirect dependencies among different variables. Furthermore, SPA-SPN only considers individual pairs of variables and does not involve global information, e.g., other variables or nodes.

6. Conclusion and Future Work

We introduced a bottom-up, simple post-pruning algorithm SPA-SPN for Sum-Product Networks. In contrast to other approaches, this work presents the first post pruning method for SPNs that do not rely on a validation set. The structural information of an initial SPN is used to compare and combine similar leaf nodes. SPA-SPN uses a distance metric for comparison and a threshold value to decide whether nodes are similar enough to be merged. The SPA-SPN algorithm pruned SPN structures generated with the Learn-MSPN algorithm. Moreover, different hyperparameters for the conditioning and decomposition steps, such as the randomized dependency coefficient and the minimum instances to slice. SPA-SPN was applied to initial SPN structures learned from five real-world data sets, including binary variables. Many insights can be gained from evaluating the SPA-SPN. For instance, the method leads to a better understanding of the relationship between thresholds, hyperparameters for the initial structures, and evaluation measures. Generally, an increasing threshold value leads to SPNs with a higher Log-Likelihood and reduced size. It was found that initial SPNs with a larger size in parameters improved or maintained the Log-Likelihood when increasingly merging more dissimilar leaf nodes. This demonstrates that the SPA-SPN method reduces the number of parameters of a given SPN while maintaining its performance. Utilizing a post-pruning method that does not rely on a validation set has the advantage of a simpler and faster framework because extensive data set evaluations are left out.

However, SPA-SPN also entails some disadvantages. Data sets with a large number of variables yield SPNs with an increased number of leaf nodes. Therefore the pairwise comparison of similar leaf nodes requires evaluating many combinations, leading to a higher computation time when pruning an SPN for all variables individually. As a result, future work could include a framework to select a sequence of merge candidates for different variables that maximize the performance.

Another point is that SPN structures modeling data with larger amounts of random variables tend to have a decrease in performance in log-likelihood when processed by SPA-SPN. There are two ideas to address the problem of poor performance when it comes to post-pruning SPNs. The first is motivated by the reduced error pruning method for decision trees. Merging further leaf nodes could be terminated when the Log-Likelihood for a validation set decreases.

There is another idea, which incorporates a similarity measure to include leaf nodes with different scopes within a sub-SPN. The benefit of including different scopes allows SPNs to model indirect interactions between variables. For example, the ID-SPN Rooshenas and Lowd (2014) structure learner achieved state-of-the-art results by considering indirect variable interactions. According to Nguyen and Vreeken (2015), the JSD extends to the multivariate case, which allows the comparison of leaf nodes with different scopes.

Finally, the SPA-SPN algorithm was evaluated on data sets with binary variables. Therefore, the merging strategy can be extended for categorical and continuous distributions.

Bibliography

Augasta, M. Gethsiyal and T. Kathirvalavakumar (2011). “A Novel Pruning Algorithm for Optimizing Feedforward Neural Network of Classification Problems”. In: *Neural Process. Lett.* 34.3, pp. 241–258. DOI: 10.1007/s11063-011-9196-7. URL: https://doi.org/10.1007/s11063-011-9196-7.

Bartoldson, Brian et al. (2020). “The Generalization-Stability Tradeoff In Neural Network Pruning”. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle et al. URL: https://proceedings.neurips.cc/paper/2020/hash/ef2ee09ea9551de88bc11fd7eeea93b0-Abstract.html.

Bekker, Jessa et al. (2015). “Tractable Learning for Complex Probability Queries”. In: *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*. Ed. by Corinna Cortes et al., pp. 2242–2250. URL: https://proceedings.neurips.cc/paper/2015/hash/bb7946e7d85c81a9e69fee1cea4a087c-Abstract.html.

Breiman, Leo (2001). “Random forests”. In: *Machine learning* 45.1, pp. 5–32.

Breiman, Leo et al. (1984). *Classification and Regression Trees*. Wadsworth. ISBN: 0-534-98053-8.

Cheng, Wei-Chen et al. (2014). “Language modeling with sum-product networks”. In: *Interspeech 2014, 15th Annual Conference of the International Speech Communication Association, Singapore, September 14-18, 2014*. Ed. by Haizhou Li et al. ISCA, pp. 2098–2102. URL: http://www.isca-speech.org/archive/interspeech%5C_2014/i14%5C_2098.html.

Chow, C. K. and C. N. Liu (1968). “Approximating discrete probability distributions with dependence trees”. In: *IEEE Trans. Inf. Theory* 14.3, pp. 462–467. DOI: 10.1109/TIT.1968.1054142. URL: https://doi.org/10.1109/TIT.1968.1054142.

Clemen, R. and R. Winkler (1999). “Combining Probability Distributions From Experts in Risk Analysis”. In: *Risk Analysis* 19, pp. 187–203.

Costa, Eduardo P., Sicco Verwer, and Hendrik Blockeel (2013). “Estimating Prediction Certainty in Decision Trees”. In: *Advances in Intelligent Data Analysis XII - 12th International Symposium, IDA 2013, London, UK, October 17-19, 2013. Proceedings*. Ed. by Allan Tucker et al. Vol. 8207. Lecture Notes in Computer Science. Springer, pp. 138–149. DOI: 10.1007/978-3-642-41398-8_13. URL: https://doi.org/10.1007/978-3-642-41398-8%5C_13.

Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). “Maximum likelihood from incomplete data via the EM algorithm”. In: *Journal of the Royal Statistical Society: Series B* 39, pp. 1–38. URL: http://web.mit.edu/6.435/www/Dempster77.pdf.

Dennis, Aaron W. and Dan Ventura (2012). “Learning the Architecture of Sum-Product Networks Using Clustering on Variables”. In: *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*. Ed. by

Peter L. Bartlett et al., pp. 2042–2050. URL: https://proceedings.neurips.cc/paper/2012/hash/f33ba15effa5c10e873bf3842afb46a6-Abstract.html.

Elomaa, Tapio and Matti Kääriäinen (2001). "An Analysis of Reduced Error Pruning". In: *J. Artif. Intell. Res.* 15, pp. 163–187. DOI: 10.1613/jair.816. URL: https://doi.org/10.1613/jair.816.

Frank, Eibe (2000). "Pruning Decision Trees and Lists". In:

Gens, Robert and Pedro M. Domingos (2013). "Learning the Structure of Sum-Product Networks". In: *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*. Vol. 28. JMLR Workshop and Conference Proceedings. JMLR.org, pp. 873–880. URL: http://proceedings.mlr.press/v28/gens13.html.

Geurts, Karolien et al. (2003). "Profiling High Frequency Accident Locations Using Association Rules". In: *'Proceedings of the 82nd Annual Transportation Research Board, Washington DC. (USA), January 12-16*, 18pp.

Gogate, Vibhav, William Austin Webb, and Pedro M. Domingos (2010). "Learning Efficient Markov Networks". In: *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada*. Ed. by John D. Lafferty et al. Curran Associates, Inc., pp. 748–756. URL: https://proceedings.neurips.cc/paper/2010/hash/e5e63da79fcd2bebbd7cb8bf1c1d0274-Abstract.html.

Haaren, Jan Van and Jesse Davis (2012). "Markov Network Structure Learning: A Randomized Feature Generation Approach". In: *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*. Ed. by Jörg Hoffmann and Bart Selman. AAAI Press. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/5107.

Joyce, James M. (2011). "Kullback-Leibler Divergence". In: *International Encyclopedia of Statistical Science*. Ed. by Miodrag Lovric. Springer, pp. 720–722. DOI: 10.1007/978-3-642-04898-2_327. URL: https://doi.org/10.1007/978-3-642-04898-2%5C_327.

Knuth, Donald E. and Ronald W. Moore (1975). "An Analysis of Alpha-Beta Pruning". In: *Artif. Intell.* 6.4, pp. 293–326. DOI: 10.1016/0004-3702(75)90019-3. URL: https://doi.org/10.1016/0004-3702(75)90019-3.

Ko, Ching-Yun et al. (2020). "Deep Model Compression and Inference Speedup of Sum-Product Networks on Tensor Trains". In: *IEEE Trans. Neural Networks Learn. Syst.* 31.7, pp. 2665–2671. DOI: 10.1109/TNNLS.2019.2928379. URL: https://doi.org/10.1109/TNNLS.2019.2928379.

Lin, Jianhua (1991). "Divergence measures based on the Shannon entropy". In: *IEEE Trans. Inf. Theory* 37.1, pp. 145–151. DOI: 10.1109/18.61115. URL: https://doi.org/10.1109/18.61115.

Lowd, Daniel and Jesse Davis (2010). "Learning Markov Network Structure with Decision Trees". In: *ICDM 2010, The 10th IEEE International Conference on Data Mining, Sydney, Australia, 14-17 December 2010*. Ed. by Geoffrey I. Webb et al. IEEE Computer Society, pp. 334–343. DOI: 10.1109/ICDM.2010.128. URL: https://doi.org/10.1109/ICDM.2010.128.

Lowd, Daniel and Amirmohammad Rooshenas (2013). "Learning Markov Networks With Arithmetic Circuits". In: *Proceedings of the Sixteenth International Conference on Artificial*

Intelligence and Statistics, AISTATS 2013, Scottsdale, AZ, USA, April 29 - May 1, 2013. Vol. 31. JMLR Workshop and Conference Proceedings. JMLR.org, pp. 406–414. URL: http://proceedings.mlr.press/v31/lowd13a.html.

Mansour, Y. (1997). "Pessimistic Decision Tree Pruning Based on Tree Size". In: *ICML 1997*.

Molina, Alejandro, Antonio Vergari, Nicola Di Mauro, et al. (2018). "Mixed Sum-Product Networks: A Deep Architecture for Hybrid Domains". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*. Ed. by Sheila A. McIlraith and Kilian Q. Weinberger. AAAI Press, pp. 3828–3835. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16865.

Molina, Alejandro, Antonio Vergari, Karl Stelzner, et al. (2019). "SPFlow: An Easy and Extensible Library for Deep Probabilistic Learning using Sum-Product Networks". In: *CoRR* abs/1901.03704. arXiv: 1901.03704. URL: http://arxiv.org/abs/1901.03704.

Murphy, Kevin P. (2012). *Machine learning - a probabilistic perspective*. Adaptive computation and machine learning series. MIT Press. ISBN: 0262018020.

Newman, C.L. Blake D.J. and C.J. Merz (1998). *UCI Repository of machine learning databases*. URL: http://www.ics.uci.edu/$%5Csim$mlearn/MLRepository.html.

Nguyen, Hoang Vu and Jilles Vreeken (2015). "Non-parametric Jensen-Shannon Divergence". In: *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part II*. Ed. by Annalisa Appice et al. Vol. 9285. Lecture Notes in Computer Science. Springer, pp. 173–189. DOI: 10.1007/978-3-319-23525-7_11. URL: https://doi.org/10.1007/978-3-319-23525-7%5C_11.

Nicolson, Aaron and Kuldip K. Paliwal (2020). "Sum-Product Networks for Robust Automatic Speaker Identification". In: *Interspeech 2020, 21st Annual Conference of the International Speech Communication Association, Virtual Event, Shanghai, China, 25-29 October 2020*. Ed. by Helen Meng, Bo Xu, and Thomas Fang Zheng. ISCA, pp. 1516–1520. DOI: 10.21437/Interspeech.2020-1501. URL: https://doi.org/10.21437/Interspeech.2020-1501.

París, Iago, Raquel Sánchez-Cauce, and Francisco Javier Díez (2020). "Sum-product networks: A survey". In: *CoRR* abs/2004.01167. arXiv: 2004.01167. URL: https://arxiv.org/abs/2004.01167.

Patel, N. and S. Upadhyay (2012). "Study of Various Decision Tree Pruning Methods with their Empirical Comparison in WEKA". In: *International Journal of Computer Applications* 60, pp. 20–25.

Pedersen, M. W. and D. G. Stork (1996). "Pruning Boltzmann networks and hidden Markov models". In: *Conference Record of The Thirtieth Asilomar Conference on Signals, Systems and Computers* 1, 258–261 vol.1.

Peharz, Robert, Bernhard C. Geiger, and Franz Pernkopf (2013). "Greedy Part-Wise Learning of Sum-Product Networks". In: *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part II*. Ed. by Hendrik Blockeel et al. Vol. 8189. Lecture Notes in Computer Science. Springer, pp. 612–627. DOI: 10.1007/978-3-642-40991-2_39. URL: https://doi.org/10.1007/978-3-642-40991-2%5C_39.

Poon, Hoifung and Pedro M. Domingos (2011). “Sum-product networks: A new deep architecture”. In: *IEEE International Conference on Computer Vision Workshops, ICCV 2011 Workshops, Barcelona, Spain, November 6-13, 2011*. IEEE Computer Society, pp. 689–690. DOI: 10.1109/ICCVW.2011.6130310. URL: https://doi.org/10.1109/ICCVW.2011.6130310.

Queyranne, Maurice (1998). “Minimizing symmetric submodular functions”. In: *Math. Program.* 82, pp. 3–12. DOI: 10.1007/BF01585863. URL: https://doi.org/10.1007/BF01585863.

Quinlan, J. Ross (1987). “Simplifying Decision Trees”. In: *Int. J. Man Mach. Stud.* 27.3, pp. 221–234. DOI: 10.1016/S0020-7373(87)80053-6. URL: https://doi.org/10.1016/S0020-7373(87)80053-6.

– (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann. ISBN: 1-55860-238-0.

Rahman, Tahrima and Vibhav Gogate (2016). “Merging Strategies for Sum-Product Networks: From Trees to Graphs”. In: *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence, UAI 2016, June 25-29, 2016, New York City, NY, USA*. Ed. by Alexander T. Ihler and Dominik Janzing. AUAI Press. URL: http://auai.org/uai2016/proceedings/papers/71.pdf.

Rooshenas, Amirmohammad and Daniel Lowd (2013). “Learning Tractable Graphical Models Using Mixture of Arithmetic Circuits”. In: *Late-Breaking Developments in the Field of Artificial Intelligence, Bellevue, Washington, USA, July 14-18, 2013*. Vol. WS-13-17. AAAI Workshops. AAAI. URL: http://www.aaai.org/ocs/index.php/WS/AAAIW13/paper/view/7183.

– (2014). “Learning Sum-Product Networks with Direct and Indirect Variable Interactions”. In: *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*. Vol. 32. JMLR Workshop and Conference Proceedings. JMLR.org, pp. 710–718. URL: http://proceedings.mlr.press/v32/rooshenas14.html.

Rudin, C. (2018). “Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead”. In: *Nature Machine Intelligence* 1, pp. 206–215.

Sguerra, Bruno Massoni and Fábio Gagliardi Cozman (2016). “Image Classification Using Sum-Product Networks for Autonomous Flight of Micro Aerial Vehicles”. In: *5th Brazilian Conference on Intelligent Systems, BRACIS 2016, Recife, Brazil, October 9-12, 2016*. IEEE Computer Society, pp. 139–144. DOI: 10.1109/BRACIS.2016.035. URL: https://doi.org/10.1109/BRACIS.2016.035.

Vergari, Antonio, Nicola Di Mauro, and Floriana Esposito (2015). “Simplifying, Regularizing and Strengthening Sum-Product Network Structure Learning”. In: *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part II*. Ed. by Annalisa Appice et al. Vol. 9285. Lecture Notes in Computer Science. Springer, pp. 343–358. DOI: 10.1007/978-3-319-23525-7_21. URL: https://doi.org/10.1007/978-3-319-23525-7%5C_21.

Zhou, Zhi-Hua (2015). “Ensemble Learning”. In: *Encyclopedia of Biometrics, Second Edition*. Ed. by Stan Z. Li and Anil K. Jain. Springer US, pp. 411–416. DOI: 10.1007/978-1-4899-7488-4_293. URL: https://doi.org/10.1007/978-1-4899-7488-4_293.

A. Appendix

A.1. Description of Figures

Figure 5.1 shows the quantity and performance of the individual SPNs trained with the Learn-MSPN algorithm and the configuration of the column parameter as rdc. The x-axis is a tuple of rdc and *mis* hyperparameters used to learn the structure of the respective SPNs. Furthermore, the rdc and *mis* tuple show in ascending order from left to right on the x-axis. The figure shows SPNs generated with the rdc values 0.1, 0.2, and 0.3 and *mis* values in the range of 0.0001 to 0.5. The y-axis shows two different domains, the avg. LL and the statistics in the SPNs quantity labeled as the "# of parameters". Additionally, the logarithm with base 10 scales the y-axis by the number of parameters. This visualization ensures better readability, which allows comparing different figures. The bar diagrams and the line plot with the feature "# nodes" refer to the domain with "# of parameters", the avg. LL is shown with the dark red line in the graph. The legend shows the features and their respective color, e.g., "# product nodes" shows the number of product nodes with a green line.

A.2. Initial structures with rdc decomposition

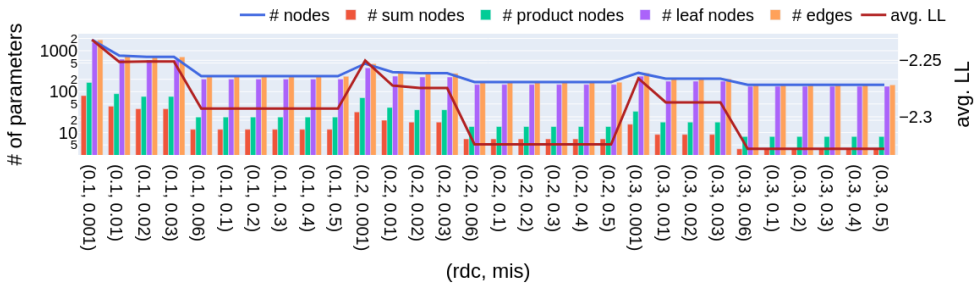

Figure A.1.: This figure represents the initial SPN structures learned on the KDD data set by the Learn-MSPN algorithm with the column parameter set to **rdc**.

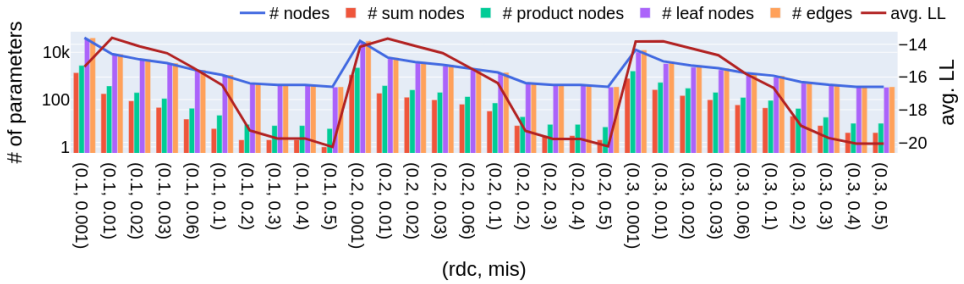

Figure A.2.: This figure represents the initial SPN structures learned on the plants data set by the Learn-MSPN algorithm with the column parameter set to **rdc**.

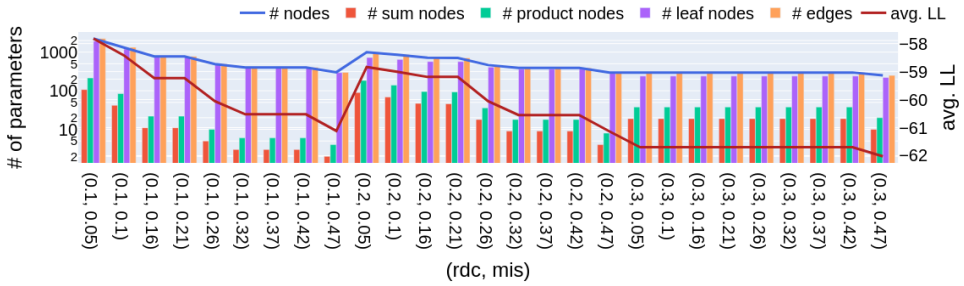

Figure A.3.: This figure represents the initial SPN structures learned on the Netflix data set by the Learn-MSPN algorithm with the column parameter set to **rdc**.

A.3. Pruned structures evaluated on CMLL

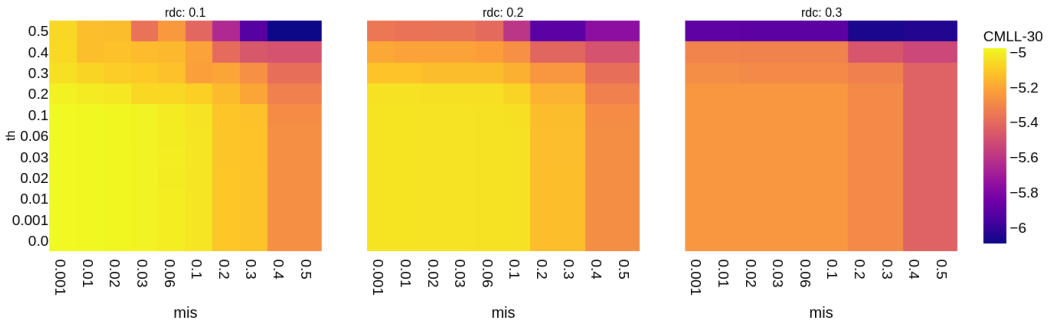

Figure A.4.: This Figure depicts the **CMLL-30** concerning the hyperparameters. Learn-MSPN generated the SPN structures for th 0 on the **NLTCS** data set and column parameter set to **rdc**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

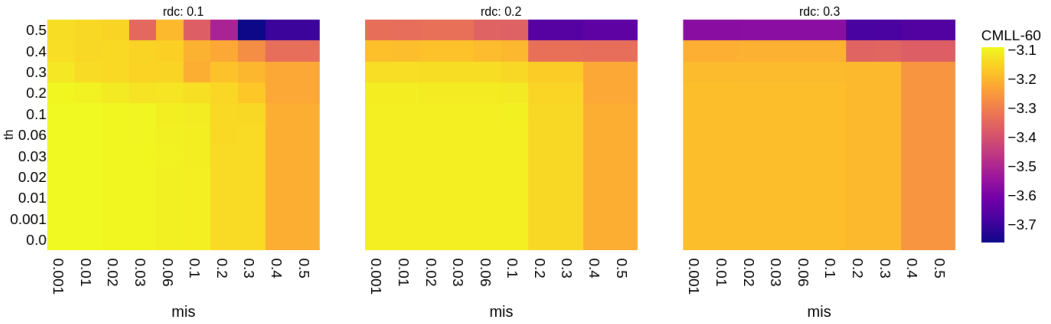

Figure A.5.: This Figure depicts the **CMLL-60** concerning the hyperparameters. LearnMSPN generated the SPN structures for th 0 on the **NLTCS** data set and column parameter set to **rdc**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

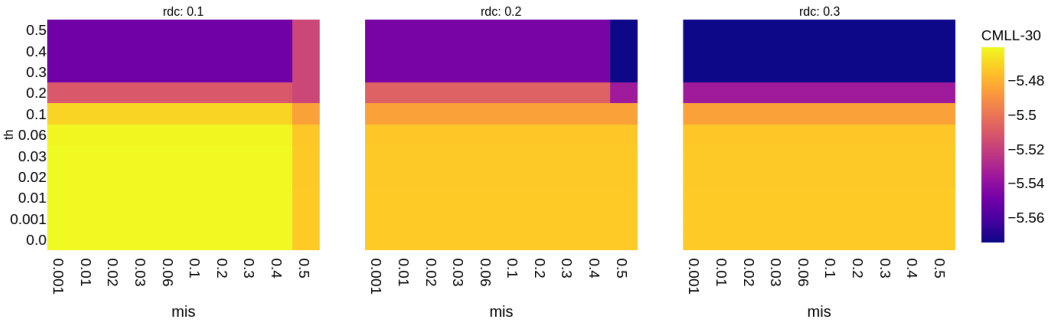

Figure A.6.: This Figure depicts the **CMLL-30** concerning the hyperparameters. LearnMSPN generated the SPN structures for th 0 on the **MSNBC** data set and column parameter set to **rdc**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

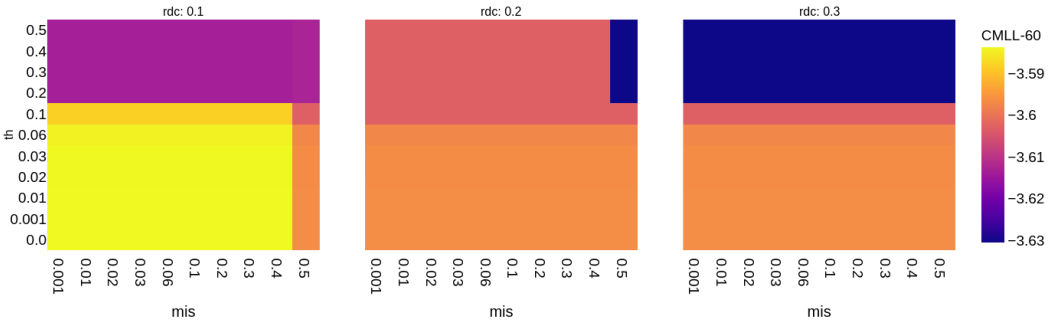

Figure A.7.: This Figure depicts the **CMLL-60** concerning the hyperparameters. LearnMSPN generated the SPN structures for th 0 on the **MSNBC** data set and column parameter set to **rdc**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

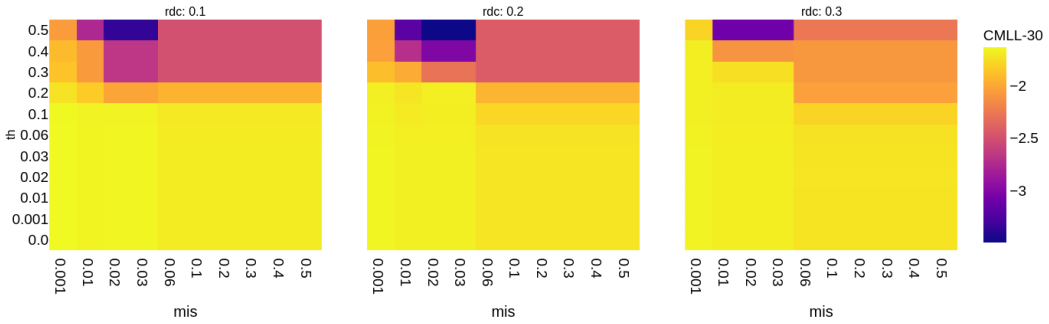

Figure A.8.: This Figure depicts the **CMLL-30** concerning the hyperparameters. LearnMSPN generated the SPN structures for th 0 on the **KDD** data set and column parameter set to **rdc**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

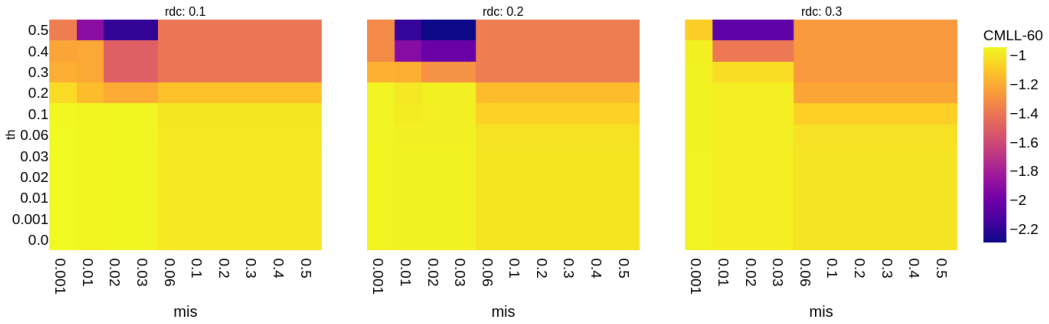

Figure A.9.: This Figure depicts the **CMLL-60** concerning the hyperparameters. LearnMSPN generated the SPN structures for th 0 on the **KDD** data set and column parameter set to **rdc**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

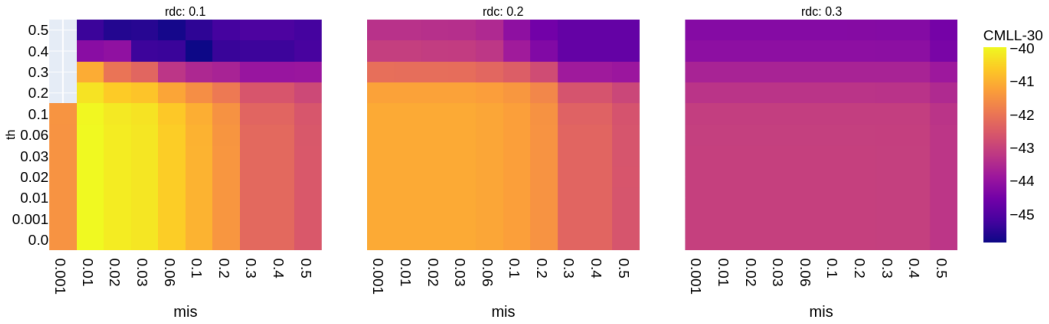

Figure A.10.: This Figure depicts the **CMLL-30** concerning the hyperparameters. LearnMSPN generated the SPN structures for th 0 on the **Netflix** data set and column parameter set to **rdc**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

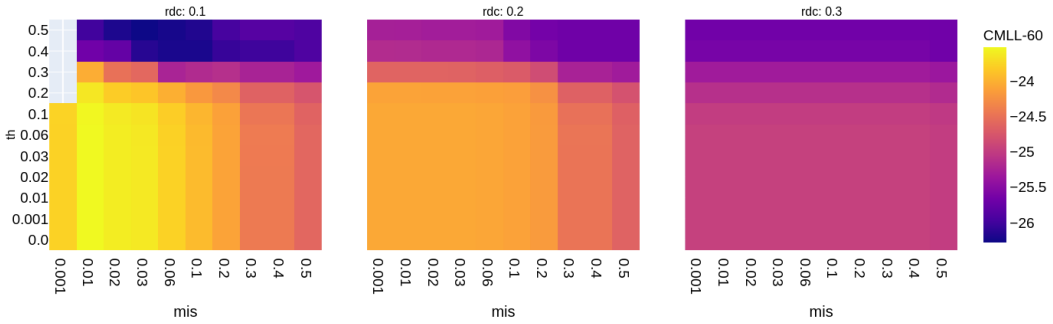

Figure A.11.: This Figure depicts the **CMLL-60** concerning the hyperparameters. LearnMSPN generated the SPN structures for th 0 on the **Netflix** data set and column parameter set to **rdc**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

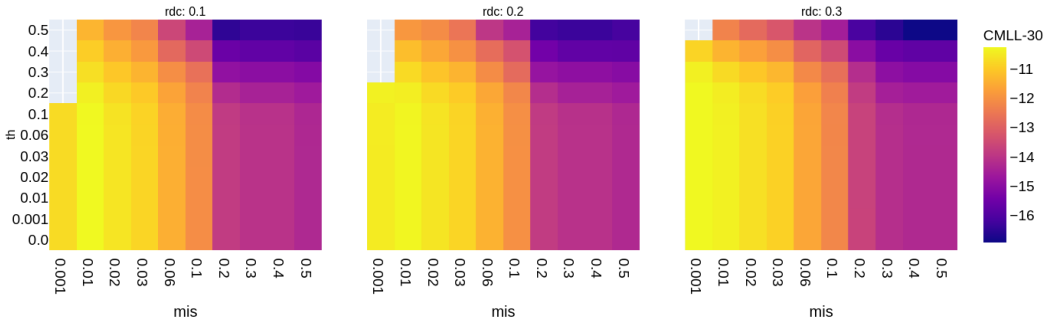

Figure A.12.: This Figure depicts the **CMLL-30** concerning the hyperparameters. LearnMSPN generated the SPN structures for th 0 on the **plants** data set and column parameter set to **rdc**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.

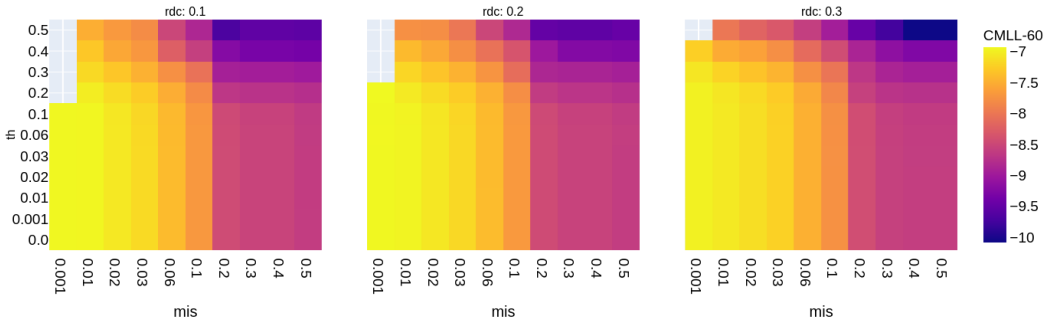

Figure A.13.: This Figure depicts the **CMLL-60** concerning the hyperparameters. LearnMSPN generated the SPN structures for th 0 on the **plants** data set and column parameter set to **rdc**. The SPNs with th bigger than 0 relate to SPNs resulting from SPA-SPN.