A hierarchy of sum-product networks using robustness

A Hierarchy of Sum-Product Networks using Robustness

Diarmaid Conaty, Jesús Martínez del Rincon

Centre for Data Science and Scalable Computing, Queen's University Belfast, U.K.

Cassio P. de Campos

Dept. of Information and Computing Sciences, Utrecht University, The Netherlands

Abstract

Sum-product networks are a popular family of probabilistic graphical models that have been shown to achieve state-of-the-art performance in several tasks. When learning sum-product networks from scarce data, the obtained model may be prone to robustness issues, and where small variations of parameters could lead to different conclusions. We discuss the characteristics of sum-product networks as classifiers and study the robustness of them with respect to their parameters. Using a robustness measure to identify (possibly) unreliable decisions, we build a hierarchical approach where the classification task in testing time is deferred to another model if the outcome is deemed unreliable. We apply this approach on benchmark classification tasks across 47 datasets. Experiments show that the robustness measure can be a meaningful manner to build dynamic ensemble of classifiers and that our Hierarchical Sum-Product Network guarantees an improvement in accuracy.

Keywords: Sum-product networks; sensitivity analysis; robustness; classification..

1. Introduction

Probabilistic graphical models allow for the compact specification of uncertain knowledge through a graphical language which facilitates elicitation, improves interpretability, and achieves good inferential performance [8, 17]. Sum-Product Networks (SPNs) are a class of probabilistic graphical models that allow for the explicit representation of context-specific independence. They are popular due to their ability to represent complex distributions while retaining efficient marginal inference [22, 27, 29]. The internal nodes of an SPN perform (weighted) sums and multiplications, while leaves represent variable assignments. The sum nodes can be interpreted as latent variables inducing mixtures of distributions, while the product nodes can be interpreted as encoding probabilistic independences [14, 25]. We discuss how SPNs naturally encompass the Naive Bayes classifier, which partially justifies their good performance in classification tasks (given that Naive Bayes, albeit quite simplistic, is one of the top performing classifiers in the literature for a wide range of problems). We also discuss on other relations of SPNs and Bayesian network classifiers.

On par with other probabilistic graphical models, SPNs learned from data may generalise poorly for configurations of the variables that do not appear often, and produce unreliable and overconfident conclusions. One way in which these problems have been dealt with in the case of other machine learning algorithms has been through the use of ensemble learning techniques. Such an approach takes the form of using several base learners and combining these weak models into a composite strong learner which makes up for their individual weaknesses. This amounts to a machine learning interpretation of the principle of the wisdom of crowds [31]. Much of the modern work in this area owes itself to initial achievements of Schapire [32] and Hansen and Salamon [15]. Examples of such methods include Boosting [12], Bagging [2] and Random Forests [3]. In each of these the basic premise of combining a number of learners together to create a stronger model is shared but the techniques used can differ in a number of ways. While boosting and bagging both induce new weak learners by manipulating the data used as input to learning, boosting does so in a manner whereby the output of each weak learner will have an effect on the next weak learner created. Meanwhile, weak learners in bagging are created independent from their respective outputs. Random Forests also achieve its weak learners independently in a manner similar to bagging to create its decision trees but with added layers of selecting random subsets of features for its weak learners.

Such techniques have been used with a number of machine learning algorithms but never with SPNs in conjunction with a measurement of each learner's robustness in order to inform the ensemble learning process. The main contribution of this paper is to introduce an ensemble approach based on a hierarchy of multiple SPNs classifiers which defers the decision to the next classifier when the current prediction is not reliable. As differentiation factor, this deferral is performed in real time using testing robustness information rather than training information. Therefore, this new *Hierarchical*

Sum-Product Network approach builds an ensemble model that is not fully pre-learned with training data. Through the use of a robustness measurement in the procedure of combining our independently created weak learners as part of the ensemble learning process we seek to overcome some of the sensitivity to overfitting of pre-learned ensembles and provide an accurate and reliable technique for classification.

. Such robustness measure can be computed both at training and testing time and is capable of discriminating reliable versus unreliable predictions, but it requires the tuning of its threshold parameter (used to split between reliable and unreliable predictions). For that, we propose to learn the threshold from data, and we empirically show that this framework increases classification accuracy with small extra effort and performs no worse than the strongest component SPN, that is, we theoretically and empirically verify that the accuracy of the *Hierarchical Sum-Product Network* increases with respect to its strongest component. The hierarchy of SPNs decides, based on their robustness measure, whether to defer the responsibility of prediction to another SPN in testing time. The measure of robustness used to build the hierarchy and learn the automatic threshold is obtained similarly to recent work on Credal SPNs [21]. We also explore the effects of parameter tuning such as the impact of various criteria for learning a robustness threshold as well as the behaviour when the number of layers used in the ensemble model are increased.

The paper is divided as follows. Section 2 describes the notation and defines the SPNs that are used in this work. It also discusses on a learning approach and how it relates to some Bayesian network classifiers. Section 3 describes our approach to defer the decision to another SPN when the prediction is not reliable enough. Section 4 presents our experimental setup and the obtained results, and finally Section 5 concludes the paper.

2. Sum-product networks

Random variables are denoted by X with a subscript (e.g., X_1, X_i). A collection of random variables indexed by a set $\mathcal{V}$ is denoted by $X_{\mathcal{V}} = \{X_i : i \in \mathcal{V}\}$. A configuration of a collection of random variables is denoted as $X_{\mathcal{V}} = x_{\mathcal{V}}$. We assume that every random variable X_i is categorical and take values in $\{1, \ldots, |X_i|\}$. *Indicator variables* $\{\lambda_{i,j} : j = 1, \ldots, |X_i|\}$ are used to indicate an outcome of the variable X_i. For any configuration $X_{\mathcal{V}} = x_{\mathcal{V}}$ we write λ^{x_v} to denote the configuration of indicator variables such that

$\lambda_{i,x_i} = 1$ and $\lambda_{i,j} = 0$ for all $j \neq x_i$. When the configuration mentions only a subset of all the variables, say $X_{\mathcal{E}} = e$ for $\mathcal{E} \subset \mathcal{V}$, we write λ^e to denote the configuration of indicator variables that assigns $\lambda_{i,j} = 0$ if $i \in \mathcal{E}$ and $e_i \neq j$, and $\lambda_{i,j} = 1$ otherwise. That is, λ^e is the configuration of indicator variables that is *consistent* with the configuration e and assigns 1 to indicator variables associated to unrealised random variables.

An SPN is a concise graphical representation of the multilinear polynomial specifying a (discrete) probability measure [7]. In more detail, an SPN is a weighted, rooted and acyclic directed graph where internal nodes are labelled as either sum or product operations and leaves are associated with indicator variables. We assume that every indicator variable appears in at most one leaf node. Every arc from a sum node i to a child j is associated with a non-negative weight w_{ij}. Given an SPN S and a node i, we denote S^i the SPN obtained by rooting the network at i, that is, by discarding any non-descendant of i (other than i itself). We call S^i the sub-network rooted at i. If $\mathbf{w}$ are the weights of an SPN S and i is a node, we denote by $\mathbf{w}_i$ all the weights in (any node of) the sub-network S^i rooted at i, and by w_i the vector of weights w_{ij} associated with arcs from i to children j. The height of a (sub)network S equals the longest path (in number of arcs) from the root to the deepest internal node (hence we do not count leaves as for the height).

The *value* of an SPN S at a given configuration λ of its indicator variables, written $S(\lambda)$, is defined recursively in terms of its root node i. If i is a leaf node associated with indicator variable λ_{i,x_i} then $S^i(\lambda) = \lambda_{i,x_i}$. Else, if i is a product node, then $S^i(\lambda) = \prod_j S^j(\lambda)$, where j ranges over the children of i. Finally, if i is a sum node then $S^i(\lambda) = \sum_j w_{ij} S^j(\lambda)$, where again j ranges over the children of i. The *scope* of an SPN with a single (leaf) node is the respective random variable. The scope of an SPN with a root node which is not a leaf is the union of the scopes of the sub-networks rooted at every child of such root node. Every joint distribution over categorical random variables can be represented by an SPN. In order to ensure that any SPN computes a valid distribution and its marginals, we impose the following properties [26]:

Completeness: The scopes of children of a sum node are identical;

Decomposition: The scopes of children of a product node are pairwise disjoint;

Normalisation: Weights are positive and the sum of the weights of arcs leaving a sum node is one (the latter is without loss of generality).

Every SPN specifies a probability measure $\mathbb{P}$ such that $\mathbb{P}(X = x) = S(\lambda^x)$ under such conditions, and a marginal probability can be computed by setting all indicator variables of the summed out variables to one. Let $\mathcal{E} \subseteq \mathcal{V}$ and consider some *evidence* $X_{\mathcal{E}} = e$. Then $\mathbb{P}(X_{\mathcal{E}} = e)$ can be computed as $S(\lambda^e)$ [27]. Hence, it follows that $S(\lambda^e) = \sum_{x \sim e} S(\lambda^x)$, where $x \sim e$ represents all configurations of $X = x$ that agree with evidence $X_{\mathcal{E}} = e$. The evaluation of an SPN for a given configuration λ of the indicator variables can be performed by a bottom-up message propagation scheme where each node sends to its parent its value. $\mathbb{P}(X = x)$ can also be computed by partial propagation over a subset of the SPN nodes rather than a full propagation over all nodes in the SPN [5]. The whole procedure takes linear time and space (in the SPN size). Conditional probabilities can also be obtained in linear time either by evaluating the network at query and evidence (then dividing the result) or by applying Darwiche's differential approach, that propagates messages up and down the network [7, 25]. Other inferences such as maximum-a-posteriori inference are however NP-hard to compute or even to approximate [6].

2.1. Learning from data

Many algorithms have been devised to "learn" SPNs from data [1, 11, 14, 18, 23, 24, 28, 30, 35]. A well-used approach is to employ a greedy search on the space of SPNs augmenting the network in either a top-down or bottom-up fashion. The sum nodes in an SPN can be interpreted as hidden (latent) variables in a mixture model, and the product nodes can be seen as defining context-specific independences [25, 27]. The number of values of the hidden variable (and hence the number of mixtures) corresponding with a sum node is the number of outgoing arcs. For instance, [14]'s LearnSPN algorithm starts with a single node representing the entire dataset, and recursively adds product and sum nodes that divide the dataset into smaller datasets (if columns are variables and rows are samples, then sum nodes can be seen as horizontal partitions, while product nodes are vertical partitions) until a stopping criterion is met. Product nodes are created by using independence tests (pairwise tests will form a dependency graph, and variables in distinct components of the graph become the scope of the children of the product node), while sum nodes are created by performing clustering on the row instances. The weights associated with sum nodes are learned as the proportion of instances assigned to a cluster. In principle any independence test and clustering method can be used. Since we are mainly interested in studying the possibility of cascading models and how that can improve accuracy, we decided to employ efficient and well-known approaches for those tasks. During clustering, we run an improved partition around medoids (PAM) [16, 19] method for its superior efficiency in comparison to the EM and original PAM algorithm suggested in previous approaches [5, 10, 14, 34]. This allows for faster and efficient learning for larger datasets., and as independence test we employ the G-test [33] (leaving the exploration of other methods for future work – in spite of that, these choices have been shown to yield good accuracy, as we will also see in the experiments).

One of the possible uses of SPNs is in building probabilistic classifiers, that is, in estimating a probability distribution over class and attribute values, which can then be used to classify objects into classes by maximising the class conditional probability [13]. Image-completion is another common application of SPNs [1, 4, 10] closely related to classification. For instance, [27] learned SPNs to predict the missing pixels of an image. We employ two variations that can be valuable for classification: (1) we allow the SPN learning to start with either a product or a sum node (a parameter controls that; we call it a sum-rooted SPN if the root is a sum node, and product-rooted SPN otherwise); (2) we may force the first sum node containing a target variable (the class variable in classification problems) to be partitioned based on the values of that variable (we call this as class-discriminative SPN, since the data is partitioned by the class values). Finally, we also control the maximum height of the learned SPN. If we define an upper bound height h, then when a node reaches depth $h-1$, it is forced to become a product node with all variables independent of each other, where each child will be a single sum node (at height h) with univariate scope defining an univariate probability distribution for that variable (and will have as children the indicator functions for it). The height control is intended to analyse if the learned SPNs are prone to overfitting related to their depth, but it also allows us to create a clear relationship between SPNs and other classifiers. The learning algorithm is displayed below.

```
LEARN(D, PRODUCT-FIRST, CLASS-DISCRIMINATIVE, MAX-HEIGHT, HEIGHT): returns an SPN
Inputs: $D$: dataset; PRODUCT-FIRST and CLASS-DISCRIMINATIVE: Booleans, MAX-HEIGHT: controls the height; HEIGHT: starts at 0 for the main root node

  1. If $D$ contains a single variable, then
      (a) Create a sum node $S$ with children as the leaf nodes corresponding to the values of that variable, and weights according to their frequencies (with possible
```

   regularisation) in the data.

   (b) Return S.
2. If HEIGHT equals MAX-HEIGHT-1, then
   (a) Create a product node S and partition the dataset into $D_1, \ldots, D_t$ (where t is the number of variables in D), with one single variable per D_i.
   (b) For $i = 1, \ldots, t$, call LEARN(D_i, FALSE, FALSE, MAX-HEIGHT, HEIGHT$+1$) and add these SPNs as children of S.
   (c) Return S.
3. If PRODUCT-FIRST, then
   (a) Create an empty product node S. Create an empty (undirected) graph.
   (b) For every i, j, compute G-TEST(D, X_i, X_j) and if the p-value is below PVAL-THRESHOLD (a global parameter), include an edge (i, j) in the graph.
   (c) Compute the connected components $C_1, \ldots, C_t$ of the graph, and partition the dataset D into $D_1, \ldots, D_t$ based on the variables that appear in each component (D_i shall contain all data related to variables in C_i).
   (d) For $i = 1, \ldots, t$, call LEARN(D_i, FALSE, CLASS-DISCRIMINATIVE, MAX-HEIGHT, HEIGHT$+1$) and add each returned SPN as a child of S.
   (e) Return S.
4. Create an empty sum node S.
5. If CLASS-DISCRIMINATIVE (and the class variable is part of D), then
   (a) Partition the dataset D into $D_1, \ldots, D_t$, with t the number of classes, based on the values of the class variable in D.
   (b) For $i = 1, \ldots, t$, call LEARN(D_i, FALSE, FALSE, MAX-HEIGHT, HEIGHT$+1$) and add each returned SPN as a child of S with associated weight proportional to the number of occurrences (with a possible regularisation) of i in the class variable.
   (c) Return S.
6. Partition the dataset D into $D_1, \ldots, D_t$, using a call to the PARTITION-AROUND-MEDOIDS clustering algorithm, where samples are seen as multi-dimensional vectors.
7. For $i = 1, \ldots, t$, call LEARN(D_i, FALSE, FALSE, MAX-HEIGHT, HEIGHT$+1$) and add each returned SPN as a child of S with associated weight proportional to number of samples in cluster i.
8. Return S.

By using some particular settings when calling the learning algorithm, we obtain variations/generalisations of some Bayesian network classifiers.

Lemma 1. *Let a Bayesian network classifier be defined as a model where the class variable is the only root node and has all features (that is, non-class variables) as children. A class-discriminative sum-rooted* SPN *generalises a Bayesian network classifier, that is, it can encode the same model as a Bayesian network classifier.*

Proof. Because the class variable of the Bayesian network classifier is a parent of every single variable, their conditional probability tables will be indexed by the values of the class variable, and hence will be learned using the data related to that class only. Each child of the root node of the class-discriminative sum-rooted SPN can represent the conditional probability of the features given that particular value of the class, so it can encode the same distribution as the Bayesian network classifier (since an SPN can represent any distribution, even if that may be resource demanding). Finally, the marginal probability of the class, which is encoded in the root node of the Bayesian network classifier, can be encoded in the weights of the sum node which is the root of such SPN. □

Lemma 2. *A class-discriminative sum-rooted* SPN *of height 2 generated by Algorithm* LEARN *is equivalent to a Naive Bayes model.*

Proof. The result follows from Lemma 1 and the height restriction, since every node which is a child of the root node will reach the limit of the height and will become a product node that makes all variables independent. Therefore, the sum nodes of the next layer represent the conditional probability of each feature given the class (for the appropriate value of the class according to the path from the root of the SPN). □

Lemma 3. *A class-discriminative product-rooted* SPN *of height 3 generated by Algorithm* LEARN *is equivalent to a Naive Bayes model over variables that were not discarded by a feature selection procedure (based on components of the independence graph constructed by pairwise tests).*

Proof. There are two points to realise here:

1. The product root node will act as a feature selection procedure, since the scopes of the children are disjoint, only one of them will have the class variable; during testing, the messages coming from all other children will be irrelevant, since they will be the same whichever is the class value, and thus they could be safely ignored (computing them does not cost much in SPNs, but crucially they do not interfere in the class prediction).
2. The only child of the product root node containing the class will be a class-discriminative sum-rooted SPN of height 2, because (by the learning algorithm) the child of a product node cannot be another product node (it is redundant to have a product node as child of a product

node). Finally, by Lemma 2 this sub-network is equivalent to a Naive Bayes model.

□

3. Hierarchical Sum-Product Networks

To expand on the ability of SPNs to apply successfully to classification on datasets which are perhaps noisy or otherwise difficult to decisively classify, we propose an approach of cascading SPNs into what we term a *Hierarchical Sum-Product Network* (HSPN). This is achieved through a combination of ensemble learning methods and the deferring of predictions using multiple SPN models achieved through the use of a *Credal Sum-Product Network* (CSPN).

As a high-level overview, our approach is one wherein a list of SPNs ordered according to their training prior, are learned using subsets of the training data and used to return a classification. The decision-making process in testing time iterates through the hierarchy of SPNs as each fails our expected robustness target until one finally meets it or the last model has been reached. The ranking of the models in the list, the model which is used as the final default model, and the criteria and thresholds learned to make this possible are described in further detail here.

To begin with, our approach takes the existing data and generates a training and test set through cross-validation. Within each split, the training set is used to generate a predetermined number u of smaller subsets of equal size as part of a bagging process with repetition. For each of these generated bags, a different SPN S_i is learned with $i = 1, ..., u$, all with identical parameters in order to produce a diverse set of models in a generalisable way (the SPNs vary because they are trained in different subsets of the training set).

As the next step, a criterion needs to be applied in order to combine or order these models and their decisions. Existing common approach based that decision solely in their training or validation accuracy. While this is logical, it is also susceptible to overfitting to the training data. Instead, we propose a schema where the models are ordered by the expected training accuracy, and the final decision (about which model to use for prediction of a given test instance) is taken on the basis of the estimated robustness of the given test instance. This allows to generalise better to new unseen samples and circumstances and mitigate severe overfitting to training data.

In order to obtain a measure of robustness for issued predictions, we allow parameters of the model to vary within a certain set and verify whether the outcome (predicted class) remains the same whichever choice of parameters we make. Let $S_{\mathbf{w}}$ denote a SPN whose weights are $\mathbf{w}$. We can investigate the robustness of a model by varying the weights $\mathbf{w}$ inside some fixed space, subject to the constraint that they still define a (normalised) SPN, following on the work of [21]. A CSPN is a set $\{S_{\mathbf{w}} : \mathbf{w} \in \mathcal{C}\}$, where $\mathcal{C}$ is a subset of the Cartesian product of probability simplexes, and each probability simplex constrains only the weights associated with a single sum node. In this way, an SPN is a CSPN where weights take values in a singleton $\mathcal{C}$, and every choice of weights $\mathbf{w}$ inside $\mathcal{C}$ specifies an SPN. Thus, the CSPN induces a *credal set*, that is, a (not necessarily convex) set of probability measures [20]. We are particularly interested in CSPNs formed as follows: for each sum node with local weights w, we use an ϵ-contamination of w (with $0 \leq \epsilon \leq 1$) such that no child value v_j is below zero and the value of all children sum to 1

$$\mathcal{C}_{w,\epsilon} = \left\{ (1-\epsilon)w + \epsilon v : v_j \geq 0, \sum_j v_j = 1 \right\} . \tag{1}$$

Under such definition, we obtain w when $\epsilon = 0$ (a precise SPN) and the whole simplex when $\epsilon = 1$ (a vacuous SPN).

Now, given a class variables X_c, evidence $X_{\mathcal{E}} = e$, and an CSPN, we say that an assignment c_1 for X_c *credally dominates* another assignment c_2 if

$$\min_{\mathbf{w} \in \mathcal{C}} \Big[S_{\mathbf{w}}(\lambda^{c_1,e}) - S_{\mathbf{w}}(\lambda^{c_2,e}) \Big] > 0 \tag{2}$$

whereby the configuration c_1 alongside evidence e produces an SPN more probable at the smallest level of contamination than an SPN with configuration c_2. This task can be performed efficiently in polynomial time [21] when the number of classes is bounded and the internal graph of the SPN is a tree. Following the proposals of [9], assume an SPN has been learned from data, and used to issue a classification based on the maximum probability class label. Given a value $\epsilon > 0$, we say that a classification is ϵ-robust if the respective class label is not credally dominated by any other class label in the CSPN obtained by ϵ-contamination of the SPN. The robustness of a prediction ρ is the largest value of ϵ for which the maximum probability class is robust.

In this work, we employ the robustness of a prediction ρ to decide whether to defer the decision to another model. Our approach here is based on cascading our two or more CSPNs obtained through our earlier bagging until one of them is confident in providing a decision or the last CSPN is reached. Because the CSPNs issue the same prediction of the corresponding SPN, from now on we call them SPNs (but bear in mind that they are equipped to issue a robustness value). In other words, a list of SPNs $S_1, \ldots, S_t$ (with $t \geq 2$) has been learned from data and a list of robustness thresholds $\tau_1, \ldots, \tau_t$ are also constructed during training. These SPNs are then used in order to predict the class variable, in sequence. When S_i is employed to predict instance x, we compute the robustness value $\rho_{S_i}(x)$ of the issued prediction, and if $\rho_{S_i}(x) \leq \tau_i$, then we ignore the prediction and increment i, moving to the next SPN until we reach the last level of our hierarchy (in that case, the last SPN is used regardless of its robustness value).

Before doing this however, we order our list of SPNs in a meaningful way to attempt to make the most out of their respective strengths. Firstly we sort our models by their training accuracy Acc so that the SPN S_b with the highest accuracy is selected as initial level:

$$S_b = \arg \max_{\forall i=1\ldots t} \{\mathrm{Acc}(S_i|D_i\} \,. \tag{3}$$

where D_i is the training dataset used to learn S_i and $\mathrm{Acc}(S|D)$ means the accuracy of classifier S over instances D. This initial level is followed by the rest of the models in decreasing order of accuracy, repeating the process with the remaining SPNs. Assuming perfect additional models, the maximum margin of improvement that our hierarchical ensemble could gain, in this ideal case, regarding this first level is defined as the difference between the maximum possible accuracy (that is, 100%) and $\mathrm{Acc}(S_b|D)$. Since those ideal models cannot be expected, we define a parameter g to represent the targeted gain we wish to see from each following weaker model in the hierarchy:

$$\mathrm{Acc}(S_b|D) + \Big(1 - \mathrm{Acc}(S_b|D)\Big) \cdot g \,. \tag{4}$$

where g is a percentage $0 \leq g \leq 1$ of the difference between our best SPN and a perfect 100% accuracy which we will set as our improvement target when learning thresholds. If $g = 0$, we simply want accuracy better than $\mathrm{Acc}(S_b|D)$, while $g \to 1$ tends to 100% accuracy as goal ($g = 1$ would ensure that specific classifier is never used to classify).

To illustrate the purpose of this, let's imagine a hierarchy with two ranked SPN with training accuracies of 0.8 and 0.7 respectively. Since the second SPN model SPN_2 is weaker a priori, we only expect decisions to be taken by this model in the subset of samples where the decisions are very accurate, in other words, more accurate than the targeted gain defined in Eq. 4. For achieving this accuracy with SPN_2, weaker and less robust samples are discounted from the subset until $\mathrm{Acc}(S'_2|\textit{More robust decisions only}) \geq \textit{Targeted Gain}$.

Let $D(S,\tau) = \{x \in D : \ \rho_S(x) \geq \tau\}$, that is, $D(S,\tau)$ is the subset of instances of D over which S has robustness at least τ. Since those weaker models are only deemed to be used in those cases where there is some extra confidence on their prediction, their prediction will only be considered if the robustness value ρ is above a model's robustness threshold τ_i. Following the previous example, decisions at testing time are only referred to teh second and following levels if the prediction of SPN_1 is deemed not to be reliable, which is measured by the robustness of its decision, that is below the learned robustness threshold τ_i. Thus, these thresholds values τ_i are learned during training as the minimum threshold for which the accuracy of the subset of predictions fulfilling the robustness condition is such that $\mathrm{Acc}(S_i|D(S_i,\tau_i)) > \mathrm{Acc}(S_b|D) + \Big(1 - \mathrm{Acc}(S_b|D)\Big) \cdot g$. No τ threshold is needed for the last level of the hierarchy since the last SPN prediction is used regardless of its robustness value.This is obtained by algorithms HSPN_LEARN and HSPN_PREDICT.

```
HSPN_LEARN($D, t, g$): returns the learned models and threholds required for the Hierarchical SPN
Inputs: $D$: training dataset; $t$: number of of SPNs to be learned through bagging. $g$: improvement percentage used to calculate the expected gain in the hierarchy.

1. For $i = 1, \ldots, t$
    (a) Create a subsampled dataset $D_i$ using bagging.
    (b) Train $S_i \leftarrow$LEARN($D_i$, PRODUCT-FIRST, CLASS-DISCRIMINATIVE, MAX-HEIGHT, HEIGHT) using global pre-defined parameters.
2. Estimate the SPN of maximum accuracy $S_b = \arg\max_i \{\mathrm{Acc}(S_i, D_i)\}, \forall i \in 1, \ldots, t$.
3. Sort the SPN models in decreasing order of accuracy $S' = S_{i_1}, \ldots, S_{i_t}$, with $i_1 = b$.
4. Add $S_b$ also to the end of the list $S'$. (A possible add-on here is to replace this last layer or even first and last layer with a different model, for instance with the SPN learned from the full training data $S_{\mathrm{full}}$ =LEARN($D$, PRODUCT-FIRST, CLASS-DISCRIMINATIVE, MAX-HEIGHT, HEIGHT).)
5. For $i$ over the indexes of SPNs in the list $S' - 1$:
    (a) Select $\tau_i$ as the smallest $\tau \in [0,1]$ so that $\mathrm{Acc}(S'_i|D(S'_i,\tau)) > \mathrm{Acc}(S_b|D) + \Big(1 - \mathrm{Acc}(S_b|D)\Big) \cdot g$ (if that never happens, then $\tau_i \leftarrow 1$).
6. Return the learned list of sorted models $S'$ and associated set of $\tau$s.
```

```
HSPN_PREDICT($D, S', \tau$): returns a set of classifications
```

```
Inputs: $D$: test dataset; $S'$: list of SPNs learned in training and sorted by accuracy; $\tau$: set of learned thresholds

1. For $D_j$ as the samples in $D$:
    (a) For $i$ over indexes in the list $S'$:
        i. If $\rho_{S'_i}(D_j) > \tau_i$ or $S'_i$ is last in $S'$, then
            A. Take the prediction from $S'_i$: $pred(D_j) = \arg\max_C \Pr_{S'_i}(C|D_j)$.
            B. BREAK
2. Return the set of HSPN classifications $pred$.
```

When defining the order of our cascading we again sorted the models by accuracy ranging from the best model to the weakest by training accuracy with our strongest SPN being repeated again as the last model in cascading (so as to become the surrogate model if no other has issued a prediction). In doing so we sought to create a model which discriminated heavily against weaker models except when they were very confident in their predictions, surpassing in robustness the (expected) better models in the initial levels of the hierarchy and only if those already did not issue a prediction. The initial model accepts only the most confident of classifications from its strongest SPN in the list before then deferring to the next strongest if the first model's threshold is not met. This repeats with each weaker model. To deal with those cases where no decision is taken with sufficient robustness throughout the full hierarchy, a final level is added at the bottom of the hierarchy. This could be either the model with the highest accuracy in training (that is, repeating the first layer of the hierarchy) or a new model with access to all of the data in training. The result of this combination is that the intermediate levels of the hierarchy (which are potentially weaker models) are only used on the condition that they are extremely confident. Our intuition is that in doing this our model should only ever improve with cascading with additional layers. In deciding whether to use a model with access to all the information or a bag which performed the strongest in training as the final layer is something that we leave for the experimental section.

The following lemma shows that our HSPN approach can only improve classification accuracy (in asymptotic terms) with respect to the best classifier in the collection.

Lemma 4. *Let $S' = S_1, \ldots, S_{t-1}, S_t$ (with $S_1 = S_t$) be a list of classifiers and corresponding τ as obtained by the method* HSPN_LEARN*, D the training data and D^{val} the validation data, both sampled independently and identically* *distributed. Let C be the classifier as defined by the method* HSPN_PREDICT *with data, S' and τ as input. As $|D|$ and $|D^{val}|$ go to infinity, we have $Acc(C|D^{val}) \geq Acc(S_1|D^{val})$.*

Proof. As $|D|$ and $|D^{\text{val}}|$ go to infinity, $\text{Acc}(S_i|D^{\text{val}}(S_i, \tau_i))$ approaches the value $\text{Acc}(S_i|D(S_i, \tau_i)) \geq \text{Acc}(S_1|D)$ for every i (by construction). We also have $\text{Acc}(S_1|D^{\text{val}})$ approaching $\text{Acc}(S_1|D)$, and hence all classifiers, whenever they are used, obtain accuracy equal or superior to that of S_1, and the result follows.

□

4. Experiments

We begin the experiments by creating a collection of SPNs based on identical learning constraints but obtained via random bagging of the initial training set with repetition. A total of 47 datasets from UCI (`http://archive.ics.uci.edu/ml/`) were used and the accuracy of the multiple SPNs obtained using a 10-fold cross-validation (repeated 6 times and averaged). All datasets were curated and contain only categorical variables. Our goal is to compare our deferring hierarchical model against non-hierarchical competitors that may use multiple learned SPNs. We also aim to perform ablation experiments for our method in order to examine the effect of different parameters on the outcome and deferral process among the HSPN layers. Thus, we will explore the effect of parameters such as the threshold g that define when an SPN issues predictions, and the number of layers in the hierarchy.

4.1. Comparison with other ensemble ideas

First of all, some alternative well-performing methods were designed to act as comparison in our experiments. The goal of this initial experiment is to demonstrate that the HSPN approach performs on par with well-performing approaches, three of which we compare against our HSPN and describe here. As first competitor, we use the best learned individual SPN out of all the bagged models in each fold of the cross-validation. Our second competitor is an SPN obtained using the entire training set (respecting the folds, obviously). Note that these two approaches are equivalent to our HSPN with threshold parameter g set to 100% and an appropriate last layer, that is, where all decision are referred to the last level of the hierarchy, which could be defined

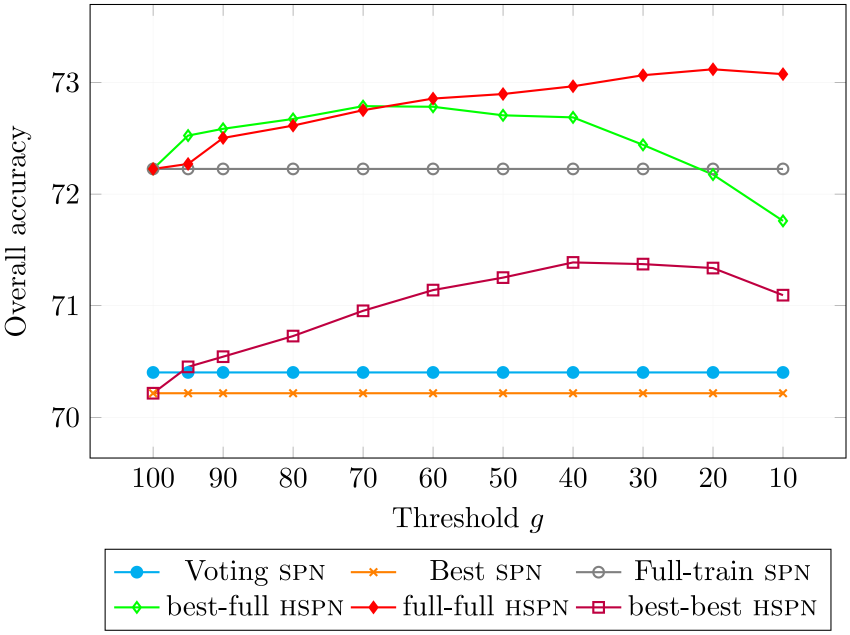

Figure 1: Five-layer HSPN average accuracy across all 47 UCI datasets. Threshold g as defined in Expression 4. Voting is by majority, full-train SPN uses all data and no cascading, while best-full, full-full and best-best indicate which SPNis used in the first and last layers of the HSPN model.

as either the *best* bagging model or the one with the *full* training set. Finally, the third comparative approach was implemented comprising of a voting system whereby the classification which received the most votes from the SPNs generated through bagging would be issued with tie-breakers resolved randomly. Since it is possible to use strongly performing models other than the best performing SPN as first or last layer and still observe an increase in accuracy as by Lemma 4, three variations of our HSPN are tested where the stronger classifier used for the first and last layers are respectively: the best SPN out of the bagging S_b (*best-best*) in both layers, the best SPN S_b and a SPN trained with the full dataset D before bagging S_{full} (*best-full*), and a SPN trained with the full dataset D S_{full} in both layers (*full-full*). All our HSPN versions in these experiments are composed of five layers, resulting in three SPNs created by bagging plus the stronger classifiers as first and final layers. The comparison is performed ensuring the same conditions to all compared methods so that the datasets in each fold is exactly the same. The threshold learning parameter g is varied between 10% and 100%. Each layer is learned as a product-rooted SPN without learning constraints with regard to height or class-discrimination.

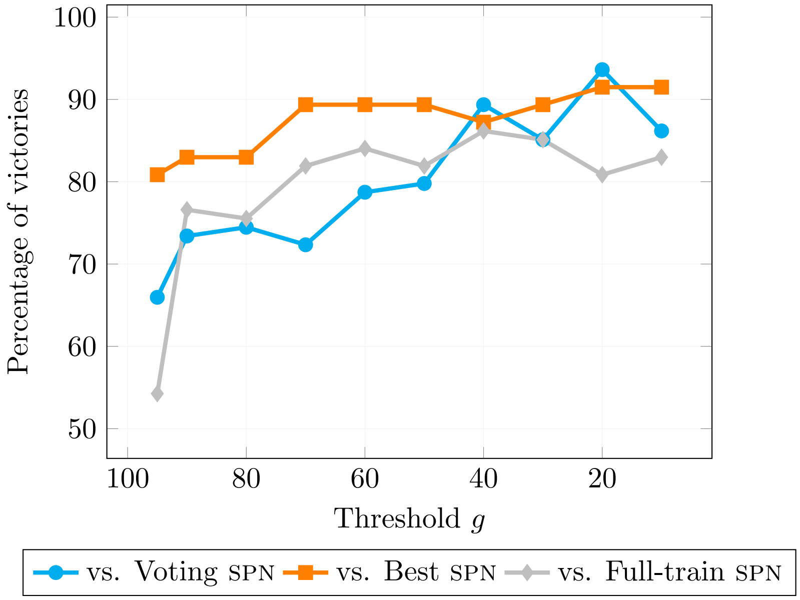

Figure 2: Percentage of datasets for which the five-layer full-full HSPN wins (in terms of overall accuracy for a dataset) against competitors (ties are equally split). Therefore, higher values are better.

In Figure 1 we show the average accuracy across all datasets showing that on average the HSPN model using the full training set as first and last layers performed the strongest overall with accuracy increasing as the threshold for deferral was relaxed and more models were utilised in predicting the classes. Moreover, it can be observed that, while maximum accuracy for our chosen model is obtained using a threshold g around 20%, this parameter is not critical and mostly easy to tune since no large decrease is produced for a large range of values. We also observe that all HSPN models obtain average accuracies higher than that of their initial layer model for all values of g. Since the *full-full* HSPN model (that is, the one with first and last layer equal to the SPN trained with the full training data, while other layers are produced by bagging) performs best, and hence we have decided to proceed with it from now on. In Figure 2 we see the percentage of datasets for which the full-full HSPN obtained a better result than its competitor (hence values above 50% means that it performed better overall).

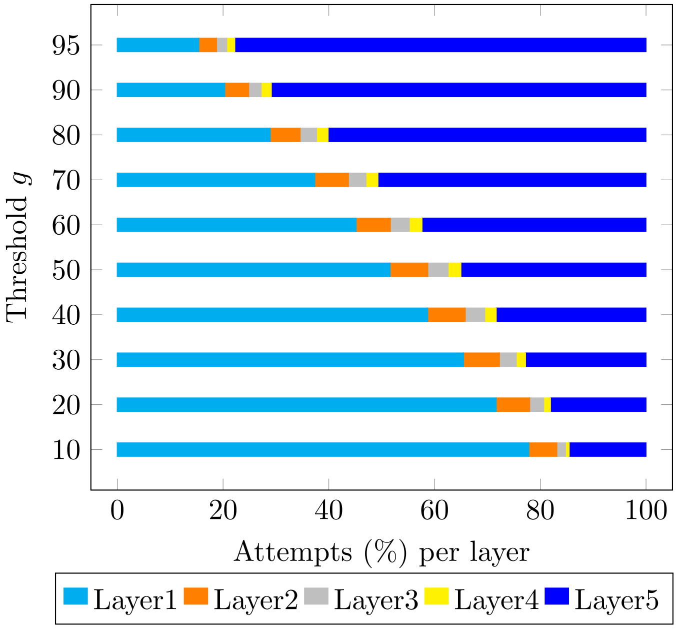

Figure 3: Number of attempts per layer of a five-layer full-full HSPN at different thresholds.

4.2. Analysis of HSPN *Behaviour*

The expanded use of other layers can be observed in Figure 3 whereby as threshold gain is decreased the number of instances being attempted by each layer increases in proportion to that model's position in our hierarchy. Eventually, however the increase in accuracy levels off to varying degrees as the threshold continues to become more and more flexible. This is more apparent as more and more instances are classified using weaker layers without the same level of confidence in that layer's ability to classify it correctly. However if both first and last layers are the same then this levelling becomes more protracted. If we look then to Figure 3 it can be seen clearly how as the threshold is reduced, while each bag layer increases the number and proportion of instances it classifies, the first layer takes on the vast majority of new instances initially deferred to the end. Even at a threshold g of 0 we should not expect to see all of the instances being classified by the initial layer. Most interestingly, the varying value of the threshold has mostly shifted predictions between first and last layer, while the intermediate layers take care of a somewhat similar number of predictions where they are confident enough to issue a classification. In Figure 4 we also note the accuracies achieved on the instances classified by those layers. This may suggest that these layers are only classifying instances for which they are quite certain of a correct answer. Meanwhile it is apparent from averaging over the accuracy of all datasets that our HSPN does not perform much stronger than that of a single SPN trained using the full training set. However, Figure 2 shows the number of individual cases in which our model outperformed each competitor, making it clear that the HSPN succeeded in outperforming each competitor including the full train SPN in considerably more than 50% of the 47 datasets.

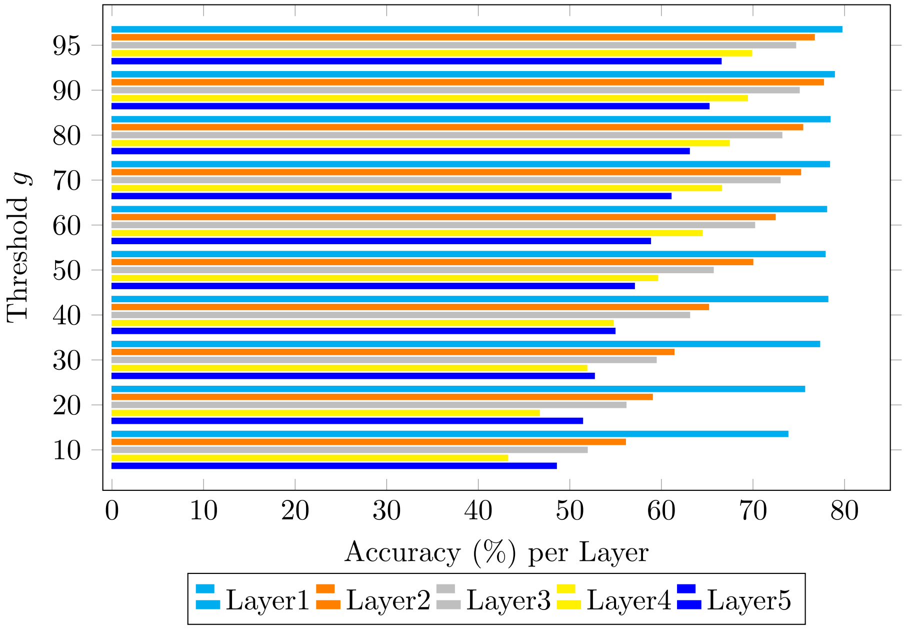

Figure 4: Average accuracy achieved by each layer in a five-layer full-full HSPN.

4.3. Ablation Experiments: Increasing the number of layers

Further to this we compare the behaviour of our method when the number of layers is increased from 5 to 11. A collection of 10 SPNs are learnt through bagging as above alongside an SPNs with access to the full fold training set and together these constitute a larger HSPN. In our results we observed that the overall behaviour noted with five layers held for extra layers. The behaviour of proportions of deferral between layers across threshold values and the relative accuracy of each layer is much the same as with 5 layers. As a main variation regarding the smaller HSPN it seems apparent that there is a lower gradient in the accuracy changes regarding the threshold value as the number of layers increases. However, this levelling off is inevitable and it would seem that with additional layers comes less requirement for the threshold to be strict. The number of cases in which the HSPN outperforms the Voting SPN model reducing with the increased layers suggests that adding additional layers will add to the overall complexity of the model and that the improved performance will not grow linearly with the number of layers. The process of learning larger numbers of SPN also brings with it proportionally increased time costs for relatively little performance gain. A single small dataset from our experiments saw its runtime increase from 32 seconds to 1 minute and 13 seconds as we increased the number of layers from 5 to 11. Similarly for a larger dataset the runtime increased from 6 minutes and 53 seconds to 14 minutes and 49 seconds. In total each experiment across all datasets took around 10 days to complete (if using a single modern CPU).

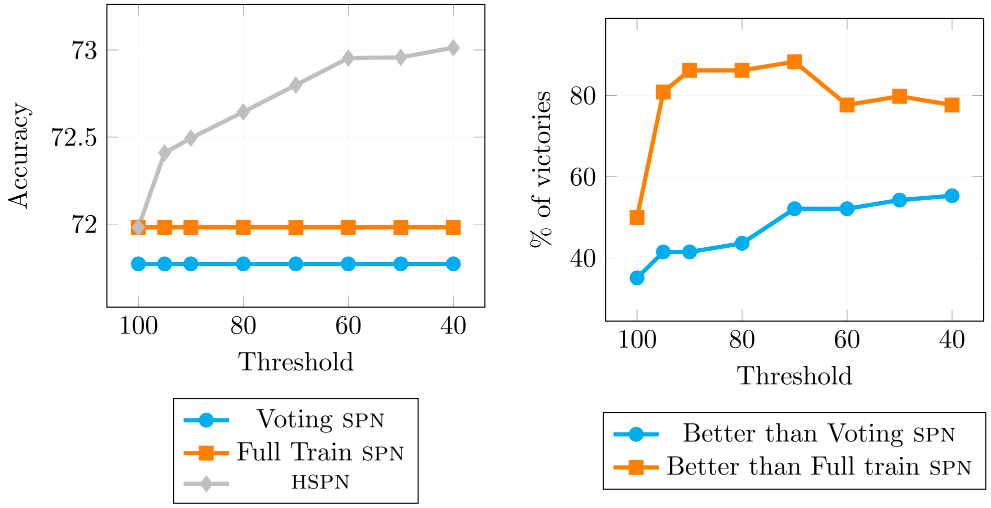

Figure 5: Ten-layer HSPN analysis. On the left, average accuracy against voting and full train SPNs, with superior performance. On the right, percentage of victories against the same competitors (ties are equally split).

5. Conclusions

In this work we developed a new method of ensemble learning using SPNs through the creation of a hierarchy of learned classifiers. In testing time, this herarchical approach defers the classification of the ensemble model to the hierarchical layer deemed most confident according to its robustness value. This robustness having been computed by a credal sum-product network, obtained by perturbing parameters of the original sum-product network. The proof for Lemma 4 is presented to show that our approach can only improve classification accuracy with respect to the initial classifier in the ensemble hierarchy. This proof is given empirical weight through multiple experiments using UCI datasets. The behaviour of the hierarchical SPN continues to be observed when the number of hierarchical layers is increased substantially and also when using different types of models for its initial and final layers. These experiments also suggest that this approach can be more powerful than a number of state of the art competitors.

References

References

[1] T. Adel, D. Balduzzi, A. Ghodsi, Learning the structure of sum-product networks via an SVD-based algorithm, in: Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence, 2015.

[2] L. Breiman, Bagging predictors, Machine Learning 24 (2) (1996) 123–140.

[3] L. Breiman, Random forests, Machine Learning 45 (1) (2001) 5–32.

[4] C. J. Butz, J. S. Oliveira, A. E. dos Santos, A. L. Teixeira, Deep convolutional sum-product networks, in: Thirty-Third AAAI Conference on Artificial Intelligence, 2019.

[5] C. J. Butz, J. S. Oliveira, A. E. dos Santos, A. L. Teixeira, P. Poupart, A. Kalra, An empirical study of methods for spn learning and inference, in: V. Kratochvíl, M. Studený (eds.), Proceedings of the Ninth International Conference on Probabilistic Graphical Models, vol. 72 of Proceedings of Machine Learning Research, PMLR, Prague, Czech Republic, 2018.
URL `http://proceedings.mlr.press/v72/butz18a.html`

[6] D. Conaty, D. D. Mauá, C. P. de Campos, Approximation complexity of maximum a posteriori inference in sum-product networks, in: Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, 2017.

[7] A. Darwiche, A differential approach to inference in Bayesian networks, Journal of the ACM 50 (3) (2003) 280–305.

[8] A. Darwiche, Modeling and Reasoning with Bayesian Networks, Cambridge University Press, 2009.

[9] J. de Bock, A. Antonucci, C. P. de Campos, Global sensitivity analysis for MAP inference in graphical models, in: Neural Information Processing Systems, 2014.

[10] A. Dennis, D. Ventura, Learning the architecture of sum-product networks using clustering on variables, in: F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger (eds.), Advances in Neural Information Processing Systems 25, Curran Associates, Inc., 2012, pp. 2033–2041.

[11] A. Dennis, D. Ventura, Greedy structure search for sum-product networks, in: Proceedings of the 24th International Joint Conference on Artificial Intelligence, 2015.

[12] Y. Freund, R. E. Schapire, A decision-theoretic generalization of online learning and an application to boosting, Journal of Computer and System Sciences 55 (1) (1997) 119 – 139.

[13] R. Gens, P. Domingos, Discriminative learning of sum-product networks, in: Advances in Neural Information Processing Systems 25, 2012.

[14] R. Gens, P. Domingos, Learning the structure of sum-product networks, in: Proceedings of the Thirtieth International Conference on Machine Learning, 2013.

[15] L. K. Hansen, P. Salamon, Neural network ensembles, IEEE Transactions on Pattern Analysis and Machine Intelligence 12 (10) (1990) 993–1001.

[16] L. Kaufman, P. Rousseeuw, Clustering by means of medoids, in: Statistical Data Analysis Based on the L1-Norm and Related Methods, 1987.

[17] D. Koller, N. Friedman, Probabilistic Graphical Models, The MIT press, 2009.

[18] S.-W. Lee, C. Watkins, B.-T. Zhang, Non-parametric Bayesian sum-product networks, in: ICML Workshop on Learning Tractable Probabilistic Models, vol. 32, 2014.

[19] P. J. R. Leonard Kaufman, Clustering Large Applications (Program CLARA), chap. 3, John Wiley & Sons, Ltd, 2008, pp. 126–163.

[20] I. Levi, The Enterprise of Knowledge, MIT Press, London, 1980.

[21] D. D. Mauá, F. G. Cozman, D. Conaty, C. P. de Campos, Credal sum-product networks, in: Proceedings of the Tenth International Symposium on Imprecise Probability: Theories and Applications, 2017.

[22] A. Nath, P. Domingos, Learning tractable probabilistic models for fault localization, in: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, 2016.

[23] R. Peharz, B. C. Geiger, F. Pernkopf, Greedy part-wise learning of sum-product networks, in: Machine Learining and Knowledge Discovery in Databases, vol. 8189 LNAI, 2013.

[24] R. Peharz, R. Gens, P. Domingos, Learning selective sum-product networks, in: ICML Workshop on Learning Tractable Probabilistic Models, vol. 32, 2014.

[25] R. Peharz, R. Gens, F. Pernkopf, P. Domingos, On the latent variable interpretation in sum-product networks, IEEE Transactions on Pattern Analysis and Machine Intelligence (2016) 1–14.

[26] R. Peharz, S. Tschiatschek, F. Pernkopf, P. Domingos, On theoretical properties of sum-product networks, in: Proceedings of the 18th International Con- ference on Artificial Intelligence and Statistics, 2015.

[27] H. Poon, P. Domingos, Sum-product networks: A new deep architecture, in: Proc. 27th Conf. on Uncertainty in Artif. Intell., 2011.

[28] T. Rahman, V. Gogate, Merging strategies for sum-product networks: From trees to graphs, in: Proceedings of the 32nd Conference on Uncertainty in Artificial Intelligence, 2016.

[29] F. Rathke, M. Desana, C. Schnörr, Locally adaptive probabilistic models for global segmentation of pathological oct scans, in: Proceedings of the International Conference on Medical Image Computing and Computer Assisted Intervention, 2017.

[30] A. Rooshenas, D. Lowd, Learning sum-product networks with direct and indirect variable interactions, in: Proceedings of the 31th International Conference on Machine Learning, 2014.

[31] O. Sagi, L. Rokach, Ensemble learning: A survey, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 8 (4) (2018) e1249.

[32] R. E. Schapire, The strength of weak learnability, Machine Learning 5 (2) (1990) 197–227.

[33] R. Sokal, F. Rohlf, Biometry: The Principles and Practice of Statistics in Biological Research, Freeman, 1981.

[34] A. Vergari, N. Di Mauro, F. Esposito, Simplifying, regularizing and strengthening sum-product network structure learning, in: Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases, 2015.

[35] H. Zhao, P. Poupart, G. Fordon, A unified approach for learning the parameters of sum-product networks, in: Advances in Neural Information Processing Systems 29, 2016.