Cascading Sum-Product Networks using Robustness

Diarmaid Conaty DCONATY01@QUB.AC.UK
Jesús Martínez del Rincon J.MARTINEZ-DEL-RINCON@QUB.AC.UK
Queen's University Belfast, Northern Ireland, UK

Cassio P. de Campos C.DECAMPOS@UU.NL
Utrecht University, The Netherlands

Abstract

Sum-product networks are an increasingly popular family of probabilistic graphical models for which marginal inference can be performed in polynomial time. They 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. In particular, 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 is deferred to another model if the outcome is deemed unreliable. We apply this approach on benchmark classification tasks and experiments show that the robustness measure can be a meaningful manner to improve classification 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 (Koller and Friedman, 2009; Darwiche, 2009). In spite of that, many independences are not captured (Chavira and Darwiche, 2005; Zhang and Poole, 1996). Sum-Product Networks (SPNs) are a class of probabilistic graphical models that allow for the explicit representation of context-specific independence (Poon and Domingos, 2011). They are popular due to their ability to represent complex distributions while retaining efficient marginal inference (Nath and Domingos, 2016; Rathke et al., 2017). 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 (Gens and Domingos, 2013; Peharz et al., 2016). We discuss how SPNs naturally generalise the Naive Bayes classifier, which somehow 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. The main contribution of this paper is to introduce an approach to overcome such drawback based on cascading multiple classifiers and deferring the decision when the current prediction is not reliable. In order to measure such reliability, we make use of recent algorithms proposed for credal SPNs (Mauá et al., 2017). 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. For that, we propose to learn the threshold from data, and we empirically show that this framework increases classification accuracy with small extra effort. We study how different SPNs perform under certain network learning constraints, and how they improve when combined using a hierarchy of SPNs that decide, based on their robustness measure, whether to defer the responsibility of prediction to another SPN.

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 $X_{\mathcal{V}} = x_{\mathcal{V}}$ or $X_{\mathcal{V}} = z$. We assume every random variable X_i is categorical and take values in $\{0, \ldots, |X_i| - 1\}$. *Indicator variables* $\{\lambda_{i,j} : j = 0, \ldots, |X_i| - 1\}$ 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 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 (Darwiche, 2003). 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 (this is not a restriction for discrete domains). 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 $\mathbf{w}_i$ the weights in 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.

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 $S(\lambda) = \lambda_{i,x_i}$. Else, if i is a product node, then $S(\lambda) = \prod_j S^j(\lambda)$, where j ranges over the children of i. Finally, if i is a sum node then $S(\lambda) = \sum_j w_{ij} S^j(\lambda)$, where again j ranges over the children of i. The *scope* of an SPN with a single 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 (Peharz et al., 2015):

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

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

Normalisation: The sum of the weights of arcs leaving a sum node is one.

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 $S(\lambda^e)$ (Poon and Domingos, 2011). 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. 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 (Darwiche, 2003; Peharz et al., 2016). Other inferences such as maximum-a-posteriori inference are however NP-hard to compute or even to approximate (Conaty et al., 2017).

2.1 Learning from data

Many algorithms have been devised to "learn" SPNs from data (Gens and Domingos, 2013; Peharz et al., 2013, 2014; Lee et al., 2014; Rooshenas and Lowd, 2014; Dennis and Ventura, 2015; Adel et al., 2015; Rahman and Gogate, 2016; Zhao et al., 2016). Most learning algorithms 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 (Poon and Domingos, 2011; Peharz et al., 2016). 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, Gens and Domingos (2013)'s 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 a partition around medoids (PAM) method (Kaufman and Rousseeuw, 1987), and as independence test we employ the G-test (Sokal and Rohlf, 1981) (we leave the exploration of other methods for future work).

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. For instance, Poon and Domingos (2011) 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 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 $D_1, \dots, D_t$ (where $t$ is the number of variables in $D$), with one single variable per $D_i$.
      (b) For $i = 1, \dots, t$, call LEARN($D_i$, FALSE, FALSE, MAX-HEIGHT, HEIGHT$+1$) and these SPNs as a 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($D, X_i, X_i$) and if the p-value is below PVAL-THRESHOLD, include an edge $(i, j)$ in the graph.
      (c) Compute the connected components $C_1, \dots, C_t$ of the graph, and partition the dataset $D$ into $D_1, \dots, D_t$ based on the variables that appear in each component ($D_i$ shall contain all data related to variables in $C_i$).
      (d) For $i = 1, \dots, t$, 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 $D_1, \dots, D_t$, with $t$ the number of classes, based on the values of the class variable in $D$.
      (b) For $i = 1, \dots, t$, 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 of $i$ in the class variable.
      (c) Return $S$.
  6. Partition the dataset $D$ into $D_1, \dots, D_t$, using a call to the PARTITION-AROUND-MEDOIDS clustering algorithm, where samples are seen as multi-dimensional vectors.
  7. For $i = 1, \dots, t$, 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 contains a Bayesian network classifier as a subcase.*

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. One can see each child of the root node of the class-discriminative sum-rooted SPN as representing the conditional probability of the features given that particular value of the class, so it can represent 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 the SPN. ■

Lemma 2 *A class-discriminative sum-rooted* SPN *of height 2 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 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 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. Deferring predictions

In order to obtain a measure of robustness for issued predictions, we can 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 $\mathbf{w}$. Following the work of Mauá et al. (2017), 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. A *Credal Sum-Product Network* (CSPN) is a set $\{S_{\mathbf{w}} : \mathbf{w} \in \mathcal{C}\}$, where $\mathcal{C}$ is 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 (Levi, 1980). 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 $0 \leq \epsilon \leq 1$)

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

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

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

This task can be performed efficiently in polynomial time (Mauá et al., 2017) when the number of classes is bounded and the internal graph of the SPN is a tree. Following the proposals of de Bock et al. (2014), 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 is the largest value of ϵ for which the maximum probability class is robust. In this work, we employ the robustness of a prediction to decide whether to defer the decision to another model.

Our approach is based on cascading two or more CSPNs until one of them is confident in providing a decision. In other words, a list of SPNs $S_1, \ldots, S_t$ (with $t \geq 2$) is learned from data and a list of robustness thresholds $\tau_1, \ldots, \tau_{t-1}$ is constructed. These SPNs are then used in order to predict the class variable, in sequence. When S_i is employed, we compute the robustness value ϵ of the issued prediction, and if $\epsilon < \tau_i$, then we ignore the prediction and increment i, moving to the next SPN (if $i = t$, we issue a prediction no matter the robustness measure). The robustness thresholds can be manually fixed (since the contamination has a clear meaning, an expert could for instance choose 0.01 to lean towards accepting predictions, or for instance 0.5 to only issue very certain predictions).

We have also implemented an approach of learning the threshold from data: we apply the cascading model of SPNs to the training data using different threshold and we fixed them based on the maximum accuracy (over the training samples). We argue that this does not overfit, because we are only learning when to defer decisions (we will show in the sequel that indeed there is good improvement in overall accuracy by such approach). There are obviously multiple ideas that are reasonable to choose the thresholds, since there is a clear trade-off about their choices and which SPN will end up making the decision. As for the choices of SPNs to include in the list, Section 2 discussed some insights about the relation of SPNs with other classification approaches. With that in mind, a possible strategy is to select SPNs that relate to different models, or to select SPNs with different amount of fitness (and hence a diverse balance between under- and overfitting) to include in the list. We experiment with multiple options in the next section.

4. Experiments

We start the experiments by creating a collection of SPNs based on different learning constraints. Table 4 describes the datasets of UCI (`http://archive.ics.uci.edu/ml/`) that we use and the accuracy results of multiple SPNs using a 10-fold cross-validation (each run 6 times). All datasets were curated and contain only categorical variables. Our goal is to show different accuracies that might be obtained depending on the constraints, and not to select a best SPN against others. In spite of that, the class-discriminative/general SPNs seem to perform better than the Naive Bayes-like SPNs, and the stronger constraints on the height seem to produce poorer classifiers.

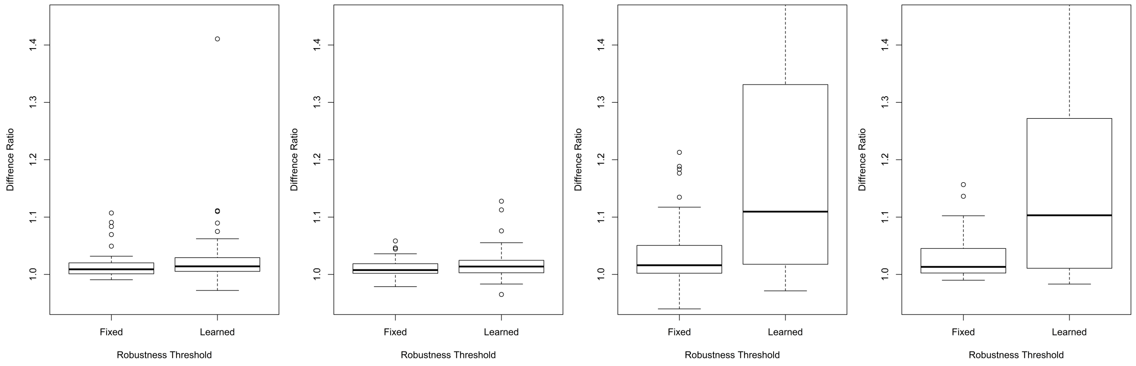

Figure 1: Boxplots of cascading two SPNs which differ by their structural constraints. From left to right, we have a product-rooted SPN without learning constraints, a discriminative SPN, the Naive Bayes-equivalent SPN, and the feature-selection plus Naive Bayes equivalent SPN. The second model of the cascading is always the product-rooted SPN (except for itself, which has the discriminative as second model). In each graph, we show results based on a fixed robustness threshold and based on learning the threshold from data.

Our main goal is to examine the effects of cascading layers of networks. We performed experiments with a variation of constraints. In each cascading experiment, we train the SPNs using all the data, and then we apply the cascading idea through a chosen robustness threshold to defer the decision. The SPNs are evaluated in sequence until one issues a reliable prediction (or the list is exhausted). Figure 1 has four distinct SPNs as first model in the cascading, from left to right: (i) product-rooted SPN without learning constraints, (ii) class-discriminative sum-rooted SPN, (iii) class-discriminative sum-rooted SPN (equivalent to a Naive Bayes classifier), and (iv) product-rooted class-discriminative SPN of height 3 (similar to feature-selection based on independence tests followed by a Naive Bayes classification approach). As deferring model, we employed the product-rooted SPN without learning constraints, so each box plot shows the percentage gain (that

Model	NB	FS+NB	SPN	Dis.SPN	Height3	Height5	Height9
audiology	30.09	29.79	54.13	57.52	29.72	39.16	50.52
autos	88.94	87.89	88.37	87.97	88.62	87.24	88.21
balance scale	67.57	72.11	72.45	71.65	73.60	73.87	72.48
breast cancer	69.81	71.16	72.09	73.08	70.69	73.08	73.02
bridges	56.85	57.17	56.85	57.63	56.70	55.92	56.85
cars	70.15	70.79	87.87	83.54	70.35	74.58	82.86
cmc	42.24	43.47	50.07	49.40	43.09	45.89	48.80
colic	69.43	71.38	81.34	80.84	72.69	78.17	80.21
cylinder bands	56.42	56.51	66.08	68.55	56.73	59.69	65.65
dermatology	48.18	48.64	94.08	94.35	49.87	65.03	85.43
diabetes	67.06	68.34	70.14	70.09	67.90	68.60	70.42
ecoli	60.17	62.35	72.17	71.68	62.15	66.97	71.13
flags	37.80	38.57	40.72	42.70	38.32	39.95	40.38
flare	40.02	51.43	72.06	72.26	51.23	65.16	71.48
glass	38.08	39.88	57.87	55.38	40.34	49.92	56.70
haberman	73.80	73.53	73.53	73.10	73.53	73.53	73.53
hayes roth	42.30	35.10	35.36	51.90	34.98	34.60	35.10
heart h	74.77	74.60	79.59	79.87	73.81	79.14	78.46
heart statlog	73.33	73.39	71.85	72.78	72.22	72.47	72.04
hepatitis	79.35	79.35	81.18	80.11	79.46	81.08	80.76
hypothyroid	92.29	92.29	92.12	92.02	92.29	92.29	92.29
ionosphere	64.15	72.79	80.30	79.25	75.07	81.86	81.06
iris	66.34	67.11	80.56	80.56	65.67	78.56	79.56
kr vs kp	54.12	53.48	92.48	92.97	54.50	60.75	72.74
labor	80.70	76.61	88.31	88.89	73.39	80.12	91.81
liver disorders	54.06	54.01	62.42	63.72	54.49	62.08	62.22
lung cancer	35.94	44.27	42.19	44.79	40.11	42.71	33.85
lymph	69.03	71.06	76.01	75.00	71.85	74.78	76.24
molecular biology	48.74	52.20	71.54	72.8	51.73	67.92	72.49
mushroom	71.90	87.64	99.99	99.99	86.17	92.49	98.15
page blocks	89.77	89.97	93.75	93.72	89.85	91.81	93.48
postoperative	70.74	71.11	71.11	70.37	71.11	71.11	71.11
primary tumor	26.45	26.79	35.60	34.51	26.79	30.63	33.68
segment	26.36	46.00	81.31	81.14	46.85	63.53	76.16
shuttle landing	55.56	60.00	60.00	52.22	60.00	60.00	60.00
sick	93.88	93.88	93.73	93.71	93.88	93.88	93.81
solar flare	45.53	47.81	73.11	73.86	50.99	64.46	73.08
sonar	60.02	62.18	68.19	67.47	60.90	63.94	65.31
soybean	20.21	20.13	75.67	75.21	20.11	33.87	57.76
spambase	73.48	76.02	79.66	79.68	75.48	77.31	78.78
tae	33.11	32.78	41.61	42.39	35.43	41.50	43.60
tic tac toe	65.32	65.78	80.48	81.12	65.29	66.89	77.40
vehicle	35.76	38.50	62.37	62.55	39.08	44.78	57.80
vote	87.90	87.74	95.09	94.94	87.86	91.99	94.56
waveform 5000	61.81	61.76	78.37	77.44	61.80	70.49	78.21
zoo	59.74	60.24	85.98	86.14	60.24	74.59	85.15

Table 1: UCI datasets and classification accuracy (%) for different learned SPNs based on constraints during the training are presented. NB is the Naive Bayes-like SPN, FS+NB is the Naive Bayes-like SPN preceded by feature selection, SPN is the product-rooted network with no constraints on learning, Dis.SPN is the sum-rooted class-discriminative SPN, and those labelled Height are general SPNs restricted by maximum height of 3, 5 and 9 (height 7 is omitted for the sake of space).

is, cascaded accuracy divided by single SPN accuracy over each dataset). In the left-most graph using model (i), we use the class-discriminative sum-rooted SPN as deferring model (otherwise results would always be equal to 1). For each graph, we show both the gain with a fixed threshold at 0.01, and with a learned threshold from training data (as discussed in the previous section). All results show a significant gain by the use of cascading, with more prominent results when the threshold is learned. This suggests that cascading by this robustness measure approach can improve the classification accuracy regardless of the learning constraints that employed in the models.

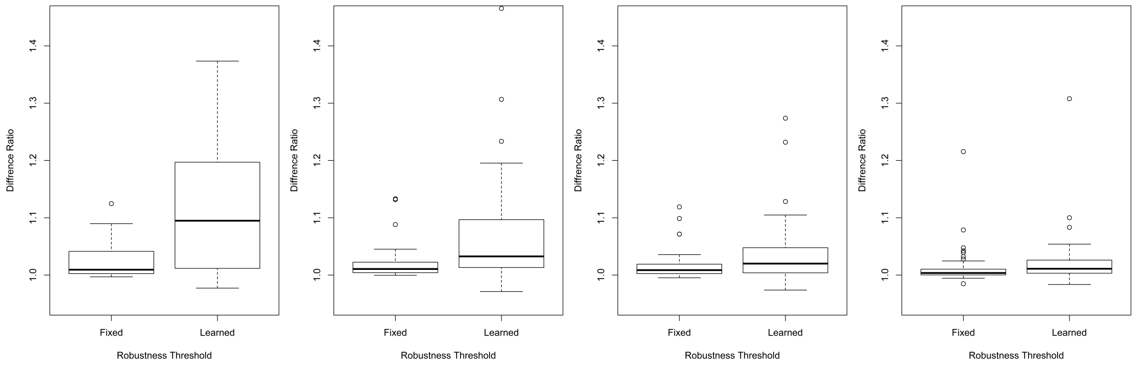

Figure 2: Boxplots of cascading two SPNs which differ by their height upper bound. From left to right, they show the improve in accuracy (cascaded divided by single SPN) of SPNs with height limit at 3, 5, 7 and 9, followed by the SPN of height limit 7 (the one of height 7 is followed by the height 9). We show results based on a fixed robustness threshold and based on learning the threshold from data.

We have further investigated the cascading approach by combining two SPNs of different maximum heights. Clearly the maximum height relates to the amount of fitness of the model, and might related to the trade-off between underfit and overfit in classification. We have chosen to run experiments with four different maximum heights and product-rooted SPNs without further constraints at learning. Figure 2 shows the results of cascading in two layers, starting with maximum height 3, 5, 7, and 9 (from left to right) as the first SPN in the cascading, followed by the SPN of maximum height equal to 7 (except for itself, which is followed by the SPN of maximum height of 9). Again, we show results with robustness threshold to defer decisions of 0.01, and also with a learned threshold using the training data. All results show superior performance of cascading with respect to the single SPN classification (that is, the boxplots are all above the ratio of 1).

Finally, we empirically investigate whether cascading is useful if we do multiple layers instead of only two. We have used a new set of four SPNs with learned threshold and forced it to stop both at layer two or at layer four. The results are shown in Figure 3. We decided to use a combination of different SPNs for the sequences (using product-rooted, sum-rooted, class-discriminative, and different p-values for the G-test of independence). For the sake of space, in the plots we show only some results: from left to right, the cascading sequences are (a),(d),(c),(b); (b),(a),(d),(c); (c),(b),(d),(a); (d),(b),(a),(c), where: (a) product-rooted SPN with p-value 0.05, (b) product-rooted SPN with p-value 0.01, (c) product-rooted class-discriminative SPN with p-value 0.05, (d) class-discriminative sum-rooted SPN with p-value 0.05. Results are always superior to the single SPN classifier, and never decrease significantly by the four layers (and actually increase in most combinations; in the four displayed cases, it has increases in the two on the left but not in the two in the right). Overall, cascading (with two or four layers) has significantly increased the classification accuracy.

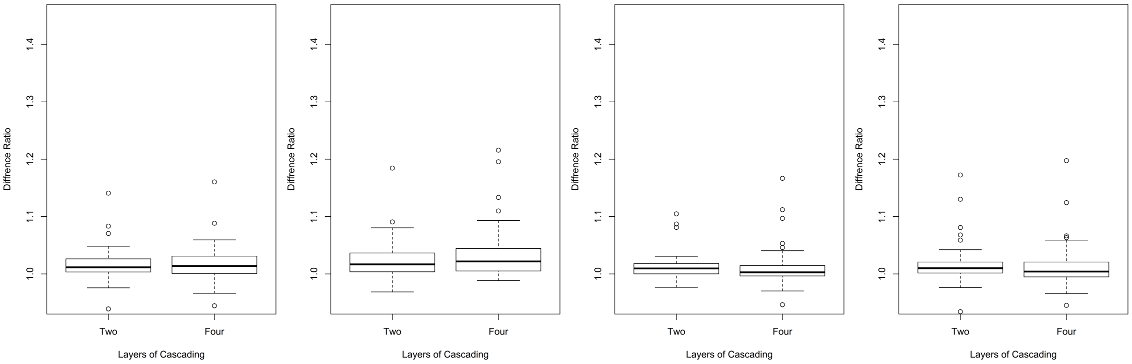

Figure 3: Boxplots of cascading two/four SPNs which differ by their structural constraints. Each graph represents a different order for deferring the classification (chosen arbitrarily to demonstrate that the gains are not highly dependent on the chosen models). We show results based on on learning the threshold from data.

5. Conclusions

In this work we discussed the use of sum-product networks for classification and their relation with some well-known classifiers, notably the Naive Bayes classifier. We created a procedure to cascade multiple sum-product networks in order to build a classifier that predicts in a more robust manner, and thus may achieve higher accuracy than any single sum-product network alone. The choice of which sum-product network should be employed for each instance to be classified depends on the robustness of the prediction as computed by a credal sum-product network, obtained by perturbing parameters of the original sum-product network. Multiple experiments using UCI datasets suggest that the use of a model's own robustness to decide whether to defer issuing predictions to other models has positive implications in the overall accuracy of the classifier, and we argue that such approach through the analysis of robustness may bring benefits also for other classifiers and/or combinations of classifiers. We leave for future work the study of robustness based on variations of the data and/or prior distributions, instead of direct perturbation of the model. We have already started to investigate such avenue, and results look promising.

Acknowledgments

We thank Renato Geh for making his source code publicly available (at `https://github.com/RenatoGeh/gospn`), and Denis Mauá for numerous chats about robustness and SPNs.

References

T. Adel, D. Balduzzi, and A. Ghodsi. Learning the structure of sum-product networks via an SVD-based algorithm. In *Proc. of the 31st Conf. on Uncertainty in Artificial Intell.*, pages 32–41, 2015.

M. Chavira and A. Darwiche. Compiling Bayesian networks with local structure. In *Proc. of the 19th Int. Joint Conf. on Artificial Intell.*, pages 1306–1312, 2005.

D. Conaty, D. D. Mauá, and C. P. de Campos. Approximation complexity of maximum a posteriori inference in sum-product networks. In *Proc. of the 33rd Conf. on Uncertainty in Artificial Intell.*, pages 322–331, 2017.

A. Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009.

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

J. de Bock, A. Antonucci, and C. P. de Campos. Global sensitivity analysis for MAP inference in graphical models. In *Neural Information Processing Systems*, pages 2690–2698, 2014.

A. Dennis and D. Ventura. Greedy structure search for sum-product networks. In *Proc. of the 24th Int. Joint Conf. on Artificial Intell.*, pages 932–938, 2015.

R. Gens and P. Domingos. Learning the structure of sum-product networks. In *Proc. of the 30th Int. Conf. on Machine Learning*, pages 873–880, 2013.

L. Kaufman and P.J. Rousseeuw. Clustering by means of Medoids. In *Statistical Data Analysis Based on the L1–Norm and Related Methods*, pages 405–416, 1987.

D. Koller and N. Friedman. *Probabilistic Graphical Models*. The MIT press, 2009.

Sang-Woo Lee, C. Watkins, and Byoung-Tak Zhang. Non-parametric Bayesian sum-product networks. In *ICML Workshop on Learning Tractable Probabilistic Models*, volume 32, 2014.

I Levi. *The Enterprise of Knowledge*. MIT Press, London, 1980.

D. D. Mauá, F. G. Cozman, D. Conaty, and C. P. de Campos. Credal sum-product networks. In *Proc. of the 10th ISIPTA: Int. Symp. on Imprecise Probability*, pages 205–216, 2017.

A. Nath and P. Domingos. Learning tractable probabilistic models for fault localization. In *Proc. of the 30th AAAI Conf. on Artificial Intell.*, pages 1294–1301, 2016.

R. Peharz, S. Tschiatschek, F. Pernkopf, and P. Domingos. On theoretical properties of sum-product networks. In *Proc. of the 18th Int. Conf. on Artificial Intell. and Statistics*, pages 744–752, 2015.

R. Peharz, R. Gens, F. Pernkopf, and P. Domingos. On the latent variable interpretation in sum-product networks. *IEEE Trans. on Pattern Analysis and Machine Intell.*, pages 1–14, 2016.

R. Peharz, B. C. Geiger, and F. Pernkopf. Greedy part-wise learning of sum-product networks. In *Machine Learining and Knowledge Discovery in Databases*, LNAI 8189, pages 612–627, 2013.

R. Peharz, R. Gens, and P. Domingos. Learning selective sum-product networks. In *ICML Workshop on Learning Tractable Probabilistic Models*, volume 32, 2014.

H. Poon and P. Domingos. Sum-product networks: A new deep architecture. In *Proc. of 27th Conf. on Uncertainty in Artificial Intell.*, pages 337–346, 2011.

T. Rahman and V. Gogate. Merging strategies for sum-product networks: From trees to graphs. In *Proc. of the 32nd Conf. on Uncertainty in Artificial Intell.*, pages 617–626, 2016.

F. Rathke, M. Desana, and C. Schnörr. Locally adaptive probabilistic models for global segmentation of pathological oct scans. In *Proc. of the Int. Conf. on Medical Image Computing and Computer Assisted Intervention*, pages 177–184, 2017.

A. Rooshenas and D. Lowd. Learning sum-product networks with direct and indirect variable interactions. In *Proc. of the 31th Int. Conf. on Machine Learning*, pages 710–718, 2014.

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

N. L. Zhang and D. Poole. Exploiting causal independence in Bayesian network inference. *Journal of Artificial Intell. Research*, 5:301–328, 1996.

H. Zhao, P. Poupart, and G. Gordon. A Unified Approach for Learning the Parameters of Sum-Product Networks. In *Neural Information Processing Systems (NIPS)*, Barcelona, Spain, 2016.