Intelligenza Artificiale 12 (2018) 143–159
DOI 10.3233/IA-170032
IOS Press

Sum-Product Network structure learning by efficient product nodes discovery

Nicola Di Mauro, Floriana Esposito, Fabrizio Giuseppe Ventola* and Antonio Vergari
Department of Computer Science, University of Bari, Bari, Italy

Abstract. Sum-Product Networks (SPNs) are recently introduced deep probabilistic models providing exact and tractable inference. SPNs have been successfully employed in several application domains, from computer vision to natural language processing, as accurate density estimators. However, learning their structure and parameters from high dimensional data poses a challenge in terms of time complexity. Classical SPNs structure learning algorithms work by repeating several times two high cost operations: determining independencies among random variables (RVs)–introducing product nodes–and finding sub-populations among samples–introducing sum nodes. Even one of the simplest greedy structure learner, LearnSPN, scales quadratically in the number of the variables to determine RVs independencies. In this work, we investigate the trade-off between accuracy and efficiency when employing approximate but fast procedures to determine independencies among RVs. We introduce and evaluate sub-quadratic procedures based on a random subspace approach and leveraging entropy as a proxy criterion to split independent RVs. Experimental results on many benchmark datasets for density estimation show that LearnSPN-like structure learners, when equipped by our splitting procedures, provide reduced learning and/or inference times, generally containing the degradation of inference accuracy. Ultimately, we provide an empirical confirmation of a "no free lunch" when learning the structure of SPNs.

Keywords: Machine learning, deep learning, structure learning, probabilistic models, density estimation, Sum-Product Networks

1. Introduction

Density estimation is the unsupervised task of learning an estimator of a joint probability distribution $\mathsf{p}_{\mathbf{X}}$ over a set of random variables (RVs) $\mathbf{X}$ that are assumed to have generated some observed training data[4]. When such an estimator provides a good approximation of the real target distribution, it can be effectively used to perform *inference*–computing the probability of the queries about some RVs in $\mathbf{X}$.

Density estimation can be viewed as one of the key and most general tasks in machine learning. Indeed, many machine learning tasks, such as classification and regression, can be reframed as performing inference over the probability distribution $\mathsf{p}_{\mathbf{X}}$. Hence, the objective is twofold: building a highly *expressive and accurate* estimator, while being able to *efficiently learn* it from data and performing *tractable and exact* inference on it as well.

One of the current challenges in density estimation is trading off these performances: it is rare, if not impossible, to optimize all of them, in practice.

Probabilistic graphical models (PGMs)[18], like Bayesian networks or Markov networks, are able to accurately model highly complex probability distributions. However, inference and learning on PGMs require routines that, even if approximate, may scale exponentially in the worst case [38].

Recently, many tractable probabilistic models (TPMs) have been introduced, allowing exact inference in polynomial time. TPMs like Arithmetic Circuits [7], Sum-Product Networks (SPNs)[33], and Cutset Networks [11, 36] provide a good compromise between expressiveness and tractability by

*Corresponding author: Fabrizio Giuseppe Ventola, Department of Computer Science, University of Bari, Via E. Orabona 4, 70125, Bari, Italy. E-mail: fabrizio.ventola@uniba.it.

compiling complex distributions in compact data structures. Nevertheless, building a TPM from data is a demanding task, and scaling learning algorithms to high dimensional scenarios is still an open issue.

In this paper, we focus on SPNs, as they have proven to be effective density estimators in many application domains, like computer vision [13, 33, 44], speech recognition [31], natural language processing [6] and representation learning [41, 43].

SPNs encode $\mathsf{p}_{\mathbf{X}}$ as a set of sum and product nodes, arranged in a deep architecture. By satisfying some structural conditions, SPNs guarantee several kinds of probabilistic queries over $\mathsf{p}_{\mathbf{X}}$ to be computed exactly and in time linear to the network size.

These properties, and the aforementioned successes, increased the interest around algorithms able to learn both their structure and parameters [8, 14, 26, 37, 42]. One of the first learning schemes, and yet still popular and providing state-of-the-art estimators, is LearnSPN [14]. LearnSPN greedily and recursively decomposes the observed data by either splitting the RVs after discovering statistical (in)dependence relationships, or by clustering samples according to some distance metric. While several variations of LearnSPN have been proposed in the literature—adapting the splitting and clustering routines to deal with specific data distributions, e.g. [16, 26, 27]—the high cost of iteratively performing these operations still constitutes the main computational bottleneck.

Here we focus on splitting RVs into independent subsets, a routine whose worst case complexity is quadratic in the number of the RVs considered [14]. Specifically, we investigate approximate routines for this task, with the aim of being able to improve the time complexity of both learning an SPN and performing inference on it—by obtaining a more compact structure—while trying to preserve the model expressiveness. We extend the work of [9], in which some alternative approximated procedures for splitting RVs have been presented: one based on a random subspace approach and another leveraging entropy as a proxy measure to assess statistical independence among RVs. We make the following contributions: we introduce an improved version of the original stochastic variable splitting method, and propose a new approach based on random sampling, effectively evaluating only a fraction of the samples from the full dataset.

Moreover, we validate all the proposed approaches in a more extensive experimental setting, comprising several additional benchmark datasets for density estimation. Among these, we introduce datasets containing up to 9000 RVs, as a large real-world scenario to investigate the trade-off among inference accuracy in favor of inference and learning times for our approximate routines.

Our experimental results reveal that the random subspace proposed approaches effectively trade off likelihood accuracy in favor of shorter learning times and smaller networks—i.e. faster inference. On the other hand, the entropy based one surprisingly leads to the construction of more complex networks—favoring model expressiveness but slowing inference down. All in all, these results can be interpreted as an empirical proof that there is "no free lunch" in learning SPNs via LearnSPN-like algorithms: the degradation of inference accuracy is matched by faster inference and learning times as one would expect from the inherent trade-off in density estimation.

2. Sum-Product Networks

An SPN S is a computational graph defined by a rooted directed acyclic graph, encoding an unnormalized probability distribution $\mathsf{p}_{\mathbf{X}}$ over a set of RVs $\boldsymbol{X} = \{X_1, \ldots, X_n\}$, where internal nodes can be either *weighted sum* or *product* nodes over their children, and *leaves* are univariate tractable distributions defined on a RV $X_i \in \boldsymbol{X}$.

Each node $n \in S$ has a *scope*, denoted as $\mathsf{sc}(n) \subseteq \mathbf{X}$, and defined as the set of RVs appearing as its descendant leaves. The sub-network S_i, rooted at node i, encodes the unnormalized distribution over its scope. Each edge $i \rightarrow j$ emanating from a sum node i to one of its children j has a non-negative *weight* w_{ij}. The set of all sum node weights and the leaf distribution parameters constitute the *network parameters*.

Sum nodes can be viewed as mixtures over probability distributions whose coefficients are the children weights, while product nodes identify factorizations over independent distributions. Examples of SPNs are depicted in Fig. 1. In the following, we consider $\boldsymbol{X}$ to be discrete valued RVs.

For a given state $\boldsymbol{x}$ of the RVs $\boldsymbol{X}$, we will indicate with $S(\boldsymbol{x}) = \mathsf{p}(\boldsymbol{X} = \boldsymbol{x})$ the unnormalized probability of $\boldsymbol{x}$ according to the SPN S, that is the root node value when the network is evaluated after having observed $\boldsymbol{X} = \boldsymbol{x}$. An SPN is defined *decomposable* if the scopes of the children of each product node are disjoint. It is defined *complete* when the scopes of the children of each sum node are the same. These

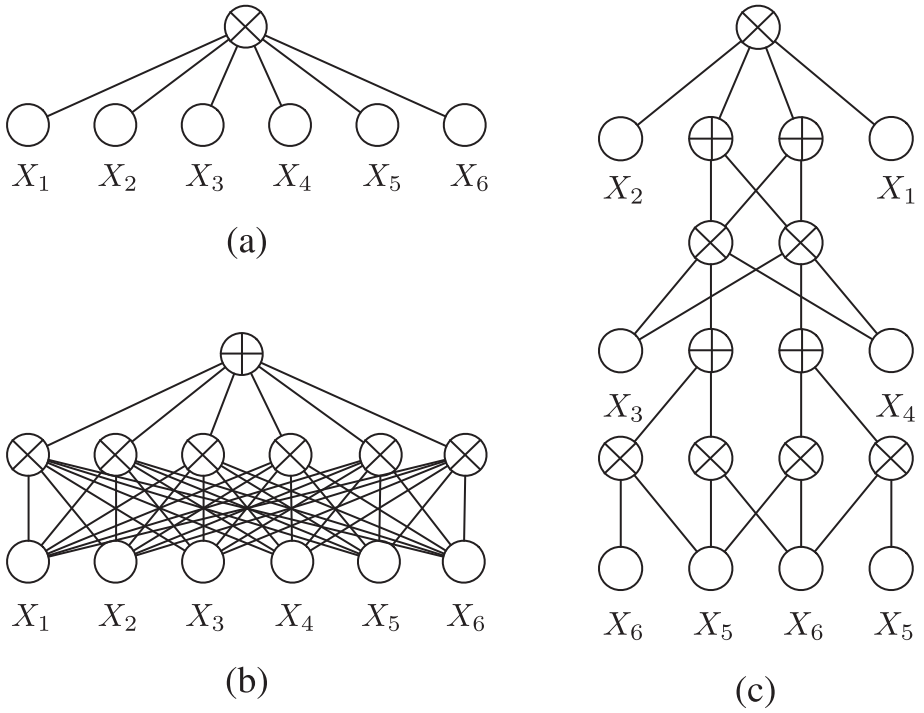

Fig. 1. Examples of SPNs. Sum nodes are graphically denoted as $\bigoplus$ while products as $\bigotimes$. Sum node weights are omitted to avoid clutter. Univariate leaf distributions are labeled by their scope. A naive factorization over 6 random variables is shown in 1a, while a shallow mixture—standing for a point-wise kernel density estimator—in 1b. A deeper architecture, encoding the same mixture in a more compact way is presented in 1c.

properties together imply *validity* i.e. the ability of exactly computing the probability of each possible complete or partial evidence configuration [32, 33].

When the weights of each sum node i in a valid network S sum to one, i.e., $\sum_j w_{ij} = 1$, and distribution at leaves are normalized, then S computes an exact normalized probability for each possible state [28, 33]. W.l.o.g., we assume the SPNs we are considering to be valid and normalized.

In order to compute $S(\boldsymbol{x})$ it is necessary to evaluate the network through a *bottom-up* step. When evaluating a leaf node i, $S_i(\boldsymbol{x}_{|\mathsf{sc}(i)})$ corresponds to the probability of the state $\boldsymbol{x}_{|\mathsf{sc}(i)}$ for the RV associated to its scope, $\mathsf{sc}(i)$: $S_i(\boldsymbol{x}) = \mathsf{p}(\mathsf{sc}(i) = \boldsymbol{x}_{|\mathsf{sc}(i)})$. The value of a product node corresponds to the product of its children values: $S_i(\boldsymbol{x}_{|\mathsf{sc}(i)}) = \prod_{i \to j \in S} S_j(\boldsymbol{x}_{|\mathsf{sc}(j)})$; while, for a sum node its value corresponds to the weighted sum of its children values: $S_i(\boldsymbol{x}_{|\mathsf{sc}(i)}) = \sum_{i \to j \in S} w_{ij} S_j(\boldsymbol{x}_{|\mathsf{sc}(j)})$.

It is possible to prove that all the marginal probabilities, the partition function and even approximate MPE queries and states can be computed in time linear in the *size* of the network—the number of edges in it [14, 28]. Intuitively, *the less the number of edges in a network, the faster the inference*. In practice, tractable inference is achieved when the number of edges is *at most polynomial* in the number of RVs.

While the size of a network implies its inference efficiency, its *depth*, defined as the longest path from the root to a leaf node in networks with strictly interleaving layers of nodes of the same kind[1], determines its *expressive efficiency* [23]. This kind of efficiency relates to the ability of a network to capture more complex distributions than other networks having the same or larger size [3, 23, 42].

Another structural metric of interest is the number of parameters of a network, i.e., the number of sum node weights and leaf parameters—also called *model capacity*—as it influences the model expressiveness. While we are not looking at weight learning per se in this work, it is worth noting that the larger a model capacity, the higher the representation space searchable by optimizing the network weights.

[1] Note that it is always possible to transform an SPN with adjacent nodes of the same type into an equivalent one with alternating types [42].

Up to now, however, the research community has put more effort in designing structure learning algorithms and comparing learned SPNs w.r.t. their *inference accuracy*, i.e., the closeness of the probability distribution estimated by the network to the one that generated the data. For density estimators in general, this is usually done in the terms of the average test set log-likelihood [18].

In this work, following [42], we argue that both a network structure quality metrics—its size, depth and model capacity—and its log-likelihood shall be taken into account to better evaluate the inherent trade-off in density estimation among learning and inference tractability and expressiveness and accuracy.

2.1. Structure Learning

As already stated, we focus our attention to a particular structure learning algorithm for SPNs, LearnSPN [14]. LearnSPN provides a simple, greedy and yet surprisingly effective learning schema that is able to infer both the structure and the parameters of an SPN from the data. Note that, while our approximate approaches are inspired directly by LearnSPN, they can be effectively adapted to other algorithmic variations aiming to learn SPNs, e.g., the ones proposed in [1, 17, 37, 42]. Furthermore, they can also be adapted to other learning scenarios, in which one has to split RVs into sets of (approximately) independent RVs [2, 12, 19, 39], e.g., when eliciting the conditional independencies among RVs for classical PGMs.

LearnSPN performs a greedy top-down structure search in the space of *tree-shaped* SPNs, i.e., networks in which each node has at most one parent. The network structure is built one node at a time by leveraging the probabilistic semantics of an SPN: sum nodes stand for mixtures over sub-populations in the data, while products represent factorizations over independent components.

A high level outline is sketched in Algorithm 1. LearnSPN proceeds by recursively partitioning the training data provided as a matrix consisting of a set $\mathcal{D}$ of rows as i.i.d instances, over $\boldsymbol{X}$, the set of columns, i.e., the RVs. For each call on a submatrix, the algorithm first tries to split the submatrix by columns. This is done by splitting the current set of RVs into different groups such that the RVs in a group are statistically dependent while the groups are independent, i.e., the joint distribution factorizes over the groups.

We denote this procedure as *Greedy Variable Splitting* (GVS). If the GVS procedure fails, that is all RVs are somewhat dependent and it is not meaningful to split them, then LearnSPN switches to split rows. If this is the case, when GVS is designed to find only two split components, we assume the second one to be empty. We will get back to GVS in Section 4.

LearnSPN then tries to aggregate similar submatrix rows (procedure clusterInstances) into a number of clusters. In the original work of [14], an online version of hard Expectation-Maximization (EM) algorithm is employed for such clustering step. Nevertheless, depending on the assumptions on the distribution of $\mathbf{X}$, different clustering algorithms might be employed [17, 26, 27, 42].

Every time a column split succeeds the algorithm adds a product node to the network whose children correspond to partitioned submatrixes. Analogously, after a row clustering step it adds a sum node where children weights represent the proportions of instances falling into the computed clusters (line 11). To avoid a naive factorization of the whole data matrix the algorithm heuristically forces a row clustering at the first algorithm iteration.

Termination happens in two cases: when the current submatrix contains only one column (line 1) or when the number of its rows falls under a certain threshold μ (line 3). In the former, a leaf node, standing for a univariate distribution, is introduced by a maximum likelihood estimation from the submatrix applying a Laplace smoothing parameter α. In the latter, the submatrix RVs are modeled with a naive factorization, i.e., they are considered to be independent and a product node is put over a set of univariate leaf nodes.

As noted in [42], the two splitting procedures depend on each other: the quality of row clusterings is likely to be enhanced by column splits correctly identifying dependent variables and vice versa. As a consequence, while we are proposing to substitute only the GVS procedure, the effects of the introduced approximation will affect also the row clustering step. We will evaluate this effect globally by measuring both the structure quality of the networks learned with our proposed approaches and their likelihood accuracy, in a series of empirical experiments in Section 5.

3. Related Work

Since their introduction in [33], researchers have tackled the task to learn an SPN structure from data in

Algorithm 1 LearnSPN($\mathcal{D}$, $\mathbf{X}$, α, μ, ρ)

Require: a set of samples $\mathcal{D}$ over RVs $\mathbf{X}$; α: Laplace smoothing parameter; μ: minimum number of instances to split; ρ: statistical independence threshold
Ensure: an SPN S encoding a pdf over $\mathbf{X}$ learned from $\mathcal{D}$
1: **If** $|\mathbf{X}| = 1$ **then**
2: $S \leftarrow$ univariateDistribution($\mathcal{D}$, $\mathbf{X}$, α)
3: **else if** $|\mathcal{D}| < \mu$ **then**
4: $S \leftarrow$ naiveFactorization($\mathcal{D}$, $\mathbf{X}$, α)
5: **else**
6: $\{\mathbf{X}_{d_1}, \mathbf{X}_{d_2}\} \leftarrow$ GVS($\mathcal{D}$, $\mathbf{X}$, ρ)
7: **if** $|\mathbf{X}_{\mathbf{d_2}}| > \mathbf{0}$ **then**
8: $S \leftarrow \prod_{j=1}^{2}$ LearnSPN($\mathcal{D}$, $\mathbf{X}_{\mathbf{d_j}}$, α, μ, ρ)
9: **else**
10: $\{\mathcal{D}_i\}_{i=1}^{R} \leftarrow$ clusterInstances($\mathcal{D}$, $\mathbf{X}$)
11: $S \leftarrow \sum_{i=1}^{R} \frac{|\mathcal{D}_i|}{|\mathcal{D}|}$ LearnSPN($\mathcal{D}_i$, $\mathbf{X}$, α, μ, ρ)
12: **return** S

several fashions. The first attempt has been presented in [8]. This top-down algorithm operates differently from LearnSPN in that it clusters instances once, at the beginning. Then, it proceeds splitting RVs without exploiting local independencies, but by directly running K-means on the columns of the data matrix. During learning it builds an SPN via a *region graph*, an abstract representation of the network structure of an SPN: in it region nodes represent non-empty sets of RVs while partition nodes decompose regions into non-overlapping sets. As the resulting region graph structure is translated into an SPN, network parameters are estimated separately with the EM algorithm.

Also in [29] a region graph is first built and then converted into an SPN. However, regions are discovered bottom-up and recursively merged into coarser grain partitions via a Bayesian-Dirichlet test. Sum nodes and their parameters are learned by maximizing the mutual information among some greedily selected RVs through an approximation of the Information Bottleneck criterion.

One of the state-of-the-art SPN learner is the ID-SPN algorithm [37]. ID-SPN learns Sum-Products of Arithmetic Circuits (SPACs) models—hybrid SPNs consisting of sum and product nodes as inner nodes, and ACs as leaf nodes. It exploits both indirect and direct interactions among RVs, unifying works on SPN structure learning and on learning tractable Markov networks [21]. Its high accuracy is due to the learning process that directly maximizes the data likelihood. However, ID-SPN is very slow in practice and requires the user to fine tune several hyperparameters needed to learn the leaf ACs.

In [30], *selective SPNs* are introduced as a less-expressive class of SPNs in which at most one sum node child can have a non-zero output, for each possible input. By exploiting determinism [7], it is possible to compute in closed form the globally optimal maximum likelihood parameters for selective SPNs. This allows the authors to exploit stochastic local search techniques to learn their structure from data.

The work of [35] goes in a different direction in order to learn a graph SPN. It starts by learning a tree SPN and continues by merging common sub-structures recursively by approximating the Kullback-Leibler divergence among them. While potentially resulting in more expressive networks, the accuracy gain of merged graph SPNs is minimal.

In [40], a Bayesian nonparametric approach to learn infinite Sum-Product Trees (SPTs) is proposed. There, the authors introduce an approximate inference scheme via Gibbs sampling to deal with the posterior distribution over the SPN *induced trees* [45, 46]. The heuristic approaches deriving from LearnSPN variants, even if less principled and more greedy, scale better and at the same time deliver effective probability estimates.

An ad-hoc SPN structure learning algorithm for sequence data is conceived in [24]. The resulting structure, named Dynamic SPNs (DSPNs) similarly to Dynamic Bayesian Networks, can be learned via an adaptation of LearnSPN to sequence data via template network blocks representing interactions among RVs at a certain step in the sequence. Such a learning scheme is flexible enough to model data sequences of any length.

Lastly, online structure learning for SPNs has been tackled in [16]. The idea is to update both the structure and the parameters of an SPN keeping completeness and decomposability after each new sample is observed. Basically, correlation between RVs modeled by a product node different children is checked whether still holds. If the overall structure need to be accommodated for newly found correlations, either a multivariate leaf node or a mixture of two components over the considered RVs are introduced.

4. Variable splitting

We now describe in detail the *Greedy Variable Splitting* (GVS) procedure applied by LearnSPN to discover groups of dependent RVs, in order to introduce later our variants.

To exactly determine a partitioning of independent RVs, one could recur to Queyranne's algorithm to retrieve two subsets sharing minimum empirical

mutual information (MI). However, this will scale in cubic time w.r.t. the number of RVs considered [34]. Instead, as the name suggests, GVS proceeds in a greedy way, lowering the time complexity to be quadratic in the worst case (see Algorithm 2). By picking a RV at random (line 1), GVS tries to discover connected components in a graph of dependencies by introducing one edge among two RVs if they are dependent.

While pairwise MI could be used to test the independence among the two considered RVs, a more sophisticated statistical test, the G-Test, is applied in the original formulation of GVS in [14]. In both cases, two RVs X_i, X_j are assumed to be independent if the statistical test result falls under a used defined threshold ρ. In the case of a G-Test over RVs X_i, X_j over a sample dataset $\mathcal{D}$, one computes:

$$\mathsf{GTest}(\mathcal{D}, X_i, X_j) = 2 \sum_{x_i \sim X_i} \sum_{x_j \sim X_j} c(x_i, x_j) \cdot \log \frac{c(x_i, x_j) \cdot |\mathcal{D}|}{c(x_i) c(x_j)}. \quad (1)$$

where $c(\cdot)$ (resp. $c(\cdot, \cdot)$) represents the empirical marginal (resp. joint) probability of observing some configuration of RVs in $\mathcal{D}$.

At the first iteration if the algorithm does not find any dependency it returns the first considered node assuming it to be independent from all the other RVs. The opposite case is when the algorithm finds a connected component comprising all the RVs currently considered by LearnSPN.

To understand the worst case time complexity of GVS consider the following case. At most, the algorithm needs to perform $(n^2 + n)/2$ G-tests with $n = |\boldsymbol{X}|$ by looking at all possible pairwise cases (lines 3-12). If we assume the complexity of performing a G-Test to be linear in the number of samples ($m = |\mathcal{D}|$), then the worst case complexity of GVS is $O(n^2 m)$. As a last remark, consider that the alternative splitting procedures presented in other SPN learning algorithms, e.g. [26, 37], even if employing different pairwise statistical independence tests, still require a quadratic number of comparisons, in the worst case.

4.1. Stochastic Variable Splitting

The intuition behind our first proposed alternative method for splitting variables is the same behind *random subspace techniques* that have been successfully employed for building ensembles for predictive tasks [5] but also for density estimation [10]. We name this method *Random Greedy Variable Splitting* (RGVS) and we sketch it in Algorithm 3.

Algorithm 2 GVS($\mathcal{D}$, $\mathbf{X}$, ρ)

Require: set of samples $\mathcal{D}$ over RVs $\mathbf{X}$; ρ: a statistical independence threshold
Ensure: a split of two groups of dependent variables $\{\mathbf{X_{d_1}}, \mathbf{X_{d_2}}\}$,
1: $\mathbf{X_{d_2}} \leftarrow \mathbf{X}$, $X_0 \leftarrow$ drawVariableAtRandom($\mathbf{X}$)
2: $\mathbf{X} \leftarrow \mathbf{X} \setminus \{\mathbf{X_0}\}$, $\mathbf{P} \leftarrow \{\mathbf{X_0}\}$, $\mathbf{X_{d_1}} \leftarrow \{\mathbf{X_0}\}$, $\mathbf{R} \leftarrow \emptyset$
3: **repeat**
4: $X_p \leftarrow \mathsf{pop}(\mathbf{P})$ $\quad \triangleright\ \mathbf{P} \leftarrow \mathbf{P} \setminus \{\mathbf{X_p}\}$
5: **foreach** $X_k \in \mathbf{X}$ **do**
6: **if** not GTest($\mathcal{D}$, X_p, X_k) $< \rho$ **then**
7: $\mathbf{R} \leftarrow \mathbf{R} \cup \{\mathbf{X_k}\}$
8: $\mathbf{X_{d_1}} \leftarrow \mathbf{X_{d_1}} \cup \{\mathbf{X_k}\}$
9: $\mathbf{P} \leftarrow \mathbf{P} \cup \{\mathbf{X_k}\}$
10: **for each** $X_j \in \mathbf{R}$ **do**
11: $\mathbf{X} \leftarrow \mathbf{X} \setminus \{\mathbf{X_j}\}$
12: **until** $|\mathbf{P}| > 0 \wedge |\mathbf{X}| > 0$
13: **return** $\{\mathbf{X_{d_1}}, \mathbf{X_{d_2}} \setminus \mathbf{X_{d_1}}\}$

RGVS randomly selects a subset $\mathbf{R}$ of k RVs from $\boldsymbol{X}$ and then applies GVS only on this subset. Since we are not applying it in an ensemble scenario, but to learn a single model, we do expect this model inference accuracy to drop since the independencies discovered by RGVS can only be a subset of those discovered by GVS in a single call. Nevertheless, how much the accuracy degrades depends to the ability of RGVS to recover the dependencies missed during one call in the following calls that LearnSPN performs on the same RVs (see Section 2.1). At the same time, we aim to reduce the learning times as well as the inference times, by obtaining smaller networks.

If GVS fails on the set of k RVs $\boldsymbol{R}$, then even RGVS will fail assuming dependent the remaining variables in $\boldsymbol{S} = \boldsymbol{X} \setminus \boldsymbol{R}$ (lines 7-8). Otherwise, if the second component returned from GVS is not empty then RGVS will make another stochastic decision about where to put the remaining variables in $\boldsymbol{S}$ (lines 10-13). Consequently, the worst case complexity of RGVS depends on the choice of the parameter k. If k is chosen to be $\sqrt{n}$, then[2] we end up scaling GVS as if it were linear, since $O((\sqrt{n})^2 m) = O(nm)$. Otherwise, depending on a choice of $k < n$ the complexity of a single call varies but remains sub-quadratic.

However, note that determining the resulting complexity of a whole run of LearnSPN equipped with RGVS instead of GVS is not immediate. Consider, for example the case in which k is chosen as a small fraction of the RVs in $\boldsymbol{X}$, then, it might happen that

[2] We set k to be at least 2 when $n = |\boldsymbol{X}| < 4$.

Algorithm 3 RGVS($\mathcal{D}$, $\boldsymbol{X}$, ρ)

Require: set of samples $\mathcal{D}$ over RVs $\boldsymbol{X}$; ρ: a statistical independence threshold
Ensure: a split of two groups of dependent variables $\{\boldsymbol{X}_{d_1}, \boldsymbol{X}_{d_2}\}$
1: $\boldsymbol{X}_{d_1} \leftarrow \emptyset$, $\boldsymbol{X}_{d_2} \leftarrow \emptyset$, $n \leftarrow |\boldsymbol{X}|$, $k \leftarrow max(\lfloor \sqrt{n} \rfloor, 2)$
2: **If** $k = n$ **then**
3: **return** GVS($\mathcal{D}$, $\boldsymbol{X}$, ρ)
4: $\boldsymbol{R} \leftarrow$ randomSubspace($\boldsymbol{X}$, k)
5: $\boldsymbol{S} \leftarrow \boldsymbol{X} \setminus \boldsymbol{R}$
6: $\{\boldsymbol{X}_{d_1}, \boldsymbol{X}_{d_2}\} \leftarrow$ GVS($\mathcal{D}$, $\boldsymbol{R}$, ρ)
7: **If** $\boldsymbol{X}_{d_2} = \emptyset$ **then**
8: **return** $\{\boldsymbol{X}_{d_1} \cup \boldsymbol{S}, \emptyset\}$
9: $r \leftarrow$ Bernoulli(0.5)
10: **If** $r = 0$
11: **return** $\{\boldsymbol{X}_{d_1} \cup \boldsymbol{S}, \boldsymbol{X}_{d_2}\}$
12: **else**
13: **return** $\{\boldsymbol{X}_{d_1}, \boldsymbol{X}_{d_2} \cup \boldsymbol{S}\}$

Algorithm 4 WRGVS($\mathcal{D}$, $\boldsymbol{X}$, ρ)

Require: set of samples $\mathcal{D}$ over RVs $\boldsymbol{X}$; ρ: a statistical independence threshold
Ensure: a split of two groups of dependent variables $\{\boldsymbol{X}_{d_1}, \boldsymbol{X}_{d_2}\}$
1: $\boldsymbol{X}_{d_1} \leftarrow \emptyset$, $\boldsymbol{X}_{d_2} \leftarrow \emptyset$, $n \leftarrow |\boldsymbol{X}|$, $k \leftarrow max(\lfloor \sqrt{n} \rfloor, 2)$
2: **If** $k = n$ **then**
3: **return** GVS($\mathcal{D}$, $\boldsymbol{X}$, ρ)
4: $\boldsymbol{R} \leftarrow$ randomSubspace($\boldsymbol{X}$, k)
5: $\boldsymbol{S} \leftarrow \boldsymbol{X} \setminus \boldsymbol{R}$
6: $\{\boldsymbol{X}_{d_1}, \boldsymbol{X}_{d_2}\} \leftarrow$ GVS($\mathcal{D}$, $\boldsymbol{R}$, ρ)
7: **if** $\boldsymbol{X}_{d_2} = \emptyset$ **then**
8: **return** $\{\boldsymbol{X}_{d_1} \cup \boldsymbol{S}, \emptyset\}$
9: $X_j \leftarrow$ drawVariableAtRandom($\boldsymbol{X}_{d_1}$)
10: $X_k \leftarrow$ drawVariableAtRandom($\boldsymbol{X}_{d_2}$)
11: **foreach** $X_i \in \mathbf{S}$ **do**
12: **if** GTest($\mathcal{D}$, X_i, X_j) $\geq$ GTest($\mathcal{D}$, X_i, X_k) **then**
13: $\boldsymbol{X}_{d_1} \leftarrow \boldsymbol{X}_{d_1} \cup X_i$
14: else
15: $\boldsymbol{X}_{d_2} \leftarrow \boldsymbol{X}_{d_2} \cup X_i$
16: **return** $\{\boldsymbol{X}_{d_1}, \boldsymbol{X}_{d_2}\}$

LearnSPN calls the splitting routines a larger number of times since in each call only a few or no RVs are considered independent from the rest. The result is that the network built may be larger, hence the learning time increased, w.r.t. a network learned with a larger k. In Section 5 we empirically evaluate the sensitivity of RGVS to the choice of k w.r.t. the model inference accuracy and its learning time.

An approach to mitigate the randomness of assigning the remaining RVs not selected by RGVS is implemented in WRGVS (Algorithm 4) where W stands for *wiser*. This method, firstly extracts one RV selected at random from each subset found by GVS and uses it as a representative of that subset. Then, each remaining RV is added to the subset whose representative variable shows the stronger dependency according to a G-test. With this approach, we expect to improve the accuracy of learned networks w.r.t. the ones learned by RGVS while keeping, at the same time, the complexity linear in the number of variables.

4.2. Entropy-based Variable Splitting

The second proposed variable splitting method, named *Entropy-Based Variable Splitting* (EBVS), is based on the concept of entropy as taken in information theory[3], whose pseudocode is listed in Algorithm 5.

EBVS performs a linear scan of the RVs in $\boldsymbol{X}$, grouping in one split those RVs whose entropy value falls under a user defined threshold η. The rationale behind this idea is that a RV with exactly zero entropy is independent from all other RVs in $\boldsymbol{X}$.

To understand why this is true, consider computing MI between RV X and any RV $X_* \in \boldsymbol{X}$, denoted as $\mathsf{MI}(X; X_*)$ then we have that $\mathsf{MI}(X; X_*) = H(X) - H(X|X_*)$ where $H(\cdot)$ denotes marginal entropy and $H(X|X_*)$ conditional entropy respectively [22]. But since $H(X) = 0$ by hypothesis and it must hold that $H(X) \geq H(X|X_*)$, then it turns out that $\mathsf{MI}(X; X_*) = 0$, hence X must be independent to all other RVs in X.

Since the empirical estimator for the entropy is rarely zero on real data, we apply thresholding to increase the method robustness and regularization. Moreover, we apply Laplacian smoothing to compute the probabilities involved in the entropy computation[4].

While η can be heuristically determined by the user by performing cross-validation, having a global threshold can impose a too strict inductive bias. Therefore, we introduce another algorithmic variant, called EBVS-AE, where AE stands for "Adaptive Entropy", in which η is adaptively computed based on the size (the number of instances) of the current submatrix processed by LearnSPN. Specifically, given a global hyperparameter η, we compute the actual value of η_t employed in the t-th call to EBVS-AE by LearnSPN proportionally to the number of samples $|\mathcal{D}_t|$ (see Section 2.1) processed w.r.t. the number of samples in the whole dataset $|\mathcal{D}_t|$, i.e. $\eta_t = \eta \frac{|\mathcal{D}_t|}{|\mathcal{D}|}$.

[3] For a discrete RV X, having values in $\mathcal{X}$, we consider its discrete entropy as $H(X) = -\sum_{x \in \mathcal{X}} p(x) \log(p(x))$.

[4] For convenience, and to avoid the addition of a new hyperparameter, the Laplacian smoothing parameter value will be the same of the hyperparameter α of LearnSPN, used to smooth the univariate distributions at leaves. Note that the number of hyperparameters of our variant of LearnSPN does not change: now η substitutes the ρ which is not needed anymore.

Algorithm 5 EBVS($\mathcal{D}$, $\mathbf{X}$, η, α)

Require: set of samples $\mathcal{D}$ over RVs $\boldsymbol{X}$; η: a threshold for entropies, α: Laplacian smoothing parameter
Ensure: a split of two groups of dependent variables $\{\boldsymbol{X}_{d_1}, \boldsymbol{X}_{d_2}\}$
$\boldsymbol{X}_{d_1} \leftarrow \emptyset, \boldsymbol{X}_{d_2} \leftarrow \emptyset$
$//\eta = \eta \frac{|\mathcal{D}|}{|\mathcal{D}_{\text{full}}|}$ if EBVS-AE
foreach $X_i \in \mathbf{X}$ **do**
  if computeEntropy($\mathcal{D}, X_i, \alpha$) $< \eta$ **then**
    $\boldsymbol{X}_{d_1} \leftarrow \boldsymbol{X}_{d_1} \cup \{X_i\}$
  else
    $\boldsymbol{X}_{d_2} \leftarrow \boldsymbol{X}_{d_2} \cup \{X_i\}$
if $|\boldsymbol{X}_{d_1}| = 0$ **then**
  return $\{\boldsymbol{X}_{d_2}, \boldsymbol{X}_{d_1}\}$
else
  return $\{\boldsymbol{X}_{d_1}, \boldsymbol{X}_{d_2}\}$

Both EBVS and EBVS-AE involve the computation of the entropy for each RV in $\boldsymbol{X}$, hence their complexity is simply *linear* in both the number of RVs and samples, i.e. $O(nm)$.

While both the proposed variants drastically reduce the time complexity of the RV splitting procedure, their effect on the overall learning algorithm and the consequent degradation of model accuracy need to be evaluated empirically, see Section 5. Nevertheless, it is reasonable to expect the structures learned by these methods to be quite different from those learned by GVS and RGVS, since the procedure to test for statistical independence relies on a single RV metric for the former and for pairwise metrics for the latter.

4.3. Random Sampling based Variable Splitting

We now discuss another possible alternative to speed-up the RV splitting step of LearnSPN, by performing *random sampling* over $\mathcal{D}$, that is, performing a G-Test only on a fraction of the samples considered by the algorithm. We denote this version as RSBVS—random sampling based variable splitting—in the following.

Specifically, approximation proposed in RSBVS randomly draws without replacement an amount of samples equal to $\beta|\mathcal{D}|$, where $\beta \in (0, 1]$, and then computes the G-test on this subset of samples. The assumption here is that this sample subset, by sharing the same distribution of the whole data matrix slice, would still allow to detect the dependencies among the RVs considered. Of course, this assumption does not hold in general for small fractions of β.

Since we compute the counts on a subset of $\mathcal{D}$, statistics should be proportional to the considered amount of samples. Given that, we need to fix both the computation of the G-test for two variables and the threshold accordingly. Therefore, we update the computation of the empirical co-occurences $c(x_i, x_j)$ as $c(x_i, x_j)/\beta$. For other quantities in Equation 1 we make a similar fix, leading to the following formula:

$$\mathsf{GTest}(\mathcal{D}, X_i, X_j) = 2 \sum_{x_i \sim X_i} \sum_{x_j \sim X_j} \frac{c(x_i, x_j)}{\beta} \cdot \log \frac{c(x_i, x_j) \cdot |\mathcal{D}|}{c(x_i)c(x_j)}. \quad (2)$$

It is clear that the complexity of this last introduced variable splitting procedure is still quadratic in the number of random variables, but we are confident that taking a percentage of the samples considerably reduces the learning times and possibly reduces inference times since the learned networks could be small in size.

5. Experiments

We empirically evaluated all the proposed methods (RGVS, WRGVS, EBVS, EBVS-AE, RSBVS) as alternative methods for variable splitting by plugging them in LearnSPN-b [42], a simplified variant of LearnSPN. We implemented them in Python[5], in the freely available version of LearnSPN-b [42] and we refer to GVS in the following as the original version of LearnSPN-b employing Algorithm 2. LearnSPN-b only performs binary splits even for the row clustering step while learning an SPN. By doing this, it slows down the greedy search process avoiding too complex structural choices at early stages, therefore, implementing a form of the "simplicity bias".

With the following experimental evaluation we aim to answer the following research questions: **(Q1)** how does RGVS (fixing the parameter $k = \sqrt{n}$) compare to GVS in terms of inference and learning time and model accuracy? **(Q2)** is WRGVS able to learn more accurate and compact networks in a faster way than RGVS (fixing the parameter $k = \sqrt{n}$)? **(Q3)** how does RSBVS compare to GVS in terms of inference and learning time and model accuracy? **(Q4)** how does EBVS compare to GVS in terms of inference and learning time and model accuracy? **(Q5)** is EBVS-AE able to learn more accurate and compact networks in a faster way than EBVS?

To answer the aforementioned questions, we evaluated the proposed methods on 18 datasets where 17 of them are employed as standard benchmarks to

[5] Code is available at https://github.com/fabriziov/alt-vs-spyn.

compare TPMs, being introduced in [20] and [15]. They comprise binarized data coming from tasks such as frequent item mining, recommendation and classification. Additionally, we prepared a binarized[6] version of the "EUR-Lex" dataset [25], in order to better assess whether and to what extent our approximate methods scale on datasets with a huge number of random variables. In Table 1 are reported the datasets used in our experiments and their statistics.

For each method, we selected the best parameter configurations based on the average validation log-likelihood scores, then evaluated such models on the test sets. We performed an exhaustive grid search for $\rho \in \{5, 10, 15, 20\}$, $\mu \in \{10, 50, 100, 500\}$ and $\alpha \in \{0.1, 0.2, 1, 2\}$, leaving all other parameters to the default values. For EBVS and EBVS-AE the grid search involved, instead of ρ, also $\eta \in \{0.05, 0.1, 0.3, 0.5\}$. For the "EUR-Lex" dataset we had to considerably restrict the hpyerparameter space of the grid search due to limited amount of computational resources. For this dataset, instead, we fixed $\alpha = 0.1$, $\mu \in \{1500, 3000\}$, $\rho \in \{20, 30\}$ and $\eta \in \{0.3, 0.5\}$ for the methods based on entropy. In order to mitigate the intrinsic randomness of the approximate methods based on stochastic decisions, namely RGVS, WRGVS and RSBVS, we averaged all the results computed by running the algorithms with five different seeds, reporting both mean and standard deviation of learning time and accuracy.

All the experiments have been run on an 8-core AMD Opteron 63XX @ 2.3 GHz with 16GB of RAM and Ubuntu 14.04.4 LTS, kernel 3.13.0-45.

5.1. (Q1) Evaluating *RGVS*

Regarding RGVS, as expected, making the variable splitting method partially random fosters the learning speed of the models. Table 2 reports the global learning times for the best validation networks (ranking positions are between parentheses and average ranking position of each method are shown at the bottom of the table). From it is visible how LearnSPN-b equipped with RGVS takes less time to grow a full structure than GVS, requiring in some cases less than half the time. The improvement in learning times is even more noticeable on datasets with a high number of RVs such as Reuters-52, BBC, Ad and very appreciable, in terms of real saved time, on EUR-Lex. To have a clearer vision of the gain in learning times, in Tables 3 and 4 we show the learning time ratio between each introduced method and GVS taken as baseline and the number of victories on learning time respectively. Table 9 reports the structure quality of the learned networks in terms of their number of edges (size), number of layers (depth) and parameters (model capacity). See Section 2 for how these values influence both inference and learning. It is evident how RGVS is able to learn very compact models, speeding up inference times. However, this is done trading-off accuracy. In Table 5 average test log-likelihoods are reported for all methods on all datasets (ranking positions are in parentheses). There, accuracy degrades on all datasets–significantly on Pumsb-star, DNA, WebKB, Reuters-52, Ad and EUR-Lex– being comparable to other methods on some datasets such as Retail, Jester, Kosarek and MSWeb. In Table 6 the number of victories on accuracy for all methods are reported.

Additionally, to better understand the behavior of RGVS, we evaluate how changing the proportion of RVs involved in GSV affects accuracy and learning times. We assess this by determining k as the varying proportion of the RVs actually involved in the statistical test, making it range in the 10%, 30%, 50%, 70%, 90% of all of them, for the best configurations found on the validation sets. From Table 7, one can see that generally, when the proportion of involved RVs in the G-test is between 30% and 70%, we obtain faster learning times and less accurate models. The reason behind this is that when this proportion is either small or close to 100%, LearnSPN-b just makes much more calls to the RGVS, evaluating fewer RVs every time. Given the random nature of the method, we also present the standard deviation of both learning time and log-likelihood. It is possible to observe that random fluctuations are slightly more increased on datasets with a high number of variables. Concluding, we can answer **Q1** by stating that RGVS is able to learn more compact models in less time than GVS, yet compromising on inference accuracy.

5.2. (Q2) Evaluating *WRGVS*

An additional stochastic method introduced in the previous Section is WRGVS. This method attempts to reduce the error of stochastic decisions made by RGVS.

Here we evaluate whether it improves the accuracy w.r.t. the one gained with networks learned with RGVS. From our results in Table 5, it turns out that

[6] The dataset is composed by 5000 continuous features and 3993 binary labels, we binary discretized the continuous features w.r.t. the mode of each feature.

Table 1
Datasets used and their statistics

	$\|V\|$	$\|T_{train}\|$	$\|T_{val}\|$	$\|T_{test}\|$		$\|V\|$	$\|T_{train}\|$	$\|T_{val}\|$	$\|T_{test}\|$
NLTCS	16	16181	2157	3236	Kosarek	190	33375	4450	6675
Plants	69	17412	2321	3482	MSWeb	294	29441	3270	5000
Audio	100	15000	2000	3000	Book	500	8700	1159	1739
Jester	100	9000	1000	4116	EachMovie	500	4525	1002	591
Netflix	100	15000	2000	3000	WebKB	839	2803	558	838
Accidents	111	12758	1700	2551	Reuters-52	889	6532	1028	1540
Retail	135	22041	2938	4408	BBC	1058	1670	225	330
Pumsb-star	163	12262	1635	2452	Ad	1556	2461	327	491
DNA	180	1600	400	1186	EUR-Lex	8993	15539	1500	2309

Table 2
Times (in seconds) taken to learn the best models on each dataset with EBVS, EBVS-AE, RGVS, WRGVS and GVS (ranking positions in parentheses)

	learning time					
	EBVS	EBVS-AE	RGVS	WRGVS	RSBVS 50%	GVS
NLTCS	97 (6)	27 (4)	2±0.2 (1)	3±2.1 (2)	31 (5)	4 (3)
Plants	218 (6)	115 (4)	11±3.5 (1)	13±10.6 (2)	171 (5)	30 (3)
Audio	63 (6)	42 (3)	19±6.7 (1)	19±13.2 (2)	59 (5)	45 (4)
Jester	31 (4)	27 (3)	10±2.6 (1)	11±8.3 (2)	33 (6)	32 (5)
Netflix	57 (5)	46 (3)	33±12.2 (2)	28±19.1 (1)	49 (4)	68 (6)
Accidents	463 (6)	39 (4)	11±4.6 (1)	12±10.9 (2)	196 (5)	32 (3)
Retail	55 (6)	55 (5)	6±1.0 (1)	6±4.3 (2)	39 (4)	16 (3)
Pumsb-star	176 (5)	81 (4)	13±5.0 (2)	11±8.1 (1)	437 (6)	38 (3)
DNA	6 (4)	5 (3)	1±0.6 (1)	2±1.3 (2)	15 (6)	10 (5)
Kosarek	193 (6)	152 (5)	22±3.9 (2)	20±10.3 (1)	152 (4)	52 (3)
MSWeb	449 (6)	204 (5)	25±5.0 (1)	29±11.6 (2)	140 (4)	118 (3)
Book	105 (5)	65 (3)	24±9.4 (2)	24±12.2 (1)	112 (6)	92 (4)
EachMovie	58 (4)	51 (3)	16±5.5 (1)	17±10.4 (2)	64 (5)	82 (6)
WebKB	76 (4)	24 (3)	17±6.9 (1)	18±10.3 (2)	96 (6)	77 (5)
Reuters-52	185 (4)	90 (3)	40±11.4 (1)	43±23.4 (2)	210 (5)	341 (6)
BBC	37 (4)	17 (3)	10±4.4 (1)	12±6.3 (2)	73 (5)	253 (6)
Ad	183 (4)	165 (3)	25±15.3 (1)	33±19.4 (2)	228 (5)	350 (6)
EUR-Lex	2146 (1)	2641 (2)	5906±319.7 (3)	6847±661.2 (4)	15025 (5)	26090 (6)
AVG	4.8	3.5	1.3	1.9	5.1	4.4

Table 3
Ratio of times (in seconds) taken to learn the best models on each dataset with EBVS, EBVS-AE, RGVS, WRGVS w.r.t. GVS (used as baseline)

	learning time ratio					
	EBVS	EBVS-AE	RGVS	WRGVS	RSBVS 50%	GVS (seconds)
NLTCS	23.53	6.65	0.56	0.66	7.63	4
Plants	7.23	3.81	0.38	0.45	5.69	30
Audio	1.41	0.94	0.41	0.42	1.32	45
Jester	0.99	0.86	0.33	0.33	1.05	32
Netflix	0.83	0.68	0.49	0.42	0.71	68
Accidents	14.29	1.20	0.34	0.36	6.04	32
Retail	3.40	3.38	0.36	0.38	2.39	16
Pumsb-star	4.59	2.10	0.33	0.28	11.37	38
DNA	0.55	0.52	0.14	0.18	1.50	10
Kosarek	3.74	2.95	0.42	0.39	2.94	52
MSWeb	3.81	1.73	0.21	0.25	1.18	118
Book	1.15	0.70	0.26	0.26	1.21	92
EachMovie	0.71	0.62	0.20	0.21	0.79	82
WebKB	1.00	0.31	0.23	0.23	1.26	77
Reuters-52	0.54	0.26	0.12	0.13	0.62	341
BBC	0.15	0.07	0.04	0.05	0.29	253
Ad	0.52	0.47	0.07	0.09	0.65	350
EUR-Lex	0.08	0.10	0.23	0.26	0.58	26090

Table 4
Number of victories on learning time for the algorithms on the rows compared to those on columns

	GVS	EBVS	EBVS-AE	RGVS	WRGVS	RSBVS 50%
GVS	-	9	7	0	0	12
EBVS	9	-	1	1	1	10
EBVS-AE	11	17	-	1	1	15
RGVS	18	17	17	-	14	18
WRGVS	18	17	17	4	-	18
RSBVS 50%	6	8	3	0	0	-

Table 5
Average test log-likelihood of the best models on each dataset of EBVS, EBVS-AE, RGVS, WRGVS, RSBVS 50% and GVS (ranking positions in parentheses)

	log-likelihood					
	EBVS	EBVS-AE	RGVS	WRGVS	RSBVS 50%	GVS
NLTCS	**–6.051** (1)	**–6.046** (1)	–6.370±0.5 (4)	–6.125±0.3 (3)	–6.083 (2)	**–6.040** (1)
Plants	–12.890 (2)	**–12.855** (1)	–16.784±1.1 (5)	–14.164±1.1 (4)	–13.703 (3)	**–12.880** (1)
Audio	–40.763 (2)	**–40.632** (1)	–41.935±0.8 (5)	–41.493±1.1 (3)	–41.566 (4)	**–40.698** (1)
Jester	–53.897 (2)	**–53.528** (1)	–54.968±1.0 (5)	–54.637±1.2 (4)	–54.322 (3)	–53.919 (2)
Netflix	–58.234 (2)	**–58.019** (1)	–59.842±1.2 (6)	–59.082±1.3 (4)	–59.525 (5)	–58.436 (3)
Accidents	–35.336 (4)	–35.635 (5)	–40.232±1.5 (6)	–35.058±1.8 (3)	–32.270 (2)	**–29.002** (1)
Retail	–11.245 (3)	–11.198 (3)	–11.336±0.4 (5)	–11.289±0.4 (4)	–11.027 (2)	**–10.969** (1)
Pumsb–star	–29.235 (3)	–29.486 (4)	–42.423±2.6 (6)	–30.931±2.2 (5)	–27.558 (2)	**–23.282** (1)
DNA	–97.876 (3)	–97.764 (3)	–99.266±0.7 (5)	–98.498±1.0 (4)	–83.493 (2)	**–81.931** (1)
Kosarek	–11.032 (2)	–11.033 (2)	–11.490±0.2 (5)	–11.165±0.4 (3)	–11.175 (4)	**–10.724** (1)
MSWeb	–10.100 (3)	–10.042 (2)	–11.001±0.3 (6)	–10.901±0.4 (5)	–10.352 (4)	**–9.860** (1)
Book	–35.416 (3)	–35.395 (2)	–35.665±0.3 (4)	–35.506±0.5 (4)	–36.348 (4)	**–34.298** (1)
EachMovie	–51.753 (1)	–52.232 (2)	–64.456±1.5 (5)	–57.039±2.8 (4)	–55.205 (3)	**–51.512** (1)
WebKB	–157.941 (2)	–158.202 (2)	–167.547±1.9 (5)	–166.193±2.5 (4)	–163.314 (3)	**–154.909** (1)
Reuters–52	–86.443 (3)	–85.949 (2)	–97.271±2.2 (6)	–95.905±2.5 (5)	–92.061 (4)	**–84.090** (1)
BBC	**–250.883** (1)	**–250.783** (1)	–269.029±1.3 (4)	–267.051±2.2 (3)	–266.339 (2)	**–248.585** (1)
Ad	–20.682 (2)	–22.379 (3)	–57.545±3.2 (6)	–36.607±7.0 (5)	–33.252 (4)	**–15.792** (1)
EUR–Lex	**–89.132** (1)	–91.732 (2)	–103.080±6.5 (5)	–101.357±5.7 (4)	–102.040 (4)	–94.085 (3)
AVG	2.6	2.4	5.9	4.7	3.8	1.5

Table 6
Number of statistically significant victories (numerator) and ties (denominator) on accuracy (Wilcoxon signed rank test, p-value = 0.05) for the algorithms on the rows compared to those on columns

	GVS	EBVS	EBVS-AE	RGVS	WRGVS	RSBVS 50%
GVS	-	12 / 4	11 / 4	18 / 0	18 / 0	18 / 0
EBVS	2 / 4	-	5 / 6	18 / 0	18 / 0	18 / 0
EBVS-AE	3 / 4	7 / 6	-	18 / 0	18 / 0	18 / 0
RGVS	0 / 0	0 / 0	0 / 0	-	0 / 1	0 / 1
WRGVS	0 / 0	0 / 0	0 / 0	17 / 1	-	3 / 2
RSBVS 50%	0 / 0	0 / 0	0 / 0	17 / 1	13 / 2	-

WRGVS is significantly more accurate than RGVS on all datasets except Book. The improvement in accuracy is highly remarkable on Accidents, Pumsb-star, EachMovie and Ad having more than 100 variables. Regarding learning times, WRGVS needs, reasonably, a relatively small fraction of time more than RGVS even on datasets where the improvement in accuracy is very high. As reported in Table 9, networks learned with WRGVS are in general bigger than the ones learned with RGVS but smaller than the ones learned with the original procedure GVS. However, RGVS is capable to learn networks with fewer parameters, therefore resorting less frequently to row clustering, than WRGVS on datasets with a high number of variables (Jester, DNA, Kosarek, MSWeb, Book, EachMovie, Reuters-52, BBC, Ad) while WRGS tends to be more efficient than RGVS in this sense on datasets with a smaller number of variables (NLTCS, Plants, Audio, Netflix, Accidents, Retail, Pumsb-star and WebKB). To wrap-up, WRGVS needs a fraction of time more than RGVS but it is still faster than GVS, significantly improving

Table 7
Learning times in seconds and average test log-likelihoods (with standard deviation) for the best validation models for RGVS, varying the amount of RVs involved in GVS

	learning time					log-likelihood				
	10%	30%	50%	70%	90%	10%	30%	50%	70%	90%
NLTCS	5±2.4	4±0.4	4±0.6	4±0.4	4±0.8	–6.229±0.35	–6.356±0.4	–6.456±0.5	–6.397±0.5	–6.251±0.4
Plants	25±11.5	16±3.5	15±2.2	21±2.3	26±3.0	–16.299±1.2	–17.370±1.5	–17.601±1.6	–16.340±1.7	–14.894±1.2
Audio	30±7.5	26±2.9	29±3.9	31±5.7	41±7.8	–41.868±0.8	–42.114±0.9	–42.106±1.1	–41.831±1.1	–41.352±1.0
Jester	19±8.6	16±2.8	17±2.8	19±3.1	30±3.0	–54.890±1.0	–54.880±0.9	–54.924±0.9	–54.723±1.0	–54.317±1.1
Netflix	45±12.5	42±9.3	43±7.2	44±5.5	64±6.8	–59.689±1.2	–60.035±1.4	–60.010±1.4	–59.878±1.5	–59.397±1.4
Accidents	19±10.9	16±2.4	17±1.9	18±2.8	29±5.7	–40.263±1.6	–40.486±1.6	–40.480±1.7	–38.677±1.7	–35.939±1.8
Retail	13±6.2	10±3.2	11±1.9	13±3.3	17±6.1	–11.343±0.4	–11.344±0.4	–11.310±0.4	–11.307±0.4	–11.134±0.5
Pumsb-star	18±4.9	19±3.5	18±1.6	21±2.2	30±8.2	–42.149±2.1	–43.027±2.1	–41.890±1.8	–37.595±1.8	–33.429±2.9
DNA	4±2.7	2±0.6	3±1.1	3±0.7	4±0.7	–99.134±0.8	–99.043±0.8	–98.675±1.0	–97.757±1.2	–94.136±2.0
Kosarek	26±8.2	28±8.8	30±10.7	32±14.3	35±15.6	–11.396±0.5	–11.394±0.5	–11.492±0.5	–11.331±0.5	–11.080±0.5
MSWeb	43±12.0	39±9.4	46±16.0	50±22.4	54±17.8	–10.810±0.6	–10.862±0.5	–10.780±0.6	–10.728±0.6	–10.495±0.7
Book	38±9.7	36±8.2	37±9.2	45±14.5	72±22.5	–35.621±0.9	–35.625±0.9	–35.326±0.9	–35.329±0.9	–35.079±1.0
EachMovie	18±6.3	21±9.7	22±8.2	28±12.1	36±18.3	–62.674±3.6	–62.658±3.6	–61.950±3.1	–60.286±3.0	–56.459±2.7
WebKB	27±7.3	27±11.1	42±16.3	44±18.6	79±35.7	–166.195±5.0	–165.302±4.2	–164.458±4.3	–162.870±4.6	–160.624±3.9
Reuters-52	62±25.6	58±23.5	88±32.7	116±32.8	206±98.1	–95.497±4.0	–95.279±3.9	–94.741±4.2	–94.614±3.3	–91.227±3.0
BBC	17±6.3	21±9.7	28±12.3	26±12.6	71±41.1	–266.533±5.4	–266.555±5.2	–266.122±5.4	–265.563±5.2	–260.271±4.4
Ad	32±14.1	34±17.6	55±27.8	90±29.2	167±50.9	–49.703±16.3	–50.103±14.3	–49.145±14.4	–45.328±14.1	–35.567±10.2
EUR-Lex	4646±551	7255±4288	9612±5323	15516±7473	41742±18454	–103.055±6.5	–103.108±6.4	–103.007±6.4	–103.095±6.2	–101.668±5.7

Table 8
Learning times in seconds and average test log-likelihoods (with standard deviation) for the best validation models for RSBVS, varying the proportion of samples involved in GVS

	learning time				log-likelihood			
	50%	30%	20%	15%	50%	30%	20%	15%
NLTCS	31±2.9	11±2.5	6±0.7	5±0.4	–6.083±0.2	–6.219±0.2	–6.324±0.2	–6.550±0.2
Plants	171±11.4	66±3.8	33±1.2	25±1.9	–13.703±0.6	–14.915±0.8	–16.703±0.9	–18.673±0.9
Audio	59±2.8	27±3.1	17±2.7	10±1.3	–41.566±0.7	–42.789±0.6	–44.184±0.7	–45.163±0.1
Jester	33±4.3	12±1.4	5±0.4	4±0.6	–54.322±0.7	–55.397±0.7	–57.420±0.4	–57.841±0.1
Netflix	49±6.5	27±1.5	19±3.3	11±1.4	–59.525±1.0	–61.090±0.9	–61.961±0.6	–62.882±0.4
Accidents	196±4.4	81±6.0	51±2.7	38±1.6	–32.270±1.0	–33.344±0.9	–34.469±1.0	–36.309±1.2
Retail	39±6.7	24±2.9	15±1.2	13±2.4	–11.027±0.3	–11.093±0.3	–11.238±0.3	–11.324±0.4
Pumsb-star	437±11.5	158±10.7	89±7.8	68±5.6	–27.558±1.0	–29.496±1.1	–31.926±1.1	–34.048±0.9
DNA	15±2.2	3±0.2	3±0.6	3±0.3	–83.493±1.5	–99.666±0.5	–99.666±0.5	–99.666±0.5
Kosarek	152±14.7	104±5.7	57±6.3	50±5.0	–11.1750.3	–11.478±0.2	–11.682±0.1	–11.757±0.1
MSWeb	140±15.2	98±4.7	74±7.9	75±7.1	–10.352±0.4	–10.563±0.3	–10.755±0.3	–10.854±0.2
Book	112±7.5	75±4.7	59±4.0	57±3.8	–36.348±0.5	–36.941±0.2	–37.176±0.0	–37.176±0.0
EachMovie	64±4.4	35±1.5	32±2.9	30±2.4	–55.205±1.1	–59.396±0.7	–64.217±0.5	–67.228±0.3
WebKB	96±7.1	66±7.4	64±8.4	63±5.1	–163.314±1.0	–168.000±0.5	–169.458±0.1	–169.458±0.1
Reuters-52	210±29.2	150±10.0	135±18.0	141±7.3	–92.061±1.7	–95.515±1.9	–96.889±1.9	–97.535±1.9
BBC	73±6.3	58±3.2	56±4.2	57±5.5	–266.339±1.3	–269.127±0.9	–269.361±0.8	–269.361±0.8
Ad	228±32.4	163±26.8	136±18.6	141±26.6	–33.252±4.2	–52.329±2.5	–58.168±2.2	–58.301±2.3
EUR-Lex	15025±2678	9967±5560	13078±402	11187±250	–102.040±6.5	–102.959±6.5	–103.021±6.5	–103.021±6.5

the accuracy when compared to RGVS but being still less accurate than GVS. Thus, to answer the research question **Q2**, we can state that WRGVS is able to learn more accurate but bigger networks than RGVS, needing a fraction of time more than it.

5.3. (Q3) Evaluating *RSBVS*

We evaluated the accuracy and the structural characteristics of networks learned by RSBVS varying the amount of samples taken into account when computing the G-test.

In particular, in our experiments we evaluated the method taking at random the 50%, 30%, 20% and 15% of samples from the considered data slice. We expect a monotonic behavior, the lower the amount of taken samples, the lower the accuracy and the learning time.

Looking at Table 8, our hypothesis about RSBVS monotonic behavior is experimentally validated. In fact, both accuracy and learning times decrease accordingly to the reduction of taken samples. When datasets have a small amount of training samples, such as DNA and BBC, there is small or no difference in accuracy when RSBVS takes at most the 30% of samples.

As regards the structural characteristics, we obtained the same behavior as accuracy, i.e., reducing the percentage of samples corresponds to a decrease of learned network sizes. Considering learning times, on some datasets (e.g., Reuters-52 and Ad), it happens that learning times are longer when the 15% of samples than the 20% is taken. However, these differences are rather small and may depend on the randomness of the algorithm, or when the best validation model is the one that needs additional time for its specific hyper-parameter configuration–for instance, a smaller μ.

In question **Q3** we asked how does RSBVS perform when compared to GVS in terms of accuracy and learning times. Since varying the amount of samples taken for the independencies discovering, RSBVS shows a monotonic behavior, hence we compared RSBVS to GVS taking the 50% of samples–conclusions about the results could be obtained with smaller amounts of samples follow immediately. In our experimental setting, RSBVS taking 50% of samples, when compared to GVS, significantly degrades in terms of accuracy but it is fast on datasets with many RVs like Reuters-52, BBC, Ad and EUR-Lex.

From Table 10, RSBVS, when compared to GVS, learns in general smaller networks with far fewer parameters on datasets with a high number of variables.

In conclusion, answering question **Q3**, we can state that RSBVS is able to learn smaller and less accurate networks when compared to GVS but it gains in learning times only on datasets with a high number of RVs. Moreover, in these contexts, it learns networks with far fewer parameters.

Table 9
Structural quality metrics (the number of edges, layers and network parameters) for the best validation models for EBVS, EBVS-AE, RGVS, WRGVS and GVS

	# edges					# layers					# params				
	EBVS	EBVS-AE	RGVS	WRGVS	GVS	EBVS	EBVS-AE	RGVS	WRGVS	GVS	EBVS	EBVS-AE	RGVS	WRGVS	GVS
NLTCS	8185	3969	324	676	1129	23	9	9	12	17	7999	3962	301	607	1002
Plants	42318	67821	2485	4741	15129	27	17	13	17	27	41815	67788	2302	4296	13830
Audio	36499	43243	2743	4877	17811	21	11	11	13	25	36411	43235	2574	4406	16459
Jester	26263	27609	1966	2648	12460	17	5	13	13	25	26235	27608	1842	2389	11447
Netflix	42033	44512	4999	7878	30417	21	11	15	15	31	41983	44507	4652	7103	28265
Accidents	98218	33377	1380	2333	11861	33	15	13	15	27	96624	33349	1280	2108	10539
Retail	10096	6973	353	356	1010	41	37	7	7	19	9954	6880	341	343	924
Pumsb-star	32776	137092	1463	2746	12821	29	19	12	16	27	32024	137022	1369	2513	11484
DNA	8694	8694	434	637	3384	7	7	7	7	13	8691	8691	423	594	2916
Kosarek	38097	42732	1155	1626	3692	31	41	12	13	23	37577	42587	1099	1495	3391
MSWeb	41447	57482	668	964	10341	35	35	7	10	33	40704	57423	656	907	9435
Book	128181	65002	2224	2850	3814	29	33	9	10	11	127746	64961	2174	2690	3598
EachMovie	34155	31467	2531	4546	24458	21	25	10	16	29	34024	31406	2402	4197	22833
WebKB	61125	34863	3009	4041	9344	83	7	10	11	15	60903	34860	2891	3736	8727
Reuters-52	121211	83826	3053	4789	82084	103	21	10	13	27	120851	83791	2927	4398	77125
BBC	39997	26167	2534	3435	68117	71	7	7	10	25	39862	26163	2474	3243	64590
Ad	90599	166802	3396	5468	20823	157	137	7	11	33	90179	166642	3349	5198	20155
EUR-Lex	575622	1242894	18222	20392	29515	9	115	7	7	17	575614	1242675	18186	20082	29070

Table 10
Structural quality metrics (the number of edges, layers and network parameters) for the best validation models for RSBVS varying the proportion of samples involved in GVS

	# edges				# layers				# params			
	50%	30%	20%	15%	50%	30%	20%	15%	50%	30%	20%	15%
NLTCS	1485	459	219	117	9	7	7	7	1401	434	208	109
Plants	10453	3134	1257	614	17	14	15	11	10089	3030	1212	587
Audio	3383	918	356	212	14	12	7	5	3223	867	332	210
Jester	2525	617	253	203	15	11	7	3	2381	587	244	203
Netflix	3093	870	390	255	13	11	9	5	2894	809	367	246
Accidents	5458	1646	742	495	21	13	10	8	5025	1510	682	461
Retail	549	395	303	278	9	8	5	5	528	385	300	277
Pumsb-star	19692	6011	2629	1576	27	23	18	16	19081	5794	2519	1507
DNA	1507	362	362	362	8	3	3	3	1370	362	362	362
Kosarek	3822	847	518	437	28	15	9	5	3731	812	503	430
MSWeb	1595	848	675	625	19	11	7	5	1516	821	665	621
Book	1851	1109	1002	1002	12	7	3	3	1795	1097	1002	1002
EachMovie	4376	1399	1074	1013	19	10	7	5	4280	1383	1068	1012
WebKB	2821	1770	1680	1680	13	6	3	3	2700	1757	1680	1680
Reuters-52	4433	2157	1844	1798	17	12	7	6	4253	2123	1837	1796
BBC	2530	2126	2118	2118	12	5	3	3	2487	2125	2118	2118
Ad	5941	3370	3136	3120	12	7	5	5	5772	3348	3134	3119
EUR-Lex	18259	17995	17988	17988	6	5	3	3	18230	17994	17988	17988

5.4. (Q4) Evaluating EBVS

From our results, EBVS learns much less compact networks w.r.t. GVS. Concerning learning times, while it speeds up the building procedure for a single node, the overall time required by the algorithm to grow a whole network increases (Table 2). This is due to the fact that it calls the row clustering procedure less frequently than GVS since it learns networks with more nodes but with fewer parameters (Table 9). Thus, it moves into the search space faster but it favours larger networks than GVS. An exception happens on datasets with a high number of variables, such as Reuters-52, BBC and Ad and noticeably on EUR-Lex where EBVS gains considerable shorter learning times w.r.t. GVS.

On the accuracy side, instead, one can see that EBVS has comparable results w.r.t GVS, outperforming it on a challenging dataset as EUR-Lex, and it performs better than RGVS (Table 5). To answer the research question **Q4** we can state that EBVS does not learn more compact networks than GVS but it learns more accurate networks than RGVS.

On datasets with a high number of variables ($|V| > 500$) EBVS is able to learn a network needing shorter times than GVS, gaining comparable accuracy (e.g., on BBC and in a challenging setting like EUR-Lex). Still, on datasets with few variables, due to the increased size of the learned models their learning time increased.

5.5. (Q5) Evaluating EBVS-AE

In the previous Section we questioned the introduction of an adaptive thresholding method for the entropy-based splitting procedure. Results confirm our intuition over the employed datasets.

For EBVS-AE, as showed in Table 2, on all datasets we obtained remarkable shorter learning times than EBVS (needing half time on Reuters-52 and BBC that have $|V| > 800$). On accuracy, EBVS-AE outperforms (on Plants, Audio, Jester, Netflix, Book, Reuters-52, MSWeb) EBVS or it is not significantly worse (on NLTCS, Retail, DNA, Kosarek, WebKB, BBC). Only on few datasets it performs significantly worse than EBVS (Accidents, Pumsb-star, EachMovie, Ad, EUR-Lex) but gaining a comparable accuracy on most of them while learning a smaller network (Accidents and EachMovie)–see Table 9. On many datasets our adaptive version of EBVS is capable to learn smaller networks than EBVS (see Table 9) especially on datasets with a high number of RVs (Book, EachMovie, WebKB, Reuters-52, BBC).

In general, EBVS-AE compared to EBVS learns networks with a considerable smaller number of weights, thus, it spends less time for row clustering during the learning procedure.

EBVS-AE when compared to the baseline GVS achieves comparable or better results on many considered datasets (Table 5). Hence, we can answer **Q5** by stating that EBVS-AE, in general, tends to

learn more accurate and compact networks with fewer weights than EBVS, needing less time and often being more accurate, and as such shall be preferred over it.

6. Conclusions

Learning an SPN from high dimensional data still poses a challenge in terms of time complexity. The simplest greedy structure learner, LearnSPN, scales quadratically in the number of the variables to determine RVs independencies.

In this paper, we proposed approximate but faster procedures to determine independencies among RVs whose complexity scales in sub-quadratic time.

We investigated three approaches: two based on random subspaces and another one that adopts entropy as a criterion to split RVs in linear time.

Experimental results prove that there is no free lunch: LearnSPN equipped by the formers learns networks that save on learning and inference time, providing less accurate inference; while with the latter procedure, it is able to produce networks that are still accurate estimators but requiring more time when learning and evaluating due to bigger size on datasets with few RVs. While, on datasets with a high number of variables it requires less time but gains worse accuracy than our baseline. Remarkable results in both accuracy and learning time on challenging datasets as EUR-Lex encourage future investigation in this direction.

References

[1] T. Adel, D. Balduzzi and A. Ghodsi, Learning the structure of sum-product networks via an svd-based algorithm, In *UAI*, 2015.

[2] J. Bekker, J. Davis, A. Choi, A. Darwiche and G. Van den Broeck, Tractable learning for complex probability queries, In *Advances in Neural Information Processing Systems 28 (NIPS)*, 2015.

[3] Y. Bengio, A.C. Courville and P. Vincent, Unsupervised Feature Learning and Deep Learning: A review and new perspectives, CoRR, abs/1206.5538, 2012.

[4] C.M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., 2006.

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

[6] W.-C. Cheng, S. Kok, H. Vu Pham, H.L. Chieu and K.M. Adam Chai, Language modeling with Sum-Product Networks, In *INTERSPEECH* **2014** (2014), 2098–2102.

[7] A. Darwiche, A differential approach to inference in bayesian networks, *JACM*, 2003.

[8] A. Dennis and D. Ventura, Learning the Architecture of Sum-Product Networks Using Clustering on Varibles, In *NIPS* **25** (2012), 2033–2041.

[9] N. Di Mauro, F. Esposito, F.G. Ventola and A. Vergari, Alternative variable splitting methods to learn sum-product networks. In *Proceedings of AIXIA*. Springer, 2017.

[10] N. Di Mauro, A. Vergari and T.M.A. Basile, Learning bayesian random cutset forests, In *ISMIS* Springer, 2015, pp. 122–132.

[11] N. Di Mauro, A. Vergari and F. Esposito, Learning accurate cutset networks by exploiting decomposability, In *AIXIA* Springer, 2015, pp. 221–232.

[12] A. Friesen and P. Domingos, The sum-product theorem: A foundation for learning tractable models, In *ICML*, 2016, pp. 1909–1918.

[13] R. Gens and P. Domingos, Discriminative Learning of Sum-Product Networks, In *Advances in Neural Information Processing Systems* **25** (2012), pp. 3239–3247.

[14] R. Gens and P. Domingos, Learning the Structure of Sum-Product Networks, In *ICML*, 2013, pp. 873–880.

[15] J. Van Haaren and J. Davis, Markov network structure learning: A randomized feature generation approach, In *AAAI*. AAAI Press, 2012.

[16] W. Hsu, A. Kalra and P. Poupart, Online structure learning for sum-product networks with gaussian leaves, arXiv preprint arXiv:1701.05265, 2017.

[17] P. Jaini, A. Rashwan, H. Zhao, Y. Liu, E. Banijamali, Z. Chen and P. Poupart, Online algorithms for sum-product networks with continuous variables, In *Probabilistic Graphical Models - Eighth International Conference, PGM 2016, Lugano, Switzerland, September 6-9, 2016. Proceedings*, 2016, pp. 228–239.

[18] D. Koller and N. Friedman, Probabilistic Graphical Models: Principles and Techniques, *MIT Press*, 2009.

[19] Y. Liang, J. Bekker and G. Van den Broeck. Learning the structure of probabilistic sentential decision diagrams, In *Proceedings of the 33rd Conference on Uncertainty in Artificial Intelligence (UAI)*, 2017.

[20] D. Lowd and J. Davis, Learning Markov network structure with decision trees, In *ICDM*, IEEE Computer Society Press, 2010, pp. 334–343.

[21] D. Lowd and A. Rooshenas, Learning Markov networks with arithmetic circuits, In *Proceedings of the 16th International Conference on Artificial Intelligence and Statistics, volume 31 of JMLR Workshop Proceedings*, 2013, pp. 406–414.

[22] D.J.C. MacKay, *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, New York, NY, USA, 2002.

[23] J. Martens and V. Medabalimi, On the Expressive Efficiency of Sum Product Networks. *CoRR*, abs/1411.7717, 2014.

[24] M. Melibari, P. Poupart, P. Doshi and G. Trimponias, Dynamic sum product networks for tractable inference on sequence data, In *Probabilistic Graphical Models - Eighth International Conference, PGM 2016, Lugano, Switzerland, September 6-9, 2016. Proceedings*, 2016, pp. 345–355.

[25] E. Loza Mencía and J. Fürnkranz, Efficient pairwise multilabel classification for large-scale problems in the legal domain, In *ECML/PKDD*, 2008.

[26] A. Molina, S. Natarajan and K. Kersting, Poisson sum-product networks: A deep architecture for tractable multivariate poisson distributions, In *AAAI*, 2017.

[27] A. Molina, A. Vergari, N. Di Mauro, S. Natarajan, F. Esposito and K. Kersting, Mixed sum-product networks: A deep architecture for hybrid domains, In *AAAI*, 2018.

[28] R. Peharz, *Foundations of Sum-Product Networks for Probabilistic Modeling*. PhD thesis, Graz University of Technology, SPSC, 2015.

[29] R. Peharz, B. Geiger and F. Pernkopf, Greedy Part-Wise Learning of Sum-Product Networks, In *ECMLPKDD 2013*, 2013.

[30] R. Peharz, R. Gens and P. Domingos, Learning selective sum-product networks, In *Workshop on Learning Tractable Probabilistic Models*. LTPM, 2014.

[31] R. Peharz, G. Kapeller, P. Mowlaee and F. Pernkopf, Modeling speech with sum-product networks: Application to bandwidth extension, In *ICASSP*, 2014.

[32] R. Peharz, S. Tschiatschek, F. Pernkopf and P. Domingos, On theoretical properties of sum-product networks, *The Journal of Machine Learning Research*, 2015.

[33] H. Poon and P. Domingos, Sum-Product Networks: A New Deep Architecture, *UAI* 2011, 2011.

[34] M. Queyranne, Minimizing symmetric submodular functions, *Mathematical Programming* **82**, 1998.

[35] T. Rahman and V. Gogate, Merging strategies for sum-product networks: From trees to graphs, In *UAI*, 2016, pp. ??–??.

[36] T. Rahman, P. Kothalkar and V. Gogate, Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of Chow-Liu trees, In *ECML-PKDD*, Springer, 2014, pp. 630–645.

[37] A. Rooshenas and D. Lowd, Learning Sum-Product Networks with Direct and Indirect Variable Interactions, In *ICML*, 2014.

[38] D. Roth, On the hardness of approximate reasoning, *Artificial Intelligence* **82**(1âĂŞ2) 1996, 273–302.

[39] M. Scanagatta, G. Corani, C. Polpo de Campos and M. Zaffalon. Learning treewidth-bounded Bayesian networks with thousands of variables, In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 2016, pp. 1462–1470.

[40] M. Trapp, R. Peharz, M. Skowron, T. Madl, F. Pernkopf and R. Trappl, Structure inference in sumproduct networks using infinite sum-product trees, **12**, 2016.

[41] A. Vergari, R. Peharz, N. Di Mauro, A. Molina, K. Kersting and F. Esposito, Sum-product autoencoding: Encoding and decoding representations using sum-product networks. 2018.

[42] A. Vergari, N. Di Mauro and F. Esposito, Simplifying, Regularizing and Strengthening Sum-Product Network Structure Learning, In *ECML-PKDD*, 2015.

[43] A. Vergari, N. Di Mauro and F. Esposito, Visualizing and understanding sum-product networks. *CoRR abs/1608.08266*, 2016.

[44] Z. Yuan, H. Wang, L. Wang, T. Lu, S. Palaiahnakote and C. Lim Tan, Modeling spatial layout for scene image understanding via a novel multiscale sum-product network, *Expert Systems with Applications* **63** (2016), 231–240.

[45] H. Zhao, M. Melibari and P. Poupart, On the Relationship between Sum-Product Networks and Bayesian Networks, In *ICML*, 2015.

[46] H. Zhao and P. Poupart, A unified approach for learning the parameters of sum-product networks, *arXiv preprint arXiv:1601.00318*, 2016.