Undefined 1 (2017) 1–5
IOS Press

Sum-Product Network Structure Learning by Efficient Product Nodes Discovery

Nicola Di Mauro [a], Floriana Esposito [a] and Fabrizio Giuseppe Ventola [a]* Antonio Vergari [a]
[a] *Department of Computer Science, University of Bari, Via E. Orabona 4, 70125, Bari, Italy*
E-mail: {nicola.dimauro, floriana.esposito, fabrizio.ventola, antonio.vergari}@uniba.it

Abstract. Sum-Product Networks (SPNs) are recently introduced deep probabilistic models providing exact and tractable inference. Even if SPNs have been successfully employed in several application domains, learning their structure from high dimensional data still poses a challenge in terms of time complexity. Classical SPNs structure learning algorithms work by exploiting two high cost operations repeated several times: 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 some approximate but fast procedures to determine independencies among RVs whose time complexity scales sub-quadratically. In particular, we propose two procedures based on a random subspace approach, and a third one adopting entropy as a criterion to split RVs. Experimental results, on many state-of-the-art datasets for density estimation, prove that LearnSPN equipped by our splitting procedures is able to reduce learning and/or inference times while preserving comparable inference accuracy.

Keywords: Machine Learning, Deep Learning, Structure Learning, Probabilistic Models, Density Estimation, Sum-Product Networks

1. Introduction

Density estimation represents 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 [2]. When such an estimator $\hat{\mathsf{p}}_{\mathbf{X}} \sim \mathsf{p}_{\mathbf{X}}$ 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.

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 simply reframed as performing inference on a probability distribution. Hence, the aim becomes twofold: building a highly *expressive and accurate* estimator, and being able to *efficiently learn* the estimator 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, rare to being able to optimize all of them in practice.

Probabilistic graphical models (PGMs) [14], like Bayesian networks or Markov networks, are able to accurately model highly complex probability distributions. However, performing inference and learning on PGMs requires routines that, even if approximate, may scale exponentially in the worst case [29].

Recently, different tractable probabilistic models (TPMs) have been introduced, allowing tractable inference at the expense of a stricter representation bias. TPMs like Arithmetic Circuits [5], Sum-Product Networks (SPNs) [24], and Cutset Networks [9,27] provide a good compromise between expressiveness and tractability by compiling complex distributions in compact data structures. Nevertheless, as we will see in the following, the cost of learning them when modeling high dimensional probability distribution is still an issue.

In this paper, we focus on SPNs, and we investigate approximate learning routines to improve both learning and inference complexity, trying to preserve the model expressiveness.

SPNs encode a probability distribution as a set of sum and product nodes, arranged in a deep architecture, represented as a DAG. By imposing some structural conditions on the networks, SPNs guarantee several kinds of probabilistic queries to be computed exactly and in time linear to the network size–the number of edges.

The success of SPNs in many application domains, like computer vision [24,33], speech recognition [23], natural language processing [4] and for representation learning [32], increased the interest around algorithms able to learn both their structure and parameters [6,11, 19,31].

One of the first principled top-down learning schem-es, and yet still a state-of-the-art structure learner, is LearnSPN [11]. One of the factors behind its popularity is its simplicity. LearnSPN greedily and recursively decomposes the observed data by either splitting on RVs after they have been found independent, or by clustering samples together according to some metric. While several variations of LearnSPN have been proposed in the literature by adapting the splitting RVs and clustering routines, their high complexity constitutes the main bottleneck. Our attention is on the first procedure, splitting RVs into independent subsets, whose general complexity, in the worst case, is quadratic in the number of the RVs.

In [7] alternative approximated procedures for splitting RVs in the SPNs structure learner LearnSPN have been presented. In this paper we extend the work in [7] by introducing an improved version of the stochastic variable splitting method, proposing a new stochastic method based on random sampling, and validating all the proposed approaches on additional datasets with a higher number of random variables.

The contributions of this paper are the following: i) we advance three alternative procedures to approximate the variable splitting method in LearnSPN while performing it in sub-quadratic time, and ii) we empirically evaluate their effectiveness in trading-off inference accuracy in favour of inference or learning times. We propose a random subspace approach in which we reduce the number of independence tests to be computed. Moreover, we propose another random subspace approach that reduces the amount of instances on which to compute the independence test. Additionally, we investigate an even more approximated entropy-based criterion that scales linearly in the number of RVs.

An extensive comparison w.r.t. the original version of LearnSPN on several standard benchmark datasets reveals that the random subspace proposed approaches effectively trade off the model likelihood in favor of 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 the model expressiveness.

2. Sum-Product Networks

An SPN S is a computational graph defined by a rooted DAG, encoding an unnormalized probability distribution 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 (graphically denoted resp. as $\bigoplus$ and $\bigotimes$), and leaves are univariate distributions defined on a RV $X_i \in \boldsymbol{X}$. Each node $n \in S$ has a *scope* $\mathsf{sc}(n) \subseteq \mathbf{X}$, 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, 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 corresponds to 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 Figure 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 $S(\boldsymbol{x})$ 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 properties together imply *validity* [24] i.e. the ability of exactly computing the probability of each possible complete or partial evidence configuration.

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 [20,24]. W.l.o.g., we assume the SPNs we are considering to be valid and normalized.

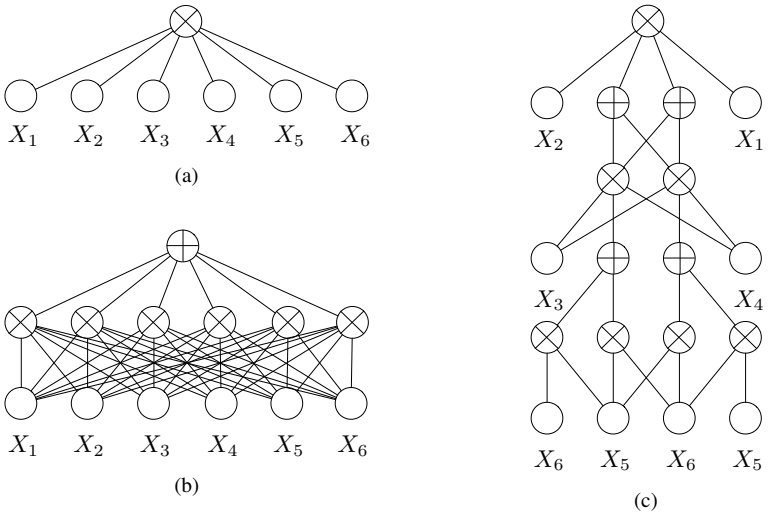

Fig. 1. Examples of SPNs: a naive factorization over 6 random variables (1a), a shallow mixture standing for a point-wise kernel density estimation (1b) and a deeper architecture (1c) over the same scope (weights are omitted for simplicity).

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, $S_i(\mathbf{x}_{|\mathsf{sc}(i)})$ corresponds to the probability of the state $\mathbf{x}_{|\mathsf{sc}(i)}$ for the RV $\mathsf{sc}(i)$: $S_i(\mathbf{x}) = P(\mathsf{sc}(i) = \mathbf{x}_{|\mathsf{sc}(i)})$. 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–its number of edges [11,20]. Hence, tractable inference is achieved when the number of edges is at most polynomial in the number of RVs. Moreover, *the less the number of edges in a network, the faster the inference*. While the size of the 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* [17]. 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 [31]. Lastly, also the number of parameters of a network, i.e., the number of sum node weights, also called *model capacity*, influences its expressiveness with respect to a weight learning algorithm. 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.

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 [14].

In this work, following [31], we argue that both the network structure quality metrics and its log-likelihood shall be taken into account to better evaluate the inherent trade-off of density estimators between learning and inference tractability and their expressiveness and accuracy.

2.1. Structure Learning

As already stated, we focus our attention to a particular structure learning algorithm for SPNs, LearnSPN [11]. 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

[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 [31].

in [1,28,31]. Furthermore, they can also be adapted to other learning scenarios, in which one has to split RVs into sets of (approximately) independent RVs [10], e.g., eliciting the 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. LearnSPN then tries to aggregate similar submatrix rows (procedure clusterInstances) into a number of clusters. In the original work of [11], the hard online EM algorithm is employed to determine an adaptive number of clusters, l, assuming RVs X_j independent given the row clusters C_i, formally: $P(\boldsymbol{X}) = \sum_i P(C_i) \prod_j P(X_j|C_i)$. To control the cluster number, an exponential prior on clusterings in the form of $e^{-\lambda l |\boldsymbol{X}|}$ is used, with λ as a tuning parameter.

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 [31], 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 4.

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

Require: a set of samples $\mathcal{D}$ over RVs $\boldsymbol{X}$; α: Laplace smoothing parameter; μ: minimum number of instances to split; ρ: statistical independence threshold

Ensure: an SPN S encoding a pdf over $\boldsymbol{X}$ learned from $\mathcal{D}$

1: **if** $|\boldsymbol{X}| = 1$ **then**
2: $\quad S \leftarrow \mathsf{univariateDistribution}(\mathcal{D}, \boldsymbol{X}, \alpha)$
3: **else if** $|\mathcal{D}| < \mu$ **then**
4: $\quad S \leftarrow \mathsf{naiveFactorization}(\mathcal{D}, \boldsymbol{X}, \alpha)$
5: **else**
6: $\quad \{\boldsymbol{X}_{d_1}, \boldsymbol{X}_{d_2}\} \leftarrow \mathsf{GVS}(\mathcal{D}, \boldsymbol{X}, \rho)$
7: $\quad$ **if** $|\boldsymbol{X}_{d_2}| > 0$ **then**
8: $\quad\quad S \leftarrow \prod_{j=1}^{2} \mathsf{LearnSPN}(\mathcal{D}, \boldsymbol{X}_{d_j}, \alpha, \mu, \rho)$
9: $\quad$ **else**
10: $\quad\quad \{\mathcal{D}_i\}_{i=1}^{R} \leftarrow \mathsf{clusterInstances}(\mathcal{D}, \boldsymbol{X})$
11: $\quad\quad S \leftarrow \sum_{i=1}^{R} \frac{|\mathcal{D}_i|}{|\mathcal{D}|} \mathsf{LearnSPN}(\mathcal{D}_i, \boldsymbol{X}, \alpha, \mu, \rho)$
12: **return** S

3. Related Work

Since the seminal paper about SPNs [24], researchers have designed different algorithms to learn a SPN structure from data. A first attempt has been the one presented in [6]. This algorithm proceeds top-down but differently from LearnSPN. First, it clusterizes instances once, at the beginning. Then, it proceeds splitting variables without exploiting local independencies, building a region graph, i.e., a rooted DAG consisting of region nodes and partition nodes.

The root node is a region node, partition nodes are restricted to being the children of region nodes and vice versa without losing representational power. The definition of scope for region and partition nodes is the same as for SPN nodes. Once built a region graph, the algorithm transforms it in an SPN with multiple roots. After that, network parameters are estimated separately with EM algorithm.

Another similar strategy has been described in [21]. As the previously described algorithm it first builds a region graph and then convert it to a multi-root SPN. To build the region graph the algorithm starts bottom-up from the univariate distributions corresponding to the RVs without partitioning them horizontally. Then, it recursively tries to merge regions of nodes discovering latent variable interactions. Network weights are estimated during the merging procedure.

One of the state-of-the-art learner is ID-SPN presented in [28]. It has been the first algorithm learning hybridized SPNs using ACs to model distributions at leaves. It leverages the union of discovering latent variable interactions with previously mentioned approaches with the discovery of direct variable interactions as developed for learning classical graphical models like Markov Networks. A part of its learning algorithm is driven directly by the likelihood on data, differently from other algorithms using different approaches like heuristic functions. Even though ID-SPN outperforms several algorithms, it results very slow in practice. In fact, ACs increase structural complexity at leaves and their learning procedure burdens the overall network learning algorithm.

In [22] the authors propose a constrained less-expressive class of SPN named "selective SPNs" where each sum node must have at most one child with non-zero output for each possible input. With this network formulation it is possible to find globally optimal maximum likelihood parameters with a closed form. Moreover, this allows the employment of stochastic local search techniques to speed up the learning procedure trading off accuracy.

In the opposite direction goes the work presented in [26] where the authors introduced a way to learn graph SPNs that try to overcome the limited expressiveness of tree SPNs. Their approach learns a graph SPN starting from different tree SPNs and merging them by looking at similar sub-structures.

Another work on SPNs expressiveness is the one made in [30] proposing a principled approach for structure learning by means of infinite Sum-Product Trees (SPTs), in other words, an extension of SPNs to a Bayesian nonparametric model with a posterior distribution based on induced trees.

An ad-hoc SPN structure learning algorithm for sequence data is conceived in [18]. In that work, the authors firstly showed how Dynamic SPNs (DSPNs) can be used to model sequence data and then they presented an algorithm to learn the structure of the template network that is repeated as many times as needed to model data sequences of any length. They compare DSPNs with Dynamic Bayesian Networks highlighting the advantages of DSPNs thank to their feasible inference.

An online structure learner has been proposed in [13]. The idea is to update both the structure and the parameters of an SPN keeping completeness and decomposability. The introduced algorithm first updates the network parameters. Then, it checks whether the correlation between variables of different children of the same product node is under a certain correlation threshold. If it turns out that two variables are over the threshold it updates the network structure by creating a multivariate leaf node or by creating a mixture of two components over the variables.

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 with minimum empirical mutual information (MI), but this will scale in a cubic time w.r.t. the number of RVs considered [25]. Instead, as the name suggests, GVS proceeds in a greedy way, lowering the time complexity to be quadratic (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, a G-Test, is applied in the original formulation of GVS. 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:

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

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. [19,28], even if employing different pairwise statistical independence tests, still require a quadratic number of comparisons, in the worst case.

Algorithm 2 GVS($\mathcal{D}$, $\boldsymbol{X}$, 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: $\boldsymbol{X}_{d_2} \leftarrow \boldsymbol{X}$, $X_0 \leftarrow$ drawVariableAtRandom($\boldsymbol{X}$)
2: $\mathbf{X} \leftarrow \mathbf{X} \setminus \{X_0\}$, $\boldsymbol{P} \leftarrow \{X_0\}$, $\boldsymbol{X}_{d_1} \leftarrow \{X_0\}$, $\boldsymbol{R} \leftarrow \emptyset$
3: **repeat**
4: $\quad X_p \leftarrow \mathsf{pop}(\boldsymbol{P})$ $\quad \triangleright\ \boldsymbol{P} \leftarrow \boldsymbol{P} \setminus \{X_p\}$
5: $\quad$ **for each** $X_k \in \mathbf{X}$ **do**
6: $\quad\quad$ **if** not GTest($\mathcal{D}, X_p, X_k$) $< \rho$ **then**
7: $\quad\quad\quad \boldsymbol{R} \leftarrow \boldsymbol{R} \cup \{X_k\}$
8: $\quad\quad\quad \boldsymbol{X}_{d_1} \leftarrow \boldsymbol{X}_{d_1} \cup \{X_k\}$
9: $\quad\quad\quad \boldsymbol{P} \leftarrow \boldsymbol{P} \cup \{X_k\}$
10: $\quad$ **for each** $X_j \in \boldsymbol{R}$ **do**
11: $\quad\quad \mathbf{X} \leftarrow \mathbf{X} \setminus \{X_j\}$
12: **until** $|\boldsymbol{P}| > 0 \wedge |\boldsymbol{X}| > 0$
13: **return** $\{\boldsymbol{X}_{d_1}, \boldsymbol{X}_{d_2} \setminus \boldsymbol{X}_{d_1}\}$

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 [3] but also for density estimation [8]. We name this method *Random Greedy Variable Splitting* (RGVS) sketched in Algorithm 3.

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 missed dependencies in one call in the following calls that LearnSPN performs on the same RVs. 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 was 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 LearnSPN calls the splitting routines a larger number of times since in each call the 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 stochasticity of RGVS in agglomerating the remaining variables not considered by GVS is implemented in WRGVS (Algorithm 4) where W stands for *wiser*. This method, firstly extracts one RV selected at random from each subset

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

found by GVS and uses it as a representative of that subset. Then, each remaining variable is added to the subset whose representative variable has the stronger dependency according to a G-test. With this approach we aim to improve the accuracy of learned networks w.r.t. the ones learned with RGVS keeping, at the same time, the complexity linear in the number of variables.

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 $\{\boldsymbol{X}_{d_1}, \boldsymbol{X}_{d_2}\}$

$\boldsymbol{X}_{d_1} \leftarrow \emptyset, \boldsymbol{X}_{d_2} \leftarrow \emptyset$, $n \leftarrow |\boldsymbol{X}|$, $k \leftarrow max(\lfloor \sqrt{n} \rfloor, 2)$
if $k = n$ **then**
  return GVS($\mathcal{D}, \boldsymbol{X}, \rho$)
$\boldsymbol{R} \leftarrow$ randomSubspace($\boldsymbol{X}, k$)
$\boldsymbol{S} \leftarrow \boldsymbol{X} \setminus \boldsymbol{R}$
$\{\boldsymbol{X}_{d_1}, \boldsymbol{X}_{d_2}\} \leftarrow$ GVS($\mathcal{D}, \boldsymbol{R}, \rho$)
if $\boldsymbol{X}_{d_2} = \emptyset$ **then**
  return $\{\boldsymbol{X}_{d_1} \cup \boldsymbol{S}, \emptyset\}$
$r \leftarrow$ Bernoulli(0.5)
if $r = 0$ **then**
  return $\{\boldsymbol{X}_{d_1} \cup \boldsymbol{S}, \boldsymbol{X}_{d_2}\}$
else
  return $\{\boldsymbol{X}_{d_1}, \boldsymbol{X}_{d_2} \cup \boldsymbol{S}\}$

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 [16]. 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 may apply Laplacian smoothing to compute the probabilities involved in the entropy computation[4].

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

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 $\{\boldsymbol{X}_{d_1}, \boldsymbol{X}_{d_2}\}$

$\boldsymbol{X}_{d_1} \leftarrow \emptyset, \boldsymbol{X}_{d_2} \leftarrow \emptyset$, $n \leftarrow |\boldsymbol{X}|$, $k \leftarrow max(\lfloor \sqrt{n} \rfloor, 2)$
if $k = n$ **then**
  return GVS($\mathcal{D}, \boldsymbol{X}, \rho$)
$\boldsymbol{R} \leftarrow$ randomSubspace($\boldsymbol{X}, k$)
$\boldsymbol{S} \leftarrow \boldsymbol{X} \setminus \boldsymbol{R}$
$\{\boldsymbol{X}_{d_1}, \boldsymbol{X}_{d_2}\} \leftarrow$ GVS($\mathcal{D}, \boldsymbol{R}, \rho$)
if $\boldsymbol{X}_{d_2} = \emptyset$ **then**
  return $\{\boldsymbol{X}_{d_1} \cup \boldsymbol{S}, \emptyset\}$
$X_j \leftarrow$ drawVariableAtRandom($\boldsymbol{X}_{d_1}$)
$X_k \leftarrow$ drawVariableAtRandom($\boldsymbol{X}_{d_2}$)
for each $X_i \in \mathbf{S}$ **do**
  if GTest($\mathcal{D}, X_i, X_j$) $\geq$ GTest($\mathcal{D}, X_i, X_k$) **then**
    $\boldsymbol{X}_{d_1} \leftarrow \boldsymbol{X}_{d_1} \cup X_i$
  else
    $\boldsymbol{X}_{d_2} \leftarrow \boldsymbol{X}_{d_2} \cup X_i$
return $\{\boldsymbol{X}_{d_1}, \boldsymbol{X}_{d_2}\}$

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 method 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. EBVS-AE determines each time the actual value of η proportionally to the number of samples $|\mathcal{D}|$ processed w.r.t. the number of samples in the whole dataset. Both EBVS and EBVS-AE involve the computation of the entropy for each RV in $\boldsymbol{X}$, hence their complexity is $O(nm)$.

[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 now η substitutes the hyperparameter ρ which is not needed anymore.

We empirically determine in the next Section whether both EBVS and EBVS-AE provide good approximations to the variable splitting method of LearnSPN. 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.

Algorithm 5 EBVS($\mathcal{D}$, $\boldsymbol{X}$, 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 $\{\boldsymbol{X}_{d_1}, \boldsymbol{X}_{d_2}\}$

$\boldsymbol{X}_{d_1} \leftarrow \emptyset, \boldsymbol{X}_{d_2} \leftarrow \emptyset$
for each $X_i \in \mathbf{X}$ **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 $\{\boldsymbol{X}_{d_1}, \boldsymbol{X}_{d_2}\}$

4.3. Random Sampling based Variable Splitting

Another proposed method to speed-up the variable splitting step of LearnSPN is based on *random sampling*. This time we apply a stochastic approach in order to reduce the number of samples to be used to estimate independencies between variables through G-test.

The method randomly selects (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 subset could have the same distribution of the set of instances in the considered data slice. 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-value for two variables and the threshold accordingly. Regarding the first one, we also need to fix the computation of the co-occurences $c(x_i, x_j)$. Now this value is computed on a subset of $\beta|\mathcal{D}|$ samples, while we need to require the correct expected count of $c(x_i, x_j)$ on the whole set $\mathcal{D}$. The expected counts on the whole $\mathcal{D}$ are computed as $c(x_i, x_j)/\beta$. For other quantities in Equation 1 we make a similar fix, leading to the following formula:

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

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 [31], a simplified variant of LearnSPN. We implemented them in Python[5], in the freely available version of LearnSPN-b [31] 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 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? **(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 17 datasets, employed as standard benchmarks to compare TPMs, being introduced in [15] and [12]. They comprise binarized

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

data coming from tasks such as frequent item mining, recommendation and classification. 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 also $\eta \in \{0.05, 0.1, 0.3, 0.5\}$.

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. 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 and Ad. Table 7 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 2 average test log-likelihoods are reported for all methods on all datasets except for RSBVS (see Table 9 for its detailed results). There, accuracy degrades on all datasets–significantly on Pumsb-star, DNA, WebKB, Reuters-52 and Ad– being comparable to other methods on some datasets such as Retail, Jester, Kosarek and MSWeb.

Additionally, to better understand the behavior of RGVS, we evaluate how changing the proportion of RVs involved in a 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 8, 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. 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 2, it turns out that WRGVS is significantly more accurate than RGVS on all datasets except Retail. 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 7, 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 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.

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					

Table 2
Times (in seconds) taken to learn the best models on each dataset and average test log-likelihoods for EBVS, EBVS-AE, RGVS, WRGVS and GVS.

	learning time					**log-likelihood**				
	EBVS	EBVS-AE	RGVS	WRGVS	GVS	EBVS	EBVS-AE	RGVS	WRGVS	GVS
NLTCS	98	31	4	6	8	**-6.051**	**-6.046**	-6.329	-6.145	**-6.040**
Plants	255	179	24	32	70	-12.890	**-12.853**	-16.633	-14.291	**-12.880**
Audio	130	108	25	42	59	-40.763	**-40.632**	-41.983	-41.380	**-40.697**
Jester	73	67	16	25	41	-53.897	**-53.528**	-54.881	-54.714	-53.919
Netflix	131	123	48	62	89	-58.234	**-58.021**	-59.752	-59.109	-58.4360
Accidents	623	96	18	31	34	-35.336	-35.635	-39.370	-34.593	**-29.002**
Retail	105	97	11	14	22	-11.245	-11.198	-11.290	-11.274	**-10.969**
Pumsb-star	256	245	18	25	34	-29.235	-29.485	-41.969	-29.689	**-23.282**
DNA	17	18	3	4	9	-97.876	-97.764	-99.123	-98.142	**-81.931**
Kosarek	193	152	22	38	52	-11.032	-11.033	-11.524	-11.058	**-10.724**
MSWeb	449	204	33	50	118	-10.100	-10.042	-10.966	-10.872	**-9.860**
Book	105	65	40	45	92	-35.416	-35.395	-35.616	-35.422	**-34.298**
EachMovie	58	51	22	35	82	**-51.753**	-52.232	-65.114	-55.781	**-51.512**
WebKB	76	24	29	36	77	-157.941	-158.202	-167.319	-165.716	**-154.909**
Reuters-52	184	90	58	85	341	-86.443	-85.949	-98.031	-96.469	**-84.090**
BBC	37	17	18	24	253	**-250.883**	**-250.783**	-269.084	-267.275	**-248.585**
Ad	183	165	52	67	350	-20.682	-22.379	-56.659	-35.785	**-15.792**

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 9, 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., WebKB and Reuters-52), 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 hyperparameter 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

Table 3
Positions in the learning times rank for GVS, EBVS, EBVS-AE, RGVS, WRGVS, RSBVS 50%.

	EBVS	EBVS-AE	RGVS	WRGVS	RSBVS 50%	GVS
NLTCS	6	4	1	3	5	2
Plants	6	4	1	3	5	2
Audio	6	2	1	3	5	4
Jester	4	3	1	2	6	5
Netflix	4	2	1	5	3	6
Accidents	6	4	1	2	5	3
Retail	6	5	1	2	4	3
Pumsb-star	5	4	1	2	6	3
DNA	4	3	1	2	6	5
Kosarek	6	5	1	2	4	3
MSWeb	6	5	1	2	4	3
Book	6	3	1	2	5	4
EachMovie	4	3	1	2	5	6
WebKB	4	1	2	3	6	5
Reuters-52	4	3	1	2	5	6
BBC	4	1	2	3	5	6
Ad	4	3	1	2	5	6
AVG	5.0	3.2	1.1	2.5	4.9	4.2

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	2	12
EBVS	8	-	0	0	1	8
EBVS-AE	10	17	-	2	4	14
RGVS	17	17	15	-	17	17
WRGVS	15	16	13	0	-	16
RSBVS 50%	5	9	3	0	1	-

Table 5
Positions in the accuracy rank for GVS, EBVS, EBVS-AE, RGVS, WRGVS, RSBVS 50%.

	EBVS	EBVS-AE	RGVS	WRGVS	RSBVS 50%	GVS
NLTCS	1	1	4	3	2	1
Plants	2	1	5	4	3	1
Audio	2	1	4	3	3	1
Jester	2	1	5	4	3	2
Netflix	2	1	6	4	5	3
Accidents	4	5	6	3	2	1
Retail	3	3	4	4	2	1
Pumsb-star	3	4	6	5	2	1
DNA	3	3	5	4	2	1
Kosarek	2	2	5	3	4	1
MSWeb	3	2	6	5	4	1
Book	3	2	5	4	5	1
EachMovie	1	2	5	4	3	1
WebKB	2	2	5	4	3	1
Reuters-52	3	2	6	5	4	1
BBC	2	2	6	5	4	2
Ad	2	3	6	5	4	1
AVG	2.7	2.5	5.9	4.7	3.8	1.4

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	17 / 0	17 / 0	17 / 0
EBVS	1 / 4	-	4 / 6	17 / 0	16 / 0	13 / 0
EBVS-AE	2 / 4	7 / 6	-	17 / 0	16 / 0	13 / 0
RGVS	0 / 0	0 / 0	0 / 0	-	0 / 1	0 / 1
WRGVS	0 / 0	1 / 0	1 / 0	16 / 1	-	3 / 1
RSBVS 50%	0 / 0	4 / 0	4 / 0	16 / 1	13 / 1	-

Table 7

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	316	677	1129	23	9	9	11	17	1190	331	58	49	271
Plants	42318	67821	2770	5177	15129	27	17	15	17	27	2304	1863	415	310	2635
Audio	36499	43243	2581	5056	17811	21	11	11	13	25	550	477	308	297	2736
Jester	26263	27609	1862	3183	12460	17	5	13	15	25	308	276	270	282	2071
Netflix	42033	44512	5097	8301	30417	21	11	15	15	31	503	465	737	209	4351
Accidents	98218	33377	1529	3127	11861	33	15	15	19	27	4315	436	240	133	2656
Retail	10096	6973	368	443	1010	41	37	7	9	19	446	258	31	20	175
Pumsb-star	32776	137092	1478	2339	12821	29	19	13	17	27	1941	1885	216	201	2679
DNA	8694	8694	475	738	3384	7	7	9	7	13	52	52	38	71	938
Kosarek	38097	42732	1130	1970	3692	31	41	13	15	23	1466	753	120	176	610
MSWeb	41447	57482	639	907	10341	35	35	7	9	33	1771	964	18	83	1824
Book	128181	65002	2311	2952	3814	29	33	9	11	11	1182	295	118	373	441
EachMovie	34155	31467	2659	3644	24458	21	25	11	17	29	485	328	317	549	3253
WebKB	61125	34863	2994	3925	9344	83	7	9	11	15	861	61	219	89	1239
Reuters-52	121211	83826	2275	4653	82084	103	21	9	13	27	1494	255	138	225	9922
BBC	39997	26167	2293	2387	68117	71	7	7	7	25	508	31	57	383	7057
Ad	90599	166802	3389	5213	20823	157	137	7	9	33	1231	894	94	238	1338

Table 8

Learning times in seconds and average test log-likelihoods 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	9.35	3.65	3.54	3.40	3.73	-6.196	-6.356	-6.576	-6.562	-6.337
Plants	45.18	21.25	18.84	21.09	26.49	-16.808	-17.786	-17.186	-16.292	-15.223
Audio	43.72	24.93	24.32	26.03	27.20	-41.866	-42.146	-42.206	-42.076	-41.702
Jester	34.33	17.85	18.77	21.51	30.95	-54.806	-54.984	-54.949	-54.866	-54.341
Netflix	67.51	54.72	51.29	49.53	72.75	-60.043	-59.832	-59.464	-59.559	-59.082
Accidents	38.43	18.10	19.55	20.80	30.40	-39.415	-39.871	-40.006	-38.759	-36.677
Retail	24.18	9.06	11.59	15.73	20.51	-11.382	-11.368	-11.358	-11.344	-11.142
Pumsb-star	24.60	21.60	18.83	19.99	29.50	-43.240	-45.732	-42.171	-37.525	-32.310
DNA	7.55	2.06	2.58	2.98	3.89	-99.453	-99.458	-98.920	-98.269	-95.968
Kosarek	11.82	13.59	12.58	8.10	9.96	-10.976	-10.926	-10.926	-10.926	-10.926
MSWeb	23.36	22.73	17.20	14.27	22.61	-10.115	-10.310	-10.115	-10.115	-10.115
Book	21.88	22.09	20.92	21.72	37.68	-35.549	-35.656	-35.662	-35.636	-35.571
EachMovie	7.30	5.87	7.48	8.17	6.34	-58.095	-58.339	-58.868	-58.295	-58.180
WebKB	15.81	9.89	12.75	13.06	17.38	-157.723	-159.217	-158.880	-156.547	-155.465
Reuters-52	17.12	24.28	31.51	67.34	44.18	-89.163	-89.840	-88.640	-90.714	-89.153
BBC	5.74	5.20	6.79	7.05	5.41	-257.590	-258.049	-257.184	-257.184	-257.184
Ad	7.93	5.94	11.90	46.25	104.57	-18.200	-22.621	-21.617	-18.755	-18.875

Table 9

Learning times in seconds and average test log-likelihoods 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	36.09	15.28	6.62	5.12	-6.078	-6.226	-6.322	-6.483
Plants	184.79	70.81	33.17	26.16	-13.659	-14.969	-16.772	-18.763
Audio	60.40	25.95	13.40	10.61	-41.550	-42.827	-44.178	-45.155
Jester	35.11	11.37	6.15	3.78	-54.212	-55.453	-57.369	-57.784
Netflix	47.52	28.19	24.62	8.66	-59.551	-61.152	-62.000	-62.952
Accidents	196.30	84.41	54.34	35.04	-32.511	-33.548	-34.657	-36.539
Retail	30.81	21.68	12.82	14.38	-11.052	-11.087	-11.265	-11.340
Pumsb-star	429.70	172.33	80.72	60.23	-27.463	-29.557	-31.744	-33.851
DNA	13.34	2.60	2.65	3.26	-83.644	-99.665	-99.665	-99.665
Kosarek	136.31	97.46	47.64	43.27	-11.185	-11.502	-11.684	-11.761
MSWeb	132.17	96.34	62.99	65.61	-10.349	-10.581	-10.757	-10.824
Book	99.84	72.96	54.14	57.37	-36.395	-36.955	-37.176	-37.176
EachMovie	59.84	34.74	27.19	30.80	-55.140	-59.143	-64.014	-67.121
WebKB	91.03	73.15	54.42	68.52	-163.292	-168.048	-169.456	-169.456
Reuters-52	187.65	140.01	123.69	139.94	-91.890	-95.322	-96.778	-97.466
BBC	68.65	61.61	54.17	58.77	-266.407	-269.355	-269.355	-269.355
Ad	196.06	159.42	132.01	112.31	-33.972	-52.284	-57.699	-57.760

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	1456	470	217	113	9	7	7	7	221	69	29	20
Plants	10983	3110	1235	597	19	15	19	11	844	252	98	58
Audio	3287	883	358	215	13	13	7	5	348	101	53	7
Jester	2519	645	258	207	15	13	5	5	318	67	21	4
Netflix	3009	881	381	247	13	11	9	5	408	146	45	19
Accidents	5441	1656	725	490	19	13	9	9	1070	353	145	83
Retail	481	392	294	280	7	9	5	5	45	31	7	5
Pumsb-star	19571	5920	2442	1674	25	23	19	17	1264	520	229	167
DNA	1542	362	362	362	9	3	3	3	373	2	2	2
Kosarek	4168	751	496	441	31	13	9	5	239	66	39	23
MSWeb	1634	839	681	627	19	13	7	5	187	69	26	14
Book	2066	1111	1002	1002	13	7	3	3	151	33	2	2
EachMovie	4232	1382	1077	1015	17	11	7	5	188	32	16	6
WebKB	2823	1758	1680	1680	15	5	3	3	320	29	2	2
Reuters-52	4552	2169	1852	1787	17	11	7	5	489	88	20	4
BBC	2519	2118	2118	2118	9	3	3	3	102	2	2	2
Ad	5718	3333	3134	3114	9	7	5	3	388	48	8	2

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 and Ad.

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.

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 sin-

gle 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 7). 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, 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 and performs better than RGVS (Table 2). 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). 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) but gaining a comparable accuracy on most of them while learning a smaller network (Accidents and EachMovie)–see Table 7. On many datasets our adaptive version of EBVS is capable to learn smaller networks than EBVS (see Table 7) especially on datasets with a high number of RVs (Book, EachMovie, WebKB, Reuters-52, BBC, Ad).

In general, EBVS-AE compared to EBVS learns networks with a considerable smaller number of weigh-ts, 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 2). 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.

Table 5 and Table 6 (resp. Table 3 and Table 4) report the cumulative results in terms of accuracy (resp. learning time) for all methods in form of rankings.

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.

References

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

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

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

[4] Wei-Chen Cheng, Stanley Kok, Hoai Vu Pham, Hai Leong Chieu, and Kian Ming Adam Chai. Language modeling with Sum-Product Networks. In *INTERSPEECH 2014*, pages 2098–2102, 2014.

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

[6] Aaron Dennis and Dan Ventura. Learning the Architecture of Sum-Product Networks Using Clustering on Varibles. In *NIPS 25*, pages 2033–2041, 2012.

[7] Nicola Di Mauro, Floriana Esposito, Fabrizio G. Ventola, and Antonio Vergari. Alternative variable splitting methods to learn sum-product networks. In *Proceedings of AIXIA*. Springer, 2017.

[8] Nicola Di Mauro, Antonio Vergari, and Teresa M.A. Basile. Learning bayesian random cutset forests. In *ISMIS*, pages 122–132. Springer, 2015.

[9] Nicola Di Mauro, Antonio Vergari, and Floriana Esposito. Learning accurate cutset networks by exploiting decomposability. In *AIXIA*, pages 221–232. Springer, 2015.

[10] Abram Friesen and Pedro Domingos. The sum-product theorem: A foundation for learning tractable models. In *ICML*, pages 1909–1918, 2016.

[11] Robert Gens and Pedro Domingos. Learning the Structure of Sum-Product Networks. In *ICML*, pages 873–880, 2013.

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

[13] Wilson Hsu, Agastya Kalra, and Pascal Poupart. Online structure learning for sum-product networks with gaussian leaves. *arXiv preprint arXiv:1701.05265*, 2017.

[14] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[15] Daniel Lowd and Jesse Davis. Learning Markov network structure with decision trees. In *ICDM*, pages 334–343. IEEE Computer Society Press, 2010.

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

[17] James Martens and Venkatesh Medabalimi. On the Expressive Efficiency of Sum Product Networks. *CoRR*, abs/1411.7717, 2014.

[18] Mazen Melibari, Pascal Poupart, Prashant Doshi, and George 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*, pages 345–355, 2016.

[19] Alejandro Molina, Sriraam Natarajan, and Kristian Kersting. Poisson sum-product networks: A deep architecture for tractable multivariate poisson distributions. In *AAAI*, 2017.

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

[21] Robert Peharz, Bernhard Geiger, and Franz Pernkopf. Greedy Part-Wise Learning of Sum-Product Networks. In *ECML-PKDD 2013*, 2013.

[22] Robert Peharz, Robert Gens, and Pedro Domingos. Learning selective sum-product networks. In *Workshop on Learning Tractable Probabilistic Models*. LTPM, 2014.

[23] Robert Peharz, Georg Kapeller, Pejman Mowlaee, and Franz Pernkopf. Modeling speech with sum-product networks: Application to bandwidth extension. In *ICASSP*, 2014.

[24] Hoifung Poon and Pedro Domingos. Sum-Product Networks: a New Deep Architecture. *UAI 2011*, 2011.

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

[26] Tahrima Rahman and Vibhav Gogate. Merging strategies for sum-product networks: From trees to graphs. In *UAI*, pages ??–??, 2016.

[27] Tahrima Rahman, Prasanna Kothalkar, and Vibhav Gogate. Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of Chow-Liu trees. In *ECML-PKDD*, pages 630–645. Springer, 2014.

[28] Amirmohammad Rooshenas and Daniel Lowd. Learning Sum-Product Networks with Direct and Indirect Variable Interactions. In *ICML*, 2014.

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

[30] Martin Trapp, Robert Peharz, Marcin Skowron, Tamas Madl, Franz Pernkopf, and Robert Trappl. Structure inference in sum-product networks using infinite sum-product trees. 12 2016.

[31] Antonio Vergari, Nicola Di Mauro, and Floriana Esposito. Simplifying, Regularizing and Strengthening Sum-Product Network Structure Learning. In *ECML-PKDD*, 2015.

[32] Antonio Vergari, Nicola Di Mauro, and Floriana Esposito. Visualizing and understanding sum-product networks. *CoRR abs/1608.08266*, 2016.

[33] Zehuan Yuan, Hao Wang, Limin Wang, Tong Lu, Shivakumara Palaiahnakote, and Chew Lim Tan. Modeling spatial layout for scene image understanding via a novel multiscale sum-product network. *Expert Systems with Applications*, 63:231 – 240, 2016.