Fast And Accurate Learning of Probabilistic Circuits by Random Projections

Renato Lui Geh[1] **Denis Deratani Mauá**[1]

[1]Department of Computer Science, Institute of Mathematics and Statistics, University of São Paulo, Brazil

Abstract

We revisit random projection trees in the context of probabilistic circuits. We show how a recursive partitioning scheme for inducing oblique kd-trees from data using random projections can be adapted to produce reasonably accurate probabilistic circuits in a fraction of the time used by typical structure learning algorithms.

1 INTRODUCTION

Probabilistic circuits unify several frameworks for tractable probabilistic modeling such as low-treewidth graphical models [Bach and Jordan, 2001, Liu et al., 2011, Elidan and Gould, 2009], sum-product networks [Poon and Domingos, 2011], probabilistic sentential decision diagrams [Kisa et al., 2014] and cutset networks [Rahman et al., 2014].

State-of-the-art algorithms for learning probabilistic circuits from data can be categorized into two different approaches. The first approach, exemplified by LearnSPN [Gens and Domingos, 2013], grows a probabilistic circuit in a top-down fashion by recursively splitting data instances into similar clusters and partitioning variables into approximately independent sets. The second approach, exemplified by LearnPSDD [Liang et al., 2017], performs a greedy search that optimizes some score measure (typically, a penalized log-likelihood) by iteratively performing local transformations of the circuit (e.g. splitting and merging nodes). Although both approaches produce accurate probabilistic models, with LearnPSDD exhibiting superior overall performance in benchmark experiments, they take significant computational resources (time and memory). The situation is aggravated by the fact that achieving top quality models usually requires grid-search for hyperparameter tuning, gradient-based parameter learning and the use of mixtures of circuits by bagging. And while some recent work has shown that such a cost can be decreased at a small or even no loss in accuracy [Mei et al., 2018, Jaini et al., 2018, Di Mauro et al., 2017, Dang et al., 2020], the computational costs of learning algorithms discourage the use of probabilistic circuits to exploring massive datasets of very high dimension.

Recently, Correia et al. [2020] showed that (ensembles of) decision trees learned for prediction tasks can be easily extended into full probabilistic models represented as probabilistic circuits. Besides equipping (ensembles of) decision trees with more principled approaches to handling missing data and diagnosing outliers, the connection of decision trees and probabilistic circuits suggests an interesting alternative to learning the latter using the efficient inductive algorithms available for the former [Correia et al., 2020, Ram and Gray, 2011, Khosravi et al., 2020].

Typical algorithms for inducing decision trees from data consist in recursively partitioning the data into axis-aligned cells, that is, they split the data according to the value of a single variable at a time. Freund et al. [2008] noted that such an approach cannot ensure that the resulting partitioning of the input space approximates the intrinsic dimensionality of the data (roughly understood as a manifold of low dimension). In constrast, they provide a simple strategy for space partitioning that consists in recursively partitioning the space according to a random separating hyperplane. The result approximates a random projection of the data and has the following theoretical guarantee [Freund et al., 2008]:

> If the data has intrinsic dimension d, then with constant probability the part of the data at level d or higher of the tree has average diameter less than half of that of the data.

Accordingly, the depth of the tree needs only to grow proportionally to the intrinsic dimension and not to the number of variables. In addition to that and to other theoretical insurances [Dhesi and Kar, 2010], the recursive partitioning scheme proposed is extremely fast, taking linearithmic time in the dataset size (no. of instances and no. of variables).

Accepted for the 4th Workshop on Tractable Probabilistic Modeling (TPM 2021).

In this work, we revisit random projection trees in the context of probabilistic circuits. More specifically, we develop different approaches for learning probabilistic circuits using recursive partitioning by random projections. Our approaches differ in how random projections and ensembles are combined. Experiments with benchmark datasets for density estimation show that our proposed approaches are competitive with state-of-the-art algorithms (LearnSPN, LearnPSDD, Strudel), while requiring a fraction of the learning times of the competitors.

2 RANDOM PROJECTION TREES

kd-trees [Bentley, 1979] are data structures that enable efficient similarity search and other operations by representing data as a hierarchy of partitions represented as a tree. They are typically obtained from data via axis-aligned binary recursive partitioning as follows. Given a rule $r : X \to \{0, 1\}$, a dataset S is split into $S_1 = \{x \in X : r(x) = 0\}$ and $S_2 = \{x \in X : r(x) = 1\}$, and the process repeats until $|S|$ is sufficiently small. Typically, the rule r selects the variable with the largest variance (or some other measure of spread) in S and separates instances according to the median value of that variable in S. The process is similar to the induction of decision trees, except that in this case the rules discriminate against a target variable [Breiman, 2001].

The statistical properties of estimates obtained from the instances at the leaves of a kd-tree depend on the rate at which the diameter of the partitions are reduced once we move down the tree. For a space of dimension n, a kd-tree induced by the process described might require n levels to halve the diameter of the original data [Dasgupta and Freund, 2008]. This is true even for datasets of low intrinsic dimension. The latter is variously defined, and different definitions lead to different theoretical properties. A common surrogate metric is the *doubling dimension* of a dataset $S \subset \mathbb{R}^n$, given by the smallest integer d such that the intersection of S and any ball of radius r centered at $x \in S$ can be covered by at most 2^d balls of radius $r/2$ [Dhesi and Kar, 2010].

Algorithm 1 SPLITSID

Input Dataset $S \subset \mathbb{R}^n$
Output A partition S_1, S_2 of S

Let m be the number of examples in S
Sample a random unit direction w
Sort $a = w \cdot x$ for $x \in S$ s.t. $a_1 \leq a_2 \leq \cdots \leq a_m$
for $i = 1, \ldots, m-1$ **do**
    $\mu_1 = \frac{1}{i} \sum_{j=1}^{i} a_i, \quad \mu_2 = \frac{1}{m-i} \sum_{j=i+1}^{m} a_i$
    $c_1 = \sum_{j=1}^{i} (a_j - \mu_1)^2 + \sum_{j+1}^{m} (a_j - \mu_2)^2$
Find i that minimizes c_i and set $\theta = (a_i + a_{i+1})/2$
$S_1 \leftarrow \{x \in S : w \cdot x \leq \theta\}$
return $(S_1, S \setminus S_1)$

Algorithm 2 SPLITMAX

Input Dataset $S \subset \mathbb{R}^n$ and constant r
Output A partition S_1, S_2 of S

Sample a random unit direction w
Pick any $x \in S$ and let y be x's farthest point in S
Sample δ uniformly in $[-c, c]$, where $c = r \cdot \text{dist}(x, y)/\sqrt{n}$
$S_1 \leftarrow \{x \in S : w \cdot x \leq \text{median}(\{z \cdot w : z \in S\}) + \delta\}$
return $(S_1, S \setminus S_1)$

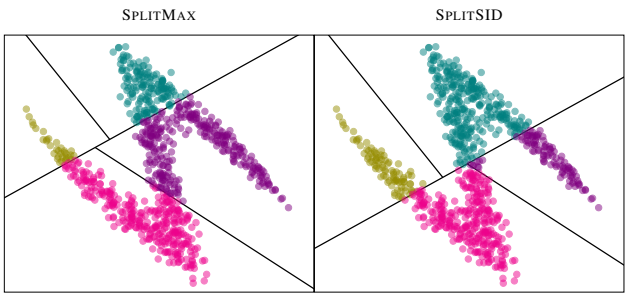

Figure 1: Example of space partitioning by RPTrees grown using different split rules but the same random directions.

Random Projection Trees (RPTrees) are a special type of kd-trees that split along a random direction of the space. Two such splitting rules are given by Algorithms 1 and 2, where in the latter dist refers to the Euclidean distance. The intuition behind either rule is to generate a random hyperplane (unit direction) and then find a threshold projection value that roughly divides dataset S into two approximately equally sized subsets. Algorithm 1 attempts at finding such a threshold by minimizing the average squared interpoint distance (SID), while Algorithm 2 uses a random noise proportional to the average diameter of S. As discussed by Dasgupta and Freund [2008] and by Freund et al. [2008], either optimizing or randomizing the threshold leads to better separation of the data than simply selecting the median point. Figure 1 shows an example of space partitioning induced by 2-level RPTrees using each of the rules with the same direction vectors w. Note that the rules produce quite different splits despite using the same random directions.

Unlike standard kd-trees, RPTrees ensure that, for a data with doubling dimension d, at most d levels are necessary to half the diameter of the data, irrespective of its dimension. This leads to improved statistical properties that are connected to that notion of low intrinsic dimensionality [Dasgupta and Freund, 2008, Dhesi and Kar, 2010].

3 PROBABILISTIC CIRCUITS

A *probabilistic circuit* is either (i) a tractable distribution $P(\mathbf{X})$ with scope $\mathbf{X}$ (i.e., a distribution that admits efficient marginalization of any subset $\mathbf{Y} \subseteq \mathbf{X}$ of its variables), (ii)

a convex combination $\sum_i \lambda_i P_i(\mathbf{X})$, $\lambda_i > 0, \sum_i \lambda_i = 1$, of probabilistic circuits $P_i(\mathbf{X})$ with identical scope $\mathbf{X}$, or (iii) a product $\prod_i P_i(\mathbf{X}_i)$ of probabilistic circuits with disjoint scopes $\mathbf{X}_i$.

Probabilistic circuits are more commonly represented as rooted acyclic directed graphs with sum ($+$) and product ($\times$) inner nodes, and distribution leaves [Gens and Domingos, 2013, Dang et al., 2020]. The arcs from sum nodes are weighted by non-negative values. The key feature of probabilistic circuits is that marginal probabilities can be computed in linear time in the size of any probabilistic circuit (plus the time to marginalize leaves). For leaf distributions in the exponential family, they also allow for the parameters to be efficiently learned by Expectation-Maximization or other gradient-based approaches [Peharz et al., 2016, Zhao et al., 2016, Desana and Schnörr, 2016].

Generative trees [Correia et al., 2020] are probabilistic circuits obtained from decision tree classifiers as follows. Each decision node (split) of the decision tree is transformed into a sum node whose weights are given by the proportion of instances in the respective partitioning, and leaves of the decision trees are associated to distributions learned from the corresponding subsets, and truncated so that their support is the space defined by the splits from the root of the decision tree up to the corresponding leaf. While in principle any distribution is valid for the leaves, the authors experiment either with simple fully factorized distributions, or with probabilistic circuits learned by standard algorithms such as LearnSPN [Gens and Domingos, 2013]. A generative forest is obtained simply as a mixture of generative trees (which in the context of probabilistic circuits simply means adding a top root node whose children are generative decision trees).

4 STRUCTURE LEARNING BY RANDOM PROJECTIONS

In their work, Correia et al. [2020] focus on decision trees induced by axis-aligned splits. Here, we investigate the use of oblique decision trees obtained by random projections to learning probabilistic circuits. Unlike Generative Forests, we do not constrain the support of the distribution at a leaf by the corresponding space partition. Instead, we learn distributions from the entire training dataset with full support, which leads to more efficient learning and inference. In this sense, our approach is more similar to LearnSPN, except that we replace clustering by random projections and do not run (costly) independence tests, and differs from other uses of decision trees with density estimators at the leaves (e.g. [Smyth et al., 1995]).

In a nutshell, our approach constructs a probabilistic circuit in a top-down fashion by recursively partitioning data through random projections and mapping splits to sum nodes. The recursion stops when data size falls below some pre-specified threshold, at which point a multivariate distribution is learned. We present two variants of that procedure. The first variant, called LEARNRP-S, is akin to generative forests, and learns a mixture of independent tree-shaped circuits obtained from recursively splitting the data by random projections. Pseudo-code Algorithm 3 describes the procedure in greater detail. The algorithm returns a mixtures of k probabilistic circuits learned from recursively splitting the dataset using either SPLITMAX or SPLITSID (only one such rule is used for building the entire circuit). To improve the quality of the random projections, we generate t splits and keep only the best according to average diameter criterion. A template subcircuit is learned once the respective dataset S is sufficiently small. For simplicity, in this paper we only consider two such templates: one where a fully factorized distribution is learned at the leaves, that is, LEARNDISTRIBUTION(S) returns a product of univariate distributions (e.g., Gaussian or Bernoulli distributions); and the other where a RAT-SPN [Peharz et al., 2019] is used as a multivariate leaf distribution.

Algorithm 3 LEARNRP-S

Input Dataset $S \subset \mathbb{R}^n$, no. of trials t, no. of trees k
Output A probabilistic circuit

if it is the first recursion **then**
  return $\sum_{i=1}^{k} \frac{1}{k}$LEARNRP-S$(S, t, k)$
else
  Sample t splits by some criteria, and select the split (S_1,S_2) that minimizes the average diameter of S
  if $|S_1|$ is small **then**
    $P_1 \leftarrow$ LEARNDISTRIBUTION(S_1)
  else $P_1 \leftarrow$ LEARNRP-S(S_1, t, k)
  if $|S_2|$ is small **then**
    $P_2 \leftarrow$ LEARNDISTRIBUTION(S_2)
  else $P_2 \leftarrow$ LEARNRP-S(S_2, t, k)
  Set $\lambda \leftarrow |S_1|/|S|$
  return $\lambda \cdot P_1 + (1 - \lambda) \cdot P_2$

The second variant, LEARNRP, instead alternates between levels of mixtures and random projection splits. The procedure is described in the pseudo-code in Algorithm 4. As shown, at each recursion the procedure creates k components C_i, each obtained by selecting the best split (S_1, S_2) out of t candidate splits (obtained with some fixed split rule) then either learning a distribution for the respective part or recurring.

Figure 2 illustrates the process for both variants: (a) shows the initial mixture of projections represented by the top root node, and a fully factorized distribution represented in the leftmost sub-circuit. In (b), a sum node representing a mixture of k recursive calls of LEARNRP-S is shown. In (c), a probabilistic circuit generated by LEARNRP alternates between sum nodes with k children, representing fine mixtures, and sum nodes with 2 children representing random

Algorithm 4 LEARNRP

Input Dataset $S \subset \mathbb{R}^n$, no. of trials t, no. of mixtures k
Output A probabilistic circuit

for $i = 1, \ldots, k$ **do**
  Sample t splits by some criteria, and select the split (S_1, S_2) that minimizes the average diameter of S
  if $|S_1|$ is small **then**
    $P_1 \leftarrow$ LEARNDISTRIBUTION(S_1)
  else $P_1 \leftarrow$ LEARNRP(S_1, t, k)
  if $|S_2|$ is small **then**
    $P_2 \leftarrow$ LEARNDISTRIBUTION(S_2)
  else $P_2 \leftarrow$ LEARNRP(S_2, t, k)
  Set $\lambda \leftarrow |S_1|/|S|$
  Compute $C_i \leftarrow \lambda \cdot P_1 + (1 - \lambda) \cdot P_2$
return $\sum_{i=1}^{k} \frac{1}{k} C_i$

projection splits. As the examples suggest, either variant produces a tree-shaped probabilistic circuit containing only sum nodes and distribution leaves (which in turn can be represented as more probabilistic circuits containing product nodes).

Since SPLITMAX and SPLITSID take time $\mathcal{O}(m \log m)$ and $\mathcal{O}(m)$, respectively, where m is the number of examples in the dataset, both variants are extremely fast and scale for very large datasets of very high dimension. In fact, as we later show in Section 5, LEARNRP takes a fraction of the time competitors take to learn.

The accuracy of the learned circuits can be improved by fine-tuning the parameters to the entire dataset, using standard parameter learning algorithms for probabilistic circuits.

5 EXPERIMENTS

We evaluate the performance of learning probabilistic circuits via random projections with respect to average held-out log-likelihood and running time on both synthetic datasets and realistic benchmark datasets. The former allows us to better investigate the differences among the several choices of split functions and hyperparameters; the latter allows to compare performance against state-of-the-art techniques with published results.

By combining procedures LEARNRP or LEARNRPS and the split rules SPLITMAX or SPLITSID, we obtain four different learning algorithms, denoted as RPRULE and RPRULES in the following, for RULE $\in \{\text{MAX}, \text{SID}\}$. These use a fully factorized circuit as leaf, while RPRULED and RPRULESD, with their respective rules, indicate that a dense random circuit (RAT-SPN) is used as leaves instead. After obtaining a probabilistic circuit, we fine-tune the parameters (weights and mean and variance for Gaussian leaves, if any) either by (standard batch) EM or by an online version that updates parameters using minibatches as:

$$\theta_{t+1} \leftarrow \eta_t \cdot \theta_{\text{EM}} + (1 - \eta_t)\theta_t \,, \tag{1}$$

where θ_{EM} is the standard EM update using a minibatch dataset, and η_t is the learning rate. In the experiments we use exponential decayed learning rates $\eta_t = (\eta_0)^t$, with η_0 varying in $[0.9, 0.99]$. In our experiments with medium-sized datasets, the online version showed improved generalization performance and increased numerical accuracy, while being much faster (in terms of overall runtime). We thus used only the online variant for the larger benchmark datasets.

The algorithms were implemented in the Julia language, and the source code is available at `https://github.com/RenatoGeh/RPCircuits.jl`. The experiments were carried out on a single computer with a 12-core Intel i7 3.7 GHz processor and 64GB RAM.

5.1 SYNTHETIC DATASETS

We first show experiments with an intricate two-dimensional dataset, which allow us to visualize the estimated models. The `2-moons` dataset is a commonly used sampling procedure consisting of two intertwining "half-moon" structures. For our experiments, we generated 1,000 points, of which 600 were kept for training, 100 for validation and 300 as test set. We further added some complexity to the data by adding a Gaussian noise of 0.1.

We used $t = 5$ for the number of candidate splits to be selected at each projection, $k = 3$ for the number of projections per mixture for the LEARNRP variant and $k = 30$ for the simpler one-mixture LEARNRP-S approach. We ran EM, both online and batch, for 100 iterations with a minibatch of 100 for the former. We set $r = 1$ for the MAX splits. Throughout all experiments, we considered the data to be small (and thus learn a fully factorized distribution over them) if they contained 30 instances or fewer.

Table 1 compares the held-out average log-likelihood of the different random projection learning strategies on the `2-moons` dataset, for different configurations of the minimum variance at the Gaussian distributions (which acts as a regularizer) and different uses of parameter learning (the standard batch EM and an online version of EM that uses minibatches). In the table, a fixed minimum variance corresponds to the case where the parameters of the Gaussian leaves are not updated during parameter learning. Note how lowering the constraint on variance improves log-likelihood on this set for all approaches. Interestingly, LEARNRP does a good job at learning the leaves by itself, sometimes showing better performance compared to EM when learning leaves. Figures 3 and 4 depict the Gaussian distributions (in red) and the overall densities of each configuration for the LEARNRP variant. The plots suggests that lowering variance increases chances of overfitting. For each value of

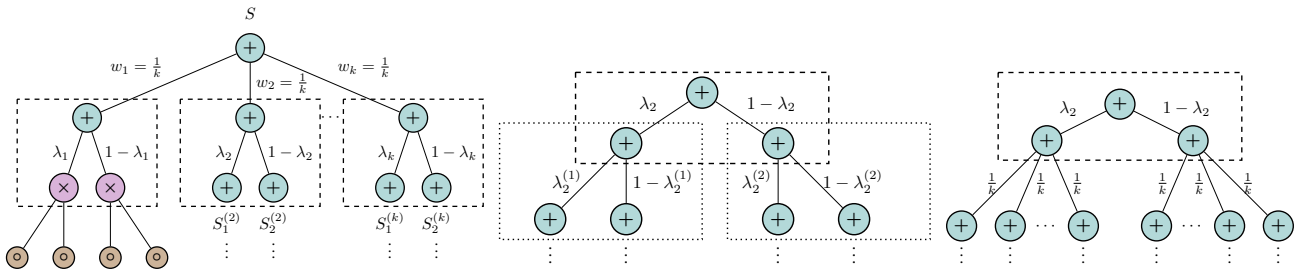

Figure 2: Example of probabilistic circuits learned by LEARNRP and LEARNRP-S. The dotted and dashed boxes represent fragments learned by random projection splits, while the remaining sum nodes are obtained by repetitions to induce mixtures. In (a), the fully grown circuit on the left is a result of both partition splits being sufficiently small. (b) LEARNRP-S recursively learns sum nodes by binary splits. (c) LEARNRP alternates between binary splits and k-ary sum nodes representing mixtures.

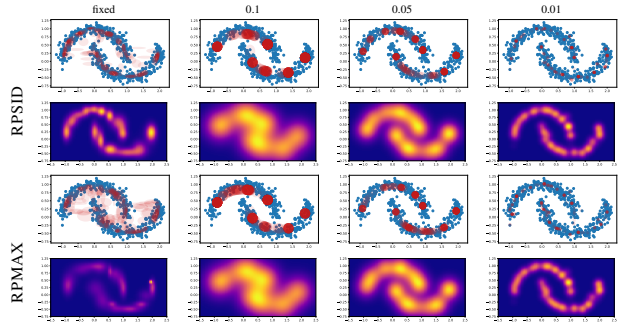

Figure 3: Gaussian components and densities of *batch* EM learned circuits on the `2-moons` dataset.

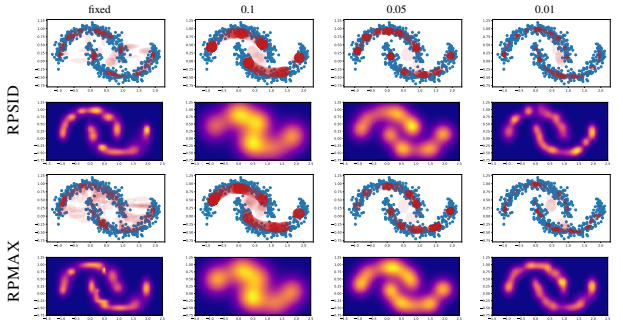

Figure 4: Gaussian components and densities of *online* EM learned circuits on the `2-moons` dataset.

EM	$\sigma^2 \geq$	RPMaxS	RPMax	RPSIDS	RPSID
Batch	fixed	-1.083	-1.139	-1.106	-1.074
	0.10	-1.616	-1.616	-1.616	-1.616
	0.05	-1.387	-1.388	-1.387	-1.385
	0.01	-1.034	-1.041	-1.035	-1.032
Online	fixed	-1.089	-1.119	-1.080	-1.108
	0.10	-1.623	-1.627	-1.624	-1.637
	0.05	-1.416	-1.402	-1.408	-1.412
	0.01	-1.139	-1.080	-1.114	-1.105

Table 1: Average log-likelihood for the `2-moons` dataset for different choices of minimum Gaussian variance σ^2.

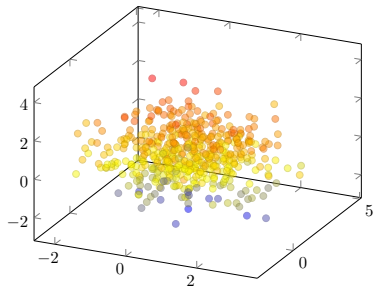

Figure 5: Example of synthetic dataset generated as in Freund et al. [2008].

minimum variance and EM version, the different approaches achieve comparable performance, with no clear winner. Results also show that online EM learns slightly less accurate models for this simple domain.

We also evaluate our approaches on a synthetic dataset generated by the following sampling strategy proposed by Freund et al. [2008]. To generate an n-dimension point x, sample $p \sim [0, 1]$ uniformly at random, then sample each coordinate $x_i \sim \mathcal{N}(p, 1)$. Figure 5 shows such a generated dataset for $n = 3$. Table 2 shows a comparison between the random projection approaches and LEARNSPN for $n = 10$.[1] LEARNSPN's behavior is notably unusual, and we are unsure as to why it behaves so poorly. A possible explanation could be due to an overfitting of the Gaussian distributions at the leaves, as we found that LEARNSPN's clustering approach often reduced sub-datasets to size 1. We use the same parameters as the `2-moons` dataset, but with only 30 iterations for EM. Note how online EM outperforms the batch version and seems to prevent overfitting.

[1] We use the python implementation of LearnSPN available at `https://gitlab.com/pgm-usp/pyspn`.

EM	$\sigma^2 \geq$	RPMaxS	RPMax	RPSIDS	RPSID	LearnSPN
Batch	fixed	-14.483	-14.609	-14.469	-14.702	-213.100
	0.10	-21.726	-25.712	-21.694	-25.519	-25.621
	0.05	-27.308	-41.701	-27.022	-41.613	-44.335
	0.01	-33.114	-117.338	-34.061	-142.422	-213.100
Online	fixed	-14.468	-14.588	-14.498	-14.660	-213.331
	0.10	-15.353	-15.284	-15.614	-14.972	-14.716
	0.05	-14.943	-14.892	-15.090	-15.472	-14.776
	0.01	-15.855	-14.831	-14.872	-15.017	-14.740

Table 2: Average log-likelihood for the dataset generated as in Freund et al. [2008] with $n = 10$.

5.2 BENCHMARK DATASETS

We further evaluate the performance of LEARNRP in the discrete domain, more specifically under binary variables. We show results for the well-known 20 datasets benchmark for density estimation [Lowd and Davis, 2010, Van Haaren and Davis, 2012] [2].

For these 20 benchmark datasets, we set $t = 10$, $k = 2$ for LEARNRP and $k = 3$ for LEARNRP-S, ran for 100 EM iterations with minibatches of 500 for the online version, and set $r = 1$ for MAX.

In Table 3, we compare average log-likelihood values of our approaches against reported results for state-of-the-art methods: LEARNSPN [Gens and Domingos, 2013], EXPC [Mauro et al., 2021], STRUDEL and LEARNPSDD [Dang et al., 2020]. For the last two, we report the best values from mixtures learned through a combination of both EM and bagging; and for EXPC, only best values for non-deterministic structure decomposable ensembles are considered. Despite never beating competitors, our approaches fair reasonably well considering their simplicity.

Table 4 shows the size of the probabilistic circuits learned by the different approaches. Here we show single model sizes for STRUDEL, LEARNPSDD and EXPC, since node counts for mixtures and ensembles are left unreported in the original texts. Similarly, since there is no information on circuit size for the original results for LEARNSPN, we ran the algorithm on the previously mentioned Python implementation and reported their circuit size.

Comparatively, we see that RPSIDS often generates the smallest circuits among all methods, followed by RPMAXS. The other approaches using random projections often produce circuits larger than LEARNSPN, STRUDEL, EXPC and LEARNPSDD, especially when using RAT-SPNs as leaves. The repeated values of sizes for the random projection approaches are due to a hard limit on the maximum depth of the RPTREE, as well as to the requirement of a minimum number of instances at the leaves. Comparing the difference in size between the single mixture (S) and multiple mixtures with the difference in accuracy (held-out log-likelihood) in Table 3, we see that the single mixture versions achieve near the same performance of their multiple mixture counterparts with considerably smaller circuit complexity.

The run-times of each algorithm are given in Table 5. These times measure only the structure learning part and do not contain the parameter tuning, which for our methods is the most time consuming part. This is because parameter tuning is a common stage for all methods, and is very affected by better implementation (CPU parallelization, GPU usage, etc). As expected, circuits learned by random projections (hence with little optimization) are up to 3 orders of magnitude faster. For both STRUDEL and LEARNPSDD, since we do not know exactly which parameters were used for the best models reported in Dang et al. [2020], we show the time taken using 1000 iterations and with no parameter learning. For our approaches, as well as STRUDEL and LEARNPSDD, we measured results run on the Julia programming language. For LEARNSPN, we measured time on a Rust implementation[3].

6 CONCLUSION

We revisited random projection tress in the context of probabilistic circuits. We showed how a simple and fast scheme previously proposed to generate an oblique recursive partitioning of a dataset by means of random projections can be used to learn accurate probabilistic circuits. Experiments with benchmark datasets showed that our approach is competitive with state-of-the-art learners while taking a fraction of the time.

This work is certainly preliminary, and can be extended in several ways. We have only used fully factorized or dense randomized circuits distributions at the leaves of the learned circuits, which certainly decreases their expressivity. There are several simple alternatives such as learning Chow-Liu Trees [Chow and Liu, 1968] or using standard algorithms for learning probabilistic circuits. While either approaches increase the representation power, they also introduce additional complexity.

Another interesting avenue to explore are the theoretical guarantees of random projection trees. Can we use those results to prove lower bounds on the sample complexity of circuits generated by the proposed approaches? Can we prove other desirable statistical properties?

Arguably, the empirical analysis carried out here is lacking. We have yet to investigate accuracy in our approaches in more diverse settings involving continuous variables and specific domains (e.g. images). Finally, we might evaluate the approaches on end tasks (expected prediction, imputation, outlier detection) or with respect to different metrics

[2] Available at `https://github.com/UCLA-StarAI/Density-Estimation-Datasets`

[3] `https://gitlab.com/marcheing/spn-rs`

Dataset	#Var.	#Train.	RPMaxSD	RPMaxD	RPSIDSD	RPSIDD	RPMaxS	RPMax	RPSIDS	RPSID	LSPN	Strudel	LPSDD	EXPC
ACCIDENTS	111	12758	-36.77	-36.67	-36.96	-36.47	-37.41	-37.48	-37.16	-37.39	-30.03	**-28.73**	|-30.16|	-31.02
AD	1556	2461	-36.58	-36.19	-36.14	-35.88	-33.40	-32.83	-34.42	-35.55	|-19.73|	-16.38	-31.78	**-15.50**
AUDIO	100	15000	-40.25	-40.25	|-40.20|	-40.20	-40.29	-40.25	-40.28	-40.23	-40.50	-41.50	**-39.94**	-40.91
BBC	1058	1670	-253.15	|-251.71|	-252.48	-252.65	-254.71	-254.57	-254.74	-254.99	-250.68	-254.41	-253.19	**-248.34**
NETFLIX	100	15000	-57.20	-57.21	-57.22	-56.89	-57.53	-57.40	-57.45	-57.44	|-57.02|	-58.69	**-55.71**	-57.58
BOOK	500	8700	-34.84	-34.92	-34.87	-34.82	|-34.75|	-34.85	-34.74	**-34.66**	-35.88	-34.99	-34.97	-34.75
20-NEWSGRP	910	11293	-154.20	-154.58	|-153.91|	**-153.51**	-155.39	-155.41	-155.59	-156.08	-155.92	-154.47	-155.97	-153.75
REUTERS-52	889	6532	-87.01	-86.70	-86.79	-86.56	-87.58	-87.23	-86.33	-87.28	-85.06	|-86.22|	-89.61	**-84.70**
WEBKB	839	2803	-157.49	-157.66	|-157.06|	-157.07	-158.46	-158.72	-158.07	-157.94	-158.20	-155.33	-161.09	**-153.67**
DNA	180	1600	-97.89	-96.86	-97.28	-97.64	-97.45	-97.17	-97.47	-96.68	**-82.52**	-86.22	-88.01	|-86.61|
JESTER	100	9000	|-53.05|	-52.99	-53.09	-53.05	-53.21	-53.24	-53.13	-53.06	-75.98	-55.03	**-51.29**	-53.43
KDD	65	180092	-2.17	-2.18	-2.17	-2.18	-2.16	-2.16	-2.16	-2.17	-2.18	-2.13	**-2.11**	|-2.15|
KOSAREK	190	33375	-11.11	-11.14	-11.14	-11.15	-11.06	-11.07	-11.02	-11.08	-10.98	-10.68	**-10.52**	|-10.77|
MSNBC	17	291326	-6.24	-6.24	-6.25	-6.32	-6.18	-6.18	-6.20	-6.28	|-6.11|	**-6.04**	-6.04	-6.18
MSWEB	294	29441	-10.51	-10.55	-10.53	-10.59	-10.25	-10.26	-10.25	-10.29	-10.25	**-9.71**	-9.89	|-9.93|
NLTCS	16	16181	-6.02	-6.02	-6.03	-6.05	|-6.01|	-6.01	-6.01	-6.01	-6.11	-6.06	**-5.99**	-6.05
PLANTS	69	17412	-13.94	-14.06	-14.00	-13.96	-14.07	-13.86	-14.02	-13.94	**-12.97**	-12.98	|-13.02|	-14.19
PUMSB-STAR	163	12262	-33.53	-34.23	-33.55	-32.73	-34.35	-34.24	-34.53	-33.92	-24.78	**-24.12**	-26.12	|-26.06|
EACHMOVIE	500	4524	-53.03	|-52.93|	-52.94	-52.96	-53.03	-53.28	-52.88	-53.15	**-52.48**	-53.67	-58.01	-54.82
RETAIL	135	22041	-11.02	-11.00	-11.01	-11.03	|-10.93|	-10.93	-10.94	-10.93	-11.04	-10.81	**-10.72**	-10.94

Table 3: Dataset characteristics and average held-out log-likelihood on the bechmark datasets. Boldface entries indicate best results, underline means second best and an entry in $|\cdot|$ indicates third place. LSPN and LPSDD stand for LEARNSPN and LEARNPSDD, respectively.

Dataset	RPMaxSD	RPMaxD	RPSIDSD	RPSIDD	RPMaxS	RPMax	RPSIDS	RPSID	LSPN	Strudel	LPSDD	EXPC
ACCIDENTS	22173	44145	22173	44145	5644	7473	5644	7473	32708	75363	8418	11921
AD	210023	416955	210023	416955	77894	102843	77894	102843	40901	13152	12238	22093
AUDIO	20743	41307	20743	41307	5094	6747	5094	6747	50130	55675	18208	29317
BBC	145283	288471	145283	288471	52994	69975	52994	69975	39389	29532	12335	14578
NETFLIX	20743	41307	20743	41307	5094	6747	5094	6747	36286	27173	10997	39868
BOOK	72743	144507	72743	144507	25094	33147	25094	33147	51493	54839	10978	13678
20-NEWSGRP	126043	250287	126043	250287	45594	60207	45594	60207		58749	15793	65881
REUTERS-52	123313	244869	123313	244869	44544	58821	44544	58821	155191	36343	10410	36440
WEBKB	116813	231969	116813	231969	42044	55521	42044	55521	223847	25406	11033	17122
DNA	31143	61947	31143	61947	9094	12027	9094	12027	12180	17507	3068	2616
JESTER	20743	41307	20743	41307	5094	6747	5094	6747	25076	27713	11322	20273
KDD	16063	32019	15565	17069	3294	4371	1908	4371	8755	6572	2915	13040
KOSAREK	32443	64527	32443	64527	9594	12687	9594	12687	19512	37583	7173	20938
MSNBC	8033	16053	8033	16053	944	1269	944	1269	11606	20795	5465	4887
MSWEB	45963	91359	45963	91359	14794	19551	14794	19551	10743	2347	6581	12135
NLTCS	7775	15539	7291	15539	894	1203	876	1203	1855	4373	1304	4401
PLANTS	16713	33309	16713	33309	3544	4701	3544	4701	36596	119194	11583	13960
PUMSB-STAR	28933	57561	28933	57561	8244	10905	8244	10905	26206	108876	8298	8866
EACHMOVIE	72743	144507	72743	144507	25094	33147	25094	33147	54184	123996	20648	21369
RETAIL	25293	50337	25293	50337	6844	9057	6844	9057	2158	3979	2989	6651

Table 4: Circuit size (in number of nodes) for each learning algorithm.

Dataset	RPMaxSD	RPMaxD	RPSIDSD	RPSIDD	RPMaxS	RPMax	RPSIDS	RPSID	LSPN	Strudel	LPSDD
NLTCS	3s	2s	5s	3s	3s	2s	6s	3s	7m	3m	6m
PLANTS	8s	4s	12s	7s	7s	4s	12s	7s	50m	41m	26m
AUDIO	9s	4s	14s	8s	8s	4s	14s	8s	2h	33m	51m
JESTER	5s	3s	9s	5s	4s	3s	9s	5s	52m	24m	37m
NETFLIX	8s	4s	15s	8s	7s	4s	14s	8s	1h	14m	33m
ACCIDENTS	7s	4s	13s	7s	7s	4s	13s	7s	47m	20m	41m
BOOK	12s	6s	18s	10s	9s	6s	17s	10s	>3h	8m	1.3h
DNA	1s	1s	3s	2s	2s	1s	2s	1s	>3h	>3h	>3h

Table 5: Run-times for each learning algorithm.

(e.g. marginal probability). In spite of such shortcomings, we hope that the already interesting results obtained encourage further research on the topic.

Acknowledgements

This work was supported by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) grants # 133787/2019-2 and 304012/2019-0, and CAPES Finance Code 001.

References

Francis R. Bach and Michael I. Jordan. Thin junction trees. In *Proceedings of the 14th International Conference on Neural Information Processing Systems*, NeurIPS, page 569–576, 2001.

Jon L. Bentley. Multidimensional binary search trees in database applicatoins. *IEEE Transactions on Software Engineering*, 5:4, 1979.

Leo Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.

Alvaro H. C. Correia, Robert Peharz, and Cassio de Campos. Joints in random forests. In *Advances in Neural Information Processing Systems 33 (NeurIPS)*, 2020.

Meihua Dang, Antonio Vergari, and Guy Van den Broeck. Strudel: Learning structured-decomposable probabilistic circuits. In *Proceedings of the 10th International Conference on Probabilistic Graphical Models*, PGM, 2020.

Sanjoy Dasgupta and Yoav Freund. Random projection trees and low dimensional manifolds. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, STOC, pages 537–546, 2008.

Mattia Desana and Christoph Schnörr. Learning arbitrary sum-product network leaves with expectation-maximization. *arXiv preprint arXiv:1604.07243*, 2016.

Aman Dhesi and Purushottam Kar. Random projection trees revisited. In *Advances in Neural Information Processing Systems*, volume 23 of *NeurIPS*, 2010.

Nicola Di Mauro, Antonio Vergari, Teresa M. A. Basile, and Floriana Esposito. Fast and accurate density estimation with extremely randomized cutset networks. In Michelangelo Ceci, Jaakko Hollmén, Ljupčo Todorovski, Celine Vens, and Sašo Džeroski, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 203–219, 2017.

Gal Elidan and Stephen Gould. Learning bounded treewidth bayesian networks. In *Advances in Neural Information Processing Systems*, volume 21 of *NeurIPS*, 2009.

Yoav Freund, Sanjoy Dasgupta, Mayank Kabra, and Nakul Verma. Learning the structure of manifolds using random projections. In *Advances in Neural Information Processing Systems*, volume 20 of *NeurIPS*, 2008.

Robert Gens and Pedro Domingos. Learning the structure of sum-product networks. In *Proceedings of the 30th International Conference on Machine Learning*, ICML, pages 873–880, 2013.

Priyank Jaini, Amur Ghose, and Pascal Poupart. Prometheus: Directly learning acyclic directed graph structures for sum-product networks. In *International Conference on Probabilistic Graphical Models*, PGM, pages 181–192, 2018.

Pasha Khosravi, Antonio Vergari, YooJung Choi, Yitao Liang, and Guy Van den Broeck. Handling missing data in decision trees: A probabilistic approach, 2020.

Doga Kisa, Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. Probabilistic sentential decision diagrams. *Knowledge Representation and Reasoning Conference*, 2014.

Yitao Liang, Jessa Bekker, and Guy Van den Broeck. Learning the structure of probabilistic sentential decision diagrams. In *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence*, 2017.

Han Liu, Min Xu, Haijie Gu, Anupam Gupta, John Lafferty, and Larry Wasserman. Forest density estimation. *Journal of Machine Learning Research*, 12:907–951, 2011. ISSN 1532-4435.

Daniel Lowd and Jesse Davis. Learning markov network structure with decision trees. In *2010 IEEE International Conference on Data Mining*, 2010.

Nicola Di Mauro, Gennaro Gala, Marco Iannotta, and Teresa M. A. Basile. Random probabilistic circuits. In *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*, 2021.

Jun Mei, Yong Jiang, and Kewei Tu. Maximum a posteriori inference in sum-product networks. In *AAAI Conference on Artificial Intelligence*, 2018.

Robert Peharz, Robert Gens, Franz Pernkopf, and Pedro Domingos. On the latent variable interpretation in sum-product networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(10):2030–2044, 2016.

Robert Peharz, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Xiaoting Shao, Kristian Kersting, and Zoubin Ghahramani. Random sum-product networks: A simple and effective approach to probabilistic

deep learning. In Amir Globerson and Ricardo Silva, editors, *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence*, 2019.

Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, pages 337–346, 2011.

Tahrima Rahman, Prasanna Kothalkar, and Vibhav Gogate. Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of chow-liu trees. In *Proceedings of the 2014th European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 630–645, 2014.

Parikshit Ram and Alexander G. Gray. Density estimation trees. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD, pages 627–635, 2011.

Padhraic Smyth, Alexander Gray, and Usama Fayyad. Retrofitting decision tree classifiers using kernel density estimation. In *Proceedings of the Twelfth International Conference on Machine Learning*, ICML, pages 506–514, 1995.

Jan Van Haaren and Jesse Davis. Markov network structure learning: A randomized feature generation approach. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

Han Zhao, Pascal Poupart, and Geoffrey J Gordon. A unified approach for learning the parameters of sum-product networks. In *Advances in Neural Information Processing Systems*, NeurIPS, 2016.