Fast and Accurate Density Estimation with Extremely Randomized Cutset Networks

Nicola Di Mauro[1(✉)], Antonio Vergari[1], Teresa M. A. Basile[2,3], and Floriana Esposito[1]

[1] Department of Computer Science, University of Bari "Aldo Moro", Bari, Italy
nicola.dimauro@uniba.it
[2] Department of Physics, University of Bari "Aldo Moro", Bari, Italy
[3] National Institute for Nuclear Physics (INFN), Bari Division, Bari, Italy

Abstract. Cutset Networks (CNets) are density estimators leveraging context-specific independencies recently introduced to provide exact inference in polynomial time. Learning a CNet is done by firstly building a weighted probabilistic OR tree and then estimating tractable distributions as its leaves. Specifically, selecting an optimal OR split node requires cubic time in the number of the data features, and even approximate heuristics still scale in quadratic time. We introduce Extremely Randomized Cutset Networks (XCNets), CNets whose OR tree is learned by performing random conditioning. This simple yet surprisingly effective approach reduces the complexity of OR node selection to constant time. While the likelihood of an XCNet is slightly worse than an optimally learned CNet, ensembles of XCNets outperform state-of-the-art density estimators on a series of standard benchmark datasets, yet employing only a fraction of the time needed to learn the competitors. Code and data related to this chapter are available at: https://github.com/nicoladimauro/cnet.

Keywords: Tractable probabilistic models · Cutset Networks

1 Introduction

Density estimation is the unsupervised task of learning an estimator for the joint probability distribution over a set of random variables (RVs) that generated the observed data. Once such an estimator is learned, it is used to do *inference*, i.e., computing the probability of the queries about certain states of the RVs. Since a perfect estimate of the real distribution would allow to solve many learning tasks exactly when reframed as different kinds of inference[1], density estimation classifies as one of the most general task in machine learning [13].

The main challenge in density estimation is balancing the *representation expressiveness* of the learned model against the *cost of learning* it and *performing*

N. Di Mauro and A. Vergari—Both authors contributed equally.

[1] E.g., classification can be framed as Most Probable Explanation (MPE) inference.

M. Ceci et al. (Eds.): ECML PKDD 2017, Part I, LNAI 10534, pp. 203–219, 2017.
https://doi.org/10.1007/978-3-319-71249-9_13

inference on it. Probabilistic Graphical Models (PGMs), like Bayesian Networks (BNs) and Markov Networks (MNs), are able to model highly complex probability distributions. However, exact inference with them is generally *intractable*, i.e., not solvable in polynomial time, and even some approximate inference routines are intractable in practice [23]. With the aim of performing exact and polynomial inference, a series of *tractable probabilistic models* (TPMs) have been recently proposed: either by restricting the expressiveness of PGMs by bounding their treewidth [24], e.g., tree distributions and their mixtures [18], or by exploiting local structures in a distribution [4]. It is worth noting that inference tractability is not a global property, but it is associated to classes of queries. For instance, computing exact marginals on a TPM may be feasible, while MPE may be not [1]. TPMs like Arithmetic Circuits [6], Sum-Product Networks (SPNs) [19], and Cutset Networks (CNets) [21] promise a good compromise between expressive power and tractable inference by compiling high treewidth distributions in compact and efficient data structures. Even if learning such TPMs may be done in polynomial time, thanks to several recent algorithmic schemes, making these algorithms scale to high dimensional data is still an issue. We focus on CNets since they (i) exactly and tractably compute several inference query types like marginals, conditionals and MPE inference [7], and (ii) promise faster learning times, when compared to other TPMs.

CNets have been introduced in [21] as weighted probabilistic model trees having tree-structured models as the leaves of an OR tree. They exploit context-specific independencies (CSIs) [2] by embedding Pearl's conditioning algorithm. While the learning algorithm originally proposed in [21] provides a heuristic approach, it still requires quadratic time w.r.t. the number RVs to select each tree inner node to condition on. A theoretically principled and more accurate version, presented in [9], overcomes many of the initial version issues, like the tendency to overfit. However, in order to do so, it increases the complexity of performing a single split to cubic time. We tackle the problem of scaling CNet learning to high dimensional data while preserving inference accuracy.

Here we introduce Extremely Randomized CNets (XCNets), as CNets that can be learned in a simple, fast and yet effective approach by performing random conditioning to grow the OR tree. In such a way, selecting a node to split on reduces to constant time w.r.t. the number of features. As we will see, while the likelihood of a single XCNet is not greater than an optimally learned CNet, ensembles of XCNets outperform state-of- the-art density estimators on a series of standard benchmark datasets, yet employing a fraction of the time needed to learn the competitors. To further reduce the learning complexity, we investigate the exploitation of a naive factorization as leaf distribution in XCNets. As a result, we can build an extremely fast mixture of density estimators that is more accurate than several CNets and comparable to a BN exploiting CSI [3].

2 Background

Notation. Let RVs be denoted by upper-case letters, e.g., X, and their values as the corresponding lower-case letters, e.g., $x \sim X$. We denote sets of RVs as

$\mathbf{X}$, and their combined values as $\mathbf{x}$. For a set of RVs $\mathbf{X}$ we denote with $\mathbf{X}_{\setminus i}$ the set $\mathbf{X}$ deprived of X_i, and with $\mathbf{X}_{|\mathbf{Y}}$ the restriction of $\mathbf{X}$ to $\mathbf{Y} \subseteq \mathbf{X}$ (the same applies to assignments $\mathbf{x}$). W.l.o.g., we assume RVs we deal with in the following to be binary valued.

Density Estimation. Let $\mathcal{D} = \{\xi^j\}_{j=1}^m$ be a set of m n-dimensional samples drawn i.i.d. according to an unknown joint probability distribution $\mathsf{p}(\mathbf{X})$, with $\mathbf{X} = \{X_i\}_{i=1}^n$. We refer to $\xi^j[X_i]$ as the value assumed by the sample ξ^j in correspondence of the RV X_i. We are interested in learning a model $\mathcal{M}$ from $\mathcal{D}$ such that its estimate of the underlying distribution, denoted as $\mathsf{p}_{\mathcal{M}}(\mathbf{X})$, is as close as possible to the original one [13]. Generally, measuring this closeness is done via the *log-likelihood* function, or one of its variants, defined as: $\ell_{\mathcal{D}}(\mathcal{M}) = \sum_{j=1}^m \log \mathsf{p}_{\mathcal{M}}(\xi^j)$. In the next sub-sections we review the approaches to density estimation as the building blocks of XCNets we propose in Sect. 4.

2.1 Product of Bernoulli Distributions

The simplest representation assumption for $\mathsf{p}(\mathbf{X})$ over RVs $\mathbf{X}$, allowing tractable inference, involves considering all RVs in $\mathbf{X}$ to be independent: $\mathsf{p}(\mathbf{x}) = \prod_{i=1}^n \mathsf{p}(x_i)$. For binary RVs, this *naive* factorization leads to the product of Bernoulli distributions (PoBs) model, where building $\mathsf{p}_{\mathcal{M}}$ equals to estimate the $\mathsf{p}_{\mathcal{M}}(x_i^0) = \theta_i^0$.

Proposition 1 (LearnPoB time complexity). *Learning a PoB from $\mathcal{D}$ over RVs $\mathbf{X}$ has time complexity $O(nm)$, where $m = |\mathcal{D}|$ and $n = |\mathbf{X}|$.*

Proof. For each Bernoulli RV $X_i \in \mathbf{X}$, estimating θ_i requires a single pass over $\{\xi^j[X_i]\}_{j=1}^m$, hence taking $O(m)$. Consequently, for all RVs in $\mathbf{X}$, it takes $O(mn)$.

Similarly to what Naive Bayes provides for classification, PoBs deliver a cheap and very fast baseline for tractable density estimation, even if the total independence assumption clearly does not hold on real data. Moreover, mixtures of PoBs, sometimes simply referred to mixtures of Bernoulli distributions (MoBs), have proved as an effective way to increase the representation expressiveness of PoBs [16]. However, while inference on MoBs is still tractable, learning them in a principled way requires running the EM algorithm for k iterations and r restarts, thus increasing the complexity up to $O(rkmn)$ [16].

2.2 Probabilistic Tree Models

A *directed tree-structured model* [18] over $\mathbf{X}$ is a BN in which each node $X_i \in \mathbf{X}$ has at most one parent, Pa_{X_i}. It encodes a distribution that factorizes as: $\mathsf{p}(\mathbf{x}) = \prod_{i=1}^n \mathsf{p}(x_i|\mathrm{Pa}_{x_i})$, where Pa_{x_i} denotes the projection of the assignment $\mathbf{x}$ on the parent of X_i. By modeling such dependencies, tree-structured models can be more expressive than PoBs, yet still performing exact complete and marginal inference in $O(n)$ [18]. To learn a model $\mathcal{M} = \langle \mathcal{T}, \{\theta_{i|\mathrm{Pa}_{X_i}}\}_{i=1}^n \rangle$, now one has

to estimate *both* a tree structure $\mathcal{T}$ and the conditional probabilities $\theta_{i|\mathrm{Pa}_{X_i}} = \mathsf{p}_{\mathcal{M}}(X_i|\mathrm{Pa}_{X_i})$. Growing an optimal model, according to the KL-divergence, can be done by employing the classical result from Chow and Liu [5]. We will refer to tree-structured models as Chow-Liu trees, or CLtrees, assuming the Chow-Liu algorithm (LearnCLTree) has been employed to learn them.

Proposition 2 (LearnCLTree time complexity [5]). *Learning a CLtree from $\mathcal{D}$ over RVs $\mathbf{X}$ has time complexity $O(n^2(m+\log n))$, where $m = |\mathcal{D}|$ and $n = |\mathbf{X}|$.*

Proof. For each pair of RVs in $\mathbf{X}$, their mutual information (MI) can be estimated from $\mathcal{D}$ in $O(mn^2)$ steps. Building a maximum spanning tree on the weighted graph induced by the adjacency matrix MI takes $O(n^2 \log n)$. Lastly, both arbitrarily rooting the tree, traversing it, and estimating the conditional probabilities $\theta_{i|\mathrm{Pa}_{X_i}}$ can be done in $O(n)$.

All in all, the complexity of learning a CLTree is quadratic in n. While this is a huge gain w.r.t. learning a higher order dependency BN, it still poses a practical issue when LearnCLTree is applied as a routine in larger learning schemes and on datasets with thousand features. Nevertheless, CLTrees have been employed as the core components of many tractable probabilistic models ranging from mixtures of them [18], SPNs [26] and CNets [8,9,21]. We will specifically tackle the problem of scaling CNet learning in the following sections.

3 Cutset Networks

Cutset Networks are TPMs introduced in [21] as a hybrid of OR trees and CLTrees as the tree leaves. Here we generalize their definition to comprise generic TPMs as leaf distributions. A CNet $\mathcal{C}$ over a set of RVs $\mathbf{X}$, is a probabilistic weighted model tree defined via a rooted OR tree $\mathcal{G}$ and a set of TPMs $\{\mathcal{M}_i\}_{i=1}^{L}$, in which each $\mathcal{M}_i$ encodes a distribution $\mathsf{p}_{\mathcal{M}_i}$ over a subset of $\mathbf{X}$, called *scope* and denoted as $\mathsf{sc}(\mathcal{M}_i)$. The scope of a CNet $\mathcal{C}$, $\mathsf{sc}(\mathcal{C})$, is the set of RVs appearing in it. A CNet may be defined recursively as follows.

Definition 1 (Cutset network). *Given binary RVs $\mathbf{X}$, a CNet is: (1) a TPM $\mathcal{M}$, with $\mathsf{sc}(\mathcal{M}) = \mathbf{X}$; or (2) a weighted disjunction of two CNets $\mathcal{C}_0$ and $\mathcal{C}_1$ graphically represented as an OR node conditioned on RV $X_i \in \mathbf{X}$, with associated weights w_i^0 and w_i^1 s.t. $w_i^0 + w_i^1 = 1$, where $\mathsf{sc}(\mathcal{C}_0) = \mathsf{sc}(\mathcal{C}_1) = \mathbf{X}_{\setminus i}$.*

A CNet over binary RVs is shown in Fig. 1: each circled node is an OR tree node and labeled by a variable X_i. Each edge emanating from it is weighted by the probability w_i^0, resp.w_i^1, of conditioning X_i to the value 0, resp. 1. The distribution encoded by a CNet $\mathcal{C}$ can be written as:

$$\mathsf{p}(\mathbf{x}) = \mathsf{p}_l(\mathbf{x}_{|\mathsf{sc}(\mathcal{C})\setminus\mathsf{sc}(\mathcal{M}_l)})\mathsf{p}_{\mathcal{M}_l}(\mathbf{x}_{|\mathsf{sc}(\mathcal{M}_l)}), \tag{1}$$

where $\mathsf{p}_l(\mathbf{x}_{|\mathsf{sc}(\mathcal{C})\setminus\mathsf{sc}(\mathcal{M}_l)}) = \prod_i (w_i^0)^{1-x_i}(w_i^1)^{x_i}$ is a factor obtained by multiplying all the weights attached to the edges of the path in the OR tree starting from the root of $\mathcal{C}$ and reaching a unique leaf node l; on the other hand, $\mathsf{p}_{\mathcal{M}_l}(\mathbf{x}_{|\mathsf{sc}(\mathcal{M}_l)})$ is the distribution encoded by the reached leaf l. $\mathsf{p}_{\mathcal{M}_l}$ can be interpreted as a conditional distribution $\mathsf{p}(\mathbf{x}_{|\mathsf{sc}(\mathcal{M}_l)}|\mathbf{x}_{|\mathsf{sc}(\mathcal{C})\setminus\mathsf{sc}(\mathcal{M}_l)})$.

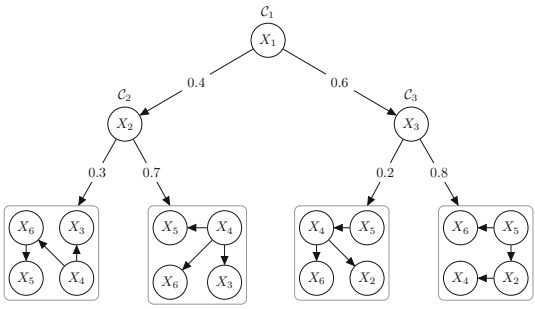


Fig. 1. Example of a CNet over binary RVs. Inner (rounded) nodes on variables X_i are OR nodes, while leaf (squared) nodes represent CLtrees.

3.1 Learning CNets

Learning both the structure and parameters of a CNet from data equals to perform searching in the space of all probabilistic weighted model trees. This would require an exponential time: for a dataset $\mathcal{D}$ over RVs $\mathbf{X}$ learning a full binary OR tree with height k has time complexity $O(n^k 2^k (n^2(m + \log n))) = O(m2^k n^{k+2})$, with $m = |\mathcal{D}|$ and $n = |\mathbf{X}|$. In practice, this problem is tackled in a *two-stage greedy fashion* by: (i) first performing a top-down search in the space of weighted OR trees, and then (ii) learning TPMs as leaf distributions according to a conditioned subset of the data. The first structure learning algorithm for CNets is the one introduced in [21], leveraging a heuristic approach to induce the OR tree and demanding pruning to combat overfitting. A following approach has been introduced in [9], growing the OR tree by a principled Bayesian search maximizing the data likelihood. In the following, we introduce a general scheme to learn CNets, showing how, by properly determining a splitting criterion to grow the OR tree, one can recover both the algorithms from [21] and [9]. This, in turn, highlights how the splitting criterion time complexity determines that of learning the whole OR tree, and hence the whole CNet. In Sect. 4, we propose a variation of the splitting procedure drastically reducing its cost.

General Learning Scheme. Algorithm 1 reports a general approach for CNets structure learning. In particular, the procedure tries to select a variable X_i on the input data slice $\mathcal{D}$ (line 4). If a such a variable exists (line 5), it then recursively (line 8) tries to decompose the two new slices $\mathcal{D}_0$ and $\mathcal{D}_1$ over $\mathbf{X}_{\setminus i}$. When the slice $\mathcal{D}$ has few instances, or it is defined on few variables, then a leaf distribution is learned (line 10). Both, the algorithms reported in [9,21] use CLtrees as leaf distribution, i.e., the learnDistribution procedure on line 10 corresponds to call the LearnCLTree algorithm.

By deriving the time complexity of both growing the OR tree and learning the leaf distributions, one can derive the whole time complexity of LearnCNet. In turn, the time complexity of growing the OR tree clearly depends by the cost of selecting the RV to split on at each step. If we assume the variations of LearnCNet

Algorithm 1. LearnCNet($\mathcal{D}$, $\mathbf{X}$, α, δ, σ)

1: **Input:** a dataset $\mathcal{D}$ over RVs $\mathbf{X}$; α: Laplace smoothing factor; δ min number of samples to split; σ min number of features to split
2: **Output:** a CNet $\mathcal{C}$ encoding $\mathsf{p}_{\mathcal{C}}(\mathbf{X})$ learned from $\mathcal{D}$
3: **if** $|\mathcal{D}| > \delta$ **and** $|\mathbf{X}| > \sigma$ **then**
4: $\quad X_i, \mathsf{success} \leftarrow \mathsf{select}(\mathcal{D}, \mathbf{X}, \alpha)$
5: $\quad$ **if** success **then**
6: $\quad\quad \mathcal{D}_0 \leftarrow \{\xi \in \mathcal{D} : \xi[X_i] = 0\}$, $\mathcal{D}_1 \leftarrow \{\xi \in \mathcal{D} : \xi[X_i] = 1\}$
7: $\quad\quad w_0 \leftarrow |\mathcal{D}_0|/|\mathcal{D}|$, $w_1 \leftarrow |\mathcal{D}_1|/|\mathcal{D}|$
8: $\quad\quad \mathcal{C} \leftarrow w_0 \cdot \mathsf{LearnCNet}(\mathcal{D}_0, \mathbf{X}_{\setminus i}, \alpha, \delta, \sigma) + w_1 \cdot \mathsf{LearnCNet}(\mathcal{D}_1, \mathbf{X}_{\setminus i}, \alpha, \delta, \sigma)$
9: **else**
10: $\quad \mathcal{C} \leftarrow \mathsf{learnDistribution}(\mathcal{D}, \mathbf{X}, \alpha)$
11: **return** $\mathcal{C}$

have grown the same sized OR trees, the time complexity of each implementation of select determines the whole OR tree growing phase complexity. Concerning learning leaf distributions, its complexity is determined by the cost of learning a single distribution, that in case of CLTrees is $O(n^2(m + \log(n)))$ (see Proposition 2). As a consequence, assuming to learn L leaves for a tree, then it would take $O(Ln^2(m + \log(n)))$ for all variations to learn such leaves. In the following Sections we revise and analyze the two variations of LearnCNet reported in [9,21].

Proposition 3. *Growing a full binary OR tree with LearnCNet on $\mathcal{D}$ over RVs $\mathbf{X}$ has time complexity $O(k(S + m))$, where $m = |\mathcal{D}|$, $n = |\mathbf{X}|$, k is the height of the OR tree, and $S = T(m, n)$, assumed to grow linearly w.r.t. m holding n constant, is the time required to compute the OR split node selection procedure on $\mathcal{D}$ (select function in Algorithm 1, line 4).*

Proof. A set $\mathcal{D}_t^h \subset \mathcal{D}$ of samples falls in each internal node t with height h, such that $\forall i \neq j : \mathcal{D}_i^h \cap \mathcal{D}_j^h = \emptyset$, and $\cup_{i=1}^{2^h} \mathcal{D}_i^h = \mathcal{D}$. Furthermore, for each internal node t with height h, $T(|\mathcal{D}_t^h|, n-h)$ has been the time required to compute the OR split selection, and $|\mathcal{D}_t^h|$ is the time required to split the samples $\mathcal{D}_t^h$. Assuming that $T(m, n)$ grows linearly w.r.t. m holding n constant, then for each height h we have a time complexity equal to $O(\sum_{i=1}^{2^h}(T(|\mathcal{D}_i^h|, n-h) + |\mathcal{D}_i^h|)) = O(T(|\mathcal{D}|, n-h) + m)$. Since the OR tree has height k, then the overall time is $O(\sum_{i=0}^{k-1}(T(|\mathcal{D}|, n-i) + m)) = O(k(T(|\mathcal{D}|, n) + m))$.

Information Gain Splitting Heuristic. The algorithm to learn CNet structures proposed in [21], that here we will call entCNet, performs a greedy top-down search in the OR-trees space that can be reframed in Algorithm 1. It implements the select function as a procedure to determine the RV X_i that maximizes a generative reformulation of the information gain from decision tree theory. Since computing the joint entropy over RVs $\mathbf{X}_{\setminus i}$ would be unfeasible to calculate, it heuristically approximates it by computing the average over marginal entropies.

To cope with the systematic overfitting showed by CNets learned by entCNet, always in [21], a post-pruning method on a validation set is introduced. Leveraging this decision tree technique, on a fully grown CNet, by advancing bottom-up, leaves are pruned and inner nodes without children replaced with a CLtree (that needs to be learned from data), if the network validation data likelihood after this operation is higher than that scored by the not pruned network.

Proposition 4 (select time complexity in entCNet [21]). *The time complexity for selecting the best splitting node on a slice $\mathcal{D}$ over RVs $\mathbf{X}$ in entCNet is $O(mn^2)$, where $m = |\mathcal{D}|$ and $n = |\mathbf{X}|$.*

Corollary 1. *Growing a full binary OR tree for entCNet when learning a CNet on $\mathcal{D}$ over RVs $\mathbf{X}$ has time complexity $O(kmn^2)$, where $m = |\mathcal{D}|$, $n = |\mathbf{X}|$, and k is the height of the OR tree.*

Proof. From Propositions 3 and 4, the overall time complexity to grow a full binary OR tree is $O(k(mn^2 + m)) = O(km(n^2 + 1))$.

dCSN: likelihood guided splitting. In [9], the authors proposed the dCSN algorithm that exploits a different approach from that in [21], by avoiding decision tree heuristics while choosing the best variable directly maximizing the data log-likelihood. As already reported in [9], the log-likelihood function of a CNet may be decomposed as follows. Given a CNet $\mathcal{C}$ learned on $\mathcal{D}$ over $\mathbf{X}$, its log-likelihood $\ell_{\mathcal{D}}(\mathcal{C})$ can be computed as follows: $\ell_{\mathcal{D}}(\mathcal{C}) = \sum_{\xi \in \mathcal{D}} \sum_{i=1,\ldots,n} \log \mathsf{p}(\xi[X_i]|\xi[\mathrm{Pa}_{X_i}])$, when $\mathcal{C}$ corresponds to a CLtree. While, in the case of a OR tree rooted on the variable X_i, the log-likelihood is:

$$\ell_{\mathcal{D}}(\mathcal{C}) = \sum_{j=0,1} m_j \log w_i^j + \ell_{\mathcal{D}_j}(\mathcal{C}_j), \tag{2}$$

being $\mathcal{C}_j$ the CNet involved in the OR, $\mathcal{D}_j = \{\xi \in \mathcal{D} : \xi[X_i] = j\}$, $m_j = |\mathcal{D}_j|$, and $\ell_{\mathcal{D}_j}(\mathcal{C}_j)$ is the log-likelihood of the sub-CNet $\mathcal{C}_j$ on the slice $\mathcal{D}_j$, for $j = 0, 1$.

By exploiting this recursive nature of CNets, a CNet is grown top-down, allowing further expansion, i.e., the substitution of a CLtree with an OR node, only if it improves the structure log-likelihood, since it is clear to see that maximizing the second term in Eq. 2, results in maximizing the global score.

As reported in [9], one starts with a single CLtree, learned from $\mathcal{D}$ over $\mathbf{X}$, and then it checks whether there is a decomposition, i.e., an OR node on the best variable X_i applied on two CLtrees, providing a better log-likelihood than that scored by the initial tree. If such a decomposition exists, than the decomposition process is recursively applied to the sub-slices $\mathcal{D}_0$ and $\mathcal{D}_1$ over $\mathbf{X}_{\setminus i}$, testing each leaf for a possible substitution.

Proposition 5 (select time complexity in dCSN). *The time complexity for selecting the best splitting node on a slice $\mathcal{D}$ over RVs $\mathbf{X}$ in dcsn is $O(n^3(m + \log n))$, where $m = |\mathcal{D}|$ and $n = |\mathbf{X}|$.*

Proof. For each variable $X_i \in \mathbf{X}$, two CLTrees have been computed on $\mathcal{D}_0$ and $\mathcal{D}_1$ leading to a splitting complexity $O(n^2(m + \log n))$. Since n splits have to be checked, the overall complexity to select the best split is $O(n^3(m + \log n))$.

Corollary 2. *Growing a full binary OR tree on $\mathcal{D}$ over RVs $\mathbf{X}$ with dCSN has time complexity $O(kmn^3)$, where $m = |\mathcal{D}|$, $n = |\mathbf{X}|$, and k is the height of the OR tree.*

Proof. From Propositions 3 and 5, the overall time complexity to grow a full binary OR tree is $O(k(mn^3 + m)) = O(km(n^3 + 1))$.

3.2 Learning Ensembles of CNets

To mitigate issues like the scarce accuracy of a single model and their tendency to overfit, since [21] CNets have been employed as the components of a mixture of the form: $\mathsf{p}(\mathbf{X}) = \sum_{i=1}^{c} \lambda_i \mathcal{C}_i(\mathbf{X})$, being $\lambda_i \geq 0 : \sum_{i=1}^{c} \lambda_i = 1$ the mixture coefficients. The first approach to learn such a mixture employs EM to alternatively learn both the weights and the mixture components. With this approach, the time complexity of learning CNets grows at least of a factor of ct, where t is the number of iterations of EM. All the classic issues about convergence and instability of EM make this approach less practical then the following ones. A more efficient method to learn Mixtures of CNets, presented in [9], adopts bagging as a cheap and yet more effective way to only increase time complexity by a factor c. For bagged CNets, mixture coefficients are set equally probable and the mixture components can be learned independently on different bootstrapped data samples. An approach adding random subspace projection to bagged CNets learned with dCSN has been introduced in [8]. While its worst case complexity is the same as for bagging, the cost of growing the OR tree reduced by random sub-spacing is effective in practice. Mixtures of CNets have been learned by exploiting three boosting approaches proposed in [20], having time complexity equals to that for bagging or even worst.

4 Extremely Randomized CNets

XCNets (Extremely Randomized CNets) are CNets that are built by LearnCNet where the OR split node procedure (the select function in Algorithm 1, line 4) is simplified in the most straightforward way: selecting a RV uniformly at random. We denote this algorithmic variant of LearnCNet as XCNet. As a consequence, the cost of the new select function in XCNet does not directly depend anymore on the number of features n and can be considered to be constant.

Proposition 6 (select time complexity in XCNet). *The time complexity for selecting the splitting node on a slice $\mathcal{D}$ over $\mathbf{X}$ in XCNet is $O(1)$.*

Proof. The time required to randomly choose a number in $(1, \ldots, |\mathbf{X}|)$.

Corollary 3. *Growing a full binary OR tree on $\mathcal{D}$ over $\mathbf{X}$ with XCNet has time complexity $O(km)$, where k is the height of the OR tree.*

Proof. From Propositions 3 and 6, the overall time complexity to grow a full binary OR tree is $O(k(1+m))$.

While we introduce this variation with the obvious aim of speeding up a CNet OR tree learning process, we argue that XCNet should still provide accurate density estimators. We support this conjecture with the following motivations.

A CNet can be seen as a sort-of *mixture of experts* in which the gating function role is demanded to the OR tree, the leaf distributions act as the local experts, and the gating function operates by selecting only one expert per input sample. Let $g : \mathbf{X} \rightarrow \{\mathcal{M}_i\}_{i=1}^{L}$ be a gating function that associates each configuration $\xi \sim \mathbf{X}$ to only one leaf model, $\mathcal{M}_\xi$. For a CNet $\mathcal{C}$, g can be built by associating to each ξ a path p in the OR tree structure $\mathcal{G}$ of $\mathcal{C}$. A path $p = p_{(1)}p_{(2)}\cdots p_{(k)}$ of length k is grown as a sequence of observed values $v_1 v_2 \cdots v_k$ in the same fashion as one performs inference according to Eq. 1: starting from the root of $\mathcal{C}$, for each OR node i traversed, corresponding to RV $X_{p(i)}$, the branching corresponding to the value $v_i = \xi[X_{p(i)}]$ is followed. At the end of the path p, a leaf model $\mathcal{M}_p = \mathcal{M}_\xi$ is reached. Alternatively, one can express g as a function of all possible combinations one can build over a set of observed RVs $\mathbf{X}$: $g(\xi) = \sum_{p\in\mathcal{G}} \prod_{i=1}^{|p|} \mathbb{1}\{\xi[X_{p(i)}] = v_i\}\mathcal{M}_p$. Now, from this construction of g, one can derive that permuting the order of appearance of the RVs values v_i does not change the value of g. In the same way, from the factorization in Eq. 1, it follows that neither the joint probability mass associated to the configuration ξ changes after such a permutation. This follows from the fact that the portion of the likelihood assigned to ξ that depends on the path p can be exactly recovered by choosing another sequence of conditionings, as different applications of the chain rule of probability still model the same joint distribution. This permutation invariance suggests that given a way to associate a sample to a leaf distribution, the way in which conditionings are performed can be irrelevant. Clearly, while this is true for an already learned CNet, for algorithms inducing the OR tree in a top-down fashion, the order in which conditionings are performed during learning obviously matter. Nevertheless, in practice, it might matter less than expected. From another perspective, building an OR tree, and hence g, is likely to perform a clustering of all possible sample configurations. For all LearnCNet variants, this clustering performs a trivial aggregation of samples based on their equal observed values for the conditioned RVs. This is one of the issues why algorithms like entCNet are very prone to overfit. For XCNets, however, the randomization introduced in this clustering phase behaves as a regularizer and helps to overcome the aforementioned issue. All in all, we argue that it is more demanding to estimate good distributions at the leaves than an overoptimized gating function.

Moreover, an additional motivation to the introduction of XCNets comes from ensemble theory. From the interpretation of CNets as mixture of experts, the leaf distribution of a CNet acts as an ensemble of density estimators.

Employing a randomized selection criterion increases the diversification of the leaf distributions, and, on the other hand, a strong diversification helps ensembles to better generalize [12]. To better understand this aspect consider a run of entCNet in which the select function has chosen a RV X_i instead of RV X_j to condition on as the first reduces the model entropy more than the second. In both branches x_i^0 and x_i^1 of such a conditioning, it is likely that RV X_j would still be considered as one of the top ranked RVs to be split on in the following iterations. By repeating this argument, it might be likely that the leaf distributions appearing in the sub trees generated by conditioning on x_i^0 and x_i^1 would have very similar scopes.

When constructing ensembles of CNets we expect this diversification effect introduced by randomization to be even more prominent and effective. In ensemble methods like bagging one employs bootstrapping as a source of randomness to diversify the ensemble components [12]. This is also the case for mixtures of CNets built by bagging (see Sect. 3.2). Differently from bagged CNets, ensemble of XCNets do not need an additional way to produce strongly different components. Therefore, when learning mixtures of XCNets, we aggregate the components by learning each component independently on the full dataset.

Lastly, we review Extremely Randomized Tree, or simply ExtraTrees [10] as they are similar in spirit and by name to XCNets. An ExtraTree is a decision tree that is learned by considering only a random subset of features for the introduction of an OR node (like for random forests [12]) and by randomly selecting a threshold for the actual split. Among those randomly generated hyperplanes, the best according to an optimization criterion is chosen. XCNets differ from ExtraTrees from several perspectives. First, they are density estimators and therefore each OR node in them has to split over all the possible values the chosen RV is defined on, otherwise the modeled distribution would not be a valid probability density. Consequently, an OR node in an XCNet is totally selected at random, while for ExtraTrees the best of the random selection is actually employed. Lastly, an XCNet only slightly underperforms a corresponding non-random model, while a single ExtraTree is generally a weak learner whose "raison d'etre" is to be a component in an ensemble [10].

It is tempting to further reduce the complexity of XCNet by substituting CLTrees with even simpler models. As stated in Proposition 1, learning PoBs reduces the complexity to be linear w.r.t. n. Clearly, we do not expect a CNet with PoBs as leaves to achieve a better likelihood than one with CLtrees. Nevertheless, we intend to measure how the likelihood degrades with less expressive leaf distributions and, at the same time, how faster this variant can be.

5 Experiments

The research questions we are validating are: (**Q1**) how much does extreme randomization affect the performance of an XCNet when compared to the optimal one learned with dCSN on real data? (**Q2**) how accurate are ensembles of XCNets and how do they compare against all other CNet ensembling techniques

Table 1. Datasets used and their statistics.

Dataset	n	m_{train}	m_{val}	m_{test}	Dataset	n	m_{train}	m_{val}	m_{test}
NLTCS	16	16181	2157	3236	DNA	180	1600	400	1186
MSNBC	17	291326	38843	58265	Kosarek	190	33375	4450	6675
KDDCup2k	65	180092	19907	34955	MSWeb	294	29441	3270	5000
Plants	69	17412	2321	3482	Book	500	8700	1159	1739
Audio	100	15000	2000	3000	EachMovie	500	4525	1002	591
Jester	100	9000	1000	4116	WebKB	839	2803	558	838
Netflix	100	15000	2000	3000	Reuters-52	889	6532	1028	1540
Accidents	111	12758	1700	2551	20NewsG	910	11293	3764	3764
Retail	135	22041	2938	4408	BBC	1058	1670	225	330
Pumsb-star	163	12262	1635	2452	Ad	1556	2461	327	491

and state-of-the-art density estimators? (**Q3**) how scalable are and how much time do actually XCNets save in practice?

We answer all the above questions by performing our experiments[2] on 20 de-facto standard benchmark datasets for density estimation. Introduced by [15] and [11], they are binarized versions of real data from different tasks like frequent itemset mining, recommendation and classification. We adopt their classic splits for training, validation (hyperparameter selection) and testing. Detailed names and statistics are reported in Table 1. Additionally, for the qualitative experiments in Sect. 5.1 we employ the first 10000 training 28×28 pixel images of digits of MNIST, binarized as in [14].

5.1 (Q1) Single Model Performances

Likelihood Performances. Table 2 reports the results, as the average test log-likelihoods, for all the benchmarks for a entropy-based CNet (entCNet) as reported in [9], a CNet learned with dCSN, and a XCNet (XCNet). Furthermore, we learned a CNet ($\mathsf{dCSN}_{\mathsf{PoB}}$) and a XCnet ($\mathsf{XCNet}_{\mathsf{PoB}}$) with PoBs as leaf distributions[3]. For the two XCnets variants for each dataset the reported results are the average and the standard deviation over 10 different runs. Clearly, the best scores are achieved by dCSN with entCNet following it soon after. Nevertheless, all the log-likelihoods achieved by XCNet are only slightly worse and always on the same order of magnitude if compared to non random models, while PoB variants perform considerably worse. We plot the training and test log-likelihoods achieved by dCSN and XCNet models, both ran with $\delta = 100$, while adding

[2] Source code of dCSN and XCNet in C++11 and the scripts to replicate the experiments are made available at https://github.com/nicoladimauro/cnet. All experiments have been run on a 4-core Intel Xeon E312xx (Sandy Bridge) @2.0 GHz with 8 Gb of RAM and Ubuntu 14.04.1, kernel 3.13.0-39.

[3] The following grid search to learn CNets with dCSN, XCNet, $\mathsf{dCSN}_{\mathsf{PoB}}$, and $\mathsf{XCNet}_{\mathsf{PoB}}$ has been performed: $\delta \in \{300, 500, 1000, 2000\}$, $\alpha \in \{0.1, 0.2, 0.5, 1, 2\}$ and $\sigma = 4$.

Table 2. Average test log likelihoods for all (for XCNet models mean and standard deviation over 10 runs are reported).

Dataset	entCNet	dCSN	XCNet	dCSN_{PoB}	$\text{XCNet}_{\text{PoB}}$
NLTCS	−6.06	**−6.03**	6.06 ± 0.01	−6.09	−6.17 ± 0.05
MSNBC	**−6.05**	**−6.05**	−6.09 ± 0.02	−6.05	−6.18 ± 0.03
KDDCup2k	-	**−2.18**	−2.19 ± 0.01	−2.19	−2.21 ± 0.01
Plants	**−13.25**	**−13.25**	−13.43 ± 0.07	−14.89	−15.66 ± 0.22
Audio	**−42.05**	−42.10	−42.66 ± 0.14	−42.95	−44.02 ± 0.22
Jester	−55.56	**−55.40**	−56.10 ± 0.19	−56.23	−57.39 ± 0.15
Netflix	**−58.71**	**−58.71**	−59.21 ± 0.06	−60.20	−61.40 ± 0.25
Accidents	−30.69	**−29.84**	−31.58 ± 0.24	−36.24	−40.22 ± 0.46
Retail	**−10.94**	−11.24	−11.44 ± 0.09	−11.06	−11.19 ± 0.04
Pumsb-star	−24.42	**−23.91**	−25.55 ± 0.34	−32.11	−39.91 ± 2.48
DNA	−87.59	**−87.31**	−87.67 ± 0.00	−98.83	−99.84 ± 0.05
Kosarek	**−11.04**	−11.20	−11.70 ± 0.13	−11.38	−11.80 ± 0.07
MSWeb	**−10.07**	−10.10	−10.47 ± 0.10	−10.19	−10.43 ± 0.07
Book	**−37.35**	−38.93	−42.36 ± 0.28	−38.21	−39.47 ± 0.33
EachMovie	−58.37	**−58.06**	−60.71 ± 0.89	−59.70	−62.58 ± 0.38
WebKB	−162.17	**−161.92**	−167.45 ± 1.59	−168.7	−174.78 ± 0.81
Reuters-52	**−88.55**	−88.65	−99.52 ± 1.93	−90.51	−100.25 ± 0.57
20NewsG	-	**−161.72**	−172.6 ± 1.40	−162.25	−167.39 ± 0.74
BBC	−263.08	**−261.79**	**−261.79** ± 0.00	−264.56	−274.83 ± 1.15
Ad	−16.92	**−16.34**	−18.70 ± 1.44	−36.44	−36.94 ± 1.41

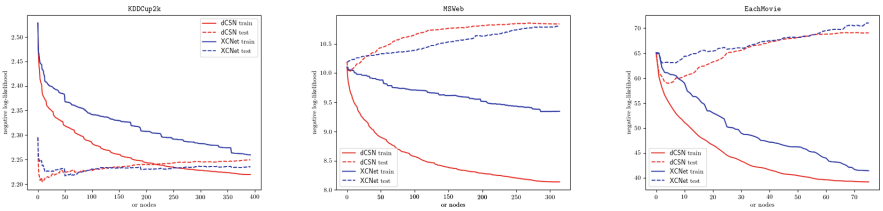
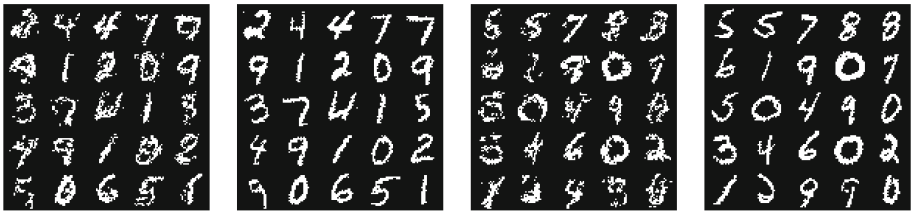


Fig. 2. Negative log-likelihood during learning CNets and XCNets.

nodes during learning in Fig. 2. It is possible to note how, on those datasets, dCSN grows CNets that start overfitting much earlier, while the aleatory nature of XCNet slows the process down and mitigates the effect.

The worst performance is obtained on Ad, with XCNet scoring a relative decrease of 14.46% of the log-likelihood w.r.t. dCSN, while PoB degrade it up to %126.07[4]. These results are very encouraging but not highly surprisingly given

[4] The relative decrease is computed as $\frac{\ell_{\mathcal{D}}(\text{XCNet}) - \ell_{\mathcal{D}}(\text{dCSN})}{\ell_{\mathcal{D}}(\text{dCSN})} \cdot 100$.



(a) (b) (c) (d)

Fig. 3. Samples obtained from a CNet (a), resp. XCNet (c), learned on samples of the binarized `MNIST` dataset, and their nearest neighbor in training set (b), resp. (d).

our interpretation of CNets as mixture of experts. Moreover this stresses the difference between XCNets with CLTrees and ExtraTrees [10], since a single extremely randomized tree performs much worse than a non-random tree, a behavior we can associate to XCNets with PoBs as leaf distributions.

Generating Samples. It is worth investigating how good is an XCNet at generating samples w.r.t. a CNet learned by dCSN. While results from the previous section can give us a fairly confident estimate according to sample log-likelihoods, these values may not align to the human evaluation of a sample quality [25]. For this reason we perform a qualitative evaluation on samples drawn from XCNets and CNets learned on the first 10000 samples of a binarized version of MNIST with fixed parameters $\delta = 50$, $\alpha = 0.01$, and $\sigma = 4$.

We randomly sampled 25 digits from both models comparing them to the nearest neighbor in the training set, ensuring that the generated samples are not simple memorization, as reported in Fig. 3. It is evident how both models have not memorized the training samples. Since it is not possible to visually spot very relevant differences between the two sample sets, we can confirm that close log-likelihoods correspond to qualitatively similar samples for XCNets and CNets.

5.2 (Q2) Ensemble Performances

To investigate the performance of ensembles of XCNets we build ensembles of 40 components to be comparable with the approaches reported in Sect. 3.2 and introduced in [9,20,21]. We report in the first half of Table 3 the best results for ensembles of bagged (CNet^{40}) and boosted ($\mathsf{CNet}^{40}_{\mathsf{boost}}$) entropy-based CNets taken from [20][5]. Additionally, we learn an ensemble of 40 bagged CNets learned with dCSN as in [9] (dCSN^{40}) with a grid search over $\delta \in \{1000, 2000\}$, $\alpha \in \{0.1, 0.2\}$ and $\sigma = 4$. Lastly, we train an ensemble of 40 XCNets (XCNet^{40}) and another ensemble of 40 XCNet with PoBs as leaf distributions by running a grid search over $\delta \in \{300, 500, 1000, 2000\}$, $\alpha \in \{0.1, 0.2, 0.5, 1, 2\}$ and $\sigma = 4$. For

[5] Note that we report the best log-likelihood across more than one algorithmic variant, hence these results can be considered to be derived from models optimized over more parameters.

Table 3. Average test log likelihoods for all ensembles and other competitors.

Dataset	Ensembles					Competitors			
	CNet^{40}	CNet$^{40}_{\text{boost}}$	dCSN^{40}	XCNet$^{40}_{\text{PoB}}$	XCNet^{40}	XCNet^{500}	ID-SPN	ACMN	WM
NLTCS	**−6.00**	−6.01	**−6.00**	−6.01 ± 0.00	**−6.00** ± 0.00	**−5.99**	−6.02	−6.00	−6.02
MSNBC	−.08	−6.15	**−6.05**	−6.11 ± 0.00	−6.06 ± 0.00	−6.06	**−6.04**	**−6.04**	**−6.04**
KDDCup2k	−2.14	−2.15	−2.15	**−2.13** ± 0.00	**−2.13** ± 0.00	**−2.13**	**−2.13**	−2.17	−2.16
Plants	−12.32	−12.67	−12.59	−13.09 ± 0.01	**−11.99** ± 0.00	**−11.84**	−12.54	−12.80	−12.65
Audio	−40.09	−39.84	−40.19	−40.30 ± 0.02	**−39.77** ± 0.02	**−39.39**	−39.79	−40.32	−40.50
Jester	−52.88	−52.82	−52.99	−53.64 ± 0.03	**−52.65** ± 0.02	**−52.21**	−52.86	−53.31	−53.85
Netflix	−56.55	−56.44	−56.69	−57.64 ± 0.03	**−56.38** ± 0.03	**−55.93**	−56.36	−57.22	−57.03
Accidents	−29.88	−29.45	**−29.27**	−36.92 ± 0.05	−29.31 ± 0.02	−29.10	−26.98	−27.11	**−26.32**
Retail	−10.84	**−10.81**	−11.17	−10.88 ± 0.00	−10.93 ± 0.01	−10.91	−10.85	−10.88	−10.87
Pumsb-star	−23.98	−23.46	−23.78	−32.91 ± 0.02	**−23.44** ± 0.01	−23.31	−22.41	−23.55	**−21.72**
DNA	**−81.07**	−85.67	−85.95	−98.28 ± 0.06	−84.96 ± 0.03	−84.17	−81.21	**−80.03**	−80.65
Kosarek	−10.74	**−10.60**	−10.97	−10.91 ± 0.01	−10.72 ± 0.01	−10.66	**−10.60**	−10.84	−10.83
MSWeb	−9.77	−9.74	−9.93	−9.83 ± 0.00	**−9.66** ± 0.01	**−9.62**	−9.73	−9.77	−9.70
Book	−35.55	**−34.46**	−37.38	−34.77 ± 0.02	−36.35 ± 0.08	−35.45	**−34.14**	−35.56	−36.41
EachMovie	−53.00	**−51.53**	−54.14	−51.66 ± 0.11	−51.72 ± 0.12	**−50.34**	−51.51	−55.80	−54.37
WebKB	−153.12	**−152.53**	−155.47	−155.83 ± 0.30	−153.01 ± 0.28	**−149.20**	−151.84	−159.13	−157.43
Reuters-52	−83.71	**−83.69**	−86.19	−85.16 ± 0.15	−84.05 ± 0.24	**−81.87**	−83.35	−90.23	−87.55
20NewsG	−156.09	−153.12	−156.46	**−152.21** ± 0.19	−153.89 ± 0.15	**−151.02**	−151.47	−161.13	−158.95
BBC	**−237.42**	−247.01	−248.84	−251.31 ± 0.52	−238.47 ± 0.69	**−229.21**	−248.93	−257.10	−257.86
Ad	−15.28	−14.36	−15.55	−26.25 ± 0.08	**−14.20** ± 0.08	**−14.00**	−19.05	−16.53	−18.35
Avrg rank	**2.7**	**2.3**	**3.85**	**3.9**	**1.95**				
	4.7	**4.35**	**6.4**	**6.75**	**3.95**	**2.2**	**3.15**	**6.35**	**6.05**

these two random models Table 3 reports the average and the standard deviation over 10 different runs. Note that we are not performing bagging for our XCNet ensembles, since we do not draw bootstrapped samples of the data. This is motivated by the intuition that randomization is a form of diversification in the ensemble by itself, and it has been confirmed with a preliminary experimentation.

Next we compare CNet ensembles to other state-of-the-art TPMs learned by employing much more sophisticated models as ID-SPN [22], ACMN [17]. The first learns a complex hybrid architecture of SPNs and ACs while the latter learns high treewidth MNs represented as tractable ACs. Lastly, we employ the WinMine toolkit (WM) [3]. WM learns a treewidth *unbounded* BN exploiting context sensitive independencies by modeling its CPTs as trees. These models results are taken from [22]. The 40 component ensemble XCNet^{40} already delivers log-likelihoods comparable to those of the aforementioned models on more than half datasets. Nevertheless, we investigate the effect of building a large ensemble, up to 500 components (XCNet^{500}) by running a grid search over $\delta \in \{300, 500, 1000, 2000\}$, $\alpha = 0.1$ and $\sigma = 4$. On many datasets the log-likelihood scores of such an ensemble are the best achieved in the literature. Compared to XCNet^{40}, results from XCNet^{500} generally improve, however, on datasets like `Nltcs` and `KDDCup2k` the improvement saturated, suggesting that adding more components does not diversify the ensemble anymore. It is worth noting that XCNet$^{40}_{\text{PoB}}$ is competitive on half datasets against a far more complex model like WM, yet outperforming it in terms of speed of learning and inference.

Table 4. Numbers of victories for the algorithms on the rows w.r.t those on columns.

Model	CNet^{40}	CNet$^{40}_{\text{boost}}$	dCSN^{40}	XCNet$^{40}_{\text{PoB}}$	XCNet^{40}	XCNet^{500}	ID-SPN	ACMN	WM	**Avrg**
CNET^{40}		7	15	16	6	1	7	14	15	**10.12**
CNET$^{40}_{\text{boost}}$	13		15	16	7	3	4	16	14	**11.00**
dCSN^{40}	4	4		12	2	1	3	11	12	**6.12**
XCNet$^{40}_{\text{PoB}}$	4	3	8		4	2	1	7	9	**4.75**
XCNet^{40}	13	13	17	15		0	7	14	15	**11.75**
XCNet^{500}	19	17	19	17	18		12	16	15	**16.62**
ID-SPN	13	15	17	18	11	7		16	13	**13.75**
ACMN	4	4	8	12	5	4	3		7	**5.87**
WM	5	6	8	11	5	5	5	12		**7.12**

We summarize comparisons among the algorithms in the first half of the table (resp. all algorithms) through ranking over the twenty datasets. For each dataset, we ranked the performance of the algorithms in the first half of the table (resp. all the algorithms) from 1 to 5 (resp. 9). The average rank of the algorithms is reported in the last two rows of the Table 3, showing that a mixture of XCNets performs the best. Finally, Table 4, reporting the number of victories for each algorithm w.r.t. the others, shows again the performances of mixtures of XCNet against the competitors that obtains 16.62 victories on average.

Table 5. Times (in seconds) taken to learn the best models on each dataset for dCSN, XCNet, dCSN$_{\text{PoB}}$, XCNet$_{\text{PoB}}$, their ensembles and ID-SPN with default parameters.

Dataset	dCSN	XCNet	dCSN$_{\text{PoB}}$	XCNet$_{\text{PoB}}$	dCSN^{40}	XCNet$^{40}_{\text{PoB}}$	XCNet^{40}	XCNet^{500}	ID-SPN
NLTCS	0	0.2	0.1	0.01	10	0.2	0	3	310
MSNBC	12	0.3	0.7	0.01	499	13.1	13	155	46266
KDDCup2k	112	0.5	12.0	0.32	4126	21.2	16	247	32067
Plants	15	0.3	45.5	0.22	325	1.0	6	77	18833
Audio	58	0.3	74.8	0.48	980	0.8	6	136	21009
Jester	50	0.2	95.6	0.26	989	0.3	4	83	10412
Netflix	75	0.2	2.8	0.02	1546	0.4	9	118	30294
Accidents	54	0.2	153.7	0.04	996	0.7	11	138	15472
Retail	263	0.8	5.8	0.01	3780	3.2	13	164	4041
Pumsb-star	118	0.6	26.2	0.02	2260	0.8	23	290	20952
DNA	30	0.1	4.4	0.01	224	0.06	3	40	3040
Kosarek	588	2.4	41.2	0.01	10033	10.8	43	524	17799
MSWeb	1215	7.2	7.4	0.01	17123	13.2	129	1592	19682
Book	9235	9.7	113.0	0.04	155634	1.9	316	3476	61248
EachMovie	1297	7.1	4.7	0.01	16962	1.1	127	2601	118782
WebKB	4997	11.0	238.0	0.03	18875	0.9	190	2237	45451
Reuters-52	9947	39.3	24.3	0.05	65498	2.7	414	8423	70863
20NewsG	16866	51.3	40.7	0.01	153908	4.4	506	9883	163256
BBC	21381	8.4	7.3	0.02	69572	0.4	256	4251	61471
Ad	5212	116.5	134.0	0.08	75694	4.2	2403	30538	87522

5.3 (Q3) Running Times

We derived the complexity for all considered variants of CNet learning schemes thus proving that XCNets are the ones scaling better w.r.t. the number of the features. Nevertheless, we empirically analyze XCNets learning times since we want (i) to evaluate whether and how much learning the leaf distribution actually impacts on real data, (ii) to compare the learning times of the density estimators employed in the previous sections. While a non-theoretical comparison may fall into the pitfalls of comparing different programming languages and optimization schemes, we provide it as a rule of thumb for practitioners to decide on which off-the-shelf density estimator toolbox to use.

In Table 5 we report the time, in seconds, spent by each algorithm to learn the best model on each dataset. Even increasing the number of components one order of magnitude more than what competitors are able to do in a reasonable time, XCNet still learn a competitive model (see Table 3) in time lesser than that of the competitors (see for instance the comparison w.r.t. ID-SPN).

6 Conclusions

We introduced XCNets, simplifying CNet learning through random conditioning. When learned in ensembles, XCNets achieve the new state-of-the-art results for density estimation on several benchmark datasets. Due to their simplicity to implement, fast learning times, and accurate inference performances, XCNets set the new baseline to compare against for density estimation with TPMs. As future work we plan to exploit their mixture of experts interpretation to devise more expressive gating functions that still perform exact and fast inference.

References

1. Bekker, J., Davis, J., Choi, A., Darwiche, A., Van den Broeck, G.: Tractable learning for complex probability queries. In: NIPS (2015)
2. Boutilier, C., Friedman, N., Goldszmidt, M., Koller, D.: Context-specific independence in Bayesian networks. In: UAI (1996)
3. Chickering, M.: The Winmine Toolkit. Microsoft, Redmond (2002)
4. Choi, A., Van den Broeck, G., Darwiche, A.: Tractable learning for structured probability spaces: a case study in learning preference distributions. In: IJCAI (2015)
5. Chow, C., Liu, C.: Approximating discrete probability distributions with dependence trees. IEEE Trans. Inf. Theory **14**, 462–467 (1968)
6. Darwiche, A.: A differential approach to inference in Bayesian networks. JACM **50**, 280–305 (2003)
7. Di Mauro, N., Vergari, A., Esposito, F.: Multi-label classification with cutset networks. In: PGM (2016)
8. Di Mauro, N., Vergari, A., Basile, T.: Learning Bayesian random cutset forests. In: ISMIS (2015)
9. Di Mauro, N., Vergari, A., Esposito, F.: Learning accurate cutset networks by exploiting decomposability. In: AIXIA (2015)

10. Geurts, P., Ernst, D., Wehenkel, L.: Extremely randomized trees. MLJ **63**, 3–42 (2006)
11. Haaren, J.V., Davis, J.: Markov network structure learning: a randomized feature generation approach. In: AAAI (2012)
12. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning. Springer, Heidelberg (2009). https://doi.org/10.1007/978-0-387-21606-5
13. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. MIT Press, Cambridge (2009)
14. Larochelle, H., Murray, I.: The neural autoregressive distribution estimator. In: AISTATS (2011)
15. Lowd, D., Davis, J.: Learning Markov network structure with decision trees. In: ICDM (2010)
16. Lowd, D., Domingos, P.: Naive Bayes models for probability estimation. In: ICML (2005)
17. Lowd, D., Rooshenas, A.: Learning Markov networks with arithmetic circuits. In: AISTATS (2013)
18. Meil, M., Jordan, M.I.: Learning with mixtures of trees. JMLR **1**, 1–48 (2000)
19. Poon, H., Domingos, P.: Sum-product network: a new deep architecture. In: NIPS Workshop on Deep Learning and Unsupervised Feature Learning (2011)
20. Rahman, T., Gogate, V.: Learning ensembles of cutset networks. In: AAAI (2016)
21. Rahman, T., Kothalkar, P., Gogate, V.: Cutset networks: a simple, tractable, and scalable approach for improving the accuracy of Chow-Liu trees. In: ECML/PKDD (2014)
22. Rooshenas, A., Lowd, D.: Learning sum-product networks with direct and indirect variable interactions. In: ICML, pp. 710–718 (2014)
23. Roth, D.: On the hardness of approximate reasoning. AI **82**, 273–302 (1996)
24. Scanagatta, M., Corani, G., de Campos, C.P., Zaffalon, M.: Learning treewidth-bounded Bayesian networks with thousands of variables. In: NIPS (2016)
25. Theis, L., van den Oord, A., Bethge, M.: A note on the evaluation of generative models. In: ICLR (2016)
26. Vergari, A., Di Mauro, N., Esposito, F.: Simplifying, regularizing and strengthening sum-product network structure learning. In: ECML/PKDD (2015)