Learning Bayesian Random Cutset Forests

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

[1] Department of Computer Science, University of Bari "Aldo Moro"
Via E. Orabona 4, I-70125 Bari, Italy
[2] Department of Physics, University of Bari "Aldo Moro"
Via G. Amendola 173, I-70126 Bari, Italy
{nicola.dimauro,antonio.vergari,teresamaria.basile}@uniba.it

Abstract. In the Probabilistic Graphical Model (PGM) community there is an interest around tractable models, i.e., those that can guarantee exact inference even at the price of expressiveness. Structure learning algorithms are interesting tools to automatically infer both these architectures and their parameters from data. Even if the resulting models are efficient at inference time, learning them can be very slow in practice. Here we focus on Cutset Networks (CNets), a recently introduced tractable PGM representing weighted probabilistic model trees with tree-structured models as leaves. CNets have been shown to be easy to learn, and yet fairly accurate. We propose a learning algorithm that aims to improve their average test log-likelihood while preserving efficiency during learning by adopting a random forest approach. We combine more CNets, learned in a generative Bayesian framework, into a generative mixture model. A thorough empirical comparison on real word datasets, against the original learning algorithms extended to our ensembling approach, proves the validity of our approach.

1 Introduction

A key task in *Probabilistic Graphical Models* (PGMs) [11] is *inference*, i.e., the possibility to answer queries about the probability of observing some states of the random variables whose joint distribution is compactly represented in the model. *Exact* inference is known to be a hard task in general, but even approximate inference routines are unfeasible sometimes [22]. In order to gain efficiency, and preserving exactness, one has to renounce expressiveness, that is coping with the possibility of not capturing all the independencies among random variables. The growing data availability demands PGMs able to guarantee tractable inference leading to *tractable* PGMs, an accepted trade-off in the AI community.

Works on learning tractable models structure from data date back to *tree-structured models*, like those inferred by the classical Chow-Liu algorithm [4]. Recently, more accurate tractable PGMs have been proposed such as extensions of tree-structured models by composing them in mixtures [16] or by introducing latent variables [3], Arithmetic Circuits (ACs) capturing the expressiveness of more complex Bayesian and Markov Networks [13,14], or by compacting latent interactions in a deep architecture as done by Sum-Product Networks (SPNs) [18].

As the expressiveness of these models increases, the complexity of learning their structure increases as well, being, in general, the problem formulated as a search in the structure space guided by complex statistical independence tests.

With the objective of making structure learning efficient, *Cutset Networks* (CNets) have been recently introduced in [19] as tractable PGMs embedding Pearl's conditioning algorithm [17]. They are weighted probabilistic model trees in the form of OR-trees having tree-structured models as leaves, and non-negative weights on inner edges emanating from inner OR nodes, representing conditioning on the values of the random variables associated to those nodes. Structure learning in [19] corresponds to a greedy top-down search process in the OR trees space that leverages decision tree learning heuristics to determine the random variable to condition on at each step. The corresponding proposed algorithm, CNet, recursively partitions the data instances into subsets by selecting heuristically the best variable that maximizes an approximation of a reformulation of the information gain based on the joint entropy. Once this variable has been found, the algorithm creates a corresponding inner OR node and proceeds until no more splits can be done, in which case a tree learned with the Chow-Liu algorithm [4] is introduced as a leaf node to approximate the distribution on the current data partition. The cheap computation of an entropy based heuristic makes learning efficient, however the resulting models are *far* from being accurate as density estimators compared to other PGM structure learners [16,21]. Introducing mixtures of CNets learned via the Expectation Maximization algorithm (EM) lead to really competitive results. Such results shed light on the trade-off between the simplicity of such models and their accuracy: to effectively make them accurate one cannot ignore direct likelihood maximization and shall recur to ensemble techniques, making, in the end, learning more expensive if not complex.

In [7] a more principled way to learn CNets has been proposed. Structure learning is reformulated in a Bayesian framework as a likelihood-principled search guided by the Bayesian Information Criterion (BIC). A tractable model score, obtained by exploiting the decomposability of the likelihood of such models, enables the feasibility of finding the best feature for an OR split by directly evaluating part of the model on a portion of the data. Regularization is achieved both by the introduction of the BIC score and by putting informative Dirichlet priors on the probability parameters. The introduced algorithm, dCSN, proved to be more accurate than the original one. As expected, mixtures of models learned in this way, ensembled via *bagging* [10], outperformed both CNet and dCSN, again at the expense of learning time.

Here we extend the work in [7], by learning ensembles of CNets as density estimators by adopting a *random forest* approach [2]. Our main objective is to greatly reduce the evaluation time to choose the best variable to split on in dCSN by considering only a fraction of all candidates variables. Nevertheless, such state space reduction could potentially lead to less than optimal models, likelihood-wise, in our framework. We devised a fair and thorough set of experiments to test our proposed approach along the two dimensions of accuracy and time. We evaluated our mixture models on 18 real world datasets against the original algorithm as proposed in [19] extended to our ensemble framework, proving that not only learning times are reduced, but also the model accuracy increased.

2 Cutset Networks

Let $\mathcal{D} = \{\xi_1, \ldots, \xi_M\}$ be a set of M i.i.d. instances over $\mathbf{X} = \{X_1, \ldots, X_n\}$ discrete random variables (features), whose domains are the sets $\mathrm{Val}(X_i) = \{x_i^j\}_{j=1}^{k_i}, i = 1, \ldots, n$. We denote as $\xi_m[X_i]$ the value assumed by an instance ξ_m in correspondence to a particular variable X_i .

A *Cutset Network* (CNet) is a pair $\langle \mathcal{G}, \boldsymbol{\gamma} \rangle$, where $\mathcal{G} = \mathcal{O} \cup \{\mathcal{T}_1, \ldots, \mathcal{T}_L\}$ is the graphical structure, composed by a rooted OR tree, $\mathcal{O}$, and by leaf trees $\mathcal{T}_l$; and $\boldsymbol{\gamma} = \boldsymbol{w} \cup \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_L\}$ is the parameter set containing the OR tree weights $\boldsymbol{w}$ and the leaf tree parameters $\boldsymbol{\theta}_l$. The *scope* of a CNet $\mathcal{G}$ (resp. a leaf tree $\mathcal{T}_l$), denoted as $\mathrm{scope}(\mathcal{G})$ (resp. $\mathrm{scope}(\mathcal{T}_l)$), is the set of random variables that appear in it. Each node in the OR tree is labeled by a variable X_i, and each edge emanating from it represents the conditioning of X_i by a value $x_i^j \in Val(X_i)$, weighted by the conditional probability $w_{i,j}$ of conditioning the variable X_i to the value x_i^j.

A CNet can be thought of a model tree associating to each instance a weighted probabilistic leaf model. From now on, for the sake of simplicity, we will refer to the leaf trees as CLtrees as they may be learned by the classical Chow-Liu algorithm [4] as the trees best approximating a probability distribution from data in the terms of the Kullback-Leibler divergence, as done in [19].

In [7] a recursive definition of a CNet has been proposed along with the resulting log-likelihood and BIC score decomposition, leading to the principled dCSN algorithm for learning the structure of CNets, that we briefly review here.

Definition 1 (Cutset network [7]). *Let* $\mathbf{X}$ *be a set of discrete variables, a* Cutset Network *is defined as follows:*

1. *a CLtree, with scope* $\mathbf{X}$*, is a CNet;*
2. *given* $X_i \in \mathbf{X}$ *a variable with* $|Val(X_i)| = k$*, graphically conditioned in an OR node, a weighted disjunction of* k *CNets* $\mathcal{G}_i$ *with same scope* $\mathbf{X}_{\setminus i}$ *is a CNet, where all weights* $w_{i,j}$*,* $j = 1, \ldots, k$*, sum up to one, and* $\mathbf{X}_{\setminus i}$ *denotes the set* $\mathbf{X}$ *minus the variable* X_i*.*

Following this definition, the computation of the log-likelihood of a CNet can be decomposed as follows [7]. Given a CNet $\langle \mathcal{G}, \boldsymbol{\gamma} \rangle$ over variables $\mathbf{X}$ and a set of instances $\mathcal{D}$, its log-likelihood $\ell_{\mathcal{D}}(\langle \mathcal{G}, \boldsymbol{\gamma} \rangle)$ can be computed as:

$$\ell_{\mathcal{D}}(\langle \mathcal{G}, \boldsymbol{\gamma} \rangle) = \begin{cases} \sum_{\xi \in \mathcal{D}} \sum_{i=1}^{n} \log P(\xi[X_i] | \xi[\mathrm{Pa}_i]), & \text{if } \mathcal{G} = \{\mathcal{T}\} \\ \sum_{j=1}^{k} M_j \log w_{i,j} + \ell_{\mathcal{D}_j}(\langle \mathcal{G}_j, \boldsymbol{\gamma}_{\mathcal{G}_j} \rangle), & \text{otherwise,} \end{cases} \tag{1}$$

where the first equation refers to the case of a CNet composed by a single CLtree, while the second one specifies the the case of an OR tree rooted on the variable X_i, with $|Val(X_i)| = k$, where, for each $j = 1, \ldots, k$, $\mathcal{G}_j$ is the CNet involved in the disjunction with parameters $\boldsymbol{\gamma}_{\mathcal{G}_j}$, $\mathcal{D}_j = \{\xi \in \mathcal{D} : \xi[X_i] = x_i^j\}$ is the slice of $\mathcal{D}$ after conditioning on X_i, and $M_j = |\mathcal{D}_j|$ its cardinality. $\ell_{\mathcal{D}_j}(\langle \mathcal{G}_j, \boldsymbol{\gamma}_{\mathcal{G}_j}\rangle)$ denotes the log-likelihood of the sub-CNet $\mathcal{G}_j$ on $\mathcal{D}_j$.

The dCSN algorithm, proposed in [7], exploits a different approach from that in [19], avoiding decision tree heuristics and instead choosing the best variable by directly maximizing the data log-likelihood. By exploiting the recursive nature of Definition 1, a CNet is grown top-down allowing further expansion, i.e., substituting a CLtree with an OR node only if it improves the log-likelihood (it is clear that maximizing the second term in Equation 1 results in maximizing the global score). In detail, dCSN starts with a single CLtree, for variables $\mathbf{X}$, learned from $\mathcal{D}$ and it checks whether there is a decomposition, i.e. an OR node applied on as many CLtrees as the values of the best variable X_i, providing a better log-likelihood than that scored by the initial tree. If a such decomposition exists, than the decomposition process is recursively applied to the sub-slices $\mathcal{D}_i$, testing each leaf for a possible substitution.

In order to penalize complex structures and thus avoiding overfitting, a Bayesian Information Criterion (BIC) [8] has been adopted. The BIC score of a CNet $\langle \mathcal{G}, \boldsymbol{\gamma}\rangle$ on data $\mathcal{D}$ is defined as $\text{score}_{\text{BIC}}(\langle \mathcal{G}, \boldsymbol{\gamma}\rangle) = \log P_{\mathcal{D}}(\langle \mathcal{G}, \boldsymbol{\gamma}\rangle) - \frac{\log M}{2}\text{Dim}(\mathcal{G})$, where $\text{Dim}(\mathcal{G})$ is the model dimension, set to the number of OR nodes appearing in $\mathcal{G}$, and M is the size of the dataset $\mathcal{D}$. Given $\mathcal{G}$ and $\mathcal{G}'$ be two CNets, where $\mathcal{G}'$ has been obtained from $\mathcal{G}$ substituting a leaf tree by adding a new sub-CNet rooted in an OR node, then: $\text{score}_{\text{BIC}}(\langle \mathcal{G}', \boldsymbol{\gamma}'\rangle) - \text{score}_{\text{BIC}}(\langle \mathcal{G}, \boldsymbol{\gamma}\rangle) = \ell_{\mathcal{D}}(\langle \mathcal{G}', \boldsymbol{\gamma}'\rangle) - \ell_{\mathcal{D}}(\langle \mathcal{G}, \boldsymbol{\gamma}\rangle) - \frac{\log M}{2}$. Hence, $\mathcal{G}'$ is accepted when $\ell_{\mathcal{D}}(\langle \mathcal{G}', \boldsymbol{\gamma}'\rangle) - \ell_{\mathcal{D}}(\langle \mathcal{G}, \boldsymbol{\gamma}\rangle) > \frac{\log M}{2}$, i.e., a leaf node may be decomposed when the improvement on the global loglikelihood is greater than $\frac{\log M}{2}$.

Proposition 1 (CNet BIC score decomposition [7]). *Given a CNet $\langle \mathcal{G}, \boldsymbol{\gamma}\rangle$, over variables $\mathbf{X}$ and instances $\mathcal{D}$, made up of $\{\mathcal{T}_l\}_{l=1}^{L}$ CLtrees, a decomposition of a tree $\mathcal{T}_l$, having scope $\mathbf{X}_l \subset \mathbf{X}$, with parameters $\boldsymbol{\theta}_l$, with a sub-CNet $\mathcal{G}_i$ rooted in a OR node associated to the variable $X_i \in \mathbf{X}_l$ with parameters $\boldsymbol{\gamma}_i$, leading to a new CNet $\langle \mathcal{G}', \boldsymbol{\gamma}'\rangle$, is accepted iff $\ell_{\mathcal{D}_l}(\langle \mathcal{G}_i, \boldsymbol{\gamma}_i\rangle) - \ell_{\mathcal{D}_l}(\langle \mathcal{T}_l, \boldsymbol{\theta}_l\rangle) > \frac{\log M}{2}$, where $M = |\mathcal{D}|$, and D_l is the slice of D containing only instances associated to $\mathcal{T}_l$.*

Again, due to the decomposability of the likelihood score, instead of recomputing it on the complete dataset $\mathcal{D}$, we can evaluate only the local improvements.

A Bayesian approach to learn a CLtrees $\mathcal{T}$ with parameters $\boldsymbol{\theta}$ from data $\mathcal{D}$ has been adopted, by exploiting as scoring function $P(\boldsymbol{\theta}|\mathcal{D}) \approx P(\mathcal{D}|\boldsymbol{\theta})P(\boldsymbol{\theta})$. Considering Dirichlet priors, and indicating with Pa_i the parent variable of X_i, the regularized model parameter estimates are:

$$\theta_{x_i|\text{Pa}_i} \approx E_{P(\theta_{x_i|\text{Pa}_i}|\mathcal{D},\mathcal{T})}[\theta_{x_i|\text{Pa}_i}] = \frac{M_{x_i,\text{Pa}_i} + \alpha_{x_i|\text{Pa}_i}}{M_{\text{Pa}_i} + \alpha_{\text{Pa}_i}},$$

where $M_{\mathbf{z}}$ is the number of entries in a dataset $\mathcal{D}_{\mathbf{z}}$ having the set of variables $\mathbf{Z}$ instatiated to $\mathbf{z}$. As pointed out in [8], a different Dirichlet prior for each distribution of X_i given a particular value of its parents may be used, leading to choosing the regularized parameter estimates as:

$$\hat{\boldsymbol{\theta}}_{X_i|\mathrm{Pa}_i} = \frac{M \cdot P(\mathrm{Pa}_i)P(X_i|\mathrm{Pa}_i)}{M \cdot P(\mathrm{Pa}_i) + \boldsymbol{\alpha}_{X_i|\mathrm{Pa}_i}} + \frac{\boldsymbol{\alpha}_{X_i|\mathrm{Pa}_i}\boldsymbol{\theta}^0(X_i|\mathrm{Pa}_i)}{M \cdot P(\mathrm{Pa}_i) + \boldsymbol{\alpha}_{X_i|\mathrm{Pa}_i}},$$

where $\boldsymbol{\theta}^0(X_i|\mathrm{Pa}_i)$ is the prior estimate of $P(X_i|\mathrm{Pa}_i)$ and $\boldsymbol{\alpha}_{X_i|\mathrm{Pa}_i}$ is the confidence associated with that prior. A reasonable choice uses the marginal probability of X_i in the data as the prior probability. Thus, $\boldsymbol{\theta}^0(X_i|\mathrm{Pa}_i)$ has been set to $P_D(X_i)$, and with fixed $\alpha_{x_i|\mathrm{Pa}_i} = \alpha$, then $\hat{\theta}_{X_i|\mathrm{Pa}_i}$ has been set to $\frac{M_{x_i,\mathrm{Pa}_i}+\alpha P_D(X_i)}{M_{\mathrm{Pa}_i}+\alpha}$.

The dCSN algorithm starts by learning a single CLTree on the whole dataset $\mathcal{D}$, and then calls a *decomposition* procedure on this tree. Two input parameters, δ and σ, act as regularizers, halting the decomposition process by requiring a minimum number of instances, resp. of features, in a slice to split it. The aim of dCSN is to attempt to extend the model by replacing one of the CLtree leaf nodes with a new CNet on the same variables. In particular, the decomposition prodecure checks for each variable X_i on the slice $\mathcal{D}$, whether the OR decomposition associated to that variable (a new CNet) has a better score than that of the input CLtree. If a better decomposition is found, it then recursively tries to decompose the sub-CLtrees of the newly introduced CNet. It is clear that the evaluation of all possible decompositions, i.e. of all possible X_i, even if the likelihood score is decomposable, extends learning time.

3 Random Forests of CNets

The joint probability distribution defined by a mixture K of probabilistic models can be formulated as $P(\mathbf{X}) = \sum_{i=1}^{K} \mu_i P(\mathbf{X} : \mathcal{Q}_i)$, where $\{\mathcal{Q}_i\}_{i=1}^{K}$ are the mixture *components* and $\{\mu_i\}_{i=1}^{k}$ are their *responsibilities*, i.e. the positive weights summing to one representing the contribution of each component to the final model. In our context, the components would be CNets, i.e. $\mathcal{Q}_i = \langle \mathcal{G}_i, \boldsymbol{\gamma}_i \rangle$; instead, for MT [16], a highly accurate algorithm learning mixtures of tree structured models, they would be CLTrees.

The most common algorithmic choice when learning the structure of a mixture of PGMs is the Expectation Maximization algorithm EM [6]. In a nutshell, by considering the mixture responsibilities as the probabilities of observing as many values of a latent variable, the optimization carried out by EM involves the iteration of two steps: the computation of the expected mixture components from the current distribution, followed by the maximization of the likelihood as a function of the responsibilities given the previously computed components [16,19]. To shorten unfeasible learning times, since learning the components structure from zero is quite expensive, in practice, only the responsibilities are updated, having the components structures being learned only once, at the first iteration.

In [7] we explored another direction in mixture structure learning, ensembling techniques exploiting bagging [10]. Each component is learned once on bootstrapped sample $\mathcal{D}_{B_i}$ of the initial data $\mathcal{D}$, with $|\mathcal{D}_{B_i}| = |\mathcal{D}|$, and their responsibilities are computed as being proportional to the likelihood score they

Algorithm 1 dCSN-RF($\mathcal{D}$, $\mathbf{X}$, α_f, δ, σ, K)

```
1: Input: instances $\mathcal{D}$ over features $\mathbf{X}$; $\alpha_f \in [0,1]$: ESS factor; $\delta$ min num of instances
   to split, $\sigma$ min num of features to split, $K$ num of the mixture components
2: Output: a mixture of $K$ CNets components $\{\langle \mathcal{G}_i, \boldsymbol{\gamma}_i \rangle\}_{i=1}^{K}$ with responsibilities
   $\{\mu_i\}_{i=1}^{K}$, encoding a pdf over $\mathbf{X}$ learned from $\mathcal{D}$
3: $\alpha \leftarrow \alpha_f |\mathcal{D}|, \hat{\mu} \leftarrow 0$
4: for $i = 1, \ldots, K$ do
5:     $\mathcal{D}_{B_i} \leftarrow$ bootstrapSample($\mathcal{D}$)
6:     $\langle \mathcal{T}, \boldsymbol{\theta} \rangle \leftarrow$ LearnCLTree($\mathcal{D}_{B_i}, \mathbf{X}, \alpha$)
7:     $\boldsymbol{w} \leftarrow \emptyset$
8:     $\langle \mathcal{G}_i, \boldsymbol{\gamma}_i \rangle \leftarrow$ decompose-RF($\mathcal{D}_{B_i}, \mathbf{X}, \alpha, \mathcal{T}, \boldsymbol{\theta}, \boldsymbol{w}, \delta, \sigma$)
9:     $\mu_i \leftarrow \ell_{\mathcal{D}}(\langle \mathcal{G}_i, \boldsymbol{\gamma}_i \rangle)$, $\hat{\mu} \leftarrow \hat{\mu} + \mu_i$
10: for $i = 1, \ldots, K$ do
11:     $\mu_i \leftarrow \mu_i / \hat{\mu}$
```

got on $\mathcal{D}_{B_i}$, resulting in the more robust estimate for each instance: $\hat{P}(\xi) = \sum_{i=1}^{K} \mu_i P(\xi : \langle \mathcal{G}_i, \boldsymbol{\gamma}_i \rangle)$, where $\mu_i = \ell_{\mathcal{D}}(\langle \mathcal{G}_i, \boldsymbol{\gamma}_i \rangle) / \sum_{j=1}^{K} \ell_{\mathcal{D}}(\langle \mathcal{G}_j, \boldsymbol{\gamma}_j \rangle)$. A sketch of our bagged approach in dCSN-RF is visible in Algorithm 1, where LearnCLTree is the procedure, as already reported in [7], learning leaf trees implementing the Chow-Liu algorithm [4] and smoothing (α) probability estimations.

As the complexity of such approach grows linearly in the number of components, a natural time saving extension would be to consider only a portion of the features while growing the CNet, and then select the best one among them. The extension of the decompose procedure used in dCSN [7], as described in the previous section, to dCSN-RF is outlined in Algorithm 2. The difference now lies in line 5, where only a random subset of $\mathbf{X}$ is evaluated. We choose to sample $\sqrt{|\mathbf{X}|}$ as suggested in [10]. While this can clearly lead to sub optimal structures when a single component is involved, it would derive a better density estimation by forcing each of them to specialize on a particular, partial, view of the joint distribution as the number of components increases. Adopting this strategy within the bagging framework equals to a *random forest* approach [2] translated in a Bayesian generative framework. Random forests have been successfully applied in discriminative tasks like classification and regression, leading to accurate, low variance ensembles [10]. They have also recently being proposed as density estimators in [5]. More in general, our approach is affine to those shown in [20,1], where mixtures of tree structured density estimators are learned by perturbing their components via bagging and random subspace combination. Here we are willingly to trade off accuracy for shorter learning times.

4 Experiments

We evaluated dCSN-RF against the mixture version in our Bayesian framework, using only bagging, dCSN-B, and the mixture extensions of the original single CNet learning algorithms, as presented in [19]: CNet-B and CNet-RF for the

Algorithm 2 decompose-RF($\mathcal{D}$, $\mathbf{X}$, α, $\mathcal{T}, \boldsymbol{\theta}, \boldsymbol{w}, \delta, \sigma$)

```
1: Input: instances $\mathcal{D}$ over $\mathbf{X}$; $\alpha$: ESS; $\mathcal{T}$: a tree structured model and its parameters
   $\boldsymbol{\theta}$; $\delta$ min num of instances to split, $\sigma$ min num of features to split
2: Output: a CNet encoding a pdf over $\mathbf{X}$ learned from $\mathcal{D}$
3: if $|\mathcal{D}| > \delta$ and $|\mathbf{X}| > \sigma$ then
4:     $\ell_{\text{best}} \leftarrow -\infty$, $\mathbf{X}_R \leftarrow$ randomSubset($\mathbf{X}$, $\sqrt{|\mathbf{X}|}$)
5:     for $X_i \in \mathbf{X}_R$ do
6:         $\mathcal{G}_i \leftarrow \emptyset, \boldsymbol{w}_i \leftarrow \emptyset, \boldsymbol{\theta}_i \leftarrow \emptyset, C_i$ is the OR Node associate to $X_i$
7:         for $x_i^j \in Val(X_i)$ do
8:             $\mathcal{D}_j \leftarrow \{\xi \in \mathcal{D} : \xi[X_s] = x_s^j\}$, $w_{ij} \leftarrow |\mathcal{D}_j|/|\mathcal{D}|$
9:             $\langle \mathcal{T}_j, \boldsymbol{\theta}_{ij} \rangle \leftarrow$ LearnCLTree($\mathcal{D}_j, \mathbf{X}_{R\setminus s}, \alpha w_{ij}$)
10:            $\mathcal{G}_i \leftarrow$ addSubTree($C_i, \mathcal{T}_j$)
11:            $\boldsymbol{w}_i \leftarrow \boldsymbol{w}_i \cup \{w_{ij}\}, \boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_i \cup \{\theta_{ij}\}$
12:        $\ell_i \leftarrow \ell_{D_i}(\langle \mathcal{G}_i, \boldsymbol{w}_i \cup \boldsymbol{\theta}_i \rangle)$
13:        if $\ell_i > \ell_{\text{best}}$ and $\ell_i > \ell_{D_i}(\langle \mathcal{T}, \boldsymbol{\theta} \rangle)$ then
14:            $\ell_{\text{best}} \leftarrow \ell_i$, $X_{\text{best}} \leftarrow X_i$, $\mathcal{G}_{best} \leftarrow \mathcal{G}_i$, $\boldsymbol{\theta}_{best} \leftarrow \boldsymbol{\theta}_i, \boldsymbol{w}_{best} \leftarrow \boldsymbol{w}_i$
15:    if $\ell_{\text{best}} - \ell_{\mathcal{D}}(\langle \mathcal{T}, \boldsymbol{\theta} \rangle) > (log|\mathcal{D}|)/2$ then
16:        substitute $\mathcal{T}$ with $\mathcal{G}_i$ and set $\boldsymbol{w} \leftarrow \boldsymbol{w} \cup \boldsymbol{w}_{best}$
17:        for $x_b^j \in Val(X_{\text{best}})$ do
18:            $\mathcal{D}_j \leftarrow \{\xi \in \mathcal{D} : \xi[X_b] = x_b^j\}$
19:            decompose-RF($\mathcal{D}_j, \mathbf{X}_{\setminus\text{best}}, \alpha w_{ij}, \mathcal{T}_j, \boldsymbol{\theta}_j, \boldsymbol{w}, \delta, \sigma$)
```

bagged and random forest variants without pruning and CNetP-B and CNetP-RF for those including post pruning, the technique used in [19] to cope with the risk of overfitting. In addition to these, we employed MT [16] as one of the more solid mixture learning competitors[21,19]. We implemented all algorithms in Python[3], while for MT we used the implementation available in the Libra toolkit [15]. We tested them on an array of 18 real world datasets introduced in [12] and [9] as binary variants of frequent itemset mining, recommendation and classification datasets[4].

For all CNet learners, and for each dataset, we bootstrapped 40 dataset samples, then we learned mixture models incrementally, by adding components whose number K ranged from 5 to 40, by steps of 5, looking for the best average log-likelihood on the validation set. For CNet-B, CNet-RF, CNetP-B and CNetP-RF we set $\alpha = 1.0$ as done in [19]. For dCSN-B and dCSN-RF we employed a grid search on the validation sets for the parameters $\alpha_f \in \{5, 10\}$ and $\delta \in \{100, 200, 300, 400, 500, 1000\}$, while fixing $\sigma = 3$. For MT we reproduced the experiment in [21], setting K from 2 to 30 by steps of 2. We found out that while all bagged models achieved the best validation scores with $K = 40$, the best value for this parameter for MT greatly oscillates from 2 to 30, highlighting how EM could lead to different local optima.

[3] Source code is available at `http://www.di.uniba.it/~ndm/dcsn/`.

[4] All experiments have been run on a 4-core Intel Xeon E312xx (Sandy Bridge) @2.0 GHz with 8Gb of RAM and Ubuntu 14.04.1, kernel 3.13.0-39.

	CNet-B	CNet-RF	CNetP-B	CNetP-RF	dCSN-B	dCSN-RF	MT
NLTCS	154	111	224	138	144	87	290
MSNBC	2458	1911	2997	1825	2887	1783	8645
Plants	724	345	847	361	2247	441	7414
Audio	1789	571	1903	658	4992	760	6566
Jester	1375	436	1993	691	4798	551	3064
Netflix	2716	861	3604	1184	9681	1177	11402
Accidents	1308	402	1413	405	6844	859	14073
Retail	2822	585	2776	591	1838	924	320
Pumsb-star	2421	495	2480	561	12274	1257	18533
DNA	305	102	388	119	2892	254	228
Kosarek	10565	1927	10119	1888	7248	1616	18782
MSWeb	17667	2331	19425	2028	23168	3509	36076
Book	17209	1488	18028	1552	16111	2385	5918
EachMovie	8056	801	9127	978	18060	941	12100
WebKB	9610	917	11589	843	14109	1195	931
Reuters-52	34170	2381	36106	2392	68296	3256	15082
BBC	8583	415	8473	467	14144	1637	1324
Ad	7499	791	7436	829	42707	1930	6850

Table 1: Times (in seconds) taken to learn the best models on each dataset for all algorithms.

Concerning learning times (see Table 1), as expected, random forest variants were up to one magnitude order faster than the bagged versions. For instance for $K = 40$, on EachMovie the times in seconds are: 8056, 801, 9127, 978, 18060, 941 for CNet-B, CNet-RF, CNetP-B CNetP-RF, dCSN-B, dCSN-RF respectively. For all the implemented versions, we run a pairwise Wilcoxon signed rank test to

	CNet-B	CNet-RF	CNetP-B	CNetP-RF	dCSN-B	dCSN-RF	MT
NLTCS	-6.09	-6.05	-6.02	-6.01	-6.02	-6.01	**-6.01**
MSNBC	-6.06	-6.06	-6.04	-6.05	-6.04	**-6.04**	-6.08
Plants	-12.31	**-12.18**	-12.38	-12.25	-12.21	-12.27	-12.93
Audio	-42.14	-41.64	-40.68	-40.06	-40.17	**-39.96**	-40.14
Jester	-57.60	-57.43	-53.08	**-52.85**	-52.99	-52.89	-53.06
Netflix	-63.03	-62.15	-57.54	-56.89	-56.63	**-56.53**	-56.71
Accidents	-30.26	-29.75	-30.26	-29.84	**-28.99**	-29.00	-29.69
Retail	-11.00	**-10.84**	-10.88	-10.90	-10.87	**-10.82**	**-10.84**
Pumsb-star	-24.37	-23.98	-24.20	-23.93	-23.32	**-23.32**	-23.70
DNA	-91.13	-87.61	-86.88	-87.26	-84.93	**-84.83**	-85.57
Kosarek	-10.97	-10.68	-10.85	-10.66	-10.85	-10.67	**-10.62**
MSWeb	-9.96	-9.88	-9.91	-9.93	-9.86	**-9.71**	-9.82
Book	-35.91	-35.81	-35.60	-35.92	-35.92	-35.94	**-34.69**
EachMovie	-54.35	-53.35	-54.02	**-53.20**	-53.91	-53.84	-54.51
WebKB	-156.43	-155.47	-156.68	-158.59	**-155.20**	**-155.10**	-157.00
Reuters-52	-86.36	**-84.83**	-86.89	-85.24	**-85.69**	**-84.58**	-86.53
BBC	-252.26	-251.12	-257.09	-257.08	-251.14	**-249.56**	-259.96
Ad	-15.98	-16.04	-16.04	-16.04	**-13.73**	-14.35	-16.01

Table 2: Empirical risk for all algorithms.

assess the statistical significance of the scores, even if a correction to counteract the problem of multiple comparisons should be used. For instance, adopting the Bonferroni correction and testing the single six hypotheses with a significance

	CNet-B	CNet-RF	CNetP-B	CNetP-RF	dCSN-B	dCSN-RF	MT
CNet-B	-	2	3	3	0	1	5
CNet-RF	15	-	9	8	3	3	4
CNetP-B	13	7	-	6	3	1	4
CNetP-RF	12	9	9	-	5	5	5
dCSN-B	16	10	14	11	-	4	11
dCSN-RF	17	14	15	12	9	-	15
MT	12	12	11	11	5	2	-

Table 3: Numbers of statistically significant victories for the algorithms on the rows compared to those on columns

level of 0.05 corresponds to use a whole significance level of 0.3. In Table 2, in bold are reported the best values (round off to two decimal places), compared to all others, for each dataset (p-value= 0.05). As expected, the principled Bayesian learning of dCSN-B and dCSN-RF produces more accurate models than entropy based structure learners (those are the only two algorithms performing consistently better than MT, see Table 3). Moreover, it is clearly visible that the random forest approach generally *outperforms* all algorithmic variants using bagging, even in our Bayesian framework, becoming the significantly best performer on 10 datasets.

5 Conclusions

We extended the principled Bayesian approach to learn the structure of CNets proposed in [7] to a random forest ensemble framework with the main aim of reducing learning times considerably while preserving test accuracy. We proved empirically that such an approach leads to significantly more accurate models than a simple bagging scheme with the same number of components on entropy based models as well as on the Bayesian one. This, combined with the great time reduction, substantially proves our approach, making bayesian random Cutset forests an even more attractive tractable PGM.

Acknowledgements

Work supported by the project PUGLIA@SERVICE (PON02 00563 3489339) financed by the Italian Ministry of University and Research (MIUR) and by the European Commission through the project MAESTRA, grant no. ICT-2013-612944.

References

1. S. Ammar, P. Leray, B. Defourny, and L. Wehenkel. Probability density estimation by perturbing and combining tree structured Markov networks. In *Proceedings of the 10th ECSQARU*, pages 156–167. Springer, 2009.
2. L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

3. M. J. Choi, V. Y. F. Tan, A. Anandkumar, and A. S. Willsky. Learning latent tree graphical models. *Journal of Machine Learning Research*, 12:1771–1812, 2011.
4. C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.
5. A. Criminisi, J. Shotton, and E. Konukoglu. Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. *Foundations and Trends® in Computer Graphics and Vision*, (7):81–227, 2011.
6. A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society*, pages 1–38, 1977.
7. N. Di Mauro, A. Vergari, and F. Esposito. Learning accurate cutset networks by exploiting decomposability. In M. Gavanelli, E. Lamma, and F. Riguzzi, editors, *AI*IA 2015: Advances in Artificial Intelligence*, 2015.
8. N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine learning*, 29(2-3):131–163, 1997.
9. J. V. Haaren and J. Davis. Markov network structure learning: A randomized feature generation approach. In *Proceedings of the 26th Conference on Artificial Intelligence*. AAAI Press, 2012.
10. T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2009.
11. D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.
12. D. Lowd and J. Davis. Learning Markov network structure with decision trees. In *Proceedings of the 10th IEEE International Conference on Data Mining*, pages 334–343. IEEE Computer Society Press, 2010.
13. D. Lowd and P. Domingos. Learning arithmetic circuits. *CoRR*, abs/1206.3271, 2012.
14. D. Lowd and A. Rooshenas. Learning Markov networks with arithmetic circuits. In *Proceedings of the 16th International Conference on Artificial Intelligence and Statistics*, volume 31 of *JMLR Workshop Proceedings*, pages 406–414, 2013.
15. D. Lowd and A. Rooshenas. The Libra Toolkit for Probabilistic Models. *CoRR*, abs/1504.00110, 2015.
16. M. Meilă and M. I. Jordan. Learning with mixtures of trees. *Journal of Machine Learning Research*, 1:1–48, 2000.
17. J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.
18. H. Poon and P. Domingos. Sum-product network: a new deep architecture. *NIPS 2010 Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.
19. T. Rahman, P. Kothalkar, and V. Gogate. Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of Chow-Liu trees. In *Machine Learning and Knowledge Discovery in Databases*, volume 8725 of *LNCS*, pages 630–645. Springer, 2014.
20. G. Ridgeway. Looking for lumps: Boosting and bagging for density estimation. *Computational Statistics & Data Analysis*, 38(4):379–392, 2002.
21. A. Rooshenas and D. Lowd. Learning sum-product networks with direct and indirect variable interactions. In *Proceedings of the 31st International Conference on Machine Learning*, pages 710–718. JMLR Workshop and Conference Proceedings, 2014.
22. D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1–2):273–302, 1996.