Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)

Learning Ensembles of Cutset Networks

Tahrima Rahman and **Vibhav Gogate**
Department of Computer Science
The University of Texas at Dallas
Richardson, TX 75080, USA.
{tahrima.rahman,vibhav.gogate}@utdallas.edu

Abstract

Cutset networks — OR (decision) trees that have Bayesian networks whose treewidth is bounded by one at each leaf — are a new class of tractable probabilistic models that admit fast, polynomial-time inference and learning algorithms. This is unlike other state-of-the-art tractable models such as thin junction trees, arithmetic circuits and sum-product networks in which inference is fast and efficient but learning can be notoriously slow. In this paper, we take advantage of this unique property to develop fast algorithms for learning ensembles of cutset networks. Specifically, we consider generalized additive mixtures of cutset networks and develop sequential boosting-based and parallel bagging-based approaches for learning them from data. We demonstrate, via a thorough experimental evaluation, that our new algorithms are superior to competing approaches in terms of test-set log-likelihood score and learning time.

Introduction

Learning tractable probabilistic models, namely models that admit polynomial-time marginal and maximum-a-posteriori inference, has been the subject of much recent research (Poon and Domingos 2011; Gens and Domingos 2013; Rooshenas and Lowd 2014; Rahman, Kothalkar, and Gogate 2014; Choi, Van den Broeck, and Darwiche 2015). Although, these models have higher bias and are a strict subclass of the general class of probabilistic models, it has been shown in numerous empirical studies that they typically have higher predictive accuracy than the latter (Rooshenas and Lowd 2014). The key reason for this superior performance is the accuracy of inference; since exact inference in real-world probabilistic models is often computationally infeasible, one has to resort to approximate inference approaches which can be quite unreliable and inaccurate in practice. Learning tractable models helps circumvent these issues associated with approximate inference.

In this paper, we consider a sub-class of tractable models called cutset networks (Rahman, Kothalkar, and Gogate 2014) or `CNets` in short. Roughly speaking, cutset networks are rooted OR trees that have a Bayesian network whose treewidth is bounded by one at each leaf. Inference over them is linear in the size of the network using a straight-forward generalization of Pearl and Dechter's cutset conditioning algorithm (Pearl 1988; Dechter 1990). Unlike other state-of-the-art models such as arithmetic circuits (Darwiche 2003; Lowd and Domingos 2008), sum-product networks (Poon and Domingos 2011), and thin junction trees (Bach and Jordan 2001) in which learning is computationally expensive, `CNets` admit tractable learning algorithms assuming that the number of nodes in the OR tree is bounded by a constant. Moreover, fast, heuristic algorithms for learning `CNets` often yield performance that is close in terms of test-set log-likelihood to the exact learning algorithm (Rahman, Kothalkar, and Gogate 2014; Di Mauro, Vergari, and Esposito 2015). We take advantage of this fast learnability property of `CNets` to learn their ensembles, namely we develop techniques that combine multiple `CNets` to improve the prediction accuracy.

We make the following contributions. First, we develop several sequential boosting-based as well as parallel bagging-based algorithms for learning `CNets` from data leveraging vast amount of previous work on boosting and bagging algorithms (cf. (Zhou 2012)) as well as their generalizations for density estimation (Rosset and Segal 2002; Ridgeway 2002; Welling, Zemel, and Hinton 2002). Second, we perform and report on a comprehensive empirical evaluation, comparing our new algorithms with several state-of-the-art systems such as sum-product networks with direct and indirect interactions (Rooshenas and Lowd 2014), latent tree models (Choi et al. 2011) and mixtures of cutset networks (Rahman, Kothalkar, and Gogate 2014) on a wide variety of benchmark datasets. Our empirical evaluation clearly demonstrates the power of our new approaches; our new algorithms are better than competing systems on 13 out of 20 datasets in terms of test-set log-likelihood. This is significant because we compare with well-engineered, state-of-the-art systems.

Background

Let $X = \{X_1, \ldots, X_n\}$ be a set of n discrete random variables. We assume that each variable X_i can be assigned a value from a finite set Δ_i, called its domain. Let x denote an assignment to all variables in X.

OR Trees

An OR tree represents the search space explored during probabilistic inference by conditioning. Each internal node v in the OR tree is labeled with a variable. Each edge emanating from v represents the conditioning of the variable at v, say X_i, by a value $x_i \in \Delta_i$ and is labeled by the conditional probability of the variable-value assignment $X_i = x_i$ given the assignment along the path from the root node to v. The leaf nodes have no labels. An OR tree, denoted by o, represents the following probability distribution:

$$o(x) = \prod_{(v_i,v_j) \in path_o(x)} \rho(v_i, v_j) \tag{1}$$

where $path_o(x)$ is the sequence of edges from the root to a unique leaf corresponding to the assignment x and $\rho(v_i, v_j)$ is the label (conditional probability) of the edge between v_i and v_j. It takes $O(s)$ time to compute marginal and maximum-a-posteriori (MAP) estimates using an OR tree where s is the number of OR nodes in the tree.

Tree Bayesian Networks

A tree Bayesian network is a Bayesian network (Pearl 1988) in which each variable has at most one parent. The treewidth of these networks equals one and therefore both marginal and maximum-a-posteriori (MAP) inference queries can be answered in $O(n\delta_{max}^2)$ time and space where δ_{max} is the maximum domain size. Unlike general Bayesian networks in which learning the structure and parameters of the network given data is NP-hard in general, in tree Bayesian networks both structure and parameter learning tasks can be solved in polynomial time using the classic Chow-Liu algorithm (Chow and Liu 1968). The time complexity of this algorithm is $O(n^2 e)$ where n is the number of variables and e is the number of training examples.

A tree Bayesian network represents the following distribution:

$$T(x) = \prod_{X_i \in X} P(x_{\{X_i\}} | x_{\{pa(X_i)\}}) \tag{2}$$

where x is an assignment of values to all variables in X, $x_{\{X_i\}}$ denotes the projection of x on the variables in the set $\{X_i\}$ and $pa(X_i)$ denotes the parent of X_i in T.

Cutset Networks

Cutset networks (`CNets`) enhance the power of tree Bayesian networks by combining them with OR trees. Formally, a `CNet` denoted by c, is a pair $\langle o, \mathcal{T} \rangle$ where o is a rooted OR search tree having L leaves and $\mathcal{T} = \{T_l\}_{l=1}^{L}$ is a collection of tree Bayesian networks such that T_l is associated with the l-th leaf of o. It represents the following probability distribution:

$$c(x) = \left(\prod_{(v_i,v_j) \in path_o(x)} \rho(v_i, v_j) \right) \left(T_{l(x)}(x_{V(T_{l(x)})}) \right) \tag{3}$$

where $l(x)$ is the index of the leaf node corresponding to the assignment x in o and $V(T_l)$ is the subset of variables over which T_l is defined. Both marginal and MAP inference over `CNets` can be answered in $O(n\delta_{max}^2 l + s)$ time where l is the number of leaf nodes in the OR tree and other quantities are defined as before. `CNets` also admit a polynomial-time learning algorithm when the number of nodes s in the OR tree is bounded by a constant. A straight-forward approach is to generate all possible `CNets` having s OR nodes (which is polynomial in n) and select the one that has the highest log-likelihood score on the training data.

Although learning is tractable in `CNets`, it has high polynomial complexity. To remedy this problem, (Rahman, Kothalkar, and Gogate 2014) proposed a heuristic, recursive algorithm (having low polynomial complexity) to learn them from data. In this method, the OR tree is built in a top-down manner. At each recursive step, a variable that maximally reduces the expected entropy is selected and conditioned on. Then the algorithm recurses on each part of the two-way partition of the data (assuming binary variables) created by conditioning on the variable until some termination condition is satisfied (e.g., the number of examples or the entropy of the data at the node is small). At this point, the algorithm uses the Chow-Liu algorithm to learn a tree Bayesian network over the remaining variables. To avoid overfitting, Rahman et al. proposed to *post-prune* the network, by replacing an internal node of the OR tree by a tree Bayesian network in a bottom up fashion until the likelihood on the held-out set stops improving. Recently, (Di Mauro, Vergari, and Esposito 2015) proposed a Bayesian learning approach to prevent overfitting, in lieu of the costly post-pruning step.

In this paper, instead of using costly methods such as post-pruning and Bayesian learning to avoid overfitting, we propose to limit the depth of the OR tree, yielding a weak learner. We then improve the accuracy of these weak learners using ensemble methods. We also use a different heuristic than Rahman et al. for choosing the next variable to condition on. We score each variable X_i using $Score(X_i) = \sum_{X_j \in X \setminus \{X_i\}} I(X_i, X_j)$ where $I(X_i, X_j)$ is the mutual information between X_i and X_j and choose a variable having the highest score. The rationale for this heuristic is that variables having a large score are likely to be involved in a large number of edges and thus likely to be a part of a cutset. In our experiments, we found that this heuristic is much superior to the one used by Rahman et al.

Ensembles of CNets

We define an ensemble of cutset networks (`ECNets`), denoted by f_M, as a collection of pairs $\{\langle \alpha_m, c_m \rangle\}_{m=1}^{M}$ where c_m is a `CNet` and α_m is its associated weight or coefficient such that $\alpha_m \geq 0$ and $\sum_{m=1}^{M} \alpha_m = 1$. f_M represents the following probability distribution.

$$f_M(x) = \sum_{m=1}^{M} \alpha_m c_m(x)$$

where x is an assignment of values to all variables in X and $c_m(x)$ is the probability of x w.r.t. c_m. We assume that each cutset network c_m in the ensemble is a member of some class $\mathcal{Q}^d$ of cutset networks, such that the depth of the OR tree of all networks in the class $\mathcal{Q}^d$ is bounded by d. Frequently, we will refer to the cutset networks as the *base models* of f_M.

Given training data $\mathcal{D} = \{x^{(1)}, ..., x^{(N)}\}$, we can learn f_M by solving the following optimization problem:

$$f_M^* = \underset{f_M}{\operatorname{argmax}}\ \mathcal{L}(f_M; \mathcal{D}) \tag{4}$$

where $\mathcal{L}(f_M; \mathcal{D}) = \sum_{i=1}^{N} \log(f_M(x^{(i)}))$ is the log-likelihood of $\mathcal{D}$. Since α_m's cannot be directly estimated from data (they are not observed), the optimization problem is multi-modal. Therefore, we seek approximate heuristic approaches to solve it. One such approach is to fix M and then use the EM algorithm (Dempster, Laird, and Rubin 1977) to solve the optimization problem. This approach yields the MCNet algorithm described in (Rahman, Kothalkar, and Gogate 2014).

In this paper, we propose to solve the optimization problem using sequential boosting-based approaches as well as parallel bootstrapping (bagging) approaches. Intuitively, our new algorithms are likely to yield more accurate models than the conventional EM algorithm which updates all base models and mixture weights *simultaneously* because they will have better convergence properties (Neal and Hinton 1998), smaller computational complexity (since networks will be added sequentially or via bootstrapping), and superior ability to escape local maxima than the latter. As we will describe in the section on experiments, our experimental results clearly validate this intuition.

Boosting Cutset Networks

In this subsection, we present three approaches for boosting `CNets`. The first approach is based on the boosting density estimation approach of (Rosset and Segal 2002) (henceforth, called the `BDE` method); the second is based on a kernel-based generalization of the `BDE` method; and the third approach uses the sequential (or incremental) EM algorithm. We describe the three approaches in order next.

The `BDE` method sequentially grows the ensemble, adding a weak base learner at each stage. Formally, at each boosting stage m it seeks to find a weak learner c_m belonging to a class $\mathcal{Q}^d$ and a coefficient η_m to add to the current model f_{m-1} such that the log-likelihood of the new model $f_m = (1-\eta_m)f_{m-1} + \eta_m c_m$, denoted by $\mathcal{L}(f_m; \mathcal{D})$, is maximized. Since optimizing the log-likelihood is hard (there is no closed form solution), the log-likelihood is approximated using the following expression (derived using Taylor series).

$$\mathcal{L}(f_m; \mathcal{D}) \approx \mathcal{L}(f_{m-1}; \mathcal{D}) + \frac{\eta_m}{1-\eta_m} \sum_i \frac{c_m(x^{(i)})}{f_{m-1}(x^{(i)})} \tag{5}$$

Assuming that η_m is a (small) constant, $\mathcal{L}(f_m; D)$ can be maximized by maximizing $\sum_i \frac{c_m(x^{(i)})}{f_{m-1}(x^{(i)})}$. To maximize the latter, at each boosting step m, we search for a `CNet` $c_m \in \mathcal{Q}^d$ that maximizes the weighted log-likelihood of data, in which each example $x^{(i)}$ is weighted by $\frac{1}{f_{m-1}(x^{(i)})}$. In other words, examples which have lower probability according to the previous model receive higher weight and vice versa.

An algorithmic description of a scheme that uses the `BDE` method for boosting `CNets` is given in Algorithm 1. The algorithm begins by initializing the model f_0 to the uniform distribution. Then, at each iteration m, it updates the weight $\omega_m^{(i)}$ of each example $x^{(i)}$ to $\frac{1}{f_{m-1}(x^{(i)})}$ (step 4), learns a new `CNet` c_m that maximizes the weighted log-likelihood using the algorithm described in (Rahman, Kothalkar, and Gogate 2014; Di Mauro, Vergari, and Esposito 2015) (step 5), finds a weighting co-efficient η_m for the new model by performing a line search (step 6), and finally updates the model f_m with the new weighting co-efficient (step 7).

Algorithm 1: `CNet`-Boosting

Input : Dataset $\mathcal{D}$, An integer M, maximum-depth d
Output: f_M

1 **begin**
2 $\quad f_0 =$ uniform distribution
3 $\quad$ **for** $m = 1$ *to* M **do**
4 $\quad\quad \omega_m^{(i)} = \frac{1}{f_{m-1}(x^{(i)})}\ \forall x^{(i)} \in \mathcal{D}$
5 $\quad\quad c_m = \underset{c}{\operatorname{argmax}} \sum_i \omega_m^{(i)} c(x^{(i)})$ from $\mathcal{Q}^d$
6 $\quad\quad \eta_m = \underset{\eta}{\operatorname{argmax}} \sum_i \log((1-\eta) f_{m-1}(x^{(i)}) + \eta c_m(x^{(i)}))$
7 $\quad\quad f_m = (1-\eta_m) f_{m-1} + \eta_m c_m$
8 $\quad$ **return** f_M

Next, we derive an AdaBoost (Freund and Schapire 1997) style algorithm for learning `ECNets` by generalizing the weight update rule in step 4 of Algorithm 1. To derive this rule, we rewrite the weight as

$$\omega_{m+1}^{(i)} = \frac{1}{f_m(x^{(i)})} = \frac{1}{(1-\eta_m) f_{m-1}(x^{(i)}) + \eta_m c_m(x^{(i)})}$$

Dividing both the numerator and the denominator by $f_{m-1}(x^{(i)})$ and factoring out η_m we get:

$$\omega_{m+1}^{(i)} = \frac{\omega_m^{(i)}}{1 + \eta_m(\frac{c_m(x^{(i)})}{f_{m-1}(x^{(i)})} - 1)} \tag{6}$$

From (6), we can see that the ratio $\frac{c_m(x^{(i)})}{f_{m-1}(x^{(i)})}$ determines the relationship between $\omega_m^{(i)}$ and $\omega_{m+1}^{(i)}$:

- **Case 1:** $\frac{c_m(x^{(i)})}{f_{m-1}(x^{(i)})} \geq 1 \Rightarrow \omega_{m+1}^{(i)} \leq \omega_m^{(i)}$: the weight of an example having higher probability in c_m than f_{m-1} is decreased.
- **Case 2:** $\frac{c_m(x^{(i)})}{f_{m-1}(x^{(i)})} < 1 \Rightarrow \omega_{m+1}^{(i)} > \omega_m^{(i)}$: the weight of an example having higher probability in f_{m-1} than c_m is increased.

We can express the relationship between $\omega_{m+1}^{(i)}$ and $\omega_m^{(i)}$, in a more general form using the following expression:

$$\omega_{m+1}^{(i)} = \frac{\omega_m^{(i)}}{1 + \beta_m K(c_m(x^{(i)}), f_{m-1}(x^{(i)}), \epsilon)} \tag{7}$$

where K is a kernel-function for *smoothing* the (gradient) updates, $\beta_m \in (0, 1)$ is the step size and ϵ is a tolerance

Table 1: Test set log-likelihood scores of boosting algorithms. Winning scores are bolded and underlines highlight GBDE over BDE. FD:= Fixed Depth and VD:=Variable Depth.

Datasets	CNet						MCNet					
	FD			VD			FD			VD		
	BDE	GBDE	SEQEM	BDE	GBDE	SEQEM	BDE	GBDE	SEQEM	BDE	GBDE	SEQEM
NLTCS	-6.03	-6.01	-6.01	-6.03	-6.02	-6.01	-6.00	-6.00	-6.00	-6.00	-6.02	-6.02
MSNBC	-6.26	-6.21	-6.15	-6.25	-6.23	-6.15	-6.21	-6.17	-6.25	-6.22	-6.23	-6.24
KDDCup2K	-2.19	-2.15	-2.15	-2.18	-2.17	-2.15	-2.14	-2.13	-2.13	-2.15	-2.14	-2.14
Plants	-13.03	-12.76	-12.72	-13.42	-13.11	-12.65	-12.44	-12.42	**-12.32**	-12.58	-12.69	-12.64
Audio	-40.59	-40.37	-39.94	-40.95	-40.67	-39.84	-39.80	-39.86	**-39.67**	-39.89	-40.08	-40.11
Jester	-53.25	-52.98	-52.87	-53.55	-53.45	-52.82	-52.57	-52.69	**-52.44**	-52.82	-52.94	-52.78
Netflix	-56.76	-56.73	-56.47	-57.62	-57.61	-56.44	-56.29	-56.39	**-56.13**	-56.47	-56.65	-56.65
Accidents	-30.09	-30.09	-29.45	-30.52	-30.42	-29.45	-29.41	-29.33	**-29.27**	-29.50	-29.67	-30.26
Retail	-10.93	-10.89	-10.81	-10.96	-10.88	-10.82	-10.83	-10.85	**-10.79**	-10.84	-10.83	-10.83
Pumsb-star	-24.08	-24.09	-23.45	-24.37	-24.25	-23.47	-23.43	-23.48	**-23.37**	-23.53	-23.80	-26.03
DNA	-86.24	-86.30	-86.12	-86.18	-85.82	-85.67	-85.00	-84.93	**-82.67**	-84.03	-84.68	-85.12
Kosarek	-11.03	-10.77	-10.62	-10.83	-10.78	-10.60	-10.57	-10.57	**-10.54**	-10.58	-10.59	-10.56
MSWeb	-10.04	-9.86	-9.73	-10.03	-9.89	-9.75	-9.85	-9.80	**-9.72**	-9.89	-9.83	-9.79
Book	-36.08	-35.91	-34.46	-36.02	-35.74	-34.48	-33.93	-33.85	-33.95	-33.99	-33.85	**-33.78**
EachMovie	-55.18	-53.46	-52.00	-54.61	-53.87	-51.53	-51.63	-51.75	**-51.14**	-51.75	-51.48	-51.92
WebKB	-156.46	-155.08	-152.86	-154.95	-155.10	-152.53	-151.64	-151.45	-151.60	-151.51	**-150.71**	-150.84
Reuters-52	-85.82	-84.90	-84.03	-85.16	-84.83	-83.69	-83.65	-83.61	**-82.29**	-84.09	-83.73	-82.65
20Newsgrp.	-156.16	-155.61	-153.57	-155.85	-155.77	-153.12	-153.52	-152.90	**-151.75**	-152.59	-152.89	-153.17
BBC	-247.01	-247.44	-251.96	-250.92	-249.53	-251.81	-244.61	**-237.87**	-237.94	-242.43	-238.59	-248.32
Ad	-15.74	-15.90	-14.37	-15.75	-16.09	**-14.36**	-14.48	-14.97	-14.58	-14.65	-14.65	-14.50
Average LL	-55.15	-54.87	-54.49	-55.31	-55.11	-54.37	-53.90	-53.67	**-53.23**	-53.78	-53.60	-54.22

Table 2: Runtime comparison (in seconds). †indicates that the algorithm did not terminate in 48 hrs.

Datasets	Boosting						Bagging		MCNet	ID-SPN	ACMN
	BDE		GBDE		SEQEM						
	CNet	MCNet	CNet	MCNet	CNet	MCNet	CNet	MCNet			
NLTCS	27.60	512.80	82.60	289.51	181.17	193.04	2.90	116.42	36.57	307.00	242.40
MSNBC	366.28	4224.64	992.60	5787.68	14962.22	106.38	671.81	1689.74	2177.73	90354.00	579.90
KDDCup2K	600.73	15454.69	6966.55	13703.42	5785.30	1731.47	1344.17	2745.99	1988.49	38223.00	645.50
Plants	973.17	2494.60	2668.30	1829.74	2773.67	2340.01	54.80	824.44	135.00	10590.00	119.40
Audio	1384.60	2744.51	1529.60	2045.94	2889.43	2865.37	89.39	743.71	187.78	14231.00	1663.90
Jester	1539.95	1695.21	3354.30	1063.90	2152.58	6437.08	52.52	939.97	101.15	†	3665.80
Netflix	5610.40	3328.44	3982.86	2243.79	4569.44	5321.28	106.41	679.76	224.38	†	1837.40
Accidents	2049.23	3484.30	1888.71	2334.04	3385.86	7228.44	100.93	496.82	195.49	†	793.40
Retail	213.41	4416.32	285.11	1132.00	7641.05	4065.44	247.30	813.84	104.67	2116.00	12.50
Pumsb-star	1055.03	4173.11	1015.13	2908.71	4419.91	3864.32	141.54	386.63	233.79	18219.00	374.00
DNA	89.69	446.61	128.02	331.83	421.34	153.18	30.49	119.26	57.69	150850.00	39.90
Kosarek	967.93	9474.57	2564.82	5693.66	13352.09	10891.22	1264.40	624.03	141.16	†	585.40
MSWeb	1309.71	13207.53	3646.54	12260.44	17431.33	16136.63	3515.94	1987.20	642.80	†	286.30
Book	411.21	6650.12	4451.52	1484.70	8756.64	5451.08	2349.20	1673.81	154.42	125480.00	3035.00
EachMovie	3053.31	4646.34	2973.99	3560.84	4037.28	2073.33	648.18	1318.09	204.81	78982.00	9881.10
WebKB	493.07	4754.49	1185.25	1643.65	2537.48	2038.20	943.17	1346.62	160.40	†	7098.30
Reuters-52	5309.41	14995.21	5770.73	4637.31	10680.87	5268.56	2109.27	1901.74	1525.20	†	2709.60
20Newsgrp.	12950.45	27310.00	7463.05	7774.85	25961.20	10045.16	3867.26	2899.56	1177.16	†	16255.30
BBC	2353.06	3774.40	1497.82	1823.51	1190.47	1697.80	810.38	619.17	70.21	4157.00	1862.20
Ad	2314.91	7834.92	263.50	3188.44	8800.22	3040.97	1744.25	720.76	155.38	†	6496.40
Average Time	2153.66	6781.14	2635.55	3786.90	7096.48	4547.45	1004.72	1132.38	**483.71**	†	2909.19

measure. For example, the following function can be used to smooth the updates:

$$K(c_m(x), f_{m-1}(x), \epsilon) = \begin{cases} +1 & \text{if } (\frac{c_m(x)}{f_{m-1}(x)} - 1) > \epsilon \\ -1 & \text{if } (1 - \frac{c_m(x)}{f_{m-1}(x)}) > \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

Using (7) (with K given by (8)) instead of the rule given in step 4 of Algorithm 1 yields *smooth* updates in the following sense. In the BDE method, all weights change at each iteration by a different amount. When a smooth K such as the one given in (8) is used, a weight $\omega_{m+1}^{(i)}$ is updated iff the relative difference between $c_m(x^{(i)})$ and $f_{m-1}(x^{(i)})$ for the corresponding example $x^{(i)}$ is larger than the tolerance measure ϵ. Moreover, all weights (that are updated) are updated by the same amount, similar to AdaBoost (which updates all misclassified examples).

Our third approach uses the EM algorithm (Dempster, Laird, and Rubin 1977) to solve the optimization problem at each iteration m. Note that in the approximation given in (5), η_m is assumed to be a (small) constant. In our EM-based

approach similar to (Ridgeway 2002), we remove this relaxation and jointly optimize η_m and c_m, keeping f_{m-1} fixed.[1] The algorithm operates as follows. At each iteration m, the algorithm randomly guesses η_m and c_m. It then alternates between the following expectation (E-Step) and maximization steps (M-Step) until convergence:

- E-Step : $\gamma_m^{(i)} = \frac{\eta_m c_m(x^{(i)})}{(1-\eta_m) f_{m-1}(x^{(i)}) + \eta_m c_m(x^{(i)})}$
- M-step : $\begin{cases} \eta_m = \frac{1}{N} \sum_{i=1}^{N} \gamma_m^{(i)} \\ c_m = \underset{c}{\operatorname{argmax}} \sum_{i=1}^{N} \gamma_m^{(i)} \log(c(x^{(i)})), c \in \mathcal{Q}^d \end{cases}$

Bagging Cutset Networks

Bootstrap aggregation (Bagging) is a method for combining several high variance models of the same kind trained on different subsets of the data (Breiman 2001). The subsets are called *bootstrap samples* and are generated by sampling the original data with replacement. In various empirical studies, Bagging has been shown to improve the accuracy of several supervised learning algorithms by reducing the variance of the learning algorithm when data is scarce. In this paper, we use the following straight-forward extension of the general bagging method to learn ECNets.

Each constituent CNet c_m in the ensemble is learned by generating *bootstrap samples* from the data, and then maximizing the log-likelihood of the generated samples. The coefficients α_m's can be learned in several ways. One approach is to attach the same weight $1/M$ to each c_m, yielding a simple average mixture. Another approach is to weigh each c_m by a value that is proportional to its likelihood. In this paper, we use the latter approach because it often performs better than simple averaging in practice. Note that the negative log-likelihood measures the error of the base models and thus our one-shot weighing technique assigns high weight to low error models and vice versa. Since each bootstrap uses roughly 63.2% of the unique examples of $\mathcal{D}$, this measure takes into account the out-of-bag sample error and therefore exempts us from using a validation set to estimate the coefficients.

To de-correlate the cutset networks in the bagged ensemble, we use the following *random forests* inspired approach (Breiman 2001). At each node of the OR tree, we pick r variables uniformly at random and then use the splitting heuristic to select the best variable from these r variables.

Experiments

We evaluated the performance of ECNets on 20 real world benchmark datasets (Lowd and Davis 2010; Gens and Domingos 2013; Rooshenas and Lowd 2014; Rahman, Kothalkar, and Gogate 2014; Van Haaren and Davis 2012; Davis and Domingos 2010) listed in Table 4. The number of variables range from 16 to 1556 and the number of training examples vary from 1.6K to 291K examples. All variables are binary. We ran all our experiments on a quad-core Intel i7 2.7 GHz machine with 16GB RAM and ran each algorithm for 48 hours or until termination, whichever was earlier.

Table 3: Test set log-likelihood scores of bagged CNets and MCNets. Bold values indicate the winning scores. FD:= Fixed Depth and VD:=Variable Depth.

	CNet		MCNet	
Dataset	FD	VD	FD	VD
NLTCS	-6.00	-6.02	-6.00	-6.00
MSNBC	-6.08	-6.12	-6.06	**-6.05**
KDDCup 2K	-2.14	-2.14	-2.13	-2.13
Plants	-12.32	-12.44	**-12.19**	-12.20
Audio	-40.09	-40.37	**-39.74**	-39.88
Jester	-52.88	-53.04	-52.62	**-52.59**
Netflix	-56.55	-56.99	**-56.22**	-56.43
Accidents	-29.88	-30.16	**-29.25**	-29.46
Retail	-10.84	-10.84	-10.79	**-10.78**
Pumsb-star	-23.98	-24.27	**-23.34**	-23.56
DNA	-83.55	-81.07	-83.03	**-80.66**
Kosarek	-10.75	-10.74	-10.70	**-10.55**
MSWeb	-9.77	-9.80	-9.74	**-9.70**
Book	-35.84	-35.55	-34.97	**-33.96**
EachMovie	-53.37	-53.00	-52.00	**-51.22**
WebKB	-158.12	-153.12	-156.14	**-150.10**
Reuters-52	-84.36	-83.71	-83.77	**-82.19**
20Newsgrp.	-156.60	-155.09	-156.34	**-153.00**
BBC	-239.88	-237.42	-241.46	**-236.82**
Ad	-15.28	-15.32	-15.09	**-14.67**
Average LL	-54.41	-53.86	-54.08	**-53.10**

Our chosen base models are CNets and small MCNets with the number of components fixed to either 2 or 3 in an ensemble(chosen by the validation set). We also ensured that the total number of component Nets learned does not exceed 40. This was done to make fair comparisons with other tractable model learners. We introduced two types of randomizations in learning the base models: randomizing the pool of cutset variables at each splitting node, and randomizing the maximum depth of the OR tree. The former is equivalent to learning an ensemble of randomized cutset networks as done in random forests – at each splitting node we randomly sample without return 50% of the variables and apply our heuristic to choose the best one. The latter is similar to stacking – combining models of different complexities. At each iteration we randomly picked an integer depth $0 \leq \ell \leq$ max depth to learn the next base model. Here max depth is the maximum depth that a cutset network can have in an ensemble. Therefore, both boosting and bagging are performed using fixed depth as well as variable depth base models leading to four categories of algorithms: learning with fixed depth CNets, learning with variable depth CNets, learning with fixed depth MCNets, and learning with variable depth MCNets. We used 1-laplace smoothing for the parameters.

[1] The difference between this EM-based method and the EM-based MCNet algorithm proposed in (Rahman, Kothalkar, and Gogate 2014) is that in the latter at each EM iteration, all cutset networks c_j and coefficients η_j for $j = 1$ to m are updated simultaneously while in the former only c_m and η_m are updated.

Table 4: Test set log-likelihood comparison. (Ties are not bolded).

Dataset	#Var	#Train	#Valid	#Test	ECNet		MCNet	ID-SPN	ACMN	MT	SPN	LTM
					Bag	Boost						
NLTCS	16	16181	2157	3236	-6.00	-6.00	-6.00	-6.02	-6.00	-6.01	-6.11	-6.49
MSNBC	17	291326	38843	58265	-6.05	-6.15	-6.04	-6.04	-6.04	-6.07	-6.11	-6.52
KDDCup2K	64	180092	19907	34955	-2.13	-2.13	**-2.12**	-2.13	-2.17	-2.13	-2.18	-2.18
Plants	69	17412	2321	3482	**-12.19**	-12.32	-12.74	-12.54	-12.80	-12.95	-12.98	-16.39
Audio	100	15000	2000	3000	-39.74	**-39.67**	-39.73	-39.79	-40.32	-40.08	-40.50	-41.90
Jester	100	9000	1000	4116	-52.59	**-52.44**	-52.57	-52.86	-53.31	-53.08	-53.48	-55.17
Netflix	100	15000	2000	3000	-56.22	**-56.13**	-56.32	-56.36	-57.22	-56.74	-57.33	-58.53
Accidents	111	12758	1700	2551	-29.25	-29.27	-29.96	**-26.98**	-27.11	-29.63	-30.04	-33.05
Retail	135	22041	2938	4408	**-10.78**	-10.79	-10.82	-10.85	-10.88	-10.83	-11.04	-10.92
Pumsb-star	163	12262	1635	2452	-23.34	-23.37	-24.18	**-22.40**	-23.55	-23.71	-24.78	-31.32
DNA	180	1600	400	1186	-80.66	-82.67	-85.82	-81.21	**-80.03**	-85.14	-82.52	-87.60
Kosarek	190	33375	4450	6675	-10.55	**-10.54**	-10.58	-10.60	-10.84	-10.62	-10.99	-10.87
MSWeb	294	29441	32750	5000	**-9.70**	-9.72	-9.79	-9.73	-9.77	-9.85	-10.25	-10.21
Book	500	8700	1159	1739	-33.96	**-33.78**	-33.96	-34.14	-36.56	-34.63	-35.89	-34.22
EachMovie	500	4524	1002	591	-51.22	**-51.14**	-51.39	-51.51	-55.80	-54.60	-52.49	†
WebKB	839	2803	558	838	**-150.10**	-150.71	-153.22	-151.84	-159.13	-156.86	-158.20	-156.84
Reuters-52	889	6532	1028	1540	**-82.19**	-82.29	-86.11	-83.35	-90.23	-85.90	-85.07	-91.23
20Newsgrp.	910	11293	3764	3764	-153.00	-151.75	**-151.29**	-151.47	-161.13	-154.24	-155.93	-156.77
BBC	1058	1670	225	330	**-236.82**	-237.87	-250.58	-248.93	-257.10	-261.84	-250.69	-255.76
Ad	1556	2461	327	491	-14.67	**-14.36**	-16.68	-19.00	-16.53	-16.02	-19.73	†
Average LL					**-53.06**	-53.16	-54.50	-53.89	-55.83	-55.55	-55.32	†

Boosting Performance

We compare the following three different boosting techniques using CNets and MCNets as base models: the BDE method, our proposed modification to the BDE method (see Eq. (8)) which we call the GBDE and the sequential EM-algorithm which we refer to as SEQEM in brief. For the GBDE, we tried the following values for ϵ: $\{0.1, 0.3, 0.5, 0.7, 0.9\}$ and β_m: $\{0.01, 0.05, 0.2, 0.4, 0.6, 0.8\}$. The best values were selected based on the accuracy attained on the validation set. In all three algorithms, we added a base model to the ensemble until the validation set likelihood decreased or the total number of boosting iterations reached 40. The maximum depth of CNets and MCNets was varied from 0 to 5 and the best depth was also chosen by the validation set. To learn simpler models than fully connected Chow-Liu trees at the leaves of the base models (and thus avoid overfitting), we removed all edges from the Chow-Liu trees whose mutual information was smaller than 0.005. For learning MCNets we generated bootstrap samples from the weighted datasets in both BDE and GBDE algorithms (Freund and Schapire 1996). The parameters of the mixture were then optimized via the EM algorithm using the original training set for 50 iterations or until convergence.

Table 1 reports the test set log-likelihood scores achieved by each algorithm in four different categories of algorithms. The last row reports the average score on all datasets for each algorithm and represents a quick shot statistic for comparing the performance of the various algorithms. SEQEM-boosting with fixed depth MCNets is the best performing algorithm scoring the highest average log-likelihood. GBDE yields better generalization performance than the BDE in all four categories. Table 2 compares the running time of these algorithms. Because of space restrictions, we are only reporting the running time for fixed depth boosted models. We see that SEQEM was the slowest while GBDE and BDE were the fastest, clearly demonstrating the time versus accuracy trade-off for the three boosting algorithms.

Bagging Performance

We varied the number of bags from 5 to 40 with increments of 5. The maximum depth of the OR trees was varied from 5 to 10. The number of bags and the depth was chosen based on the accuracy on the validation set. In bagging MCNets, we randomized the structure using bootstrap replicates and then learned the best parameters for that structure on the training set (Ammar et al. 2010). EM was run for 100 iterations or until convergence and no restarts were performed. The test set log-likelihood scores are given in Table 3. Bag of variable depth MCNets performs the best out of the four bagging schemes. Comparing the time for Bagging from Table 2 with that of Boosting, we see that bagging is the fastest algorithm among the ensemble learning techniques for cutset networks. Unlike boosting, larger depth trees and randomization significantly improves the accuracy of the model.

Comparison with State-of-the-art

We also compared the accuracy and learning efficiency of ECNets to five other well-cited state-of-the-art tractable model learners: learning sum-product network with direct and indirect variable interactions (ID-SPN), learning Markov networks using arithmetic circuits (ACMN) (Lowd and Rooshenas 2013), learning mixtures of trees(MT) (Meila and Jordan 2001), learning sum-product networks (SPN) and learning latent tree models (LTM). Table 4 reports the performance of these algorithms. These results were taken from (2014; 2014). For bagging and boosting

we are only reporting results for settings selected using the validation set for a fair comparison. ECNets algorithms are clearly the best performing algorithms outperforming the competition on 13 out of the 20 datasets with 1 tie. Fast bagging methods perform slightly better than boosting on high-dimensional datasets having few examples. This is possibly because of superior randomization in bagging which helps reduce the variance. ECNets are almost always better than MCNets. This shows that our new bootstrap and sequential optimization approaches are superior to the non-sequential EM algorithm used in MCNets.

Summary

In this paper we presented novel boosting and bagging algorithms for learning cutset networks from data. We performed a comprehensive empirical evaluation comparing our algorithms to state-of-the-art algorithms as well as to each other. Our results clearly show that our new additive models are quite powerful and superior to state-of-the-art algorithms. Future work includes: learning ensembles of tractable lifted CNets (Gogate and Domingos 2010); adding AND or product nodes to CNets; inducing OR graphs instead of OR trees; etc.

Acknowledgments. We gratefully acknowledge the support of the Defense Advanced Research Projects Agency (DARPA) Probabilistic Programming for Advanced Machine Learning Program under Air Force Research Laboratory (AFRL) prime contract number FA8750-14-C-0005.

References

Ammar, S.; Leray, P.; Schnitzler, F.; et al. 2010. Subquadratic Markov tree mixture learning based on randomizations of the Chow-Liu algorithm. In *Proceedings of the 5th European Workshop on Probabilistic Graphical Models*, 17–24.

Bach, F. R., and Jordan, M. I. 2001. Thin Junction Trees. In *Advances in Neural Information Processing Systems*, 569–576.

Breiman, L. 2001. Random forests. *Machine learning* 45(1):5–32.

Choi, M. J.; Tan, V. Y.; Anandkumar, A.; and Willsky, A. S. 2011. Learning latent tree graphical models. *The Journal of Machine Learning Research* 12:1771–1812.

Choi, A.; Van den Broeck, G.; and Darwiche, A. 2015. Tractable learning for structured probability spaces: A case study in learning preference distributions. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, 2861–2868.

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

Darwiche, A. 2003. A Differential Approach to Inference in Bayesian Networks. *Journal of the ACM* 50:280–305.

Davis, J., and Domingos, P. 2010. Bottom-up learning of markov network structure. In *Proceedings of the 27th International Conference on Machine Learning*, 271–278.

Dechter, R. 1990. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence* 41(3):273–312.

Dempster, A. P.; Laird, N. M.; and Rubin, D. B. 1977. Maximum Likelihood from Incomplete Data Via the EM Algorithm. *Journal of the Royal Statistical Society, Series B* 39:1–38.

Di Mauro, N.; Vergari, A.; and Esposito, F. 2015. Learning accurate cutset networks by exploiting decomposability. In *AI*IA 2015 Advances in Artificial Intelligence*, 221–232.

Freund, Y., and Schapire, R. E. 1996. Experiments with a new boosting algorithm. In *Proceedings of the 13th International Conference on Machine Learning*, 148–156.

Freund, Y., and Schapire, R. E. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences* 55(1):119–139.

Gens, R., and Domingos, P. 2013. Learning the structure of sum-product networks. In *Proceedings of The 30th International Conference on Machine Learning*, 873–880.

Gogate, V., and Domingos, P. 2010. Exploiting Logical Structure in Lifted Probabilistic Inference. In *AAAI 2010 Workshop on Statistical Relational Learning*.

Lowd, D., and Davis, J. 2010. Learning Markov network structure with decision trees. In *Proceedings of the 10th International Conference on Data Mining*, 334–343.

Lowd, D., and Domingos, P. 2008. Learning Arithmetic Circuits. In *Proceedings of the 24th International Conference on Uncertainty in Artificial Intelligence*, 383–392.

Lowd, D., and Rooshenas, A. 2013. Learning markov networks with arithmetic circuits. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, 406–414.

Meila, M., and Jordan, M. I. 2001. Learning with mixtures of trees. *The Journal of Machine Learning Research* 1:1–48.

Neal, R. M., and Hinton, G. E. 1998. A view of the EM algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*. Springer. 355–368.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Poon, H., and Domingos, P. 2011. Sum-Product Networks: A New Deep Architecture. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence*, 337–346.

Rahman, T.; Kothalkar, P.; and Gogate, V. 2014. Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of chow-liu trees. In *Proceesings of ECML and PKDD*, 630–645.

Ridgeway, G. 2002. Looking for lumps: Boosting and bagging for density estimation. *Computational Statistics & Data Analysis* 38(4):379–392.

Rooshenas, A., and Lowd, D. 2014. Learning sum-product networks with direct and indirect variable interactions. In *Proceedings of The 31st International Conference on Machine Learning*, 710–718.

Rosset, S., and Segal, E. 2002. Boosting density estimation. In *Advances in Neural Information Processing Systems*, 641–648.

Van Haaren, J., and Davis, J. 2012. Markov network structure learning: A randomized feature generation approach. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, 1148–1154.

Welling, M.; Zemel, R. S.; and Hinton, G. E. 2002. Self supervised boosting. In *Advances in Neural Information Processing Systems*, 665–672.

Zhou, Z.-H. 2012. *Ensemble Methods: Foundations and Algorithms*. Chapman & Hall/CRC, 1st edition.