Greedy Learning of Binary Latent Trees

Stefan Harmeling and Christopher K. I. Williams

Abstract—Inferring latent structures from observations helps to model and possibly also understand underlying data generating processes. A rich class of latent structures are the latent trees, i.e. tree-structured distributions involving latent variables where the visible variables are leaves. These are also called hierarchical latent class (HLC) models. Zhang (2004) proposed a search algorithm for learning such models in the spirit of Bayesian network structure learning. While such an approach can find good solutions it can be computationally expensive. As an alternative we investigate two greedy procedures: the BIN-G algorithm determines both the structure of the tree and the cardinality of the latent variables in a bottom-up fashion. The BIN-A algorithm first determines the tree structure using agglomerative hierarchical clustering, and then determines the cardinality of the latent variables as for BIN-G. We show that even with restricting ourselves to binary trees we obtain HLC models of comparable quality to Zhang's solutions (in terms of cross-validated log-likelihood), while being generally faster to compute. This claim is validated by a comprehensive comparison on several datasets. Furthermore, we demonstrate that our methods are able to estimate interpretable latent structures on real-world data with a large number of variables. By applying our method to a restricted version of the 20 newsgroups data these models turn out to be related to topic models, and on data from the PASCAL Visual Object Classes (VOC) 2007 challenge we show how such tree-structured models help us understand how objects co-occur in images. For reproducibility of all experiments in this paper, all code and datasets (or links to data) is available[1].

Index Terms—Unsupervised Learning, Latent Variable Model, Hierarchical Latent Class Model, Greedy Methods.

1 Introduction

It is widely recognized that a distribution $p(\mathbf{x})$ over a vector of variables $\mathbf{x} = (x_1, \ldots, x_D)$ can often usefully be modelled with the aid of some latent (or hidden) variables. Our goal is to learn tree- or forest-structured distributions involving latent variables where the visible variables $\mathbf{x}$ are leaves, as shown in Figures 1–4. Our focus is primarily on discrete visible variables.

A simple latent-variable model for discrete data is the *latent class model* (LCM; see e.g. Lazarsfeld & Henry, 1968). In this model there is one discrete latent variable that can take on K different states, and the visible variables are conditionally independent given the latent variable[2]. This model can readily be fitted to data using the EM algorithm. However, it has strong assumptions of conditional independence that in general will not be justified.

These strong assumptions can be relaxed by proposing a richer, tree-structured latent variable model as proposed for example in Zhang (2004). Following Zhang we call this a *hierarchical latent class* (HLC) model. The network structure is a rooted tree and the leaves of the tree are the visible variables. An attraction of a latent *tree* structure (compared to more complex DAGs) is that it allows linear time inference (Pearl, 1988). Furthermore, such a latent structure reflects a hierarchical grouping of the visible variables, making HLC models often interpretable and giving insights into the data generating processes. We emphasize the difference between the HLC model and the work of Chow and Liu (1968); the latter algorithm produces a tree-structured model defined solely on the visible variables, and does not induce latent variables.

To specify an HLC model there are two issues to be addressed: (i) the structure of the latent tree and (ii) the cardinality of the latent variables in the tree. Zhang (2004) defines the set of regular HLC models; essentially these are HLC models that are not overparameterized. (For example, for two binary visible variables we only need $K = 2$ latent states to model their joint distribution exactly; a model with $K > 2$ is overparameterized, as can be observed by parameter counting.) Zhang's algorithm conducts a search in the space of regular HLC models, starting the search from an LCM. The search involves moves which perform node introduction, node elimination, neighbour relocation, and changing the cardinality of a latent variable. Its runtime is dominated by the number of times the EM estimation procedure is called. In Zhang (2004) the EM algorithm is called $O(D^4)$ times (where D is the number of variables), and each run of EM takes $O(K^2DN)$ time (see end of next section for an explanation of the symbols) leading to an overall runtime in D of $O(D^5)$. Zhang and Kočka (2004) reduced this to $O(D^3)$. However, the EM algorithm is still called quadratically often in the number of variables.

We focus on HLC models that are binary trees or

- *S. Harmeling is with the Max Planck Institute for Biological Cybernetics, Tübingen, Germany.*
  E-mail: stefan.harmeling@tuebingen.mpg.de
- *C. K. I. Williams is with the Institute for Adaptive and Neural Computation, University of Edinburgh, Scotland.*
  E-mail: ckiw@inf.ed.ac.uk

1. http://people.kyb.tuebingen.mpg.de/harmeling/code/ltt-1.4.tar

2. This model has the same graphical structure as the naïve-Bayes classifier, but as it is trained in an unsupervised manner we refer to it as the LCM.

forests. In Section 2 we investigate two bottom-up procedures: the first, BIN-G, determines both the structure of the tree and the cardinality of the latent variables in a bottom-up fashion. The second algorithm BIN-A uses agglomerative hierarchical clustering to determine the structure of the tree, and then estimates the cardinality of the latent variables as for BIN-G. We show that both call the EM algorithm only $D-1$ times, i.e. they are linear in the number of variables, resulting in overall runtimes of $O(D^2)$.

In Section 3 we apply our methods to several real-world datasets and show experimentally that their performance (in terms of log-likelihood) is comparable to Zhang's method, while being computationally more efficient. Related work is discussed in section 4, and we give our conclusions in section 5.

2 Learning Binary Latent Tree Models

Our goal is to induce from N data samples an HLC model which is a good model for the underlying distribution from which the data was drawn. The quality of the induced model could be estimated e.g. using cross-validated predictive log-likelihood, or by a penalized maximum likelihood criterion such as the Bayesian Information Criterion (BIC) $J(\hat{\theta}) = l(\hat{\theta}) - P/2 \log N$, where $l(\hat{\theta})$ is the log-likelihood corresponding to optimal parameters $\hat{\theta}$ and structure, P is the number of free parameters of the model and N is the number of data points. The ability to generalize is an important feature as otherwise a sufficiently large model can simply "remember" the input data distribution. For example, consider an LCM which has as many latent states as there are unique data vectors: by making the conditional distribution for a given latent state be a delta function on the corresponding data vector the dataset can be memorized. Thus to encourage generalization the size of the model must be controlled.

We first describe how to learn the basic building block, the LCM, and then discuss two greedy algorithms for building a binary latent tree.

2.1 Learning Latent Class Models

We describe the simple case where the parent node has two children. Let x_1 and x_2 be two variables which not necessarily refer to visible variables, and a latent variable (denoted z). We focus on discrete random variables. The LCM for two variables x_1 and x_2 introduces a latent variable z, so that

$$p(x_1, x_2) = \sum_k p(z{=}k) p(x_1|z{=}k) p(x_2|z{=}k). \tag{1}$$

Its parameters $\pi_k = p(z{=}k)$, $\theta_{ik} = p(x_1{=}i|z{=}k)$ and $\eta_{jk} = p(x_2{=}j|z{=}k)$ are learned from data by the Expectation Maximization (EM) algorithm.

Deriving the EM updates for this model is a simple exercise if x_1 and x_2 are both observed, but it becomes more interesting when they are not observed. The relevant variables z, x_1, x_2 and the children of x_1 and x_2 form a tree-structured belief network (TSBN) with z at the root. The EM updates for this case are derived in Appendix A making use of Pearl's belief propagation algorithm for trees (Pearl, 1988). Similar updates have been used in Feng et al., (2002, Eq. (6)),

In practice we try different cardinalities for the latent variable z between 1 and $K_{\max}$, and select automatically using the BIC. We also use a number of random restarts to reduce the problem of local optima (in the reported experiments we did 10 restarts). We denote by LCM the method that automatically learns a latent class model.

2.2 Incremental Learning of Binary Tree Models

Our first approach to finding a binary-tree HLC model is by a greedy growing a tree-structured probabilistic model. We start with a simple model in which all visible variables are assumed independent. These variables form the "working set". We then choose from that set the pair with the highest mutual information (MI) and model them with an LCM with a new latent variable z. We then remove the selected pair from the working set of variables, add z to this set, and repeat either (i) until either the working set contains only a single variable (which would be the root the overall learned tree), or (ii) if we try to introduce a latent variable with a single state. In the latter case we stop early, and we fix the tree structure and the number of hidden states. Finally, we refine the conditional probability tables (CPTs, see line 19 of Alg. 1 and Appendix B), before we output the current forest of trees. The resulting algorithm, called BIN-G, is stated formally as Alg. 1.

The rationale for selecting the pair of variables with the highest MI is based on the following observation. Consider a distribution $p(\mathbf{x})$ over a vector of random variables $\mathbf{x}$ which is approximated by a distribution $q(\mathbf{x})$, where

$$q(\mathbf{x}) = p(x_i, x_j) \prod_{k \neq i,\, j} p(x_k) = \frac{p(x_i, x_j)}{p(x_i)p(x_j)} \prod_k p(x_k), \tag{2}$$

i.e. $q(\mathbf{x})$ models the joint distribution of x_i and x_j, but only the marginal distributions of the other variables. Then

$$\mathrm{KL}(p||q) = \sum_{\mathbf{x}} p(\mathbf{x}) \log p(\mathbf{x}) - \sum_{\mathbf{x}} p(\mathbf{x}) \log q(\mathbf{x}) \tag{3}$$

$$= -\,\mathrm{I}(x_i, x_j) + \sum_{\mathbf{x}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{\prod_k p(x_k)}, \tag{4}$$

where $\mathrm{I}(x_i, x_j)$ is the MI of x_i and x_j under the distribution $p(x_i, x_j)$. Thus in order to minimize the Kullback-Leibler divergence between $p(\mathbf{x})$ and $q(\mathbf{x})$ we should select the pair that has the highest MI.

Note that each induced tree will have a root. However, as is well-known in phylogenetics (see e.g. Felsenstein, 2004) the root can be "walked" around the tree without changing the joint distribution; there is an equivalence

Algorithm 1 BIN-G($\mathbf{x}$)

```
1:  input: a working set $V$ of variables $x_1, \ldots, x_D$
2:  $G \leftarrow$ the graph with vertices $V$ and no edges
3:  calculate pair-wise MI for all observed variables
4:  loop
5:      $W \leftarrow$ pair from $V$ with highest mutual information
6:      $V \leftarrow V \backslash W$ /* remove that pair from $V$ */
7:      $z \leftarrow$ LCM($W$) /* find latent class model by EM */
8:      if $z$ has single state then
9:          break /* outer loop */
10:     end if
11:     add vertex $z$ to graph $G$
12:     add edges from $z$ to children in $W$ to graph $G$
13:     if $V$ is empty then
14:         break /* outer loop */
15:     end if
16:     add latent variable $z$ to working set $V$
17:     calculate pair-wise MI for the new vertex $z$ and all
        variables of the working set
18: end loop
19: recursively refine the conditional probability tables
    using EM on structure $G$ (see Appendix B for details)
20: output: the graph $G$ (being a forest)
```

Algorithm 2 BIN-A($\mathbf{x}$)

```
1:  input: a working set $V$ of variables $x_1, \ldots, x_D$
2:  $G \leftarrow$ the graph with vertices $V$ and no edges
3:  calculate pair-wise MI for all observed variables
4:  loop
5:      $W \leftarrow$ pair from $V$ with highest mutual information
6:      $V \leftarrow V \backslash W$ /* remove that pair from $V$ */
7:      add vertex $z$ to graph $G$
8:      add edges from $z$ to children in $W$ to graph $G$
9:      if $V$ is empty then
10:         break /* outer loop */
11:     end if
12:     add latent variable $z$ to working set $V$
13:     approximate pair-wise MI for the new vertex $z$ and
        all variables of the working set by single, complete,
        or average linkage
14: end loop
15: recursively estimate the cardinality and a latent class
    model at each latent node by EM (i.e. calling LCM)
    beginning at the leaves
16: recursively refine the conditional probability tables
    using EM on structure $G$ (see Appendix B for details)
17: output: the graph $G$ (being a forest)
```

class of directed trees which corresponds to one undirected tree (see also discussion in Zhang, 2004, §3.2). This is not a problem for us as we only care about the distribution over the visibles induced by the equivalent undirected tree.

2.3 Learning Trees via Agglomerative Hierarchical Clustering

The BIN-G algorithm determines the tree structure and cardinality of the latent variables greedily as it proceeds by estimating a LCM for each introduced latent variable. Thus we are able to compute the MI between inferred latent nodes and other existing nodes (either latent or observed), as discussed above and in the appendix. An interesting question raised by one of the reviewers asked how such an inferred tree-structured model compares with a model based on a tree-structure determined via an agglomerative hierarchical clustering procedure (AHC, see e.g. Duda & Hart, 1973) running on the variables (*not* the datapoints). Of course such a tree structure also requires inference of LCMs locally at each node to constitute a full probabilistic model. However, instead of inferring the LCMs simultaneously with the tree structure (as BIN-G does), the LCMs can also be estimated *after* determining the tree structure.

We state such an algorithm based on AHC formally as Alg. 2 and called it BIN-A. The linkage options of AHC (mentioned in line 13 of Alg. 2) are explained at the beginning of Section 3. First of all we note that both proposed algorithms are similar. The clustering procedure of AHC for BIN-A is analogous to estimating the tree-structure in BIN-G. The differences are:

- Instead of estimating an LCM for each introduced latent node and calculating the MI in BIN-G (lines 7–10 and 17 in Alg. 1), BIN-A omits the immediate LCM estimation and approximates the MI by single, complete, or average linkage (line 13 in Alg. 2).
- In line 15 Alg. 2 estimates the LCMs bottom-up once the tree-structure is fixed, before applying EM on the whole model for refinement (line 16).

The BIN-A algorithm is related to some previous proposals for learning a tree structure by AHC, as discussed in Section 4.

2.4 Runtime Complexity of BIN-G and BIN-A

We will use the following constants for the runtime analysis:

N	number of data points
D	number of observed variables
K	maximal cardinality (actual)
$K_{\max}$	maximal cardinality (considered)
I	number of EM-iterations
S	number of EM-restarts

The runtime complexity of BIN-G is $O(D^2NK^2 + DSINK^2K_{\max})$. Its loop (see Alg. 1) is executed at most $D-1$ times, as the cardinality of the working set of variables V reduces by one on each iteration. The $O(D^2NK^2)$ term arises from the calculation of pairwise MIs at line 3, and the fact that the mutual information calculation at line 18 is $O(DNK^2)$ and that it is executed $O(D)$ times. The $O(DSINK^2K_{\max})$ term arises from the call to LCM in line 8, and the fact that this will be called at most $D-1$ times. Each iteration of EM in LCM takes

$O(NK^2)$, and there are up to I iterations and S restarts. Thus we conclude that the runtime complexity of BIN-G is quadratic in the number of variables D.

How does the second approach BIN-A compare to BIN-G? As already the structural similarity of the algorithms suggests, we will see that both have the same runtime complexity: similar reasoning as above explains that the loop of BIN-A (see Alg. 2) that determines the tree structure via AHC is executed at most $D - 1$ times. However, each iteration is much faster than for BIN-G since no latent class model is learned in that step. Thus the AHC step of BIN-A is linear in the number of variables. However, the subsequent recursive estimation procedure (line 15 in Alg. 2) estimates an LCM for each latent variable and thus calls LCM also linearly often in the number of variables D. Thus both procedures have similar times, as confirmed in the experimental section below.

2.5 Learning Latent Trees for Gaussian Variables

Above we have described a model for discrete visible variables. The analogue of the LCM for Gaussian-distributed continuous variables is the factor analysis (FA) model. Here the number of latent factors plays an analogous role to the cardinality of the latent variable in the discrete case; model selection for FA could be carried out using BIC. Going beyond factor analysis we come to a tree-structured latent-variable model, which is in fact a description of a (recursive) structural equation model (SEM, see e.g. Bollen, 1989). The greedy method above could equally be applied to learn SEMs.

3 Experiments

Restricting ourselves to binary trees results into large speed-ups in runtime which allow our methods to infer latent structures of large real-world datasets. In Section 3.1 we analyse text data from newsgroups, and search in Section 3.2 for latent structure in co-occurrence data derived from the PASCAL VOC 2007 challenge. We thoroughly evaluate our approach in Sections 3.3 and 3.4 on further datasets. The algorithms we consider are abbreviated in this section as follows: algorithm IND generates baseline results using the model in which all variables are independent of each other. Algorithm ZHANG is the compiled Java code provided by N. L. Zhang implementing the method described in Zhang and Kočka (2004). Algorithm LCM estimates a non-hierarchical Latent Class Model inferring a single latent variable. We also compare against algorithm CL from Chow and Liu (1968), however we note that it does not infer a latent structure. Finally, algorithms BIN-G and BIN-A denote our proposed methods detailed above. Note that BIN-A employs *average* linkage to approximate the MI between latent and other variables, which means that the MI between two variables z_1 and z_2 (either latent or observed) is approximated by the *average* MI between any leaf below or equal to z_1 and any leaf below or equal to z_2. Results using *complete* linkage (maximum distance, i.e. minimum MI) or *single* linkage (minimum distance, i.e. maximum MI) were omitted. These AHC variants produced similar results (in terms of the applied performance measures) and generated qualitively slightly inferior forest structures on the newsgroup (Section 3.1) and visual co-occurrence datasets (Section 3.2).

Note that for reproducibility the code for all experiments is provided as supplementary material (see footnote on the first page of this paper). This includes code and data that generates the toy examples and interfaces to the datasets freely available on the internet as indicated.

3.1 Topics of 20 Newsgroups

To demonstrate the ability of BIN-G and BIN-A to deal with a large number of variables we applied it to a binarized word-document matrix derived by Sam Roweis[3] from the UCI 20 newsgroups dataset. The dataset has been restricted to 100 words, such there are 100 binary variables and 16,242 data points. Our methods terminated after about an hour (BIN-G) and a quarter hour (BIN-A), while Zhang's method did not finish after several days, even though ZHANG's JAVA implementation is based on the Colt framework[4] for high performance scientific computing, while our code is Matlab. For completeness we also run LCM on this data and obtained the best log-likelihood. However, LCM is slower by factor of ten and it estimates only a single latent variable so it provides no interpretable graph structure.

Figure 1 shows a subtree of the whole latent forest model (shown in Figure 2) which BIN-G has estimated. The trees obtained by BIN-A are shown in Figure 3. Note that BIN-G and BIN-A reached similar log-likelihoods with BIN-G being slightly better (see Table 1). Each leaf corresponds to the word with which it is labelled, and each internal node displays an order number. The lowest order number for this dataset is 101 which corresponds to one plus the number of observed variables. The smaller the order number the earlier its children were merged during the tree building process.

Examining the labels of the tree inferred by algorithm BIN-G reveals that the inferred model can be interpreted in a meaningful way (similar to the work of Blei et al. (2003)): For instance the leaves of the subtree in Figure 1 capture the words from the topic "medicine", such as:

> **medicine** (subtree at node 192): doctor, medicine, disease, patients, cancer, studies, aids, health, insurance.

Similarly, other subtrees (see Figure 2) of the whole latent tree model collect words from other topics:

> **sports** (subtree at node 175): puck, hit, won, win, fans, league, nhl, games, baseball, hockey, players, season, team

3. http://www.cs.toronto.edu/~roweis/data.html
4. http://acs.lbl.gov/~hoschek/colt/

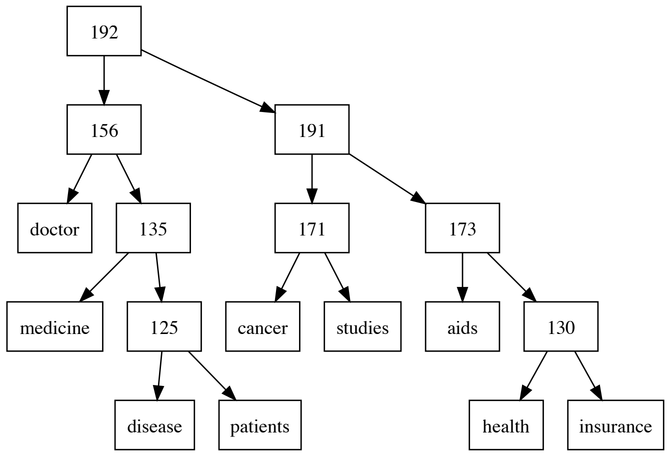


Fig. 1. Topics of 20 newsgroups: subtree collecting words of topic "medicine".

politics/religion (subtree at node 170): children, human, president, gun, state, law, government, rights, israel, jews, war, world, religion, christian, bible, god, jesus, evidence, fact

computer (subtree at node 187): email, phone, help, problem, technology, computer, science, data, system, mac, scsi, disk, drive, memory, graphics, card, video, pc, software, driver, ftp, version, program, files, dos, windows, format, image, display, server

spaceflight (subtree at node 162): mars, satellite, lunar, moon, launch, shuttle, nasa, space, earth, orbit, mission, solar

others (subtree at nodes 140): research, university; (subtree at node 188): water, vitamin, food, msg, (subtree at node 178): power, question

car (subtree at node 183): oil, bmw, honda, dealer, car, engine

We note, that the forest of BIN-G is not perfect. For comparison Figure 3 shows the forest inferred by BIN-A which has almost the same subtrees, but did split them, which might be preferable. However, the overall inferred latent structures of both methods do reflect the mixture of topics in the selected newsgroups. Furthermore, there is also interesting structure inside a single tree itself: e.g. in subtree "politics" the word "law" is closer to "government" than to "world" (see both Figures. 2 and 3).

3.2 PASCAL VOC 2007 data

A vision-related dataset to which we applied BIN-G and BIN-A was derived from the PASCAL VOC 2007 challenge on object recognition and localization in images[5] For each image in the dataset, each instance of the 20 considered object classes was labelled with a bounding box. The dataset consists of $D = 20$ variables that encode the location of each object's bounding box by a number from 0 to 9. Horizontally, there are three possibilities for the bounding box: either (i) the left edge of a small bounding box is on the far left or (ii) the right edge of a small bounding box is far right or (iii) the bounding is large horizontally and extends from left to right over the image. Similarly we get three possibilities vertically, which amounts to 9 options for an existing bounding box. Option 0 denotes that the corresponding object class is absent. If there is more than one object of a particular class in an image, multiple data points are generated. The challenge of this dataset is that each of the 20 variables has 10 states. Our methods BIN-G and BIN-A terminated after 0.5 and 1.5 hours respectively (see Table 1). We also tried to apply ZHANG. However, it did not finish after several days. Our methods, which estimate reasonable latent structures on this dataset (see Figure 4), are not able to infer a model that beats the slower LCM algorithm and the faster procedure of Chow and Liu (1968) in terms of log-likelihood (see Table 1). However, we note that a latent tree model is much more readily interpretable than the Chow-Liu tree or the flat LCM model.

Figure 4 shows the latent forest inferred by BIN-G. Visual results for BIN-A are omitted since they are similar. Each node either contains the name of a variable, such as "aeroplane", or the node number with the number of states in round brackets. Nodes with smaller numbers have been introduced earlier than nodes with larger numbers.

The learned trees arranges the objects (which correspond to variables) in a meaningful way: the biggest tree groups road scenes (subtree at node 26: person, car, bus, motorbike) and indoor scenes (subtree at node 28: sofa, tvmonitor, pottedplant, bottle, chair, diningtable). The singleton trees (e.g. the pets: cat, dog) make sense since people often take pictures of their pets without other objects. Similarly we wouldn't expect pictures of "aeroplane" to co-occur with other objects. On the other hand "cow" and "horse" can co-occur in images, which is reflected by the fact that they are grouped in a tree (subtree at node 30).

3.3 Comparative Study on 10 Datasets

To survey how well our fast greedy methods learn latent trees, we apply our algorithms to several toy and real-world datasets. As a performance measure we choose 10-fold cross-validated predictive log-likelihood (CVPLL). That is, we divide each dataset into 10 equal-sized folds, train a model on 9 of the folds and compute the predictive log-likelihood on the remaining fold. Averaging these results over the 10 folds gives the results in Tables 2 and 3 which uses the algorithm abbreviations introduced at the beginning of this section.

All not publicly available datasets are included in the supplementary in form of generating code or files. For

5. See http://www.pascal-network.org/challenges/VOC/voc2007 for example images, dataset statistics, etc.

TABLE 1
Log-likelihoods and running times on 20 newsgroups, and PASCAL VOC data.

			LOG-LIKELIHOOD						TIME IN SECONDS					
	D	N	IND	CL	LCM	BIN-G	BIN-A	ZHANG	IND	CL	LCM	BIN-G	BIN-A	ZHANG
NEWS20W100	100	16242	-255,614	-238,713	**-223,046**	*-231,764*	-232,166	—	0.1	9	42,298	*4,008*	**884**	—
VOC	20	20961	-209,725	*-184,577*	**-168,797**	-188,532	-188,345	—	0.06	6	11,414	**2,218**	*4,750*	—

TABLE 2
Ten-fold cross-validated log-likelihood for various datasets (rows) and algorithms (columns). The winner in each row is shown in boldface, the runner-up in italics. Column D shows the number of variables, N the number of data points.

	D	N	IND	CL	LCM	BIN-G	BIN-A	ZHANG
BINARY-FOREST	5	900	-832.84± 1.72	-556.57± 1.78	-556.60± 2.24	**-555.16**± 1.46	**-555.16**± 1.46	*-555.43*± 1.27
THREE-LEVEL-BINARY	4	900	-833.32± 2.44	-488.56± 2.50	-487.42± 2.45	**-486.36**± 1.61	**-486.36**± 1.61	*-486.42*± 1.52
THREE-COINS	3	900	-416.65± 1.20	-278.17± 0.88	*-208.46*± 0.53	-347.29± 1.23	-347.29± 1.23	**-208.46**± 0.53
SIX-COINS	4	900	-834.24± 2.08	**-488.80**± 2.86	-623.33±19.18	*-556.45*± 1.55	*-556.45*± 1.55	-556.65± 2.14
COLEMAN	4	1521	-460.79± 9.12	-426.26± 4.71	*-425.18*± 4.57	**-425.15**± 4.66	**-425.15**± 4.66	**-425.15**± 4.57
HIV-TEST	4	189	-57.89± 4.02	-32.99± 5.44	-33.11± 5.45	*-32.77*± 5.39	*-32.77*± 5.39	**-32.69**± 5.71
HOUSE-BUILDING	4	531	-162.62± 2.55	-147.96± 4.67	*-145.86*± 5.33	**-145.51**± 5.15	**-145.51**± 5.15	-145.87± 5.33
HANNOVER-5	5	3222	-878.00±42.55	-777.98±40.23	**-761.17**±39.86	-764.63±41.10	-764.65±41.11	*-764.62*±41.78
HANNOVER-8	8	3222	-1417.52±68.93	-1162.82±60.46	*-1116.42*±58.26	-1129.16±55.61	-1129.15±55.63	**-1116.23**±58.81
CAR-EVALUATION	7	774	-717.56±13.30	**-679.24**±10.03	*-686.62*±10.55	-689.73±10.21	-689.73±10.21	-686.83±10.41

TABLE 3
Ten-fold cross-validated running times in seconds for various datasets (rows) and algorithms (columns). Note that these numbers show only tendencies since different dataset/algorithm combinations run on different cluster nodes.

	D	N	IND	CL	LCM	BIN-G	BIN-A	ZHANG
BINARY-FOREST	5	900	0.00± 0.00	0.02± 0.03	23.61± 4.54	*3.70*± 0.23	**2.78**± 0.21	134.35± 1.30
THREE-LEVEL-BINARY	4	900	0.00± 0.00	0.02± 0.03	32.02± 3.83	**2.81**± 0.10	*2.82*± 0.20	66.23± 0.75
THREE-COINS	3	900	0.00± 0.00	0.02± 0.03	2.40± 0.03	*1.19*± 0.04	**1.13**± 0.05	2.98± 0.14
SIX-COINS	4	900	0.00± 0.00	0.03± 0.04	8.21± 0.43	*3.36*± 0.17	**2.73**± 0.16	43.62± 0.63
COLEMAN	4	1521	0.00± 0.01	0.01± 0.03	16.30± 0.86	8.07± 1.85	*7.32*± 1.66	**2.80**± 0.36
HIV-TEST	4	189	0.00± 0.01	0.01± 0.03	0.95± 0.04	*0.42*± 0.09	**0.28**± 0.09	3.38± 0.43
HOUSE-BUILDING	4	531	0.00± 0.01	0.02± 0.04	2.36± 0.15	*1.55*± 0.14	**1.03**± 0.14	1.73± 0.38
HANNOVER-5	5	3222	0.01± 0.01	0.03± 0.08	64.22±14.44	25.13± 0.58	*22.05*± 0.54	**6.73**± 1.25
HANNOVER-8	8	3222	0.01± 0.01	0.04± 0.05	348.63±66.80	*32.97*± 1.14	**30.96**± 5.63	229.76±45.15
CAR-EVALUATION	7	774	0.01± 0.01	0.02± 0.03	4.97± 0.58	*3.60*± 0.11	**2.90**± 0.12	93.04± 2.50

completeness we briefly describe in the following the various datasets.

The BINARY-FOREST data is generated from a model with two separated binary trees. One tree has three nodes (two leaves), the other one five nodes (three leaves). At each node a coin is flipped and incorporated into the state. THREE-LEVEL-BINARY is a full binary tree with 4 leaves, each having 8 states (again with a fair coin flip at each node). THREE-COINS consists of three variables $x_1 = (a_1, a_2)$, $x_2 = (a_2, a_3)$, $x_3 = (a_3, a_1)$ with a_1, a_2, a_3 being three coin flips. Thus x_1 shares a single bit with x_2 and also with x_3, similarly x_2 and x_3. This dataset can not be modelled properly with a binary tree. The SIX-COINS example has four observed variables based on 6 latent variables. Six coins $a_1, \ldots, a_6$ (with values zero and one) are tossed, and $x_1, \ldots, x_4$ are set as $x_1 = a_1 + 2a_3 + 4a_5,\ x_2 = a_2 + 2a_3 + 4a_5,\ x_3 = a_1 + 2a_4 + 4a_6,\ x_4 = a_2 + 2a_4 + 4a_6$. x_1 and x_2 share two bits, x_3 and x_4 share two bits as well. All other pairs share a single bit or none.

For a direct comparison to the work of Zhang (2004) we also use four datasets provided in his paper, namely the Hannover rheumatoid arthritis data on five binary visible variables (HANNOVER-5), and three datasets COLEMAN, HIV-TEST and HOUSE-BUILDING that are all on 4 binary visible variables. The HANNOVER-5 is reduced from a larger dataset given by Kohlmann and Formann (1997) which has 8 binary visible variables; we include this dataset as HANNOVER-8 in the comparison. To have real data with visible cardinalities greater than two we also selected the UCI dataset[6] CAR-EVALUATION which has 7 variables all with cardinality 3 or more.

Summarizing Tables 2 and 3 we conclude that BIN-G and BIN-A provide competitive HLC models in comparison with ZHANG, while at the same time often being faster. In Table 2 BIN-G and BIN-A are the winning algorithms for 4/10 datasets, and are close to ZHANG in all cases except for THREE-COINS (which was designed to make BIN-G and BIN-A fail). Note that LCM is comparatively slow because estimating a single latent node with many states is more expensive than estimating several latent nodes each with few states.

6. available from http://archive.ics.uci.edu/ml/

3.4 Results on COIL-42 and COIL-86

The COIL-86 data[7] is the training set of the COIL Challenge 2000, containing 5822 records and 86 attributes. As in Zhang and Kočka (2004) this dataset was reduced to 42 attributes to produce COIL-42; in this process the cardinalities of the variables was also reduced for many variables. Table 4 compares the BIC scores of BIN-G and BIN-A with the BIC score of ZHANG reported in Zhang and Kočka (2004). All of BIN-G, BIN-A, and ZHANG are better than LCM, with ZHANG being slightly better. However, looking at the run times BIN-G and especially BIN-A excels: our implementation of BIN-G takes 505 seconds, BIN-A takes only 87 seconds, while the reported runtime in Zhang and Kočka (2004) is 121 hours (= 435,600 seconds). Even though computers have got faster since 2004, BIN-G and BIN-A provide a large speedup while delivering good performance.

Additionally, we applied BIN-G and BIN-A to COIL-86. Comparing their BIC scores with LCM and IND we see that our methods have extracted some additional structure in the data. To our knowledge no HLC learning method has previously been successfully applied to COIL-86. The steep increase in runtime from COIL-42 to COIL-86 even though the number of variables only doubled is due to the fact that the original COIL-86 data has many more states per variable than the reduced version COIL-42 (as discussed above).

4 Related Work

Our work is partly inspired by recent ideas by Hinton et al. (2006) on deep belief networks (DBNs), where a greedy layerwise learning procedure is used, with the same learning algorithm applied at each level to the transformed data. Our work uses the same idea, except that rather than transforming all variables in layer ℓ, we select a pair of variables which are replaced by a new latent variable, with all other variables from layer ℓ being copied to the higher layer $\ell + 1$.

In very interesting work, Pearl (1988, Section 8.3) discusses the recovery of latent trees of binary and Gaussian variables in the case that the joint distribution can be exactly decomposed into a latent tree. His algorithm relies on the fact that the correct configuration out of four possibilities for a tree with four visible variables and two internal nodes can be decided based on relationships of pairwise correlations between the visible variables. This fact can be used recursively to connect visible variables one by one into the correct tree structure. However, we note that (i) Pearl does not address the approximation of a distribution $p(\mathbf{x})$ by a latent tree, but only the reconstruction of the underlying true latent tree, (ii) that he assumes that exact pairwise statistics are available (not samples), and (iii) that he only considers *binary* discrete variables.

There has also been much recent work on branching tree models relating to clustering (see e.g. Williams, 2000; Neal, 2003; Teh et al., 2008). However, there is a important difference between such models and our latent trees: in the clustering models each leaf corresponds to a *datapoint*, not a variable. In these models all nodes (both latent and leaf) have the same type and dimension, and the branching tree represents an evolutionary process, inspired by phylogenetic trees (see e.g. Felsenstein, 2004). In contrast we note that for our latent trees the visible variables can have different number of states, and that a latent variable will in general have a different number of states to either of the variables it replaces.

Kemp and Tenenbaum (2008) proposed a method for comparing different model structures for data. Their method builds models (including latent trees) where the leaves correspond to datapoints (or entities in their terms) rather than variables. However, by transposing the data matrix such a method could be used to build a latent tree model. With respect to the details, they use graphical Gaussian models for the tree and this is less suitable for discrete data (Kemp, pers comm, 2008). Also they use a divisive (as opposed to agglomerative) algorithm for tree construction, using a randomized splitting of a node which is repeated several times, and the best split chosen. In our context the divisive approach would have been expensive since the number of possible splits to be evaluated can be large and each evaluation requires the estimation of a local LCM. Thus we opted for an agglomerative algorithm where each step only requires the selection of a pair of variables to be merged.

There is also some recent work on probabilistic hierarchical clustering, e.g. Heller and Ghahramani (2005); Friedman (2003). It might be thought that by transposing the role of the variables and datapoints in the data matrix these methods could be used to obtain a latent tree model for the data. However, neither of the two constructs a generative model as our approach does. In fact Heller and Ghahramani (2005, Section 6) themselves say "[our model] is not in fact a hierarchical generative model of the data, but rather a hierarchical way of organizing nested clusters."

Especially related to the proposed BIN-A algorithm is the work of Connolly (1993) who also constructs the tree topology by running AHC on the variables, measuring the similarity between two variables by their MI. To define the similarity between groups of variables, he computes the criterion (mean pairwise inter-cluster MI) / (mean pairwise intra-cluster MI) and seeks the join that minimizes this criterion. This is an "average link" type criterion in hierarchical clustering parlance. To construct latent variables Connolly uses Fisher's conceptual clustering algorithm COBWEB (Fisher, 1987), rather than LC modelling and EM. In our view Connolly's paper provides some interesting ideas for HLC model learning, but is lacking a firm statistical framework.

Kojadinovic (2004) also discusses a method for the hierarchical clustering of variables based on their mutual information. He considers the single link, average link and complete link criteria. His method does not actually

7. available from http://kdd.ics.uci.edu/databases/tic/tic.html

TABLE 4
BIC scores and running times on COIL-42 and COIL-86 datasets. The BIC score and running times in columns ZHANG are taken from (Zhang & Kočka, 2004). All running times are in seconds.

			BIC SCORE						RUNNING TIME IN SECONDS					
	D	N	IND	CL	LCM	BIN-G	BIN-A	ZHANG	IND	CL	LCM	BIN-G	BIN-A	ZHANG
COIL-42	42	5822	-79,844	*-52,183*	-67,145	-53,533	-53,783	**-51,465**	0.04	1	13,155	*506*	**88**	≈435,600
COIL-86	86	5822	-483,256	-395,703	-456,900	**-375,597**	*-390,597*	—	0.05	19	*13,255*	23,589	**9,799**	—

produce a probability model for the data, but only a hierarchical clustering of the variables.

Wang et al. (2008) proposed an algorithm to construct HLC models to approximate inference in Bayesian networks. In their work the tree structure is determined via a lower bound between the latent variables and the visible variables; this turns out to select the two variables (one from each group under consideration) that maximize the MI. Note that as maximizing similarity (MI) is equivalent to minimizing dissimilarity, this is analogous to single link clustering. In their work the same cardinality is used for all latent nodes, based on trading off inferential complexity against fidelity of approximation of the original Bayesian network. Note that the goal of Wang et al. (2008) is rather different from ours: their HLC model is used to approximate inference in Bayesian networks, while we infer interpretable latent structure, as demonstrated e.g. with the 20 newsgroups example.

5 Conclusion

Estimating HLC models without any restrictions requires searching over a large number of possible latent structures, as implemented in Zhang's framework (Zhang, 2004; Zhang & Kočka, 2004). However, in its full generality it remains a difficult problem and is (like structure learning for Bayes nets) prone to lots of suboptimal solutions and large runtime complexity.

In this paper we have considered the alternative of greedy algorithms to learn *binary* latent tree structures. This leads to a sensible trade-off between model complexity and runtime while preserving expressiveness, as demonstrated in our comprehensive experiments. Due to its favorable runtime scaling, inferring a binary latent tree was even possible on datasets with a large number of variables. Binary latent trees can also be sensible starting points for heuristic procedures that learn more sophisticated models. We did explore various extensions of our algorithm to infer non-binary trees using ideas from information theory. However, preliminary experiments suggest that such approaches are not worth the extra computational costs they require.

Restricting the branching factor of latent trees *does* limit the model class, but still allows rich enough models to take advantage of the hierarchical modelling power of HLC models. Furthermore, the binary tree structure implicitly limits the cardinality of the latent nodes, which facilitates EM estimation; this may be part of the reason why the simple binary latent trees performed comparably to much richer models.

Appendix A
EM-Updates

A.1 Notation

A single observation consists of D variables $x_1, \ldots, x_D$. These observed variables are the leaves of a binary latent tree, which is learned in a greedy manner. The latent nodes are denoted by $x_{D+1}, \ldots, x_R$ with $R = 2D - 1$ being the index of the root. The index of the parent of x_t is denoted by $\mathrm{pa}(t)$. Define the parameters at node t as

$$\theta_{ij}(t) = p(x_t{=}i | x_{\mathrm{pa}(t)}{=}j) \tag{5}$$

$$\pi_i(t) = \bar{p}(x_t{=}i). \tag{6}$$

Note that $\pi_i(t)$ is defined in terms of $\bar{p}$ which is the distribution of the latent subtree with t as the root ignoring all other nodes.

Let $R = 2D - 1$ be the index of the root of the completed binary tree for which $\pi_i(R) = \bar{p}(x_R = i) = p(x_R = i)$. Then we can write the cumulative distribution as

$$p(x_1, \ldots, x_R) = \pi_{x_R}(R) \prod_{t=1}^{R-1} \theta_{x_t, x_{\mathrm{pa}(t)}}(t). \tag{7}$$

Note that the cumulative distribution uses only $\pi(R)$. All other $\pi(t)$ for $t \neq R$ are only used during the greedy building of the tree.

We denote by $\mathrm{in}(t)$ the indices of the observed variables which are descendents of x_t, i.e. the leaves below x_t, briefly denoted by $x_{\mathrm{in}(t)}$. Similar to the inside-outside algorithm for probabilistic context free grammars (or forward-backward algorithm for hidden Markov chains) we define

$$\beta_i(t) = p(x_{\mathrm{in}(t)} | x_t{=}i), \tag{8}$$

which can be calculated recursively as follows: For the leaves, i.e. $t \leq D$, we set

$$\beta_i(t) = \begin{cases} 1 & \text{if } x_t{=}i \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

Latent nodes are calculated recursively from the values of the children, i.e. for $t > D$ and assuming that t has only two children u and v we have

$$\beta_k(t) = p(x_{\mathrm{in}(t)}|x_t{=}k) \tag{10}$$
$$= \sum_{ij} p(x_{\mathrm{in}(u)}, x_{\mathrm{in}(v)}, x_u{=}i, x_v{=}j|x_t{=}k) \tag{11}$$
$$= \sum_{ij} p(x_{\mathrm{in}(u)}|x_u{=}i)p(x_u{=}i|x_t{=}k) \tag{12}$$
$$p(x_{\mathrm{in}(v)}|x_v{=}j)p(x_v{=}j|x_t{=}k) \tag{13}$$
$$= \left(\sum_i \beta_i(u)\theta_{ik}(u)\right)\left(\sum_j \beta_j(v)\theta_{jk}(v)\right). \tag{14}$$

For more children there is a factor for each child. Note that for the leaves $\beta(t)$ are distributions, but the $\beta(t)$s for the other nodes are not necessarily normalized.

A.2 Greedy learning

To simplify notation assume we have only observed a single data point. The current state of the learning process is defined by the frontier which are the variables which have no parents yet. Their distributions are defined by π. For the leaves we have (for $t \leq D$)

$$\pi_i(t) = \beta_i(t). \tag{15}$$

For these distributions we can calculate all pairwise mutual informations and choose the maximizing pair of variables. Let's denote those two variables by u and v, for which we wish to learn a common parent x_t. For this we maximise the likelihood of the leaves below u and v which will be denoted by $x_{\mathrm{in}(t)} = x_{\mathrm{in}(u)} \cup x_{\mathrm{in}(v)}$,

$$p(x_{\mathrm{in}(t)}) = \sum_k \beta_k(t)\pi_k(t) \tag{16}$$
$$= \sum_k \left(\sum_i \beta_i(u)\theta_{ik}(u)\right)\left(\sum_j \beta_j(v)\theta_{jk}(v)\right)\pi_k(t). \tag{17}$$

From the previous iterations we have $\beta(u)$ and $\beta(v)$. The parameters we need to learn are $\pi(t)$ and $\theta(u)$ and $\theta(v)$. For several observations $x^{(1)}, \ldots, x^{(N)}$ we maximize

$$L = \prod_{n=1}^N p(x^{(n)}_{\mathrm{in}(t)}) = \prod_{n=1}^N \sum_k \beta^{(n)}_k(t)\pi_k(t) \tag{18}$$
$$= \prod_{n=1}^N \sum_k \left(\sum_i \beta^{(n)}_i(u)\theta_{ik}(u)\right)\left(\sum_j \beta^{(n)}_j(v)\theta_{jk}(v)\right)\pi_k(t) \tag{19}$$
$$= \prod_{n=1}^N \sum_{ijk} \beta^{(n)}_i(u)\theta_{ik}(u)\beta^{(n)}_j(v)\theta_{jk}(v)\pi_k(t) \tag{20}$$
$$= \prod_{n=1}^N \sum_{ijk} p(\text{"latent"} = (i,j,k), \text{"observed"}^{(n)}). \tag{21}$$

A.3 Updates

For the EM-updates we need to consider the log-likelihood from Equation (20)

$$\log L = \sum_{n=1}^N \log \sum_{ijk} \beta^{(n)}_i(u)\theta_{ik}(u)\beta^{(n)}_j(v)\theta_{jk}(v)\pi_k(t). \tag{22}$$

As usual the difficulty is that the logarithm cannot be moved past the inner sum. Thus we assume a distribution $q^{(n)}(i,j,k) = q(x^{(n)}_u = i, x^{(n)}_v = j, x^{(n)}_t = k|x^{(n)}_{\mathrm{in}(t)})$ and consider the expected complete data log-likelihood

$$Q = \sum_{n=1}^N \sum_{ijk} q^{(n)}(i,j,k) \log \beta^{(n)}_i(u)\theta_{ik}(u)\beta^{(n)}_j(v)\theta_{jk}(v)\pi_k(t). \tag{23}$$

Q is then maximized with respect to the parameters $\theta_{ik}(u)$, $\theta_{jk}(v)$ and $\pi_k(t)$. For instance for $\pi_k(t)$ we take the derivatives of the Lagrangian

$$Q - C(\sum_k \pi_k(t) - 1), \tag{24}$$

where we added a term to enforce $\sum_k \pi_k(t) = 1$. For the other parameters we add similar terms. Equating the derivatives to zero, we obtain the updates for the M-step

$$\theta_{ik}(u) \propto \sum_{n=1}^N \sum_j q^{(n)}(i,j,k) = \sum_{n=1}^N q^{(n)}(i,k) \tag{25}$$
$$\theta_{jk}(v) \propto \sum_{n=1}^N \sum_i q^{(n)}(i,j,k) = \sum_{n=1}^N q^{(n)}(j,k) \tag{26}$$
$$\pi_k(t) \propto \sum_{n=1}^N \sum_{ij} q^{(n)}(i,j,k) = \sum_{n=1}^N q^{(n)}(k). \tag{27}$$

For brevity we write $q^{(n)}(i,k) = \sum_j q^{(n)}(i,j,k)$, similarly for $q^{(n)}(j,k)$ and $q^{(n)}(k)$.
The E-step employs the current parameters θ and π

$$q^{(n)}(i,j,k) = p(x^{(n)}_u{=}i, x^{(n)}_v{=}j, x^{(n)}_t{=}k|x^{(n)}_{\mathrm{in}(t)}) \tag{28}$$
$$= \frac{p(x^{(n)}_u{=}i, x^{(n)}_v{=}j, x^{(n)}_t{=}k, x^{(n)}_{\mathrm{in}(t)})}{p(x^{(n)}_{\mathrm{in}(t)})} \tag{29}$$
$$= \frac{\beta^{(n)}_i(u)\theta_{ik}(u)\beta^{(n)}_j(v)\theta_{jk}(v)\pi_k(t)}{\sum_k \beta^{(n)}_k(t)\pi_k(t)}. \tag{30}$$

This leads to expressions for the subterms

$$q^{(n)}(i,k) = \frac{\beta^{(n)}_i(u)\theta_{ik}(u)\pi_k(t)\sum_j \beta^{(n)}_j(v)\theta_{jk}(v)}{\sum_k \beta^{(n)}_k(t)\pi_k(t)} \tag{31}$$
$$q^{(n)}(j,k) = \frac{\beta^{(n)}_j(v)\theta_{jk}(v)\pi_k(t)\sum_i \beta^{(n)}_i(u)\theta_{ik}(u)}{\sum_k \beta^{(n)}_k(t)\pi_k(t)} \tag{32}$$
$$q^{(n)}(k) = \frac{\beta^{(n)}_k(t)\pi_k(t)}{\sum_k \beta^{(n)}_k(t)\pi_k(t)}. \tag{33}$$

Combining these formulas of the E-step with the M-step we get the following multiplicative updates

$$\theta_{ik}(u) \propto \theta_{ik}(u)\pi_k(t) \sum_j \theta_{jk}(v) \sum_n \frac{\beta_i^{(n)}(u)\beta_j^{(n)}(v)}{\sum_k \beta_k^{(n)}(t)\pi_k(t)} \quad (34)$$

$$\theta_{jk}(v) \propto \theta_{jk}(v)\pi_k(t) \sum_i \theta_{ik}(u) \sum_n \frac{\beta_i^{(n)}(u)\beta_j^{(n)}(v)}{\sum_k \beta_k^{(n)}(t)\pi_k(t)} \quad (35)$$

$$\pi_k(t) \propto \pi_k(t) \sum_{ij} \theta_{ik}(u)\theta_{jk}(v) \sum_n \frac{\beta_i^{(n)}(u)\beta_j^{(n)}(v)}{\sum_k \beta_k^{(n)}(t)\pi_k(t)}. \quad (36)$$

Appendix B
Tree refinement

Tree refinement requires three steps:

1) bottom-up pass to generate β messages
2) top-down pass to generate α and $\overline{\alpha}$ messages
3) updating the conditional probability tables (CPTs)

In this section we define the messages as follows

$$\beta_i(t) := p(x_{\text{in}(t)}|x_t{=}i) \quad (37)$$

$$\alpha_i(t) := p(x_t{=}i|x_{\text{out}(t)}). \quad (38)$$

Note that in Pearl's notation (1988, see Eqs. (4.15) and (4.16)) our messages $\beta_i(t)$ correspond to $\lambda(x_t) = p(e^-_{x_t}|x_t)]$, and $\alpha_i(t)$ correspond to $\pi(x_t) = p(x_t|e^+_{x_t})$.

B.1 Bottom-up propagation

For a leaf x_t we have

$$\beta_i(t) = \begin{cases} 1 & \text{if } x_t = i \text{ is observed} \\ 0 & \text{otherwise.} \end{cases} \quad (39)$$

For all other nodes x_t we have

$$\beta_i(t) = \prod_{s \in \text{children}(t)} \sum_j \theta_{ji}(s)\beta_j(s). \quad (40)$$

B.2 Top-down propagation

For the root node we simply copy the current distribution (denoted by α without round brackets)

$$\alpha_i(t) = \alpha_i. \quad (41)$$

For children we have

$$\overline{\alpha}_k(t) \propto \alpha_k(\text{pa}(t)) \prod_{s \in \text{siblings}(t)} \sum_j \theta_{jk}(s)\beta_j(s) \quad (42)$$

$$\alpha_i(t) = \sum_k \overline{\alpha}_k(t)\theta_{ik}(t). \quad (43)$$

Note that $\overline{\alpha}_k(t)$ needs to be normalized.

B.3 Update CPTs

To estimate the parameters θ_{ik} we need

$$q(i,k) = p(x_t = i, x_{\text{pa}(t)} = k|x_{\text{all}}) \quad (44)$$

$$\propto \beta_i(t)\theta_{ik}(t)\overline{\alpha}_k(t). \quad (45)$$

Such $q(i,k)$ is calculated for each data vector, thus we write $q^{(n)}(i,k)$. All of these are summed up and normalized, to yield updates for θ_{ik}

$$\theta_{ik} \propto \sum_n q^{(n)}(i,k). \quad (46)$$

Similarly, for parameter π_k we have

$$q(k) = p(x_{\text{root}} = k|x_{\text{all}}) \quad (47)$$

$$\propto \beta_k(\text{root})\alpha_k \quad (48)$$

$$\pi_k \propto \sum_n q^{(n)}(k). \quad (49)$$

Acknowledgments

We thank Nevin L. Zhang for making compiled code of the algorithm in Zhang and Kočka (2004) available, and Christoph Lampert for generating the vision datasets. We thank the anonymous reviewers for their comments that helped improve the paper; in particular we thank the reviewer who pushed us to formalize the BIN-A algorithm and compare it with BIN-G. Furthermore, SH thanks Dominik Janzing and Hannes Nickisch for general discussion. This work is supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778. This publication only reflects the authors' views.

References

Blei, D., Ng, A., & Jordan, M. (2003). Latent Dirichlet Allocation. *Journal of Machine Learning Research*, *3*, 993–1022.

Bollen, K. A. (1989). *Structural Equations with Latent Variables*. Wiley.

Chow, C. K., & Liu, C. N. (1968). Approximating Discrete Probability Distributions with Dependence Trees. *IEEE Transactions on Information Theory*, *14*, 462–467.

Connolly, D. (1993). Constructing Hidden Variables in Bayesian Networks via Conceptual Clustering. *Proceedings of the Tenth International Conference on Machine Learning* (pp. 65–72).

Duda, R. O., & Hart, P. E. (1973). *Pattern Classification and Scene Analysis*. New York: Wiley.

Felsenstein, J. (2004). *Inferring Phylogenies*. Sunderland, Mass.: Sinauer Associates.

Feng, X., Williams, C. K. I., & Felderhof, S. N. (2002). Combining Belief Networks and Neural Networks for Scene Segmentation. *IEEE Transasctions on Pattern Analysis and Machine Intelligence*, *24*, 467–483.

Fisher, D. H. (1987). Knowledge Acquisition Via Incremental Conceptual Clustering. *Machine Learning*, *2*, 139–172.

Friedman, N. (2003). *Pcluster: Probabilistic Agglomerative Clustering of Gene Expression Profiles* (Technical Report 80). Hebrew University.

Heller, K., & Ghahramani, Z. (2005). Bayesian Hierarchical Clustering. *Twenty-second International Conference on Machine Learning* (pp. 297–304).

Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, *18*, 1527–1554.

Kemp, C., & Tenenbaum, J. B. (2008). The Discovery of Structural Form. *Proceedings of the National Academy of Sciences*, *105*, 10687–10692.

Kohlmann, T., & Formann, A. K. (1997). Using Latent Class Models to analyze Response Patterns in Epidemiologic Mail Surveys. In J. Rost and R. Langeheine (Eds.), *Applications of latent trait and latent class models in the social sciences*. Waxman Verlag.

Kojadinovic, I. (2004). Agglomerative Hierarchical Clustering of Continuous Variables based on Mutual Information. *Computational Statistics and Data Analysis*, *46*, 269–294.

Lazarsfeld, P. F., & Henry, N. W. (1968). *Latent Structure Analysis*. Boston, Mass.: Houghton Mifflin.

Neal, R. M. (2003). Density Modeling and Clustering using Dirichlet Diffusion Trees. In J. M. Bernardo et al. (Eds.), *Bayesian Statistics 7*, 619–629. Oxford University Press.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann.

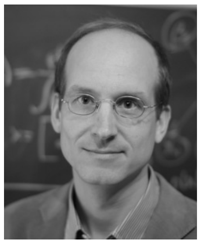
Teh, Y. W., Daume III, H., & Roy, D. M. (2008). Bayesian Agglomerative Clustering with Coalescents. In J. Platt, D. Koller, Y. Singer and S. Roweis (Eds.), *Advances in Neural Information Processing Systems 20*. MIT Press.

Wang, Y., Zhang, N., & Chen, T. (2008). Latent Tree Models and Approximate Inference in Bayesian Networks. *Journal of Artificial Intelligence Research*, *32*, 879–900.

Williams, C. K. I. (2000). A MCMC approach to Hierarchical Mixture Modelling . In S. A. Solla, T. K. Leen and K.-R. Müller (Eds.), *Advances in Neural Information Processing Systems 12*. MIT Press.

Zhang, N. L. (2004). Hierachical Latent Class Models for Cluster Analysis. *Journal of Machine Learning Research*, *5*, 697–723.

Zhang, N. L., & Kočka, T. (2004). Efficient Learning of Hierarchical Latent Class Models. *Proceedings of the 16th IEEE International Conference on Tools with AI (ICTAI-2004)*.



Stefan Harmeling is a Research Scientist at the Max Planck Institute for Biological Cybernetics in Prof Bernhard Schölkopf's department of Empirical Inference. His interests include machine learning, image processing, probabilistic and causal inference, and general computer science.

Dr Harmeling studied mathematics and logic at the University of Münster (Dipl Math 1998) and computer science with an emphasis on artificial intelligence at Stanford University (MSc 2000). During his doctoral studies he was a member of Prof Klaus-Robert Müller's research group at the Fraunhofer Institute FIRST (Dr rer nat 2004). Thereafter he was a Marie Curie Fellow at the University of Edinburgh from 2005 to 2007, and works since then at the Max Planck Institute of Biological Cybernetics.

Christopher K.I. Williams is Professor of Machine Learning and Director of the Institute for Adaptive and Neural Computation in the School of Informatics, University of Edinburgh. He is interested in a wide range of theoretical and practical issues in machine learning, statistical pattern recognition, probabilistic graphical models and computer vision.

He studied neural networks/AI with Prof Geoffrey Hinton at the University of Toronto (MSc 1990, PhD 1994). Subsequently he was a member of the Neural Computing Research Group at Aston University from 1994 to 1998, before joining the School of Informatics, University of Edinburgh.

Prof Williams is a member of the editorial boards of the Journal of Machine Learning Research and the International Journal of Computer Vision. He was program co-chair of the Neural Information Processing Systems (NIPS) conference in 2009.

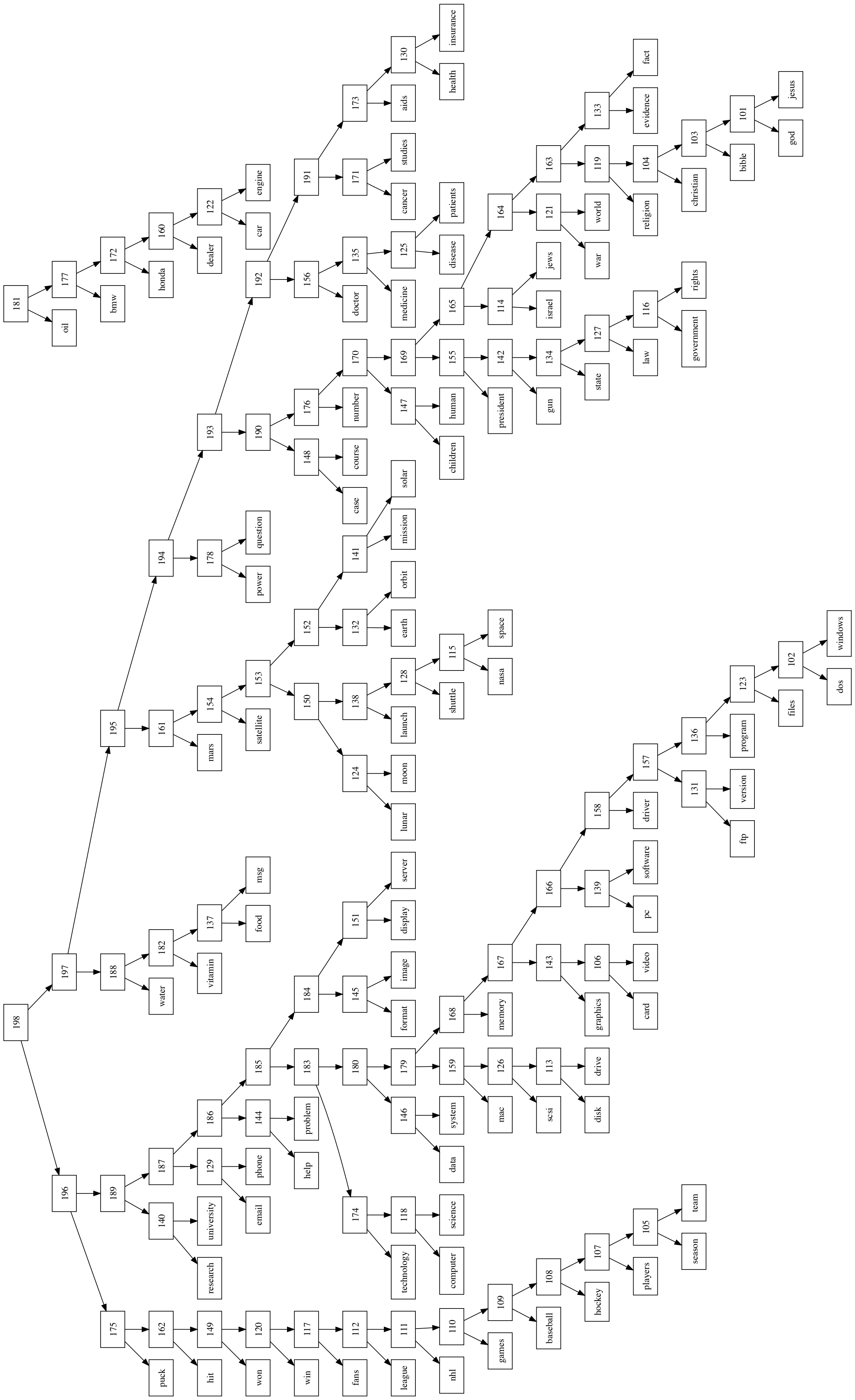


Fig. 2. Binary forest inferred by method BIN-G modelling co-occurrences in the 20 newsgroup dataset.

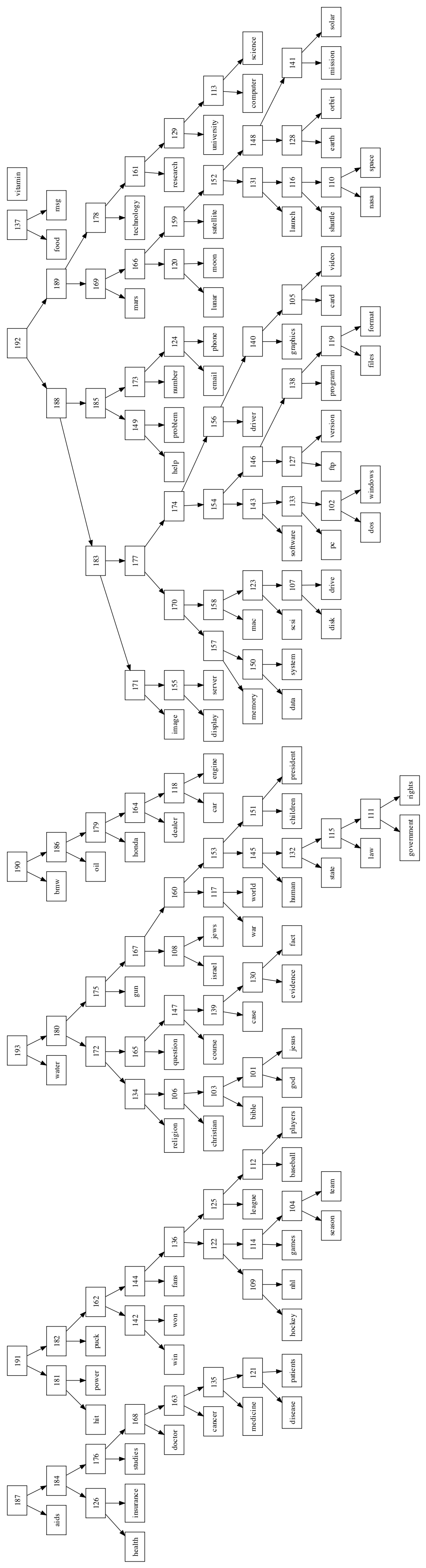


Fig. 3. Binary forest inferred by method BIN-A modelling co-occurrences in the 20 newsgroup dataset.

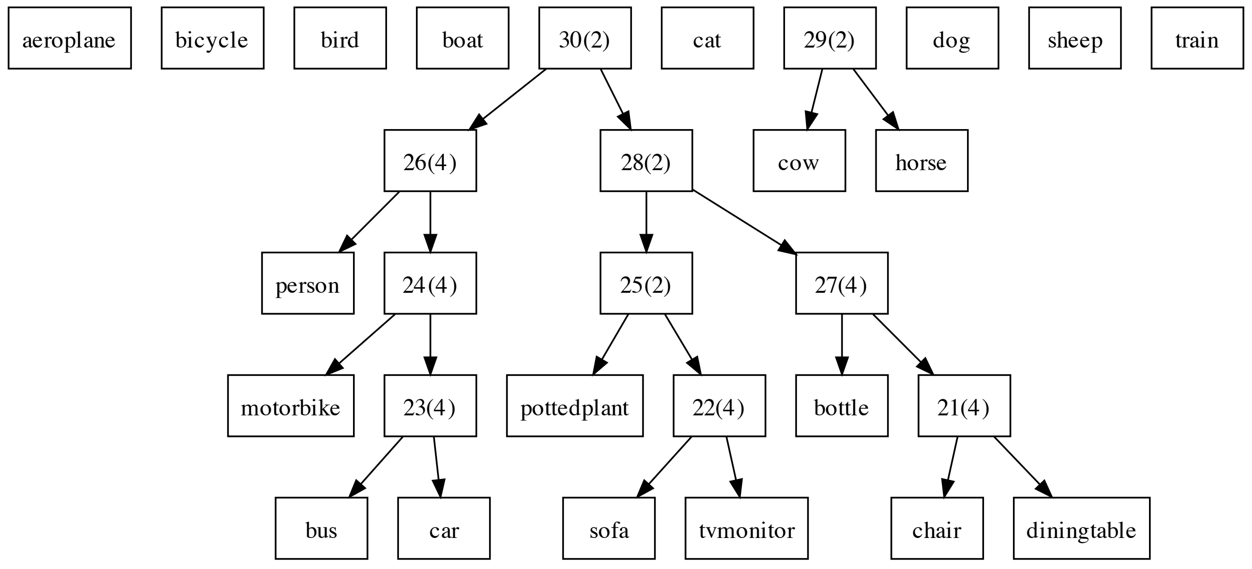


Fig. 4. Binary forest inferred by method BIN-G modelling co-occurrences in the PASCAL VOC 2007 data.