International Journal of Approximate Reasoning 54 (2013) 560–572

Contents lists available at SciVerse ScienceDirect

International Journal of Approximate Reasoning

journal homepage: www.elsevier.com/locate/ijar

LTC: A latent tree approach to classification ☆

Yi Wang [a,*], Nevin L. Zhang [b], Tao Chen [c], Leonard K.M. Poon [b]

[a] Department of Computer Science, National University of Singapore, Singapore 117417, Singapore
[b] Department of Computer Science & Engineering, The Hong Kong University of Science & Technology, Clear Water Bay, Kowloon, Hong Kong
[c] EMC Labs China, Beijing, China

ARTICLE INFO

Article history:
Available online 11 July 2012

Keywords:
Bayesian network classifier
Latent tree model

ABSTRACT

Latent tree models were proposed as a class of models for unsupervised learning, and have been applied to various problems such as clustering and density estimation. In this paper, we study the usefulness of latent tree models in another paradigm, namely supervised learning. We propose a novel generative classifier called latent tree classifier (LTC). An LTC represents each class-conditional distribution of attributes using a latent tree model, and uses Bayes rule to make prediction. Latent tree models can capture complex relationship among attributes. Therefore, LTC is able to approximate the true distribution behind data well and thus achieves good classification accuracy. We present an algorithm for learning LTC and empirically evaluate it on an extensive collection of UCI data. The results show that LTC compares favorably to the state-of-the-art in terms of classification accuracy. We also demonstrate that LTC can reveal underlying concepts and discover interesting subgroups within each class.

1. Introduction

Latent tree models (LTMs) are tree-structured Bayesian networks in which variables at leaf nodes are observed and called *manifest variables*, whereas variables at internal nodes are hidden and called *latent variables*. LTM was proposed as a class models for unsupervised learning [2]. Recently, they have received increasing attention and have been applied to various problems including clustering [2,3], density estimation [4,5], and latent structure discovery [6–8], among the others.

In this paper, we take an alternative view and study the usefulness of LTM in another learning paradigm, namely supervised learning. In particular, we explore the application of LTM to the classification problem.

Classification is one of the most active areas in machine learning research. The task is to predict the class label of an instance based on a set of attributes that describe the instance. Approaches to this problem divide into two categories: Generative and discriminative [9]. Let C be the class variable and $\mathbf{X}$ be the set of attributes. Generative approaches build models for the joint distribution $P(C, \mathbf{X})$, compute the posterior distribution $P(C|\mathbf{X})$ using Bayes rule, and assign an instance to the most likely class. In contrast, discriminative approaches directly model $P(C|\mathbf{X})$. In this paper, we focus on generative approaches and assume categorical attributes.

The simplest generative model is the naive Bayes (NB) classifier [10]. It assumes that attributes are mutually independent given the class label. All dependencies among attributes are ignored. An example NB is shown in Fig. 1(a). Despite its simplicity, NB has been shown to be surprisingly effective in a number of domains [11].

The conditional independence assumption underlying NB is rarely true in practice. Violating this assumption could lead to poor prediction performance. The past decade has seen a large body of work on relaxing this unrealistic assumption. To

☆ This is an extended version of the paper [1] presented at the 11th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU-11), Belfast, UK, 2011.

* Corresponding author.
E-mail addresses: wangy@comp.nus.edu.sg (Y. Wang), lzhang@cse.ust.hk (N.L. Zhang), tao.chen2@emc.com (T. Chen), lkmpoon@cse.ust.hk (L.K.M. Poon).

0888-613X/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ijar.2012.06.024

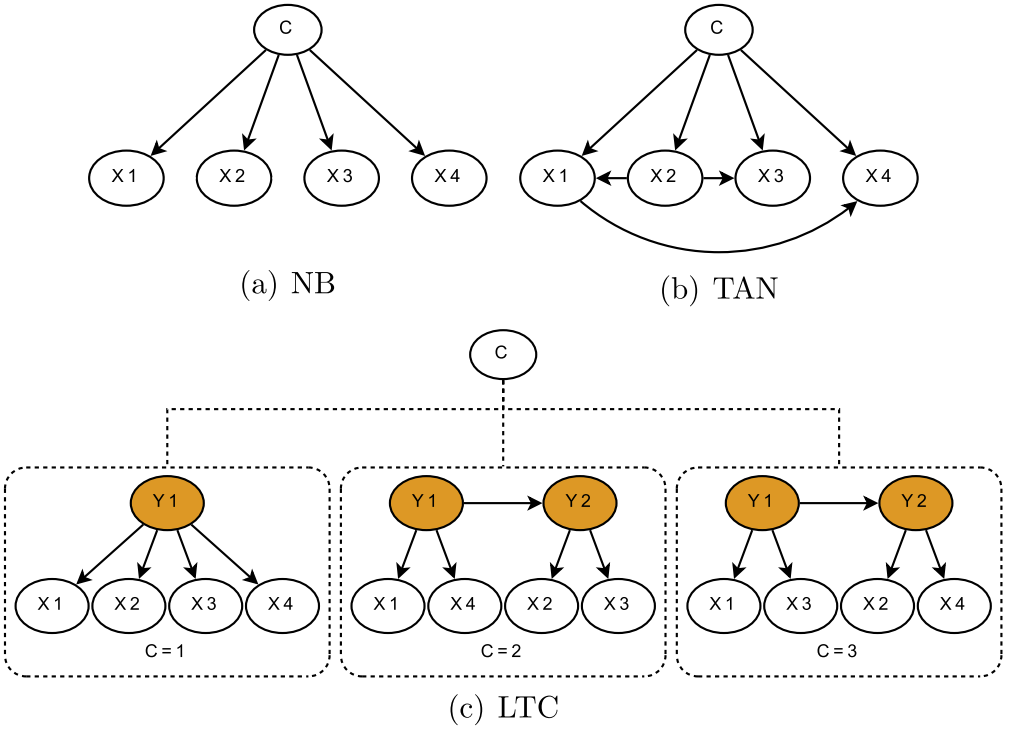


Fig. 1. Example NB, TAN, and LTC. C is the class variable with 3 classes, X_1–X_4 are four attributes, Y_1 and Y_2 are latent variables.

mention two successful instances, tree augmented naive Bayes (TAN) [12] builds a Chow–Liu tree [13] to model the attribute dependencies (see Fig. 1(b) for an example), while averaged one-dependence estimators (AODE) [14] constructs a set of tree models over the attributes and averages them to make prediction.

In this paper, we propose a novel approach to model the relationship among attributes. Our approach is based on LTM. It treats attributes as manifest variables and build LTMs to capture the relationship among them. The relationship could be different across classes. Therefore, we build a separate LTM for each class. We refer to the collection of LTMs plus the prior class distribution as a latent tree classifier (LTC). An example LTC is shown in Fig. 1(c). Each rectangle in the figure contains the LTM for a class.

Our approach exploits the merit of LTM that such model can represent a set of complex relationship among the manifest variables in a compact way. To see this point, consider eliminating all the latent variables from an LTM. This will result in a fully connected Bayesian network over all the manifest variables. As for our approach, the use of LTM makes it possible to capture complex relationship among attributes while retain the model simplicity. Therefore, we expect LTC to approximate the true distribution $P(C, \mathbf{X})$ behind data well and thus to achieve good classification accuracy. We empirically verify this hypothesis in the experiments.

In addition to good classification performance, building LTC on the basis of LTM also makes it possible to discover latent structures behind data. In particular, we will demonstrate that the latent variables introduced during the learning process can reveal concepts embedded in data as well as interesting subgroups within each class. This merit can further boost user confidence in LTC.

LTC exploits the strong expressive capability and the compactness of LTM. One can also replace LTM with other latent variable models. A straightforward choice is latent class model [15], aka naive Bayes with latent variable. It is a special LTM with one single latent variable. By restricting to latent class models, we obtain latent class classifier (LCC). In our experiments, we also compare LTC with LCC. The results show that LTC is superior to LCC. We attribute this to the flexibility of LTM.

Making prediction with LTC requires marginalizing out all the latent variables from each LTM. Since LTM is tree-structured, this can be done in time linear in the number of attributes. In the experiments we show that LTC is efficient for classification. On the other hand, we notice that learning LTC can be time consuming due to the introduction of latent variables. Therefore, LTC is currently most suitable for applications which allow a long offline training phase but demand good online classification performance.

1.1. Related work

There are a large body of literatures that attempt to improve classification accuracy by exploiting attribute dependencies. They mainly divide into two categories: Those that directly model relationship among attributes, and those that capture

such relationship using latent variables. TAN, AODE, and Bayesian network augmented naive Bayes (BAN) fall into the first category. Another representative from this category is Bayesian multinet [12]. It learns a Bayesian network for each class and uses them jointly to make prediction. Our method is related to Bayesian multinet in the sense that we also accommodate inter-class heterogeneity. However, we learn an LTM instead of a Bayesian network to represent each class-conditional distribution.

Our method falls into the second category. In this category, various latent variable models have been tested for continuous data. To give two examples, Monti and Cooper [16] combine finite mixture model with naive Bayes classifier. The resultant model is a continuous counterpart of LCC. Langseth and Nielsen [17] propose latent classification model. It uses a mixture of factor analyzers to represent attribute dependencies.

In contrast, we are aware of much less work on categorical data. The one that is the most closely related to ours is the hierarchical naive Bayes model (HNB) [18,19]. HNB also exploits LTM to capture the relationship among attributes. However, it differs from LTC in two aspects. First, the roles of LTM in HNB and LTC are different. In HNB, LTM is used as a tool for feature extraction. There, attributes are partitioned into disjoint subsets, while each subset is modeled using a separate LTM. The root (latent) nodes of those LTMs are then treated as features extracted from different subsets of attributes, and are put together with the class variable to form a naive Bayes model for classification. In contrast, LTC builds one single LTM to connect all attributes for each class. The LTM as a whole serves as a generative model for that class, which is used in the prediction phase to compute the likelihood of new data points.

Second, HNB assumes homogeneous latent structure, *i.e.*, the LTMs in HNB are identical throughout all classes. This assumption could be unrealistic in real world applications. Violating this assumption could lead to degenerated classification performance and failure in latent structure discovery. In contrast, LTC describes different classes using different LTMs. Therefore, it can accommodate the heterogeneity across different classes.

We did not include HNB in our experiments. The learning algorithm proposed by Zhang et al. [18] can only deal with several attributes and does not scale up to most data sets used in our experiments. Langseth and Nielsen [19] develop a more efficient learning algorithm but we have not been able to gain access to their implementation.

Recently, Langseth and Nielsen [20] extends the latent classification model to discrete domain. The proposed model only handles binary attributes. Its generalization to multi-valued categorical attributes is non-trivial.

1.2. Organization

The rest of this paper is structured as follows. We formally define LTC in Section 2. We then present an algorithm for learning LTC in Section 3. In Section 4, we empirically evaluate LTC on 37 UCI data sets. We first examine the impacts of two designs in the learning algorithm on the classification performance of LTC. We then compare LTC with a spectrum of generative classifiers as well as C4.5 [21]. In Section 5, we demonstrate that LTC can discover appealing latent structures using an example. Finally, we conclude this paper in Section 6 and point out several future directions.

2. Latent tree classifier

We start by briefly reviewing latent tree models (LTMs). An LTM is a pair $\mathcal{M} = (m, \boldsymbol{\theta})$. The first component m denotes the rooted tree and the set of cardinalities of the latent variables. The second component $\boldsymbol{\theta}$ denotes the collection of parameters in $\mathcal{M}$. It contains a conditional probability table (CPT) for each node given its parent.

Let $\mathbf{X}$ and $\mathbf{Y}$ be the set of manifest variables and the set of latent variables in $\mathcal{M}$, respectively. We use $P(\mathbf{X}, \mathbf{Y}|\mathcal{M})$ to denote the joint distribution represented by $\mathcal{M}$. Two LTMs $\mathcal{M}$ and $\mathcal{M}'$ are *marginally equivalent* if they share the same set of manifest variables $\mathbf{X}$ and $P(\mathbf{X}|\mathcal{M}) = P(\mathbf{X}|\mathcal{M}')$.

Let $|Z|$ denote the cardinality of variable Z. For a node Z in $\mathcal{M}$, we denote the set of its neighbors by $\mathbf{nb}(Z)$. An LTM is *regular* if for any latent node Y, $|Y| \leq \frac{\prod_{Z \in \mathbf{nb}(Y)} |Z|}{\max_{Z \in \mathbf{nb}(Y)} |Z|}$, and the inequality strictly holds when Y has only two neighbors. As shown by Zhang [2], an irregular model $\mathcal{M}$ is over-complicated and can be reduced to a regular model $\mathcal{M}'$ which is marginally equivalent but contains fewer parameters than $\mathcal{M}$. Henceforth, we consider only regular models.

We consider the classification problem where each instance is described using n attributes $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$, and belongs to one of the r classes $C = 1, 2, \ldots, r$. A *latent tree classifier* (LTC) consists of a prior distribution $P(C)$ on C and a collection of r LTMs over the attributes $\mathbf{X}$. We denote the c-th LTM by $\mathcal{M}_c = (m_c, \boldsymbol{\theta}_c)$ and the set of latent variables in $\mathcal{M}_c$ by $\mathbf{Y}_c$. The LTC represents a joint distribution over C and $\mathbf{X}$, $\forall c = 1, 2, \ldots, r$,

$$P(C = c, \mathbf{X}) = P(C = c)P(\mathbf{X}|\mathcal{M}_c) = P(C = c) \sum_{\mathbf{Y}_c} P(\mathbf{X}, \mathbf{Y}_c|\mathcal{M}_c). \tag{1}$$

Given an LTC, we classify an instance $\mathbf{X} = \mathbf{x}$ to the class $c^\star$, where

$$c^\star = \arg\max_C P(C|\mathbf{X} = \mathbf{x}) = \arg\max_C P(C, \mathbf{X} = \mathbf{x}). \tag{2}$$

Table 1
The classification time complexity.

LTC	NB	TAN	AODE
$O(rnv^2)$	$O(rn)$	$O(rn)$	$O(rn^2)$

Note that, according to Eq. (1), this requires us to sum out all the latent variables $\mathbf{Y}_c$ for each class c. Thanks to the tree structure of LTM, the summation could be done in linear time in the number of attributes, as formalized below.

Proposition 1. *The time complexity of classifying an unlabeled instance with an LTC is $O(rnv^2)$, where r is the number of classes, n is the number of attributes, and v is the maximum cardinality of variables in the LTC.*

Proof. The time complexity of summing out all the latent variables $\mathbf{Y}_c$ from the c-th LTM $\mathcal{M}_c$ is $O((|\mathbf{Y}_c| + n)v_c^2)$ [22], where $|\mathbf{Y}_c|$ denotes the number of latent variables in $\mathbf{Y}_c$, and v_c denotes the maximum cardinality of the variables in $\mathcal{M}_c$. It is known that a regular LTM contains fewer than n latent variables [2]. Therefore, the overall time complexity for classifying an instance is $O(rnv^2)$. □

We compare the time complexity of LTC with those of NB, TAN, and AODE in Table 1. We will empirically evaluate the classification efficiency of those algorithms in the experiments.

3. A learning algorithm

Given a labeled training set $\mathcal{D}$, we now present an algorithm for learning an LTC from $\mathcal{D}$. The algorithm consists of four steps:

1. Calculate the maximum likelihood estimate (MLE) $\hat{P}(C)$ of $P(C)$ from $\mathcal{D}$.
2. Split the data $\mathcal{D}$ according to class label into r subsets, one for each class. Denote the subset for class c by $\mathcal{D}_c$.
3. For each class $c = 1, 2, \ldots, r$, learn an LTM $\mathcal{M}_c$ from $\mathcal{D}_c$ to estimate the underlying class-conditional distribution $P(\mathbf{X}|C = c)$.
4. Smooth the estimate $\hat{P}(C)$ and the parameters of each LTM $\mathcal{M}_c$.

In Step 1, we calculate the MLE $\hat{P}(C)$. This can be easily done by counting the number of instances belonging to each class in $\mathcal{D}$. More specifically, we calculate

$$\hat{P}(C = c) = \frac{|\mathcal{D}_c|}{|\mathcal{D}|}.$$

Clearly, the more challenging task is Step 3, *i.e.*, to learn good LTMs to estimate the class-conditional distributions underlying the data. We address this problem in the next two subsections. We then discuss parameter smoothing in Step 4 in Section 3.3.

3.1. Model selection

To learn an LTM, one needs to determine the number of latent variables, the cardinality of each latent variable, the tree topology that connects the latent variables and attributes, as well as the model parameters. Those factors jointly lead to a huge model space, and our task is to find a good model $\mathcal{M}_c$ from the space for each subset $\mathcal{D}_c$.

An LTM $\mathcal{M}_c$ is of high quality if it is close to the true distribution $P(\mathbf{X}|C = c)$ underlying $\mathcal{D}_c$. Nonetheless, the true distribution is unknown in the first place. Therefore, we use the AIC score [23] for model selection,

$$AIC(m_c|\mathcal{D}_c) = -2\log P(\mathcal{D}_c|m_c, \boldsymbol{\theta}_c^\star) + 2d(m_c), \tag{3}$$

where $\boldsymbol{\theta}_c^\star$ is the maximum likelihood estimate to the parameters $\boldsymbol{\theta}_c$, and $d(m_c)$ is the number of free parameters in model m_c. The AIC score is an approximation to the expected KL divergence of $\mathcal{M}_c$ from the true distribution $P(\mathbf{X}|C = c)$. The lower the score, the smaller the difference between $\mathcal{M}_c$ and the true distribution, and the better the LTM.

There exists other approximations to the expected KL divergence. For example, leave-one-out cross-validation (LOOCV) is asymptotically equivalent to the AIC score [24]. However, LOOCV is computationally much more demanding to use than the AIC score. Calculating LOOCV for an LTM needs to optimize the model parameters $|\mathcal{D}_c|$ times. In contrast, calculating AIC score only requires one pass of parameter optimization. Researchers have also proposed variants to the AIC score, *e.g.*, the AICc score [25] which makes correction for handling small sample sizes. However, the original AIC score is still the most widely used approximation.

In literatures, the BIC score [26] is used more often for learning Bayesian network classifiers [12]. In contrast to AIC, BIC is based on a different philosophy. Given a training data, the assumption is that the model space contains the true model

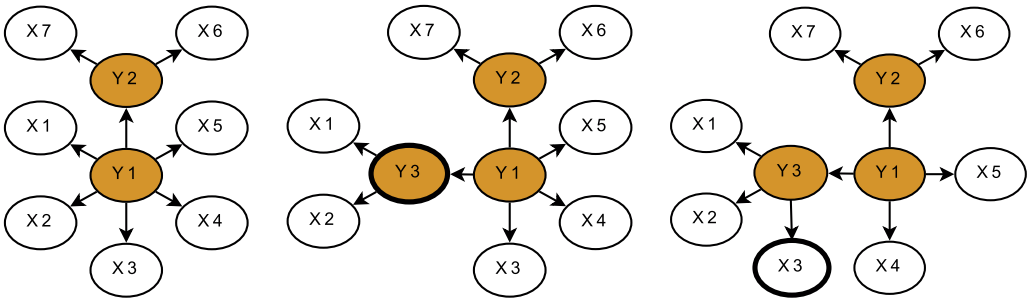


Fig. 2. Illustration of NI, ND, and NR operators.

underlying the data. The objective is thus to find the most probable model from the space. The probability of a model is known as the marginal likelihood. The BIC score is a large sample approximation to the marginal likelihood.

Clearly, the assumption of BIC does not hold in our setting. The true model of the data $\mathcal{D}_c$ is not necessarily an LTM. Therefore, we argue that BIC is not suitable for our problem. Moreover, in practice, the BIC score tends to over-penalize complex models and can lead to poor approximation to the true distribution. In Section 4.4, we empirically compare AIC with BIC for learning LTC. The results show that the LTCs learned using AIC achieve better classification accuracy than those learned using BIC.

3.2. Model search

Given the data $\mathcal{D}_c$, our task now is to find an LTM $\mathcal{M}_c$ with good AIC score from the model space. Since the model space is prohibitively large, we adopt a recently proposed hill-climbing algorithm called EAST [3] for this task.

EAST explores the model space using five search operators. They are *node introduction* (NI), *node deletion* (ND), *node relocation* (NR), *state introduction* (SI), and *state deletion* (SD). Given an LTM, NI applies to a latent variable and two of its neighbors. It adds a new latent variable to mediate the latent variable and the two neighbors, and sets the cardinality of the new latent variable to the same as the existing latent variable. Fig. 2 gives a concrete example. The model m_2 is obtained from m_1 by introducing a new latent variable Y_3 for latent variable Y_1 and its neighbors X_1, X_2. ND is the reverse of NI. It applies to two neighboring latent variables, removes one of them and links its neighbors to the other. In Fig. 2, by deleting Y_3 one goes from m_2 back to m_1. NR operator adjusts the connections in an LTM. It considers two latent variables, disconnects a neighbor from the first latent variable, and links it to the second latent variable. In Fig. 2, one relocates X_3 in m_2 from Y_1 to Y_3 and obtains m_3. The last two operators modify domains of latent variables. SI adds a new state to a latent variable. SD does the reverse. Note that for a given LTM there can exist multiple ways to apply each search operator.

Given a data set, EAST initializes the search process with the simplest LTM, *i.e.*, the LTM that contains only one latent variable whose cardinality equals to 1. EAST then iteratively improves the model. In each search step, it applies some operators to the current model, obtains a collection of candidate models, evaluates them using the AIC score, and picks the best one to seed the next search step. The process repeats itself until the AIC score of the model ceases to increase. Note that, if EAST never improves the initial model, the final LTC reduces to NB.

In each search step, one could apply all the five operators to the current model. This could, however, produce a large number of candidate models.[1] Evaluating them could take a long time. Therefore, the search procedure in EAST is structured into three phases: *expansion*, *simplification*, and *adjustment*. In each phase, we consider only one or two operators and thus obtain much fewer candidate models. In this way, the computational effort on evaluating candidate models is significantly reduced.

In the expansion phase, we only apply NI and SI. Both operators make the current model more expressive and thus improve the first term in the AIC score. In the simplification phase, we consider only ND and SD. Both operators simplify the current model and thus improve the second term in the AIC score. In the adjustment phase, we apply NR to adjust the structure of the current model. It helps escape from local optimal models. EAST iteratively goes through these three phases and alternatively improves the two terms in the AIC score. The pseudo code of EAST is given in Algorithms 1–4.

In each search step of EAST, we generate a set of candidate models and pick the one with the lowest AIC score. This is denoted by the arg min operators in Algorithms 2–4. To evaluate the AIC score of a candidate model, one needs to run the EM algorithm [27] to compute the MLE $\boldsymbol{\theta}^\star$. EM is known to be computationally expensive. To speed up the evaluation process, we run local EM instead. The key observations are that (1) the parameters of the current model have already been optimized, and (2) a candidate model only differs from the current model in a small part. Therefore, in local EM, we fix the parameters of the unaltered part, and optimize only the parameters that are foreign to the current model. As a concrete example, suppose the current model is m_1 in Fig. 2, and we need to evaluate the candidate model m_2 which is obtained from m_1 by applying

[1] Note that, for a given LTM, there can be many ways to apply each search operator and lead to a number of different candidate models.

Algorithm 1 EAST(m, $\mathcal{D}$)

while *true* **do**
  $m_1 \leftarrow$ Expand(m, $\mathcal{D}$)
  $m_2 \leftarrow$ Adjust(m_1, $\mathcal{D}$)
  $m_3 \leftarrow$ Simplify(m_2, $\mathcal{D}$)
  if $AIC(m_3|\mathcal{D}) \geq AIC(m|\mathcal{D})$ **then**
    return m
  else
    $m \leftarrow m_3$

Algorithm 2 Expand(m, $\mathcal{D}$)

while *true* **do**
  $m_1 \leftarrow \arg\min_{m' \in NI(m) \cup SI(m)} AIC(m'|\mathcal{D})$
  if $AIC(m_1|\mathcal{D}) \geq AIC(m|\mathcal{D})$ **then**
    return m
  else
    $m \leftarrow m_1$

Algorithm 3 Adjust(m, $\mathcal{D}$)

while *true* **do**
  $m_1 \leftarrow \arg\min_{m' \in NR(m)} AIC(m'|\mathcal{D})$
  if $AIC(m_1|\mathcal{D}) \geq AIC(m|\mathcal{D})$ **then**
    return m
  else
    $m \leftarrow m_1$

Algorithm 4 Simplify(m, $\mathcal{D}$)

while *true* **do**
  $m_1 \leftarrow \arg\min_{m' \in ND(m)} AIC(m'|\mathcal{D})$
  if $AIC(m_1|\mathcal{D}) \geq AIC(m|\mathcal{D})$ **then**
    break
  else
    $m \leftarrow m_1$
while *true* **do**
  $m_1 \leftarrow \arg\min_{m' \in SD(m)} AIC(m'|\mathcal{D})$
  if $AIC(m_1|\mathcal{D}) \geq AIC(m|\mathcal{D})$ **then**
    return m
  else
    $m \leftarrow m_1$

NI. In local EM, we only optimize the CPTs $P(Y_3|Y_1)$, $P(X_1|Y_3)$, $P(X_2|Y_3)$ in m_2, while fix the other CPTs the same as those in m_1.

In our implementation, we run local EM for a predetermined number of iterations. It might not converge, but the obtained estimation is accurate enough for ranking candidate models. We then pick the best candidate model, and optimize its parameters using full EM before comparing its AIC score with that of the current model.

For more details of EAST, we refer the readers to [3].

3.3. Parameter smoothing

We estimate the class prior $P(C)$ and the LTM parameters $\boldsymbol{\theta}_c$ using maximum likelihood estimation. As noticed in previous work (*e.g.* [12]), when the size of the training data is small, this could lead to unreliable estimation and thus deficient classification accuracy. We address this issue by smoothing the parameters using Laplace correction [28]. Let N be the total number of instances in training data and N_c be the number of instance labeled as class c. Let α be a predefined smoothing factor. We smooth the class prior distribution as follows:

$$\hat{P}(C = c) = \frac{N_c + \alpha}{N + r\alpha}.$$

We also smooth the parameters for each LTM $\mathcal{M}_c$. Let $\hat{\theta}_{cijk} = \hat{P}(Z_i = j|\pi(Z_i) = k, m_c)$ be the parameter estimation produced by EM. We calculate the smoothed parameter θ^s_{cijk} for all i, j, k as

$$\theta^s_{cijk} = \frac{N_c\hat{\theta}_{cijk} + \alpha}{N_c + |Z_i|\alpha}.$$

In Section 4.3, we will empirically show that the smoothing technique leads to significant improvement in classification accuracy.

4. Empirical evaluation

In this section, we empirically evaluate LTC on an extensive collection of data. We first demonstrate the necessity of parameter smoothing and investigate the impact of the choice of the model selection criterion. We then compare LTC with a spectrum of generative classifiers, ranging from the simplest NB, to more advanced TAN and AODE, and to the most general *Bayesian network augmented naive Bayes* (BAN) [12]. We also include C4.5 decision tree [21] and latent class classifier (LCC) in the comparison as references.

4.1. Data sets

We used 37 UCI data sets [29] in our experiments, as listed in Table 2. The data are from various domains such as medical diagnosis, handwriting recognition, Biology, Chemistry, etc. They have been used in [12] and are recommended by WEKA [30] for testing new classification algorithms.

We preprocessed the data as follows. The learning algorithms of TAN and AODE proposed by Friedman et al. [12] and Webb et al. [14] do not handle missing values. Thus, we removed incomplete instances from the data sets. TAN, AODE, BAN, and LTC deal with only categorical attributes. Therefore, we discretized the data sets using the supervised discretization method proposed by Fayyad and Irani [31].

Table 2
The 37 data sets used in the experiments.

Domain	# Attributes	# Classes	Sample size
anneal	38	6	898
australian	14	2	690
autos	25	7	159
balance-scale	4	3	625
breast-cancer	9	2	277
breast-w	9	2	683
corral	6	2	128
credit-a	15	2	653
credit-g	20	2	1000
diabetes	8	2	768
flare	10	2	1066
glass	9	7	214
glass2	9	2	163
heart-c	13	5	296
heart-statlog	13	2	270
hepatitis	19	2	80
ionosphere	34	2	351
iris	4	3	150
kr-vs-kp	36	2	3196
letter	16	26	20,000
lymph	18	4	148
mofn-3-7-10	10	2	1324
mushroom	22	2	5644
pima	8	2	768
primary-tumor	17	22	132
satimage	36	6	6435
segment	19	7	2310
shuttle-small	9	7	5800
sonar	60	2	208
soybean	35	19	562
splice	61	3	3190
vehicle	18	4	846
vote	16	2	232
vowel	13	11	990
waveform-21	21	3	5000
waveform-5000	40	3	5000
zoo	17	7	101

Table 2 summarizes the characteristics of the data sets obtained after preprocessing. The number of attributes ranges from 4 to 61. The number of classes ranges from 2 to 26. The sample size ranges from 80 to 20,000.

4.2. Experimental settings

We implemented LTC in Java. The detailed settings are as follows. We ran 40 iterations of local EM to evaluate each candidate model. For the best candidate model, we ran full EM to optimize its parameters. The full EM was terminated if the improvement in loglikelihood is smaller than 0.01, or the number of iterations reaches 500. For both local and full EM, we adopted the pyramid strategy proposed by Chickering and Heckerman [32] to avoid local maxima. The numbers of starting points were set at 16 and 64, respectively.

We also implemented LCC in Java. Similar to LTC, we partitioned the data by class and learned a latent class model for each class. To learn a latent class model, we set the initial cardinality of the latent variable equal to 1, and greedily increased the cardinality until the AIC score of the model ceased to increase. The EM was tuned at the same setting as the full EM of LTC.

For all the other classification algorithms, we used the WEKA implementations in our experiments. In particular, we adopted the default settings of WEKA with the following exceptions:

- AODE: As suggested in the original paper [14], we set the frequency limit on super parents at 30.
- BAN: We set the initial models to be naive Bayes and used hill-climbing to search for good BANs with low AIC scores. The Markov blanket correction built in WEKA was conducted on the final models to ensure every attribute is in the Markov blanket of the class variable.

Given a data set, we estimated the classification accuracy of an algorithm using stratified 10-fold cross validation [33]. All the algorithms were run on the same training/test splits.

4.3. Effect of parameter smoothing

We start by investigating the effect of parameter smoothing. In the experiments, we first learned an LTC using EAST for each data set. We then smoothed the model parameters using Laplace correction as described in Section 3.3. In the following, we compare the classification accuracy of the smoothed and unsmoothed LTCs.

The second and third columns of Table 3 show the mean and the standard deviation of the classification accuracy of the two versions. For each data set, the entry with the higher accuracy is highlighted in boldface. Table 3 also reports the average accuracy over all the data sets and the number of wins, *i.e.*, the number of data sets on which one version achieves higher accuracy than the other.

By examining Table 3, we can see that parameter smoothing leads to better performance in general. The smoothed version achieves higher overall classification accuracy and wins on 30 of the 37 data sets. The unsmoothed version only wins on 3 data sets. Note that the two versions tie on the other 4 data.

We also conducted two-tailed paired t-test with $p = 0.05$ to test the significance of the difference between the two versions. As shown in the last row of Table 3, the smoothed version significantly wins the unsmoothed version on 15 data sets. Moreover, the smoothed version never significantly loses to the unsmoothed version.

In summary, the smoothing technique often leads to improvement in classification accuracy and never significantly degrades the accuracy. In the following experiments, we conduct parameter smoothing for LTC by default. We also smooth the parameters of all the other generative classifiers in the same way as LTC.

4.4. AIC versus BIC

In Section 3.1, we discussed the AIC and BIC scores for model selection. We argued that AIC is more suitable for learning LTC for classification. In this subsection, we provide empirical support for this argument.

We compare the classification accuracy of the LTCs learned using AIC and BIC. They are summarized in the fourth and fifth columns of Table 3, respectively. In general, the LTCs learned using AIC outperform those learned using BIC. The former achieve higher accuracy than the latter on 25 out of the 37 data sets. By performing two-tailed paired t-test, we also find that AIC significantly wins BIC on 9 data sets and never significantly loses to BIC.

This result justifies our choice of AIC for model selection. In the rest of this section, we only consider LTCs learned using the AIC score.

4.5. Comparison with other classifiers

We now compare LTC with NB, TAN, AODE, BAN, C4.5, and LCC. The classification accuracy of those algorithms are shown in Table 4. From this table, we can see that LTC achieves the best overall accuracy, followed by AODE, TAN, BAN, LCC, C4.5, and NB, in that order. In terms of the number of wins, LTC is also the best (14 wins), with AODE (11 wins), TAN (6 wins), and LCC (5 wins) being the three runners-up.

Table 3
The classification accuracy of various versions of LTCs. Boldface numbers denote higher accuracy.

Domain	Smoothed	Unsmoothed	AIC	BIC
anneal	**97.78 ± 1.96**	97.33 ± 1.19	97.78 ± 1.96	**98.22 ± 0.78**
australian	**85.36 ± 4.29**	84.49 ± 5.42	**85.36 ± 4.29**	85.07 ± 5.29
autos	**85.50 ± 8.44**	71.71 ± 6.69	**85.50 ± 8.44**	68.58 ± 10.11
balance-scale	70.24 ± 3.10	70.24 ± 3.10	70.24 ± 3.10	**71.35 ± 4.62**
breast-cancer	**72.87 ± 9.31**	68.93 ± 7.49	**72.87 ± 9.31**	70.75 ± 4.92
breast-w	**97.51 ± 1.54**	97.07 ± 1.95	**97.51 ± 1.54**	97.36 ± 2.04
corral	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00
credit-a	**86.38 ± 4.38**	84.24 ± 4.55	**86.38 ± 4.38**	85.62 ± 4.28
credit-g	**73.20 ± 3.97**	72.30 ± 4.62	73.20 ± 3.97	**74.60 ± 3.86**
diabetes	76.44 ± 2.44	76.44 ± 2.51	76.44 ± 2.44	**78.00 ± 5.20**
flare	**83.21 ± 2.77**	82.55 ± 2.64	**83.21 ± 2.77**	82.84 ± 2.46
glass	**76.19 ± 7.41**	74.74 ± 5.63	**76.19 ± 7.41**	75.67 ± 7.40
glass2	**85.18 ± 9.44**	84.60 ± 9.82	**85.18 ± 9.44**	82.10 ± 9.62
heart-c	**82.46 ± 4.66**	81.77 ± 3.57	**82.46 ± 4.66**	80.40 ± 4.21
heart-statlog	81.85 ± 9.63	81.85 ± 9.63	81.85 ± 9.63	81.85 ± 8.63
hepatitis	88.75 ± 12.43	**92.50 ± 10.54**	**88.75 ± 12.43**	87.50 ± 10.21
ionosphere	**94.31 ± 2.99**	90.61 ± 2.96	**94.31 ± 2.99**	93.45 ± 2.69
iris	**94.00 ± 5.84**	93.33 ± 6.29	94.00 ± 5.84	**95.33 ± 4.50**
kr-vs-kp	96.62 ± 1.19	**96.68 ± 1.13**	**96.62 ± 1.19**	95.56 ± 1.54
letter	**92.71 ± 0.47**	91.08 ± 0.64	**92.71 ± 0.47**	86.67 ± 0.51
lymph	**89.14 ± 6.65**	78.57 ± 13.57	**89.14 ± 6.65**	79.19 ± 11.95
mofn-3-7-10	**94.48 ± 2.48**	94.25 ± 2.46	**94.48 ± 2.48**	93.28 ± 2.48
mushroom	**100.00 ± 0.00**	99.96 ± 0.11	100.00 ± 0.00	100.00 ± 0.00
pima	**77.35 ± 3.92**	77.22 ± 3.83	77.35 ± 3.92	**78.13 ± 3.61**
primary-tumor	45.60 ± 11.04	**49.23 ± 7.21**	**45.60 ± 11.04**	40.22 ± 10.14
satimage	**89.57 ± 1.24**	87.55 ± 1.01	**89.57 ± 1.24**	87.96 ± 1.25
segment	**95.67 ± 1.47**	93.59 ± 1.44	**95.67 ± 1.47**	94.24 ± 0.98
shuttle-small	**99.88 ± 0.14**	99.66 ± 0.14	**99.88 ± 0.14**	99.60 ± 0.14
sonar	**82.31 ± 9.19**	81.33 ± 10.06	**82.31 ± 9.19**	81.79 ± 10.65
soybean	**93.78 ± 2.52**	87.56 ± 2.84	**93.78 ± 2.52**	81.87 ± 3.79
splice	**94.51 ± 2.07**	93.98 ± 2.39	94.51 ± 2.07	**95.74 ± 1.41**
vehicle	**74.94 ± 4.29**	72.09 ± 5.45	**74.94 ± 4.29**	72.35 ± 4.51
vote	**95.86 ± 3.38**	94.04 ± 3.88	**95.86 ± 3.38**	93.35 ± 4.07
vowel	**80.30 ± 3.02**	78.28 ± 1.92	**80.30 ± 3.02**	74.24 ± 4.13
waveform-21	**86.02 ± 1.99**	85.78 ± 1.99	86.02 ± 1.99	**86.06 ± 1.19**
waveform-5000	**86.06 ± 1.39**	86.04 ± 1.49	86.06 ± 1.39	**86.08 ± 1.59**
zoo	**94.18 ± 6.60**	88.18 ± 6.09	**94.18 ± 6.60**	82.18 ± 6.29
Mean	**86.49 ± 4.26**	84.86 ± 4.22	**86.49 ± 4.26**	84.25 ± 4.35
# Wins	**30**	3	**25**	9
# Sig. wins	**15**	0	**9**	0

We also compare LTC with the other algorithms by two-tailed paired t-test. The numbers of significant wins, ties, and significant loses are given in Table 5. It shows that LTC significantly outperforms NB (17 wins/1 loses) and C4.5 (11/2). LTC also compares favorably to TAN (8/2), AODE (5/3), BAN (5/3), and LCC (4/1).

Table 4 highlights the data sets on which LTC significantly wins and loses to each alternative. In general, the data sets that favor LTC are those containing a large number of attributes with potentially complex correlations and sufficient samples to support such correlations, *e.g.*, kr-vs-kp, letter, satimage, and waveform. In contrast, for data sets with fewer attributes or samples like credit-g and diabetes, more restricted Bayesian network classifiers such as TAN and AODE are favorable. They are sufficient to model the attribute dependencies in these cases, and are less prone to overfitting.

4.6. Running time

Besides the classification accuracy, the time efficiency is also a concern. According to Proposition 1, making prediction with LTC can be done in time linear in the number of attributes. Fig. 3 plots the average time that it takes for LTC to classify an instance in each data set. The running time ranges from 0.3 ms to 7 ms. We argue that LTC is efficient enough for practical use.

Fig. 3 also shows the classification time of TAN, AODE, and LCC. LTC is consistently slower than TAN. Depending on the number of attributes and the cardinalities of latent variables, LTC is slower than AODE on some data sets and faster on the others. In most cases, the difference is small. We also see that LTC is almost as efficient as LCC.

On the other hand, we notice that learning LTC can be time consuming. As shown in Fig. 4, the training process takes seconds to hours on average for most of the data sets, while it costs a few days for the 5 largest ones, *i.e.*, kr-vs-kp, letter, mushroom, satimage, and splice. This is far more expensive than learning other Bayesian network classifiers such as TAN and BAN (see Fig. 4). The main cause of this high complexity is the introduction of latent variables in LTC. One needs to repeatedly run the expensive EM algorithm for model selection in this case. However, the effort for learning LTC pays off. In

Table 4
The classification accuracy of various algorithms. For each data set, the best accuracy is highlighted in boldface. The bullets ● and circles ○ in each column indicate the data sets on which LTC performs significantly better and worse than the corresponding classification algorithm, respectively.

Domain	LTC	NB	TAN	AODE	BAN	C4.5	LCC
anneal	97.78 ± 1.96	96.10 ± 2.54	98.66 ± 1.15	98.33 ± 1.59	97.99 ± 1.47	98.77 ± 0.98	**98.78 ± 1.43**
australian	85.36 ± 4.29	85.51 ± 2.65	85.22 ± 5.33	**86.09 ± 3.50**	85.51 ± 4.10	85.65 ± 4.07	84.93 ± 3.82
autos	**85.50 ± 8.44**	72.88 ± 10.12 ●	79.25 ± 8.84 ●	81.08 ± 7.39	77.29 ± 6.40 ●	78.58 ± 8.54 ●	77.92 ± 13.76 ●
balance-scale	70.24 ± 3.10	70.71 ± 4.08	**71.03 ± 3.51**	69.59 ± 4.01	70.24 ± 4.44	69.59 ± 4.27	69.28 ± 4.10
breast-cancer	72.87 ± 9.31	75.41 ± 6.44	71.11 ± 5.14	**76.49 ± 7.96**	71.42 ± 6.54	74.39 ± 7.34	71.51 ± 9.51
breast-w	97.51 ± 1.54	97.51 ± 2.19	96.63 ± 2.08	97.36 ± 2.04	97.07 ± 1.95	95.76 ± 2.61	**97.52 ± 1.53**
corral	**100.00 ± 0.00**	85.96 ± 7.05 ●	99.23 ± 2.43	89.10 ± 8.98 ●	97.69 ± 3.72	94.62 ± 8.92	97.63 ± 3.82
credit-a	86.38 ± 4.38	87.29 ± 3.53	86.84 ± 3.02	**87.59 ± 3.51**	85.31 ± 3.44	86.99 ± 4.48	86.68 ± 3.30
credit-g	73.20 ± 3.97	75.80 ± 4.32 ○	74.00 ± 4.40	**77.10 ± 4.38** ○	74.90 ± 3.54 ○	72.10 ± 4.46	72.90 ± 3.90
diabetes	76.44 ± 2.44	77.87 ± 3.50	78.77 ± 3.32 ○	78.52 ± 4.11 ○	**78.91 ± 3.62** ○	78.26 ± 3.97	76.44 ± 4.29
flare	83.21 ± 2.77	80.30 ± 3.42 ●	82.84 ± 2.27	82.46 ± 2.31	82.93 ± 2.33	82.09 ± 1.80 ●	**83.31 ± 2.53**
glass	**76.19 ± 7.41**	74.37 ± 8.97	**76.19 ± 9.88**	**76.19 ± 7.41**	74.46 ± 11.28	73.94 ± 9.76	75.71 ± 8.50
glass2	**85.18 ± 9.44**	83.97 ± 8.99	**85.18 ± 9.89**	83.97 ± 9.91	**85.18 ± 9.89**	84.01 ± 7.32	**85.18 ± 9.44**
heart-c	82.46 ± 4.66	**84.11 ± 7.85**	82.80 ± 5.74	83.10 ± 7.17	83.10 ± 6.99	74.66 ± 6.49 ●	80.38 ± 3.27
heart-statlog	81.85 ± 9.63	**83.33 ± 6.36**	82.22 ± 6.94	81.85 ± 6.86	80.00 ± 7.65	81.85 ± 5.91	79.63 ± 8.24
hepatitis	88.75 ± 12.43	85.00 ± 15.37	88.75 ± 13.76	85.00 ± 12.91	87.50 ± 11.79	**90.00 ± 14.19**	86.25 ± 12.43
ionosphere	**94.31 ± 2.99**	90.60 ± 3.83 ●	93.17 ± 3.60	92.31 ± 2.34 ●	93.17 ± 4.07	89.17 ± 5.35 ●	92.31 ± 2.70
iris	94.00 ± 5.84	94.00 ± 5.84	**94.67 ± 5.26**	93.33 ± 5.44	94.00 ± 5.84	94.00 ± 4.92	94.00 ± 5.84
kr-vs-kp	96.62 ± 1.19	87.89 ± 1.81 ●	92.21 ± 2.30 ●	91.18 ± 0.83 ●	97.06 ± 0.92	**99.44 ± 0.48** ○	92.49 ± 1.72 ●
letter	**92.71 ± 0.47**	74.04 ± 1.04 ●	85.61 ± 0.63 ●	88.91 ± 0.50 ●	85.01 ± 0.84 ●	78.63 ± 0.62 ●	92.47 ± 0.65
lymph	**89.14 ± 6.65**	83.67 ± 6.91 ●	85.10 ± 7.01	85.62 ± 8.66	87.05 ± 9.99	78.33 ± 10.44 ●	82.95 ± 9.07 ●
mofn-3-7-10	94.48 ± 2.48	85.35 ± 1.53 ●	91.16 ± 1.79 ●	89.05 ± 2.53 ●	**100.00 ± 0.00** ○	**100.00 ± 0.00** ○	92.38 ± 1.85
mushroom	**100.00 ± 0.00**	97.41 ± 0.72 ●	99.81 ± 0.26 ●	**100.00 ± 0.00**	99.95 ± 0.09	**100.00 ± 0.00**	**100.00 ± 0.00**
pima	77.35 ± 3.92	78.13 ± 4.24	**78.65 ± 4.62**	**78.65 ± 3.81**	77.87 ± 4.53	78.38 ± 2.90	76.83 ± 3.53
primary-tumor	45.60 ± 11.04	**47.14 ± 11.59**	41.04 ± 12.56	46.37 ± 10.12	45.60 ± 8.64	43.24 ± 10.55	45.60 ± 11.04
satimage	**89.57 ± 1.24**	82.42 ± 1.51 ●	88.50 ± 0.89 ●	89.26 ± 0.59	87.91 ± 1.01 ●	84.37 ± 1.34 ●	89.20 ± 1.22
segment	**95.67 ± 1.47**	91.52 ± 1.60 ●	95.32 ± 1.74	95.63 ± 1.23	95.06 ± 1.80	95.32 ± 1.63	95.41 ± 1.73
shuttle-small	**99.88 ± 0.14**	99.34 ± 0.27 ●	99.81 ± 0.15	99.84 ± 0.13	99.86 ± 0.11	99.59 ± 0.19 ●	99.86 ± 0.14
sonar	82.31 ± 9.19	85.62 ± 5.41	86.60 ± 7.72	**87.07 ± 6.31**	80.29 ± 8.85	79.81 ± 8.14	84.14 ± 11.24
soybean	**93.78 ± 2.52**	91.64 ± 4.44	93.41 ± 3.38	91.99 ± 4.22	92.17 ± 4.70	91.82 ± 3.75	91.28 ± 3.70 ●
splice	94.51 ± 2.07	95.36 ± 1.00	95.30 ± 1.41	**96.21 ± 1.07** ○	94.48 ± 1.40	94.36 ± 1.58	96.11 ± 1.60 ○
vehicle	**74.94 ± 4.29**	62.65 ± 4.15 ●	73.99 ± 4.44	73.06 ± 4.65	72.94 ± 5.00	71.99 ± 3.45 ●	73.76 ± 4.64
vote	**95.86 ± 3.38**	89.91 ± 4.45 ●	94.03 ± 4.72	94.03 ± 4.07	93.81 ± 4.93	95.18 ± 4.48	94.71 ± 2.67
vowel	80.30 ± 3.02	67.07 ± 6.14 ●	**87.37 ± 2.94** ○	81.92 ± 4.11	82.22 ± 3.96	80.91 ± 2.31	79.49 ± 3.44
waveform-21	86.02 ± 1.99	81.76 ± 1.49 ●	83.10 ± 1.46 ●	**86.60 ± 1.26**	83.10 ± 1.25 ●	75.44 ± 2.10 ●	86.06 ± 1.71
waveform-5000	86.06 ± 1.39	80.74 ± 1.38 ●	82.02 ± 1.26 ●	**86.36 ± 1.65**	82.44 ± 1.21 ●	76.48 ± 1.47 ●	86.10 ± 1.79
zoo	94.18 ± 6.60	93.18 ± 7.93	95.18 ± 8.15	95.09 ± 5.18	**96.09 ± 5.05**	92.18 ± 8.94	94.18 ± 6.60
Mean	**86.49 ± 4.26**	83.12 ± 4.72	85.80 ± 4.43	85.85 ± 4.40	85.66 ± 4.41	84.32 ± 4.59	85.50 ± 4.62
# Wins	**14**	3	6	11	4	4	5

Table 5
The number of times that LTC significantly wins, ties with, and significantly loses to the other algorithms.

	NB	TAN	AODE	BAN	C4.5	LCC
# Wins	17	8	5	5	11	4
# Ties	19	27	29	29	24	32
# Loses	1	2	3	3	2	1

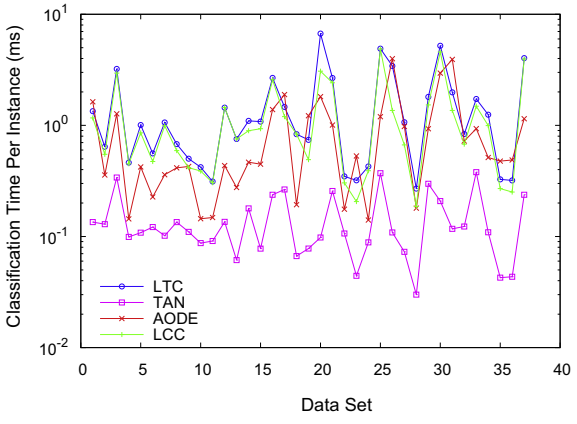


Fig. 3. The average classification time.

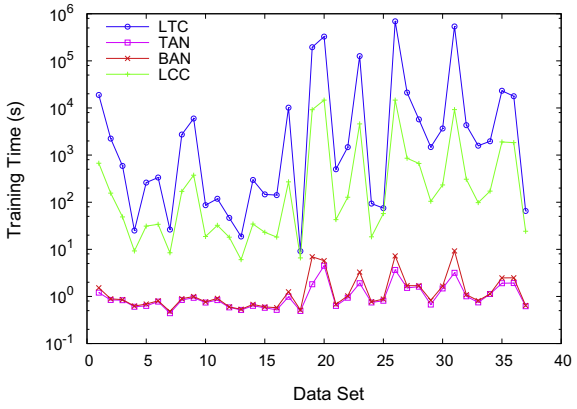


Fig. 4. The average training time.

addition to its superior classification performance, LTC can also discover interesting latent variables that give insights into problem domains, as will be demonstrated in the next section.

5. Discovery of latent structures

One advantage of LTC is that it can capture concepts underlying domains and automatically discover interesting subgroups within each class. In this section, the readers will see one such example.

The example is involved with the corral data [34]. It contains two classes *true* and *false*, and six boolean attributes A_0, A_1, B_0, B_1, *Irrelevant*, and *Correlated*. The target concept is $(A_0 \wedge A_1) \vee (B_0 \wedge B_1)$. *Irrelevant* is a random attribute which is irrelevant to the class label, and *Correlated* matches the class label 75% of the time.

We learned an LTC from the corral data and obtained two LTMs, one for each class. We denote the LTMs by $\mathcal{M}_t$ and $\mathcal{M}_f$, respectively. Their structures are shown in Fig. 5. $\mathcal{M}_t$ contains one latent variable Y_t, and $\mathcal{M}_f$ contains two latent variables Y_{f1} and Y_{f2}. All the latent variables are binary. The width of an edge denote the mutual information between the incident nodes.

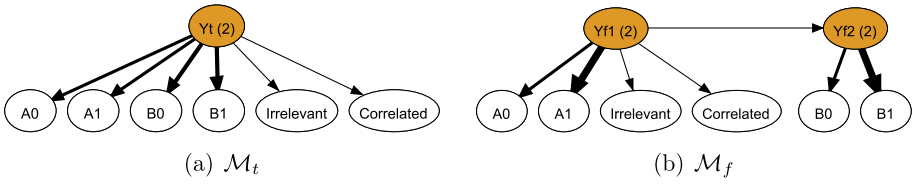
5.1. Main findings

We first observe that in both models, the four attributes A_0, A_1, B_0, and B_1 are closely correlated to their latent parents. In contrast, *Irrelevant* and *Correlated* are almost independent of their parents (notice the difference in edge widths in Fig. 5). This is interesting as both models correctly pick the four relevant attributes to the target concept.

We further studied the meanings of the latent variables and obtained more appealing findings. The latent variable Y_t in $\mathcal{M}_t$ takes two values. Therefore, Y_t represents a soft partition over the instances in the *true* class into two groups, each group corresponding to one value of Y_t. We refer to those groups as *latent groups*, and denote them by the corresponding values of Y_t. The latent variables Y_{f1} and Y_{f2} in $\mathcal{M}_f$ also take two values. Similarly, each latent variable represents a soft partition over the instances in the *false* class into two latent groups, while the two partitions are distinct from each other.

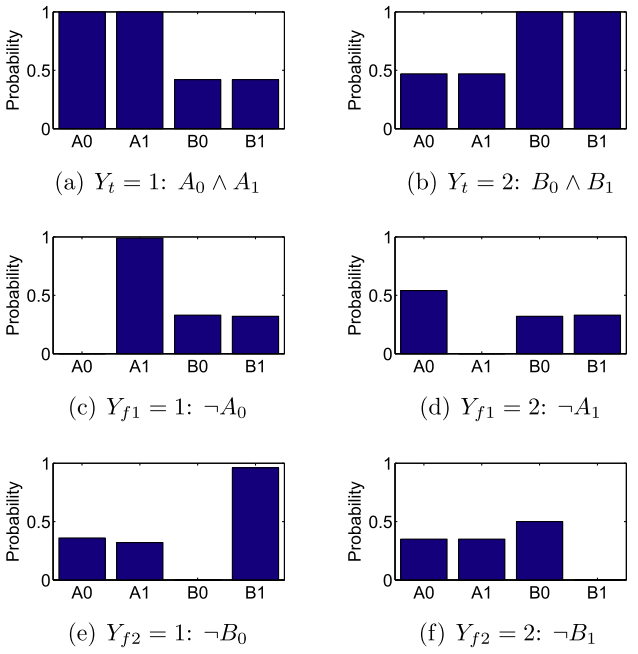
Our analysis in the next subsection will show that:

1. The latent groups $Y_t = 1$ and $Y_t = 2$ correspond to the two components of the concept, $A_0 \wedge A_1$ and $B_0 \wedge B_1$, respectively;
2. The latent groups $Y_{f1} = 1$ and $Y_{f1} = 2$ correspond to $\neg A_0$ and $\neg A_1$, while the latent groups $Y_{f2} = 1$ and $Y_{f2} = 2$ correspond to $\neg B_0$ and $\neg B_1$;
3. The latent variables Y_{f1} and Y_{f2} jointly enumerate the four cases when the target concept $(A_0 \wedge A_1) \vee (B_0 \wedge B_1)$ does not satisfy.



(a) $\mathcal{M}_t$ (b) $\mathcal{M}_f$

Fig. 5. The structures of the LTMs for corral data. The numbers in the parentheses denote the cardinalities of the latent variables. The width of an edge denote the mutual information between the incident nodes.



(a) $Y_t = 1$: $A_0 \wedge A_1$ (b) $Y_t = 2$: $B_0 \wedge B_1$

(c) $Y_{f1} = 1$: $\neg A_0$ (d) $Y_{f1} = 2$: $\neg A_1$

(e) $Y_{f2} = 1$: $\neg B_0$ (f) $Y_{f2} = 2$: $\neg B_1$

Fig. 6. The attribute distributions in each latent group and the corresponding concept.

Before diving into the details, we would like to point out that LTC successfully discovering the underlying concept and intra-class subgroups gives rise to its perfect classification result on the corral data (see Table 4). We argue that the capability of discovering such latent patterns is one reason why LTC achieves good classification accuracy.

5.2. Detailed analysis

To understand the characteristics of each latent group, we examine the conditional distribution of each attribute, *i.e.*, $P(X|Y = 1)$ and $P(X|Y = 2)$ for all $X \in \{A_0, A_1, B_0, B_1\}$ and $Y \in \{Y_t, Y_{f1}, Y_{f2}\}$. Those distributions are plotted in Fig. 6. The height of a bar indicates the corresponding probability value.

We start by the latent groups associated with Y_t. In latent group $Y_t = 1$, A_0 and A_1 always take value *true*, while B_0 and B_1 emerge at random. Clearly, this group of instances belong to class *true* because they satisfy $A_0 \wedge A_1$. In contrast, in latent group $Y_t = 2$, B_0 and B_1 always take value *true*, while A_0 and A_1 emerge at random. Clearly, this group corresponds to the concept $B_0 \wedge B_1$.

We next examine the two latent variables in $\mathcal{M}_f$. It is clear that A_0 never occurs in latent group $Y_{f1} = 1$, while A_1 never occurs in latent group $Y_{f1} = 2$. Therefore, the two latent groups correspond to $\neg A_0$ and $\neg A_1$, respectively. Y_{f1} thus reveals the two cases when $A_0 \wedge A_1$ does not satisfy. Similarly, we find that B_0 never occurs in latent group $Y_{f2} = 1$, while B_1 never occurs in latent group $Y_{f2} = 2$. Therefore, the two latent groups correspond to $\neg B_0$ and $\neg B_1$, respectively. Y_{f2} thus reveals the two cases when $B_0 \wedge B_1$ does not satisfy. Consequently, Y_{f1} and Y_{f2} jointly represent the four cases when the target concept $(A_0 \wedge A_1) \vee (B_0 \wedge B_1)$ does not satisfy.

6. Conclusions and future work

We propose a novel generative classifier, namely latent tree classifier. It builds upon the powerful yet compact representation of latent tree models, and respects the inter-class heterogeneity of the relationships among attributes. We empirically show that LTC compares favorably to NB, TAN, AODE, BAN, and C4.5 in classification accuracy. Although making prediction with LTC is slower than NB and TAN, it is as efficient as AODE and fast enough for real-world applications. We also show that LTC is superior to latent class classifier.

We demonstrate that the learned LTC can reveal underlying concepts and discover interesting subgroups within each class. As far as we know, the second feature is unique to our method. We argue that the capability of discovering such latent patterns is one reason why LTC achieves good classification performance.

We used a hill-climbing algorithm, EAST, to learn LTC. As we have shown in the experiments, EAST can be time consuming, especially for large data sets. Therefore, LTC is currently most suitable for applications which allow a long offline training phase but demand good online classification performance. On the other hand, we believe that the promising results presented in this paper warrant future research on fast learning algorithms for LTC.

Besides classification, we are also investigating the usefulness of LTM in other supervised learning problems. One ongoing research on the ranking problem. LTC can approximate the true distribution underlying data well. Therefore, we expect it to provide accurate estimation to the class posterior distribution which can be used for ranking test data. One potential application is direct marketing. Given the purchasing history of a set of customers, the task is to rank a set of new customers so that who will buy a particular product appear before who will not. Future promotions will then be send to the customers in top deciles to maximize profits. We have obtained some promising results in such applications, *e.g.*, CoIL Challenge 2000 [35]. We also believe that LTC can help marketing people discover interesting subgroups among the population.

Acknowledgments

We thank Gang Wang for his inspiring discussions. We are also grateful to the anonymous reviewers for their valuable suggestions on an earlier version of this paper. Research on this work was supported in part by MoE AcRF grant 2010-T2-2-071, China National Basic Research 973 Program under project No. 2011CB505101, and Guangzhou HKUST Fok Ying Tung Research Institute.

References

[1] Y. Wang, N.L. Zhang, T. Chen, L.K. Poon, Latent tree classifier, in: Proceedings of the 11th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU-11), pp. 410–421.
[2] N.L. Zhang, Hierarchical latent class models for cluster analysis, Journal of Machine Learning Research 5 (2004) 697–723.
[3] T. Chen, N.L. Zhang, T. Liu, K.M. Poon, Y. Wang, Model-based multidimensional clustering of categorical data, Artificial Intelligence 176 (2012) 2246–2269.
[4] Y. Wang, N.L. Zhang, T. Chen, Latent tree models and approximate inference in Bayesian networks, Journal of Artificial Intelligence Research 32 (2008) 879–900.
[5] Y. Wang, Latent Tree Models for Multivariate Density Estimation: Algorithms and Applications, Ph.D. thesis, Hong Kong University of Science & Technology, 2009.
[6] N.L. Zhang, Y. Wang, T. Chen, Discovery of latent structures: experience with the CoIL challenge 2000 data set, Journal of Systems Science and Complexity 21 (2008) 172–183.
[7] S. Harmeling, C.K.I. Williams, Greedy learning of binary latent trees, IEEE Transactions on Pattern Analysis and Machine Intelligence 33 (2011) 1087–1097.
[8] M.J. Choi, V.Y.F. Tan, A. Anandkumar, A.S. Willsky, Learning latent tree graphical models, Journal of Machine Learning Research 1 (2011) 1–48.
[9] Y.D. Rubinstein, T. Hastie, Discriminative vs informative learning, in: Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining (KDD-97), pp. 49–53.
[10] R.O. Duda, P.E. Hart, Pattern Classification and Scene Analysis, Wiley, New York, 1973.
[11] P. Domingos, M. Pazzani, On the optimality of the simple Bayesian classifier under zero-one loss, Machine Learning 29 (1997) 103–130.
[12] N. Friedman, D. Geiger, M. Goldszmidt, Bayesian network classifiers, Machine Learning 29 (1997) 131–163.
[13] C.K. Chow, C.N. Liu, Approximating discrete probability distributions with dependence trees, IEEE Transactions on Information Theory 14 (1968) 462–467.
[14] G.I. Webb, J.R. Boughton, Z. Wang, Not so naive Bayes: aggregating one-dependence estimators, Machine Learning 58 (2005) 5–24.
[15] D. Lowd, P. Domingos, Naive Bayes models for probability estimation, in: Proceedings of the 22nd International Conference on Machine Learning (ICML-05), pp. 529–536.
[16] S. Monti, G.F. Cooper, A Bayesian network classifier that combines a finite mixture model and a naive Bayes model, in: Proceedings of the 11th Annual Conference on Uncertainty in Artificial Intelligence (UAI-95), pp. 447–456.
[17] H. Langseth, T.D. Nielsen, Latent classification models, Machine Learning 59 (2005) 237–265.
[18] N.L. Zhang, T.D. Nielsen, F.V. Jensen, Latent variable discovery in classification models, Artificial Intelligence in Medicine 30 (2004) 283–299.
[19] H. Langseth, T.D. Nielsen, Classification using hierarchical naive Bayes models, Machine Learning 63 (2006) 135–159.
[20] H. Langseth, T.D. Nielsen, Latent classification models for binary data, Pattern Recognition 42 (2009) 2724–2736.
[21] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufman, 1993.
[22] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufman, 1988.
[23] H. Akaike, A new look at the statistical model identification, IEEE Transactions on Automatic Control 19 (1974) 716–723.
[24] J. Shao, An asymptotic theory for linear model selection, Statistica Sinica 7 (1997) 221–264.
[25] C.M. Hurvich, C.-L. Tsai, Regression and time series model selection in small samples, Biometrika 76 (1989) 297–307.
[26] G. Schwarz, Estimating the dimension of a model, Annals of Statistics 6 (1978) 461–464.
[27] A.P. Dempster, N.M. Laird, D.R. Rubin, Maximum likelihood from incomplete data via the EM algorithm, Journal of the Royal Statistical Society, Series B 39 (1977) 1–38.
[28] T. Niblett, Constructing decision trees in noisy domains, in: Proceedings of the 2nd European Working Session on Learning (EWSL-87), pp. 67–78.
[29] A. Frank, A. Asuncion, UCI Machine Learning Repository, University of California, School of Information and Computer Science, Irvine, CA, 2010. Available from <http://archive.ics.uci.edu/ml>.
[30] I.H. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques, Morgan Kaufman, 2005.
[31] U.M. Fayyad, K.B. Irani, Multi-interval discretization of continuous-valued attributes for classification learning, in: Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93), pp. 1022–1027.
[32] D.M. Chickering, D. Heckerman, Efficient approximations for the marginal likelihood of Bayesian networks with hidden variables, Machine Learning 29 (1997) 181–212.
[33] R. Kohavi, A study of cross-validation and bootstrap for accuracy estimation and model selection, in: Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95), pp. 1137–1145.
[34] G.H. John, R. Kohavi, K. Pfleger, Irrelevant features and the subset selection problem, in: Proceedings of the 11th International Conference on Machine Learning (ICML-94), pp. 121–129.
[35] P. van der Putten, M. de Ruiter, M. van Someren, CoIL challenge 2000: the insurance company case, LIACS Technical Report 2000-09, Sentient Machine Research, Amsterdam and Leiden Institute of Advanced Computer Science, 2000.