Mach Learn (2018) 107:1303–1331
https://doi.org/10.1007/s10994-018-5718-0

Accurate parameter estimation for Bayesian network classifiers using hierarchical Dirichlet processes

François Petitjean[1] · **Wray Buntine**[1] · **Geoffrey I. Webb**[1] · **Nayyar Zaidi**[1]

Received: 28 August 2017 / Accepted: 10 May 2018 / Published online: 22 May 2018
© The Author(s) 2018

Abstract This paper introduces a novel parameter estimation method for the probability tables of Bayesian network classifiers (BNCs), using hierarchical Dirichlet processes (HDPs). The main result of this paper is to show that improved parameter estimation allows BNCs to outperform leading learning methods such as random forest for both 0–1 loss and RMSE, albeit just on categorical datasets. As data assets become larger, entering the hyped world of "big", efficient accurate classification requires three main elements: (1) classifiers with low-bias that can capture the fine-detail of large datasets (2) out-of-core learners that can learn from data without having to hold it all in main memory and (3) models that can classify new data very efficiently. The latest BNCs satisfy these requirements. Their bias can be controlled easily by increasing the number of parents of the nodes in the graph. Their structure can be learned out of core with a limited number of passes over the data. However, as the bias is made lower to accurately model classification tasks, so is the accuracy of their parameters' estimates, as each parameter is estimated from ever decreasing quantities of data. In this paper, we introduce the use of HDPs for accurate BNC parameter estimation even with lower bias. We conduct an extensive set of experiments on 68 standard datasets and demonstrate that our resulting classifiers perform very competitively with random forest in terms of prediction, while keeping the out-of-core capability and superior classification time.

Editors: Jesse Davis, Elisa Fromont, Derek Greene, and Bjorn Bringmaan.

✉ François Petitjean
francois.petitjean@monash.edu

Wray Buntine
wray.buntine@monash.edu

Geoffrey I. Webb
geoff.webb@monash.edu

Nayyar Zaidi
nayyar.zaidi@monash.edu

1 Faculty of Information Technology, Monash University, 25 Exhibition Walk, Clayton, VIC 3800, Australia

Keywords Bayesian network · Parameter estimation · Graphical models · Dirichlet processes · Smoothing · Classification

1 Introduction

With the ever increasing availability of large datasets, Bayesian network classifiers (BNCs) show great potential because they can be learned out-of-core, i.e., without having to hold the data in main memory. This can be done in a discriminative fashion, for example, TAN (Friedman et al. 1997), kDB (Sahami 1996) and Selective kDB (SkDB) (Martínez et al. 2016) as well as generatively, using fixed-structure models such as naïve Bayes (Lewis 1998) and average n-dependence estimators—AnDE (Webb et al. 2005, 2012). In contrast, random forests (RFs) (Breiman 2001), are not easily learned out-of-core because they require either repeated sorting of the datasets or sampling. Standard implementations side-step the problem either by ensuring that the training sets for each tree of the forest is small enough to be in-core (Lyubimov and Palumbo 2016), or by relying on on-disk operations (Chen and Guestrin 2016).

Constraints on the network structure of BNCs are usually considered to be the main control on their bias-variance trade-off. If the number of parents for nodes is restricted to a relatively low number, then bias will generally be high and the variance on their estimates relatively low (we will actually show in the experiments that the variance can be high even for structures with low complexity). For large datasets, lower bias or higher complexity is preferable because it allows the models to more precisely capture fine detail in the data, translating into higher accuracy (exemplified by the success of deep networks). The number of parameters to estimate increases exponentially with the number of parents allowed for each node; thus, for larger models, accurate estimation of the parameters becomes critical.

We now turn to the aim of this current paper. One of the main issues with low-bias learners is their variance; it is logical that when increasing the number of free parameters, even with the largest possible dataset, there will be a point at which some parameters will not have sufficient examples to be learned with precision. Variance is thus not just a problem for small datasets, but can reappear when designing effective learners for large datasets because they require low bias. When the number of examples per parameter decreases, the variance increases because parameter estimation fails to derive accurate estimates. This, of course, is why maximum-likelihood estimates (MLEs) are not often used with low-bias learners unless ensembles are also involved.

Remarkably, experiments in this paper show that for networks as simple as TAN (where each node has two parents at most), which significantly underperform RFs when using Laplace smoothing, can significantly outperform RFs once more careful parameter estimation is performed. This is particularly surprising because one wouldn't expect the variance to be high for models such as TAN. This is due to the fact that the variance is not even among all combinations of feature values and can indeed be relatively high for some of them. We will see that our estimates automatically adapt to cases with high or low variance by careful use of the hierarchical Dirichlet process (HDP).

Drawing the link between BNCs and HDP Say you want to estimate the cancer rate in a population and you are only given 10 samples; you will get a very crude estimate. In effect, this happens 100's of times over at each leaf of a decision tree or clique of a Bayesian network when data is not abundant at the node. For n-gram models, where one wishes to estimate extremely low-bias categorical distributions and for which very few examples per parameter are available, MLEs have long since been abandoned in favour of sophisticated smoothing techniques such as modified Kneser-Ney (Chen and Goodman 1996). These, however, have complex back-off parameters that need to be set. For our more general and heterogeneous context of probability table estimation, there exist no techniques to set these parameters. The Hierarchical Pitman–Yor process (HPYP) is the Bayesian version of Kneser-Ney smoothing; it was introduced by Teh (2006) and uses empirical estimates for hyperparameters. This has been demonstrated to be very effective (Wood et al. 2011; Shareghi et al. 2017a). HPYP is well-suited for Zipfian contexts: where discrete variables have hundreds or more outcomes with very biased probabilities. Since we have discrete variables with mostly fewer outcomes we do not use the HPYP, and prefer the lower-variance hierarchical Dirichlet process (HDP) (Teh et al. 2006)—it is equivalent to HPYP with discount parameter fixed to 0.

In this paper, we propose to adapt the method of Teh (2006) for parameter estimation for n-gram models and apply it to parameter estimation for BNCs. Rather than the HPYP used by Teh (2006) we use the more computationally efficient HDP. In this context, the model is simpler because a HDP with a finite discrete base distribution is by definition equivalent to a Dirichlet distribution, that is HDPs become hierarchical Dirichlet distributions in our context. While conceptually simpler, we still use HDP style algorithms, albiet more recent collapsed techniques, because they are relatively efficient compared to the older Chinese restaurant style algorithms (Buntine and Mishra 2014; Lim et al. 2016).

Having shown that our approach outperforms state-of-the-art BNC parameter estimation techniques, we use RF as an exemplar of state-of-the-art machine learning because it is a widely used learning method for the types of tabular data to which our methods are suited which can be used out of the box without need for configuration. We show that our estimator allows BNCs to compete against RFs on categorical datasets. Furthermore, because our method is completely out-of-core, we demonstrate that we can obtain results on large datasets on standard computers with which RF cannot even be trained using standard packages such as Weka. Our models can also classify orders of magnitude faster than RF.

This paper is organised as follows. In Sect. 2, we review Bayesian network classifiers (BNCs). In Sect. 3 we motivate our use of hierarchical Dirichlet Processes (HDPs) for BNCs' parameter estimation. We present our method in Sect. 4 and related work in Sect. 5. We have conducted extensive experiments, reported in Sect. 6.

2 Standard Bayesian network classifiers

2.1 Notations

Let $\mathcal{D} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$ be a dataset with N objects. Each datum $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ is described over random variable $X_1, \ldots, X_n$. The following framework can be found in texts on learning Bayesian networks, such as Koller and Friedman (2009). A BN $\mathcal{B} = \langle \mathcal{G}, \Theta \rangle$, is characterised by the structure $\mathcal{G}$ (a directed acyclic graph, where each vertex i is associated to a random variable X_i), and parameters Θ, that quantifies the dependencies within the structure. The parameter object Θ, contains a set of parameters for each vertex in $\mathcal{G}$: $\theta_{x_i|\Pi_i(\mathbf{x})}$, where $\Pi_i(.)$ is a function which given the datum $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ as its input, returns the values of the attributes which are the parents of node i in structure $\mathcal{G}$. Note, each attribute is a random variable X_i and x_i represents the value of that random variable. For notational simplicity we write $\theta_{x_i|\Pi_i(\mathbf{x})}$ instead of $\theta_{X_i=x_i|\Pi_i(\mathbf{x})}$. We also use $\theta_{X_i|\Pi_i(\mathbf{x})}$ to represent the full vector of values for each x_i. A BN $\mathcal{B}$ computes the joint probability distribution as

Table 1 List of symbols used

Notation	Description
n	Number of attributes—also number of variables used to estimate the conditional probability
N	Number of data points in $\mathcal{D}$
Y	Random variable associated with class label—also X_0
y	Value taken by Y
$\|Y\|$	Number of classes
X_i	Random variable associated with attribute i
x_i	Value taken by X_i
X_c	Child variable for which we are estimating the conditional probability
$\theta_{X_c\|y,x_1,\ldots,x_n}$	Parameter vector associated with leaf node (at level $n+1$) for values $y, x_1, \ldots, x_n$
$\phi_{X_c\|y,x_1,\ldots,x_i}$	Latent prior parameter for node at level i associated with branching values $y, x_1, \ldots, x_i$
α	Concentration parameter for the Dirichlet distributions
$n_{x_c\|y,x_1,\ldots,x_n}$	Leaf-node parameter representing the number of data points with values $x_c\|y, x_1, \ldots, x_n$
$n_{x_c\|y,x_1,\ldots,x_i}$	Intermediate-node parameter representing the number of data points received from its children nodes $n_{x_c\|y,x_1,\ldots,x_{i-1}} = \sum_{x_i} t_{x_c\|y,x_1,\ldots,x_i}$
$t_{x_c\|y,x_1,\ldots,x_i}$	Latent variable representing the fraction of $n_{x_c\|y,x_1,\ldots,x_i}$ that is passed up to its parent
$n_{\cdot\|y,x_1,\ldots,x_i}$	Marginal count $n_{\cdot\|y,x_1,\ldots,x_i} = \sum_{x_c} n_{x_c\|y,x_1,\ldots,x_i}$
$t_{\cdot\|y,x_1,\ldots,x_i}$	Marginal count $t_{\cdot\|y,x_1,\ldots,x_i} = \sum_{x_c} t_{x_c\|y,x_1,\ldots,x_i}$

$$\mathrm{P}_{\mathcal{B}}(\mathbf{x}) = \prod_{i=1}^{n} \theta_{x_i|\Pi_i(\mathbf{x})}.$$

The goal of developing a BN classifier is to predict the value of an additional variable $X_0 = Y$: X_0 is the random variable associated with the class and we also denote it by Y and its values by $y \in \mathcal{Y}$. The data then takes the form $\mathcal{D} = \{(y^{(1)}, \mathbf{x}^{(1)}), \ldots, (y^{(N)}, \mathbf{x}^{(N)})\}$, the network takes an additional node and we can write:

$$\mathrm{P}_{\mathcal{B}}(y|\mathbf{x}) = \frac{\mathrm{P}_{\mathcal{B}}(y, \mathbf{x})}{\mathrm{P}_{\mathcal{B}}(\mathbf{x})} = \frac{\theta_{y|\Pi_0(\mathbf{x})} \prod_{i=1}^{n} \theta_{x_i|y,\Pi_i(\mathbf{x})}}{\sum_{y' \in \mathcal{Y}} \theta_{y'|\Pi_0(\mathbf{x})} \prod_{i=1}^{n} \theta_{x_i|y',\Pi_i(\mathbf{x})}}.$$

For simplicity, in the following, we use θ_y to denote $\theta_{y|\Pi_0(\mathbf{x})}$. Most notations are summarised in Table 1.

2.2 Structure learning for BNCs

Most approaches to learning BNCs learn the structure first and then learn the parameters as a separate step. Numerous algorithms have been developed for learning BNC network structure. The key difference that distinguishes BNC structure learning from normal BN structure learning is that the precision of the posterior estimates $\mathrm{P}_{\mathcal{B}}(y|\mathbf{x})$ matters rather than the precision of $\mathrm{P}_{\mathcal{B}}(y, \mathbf{x})$. As a result, it is usually important to ensure that all attributes in the class' Markov blanket are connected directly to the class or its children. As a consequence,

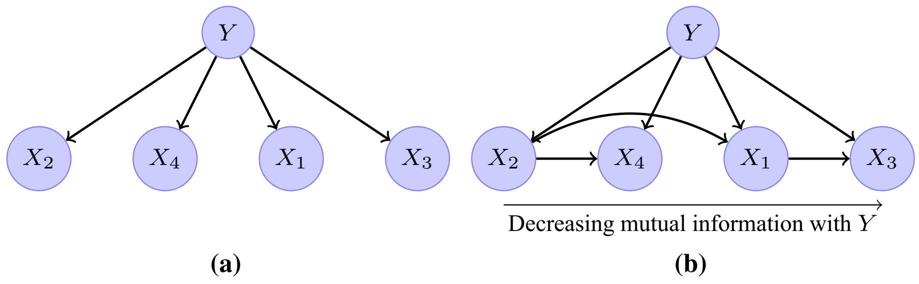


Fig. 1 Example BNC structures: **a** Naïve Bayes, **b** kDB-1

it is common for BNCs to connect all attributes to the class. Naïve Bayes [NB—see e.g. Lewis (1998)] is a popular BNC that makes the class the parent of all other attributes and includes no other edges. The resulting network is illustrated in Fig. 1a and assumes conditional independence between all attributes conditioned on the class. As a consequence, $\mathrm{P}_{\mathcal{B}}(y|\mathbf{x}) \propto \theta_y \prod_{i=1}^{n} \theta_{x_i|y}$. Tree-augmented naïve Bayes (TAN) (Friedman et al. 1997) adds a further parent to each non-class attribute, seeking to address the greatest conditional interdependencies. It uses the Chow–Liu (1968) algorithm to find the maximum-likelihood tree of dependencies among the attributes in polynomial time.

K-dependence Bayes (kDB) (Sahami 1996) allows each non-class attribute to have up to k parents, with k being a user-set value. It first sorts the attributes on mutual information with the class. Each attribute x_i is assigned the k parent attributes that maximize the conditional mutual information (CMI) with the class, $\mathrm{CMI}(y, x_i|\Pi_i(\mathbf{x}))$, out of those attributes with higher mutual information with the class. Figure 1b shows kDB-1 (for $k = 1$).

Selective kDB (SkDB) (Martínez et al. 2016) selects values $n^* \leq n$ and $k^* \leq k$ such that a kDB over the n^* attributes with highest mutual information with the class and using k^* in place of k maximizes some user selected measure of performance (in the current work, RMSE) assessed using incremental cross validation over the training data.

Other discriminative scoring schemes have been studied, see for example the work by Carvalho et al. (2011). A recent review of BNCs was written by Bielza and Larrañaga (2014).

2.3 Maximum likelihood estimates

Given data points $\mathcal{D} = \{(y^{(1)}, \mathbf{x}^{(1)}), \ldots, (y^{(N)}, \mathbf{x}^{(N)})\}$, the log-likelihood of $\mathcal{B}$ is:

$$\sum_{j=1}^{N} \log \mathrm{P}_{\mathcal{B}}\left(y^{(j)}, \mathbf{x}^{(j)}\right) = \sum_{j=1}^{N}\left(\log \theta_{y^{(j)}|\Pi_0(\mathbf{x}^{(j)})} + \sum_{i=1}^{n} \log \theta_{X_i^{(j)}|y^{(j)},\Pi_i(\mathbf{x}^{(j)})}\right), \quad (1)$$

$$\text{with} \sum_{y\in\mathcal{Y}} \theta_{y|\Pi_0(\mathbf{x})} = 1, \quad \text{and} \sum_{X_i\in\mathcal{X}_i} \theta_{X_i|y,\Pi_i(\mathbf{x})} = 1. \quad (2)$$

Maximizing the log-likelihood to optimize the parameters (Θ) yields the well-known MLEs for Bayesian networks. Most importantly, MLEs factorize into independent distributions for each node, as do most standard maximum aposterior estimates (Buntine 1996).

Theorem 1 (Wermuth and Lauritzen 1983) *Within the constraints in* Eq. 2*,* Eq. 1 *is maximized when* $\theta_{x_i|\Pi_i(\mathbf{x})}$ *corresponds to empirical estimates of probabilities from the data, that is,* $\theta_{y|\Pi_0(\mathbf{x})} = \mathrm{P}_{\mathcal{D}}(y|\Pi_0(\mathbf{x}))$ *and* $\theta_{X_i|\Pi_i(\mathbf{x})} = \mathrm{P}_{\mathcal{D}}(X_i|\Pi_i(\mathbf{x}))$.

Thus our algorithms decompose the problem into separate sub-problems, one for each $\theta_{X_i|y,\Pi_i(\mathbf{x})}$.

2.4 Efficiency of BNC learning

One often under-appreciated aspect of many BNC learning algorithms is their computational efficiency. Many BNC algorithms can be learned out-of-core, avoiding the overheads associated with retaining the training data in memory.

NB requires only a single pass through the data to learn the parameters, counting the joint frequency of each pair of a class and an attribute value. TAN and kDB require two passes through the data. The first collects the statistics required to learn the structure, and the second the joint frequency statistics required to parameterize that structure. SkDB requires three passes through the data. The first two collect the statistics required to learn the structure and parameters, as per standard kDB. The third performs an incremental cross validation to select a subset of the attributes and the $k^* \leq k$ to be used in place of k.

3 Why and how are we using HDPs?

The key contribution of this paper is to use hierarchical Dirichlet processes for each categorical distribution $\theta_{X_i|\Pi_i(\mathbf{x})}$, which yields back-off estimates that naturally smooth the empirical estimates at the leaves.

The intuition for our method is that estimation of conditional probabilities should share information with their near neighbours. Suppose you wish to estimate a conditional probability table (CPT) for $\mathrm{P}(y|x_1, x_2, x_3)$ from data where the features x_1, x_2, x_3 take on values $\{1, 2, 3, 4\}$. This CPT can be represented as a tree: the root node branches on the values of x_1 and has 4 branches, the 2nd and 3rd level nodes test x_2 and x_3 and have 4 branches. The 4th level consists of leaves and each node has a probability vector for y that we wish to estimate. The sharing intuition says that the leaf node representing $\mathrm{P}(y|x_1 = 1, x_2 = 2, x_3 = 1)$ should have similar values to the leaf for $\mathrm{P}(y|x_1 = 1, x_2 = 2, x_3 = 2)$ because they have a common parent, but should not be so similar to $\mathrm{P}(y|x_1 = 3, x_2 = 1, x_3 = 2)$, which only shares a great grandparent.

We achieve this sharing by using a hierarchical prior. So we have vectors $\mathrm{P}(Y|x_1 = 1, x_2 = 2, x_3 = u)$ (for $u = 1, 2, 3, 4$) that are generated from the same prior with a common mean probability vector, say $q(Y|x_1 = 1, x_2 = 2)$. Now $\mathrm{P}(y|x_1, x_2, x_3)$ can often be similar to $\mathrm{P}(y|x_1, x_2)$ which in turn can often be similar to $\mathrm{P}(y|x_1)$ and in turn to $\mathrm{P}(y)$. However, strictly speaking, $\mathrm{P}(y|x_1, x_2)$, $\mathrm{P}(y|x_1)$ and $\mathrm{P}(y)$ are aggregate values here derived from the underlying model which specifies $\mathrm{P}(y|x_1, x_2, x_2)$. So, to model hierarchical similarity with a HDP, instead of using the derived $\mathrm{P}(y|x_1, x_2)$, $\mathrm{P}(y|x_1)$ and $\mathrm{P}(y)$ in the hierarchical prior, we introduce some latent (hierarchical) parameters, say $q(y|x_1, x_2)$, $q(y|x_1)$ and $q(y)$. This indeed is the innovation of Teh (2006). In our case we use hierarchical Dirichlet distributions because the variables are all discrete and finite, but the algorithm relies on methods developed for a HDP (Lim et al. 2016).

3.1 Intuition developed for naïve Bayes

Imagine a simple naïve Bayes structure such as illustrated in Fig. 1a: the class is the sole parent of every node in $\mathcal{G}$. In this case, we use a (non-hierarchical) Dirichlet as suggested for Bayesian naïve Bayes (Rennie et al. 2003), for $i = 1, \ldots, n$ and all y

$$\theta_{X_i|y} \sim \text{Dir}\left(\phi_{X_i}, \alpha_i\right), \tag{3}$$

where α_i is a (Dirichlet) concentration parameter for node i (we will later develop how we *tie* these parameters in different configurations in the hierarchical case). Note the non-standard notation for the Dirichlet: for convenience we separate the vector probability ϕ_{X_i} and the concentration α_i, making it a 2-argument distribution.[1]

We can think of this model in two ways: we add a bias to the parameter estimation that encourages parameter estimates of each $\theta_{X_i|y}$ to have a common mean ϕ_{X_i} for different values of y. Alternatively, we expect $\theta_{X_i|y}$ for different values y to be similar. If they are similar, it is natural to think that they have a common mean, in this case ϕ_{X_i}. Note, however, that ϕ_{X_i} is a *prior parameter*, introduced above as $q(\cdot)$, and does not correspond to the mean estimated by marginalising with $\sum_y \hat{p}(y)\theta_{X_i|y}$ readily estimated from the data. The ϕ_{X_i} is a latent variable and a Bayesian hierarchical sampler is required to estimate it.

The hyperparameter α_i, called a concentration, controls how similar the categorical distributions $\theta_{X_i|y}$ and ϕ_{X_i} should be: if α_i is large, then each $\theta_{X_i|y}$ virtually reproduces ϕ_{X_i}; conversely, $\theta_{X_i|y}$ can vary more freely as α_i tends to 0. Estimation also involves estimating the hyperparameters, as discussed in Sect. 4.4.2.

3.2 Intuition developed for kDB-1

As described in Sect. 2.2, kDB-1 relaxes naïve Bayes' assumption about the conditional independence (given y) between the attributes by allowing one extra-parent per node as presented in Fig. 1a. The structure learning process starts from the NB structure. Then it orders the nodes by highest mutual information with the class to be ranked first, *e.g.*, $\langle x_2, x_4, x_1, x_3\rangle$ in Fig. 1a. Finally, it considers all candidate parents with higher mutual information with the class than itself (before it in the order), and chooses the one that offers the highest mutual information between the class and the child node when conditioned on it. We keep the same idea for the estimation of $\theta_{X_i|\Pi_i(\mathbf{x})}$ as in the NB case, Eq. 3, except that now X_i has 2 parents: the class and another covariate. This translates into the following, for $i = 1, \ldots, n$ and all $y, \Pi(i)$

$$\theta_{X_i|y,\Pi(i)} \sim \text{Dir}\left(\phi_{X_i|y}, \alpha_{i|y}\right), \tag{4}$$

where $\Pi(i)$ only comprises a single node for all $i > 1$ (the first node has only y as a parent). Now we could have used ϕ_{X_i} as the latent parent, so it is independent of y, but this would mean all leaves in the tree have similar probability vectors. This is a stronger statement than we need; rather we prefer adjacent nodes on the tree to be similar, not all nodes. With a hierarchical model we add another level of complexity, making the dependence on y and require a further parent above for $i = 1, \ldots, n$ and all y

$$\phi_{X_i|y} \sim \text{Dir}\left(\phi_{X_i}, \alpha_{i|1}\right). \tag{5}$$

This means that different branches in the tree can have different means, and thus the model is more flexible (and has hence relatively low bias). Our Bayesian estimation handles these additional parameters and hyperparameters and limits the effect of variance on the model.

The model naturally defines the hierarchical structure given in Fig. 2, with the formula above represented by the graphical model given in Fig. 2a.

[1] Some papers would use the notation $\text{Dir}\left(\alpha_i \phi_{X_i}\right)$ or separate the vector $(\alpha_i \phi_{X_i})$ into its $|X_i|$ arguments.

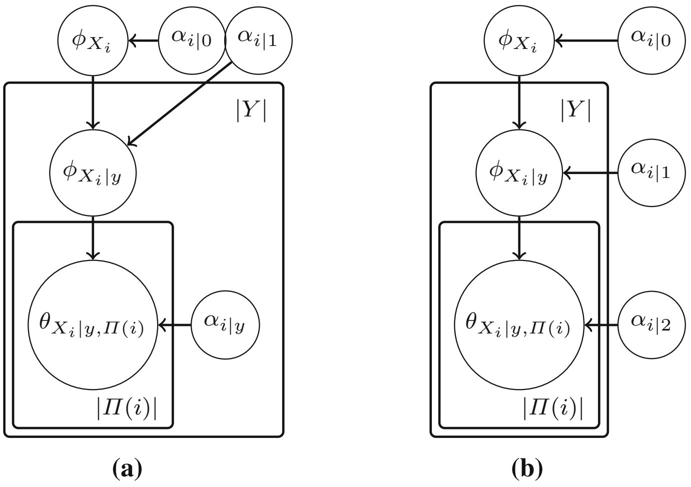


Fig. 2 Our parameter structure model for one X_i and kDB-1. **a** Tying the concentration at the parent. **b** Tying the concentration at the level. Details on tying are given in Sect. 4.4.2

3.3 Intuition—general framework

The intuition of the framework for kDB-1 naturally extends to BNs with higher numbers of parents. We structure the estimation of the conditional probability of each factor "child given parents" to have a hierarchy with as many levels as the node has parents. At each level, the hierarchy branches on the different values that the newly introduced parent takes: on the different values of y at the first level, on the different values of the first parent at the second level, etc. Once the structure is set, all we need is to have an order between the parents. For naïve Bayes, there is only one parent—y. For tree-augmented naïve Bayes (TAN), as nodes cannot have more than a single parent apart from the class, we place the class first and its other parent second. For all other structures, we place y as the first parent and then order the parents Π_i by highest mutual information between them and the child conditioned on the class. This follows both the NLP framework for n-gram estimation and kDB structure learning: position first in the hierarchy the nodes that are most likely to have an influence on the estimate. Positioning the class first allows us to pull the estimates to be most accurate in the probability space that is near $\mathrm{P}(y|\mathbf{x})$, which is our final target for classification, as we are not really interested in obtaining accurate estimates of $\mathrm{P}(X_i|y, \Pi(i))$ in parts of the probability space that are unrelated to y.

Note that the latent/prior probability vectors $\phi_{X_i|y,\Pi_i(\mathbf{x})}$ do not model observed data, as the $\theta_{X_i|y,\Pi_i(\mathbf{x})}$ do. We represent them with different symbols (ϕ versus θ) to highlight this fundamental difference.

Finally, note that in the finite discrete context, DPs are equivalent to Dirichlet distributions (Ferguson 1973), so we present our models in terms of Dirichlets, but the inference is done efficiently using a collapsed Gibbs sampler for HDPs (Du et al. 2010; Gasthaus and Teh 2010; Buntine and Mishra 2014; Lim et al. 2016). These recent collapsed samplers for the hierarchical Bayesian algorithms are considerably more efficient and accurate and so do not suffer the well-known algorithmic issues of original hierarchical Chinese restaurant algorithms (Teh et al. 2006). Note however that, unlike some applications of HDPs, there are no 'atoms' generated at the root of the HDP hierarchy because the root is just a Dirichlet, which effectively has the finite discrete set of atoms already present. The HDP formalism is used to provide an efficient algorithm as a collapsed version of a Gibbs sampler.

4 Our framework: HDPs for BNCs

This section reviews our model and sampling approach.

4.1 Model

Consider the case of estimating $\mathrm{P}(X_c|y, x_1, \dots, x_n)$ where X_c represents the child variable of which we are trying to estimate the conditional probability distribution, and $y, x_1, \dots, x_n$ are respectively used to denote the variable values $Y = y, X_1 = v_1, \dots, X_n = v_n$. The variables $X_1, \dots, X_n$ for $n \geq 0$ are ordered by mutual information with X_c as described previously. Later, we will see that X_c will represent the child variable in the Bayesian network of which we want to estimate the conditional probability distribution given its parents values $y, x_1, \dots, x_n$. We can present this as a decision tree where the root node banches on y (*i.e.*, on the values of Y), all nodes at the 1st level branch on x_1 (*i.e.*, on the values taken by X_1), at the 2nd level test x_2 and so forth. A node at the leaf (the $n+1$-th level) has the parameter vector $\theta_{X_c|y,x_1,\dots,x_n}$ for values of $y, x_1, \dots, x_n$ given by its branch on the tree. A node at the i-th level (for $i = 1, \dots, n$) has a parameter $\phi_{X_c|y,x_1,\dots,x_i}$—which is a latent prior parameter—where again values of $y, x_1, \dots, x_i$ are given by its branch on the tree. The full hierarchical model is given by

$$\begin{aligned}
\theta_{X_c|y,x_1,\dots,x_n} &\sim \mathrm{Dir}\left(\phi_{X_c|y,x_1,\dots,x_{n-1}}, \alpha_{y,x_1,\dots,x_n}\right) \\
\phi_{X_c|y,x_1,\dots,x_i} &\sim \mathrm{Dir}\left(\phi_{X_c|y,x_1,\dots,x_{i-1}}, \alpha_{y,x_1,\dots,x_i}\right) \qquad \text{for } i = 1, \dots, n-1 \\
\phi_{X_c|y} &\sim \mathrm{Dir}\left(\phi_{X_c}, \alpha_y\right) \\
\phi_{X_c} &\sim \mathrm{Dir}\left(\frac{1}{|X_c|}\mathbf{1}, \alpha_0\right).
\end{aligned}$$

Note each Dirichlet has a concentration parameter as a hyperparameter, and denote the full set of these by α_*. These are known to significantly change the characteristics of the distribution, so they must be estimated as well. We discuss below, in Sect. 4.4.2, how we can tie these hyperparameters α_* so that they are not all distinct. Experience has shown us that there should not be just one value in the entire tree, nor should there be a different value for each node.

4.2 Posterior inference

To consider how posterior inference is done with this model, first consider the simplest case of a single node with probabilities $\phi_{X_c|y}$ where a data vector $n_{X_c|y}$ is sampled with total count $n_{\cdot|y} = \sum_{x_c} n_{x_c|y}$:

$$\begin{aligned}
\phi_{X_c|y} &\sim \mathrm{Dir}\left(\phi_{X_c}, \alpha_y\right) \\
n_{X_c|y} &\sim \mathrm{multinomial}\left(\phi_{X_c|y}, n_{\cdot|y}\right).
\end{aligned}$$

For example, in Dataset 1 given later in Table 2, the values of $n_{x_1|y}$ are as follows, for each value of X_1 and Y:

$$\begin{aligned}
n_{0|0} &= 2 \\
n_{1|0} &= 0 \\
n_{0|1} &= 20 \\
n_{1|1} &= 5
\end{aligned}$$

These are contained into two vectors for $Y = 0$ and $Y = 1$:

$$n_{X_1|0} = [2, 0]$$
$$n_{X_1|1} = [20, 5]$$

The total count for the first vector are thus respectively $n_{\cdot|0} = 2$ and $n_{\cdot|1} = 25$. The marginalised likelihood for this, which marginalises out $\phi_{X_c|y}$ takes the form (Buntine 1996)

$$\mathrm{P}\left(n_{X_c|y}|\phi_{X_c}, \alpha_y, n_{\cdot|y}\right) = \binom{n_{\cdot|y}}{n_{X_c|y}} \frac{\Gamma(\alpha_y)}{\prod_{x_c} \Gamma\left(\phi_{x_c|y}\alpha_y\right)} \frac{\prod_{x_c} \Gamma\left(\phi_{x_c|y}\alpha_y + n_{x_c|y}\right)}{\Gamma\left(\alpha_y + n_{\cdot|y}\right)}. \tag{6}$$

where x_c represents the values taken by X_c. Our goal in this is to estimate the ϕ_{X_c} parameters. As it stands, this is going to be very costly because they appear in a complex form inside gamma functions, $\prod_{x_c} \frac{\Gamma(\phi_{x_c|y}\alpha_y + n_{x_c|y})}{\Gamma(\phi_{x_c|y}\alpha_y)}$. New collapsed methods developed for HDPs deal with this problem by modifying it with the introduction of new (latent) variables that make the gamma functions disappear.

While one can formalise Eq. 6 using HDPs, in this case a direct augmentation can be done using the identity (for $n \in \mathcal{N}^+$)

$$\frac{\Gamma(\alpha + n)}{\Gamma(\alpha)} = \sum_{t=1}^{n} \alpha^t S_t^n \tag{7}$$

where S_t^n is an unsigned Stirling number of the first kind. The Stirling number is a combinatoric quantity that is easily tabulated (Du et al. 2010) and simple asymptotic formula exist (Hwang 1995). This is sometimes converted into the Chinese restaurant distribution (CRD) in the form

$$\mathrm{P}(t|CRD, n, \alpha) = \frac{\Gamma(\alpha)}{\Gamma(\alpha + n)} \alpha^t S_t^n \tag{8}$$

and note the normalisation of Eq. 8 is shown by Eq. 7, where $t \in \{1, \ldots, n\}$ for $n > 0$.

To simplify Eq. 6, multiply the LHS by $\prod_{x_c} \mathrm{P}(t_{x_c|y}|CRD, n_{x_c|y}, \phi_{x_c|y}\alpha_y)$ and the RHS by the corresponding RHSs from Eq. 8. This is called an augmentation because we are introducing new latent variables $t_{x_c|y}$ for each x_c, represented in our notation as $t_{X_c|y}$. The terms in $\Gamma(\phi_{x_c|y}\alpha_y)$ etc., cancel out yielding

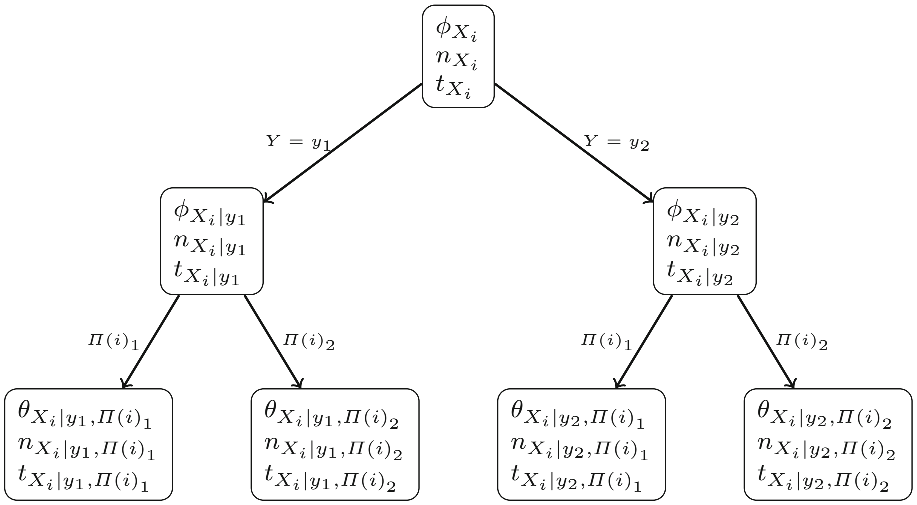
$$\begin{aligned}\mathrm{P}\left(n_{X_c|y}, t_{X_c|y}|\phi_{X_c}, \alpha_y, n_{\cdot|y}\right) &= \binom{n_{\cdot|y}}{n_{X_c|y}} \frac{\Gamma(\alpha_y)}{\Gamma\left(\alpha_y + n_{\cdot|y}\right)} \prod_{x_c} (\alpha_y \phi_{x_c})^{t_{x_c|y}} S_{t_{x_c|y}}^{n_{x_c|y}} \\ &= \binom{n_{\cdot|y}}{n_{X_c|y}} \frac{\alpha_y^{t_{\cdot|y}}}{\alpha_y^{(n_{\cdot|y})}} \prod_{x_c} \phi_{x_c}^{t_{x_c|y}} S_{t_{x_c|y}}^{n_{x_c|y}}\end{aligned} \tag{9}$$

where $\alpha^{(n)} = \alpha(\alpha + 1) \cdots (\alpha + n - 1)$ is a rising factorial.
Notice what has been done here for the current nodes X_c:

- the parent probabilities ϕ_{X_c} now appear in a simple multinomial likelihood $\prod_{x_c} \phi_{x_c}^{t_{x_c|y}}$,
- their prior complex form inside gamma functions has been eliminated,
- but at the expense of introducing new latent variables $t_{X_c|y}$.

This operation forms the basis for simplifying a full tree of such nodes recursively, presented in the next section. Equation 9 was originally developed and used in the context of the HDP, but the above alternative derivation is adequate for our purposes.

One can think of this in terms of Bayesian inference on a DAG where evidence functions are passed between nodes. Instead of passing the evidence represented by Eq. 6 from nodes



ϕ_{X_i} n_{X_i} t_{X_i}
$Y = y_1$
$Y = y_2$
$\phi_{X_i|y_1}$ $n_{X_i|y_1}$ $t_{X_i|y_1}$
$\phi_{X_i|y_2}$ $n_{X_i|y_2}$ $t_{X_i|y_2}$
$\Pi(i)_1$
$\Pi(i)_2$
$\theta_{X_i|y_1,\Pi(i)_1}$ $n_{X_i|y_1,\Pi(i)_1}$ $t_{X_i|y_1,\Pi(i)_1}$
$\theta_{X_i|y_1,\Pi(i)_2}$ $n_{X_i|y_1,\Pi(i)_2}$ $t_{X_i|y_1,\Pi(i)_2}$
$\theta_{X_i|y_2,\Pi(i)_1}$ $n_{X_i|y_2,\Pi(i)_1}$ $t_{X_i|y_2,\Pi(i)_1}$
$\theta_{X_i|y_2,\Pi(i)_2}$ $n_{X_i|y_2,\Pi(i)_2}$ $t_{X_i|y_2,\Pi(i)_2}$

Fig. 3 Context tree for our parameter structure model for kDB1 and one X_i

X_c to parent y, we pass the evidence $\prod_{x_c} \phi_{x_c}^{t_{x_c|y}}$ which is just a multinomial likelihood so it can be combined with the prior in the usual manner. So for every count $n_{x_c|y} > 0$ in the tree, one is introducing a *pseudo-count* $t_{x_c|y}$ as a latent variable, where $1 \leq t_{x_c|y} \leq n_{x_c|y}$.

How does this relate to a Chinese restaurant process (CRP)? Suppose we have a Dirichlet process with base distribution ϕ_{X_c} and we sample $n_{\cdot|y}$ data generating a Chinese restaurant configuration, where the $n_{\cdot|y}$ sample points are distributed over a number of tables. Then the $t_{x_c|y}$ variables above corresponds to the number of tables in the restaurant for data x_c, which is by definition between 1 and $n_{x_c|y}$ when $n_{x_c|y} > 0$ (Lim et al. 2016). Indeed the probability of the CRD above is the formula for a collapsed CRP (Du et al. 2010; Gasthaus and Teh 2010), where the numbers of data at each table are marginalised out, only keeping the count of tables. This represents a huge advantage computationally because one only needs to store the number of tables at each node, not the full configuration of customers at tables. This eliminates the need for dynamic memory that burdens a hierarchical CRP.

4.3 Context tree—data structure

The intuition of Eq. 9 is that each node $\theta_{X_c|y,x_1,\ldots,x_n}$ or $\phi_{X_c|y,x_1,\ldots,x_i}$ passes up some fraction of its own data as a multinomial likelihood to its parent. So the nodes will have a vector of sufficient statistics $n_{X_c|y,x_1,\ldots,x_i}$ recorded for each node. These have a virtual CRP with which we only record the number of tables $t_{X_c|y,x_1,\ldots,x_i}$, which we refer to as *pseudo-counts*. The counts $t_{X_c|y,x_1,\ldots,x_i}$ represents the fraction of $n_{X_c|y,x_1,\ldots,x_i}$ that is passed (in a multinomial likelihood) up to its parent node, as dictated by Eq. 9. An example of context tree for kDB1 is given in Fig. 3, which simply unfolds the plate notations used in Fig. 2 and adds the t and n variables.

As with hierarchical CRPs, these statistics are related for $i \geq 0$:

$$n_{x_c|y,x_1,\ldots,x_{i-1}} = \sum_{x_i} t_{x_c|y,x_1,\ldots,x_i}, \tag{10}$$

and moreover the base case $n_{x_c} = \sum_y t_{x_c|y}$. The counts $n_{x_c|y,x_1,\ldots,x_{i-1}}$ here only represent real data counts at the leaf nodes. At internal nodes, the n_* represent totals of psuedo-counts from the child nodes, as passed up by the multinomial evidence messages for the children.

The likelihood for the data with this configuration can be represented with θ and all but the root ϕ marginalised out:

$$\mathrm{P}(\mathcal{D}, n, t|\phi_{X_c}, \alpha) = \left(\prod_{x_c} \phi_{x_c}^{n_{x_c}}\right) \prod_{i=0}^{n} \left(\prod_{y,x_1,\ldots,x_i} \frac{\alpha_{y,x_1,\ldots,x_i}^{t_{\cdot|y,x_1,\ldots,x_i}}}{\left(\alpha_{y,x_1,\ldots,x_i}\right)_{\left(n_{\cdot|y,x_1,\ldots,x_i}\right)}} \prod_{x_c} S^{n_{x_c|y,x_1,\ldots,x_i}}_{t_{x_c|y,x_1,\ldots,x_i}} \right), \tag{11}$$

and the 'dot' notation is used to represent totals, so $n_{\cdot|y} = \sum_{x_c} n_{x_c|y}$. The multinomial likelihood on ϕ_{X_c} can also be marginalised out with a Dirichlet prior. Note the formula can be seen to be derived by recursive application (bottom up) of the formula in Eq. 9.

Once the parameters have been estimated (described in the next sub-section), the parameters θ can be estimated recursively using the standard hierarchical CRP estimation formula:

$$\hat{\phi}_{x_c} = \frac{n_{x_c} + \frac{1}{|X_c|}\alpha_0}{n_{\cdot} + \alpha_0} \tag{12}$$

$$\hat{\phi}_{x_c|y,x_1,\ldots,x_i} = \frac{n_{x_c|y,x_1,\ldots,x_i} + \hat{\phi}_{x_c|y,x_1,\ldots,x_{i-1}}\alpha_{y,x_1,\ldots,x_i}}{n_{\cdot|y,x_1,\ldots,x_i} + \alpha_{y,x_1,\ldots,x_i}} \tag{13}$$

$$\hat{\theta}_{x_c|y,x_1,\ldots,x_n} = \frac{n_{x_c|y,x_1,\ldots,x_n} + \hat{\phi}_{x_c|y,x_1,\ldots,x_{n-1}}\alpha_{y,x_1,\ldots,x_n}}{n_{\cdot|y,x_1,\ldots,x_n} + \alpha_{y,x_1,\ldots,x_n}} \tag{14}$$

4.4 Gibbs sampling

Note, in Eq. 11, the counts n_* are derived quantities (summed from their child pseudo-counts) and all pseudo-counts t_* are latent variables that are sampled using a Gibbs algorithm. Moreover, the parameters $\theta_{x_c|y,x_1,\ldots,x_i}$ and $\phi_{x_c|y,x_1,\ldots,x_i}$ are estimated recursively from $\phi_{x_c|y,x_1,\ldots,x_{i-1}}$ and the corresponding counts $n_{x_c|y,x_1,\ldots,x_i}$ using the standard CRP parameter estimation of Eqs. 12–14. Gibbs sampling of the pseudo-counts t_* and the concentration parameters α_* is done and the estimation of $\theta_{x_c|y,x_1,\ldots,x_i}$ is made periodically to obtain an MCMC estimate for it. This section then discusses how the Gibbs sampling of these are done.

4.4.1 Sampling pseudo-counts t_*

We use a direct strategy for sampling the t_*, sweeping through the tree sampling each pseudo-count individually using a formula derived from Eq. 11:

$$\mathrm{P}\left(t_{x_c|y,x_1,\ldots,x_i} | \mathcal{D}, n_*, t_*^{-x_c|y,x_1,\ldots,x_i}, \phi_X, \alpha\right) \propto \frac{\alpha_{y,x_1,\ldots,x_i}^{t_{x_c|y,x_1,\ldots,x_i}}}{\left(\alpha_{y,x_1,\ldots,x_{i-1}}\right)_{\left(n_{\cdot|y,x_1,\ldots,x_{i-1}}\right)}} S^{n_{x_c|y,x_1,\ldots,x_{i-1}}}_{t_{x_c|y,x_1,\ldots,x_{i-1}}} S^{n_{x_c|y,x_1,\ldots,x_i}}_{t_{x_c|y,x_1,\ldots,x_i}},$$

where $t_*^{-x_c|y,x_1,\ldots,x_i}$ represents $t_* - \{t_{x_c|y,x_1,\ldots,x_i}\}$. Note that $t_{x_c|y,x_1,\ldots,x_i}$ exists implicitly in the two sums $n_{\cdot|y,x_1,\ldots,x_{i-1}}$ and $n_{x_c|y,x_1,\ldots,x_{i-1}}$ due to Eq. 10. This sweep is made efficient because computing the Stirling numbers is a table lookup, and the Stirling numbers are shared among the different trees, so they are only calculated once for all nodes of the BNC.

The base case, $i = 0$ is different because the root parameter vector ϕ_{X_c} is marginalised using the Dirichlet integral:

$$\mathrm{P}\left(t_{x_c|y}|\mathcal{D}, n, t^{-x_c|y}, \alpha\right) \propto \frac{\Gamma\left(n_{x_c|y} + \alpha_0/|X_c|\right)}{\Gamma\left(n_{\cdot|y} + \alpha_0\right)} \alpha_y^{t_{x_c|y}} S^{n_{x_c|y}}_{t_{x_c|y}}.$$

These two sampling formula, as they stand, are also inefficient because $t_{x_c|y,x_1,\ldots,x_i}$ ranges over $1, \ldots, n_{x_c|y,x_1,\ldots,x_i}$ when $n_{x_c|y,x_1,\ldots,x_i} > 0$.

From DP theory, we know that the pseudo-counts $t_{x_c|y,x_1,\ldots,x_i}$ have a standard deviation given by $O(\log^{1/2} n_{x_c|y,x_1,\ldots,x_i})$, which is very small, thus in practice the full range is almost certainly never used. Moreover, note the mean of $t_{x_c|y,x_1,\ldots,x_i}$ changes with the concentration parameter, so in effect the sampler is coupled and large moves in the "search" may not be effective. As a safe and efficient option, we only sample the pseudo-counts within a window of ± 10 of their current value. We have tested this empirically, and due to the standard deviations, it is safer as the Monte Carlo sampling converges and smaller moves are typical.

Moreover, to initialise pseudo-counts in the Gibbs sampler, we use the expected value of the pseudo-count for a HDP given the current count and the relevant concentrations:

$$t \leftarrow \begin{cases} n & \text{if } n \leqslant 1 \\ \max(1, \lfloor \alpha\left(\psi_0(\alpha + n) - \psi_0(\alpha)\right) \rfloor & \text{if } n > 1 \end{cases} \tag{15}$$

This requires sweeping up the tree from the data at the leaves; ψ_0 represents the digamma function: $\psi_0(x) = \frac{\Gamma'(x)}{\Gamma(x)}$.

4.4.2 *Sampling and tying concentrations α_**

No proper mention has been made yet of how the concentration parameters α_* are sampled. The concentration parameters influence how similar the child probability will be to the parent probability. We know this because Dirichlet theory tells us, looking at the model in Sect. 4.1,

$$\text{Variance}\left(\theta_{X_c|y,x_1,\ldots,x_n}\right) \approx \frac{1}{\alpha_{y,x_1,\ldots,x_n}} \phi_{X_c|y,x_1,\ldots,x_{n-1}}\left(1 - \phi_{X_c|y,x_1,\ldots,x_{n-1}}\right)$$

Since we cannot be sure of how large this will be, we also sample concentration. Experience with other models using HDPs alerts us that significant improvements should be possible by judicious sampling of the concentration parameters (Buntine and Mishra 2014).

Note we expect the variance to get smaller as we go down the tree, so the concentration should be larger further down the tree.

Tying Since the number of parameters α_* is equal to the number of nodes in the tree, there are possibly too many to sample. So rather than using a separate concentration parameter $\alpha_{X_c|y,x_1,\ldots,x_i}$ for every node, we tie some, which means that we make their values equal for some different nodes. Figure 2a, b represent two different tying strategies of concentration parameters. The first one corresponds to tying the concentrations for all nodes that share a parent node: there will thus be a concentration parameter for all nodes in the tree but the lowest one. The second one has only one concentration parameter for each level of the tree. Tying is only done within one context-tree, i.e. the parameters are inferred completely independently for each conditional probability distribution $\theta_{X_i|\Pi_i(\mathbf{x})}$. Experiments on the tying of these hyperparameters are presented in Sect. 6.2.

Note that the sampling described below iterates over all the tied nodes (see j); so different tying strategies only affect the nodes that the sampler runs over.

Sampling We use an augmentation detailed in Sect. 4.3 of Lim et al. (2016). This introduces a new latent variable for each node, and then a gamma sample can be taken for the tied variable after summing the statistics across the tied nodes. The general form of the likelihood for a concentration, α, from Eq. 11 is $\prod_j \frac{\alpha^{t_j}}{\alpha^{(n_j)}}$ where j runs over the tied nodes and (n_j, t_j) are the corresponding counts at the nodes. To sample α we need to augment the denominator terms $\alpha^{(n_j)}$ because they have no match to a known distribution. This is done by adding a new term on both sides $\mathrm{P}(q|\alpha, n)$ which introduces $q_j|\alpha \sim \mathrm{Beta}(\alpha, n_j)$, then the joint posterior is derived as follows

$$\mathrm{P}(\alpha|\mathcal{D}, n, t)\mathrm{P}(q|\alpha, n) \propto \mathrm{P}(\alpha) \left(\prod_j \frac{\alpha^{t_j}}{\alpha^{(n_j)}}\right) \mathrm{P}(q|\alpha, n)$$

$$\mathrm{P}(\alpha, q|\mathcal{D}, n, t) \propto \mathrm{P}(\alpha) \prod_j \frac{\alpha^{t_j}}{\alpha^{(n_j)}} \prod_j q_j^{\alpha-1}(1-q_j)^{n_j} \frac{\Gamma(\alpha+n_j)}{\Gamma(\alpha)\Gamma(n_j)}$$

$$\propto \mathrm{P}(\alpha) \prod_j \alpha^{t_j} q_j^{\alpha-1}(1-q_j)^{n_j} \ .$$

Looking closely at this, one can see α in the augmented distribution has a gamma likelihood. Thus, using a gamma prior $\alpha \sim \mathrm{Gamma}(\nu_0, \mu_0)$ makes everything work simply. The derived sampling algorithm for α is as following:

1. sample $q_j \sim \mathrm{Beta}(\alpha, n_j)$ for all j, then
2. sample $\alpha \sim \mathrm{Gamma}\left(\nu_0 + \sum_j t_j,\ \mu_0 + \sum_j \log 1/q_j\right)$.

Note for our experiments we use an empirical Bayesian approach, so $\nu_0 = \mu_0 = 0$, and leave the issue of selecting an appropriate prior as further research.

4.5 Algorithmic description

We present here a high-level description of our sampler and associated HDP-estimates in Algorithms 1–5.

Algorithm 1 is the main algorithm: it takes as an input a dataset and returns a context tree containing our HDP estimate. It starts by creating the tree based on the dataset, i.e., creating the branches for the different cases present in the dataset, as well as storing the count statistics at the leaves. The tree is a typical hierarchical structure with a root node; nodes contain the count statistics $t_\star$ and $n_\star$, a link to its concentration α and a link to a table of children (one child per value of the branching variable at that node). It then calls the initialisation of the pseudo-counts $t_\star$ in the tree, and creates an array of concentration parameters that are tied at each level. It then proceeds with the sampling process. For each iteration of the sampling process, we first sample the $t_\star$ from the leaves up to the root, then we sample the concentration parameters (one per level except for the root node, which is not sampled). Finally, after the burn-in period has passed, we record and average the probability estimates in the tree at the current iteration. When the sampling process is terminated, these averaged estimates (stored in the tree) constitute our HDP estimates; they can be accessed by querying the tree. For brevity, we do not describe the following simple functions:

– `getNodesAtDepth`: returning all nodes at a given depth of the tree
– `initTreeWithDataset`: creating the branches of the tree down to the leaves for which data exists

- `createConcentrationArray`: creating an array of concentration objects of given size
- `recordProbabilityRecursively`: averaging the estimates for all nodes in the tree

Algorithm 1: EstimateProbHDB(data, nIters, nBurnIn)

```
   Input: $\mathcal{D}$: the dataset
   Input: nIters: number of iterations to run the sampler for
   Input: nBurnIn: number of burn-in iterations before starting to average out the $\theta$s
1  tree ← initTreeWithDataset($\mathcal{D}$)               // create tree with avail. data
2  initParametersRecursively(tree.root)                       // Algorithm 2
   // table of concentrations, one per level (Level tying)
3  cTab ← createConcentrationArray(tree.depth)
4  for depth ← 1 to tree.depth do
5      foreach node ∈ tree.getNodesAtDepth(depth) do
6          node.$\alpha$ ←cTab[depth]
7      end
8  end
9  for iter ← 1 to nIters do                                  // Gibbs sampler
       // sampling parameters for all nodes bottom-up
10     for depth ← tree.depth to 1 do
11         foreach node ∈ tree.getNodesAtDepth(depth) do
12             sampleNode(node,10,cTab[depth])                // Algorithm 3
13         end
14     end
15     for level ← 2 to tree.depth do                 // sampling concentrations
16         sampleConcentration($\alpha$,tree.getNodesAtDepth(level))    // Algorithm 5
17     end
18     if iter > nBurnIn then
19         recordProbabilityRecursively(tree.root)
20     end
21 end
22 return tree
```

Algorithm 2 describes the initialisation process of the tree's statistics, which is performed bottom-up. Starting from the leaves, we propagate the pseudo-count $t_\star$, which constitutes the $n_\star$ statistics of the parent nodes (lines 1–9). Initialisation of the pseudo-counts $t_\star$ is done following Eq. 15.

Algorithm 3 describes the sampling of the pseudo-counts $t_\star$ associated with a node, i.e., the data that should be propagated up to the parent node. Sampling happens if and only if the node is not the root node, and the $n_\star$ count statistics are strictly greater than 1.[2] The pseudo count is then sampled using the window described in Sect. 4.4.1; values either outside this window, or impossible given the pseudo-count at the parent get assigned a 0 probability of being sampled (see Algorithm 4). Valid values within the window are sampled following the Equations presented in Sect. 4.4.1.

Algorithm 4 both changes the value of a pseudo-count $t_\star$ at a node and returns its probability. As described above, it starts by checking that the new value for the pseudo-count is valid (else does not do the change and return probability 0). It then updates the pseudo-count for

[2] If $n_\star = 0$, then no data has been propagated from the children, and hence no data can be propagated up to the parent. If $n_\star = 1$, then that datapoint has to be propagated to the parent and hence needs no sampling.

Algorithm 2: initParametersRecursively(*node*)

```
Input: node: node of which we want to initialise the parameters
if node is not a leaf then                          // init. children and collect stats
    foreach child ∈ node.children do
        initParametersRecursively(child)
        for $k \leftarrow 1$ to $|X_c|$ do
            node.$n[k]$ ←node.$n[k]$+child.$t[k]$
            node.$n$ ←node.$n$+node.$n[k]$                              // marginal
        end
    end
end
if node is root then
    $\forall k$, node.$t[k] \leftarrow \min(1,$ node.$n[k])$
else
    for $k \leftarrow 1$ to $|X_c|$ do
        if node.$n[k] \leqslant 1$ then
            node.$t[k] \leftarrow$ node.$n[k]$
        else
            node.$t[k] \leftarrow \max(1, \lfloor$node.$\alpha\ (\psi_0($node.$\alpha +$ node.$n) - \psi_0($node.$\alpha))\rfloor)$
        end
    end
end
node.$t \leftarrow \sum_k$ node.$t[k]$                                     // marginal
```

Algorithm 3: sampleNode(*node*, w, α)

```
Input: node: node of which we want to sample the parameters
Input: $w$: window for sampling
Input: $\alpha$: concentration to assign to node
if node is root then
    $\forall k$, node.$t[k] \leftarrow \min(1,$ node.$n[k])$                    // no sampling
else
    node.$\alpha \leftarrow \alpha$                      // assign concentration to node
    for $k \leftarrow 1 \cdots |X_c|$ do
        if node.$n[k] \leqslant 1$ then
            node.$t[k] \leftarrow$ node.$n[k]$                                 // value fixed
        else
            minTk $\leftarrow \max(1,$ node.$t[k] - w)$
            maxTk $\leftarrow \min($node.$t[k] + w,$ node.$n[k])$
            // Constructing a vector to sample node.$t[k]$ from
            $\mathbf{v} \leftarrow \mathbf{0}$                                // length is (node.$n[k]+1$)
            for $t \leftarrow$ minTk $\cdots$ maxTk do
                $\mathbf{v}_t \leftarrow$ changeTkAndGetProbability(node, $k$, $t$)       // Algorithm 4
            end
            $\forall t, \mathbf{v}_t \leftarrow \frac{\mathbf{v}_t}{\sum_t \mathbf{v}_t}$                              // Normalize vector
            $t \sim$ multinomial $(\mathbf{v})$
            changeTkAndGetProbability(node, $k$, $t$)                      // Algorithm 4
        end
    end
end
```

that node, and the count statistic $n_\star$ at the parent. It finally returns the probability as described in Sect. 4.4.1.

Algorithm 4: changeTkAndGetProbability(node,k,newValue)

```
Input: node: node of which we want to sample the parameters
Input: $k$: index of the value we want to change in $t$
Input: newValue: value to replace $t_k$ by, if possible
1  inc $\leftarrow$ newValue $-$ node.$t[k]$
2  if inc$< 0$ then                                    // check if valid for parent
3      if node is not root and (node.parent.$n[k]$+inc) $<$ node.parent.$t[k]$ then
4          return 0
5      end
6  end
7  node.$t[k] \leftarrow$ node.$t[k]$ + inc
8  node.$t \leftarrow$ node.$t$ + inc                        // marginal
9  if node is not root then                  // update statistics at the parent
10     node.parent.$n[k] \leftarrow$ node.parent.$n[k]$ + inc
11     node.parent.$n \leftarrow$ node.parent.$n$ + inc               // marginal
12 end
13 return $\frac{\text{node.}\alpha^{\text{node.}t[k]} \cdot S^{\text{node.parent.}n[k]}_{\text{node.parent.}t[k]} \cdot S^{\text{node.}n[k]}_{\text{node.}t[k]}}{rising\_factorial(\text{node.parent.}\alpha, \text{node.parent.}n[k])}$
```

Finally, Algorithm 5 describes a simple sampling of the concentration parameters in the tree, assuming that tying is done using the `Level` strategy. As described in Sect. 4.4.2, tying requires iterating through the $t_\star$ and $n_\star$ of the 'tied' nodes. For all the 'tied' nodes, it thus performs a change of variable to q and then samples the new concentration. Other tying strategies are given in the source-code function `Concentration.java:sample()` (see beginning of Sect. 6.1 for link to source code).

Algorithm 5: sampleConcentration(α, nodes)

```
Input: $\alpha$: concentration to sample
Input: nodes: nodes sharing this concentration parameter (tying)
1 rate $\leftarrow 0$
2 foreach node $\in$ nodes do
3     $q \sim$ Beta($\alpha$, node.$n$)             // change of variable, sample $q$
4     rate $\leftarrow$ rate $- \log(q)$
5 end
6 $\alpha \sim$ Gamma $\left(\sum_{n \in nodes} n.t,\ \text{rate}\right)$                      // sample $\alpha$
7 foreach node $\in$ nodes do                       // assign new $\alpha$ to nodes
8     node.$\alpha \leftarrow \alpha$
9 end
```

4.6 Worked example

We have now fully described our HDP-based estimates. In this section, we draw all the theory together and show how our method applies to two simple datasets, highlighted in Table 2. Both datasets have two binary variables X_1 and Y, and a simple naïve Bayes structure, *i.e.*, we focus on the estimation of $\mathrm{P}(X_1|Y)$. Although this simple structure does not give full justice

Table 2 Example datasets with associated estimates

Dataset	Value for Y	Frequency $n_{X_1\|y}$	$\hat{p}(X_1\|Y)$		
			MLE	m-estimate ($m = 1$)	**HDP**
1	0	[2, 0]	[1.00, 0.00]	[0.83, 0.17]	[0.89, 0.11]
	1	[20, 5]	[0.80, 0.20]	[0.79, 0.21]	[0.79, 0.20]
2	0	[2, 0]	[1.00, 0.00]	[0.83, 0.17]	[0.86, 0.14]
	1	[4, 9]	[0.31, 0.69]	[0.32, 0.68]	[0.34, 0.66]

to our estimates for deeper hierarchies, we feel that such an example helps with understanding the different components of our method.

Our aim is to highlight how information is shared between $P(X_1|Y = 0)$ and $P(X_1|Y = 1)$ through the marginal (mean) probability $P(X_1)$. Let us describe the two datasets given in Table 2: Dataset #1 has $P(X_1|Y = 0) \approx P(X_1|Y = 1)$ – but with only little data available to estimate $P(X_1|Y = 0)$ – while Dataset #2 has $P(X_1|Y = 0) \not\approx P(X_1|Y = 1)$.

Let us start by the analysis of the cases with $Y = 0$ compared for the two datasets, cases for which the data available is identical. The first thing to observe is that, as the frequency is the same for both datasets for the cases with $Y = 0$, so are the MLEs and m-estimates,[3] respectively. MLEs and m-estimates are agnostic of the marginal; m-estimates only pull the estimates toward a uniform prior. Second, we can observe that our HDP estimates for Dataset #1 are closer to the MLEs than to the m-estimates. This is because the data available for $Y = 1$ 'corroborates' the fact that $P(X_1 = 0|Y)$ is much greater than $P(X_1 = 1|Y)$. For Dataset #2 where the two cases for Y differ, we can see that our estimate for $Y = 0$ is closer to the m-estimate than it was for Dataset #1 although the frequencies are the same; this is because now the data available for $Y = 1$ does not support the hypothesis that the marginal $P(X_1)$ is helpful to estimate $P(X_1|Y = 0)$ while having little data available. Finally, we can see that our HDP estimate for $P(X_1|Y = 1)$ in Dataset #2 goes even further than the m-estimate and pulls the estimate even closer to a uniform probability. This is again here because of the data for $Y = 0$.

5 Related work

Extensive discussions of methods for DP and PYP hierarchies are presented by Gasthaus and Teh (2010) and Lim et al. (2016). Standard Chinese restaurant process (CRP) samplers (Teh et al. 2006) use dynamic memory so are computationally demanding, and not being collapsed also makes them considerably slower. Lim et al. (2016) deal with the case where the counts at the leaves of the tree are latent, and thus are not applicable to our context. The direct samplers of Du et al. (2010), which are also *collapsed* CRP samplers, are more efficient than CRP samplers and those of Lim et al. (2016) in the current context. Gasthaus and Teh (2010) dealt with a PYP where the discount parameters change frequently so direct samplers were inefficient because the cache of Stirling numbers needed constant recomputation. On-the-fly samplers have also been developed by Shareghi et al. (2017b) for PYP hierarchies, making it possible to use PYP for deep trees and large dataset sizes. This however does not change

[3] More information about m-estimates is given in Sect. 6.1.

the issue of constant recomputation of Stirling numbers, which is why initialisations based on modified Kneser-Ney have been developed by Shareghi et al. (2016).

The use of DP and PYP hierarchies for regression and clustering—as opposed to classification in our case—has been studied by Nguyen et al. (2015) and Huynh et al. (2016), respectively.

Related work for BNCs was discussed in Sect. 2.2. There are other methods for improving BNCs. A simple back-off strategy, backing off to the root, is proposed by Friedman et al. (1997). Moreover, for some simple classes of networks, such as TAN, a disciminative generalisation of logistic regression can be used because the optimisation surface is convex (Roos et al. 2005; Zaidi et al. 2017). Neither techniques are applicable to the more complex BNCs we consider.

Bayesian model averaging methods are common for Bayesian network learning (Friedman and Koller 2003). Average n-dependence estimators—AnDE (Webb et al. 2005, 2012), another ensemble method, is competitive for smaller data sets but cannot compete against SkDB for larger data sets (Martínez et al. 2016).

Either way, these invariably use the same Laplacian prior as the m-estimates reported here in Sect. 6.

6 Experiments

The aim of this section is to assess our HDP-based estimates for Bayesian network classifiers (BNCs). In Sect. 6.1, we give the general settings that are necessary to understand and reproduce our experiments. Then, in Sect. 6.2, we start by studying how to parameterize our method: i.e., by studying the influence of the number of iterations and the tying strategy used. In Sect. 6.3, we demonstrate the superiority of our estimates over the state of the art across 8 different BNC structures. Finally, having obtained significant improvements over the state-of-the-art, we then turn to comparing the best-performing configuration (TAN and SkDB with HDP estimates) with random forest (RF) in Sect. 6.4. We show that our estimate allows even models as simple as TAN to significantly outperform RF (with statistical significance), while standard approaches to parameter estimation are beaten by RF. We conclude the experiments with a demonstration of our system's out-of-core capability and show results obtained on the Splice dataset with 50 million training examples, a quantity that RF cannot handle on most machines.

6.1 Experimental design and setting

Design All experiments are carried out on a total of 68 datasets from the UCI archive (Lichman 2013); 38 datasets with less than 1000 instances, 23 datasets with instances between 1000 and 10,000, and 7 datasets with more than 10,000 instances. The list and description of the datasets is given in Table 3 at the end of this paper. For all methods, numeric attributes are discretized by using the minimum description length (MDL) discretization method (Fayyad and Irani 1992). A missing value is treated as a separate attribute value and taken into account exactly like other values. Each algorithm is tested on each dataset using twofold cross validation repeated 5 times. We assess the results by reporting 0–1 Loss and RMSE, and report Win–Draw–Loss (W–D–L) results when comparing the 0–1 Loss and RMSE of two models. A two-tail binomial sign test is used to determine the significance of the results, using $p \leq 0.05$.

Table 3 Datasets

Domain	Case	Att	Class	Domain	Case	Att	Class
Connect-4Opening	67,557	43	3	PimaIndiansDiabetes	768	9	2
Statlog(Shuttle)	58,000	10	7	BreastCancer(Wisconsin)	699	10	2
Adult	48,842	15	2	CreditScreening	690	16	2
LetterRecognition	20,000	17	26	BalanceScale	625	5	3
MAGICGammaTelescope	19,020	11	2	Syncon	600	61	6
Nursery	12,960	9	5	Chess	551	40	2
Sign	12,546	9	3	Cylinder	540	40	2
PenDigits	10,992	17	10	Musk1	476	167	2
Thyroid	9169	30	20	HouseVotes84	435	17	2
Mushrooms	8124	23	2	HorseColic	368	22	2
Musk2	6598	167	2	Dermatology	366	35	6
Satellite	6435	37	6	Ionosphere	351	35	2
OpticalDigits	5620	49	10	LiverDisorders(Bupa)	345	7	2
PageBlocksClassification	5473	11	5	PrimaryTumor	339	18	22
Wall-following	5456	25	4	Haberman'sSurvival	306	4	2
Nettalk(Phoneme)	5438	8	52	HeartDisease(Cleveland)	303	14	2
Waveform-5000	5000	41	3	Hungarian	294	14	2
Spambase	4601	58	2	Audiology	226	70	24
Abalone	4177	9	3	New-Thyroid	215	6	3
Hypothyroid(Garavan)	3772	30	4	GlassIdentification	214	10	3
Sick-euthyroid	3772	30	2	SonarClassification	208	61	2
King-rook-vs-king-pawn	3196	37	2	AutoImports	205	26	7
Splice-junctionGeneSequences	3190	62	3	WineRecognition	178	14	3
Segment	2310	20	7	Hepatitis	155	20	2
CarEvaluation	1728	8	4	TeachingAssistantEvaluation	151	6	3
Volcanoes	1520	4	4	IrisClassification	150	5	3
Yeast	1484	9	10	Lymphography	148	19	4
ContraceptiveMethodChoice	1473	10	3	Echocardiogram	131	7	2
German	1000	21	2	PromoterGeneSequences	106	58	2
LED	1000	8	10	Zoo	101	17	7
Vowel	990	14	11	PostoperativePatient	90	9	3
Tic-Tac-ToeEndgame	958	10	2	LaborNegotiations	57	17	2
Annealing	898	39	6	LungCancer	32	57	3
Vehicle	846	19	4	Contact-lenses	24	5	3

Note the RMSE is related to the Brier score, which is a proper scoring rule for classifiers and thus generally preferable to error, especially in the context of unequally occurring classes or unequal costs. It measures how well calibrated the probability estimates are. We use it because we suspected that our methods could improve probability estimates but not necessarily errors.

Software To ensure reproducibility of our work and allow other researchers to easily build on our research, we have made our source code for HDP parameter estimation available on Github.

Compared methods We assess our estimates for 8 BNC structures with growing complexity. Our BNC structures are: naïve Bayes (NB), tree-augmented naïve Bayes (TAN) (Friedman et al. 1997), k-dependence Bayesian network (kDB) (Sahami 1996) with $k = 1$ to 5 and selective kDB (SkDB) (Martínez et al. 2016) with maximum k set to 5 also.[4] When comparing to random forest (RF), we use the Weka default parameterization, i.e., selecting $\log_2(n) + 1$ attributes in each tree,[5] no minimum leaf size and using 100 decision trees in this work.

For BNCs, we compare our HDP estimates to so-called m-estimates[6] (Mitchell 1997) as follows:

$$\hat{p}\,(x_i|\Pi(i)) = \frac{counts\,(x_i, \Pi(i)) + \frac{m}{|X_i|}}{counts\,(\Pi(i)) + m} \tag{16}$$

where $\Pi(i)$ are the parent-values of X_i. The value of m is set by cross-validation on a holdout set of size $\min(N/10, 5000)$ among with $m \in \{0, 0.05, 0.2, 1, 5, 20\}$.

Count statistics are stored in a prefix tree; for m-estimates, if zero counts are found, we back off as many levels in the tree as necessary to find at least one count. For instance, if $counts(x_4, x_0, x_3)$ is equal to zero, then $\hat{p}(x_4|x_0)$ is considered instead of $\hat{p}(x_4|x_0, x_3)$. Note that not using this strategy significantly degrades the performance of BNCs when using m-estimates (for our HDP estimates, the intermediate nodes ϕ are considered latent and thus inferred directly during sampling).

6.2 Tying and number of iterations

Before proceeding with the comparison of our method to the state of the art, it is important to study two elements: (1) for how many iterations to run the sampler and (2) how to tie the concentration parameters. These two elements are directly related because the less tying, the more parameters to infer, which means that we expect to have to run the sampler for more iterations.

We consider three different tying strategies:

1. Same Parent (SP): children of each node share the same parameter—illustrated in Fig. 2a.
2. Level (L): we use one parameter for each level of the tree—illustrated in Fig. 2b.
3. Single (S): all parameters tied together.

Number of iterations Asymptotically, the accuracy of the estimates improves as we increase the number of iterations. The question is how quickly they asymptote. We thus studied the performance of our two flagship classifiers—TAN and SkDB—on all datasets as we increase the number of iterations from 500 to 50,000. For each combination of classifier × tying strategy, we assess the win–loss profile for x iterations versus 50,000. The resulting win–loss plot in Fig. 4 shows that across all tying strategies and models, running our sampler for 50,000 iterations is significantly better than with fewer iterations. Even for models as simple as TAN with a Single concentration parameter, running the sampler for 5000 iterations wins 13 times and loses 42 times as compared to running it for 50,000 iterations. Unless specified otherwise, we thus run the sampler for 50,000 iterations. We surmise that even more iterations could further improve accuracy but leave this for future research.

[4] We do not consider higher values of k, because (1) for kDB we will see in Sect. 6.3 that the superiority of our HDP estimates is statistically significant further increases with k; (2) for SkDB, 95% of the experiments see it choose a structure with $k < 5$, differences with higher k would thus be minimal.

[5] Selecting $\sqrt{n}$ attributes produces similar results and conclusion, so the results are left out of this paper for concision.

[6] Also known as Schurmann–Grassberger's Law when $m = 1$, which is a particular case of Lidstone's law (Lidstone 1920; Hardy 1920) with $\lambda = \frac{1}{|X_i|}$, also based on a Dirichlet prior.

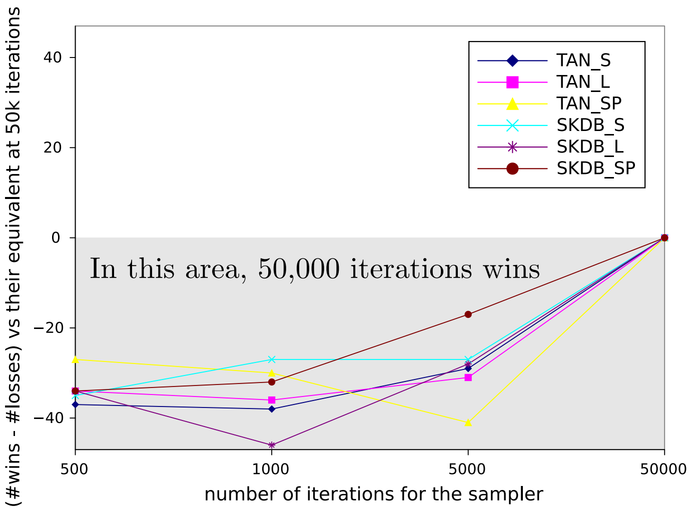


Fig. 4 Win/loss plot on RMSE for each combination of (flagship classifier) × (tying strategy). Comparison is for running each combination for x iterations versus 50,000 and include **S**ingle, **L**evel and **S**ame**P**arent

Tying strategy Having seen that 50,000 iterations seems important regardless of the tying strategy, we here show that tying per Level seems to be the best default strategy. It is important to note that we do not intend to give a definitive answer valid for all domains here, but are simply giving a reasonable 'default' parameterization. The Level strategy was illustrated for kDB-1 in Fig. 2b. To illustrate this we compare TAN and SkDB parameterized with the *same parent* (SP) and *single* (S) strategies versus using the *level* (L) tying strategy across different numbers of iterations. Figure 5 gives the win-loss plot. We see that L provides a uniformly good solution providing both the best results with 50,000 iterations but also providing solid performances as early as 500 iterations. It is worth noting that for TAN, the L and S strategies are very similar, only differing by one concentration parameter. The SP strategy seems to clearly underperform L, all the more when the complexity of the model increases, which makes sense given that the number of concentration parameters to estimate increases exponentially with the depth of the prefix tree, which is mostly controlled by the number of parents for each node i. It is possible that for large amounts of data, the SP strategy would offer a better bias/variance tradeoff but such a study falls out of the scope of this paper. We thus use L as a tying strategy for the remainder of this paper.

6.3 HDP versus m-estimates for Bayes network classifiers

So far, we have only assessed the relative performance of HDP estimates with different parameterizations. Having settled on 50,000 iterations and per Level tying, we now turn to the full comparison with the state-of-the-art in smoothing Bayesian network classifiers: using m-estimates with the value of m cross-validated on a holdout set. We also remind the reader that, to provide the best competitor, we also added the back-off strategy described above, without which m-estimates cannot compete at all.

We report in Table 4 the win–draw–loss of our HDP estimates versus m-estimates across 8 different BNCs from naïve Bayes and TAN to kDB with $1 \leqslant k \leqslant 5$ and SkDB. It is clear from this table that our HDP estimates are far superior to m-estimates. It is even quite surprising

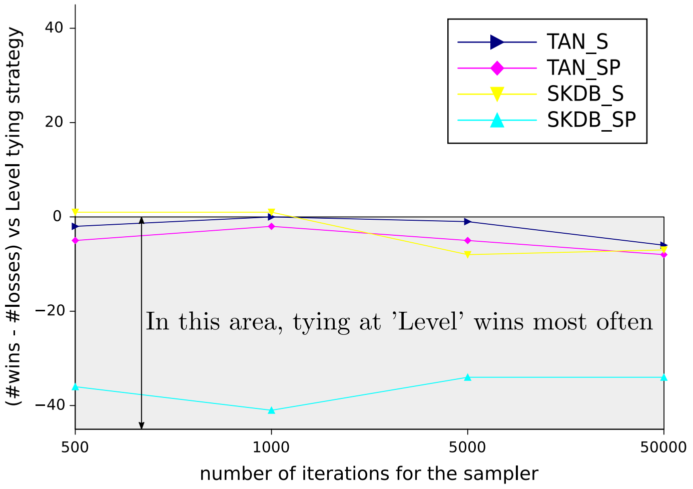


Fig. 5 Win/loss plot of each combination of (flagship classifier) × (S or SP tying strategy) versus tying at level (L)

Table 4 Win/Draw/Loss for 8 BNCs for our HDP estimate versus m-estimate

Classifier	Win–draw–loss for HDP versus m-estimate	
	0/1-loss	RMSE
Naive Bayes	**41–4–23**	40–0–28
TAN	**45–4–19**	**52–1–15**
kDB-1	**45–4–19**	**50–1–17**
kDB-2	**54–2–12**	**54–0–14**
kDB-3	**52–4–12**	**53–2–13**
kDB-4	**56–4–08**	**56–0–12**
kDB-5	**60–4–04**	**60–2–06**
SkDB	**45–4–19**	**54–0–14**

Stat. sig. ($p < 0.05$) results are depicted in boldface

to see our estimates outperform m-estimates with models as simple as Naïve Bayes, where our hierarchy only has one single level. Moreover, as the model complexity increases (the maximum number of parents for each node), this difference increases. The scatter-plot for kDB-5 HDP versus m-estimate is given in Fig. 6a and shows again the same trend with HDP significantly outperforming m-estimates. As usual when dealing with a broad range of datasets, there are a few points for which HDP loses. Interestingly, the most important loss is for the `Cylinder-Bands` dataset, which contain only 540 samples, and thus for which we would have expected that smoothing would be important; detailed inspection of this dataset show that the 540 cases seem to be relatively similar to each other (in which case the cross-validation used for the m-estimates helped discover this).

It is also interesting to study the capacity of HDP to prevent overfitting as compared to the m-estimate (with m cross-validated). In Fig. 6b, we report for m-estimates the win–loss plot for kDB-5 compared to kDBs with increasing complexity from 0 (kDB-0 is NB) to 4. Given that kDB-5 has generally lower bias than kDB $\forall k \leqslant 4$, we can typically attribute its losses to overfitting. Starting with the bottom line, which represents the behaviour of using m-estimates, we can see that kDB-5 generally loses to lower complexity kDBs. The maximum

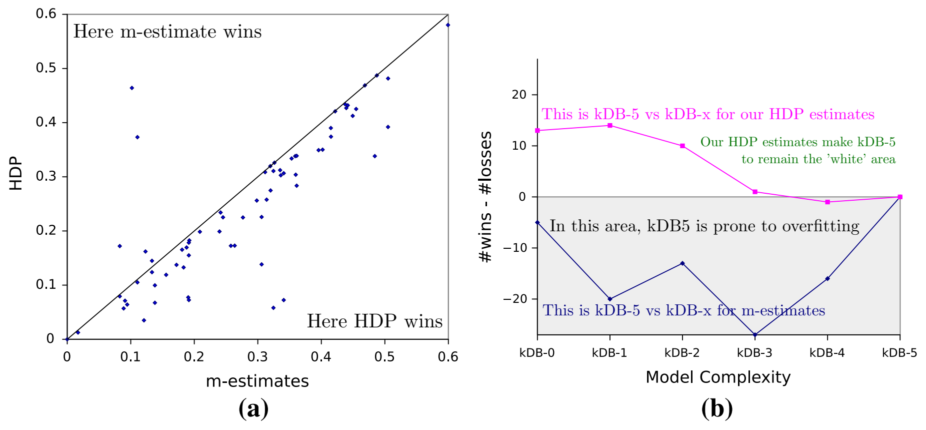


Fig. 6 **a** Scatter plot on RMSE for kDB-5 for HDP versus m-estimate. **b** Win/loss plot of kDB-5 versus kDB-x for m-estimates versus our HDP ones

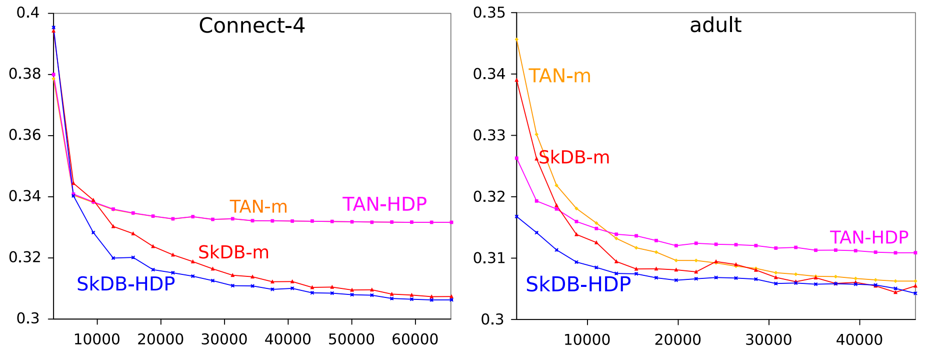


Fig. 7 Learning curves on RMSE for HDP and m-estimate. The x-axis is dataset size, the y-axis is RMSE

difference is with kDB-3 which seems to globally have a nice bias/variance tradeoff on this collection of datasets.

Conversely, we can see that HDP estimates (top-curve in Fig. 6b) allows us to nicely control for overfitting. What happens is that we make the most of the low-biased structure offered by kDB, while not being overly prone to overfitting. In some sense, our hierarchical process makes it possible to pull the probability estimates towards higher-level nodes for which we have more data, and this automatically depending on the dataset. It seems that it makes it possible to be less strict about the structure and to be powerful at controlling for the variance. In fact, controlling for overfitting is what selective kDB (SkDB) tries to achieve; in our experiments, kDB5-HDP has a slight edge over SkDB5-HDP with a win–draw–loss of 33–5–30 on RMSE. Nevertheless, it remains that HDP largely outperforms m-estimates with a win–loss—for SkDB—of 60 to 8.

Finally, we present some learning curves for TAN and SkDB on a some larger datasets in Fig. 7. Each point corresponds the mean RMSE for quantity of data x over 10 runs. Globally, we can see that our HDP estimates seem to 'learn' faster, i.e. overfit less. For the `connect-4` dataset, SkDB-HDP dominates all the way through with the difference in RMSE getting smaller as the quantity of data increases. For `adult`, we can observe the

Table 5 Win/Draw/Loss m-estimates and our HDP estimates, as compared with random forest. We use our 2 flagship classifiers TAN and SkDB

Compared classifiers	Win–draw–loss	
	0/1-loss	RMSE
TAN-*m* versus RF	26–3–39	**25–0–43**
SkDB-*m* versus RF	27–3–38	29–1–38
TAN-HDP versus RF	**42–3–23**	**42–0–26**
SkDB-HDP versus RF	35–3–30	**44–0–24**

Stat. sig. results ($p < 0.05$) are depicted in boldface

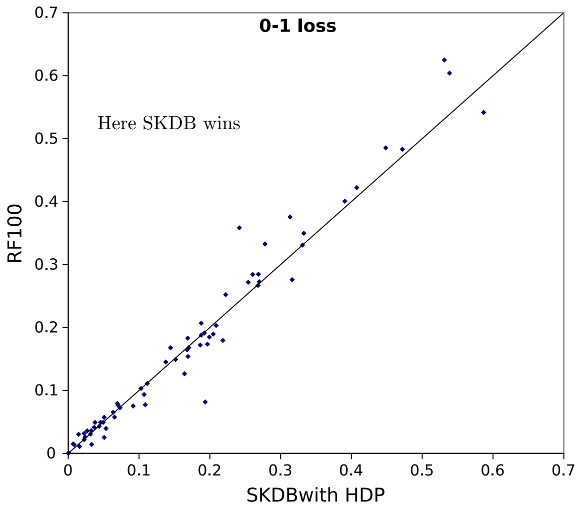


Fig. 8 0–1 loss scatter plot of SkDB with our HDP parameter estimate versus random forest

same behaviour for SkDB. Interestingly, for TAN on this dataset, although HDP estimates do learn faster, they are overtaken by m-estimates after 10,000 datapoints.

6.4 BNCs with HDP versus random forest

Having shown that our approach outperforms the state of the art for BNCs parameter estimation, we compare BNCs using our HDP estimates against random forest (RF). The aim of this section is not to suggest that BNCs should replace RF, but rather that BNCs can perform competitively.

Before proceeding, it is important to recall that RF is run on the same datasets as our BNCs with HDP estimates, i.e., with attributes discretized when necessary.

We report in Table 5 and Fig. 8 the results of TAN and SkDB. From this table we can see that RF is generally more accurate than the BNCs with m-estimates. Conversely, we can see that BNCs with HDP outperform RF more often, even with a model as simple as TAN. This result is important because our techniques are all completely out-of-core and do not need to retain the data in main memory, as do most state-of-the-art learners. Note that comparing 0–1 loss is probably fairer to RF, because RF is not a probabilistic model [even if plain RF estimates as we do have been reported to outperform other RF variations in terms of RMSE (Boström 2012)].

Obviously, for the larger datasets, RF catches up to TAN-HDP (which has a high-bias structure) but for the 10 largest datasets we considered, TAN-HDP still wins 6 times (1 draw) and SkDB-HDP is extremely competitive with a win–draw–loss of 7–0–3.

Table 6 Results on the Splice dataset on which RF cannot run

Classifier	0/1-loss	RMSE
SkDB5-*m*	1.499%	0.1093
SkDB5-HDP	0.318%	0.0544
XGBoost	0.314%	0.0594

6.5 Out-of-core capacity

Our last set of experiments aims at showcasing the out-of-core capacity of our system. We run SkDB on the Splice dataset (Sonnenburg and Franc 2010)—which contains 50 million training examples and is provided with a test dataset with 5M samples—and compare our HDP estimates to the m-estimates. Note that this dataset is imbalanced with only 1% of examples for the positive class.

On this dataset, RF could not run using Weka defaults, requiring more than our limit of 138 GB of RAM. We thus used instead XGBoost (Chen and Guestrin 2016), which is the state of the art for scalable mixtures of trees (here boosting) and used widely by data scientists to achieve state-of-the-art results on many machine learning challenges [XGBoost was used in 17 out of 29 winning solutions in the machine learning competition site Kaggle in 2015 (Chen and Guestrin 2016)]. We use XGBoost's default parameters as per version 0.6—we use maximum depth of 6 and 50 rounds of boosting. Similarly to the previous, the aim of this section is not to suggest that BNCs should replace XGBoost, but rather to show that BNCs are an interesting set of models that can perform out-of-core and perform competitively when using our HDP-estimates.

The results are reported in Table 6. They show that HDP dramatically improves both 0–1 loss and RMSE as compared to m-estimates. Note that m-estimates would even be outperformed in terms of error-rate by simply predicting the majority class. The comparison with XGBoost is interesting, it shows that SkDB5 with our HDP estimates comes very close to XGBoost in terms of 0–1 loss. In terms of probability calibration our HDP estimates even push BNCs beyond XGBoost's performance, as evidenced by the RMSE.

6.6 Running time

Although running time is not directly a focus of this paper, we give below some associated observations:

- Training time complexity increases linearly with the number of iterations the sampler runs for, linearly with the number of covariates and linearly with the number of nodes in the trees (which increases exponentially with depth).
- Training time is reasonable. As an example, training of $SkDB_5$-*HDP* (with $max\,K = 5$) on Splice with 50 million samples took under 4 h, among which 1.5 h are spent to learn the structure of the BN. $SkDB_5$ implied that the 140 independent hierarchies have a depth of 6 and we run 5000 iterations of the sampler. This also implies that $SkDB_5$-*m* takes a bit more than 1.5 h to be trained. XGBoost – which is a highly optimised package – on Splice required just under one hour of computation.
- For the Adult dataset training $SkDB_5$ with 25k samples and 50,000 iterations with level tying took 86 seconds, for the Abalone dataset training with 2k samples took 6 seconds – Classification time takes less than 1s to classify 25k samples, which is one of the strength of BNCs: once learned, classification is a simple look-up for each factor. This

classification time is actually under 1s for all models considered in this paper for the Adult dataset.

7 Conclusions

This paper presents accurate parameter estimation for Bayesian network classifiers using hierarchical Dirichlet process estimates, combining these well-researched areas for the first time. We have demonstrated that HDPs are not only capable of outperforming state-of-the-art parameter estimation techniques, but do so while functioning completely out-of-core. We have also showed that, for categorical data, this makes it possible to make BNCs highly competitive with random forest. We note that while BNCs are not currently state of the art for classification, they are still popular in applications. With this improvement in performance, and usable implementations in packages such as R, BNCs will be far more useful in real-world applications because they are readily implemented on high performance desktops, and do not require a cluster.

This work naturally opens up a number of opportunities for future research. First, we would like to perfect our sampler by assessing the influence of the different runtime configurations of our system including: how often should we sample concentration, widening the window of pseudo-counts at the start of the system and burn-in. Second, we would like to extend this work to Pitman–Yor processes, which offer an exciting avenue for research, in particular for variables with high cardinality. Third, we would like to extend this framework to the general class of Bayesian networks.

Acknowledgements This work was supported by the Australian Research Council under awards DE170100037 and DP140100087. The authors would like to thank Joan Capdevila Pujol and the anonymous reviewers for helping us strengthen the original manuscript.

References

Bielza, C., & Larrañaga, P. (2014). Discrete Bayesian network classifiers: A survey. *ACM Computing Surveys*, *47*(1), 5.

Boström, H. (2012). Forests of probability estimation trees. *International Journal of Pattern Recognition and Artificial Intelligence*, *26*(02), 125,1001.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*, 5–32.

Buntine, W. (1996). A guide to the literature on learning probabilistic networks from data. *IEEE Transactions on Knowledge and Data Engineering*, *8*(2), 195–210.

Buntine, W., & Mishra, S. (2014). Experiments with non-parametric topic models. In *20th ACM SIGKDD international conference on knowledge discovery and data Mining*, ACM, New York, NY, USA, KDD ’14 (pp. 881–890).

Carvalho, A. M., Roos, T., Oliveira, A. L., & Myllymäki, P. (2011). Discriminative learning of Bayesian networks via factorized conditional log-likelihood. *Journal of Machine Learning Research*, *12*(July), 2181–2210.

Chen, S., & Goodman, J. (1996). An empirical study of smoothing techniques for language modeling. In *34th Annual meeting on association for computational linguistics, ACL ’96* (pp. 310–318).

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, ACM, New York, NY, USA, KDD ’16 (pp. 785–794). https://doi.org/10.1145/2939672.2939785.

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

Du, L., Buntine, W., & Jin, H. (2010). A segmented topic model based on the two-parameter Poisson–Dirichlet process. *Machine Learning*, *81*(1), 5–19.

Fayyad, U., & Irani, K. (1992). On the handling of continuous-valued attributes in decision tree generation. *Machine Learning*, *8*(1), 87–102.

Ferguson, T. (1973). A Bayesian analysis of some nonparametric problems. *The Annals of Statistics*, *1*, 209–230.

Friedman, N., Geiger, D., & Goldszmidt, M. (1997). Bayesian network classifiers. *Machine Learning*, *29*(2), 131–163.

Friedman, N., & Koller, D. (2003). Being Bayesian about network structure. A Bayesian approach to structure discovery in Bayesian networks. *Machine Learning*, *50*(1–2), 95–125.

Gasthaus, J., & Teh, Y. (2010). Improvements to the sequence memoizer. In: J.D. Lafferty, C.K.I. Williams, J. Shawe-Taylor, R.S. Zemel, & A. Culotta, *Proceedings of the 23rd International Conference on Neural Information Processing Systems NIPS'10*, Vancouver, British Columbia, Canada, 6–9 December 2010 (Vol. 1, pp. 685–693). Curran Associates Inc.

Hardy, G. H. [(1889) 1920]. Letter. Transactions of the Faculty of Actuaries 8, pp. 180–181 (originally published on "Insurance Record 457").

Huynh, V., Phung, D. Q., Venkatesh, S., Nguyen, X., Hoffman, M. D., & Bui, H. H. (2016). Scalable nonparametric Bayesian multilevel clustering. In *UAI*.

Hwang, H. K. (1995). Asymptotic expansions for the Stirling numbers of the first kind. *Journal of Combinatorial Theory, Series A*, *71*(2), 343–351.

Koller, D., & Friedman, N. (2009). *Probabilistic graphical models—Principles and techniques. Adaptive computation and machine learning*. Cambridge, MA: The MIT Press.

Lewis, D. (1998). Naive Bayes at forty: The independence assumption in information retrieval. In *10th European conference on machine learning*, Springer, London, UK, ECML '98 (pp. 4–15).

Lichman, M. (2013). UCI machine learning repository. http://archive.ics.uci.edu/ml Accessed 23 Jan 2015.

Lidstone, G. (1920). Note on the general case of the Bayes–Laplace formula for inductive or a posteriori probabilities. *Transactions of the Faculty Actuaries*, *8*, 182–192.

Lim, K., Buntine, W., Chen, C., & Du, L. (2016). Nonparametric Bayesian topic modelling with the hierarchical Pitman–Yor processes. *International Journal of Approximate Reasoning*, *78*, 172–191.

Lyubimov, D., & Palumbo, A. (2016). *Apache Mahout: Beyond MapReduce* (1st ed.). North Charleston: CreateSpace Independent Publishing Platform.

Martínez, A., Webb, G., Chen, S., & Zaidi, N. (2016). Scalable learning of Bayesian network classifiers. *Journal of Machine Learning Research*, *17*(44), 1–35.

Mitchell, T. (1997). *Machine learning*. New York: McGraw-Hill.

Nguyen, V., Phung, D. Q., Venkatesh, S., & Bui, H. H. (2015). A Bayesian nonparametric approach to multilevel regression. In *PAKDD* (Vol. 1, pp. 330–342).

Rennie, J., Shih, L., Teevan, J., & Karger, D. (2003). Tackling the poor assumptions of naive Bayes text classifiers. In *20th International conference on machine learning*, AAAI Press, ICML'03 (pp. 616–623).

Roos, T., Wettig, H., Grünwald, P., Myllymäki, P., & Tirri, H. (2005). On discriminative Bayesian network classifiers and logistic regression. *Machine Learning*, *59*(3), 267–296.

Sahami, M. (1996). Learning limited dependence Bayesian classifiers. In *Second international conference on knowledge discovery and data mining* (pp. 334–338). AAAI Press, Menlo Park, CA.

Shareghi, E., Cohn, T., & Haffari, G. (2016). Richer interpolative smoothing based on modified Kneser-Ney language modeling. In *Empirical methods in natural language processing* (pp. 944–949).

Shareghi, E., Haffari, G., & Cohn, T. (2017a). Compressed nonparametric language modelling. In *IJCAI*. Accepted 23/04/2017.

Shareghi, E., Haffari, G., & Cohn, T. (2017b). Compressed nonparametric language modelling. In *Proceedings of the twenty-sixth international joint conference on artificial intelligence, IJCAI-17* (pp. 2701–2707). https://doi.org/10.24963/ijcai.2017/376.

Sonnenburg, S., & Franc, V. (2010). COFFIN: A computational framework for linear SVMs. In J. Fürnkranz & T. Joachims (Eds.), *ICML* (pp. 999–1006).

Teh, Y. (2006). A Bayesian interpretation of interpolated Kneser-Ney. Tech. Rep. TRA2/06, School of Computing, National University of Singapore.

Teh, Y., Jordan, M., Beal, M., & Blei, D. (2006). Hierarchical Dirichlet processes. *Journal of the American Statistical Association*, *101*(476), 1566–1581.

Webb, G. I., Boughton, J., & Wang, Z. (2005). Not so naive Bayes: Aggregating one-dependence estimators. *Machine Learning*, *58*(1), 5–24.

Webb, G., Boughton, J., Zheng, F., Ting, K., & Salem, H. (2012). Learning by extrapolation from marginal to full-multivariate probability distributions: Decreasingly naive Bayesian classification. *Machine Learning*, *86*(2), 233–272.

Wermuth, N., & Lauritzen, S. (1983). Graphical and recursive models for contigency tables. *Biometrika*, *70*(3), 537–552.

Wood, F., Gasthaus, J., Archambeau, C., James, L., & Teh, Y. (2011). The sequence memoizer. *Communications of the ACM*, *54*(2), 91–98.

Zaidi, N. A., Webb, G. I., Carman, M. J., Petitjean, F., Buntine, W., Hynes, M., et al. (2017). Efficient parameter learning of Bayesian network classifiers. *Machine Learning*, *106*(9–10), 1289–1329.