

Flexible learning of k-dependence Bayesian network classifiers

Arcadio Rubio, José Antonio Gámez

▶ **To cite this version:**

Arcadio Rubio, José Antonio Gámez. Flexible learning of k-dependence Bayesian network classifiers. GECCO '11: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, Jul 2011, Dublin, Ireland. pp.1219, 10.1145/2001576.2001741 . hal-02945041

HAL Id: hal-02945041
https://hal.science/hal-02945041

Submitted on 22 Sep 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Flexible Learning of k-Dependence Bayesian Network Classifiers

Arcadio Rubio
Collaborative Intelligent Systems Unit
European Centre for Soft Computing
C/ Gonzalo Gutiérrez Quirós s/n
33600 Mieres, Asturias, Spain
arcadio.rubio@softcomputing.es

José A. Gámez
Computing Systems Department
University of Castilla-La Mancha
Campus Universitario s/n
Albacete, 02071, Spain
jose.gamez@uclm.es

ABSTRACT

In this paper we present an extension to the classical k-dependence Bayesian network classifier algorithm.

The original method intends to work for the whole continuum of Bayesian classifiers, from naïve Bayes to unrestricted networks. In our experience, it performs well for low values of k. However, the algorithm tends to degrade in more complex spaces, as it greedily tries to add k dependencies to all feature nodes of the resulting net.

We try to overcome this limitation by seeking for optimal values of k on a feature per feature basis. At the same time, we look for the best feature ordering. That is, we try to estimate the joint probability distribution of optimal feature orderings and individual number of dependencies. We feel that this preserves the essence of the original algorithm, while providing notable performance improvements.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*induction, knowledge acquisition*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*heuristic methods*

General Terms

Machine learning, classification

Keywords

Bayesian network classifier, estimation of distribution algorithm, flexible learning

1. INTRODUCTION

1.1 Probabilistic classifiers

Supervised classification is a very common machine learning task with applications in many aspects of daily life. Examples include spam detection, automatic mail tagging, or product recommendations. Probabilistic classifiers offer significant advantages over other approaches for these kind of problems. They are able to deal naturally with uncertainty and estimate not only the label assigned to every object, but also the probability distribution over all possible labels for the class variable.

Among probabilistic classifiers, perhaps those based on Bayesian networks (BNs) are the most used in practice [12, 18]. They inherit important virtues from these, which stem from their *rigorous probability basis* [23]. In the probabilistic setting, classification is carried out by estimating the probability distribution of the class variable given the attributes. These probability estimates are strictly more powerful than plain class predictions, as they allow to rank and minimize the *expected cost*.

It should be noted that general-purpose Bayesian networks can be applied to the the aforementioned problem—determining the class of an instance given its attributes. However, this is an overkill. Bayesian networks can infer any probability from the joint distribution given some prior knowledge, not just the class one.

1.2 Naïve Bayes and its extensions

Since we do not require the flexibility of reasoning about any possible variable, we can build simplified custom nets focused in making predictions exclusively about the class. The most simple one is the well-known naïve Bayes (NB), which assumes independence among all features given the class [8]. Despite this strong premise about attributes, it works surprisingly well, even when compared to other more sophisticated models [7, 18].

More precisely, naïve Bayes has a good performance if we employ the usual 0-1 loss function, i.e. *accuracy*, as a benchmark. However, this does not stand if we consider a performance measure which takes into account the predicted probabilities, e.g. the *mean square error* or *Brier score*. Note that many real world applications require not only a good prediction of the most probable class label, but also a precise ranking of all the label probabilities.

Between fully general Bayesian network classifiers and naïve Bayes, there is a whole spectrum of classifiers that offer different trade-offs with respect to the model learned. The so-called *semi-naïve Bayes* approach [17, 22, 28] subsumes all those which have been designed as a direct generalisation of naïve Bayes. The idea is to relax conditional independence relations among features so that redundant attributes do not degrade performance, while preserving the naïve structure.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
GECCO'11, July 12–16, 2011, Dublin, Ireland.
Copyright 2011 ACM 978-1-4503-0557-0/11/07 ...$10.00.

Featured members in the semi-naïve family are those algorithms that increase the allowed dependencies by using *clusters* of variables instead of single ones [17, 22]. Besides, it also includes those that allow extra dependencies between predictive attributes by adding arcs, like tree-augmented naïve Bayes (TAN) [12], k-dependence BNs (KDB) [25], or super-parent TAN (SP-TAN) [16]. Finally those which use mixtures of NB-based limited structures such as AODE [27] are also comprised.

Not surprisingly, even a modest relaxation of NB assumptions leads to much larger network spaces where the algorithms have to find the optimal model for a given problem. In order to maintain tractability, semi-naïve methods mainly use greedy strategies. These are typically $O(n^2)$, where n is the number of predictive variables.

While such an approach keeps running time at a minimum, the greediness leads to locally optimal solutions, often far from the optimum. Hence, we consider it appropriate to trade time for potentially better solutions by embracing metaheuristic search procedures as a mechanism for exploring network spaces. In particular, we build our approach as a generalisation of KDB [25].

2. THE GREEDY KDB ALGORITHM

Our setting is a problem with n predictive attributes $X_1, \ldots, X_n$ and a class C.[1] We assume all of them are *discrete* variables taking values in a finite and disjoint set $\Omega(X_i)$. Therefore, our goal is to obtain a *discrete* semi-naïve classifier.

The notion of k-dependence estimators was introduced by Sahami [25]. In those, the probability of each attribute value is conditioned by the class and, at most, k other attributes. The resulting model is a k-dependence BN (KDB).

Hence, by simply setting k, i.e., the maximum number of parent nodes that any attribute may have, we can establish the complexity of the classifiers we are looking for. These range from a simple NB structure ($k = 0$) to fully generalised BN structures ($k = n$), and subsume other semi-NB approaches like TAN and super-parent one dependence estimators ($k = 1$). Due to its flexibility, we find the KDB algorithm especially appealing.

The KDB algorithm adopts a greedy strategy in order to identify the graphical structure of the resulting classifier. It is based on the information theory concepts of mutual information and conditional mutual information [5]. Algorithm 1 shows the pseudocode of the original *greedy* KDB algorithm, which takes as input the dataset, k and θ — a user-defined parameter to avoid overfitting.

The algorithm first computes the mutual information $I(X_i; C)$ between each predictive attribute X_i and the class C. Attributes are sorted in decreasing order. Then, a set $\mathbf{S}$ is initialised to the empty set. It is used for keeping track of the nodes already considered. Nodes are traversed in the ordering just established. When the j-th variable in that order is added, $k + 1$ variables are set as its parents in the graph:[2] the class C and the k variables already considered with greater conditional mutual information with respect to X_j given the class $I(X_j; X_m|C)$.

[1]We use capital letters for variables (e.g. $X_1, \ldots, X_n, C$); boldfaced capital letters for sets of variables (e.g. $\mathbf{S}$) and non-capital letters for values of variables (e.g. $\Omega(C) = \{c_1, \ldots, c_r\}$).

[2]Obviously, if $j \leq k$ only j parents are added.

Algorithm 1 Greedy KDB

```
1: function KDB(data, k, θ)
2:     C ← class(data)
3:     net ← C
4:     N ← sort(features(data), f(X_i) I(X_i; C))
5:     S ← ∅
6:     for X_j in N do
7:         add_node(X_j)
8:         add_edge(net, X_j, C)
9:         SS ← sort(S, f(X_m) I(X_j; X_m|C))
10:        SS ← filter(SS, f(X_m) I(X_j; X_m|C) > θ)
11:        SS ← take(SS, min(k, |S|))
12:        for X_m in SS do
13:            add_edge(net, X_i, X_m)
14:        end for
15:        S ← S ∪ {X_j}
16:    end for
17:    return net
18: end function
```

Figure 1 shows three different networks for a problem with 4 predictive attributes: (a) NB; (b) a KDB structure with k=1, also a TAN as the graph between the predictive attributes is a tree; (c) a KDB structure with k=2.

3. GENERALIZED KDB

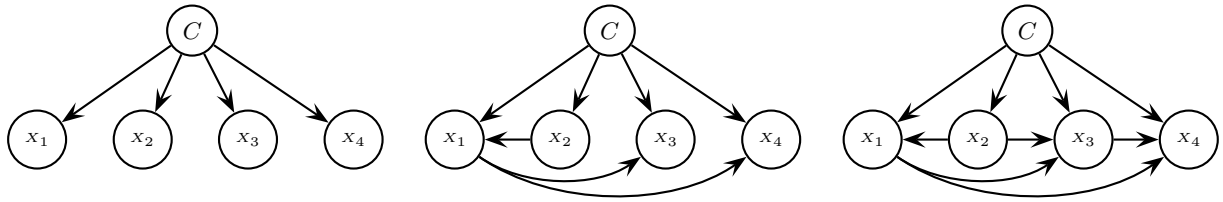
The original greedy KDB algorithm is very efficient and usually achieves significantly better performance than NB and other semi-NB algorithms. However, we can identify some limitations on its behaviour:

- The algorithm is guided by a greedy ordering obtained by using marginal knowledge between the attributes and the class only. This solution is likely to be a suboptimal one given that interactions between predictive attributes are not considered.

- Since k is the same for all nodes, it is unpractical to set it to values higher than 5 or 6 in almost every scenario. This results in overly complex networks, as all nodes are given k parents. This unnecessary complexity harms classifier performance, as it leads to overfitting. Furthermore, it is not possible to model cases where some nodes have a large number of dependencies, whereas others just have a few.

To relax this behaviour and prevent overfitting, Sahami introduced the threshold θ in order to avoid adding edges between variables (almost) conditionally independent given the class [25]. However, relying on θ for fixing that issue is not a realistic solution, as this value is difficult to determine by a user. In fact, most of the times KDB is used on the literature, θ is not set. Hence, exactly k predictive attributes are set as parents for all the nodes — but for the first k.

We can overcome these two drawbacks by providing the algorithm with such an *optimal* ordering and an *individual* k-value for each node, which leads us to the *generalised KDB* method shown in algorithm 2. Note that the parameter $\mathbf{N}$ received by the algorithm is a vector of pairs representing the ordering or permutation (first component), and the number of parents for each variable (second component).



(a) Naive Bayes (b) KDB(1)/TAN (c) KDB(2)

Figure 1: Examples of 4-predictive-attributes network structure for the BN classifiers: NB, KDB(1)/TAN and KDB(2)

For instance, if we are considering a dataset with 5 features $\{X_0, X_1, X_2, X_3, X_4\}$ $\mathbf{N}$ could be

$$\mathbf{N} = ((1,0),(3,1),(2,2),(0,3),(4,1))$$

which indicates that our generalised KDB should consider first attribute X_1 and add no parents to it.
Then it should consider attribute X_3 adding 1 parent, and so on.

Algorithm 2 Generalised KDB

```
1: function gKDB(data, N)
2:     C ← class(data)
3:     net ← C
4:     S ← ∅
5:     for p in N do
6:         add_node(p.attribute)
7:         add_edge(net, p.attribute, C)
8:         SS ← sort(S, f(X_m) I(p.attribute; X_m|C))
9:         SS ← filter(SS, f(X_m) I(p.attribute; X_m|C))
10:        SS ← take(SS, min(p.k, |S|))
11:        for X_m in SS do
12:            add_edge(net, X_i, X_m)
13:        end for
14:        S ← S ∪ {p.attribute}
15:    end for
16:    return net
17: end function
```

4. FLEXIBLE KDB

Obviously, the generalized KDB algorithm just described has an intentional omission. The mechanism for feeding in orderings and number of parents per node has not being specified.

We are facing a search problem where the evaluation function is precisely the previous algorithm. Hence we need to determine how to explore the problem space. In other words, we have to decide how to look for good node orderings and k-values for each node.

Therefore, our goal now is to design an optimization procedure than will search in a joint space consisting on the space of permutations to identify the sequence or ordering in which attributes are considered, times the space of integer vectors to identify the number of parents (k_i) for each node.

Thus, if we are dealing with n predictive attributes and $\hat{k}$ is the max number of parents a node can have, the cardinality of the search space is $n! \cdot (\hat{k}+1)^n$.

For instance, if we are considering a dataset with 5 features $\{X_0, X_1, X_2, X_3, X_4\}$ and $\hat{k} = 3$, a possible solution to our problem is the pair

$$\langle \sigma = (1,3,2,0,4) \ ; \ \mathbf{K} = [0,1,2,3,1] \rangle$$

which gives rise to vector $\mathbf{N}$ shown above.

4.1 Solution representation

Given a problem with n predictive variables $\{X_0, \ldots, X_{n-1}\}$ and a value $\hat{k}$ representing the max number of parents per variable, we encode any solution as a vector consisting on n pairs

$$((a_0,k_0),(a_1,k_1),\ldots,(a_{n-1},k_{n-1}))$$

- $a_i \in \{0,1,\ldots,n-1\}$
- $(a_0, a_1, \ldots, a_{n-1})$ is a permutation without repetition of numbers $\{0,1,\ldots,n-1\}$
- $0 \leq k_i \leq \hat{k}$, $\forall i = 0,\ldots,n-1$
- (a_i, k_i) accounts for the fact that variable X_{a_i} is used in position i-th of the ordering by generalized KDB, and has exactly k_i parents

4.2 Fitness function

Since the individual is used as input for generalized KDB, its evaluation depends on the goodness of the KDB model returned by that algorithm. We can measure this goodness in different ways:

- *Wrapper* Being in a supervised classification setting, we can use a classification-based score, e.g. accuracy. That is, we evaluate the learned model or network $\mathcal{B}$ by measuring how well it performs when used on unseen instances. Usually, a holdout validation set is used to compute this score.
- *Filter* A local score metric is considered as quality measure of a network structure $\mathcal{B}$ given the training data $Q(\mathcal{B}, \mathbf{D})$. The quality measure can be based on a Bayesian approach, minimum description length, information theory, etc.

Usually, the wrapper approach will lead to classifiers with *higher* accuracy. However, it has the disadvantage of forcing us to keep some instances for testing, which slows down the process due to the validation process, and leads to overfitting with respect to the validation set. The latter is usually amended by using a r-folds cross-validation instead of a holdout test set.

But then the process becomes even slower as a consequence of r models having to be trained and validated to assess the goodness of each individual. Because of those drawbacks and due to the fact that we need to score a large number of individuals—and hence networks—we have decided to evaluate solutions by using the filter approach.

More concretely we use the Akaike Information Criterion (AIC) [1], a metric grounded in information theory and commonly used for model selection when learning Bayesian networks.

AIC scores a network $\mathcal{B}$ with respect to a dataset $\mathbf{D}$

$$Q_{AIC}(\mathcal{B}, \mathbf{D}) = LL(\mathcal{B}, \mathbf{D}) - K$$

where the first term accounts for the log-likelihood of the data $\mathbf{D}$ given the network $\mathcal{B}$, and K is a penalty term which accounts for the number of parameters needed to specify $\mathcal{B}$. The denser the network the greater the penalty

$$LL(\mathcal{B}, \mathbf{D}) = \sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk} log \frac{N_{ijk}}{N_{ij}}$$

$$K = \sum_{i=1}^{n} (r_i - 1) q_i$$

Computation of AIC only requires to count frequencies from the data set.[3] As other scoring functions, AIC has the property of being additively decomposable. That is, the score of the whole network can be decomposed into the sum of the scores of the individual nodes, given their parents. For example, the network in figure 1 (b) leads to

$$\begin{aligned} Q_{AIC}(\mathcal{B}) &= Q_{AIC}(C) \\ &+ Q_{AIC}(C \rightarrow X_1 \leftarrow X_2) \\ &+ Q_{AIC}(C \rightarrow X_2) \\ &+ Q_{AIC}(C \rightarrow X_3 \leftarrow X_1) \\ &+ Q_{AIC}(C \rightarrow X_4 \leftarrow X_1) \end{aligned}$$

Decomposability is a very interesting property when, as in our case, we must visit and evaluate different networks. This is because we can memoize, e.g. using a cache, the local score for previously visited node families. Later, when a network must be evaluated, we may decompose it into local scores, and retrieve those previously evaluated. Therefore only those node families not visited previously are actually scored by counting frequencies from the data set. And of course, they are added immediately to our memoization structure.

To sum up, the evaluation or fitness of an individual $I = ((a_0, k_0), (a_1, k_1), \ldots, (a_{n-1}, k_{n-1}))$, for a given dataset $\mathbf{D}$ is computed as

$$f(I) = Q_{AIC}(\mathsf{gKDB}(I, \mathbf{D}), \mathbf{D}).$$

In this paper our goal is to *maximize* $f(\cdot)$, and we make use of a memoization structure or cache in order to optimise statistics computation during network scoring.

[3] n is the number of variables; r_i is the number of states for variable X_i; $\pi_{\mathbf{i}}$ is the parents set of X_i in $\mathcal{B}$; q_i is the number of configurations of $\pi_{\mathbf{i}}$, i.e., $q_i = \prod_{X_j \in \pi_{\mathbf{i}}} r_j$; N_{ijk} is the number of instances in $\mathbf{D}$ in which X_i takes its k-th value and $\pi_{\mathbf{i}}$ takes as value its $j-th$ configuration; and N_{ij} is the number of instances in $\mathbf{D}$ in which $\pi_{\mathbf{i}}$ takes as value its j-th configuration, i.e., $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$.

4.3 Search algorithm

We have adopted an estimation of distribution approach [19]. That is, we want to learn the probability distribution of good node permutations augmented with the number of parents for each one.

Unfortunately, such a joint distribution becomes unmanageable even for a modest number of nodes. At least, unless we adopt a compact representation i.e., another Bayesian network.

We can maintain tractability easily if we try to learn a simplified version of this distribution instead. Clearly, what matters more is the absolute ordering of nodes. Hence, we can try to learn the marginal distribution of node times number of parents pairs for each position in the permutation.

The node histogram-based sampling algorithm (NHBSA) has proven to perform very well on absolute permutation scenarios [26]. We can easily adapt it to learn the marginal distribution augmented with parent numbers.

NHBSA starts by generating a large population of random permutations uniformly, since the distribution is still unknown. Once those permutations have been generated and evaluated, a node histogram matrix is created. This data structure is simply a representation of the marginal frequencies at each position. Therefore, the node histogram model uses the most simple type of Bayesian network, a univariate model where no dependence relations are allowed, taking a similar form to UMDA [21].

As an example, suppose that we have generated the following population of size 3

$$\{((0,0),(1,1),(2,1)),((1,0),(2,0),(0,1)),((0,0),(1,0),(2,1))\}$$

The corresponding node histogram matrix is what follows. Rows are indexed by position numbers in the permutation $(p_i, i = 0, 1, 2)$, whereas columns correspond to pairs ordered lexicographically

$$\begin{array}{c} \\ p_0 \\ p_1 \\ p_2 \end{array} \begin{array}{c} \begin{array}{cccccc} \text{0:(0,0)} & \text{1:(0,1)} & \text{2:(1,0)} & \text{3:(1,1)} & \text{4:(2,0)} & \text{5:(2,1)} \end{array} \\ \begin{pmatrix} 2 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 2 \end{pmatrix} \end{array}$$

Hence, a value 2 in row 2 column 5 — counting begins at 0 — indicates that the population contains 2 elements which have the pair $(2, 1)$ in the last position of their permutations.

Once the matrix is built, random permutations are sampled from it using algorithm 3. This simply builds valid permutations filling in the positions in a random order, so that no bias is introduced. For each position, the corresponding row in the matrix — restricted to elements in the current candidate list ($nhbsa[i, \cdot]^{\downarrow CL}$) — is used during the sampling.

So, elements have a probability to be inserted into the string equal to their relative frequency stored in the matrix. After sampling a pair, all the elements in CL containing the sampled position (attribute) are removed from CL.

Usually the matrix taken for sampling is the node histogram matrix plus a uniform matrix with all positions filled with a bias term. This term is introduced so that elements which have not appeared in the population are given a chance to appear.

Algorithm 3 NHBSA permutation string sampling

```
1: function sample_nhbsa(nhbsa)
2:     nr ← nrows(nhbsa)                          ▷ nr = n(um vars)
3:     nc ← ncols(nhbsa)                          ▷ nc = nr · k̂
4:     p ← random_permutation([0, 1, ..., nr − 1])
5:     CL ← {0, 1, ..., nc − 1}                   ▷ CL: candidate list
6:     s ← array[0..nr − 1]
7:     for j = 0 to nr − 1 do do
8:         i ← p[j]
9:         r ← sample(nhbsa[i, ·]^↓CL)
10:        (a_r, k_r) ← the pair corresponding to r
11:        s[i] ← (a_r, min(k_r, j))
12:        CL ← CL − {r' | a_r = a'_r}
13:    end for
14:    return s
15: end function
```

Such an approach has the same purpose as the Laplace smoothing employed in standard UMDA, and helps to avoid premature convergence.

Once a new permutation is generated using *sample_nhbsa*, the population is updated using a random-selection steady state scheme. That is, the new permutation is compared against a randomly selected element from the population. If its fitness is higher, it replaces the corresponding element in the population, and the matrix *nhbsa* is updated accordingly. Given a large population size and enough iterations, the matrix estimates the distribution of good orderings accurately.

4.4 Further Improvements

We have considered two main improvements to the previously described algorithm:

- *Sampling with template* In order to approximate the joint distribution better than simply using the marginal one, we have also introduced a method presented in [26], which we only sketch here. Although absolute ordering is what matters most, it is reasonable to expect relationships among nodes. We can model those relations by selecting a template individual from the population. Then, new individuals are created by copying certain parts of the template, whereas others are created by using algorithm 3.
- *Enhanced population initialization* Plain NHBSA samples such a population at random. However, we think that good solutions might be close to those found by the greedy version of KDB. Hence, random sampling might slow down convergence. We have adopted a solution taken from scatter search [20]. Instead of sampling initial permutations from a uniform distribution, we use a GRASP algorithm [10] we have created for the occasion. This algorithm can be used to create a diverse and at the same time good initial population, so that the target distribution is found faster.

The idea consists on using the permutation created by the greedy version of KDB as a reference. Positions in new permutations are filled by sampling at random the corresponding element in the reference permutation, plus those elements *window* positions away. It is easy to see that *window* controls diversity.

By running this GRASP method a few tens of times for different *window* values and taking the fittest elements as initial population, we have found out that good solutions are found much faster. This is not surprising, as it is one of the basic elements of scatter search. Note that such a GRASP is also a metaheuristic on its own.

5. EXPERIMENTAL EVALUATION

Finally, we describe the experiments carried out to assess the validity of our approach. First we enumerate the datasets and algorithms used for our comparisons. Then, we outline the experiment setting and we discuss on the obtained results.

5.1 Test suite

We have employed 10 well-known datasets downloaded from the UCI repository [2], which are aimed at classification problems. For those datasets containing numerical variables, an entropy-based supervised discretization process has been applied [9]. As reported in [11] even if a discretization method produces different results for a particular dataset, this does not really have an effect when Bayesian network classifiers are being compared over several datasets. Thus, the only fairness criteria in this respect is to apply the same discretization method throughout the whole experiment. Table 1 displays these datasets and their main characteristics.

Table 1: Datasets—number of predictive variables (n), number of class labels (c), and number of instances (t)

Dataset	n	c	t
Anneal	38	6	898
Balance	4	3	625
Glass	9	7	214
Ionosphere	34	2	351
Mfeat	6	10	2000
Pen	16	10	10992
Segment	19	7	2310
Sonar	60	2	208
Tic-tac-toe	9	2	958
Vehicle	18	4	846

Regarding classifiers, the following algorithms have been considered:

- Bayesian network classifiers
  - Naïve Bayes (NB) [8]
  - Greedy KDB (KDB) [25]
  - Tree-augmented naïve Bayes (TAN) [12]
  - Averaging one-dependence est. (AODE) [27]
  - AODE weighted version (wAODE) [15]
- Flexible KDB approach: our NHBSA algorithm (NH)
- Other flexible approaches for learning Bayesian network classifiers: the Tabu search procedure and the genetic algorithm implemented in the Weka data mining suite [3, 14]
- Decision trees: C4.5 [24]

For those methods requiring input parameters, standard values have been selected. Thus, the maximum number of parents has been set to 4 in K2, while $k = 2$ has been used for KDB. Note that $k = 1$ gives rise to a TAN structure. For NHBSA we have set $\hat{k} = 10$.

5.2 Experiments and results

All the experiments have been done by using the Weka data mining suite [14]. We have used the algorithms available therein and coded our own proposal using Weka's API. Besides, we have also implemented the greedy version of KDB which, despite being a classical method, is not included in this framework.

In all the cases we have used 10-fold cross-validation, so we report the averaged accuracy over the ten test folds. For the sake of fairness, a maximum amount of available CPU time has been set for all the algorithms: 300 s.

There is another key point in our experiments. By earlier experimentation we realised that the three flexible classifiers and obviously C4.5 are rather unstable. That is, results for different folds suffer from significant variations in accuracy. A typical way to overcome this issue is to use bagging, a statistical re-sample and combine technique, first proposed by Breiman [4]. However, bagging can degrade the performance of stable predictors, so we only apply it to the aforementioned four algorithms.

Table 2 shows the averaged accuracy results. The last row displays the averaged values over the ten datasets. We have highlighted in bold face the results for the three flexible approaches.

5.3 Analysis

We base our study in the application of statistical analyses for multiple classifiers and multiple datasets [6] by using the software provided as companion in [13].

First, a Friedman test is carried out, which reports a significant statistical difference between the tested algorithms: p-value 1.730732779314792E-9. Table 3 shows the ranking that results from running a Friedman test for all the algorithms present in our study. Our proposal is the ranked first, followed by the other two flexible approaches.

Table 3: Average rankings of the algorithms

Algorithm	Ranking
NH	1.20
GA	2.50
Tabu	3.67
wAO	4.85
KDB	6.15
AODE	6.30
TAN	6.40
C4.5	6.75
K2	7.85
NB	9.30

Then, the process chooses the NH algorithm as a control reference and performs a post-hoc Holm test, which rejects any statistical difference between NH, wAODE, GA and Tabu (α=0.05). Table 4 shows the obtained p-values for Holm's test. Both, the original ones and adjusted p-values—which take into account that multiple tests have been conducted—are shown.

Holm's procedure rejects those hypotheses that have a p-value ≤ 0.025 (≤ 0.001428 when using adjusted p-values). These have been labeled with a * in the table.

Table 4: Post-hoc Holm's test with adjusted p-values

Algorithm	Unadjusted p	p_{Holm}
NB	$2.200809E-9^*$	$1.980728E-8^*$
K2	$9.045124E-7^*$	$7.236099E-6^*$
C4.5	$4.150346E-5^*$	$2.905242E-4^*$
TAN	$1.228067E-4^*$	$7.368403E-4^*$
AODE	$1.654860E-4^*$	$8.274303E-4^*$
KDB	$2.563639E-4^*$	0.001025^*
wAO	0.007024^*	0.021072
Tabu	0.064838	0.129676
GA	0.336998	0.336998

From the test results we can conclude that our algorithm beats non-flexible methods, except for wAODE—when using adjusted p-values. We would like to remark here that: (1) wAODE is a tuned algorithm—i.e. our flexible algorithm performs better than its non-tuned version, AODE. (2) From the non-adjusted p-value results, it seems that the inclusion of more datasets in the comparison would likely yield statistical differences between our method and wAODE.

Regarding the other two flexible classifiers, performance is equivalent in our current experiments. However, we think our proposal will improve its performance if tests were performed giving more running time, whereas the others might stall. We base this conjecture on the fact that Weka's GA and Tabu algorithms are seeded with really sparse networks, i.e. number of edges equal to number of vars, $k \simeq 1$. We allow our model to set up to 10 parents, though. To sum up, more experiments with more datasets and different parameter settings are needed in order to obtain stronger conclusions.

6. CONCLUSIONS

We have proposed an estimation of distribution method to address the limitations of KDB. Our approach not only performs a metaheuristic search over the space of node permutations, but also tries to set k on a node per node basis.

We feel the latter feature is essential to realize the original KDB goal of being able to explore a wide range of networks. Otherwise, for large values of k networks become too complex to be practical since all nodes are greedily assigned an equal number k of parents.

A deeper experimentation would be desirable to confirm the fact that NHBSA has competitive performance against other greedy and metaheuristic approaches.

We also look forward to study the effect of generating initial populations using GRASP versus random permutations drawn from uniform distributions, and running NHBSA with or without templates.

7. ACKNOWLEDGMENTS

This work has been partially supported by the Spanish Ministry of Education under Project TIN2010-20900-C04-03, the Education Council of Castilla-La Mancha under Project PCI08-0048-8577574, and FEDER funds.

Table 2: Method error rate on 10-fold cross-validation

	NB	KDB	TAN	K2	Tabu	GA	NH	AODE	wAO	C4.5
Anneal	96.66	99.11	98.00	98.33	99.16	99.38	99.34	98.33	98.89	98.78
Balance	70.72	70.88	71.04	60.80	75.18	73.37	75.59	69.60	69.92	70.72
Glass	74.30	75.70	75.70	75.23	80.01	81.12	83.19	76.64	78.04	75.23
Ionosphere	90.60	92.31	93.16	92.59	94.19	94.23	94.45	92.59	93.16	91.45
Mfeat	69.40	71.65	70.20	69.85	73.24	72.89	74.67	70.35	70.85	71.90
Pen	87.90	97.24	96.45	95.49	97.95	97.94	98.33	97.84	98.36	90.61
Segment	91.52	94.76	95.54	95.46	92.69	97.10	98.01	95.58	96.75	95.24
Sonar	85.58	78.37	85.10	77.40	86.03	87.01	87.11	86.06	87.02	85.58
Tic-tac-toe	69.62	80.06	76.72	83.82	89.88	90.47	94.03	73.07	72.65	92.80
Vehicle	62.65	74.94	73.76	72.93	77.92	78.07	78.21	73.05	73.64	72.81
mean	79.90	83.50	83.57	83.24	**86.63**	**87.16**	**88.29**	83.31	83.93	84.51

8. REFERENCES

[1] H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974.

[2] A. Asuncion and D. Newman. UCI Machine Learning Repository, 2007. University of California, Irvine, School of Information and Computer Sciences. `www.ics.uci.edu/~mlearn/MLRepository.html`.

[3] R. R. Bouckaert. Bayesian network classifiers in Weka. Technical report, University of Waikato, New Zealand, 2005.

[4] L. Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[5] T. M. Cover and J. Thomas. *Elements of Information Theory*. Wiley, 1991.

[6] J. Demsar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, 2006.

[7] P. Domingos and M. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997.

[8] R. Duda and P. Hart. *Pattern classification and scene analysis*. John Wiley and Sons, 1973.

[9] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the International Joint Conference on Uncertainty in AI*, pages 1022–1029, 1993.

[10] T. A. Feo and M. G. C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

[11] J. Flores, J. A. Gámez, A. M. Martínez, and J. M. Puerta. Analyzing the Impact of the Discretization Method When Comparing Bayesian Classifiers. In *Trends in applied intelligent systems*, volume 6096 of *Lecture Notes in Artificial Intelligence*, pages 570–579. Springer-Verlag, 2010.

[12] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29(2-3):131–163, 1997.

[13] S. García and F. Herrera. An Extension on "Statistical Comparisons of Classifiers over Multiple Data Sets" for all Pairwise Comparisons. *Journal of Machine Learning Research*, 9:2677–2694, 2008.

[14] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *SIGKDD Exploration Newsletter*, 11:10–18, November 2009.

[15] L. Jiang and H. Zhang. Weightily averaged one-dependence estimators. In *Proceedings of the 9th Pacific Rim international conference on Artificial intelligence*, pages 970–974. Springer-Verlag, 2006.

[16] E. Keogh and M. Pazzani. Learning Augmented Bayesian Classifiers: A Comparison of Distribution-based and Classification-based Approaches. In *Proc. of the 7th Int. Workshop on AI and Statistics*, pages 225–230, 1999.

[17] I. Kononenko. Semi-naive bayesian classifier. In *Proceedings of the European working session on learning on Machine learning*, pages 206–219. Springer-Verlag New York, Inc., 1991.

[18] P. Langley, W. Iba, and K. Thompson. An analysis of bayesian classifiers. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI)*, pages 223–228. MIT Press, 1992.

[19] P. Larrañaga and J. A. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[20] R. Martí, M. Laguna, and F. Glover. Principles of scatter search. *European Journal of Operational Research*, 169(2):359–372, 2006.

[21] H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions I. Binary parameters. In *Parallel Problem Solving from Nature – PPSN IV*, volume 1141 of *Lecture Notes in Computer Science*, pages 178–187. Springer Berlin / Heidelberg, 1996.

[22] M. J. Pazzani. Searching for dependencies in bayesian classifiers. In *Proceedings of the 5th Workshop on Artificial Intelligence and Statistics*, 1996.

[23] J. Pearl. *Probabilistic reasoning in intelligent systems: newtorks of plausible inference*. Morgan Kaufman, 1988.

[24] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[25] M. Sahami. Learning limited dependence bayesian classifiers. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 334–338, Menlo Park, CA, USA, 1996. AAAI Press.

[26] S. Tsutsui, M. Pelikan, and D. E. Goldberg. Node histogram vs. edge histogram: A comparison of PMBGAs in permutation domains. Technical Report 2006009, Missouri Estimation of Distribution Algorithms Laboratory, University of Missouri–St. Louis, MO, USA, 2006.

[27] G. I. Webb, J. R. Boughton, and Z. Wang. Not so naive bayes: Aggregating one-dependence estimators. *Machine Learning*, 58(1):5–24, 2005.

[28] F. Zheng and G. Webb. A Comparative Study of Semi-naive Bayes Methods in Classification Learning. In *Proc. of the 4th Australasian Data Mining Conf. (AusDM05)*, pages 141–156, Sydney, 2005. University of Technology.