Bias Management of Bayesian Network Classifiers

Gladys Castillo[1,2] and João Gama[1,3]

[1] LIACC, University of Porto, Portugal
[2] Department of Mathematics, University of Aveiro, Portugal
[3] FEP, University of Porto, Portugal
gladys@mat.ua.pt, jgama@liacc.up.pt

Abstract. The purpose of this paper is to describe an adaptive algorithm for improving the performance of Bayesian Network Classifiers (BNCs) in an on-line learning framework. Instead of choosing a priori a particular model class of BNCs, our adaptive algorithm scales up the model's complexity by gradually increasing the number of allowable dependencies among features. Starting with the simple Naïve Bayes structure, it uses simple decision rules based on qualitative information about the performance's dynamics to decide when it makes sense to do the next move in the spectrum of feature dependencies and to start searching for a more complex classifier. Results in conducted experiments using the class of Dependence Bayesian Classifiers on three large datasets show that our algorithm is able to select a model with the appropriate complexity for the current amount of training data, thus balancing the computational cost of updating a model with the benefits of increasing in accuracy.

Keywords: Bias Management, Bayesian Classifiers, Machine Learning.

1 Introduction

Efficient learning algorithms usually involve an artful trade-off of *bias* vs. *variance*. If we choose a model that is too complex for the amount of training data we have, it will *overfit* the data. The model has too much variance. Otherwise, if the model is too simple, it cannot capture the true structure in the data, it will *underfit* the data. The model has too much bias. We can improve the performance of learning algorithms if we reduce either bias or variance. When we have few training data we can reduce variance by using simpler models while not increasing our bias too much. However, as it was shown in [2] as training set size increases variance will decrease and this will become a less significant part of the error. In this case, we must place more focus on bias management.

A well-studied and effective classifier is Naïve Bayes (NB). Although NB has a high bias due to its strong feature independence assumptions, its performance is compensated by its high variance management, thus producing accurate classification. **B**ayesian **N**etwork **C**lassifiers (BNCs) have been the natural choice

A. Hoffmann, H. Motoda, and T. Scheffer (Eds.): DS 2005, LNAI 3735, pp. 70–83, 2005.

for improving the predictive capability of NB. For instance, TAN classifiers [4] reduce the NB's bias by allowing the features to form a tree. In this paper, we examine an adaptive algorithm for improving the performance of BNCs in an on-line learning framework. Instead of choosing a priori a particular model class of BNCs, we propose to scale up the model's complexity by gradually increasing the number of allowable dependencies among features. If we scale up complexity slowly enough, the use of more training data will reduce bias at a rate that also reduces variance and consequently the classification error. This structure regularization leads to the selection of simpler models at earlier learning steps and of more complex structures as the learning process advances, thus avoiding the problems caused by either too much bias or too much variance. Starting with the simple NB, we use simple heuristics based on the performance's dynamics to decide about the next move in the spectrum of feature dependencies. This bias management attempts to select models with the appropriate complexity for the current amount of data, thus balancing the computational cost of updating a model with the benefits of increasing in accuracy.

We choose the class of k-**D**ependence **B**ayesian **C**lassifiers (k-DBC) for illustrating our approach. A k-DBC [11] is a Bayesian Network, which contains the structure of NB and allows each feature to have a maximum of k feature nodes as parents. This class is very suitable for our proposal. By varying k we can obtain classifiers that move smoothly along the spectrum of feature dependencies, thus providing a flexible control over the model's complexity. For instance, NB is a 0-DBC, TAN is a 1-DBC, etc. Although the adaptive algorithm is presented here for the family of k-DBC classifiers, we believe that its underlying principles can be easily adapted for learning other classifier's classes with flexible control over their complexity.

This paper is organized as follows: Section 2 briefly reviews the problem of learning Bayesian Network Classifiers and provides the learning algorithm for the class of k-DBCs. In Section 3 we describe our adaptive algorithm in an on-line learning framework. Next, in Section 4 we describe the experiments we conducted that demonstrate the effectiveness of our adaptive approach. Finally, in Section 5 we conclude with a summary of the paper.

2 Learning k-Dependence Bayesian Classifiers

Bayesian Networks (BNs) are probabilistic graphical models that provide a sound theoretical framework to represent and manipulate probabilistic dependencies in a domain. Formally, a BN over a set of random variables $\mathbf{X} = \{X_1, X_2, , X_n\}$ is a pair $BN = (S, \mathbf{\Theta_S})$ where the first component is a directed acyclic graph with a node for each variable and the second component is the set of parameters that quantifies the network. The arcs in the structure S represent direct dependencies between variables. Assuming discrete variables, each $\Theta_i = P(X_i|\mathbf{Pa_i}) \in \mathbf{\Theta_S}$ represents a conditional probability table that enumerates the probabilities for each possible value $x_k \in X_i$ and $pa_j \in \mathbf{Pa_i}$ where $\mathbf{Pa_i}$ represents the set of parents of X_i.

In classification problems the domain variables are partitioned into features $\mathbf{F} = \{X_1, X_2, , X_n\}$ and the class variable C. A NB classifier is a BN over $(\mathbf{F} \cup \{C\})$ with a simple structure that has the class node C as the parent node of all other feature nodes in $\mathbf{F}$. A k-DBC [11] is a BN which contains the structure of NB and allows each feature X_i to have a maximum of k feature nodes as parents. k-DBCs represents in a single class, a full spectrum of allowable dependence in a given probabilistic model with the NB classifier at the most restrictive end and the full Bayesian Network at the most general extreme.

Suppose we have a set $\mathcal{M}$ of BNC model classes (e.g. NB, TAN, unrestricted BNs, etc.) and a training dataset $\mathcal{D}$ of labelled i.i.d. examples of (F, C). Since the quality of a BNC is defined in terms of its *predictive accuracy*, given the data $\mathcal{D}$ and the set $\mathcal{M}$ of BNC's hypotheses, the problem of learning BNCs is to find a BNC that provides the best classifications for future data. This learning problem - a *model selection problem* - can be approached as a discrete optimization problem where a score that measures the quality of each hypothesis is optimized in the space of feasible hypotheses. A procedure to solve this discrete optimization problem is essentially a search algorithm that explores the space of candidate hypotheses while optimizing the score. In most cases, the search space is large and an exhaustive search is impractical. One way to handle this problem is to develop heuristic search algorithms that search for a solution which is reasonably close to optimal.

There are three main factors that affect the performance of score-based approaches to model selection: the *score* and the *search method* used to learn the structure and the *estimator* used to learn the parameters. Next we describe these three factors in learning k-DBCs.

2.1 Search Algorithm

Instead of using the learning algorithm proposed by Sahami [11] based on the computation of the conditional mutual information, we apply, in conjunction with a score, a Hill Climbing procedure. Hill Climbing improves the score by *adding/removing/reversing* arcs among attributes, subject to never introducing a cycle. This process continues until no more operations improve the score.

2.2 Scores

When BNs are used for classification, we are interested in the resulting predictive distribution yielding accurate predictions for future data. We compare three frequently used scores in learning BNs: BDeu, MDL and AIC; to the prequential score (Preq) as described in [7]. BDeu, MDL and AIC are optimized for a particular loss function based on the joint distribution while Preq is optimized for classification. AIC and MDL are both derived from information theory and prefer simpler models. Minimizing AIC is approximately equivalent to minimizing the expected K-L divergence between the true model and the fitted model. MDL principle attempts to describe the data using a minimum encoding approach. BDeu is a Bayesian score, the marginal likelihood with uniform priors as proposed in

[3]. Preq is computed *predictively* and *sequentially* through a sequential updating of the predictive distribution. Alternative structures are compared by measuring their cumulative loss. While it is known that standard scores perform worse in classification than scores based on the classification error (e.g. see [7]), we are more interested in investigating how different scores handle the *bias-variance*, *complexity-performance* trade-offs in incremental learning of k-DBCs.

2.3 Parameter Estimation

We use the Bayesian estimates for parameters as described in [1]. In addition, we optionally use an extended version of Iterative Bayes (IB) [5] for parameter refinement. IB iteratively cycles through all the given examples. For each example, the corresponding entries in the contingency tables are updated so as to increase the confidence on the correct class. The procedure is given an initial error for the given examples. The iterative process stops when one of the following cases occurs: *i*) the current error is greater than the previous error; *ii*) the error remains stationary for three consecutive times; *iii*) a maximum number of allowed iterations is reached. In any of the cases, the model returned is that which attains the best results during the whole iterative process.

3 The Adaptive Algorithm for Learning k-DBCs

In this section we describe our adaptive algorithm for learning k-DBCs in an online framework. We provide our algorithm with a dataset of labelled examples and the kMax value for the maximum allowable number of feature dependencies. We assume the environment is stationary, data arrives in batches and a unique k-DBC model is maintained. The pseudo-code for the algorithm is presented in Figure 1. At each learning step, the learner accepts a batch of examples and classifies them using the current model. Next, the current performance is assessed and the model is then adapted according to the estimated performance' state.

An efficient adaptive algorithm for supervised learning must be able, above all, to improve its predictive accuracy over time, while minimizing the cost of updating. BNs suffer from several drawbacks for updating purposes. While sequential updating of the parameters is straightforward (if data is complete); updating the structure is a more costly task. In previous work different approaches have been carried out in incremental learning of BNs by optimizing the learning algorithms and/or the memory space (see [10] for a survey). The basic idea of our approach is that we can improve the performance while reducing the cost of updating if: *i*) in each learning step we choose a model with the appropriate complexity for the amount of training data we have; *ii*) we try to use new data to primarily adapt the parameters and only when it is really necessary to adapt the structure. As a result, our strategy leads to the selection of simpler models at earlier learning steps and gradually increases the model's complexity as more and more data becomes available. This bias control attempts to avoid *overfitting*

```
procedure AdaptiveOnlinekDBCs(data,kMax)
init: model   InitAsNaiveBayes()
for each new incoming batch of examples
    predictions <- predict(model,batch)
    observed <- getFeedback(batch)
    performanceState <- assesPerformance(predictions, observed)
    if (performanceState != IS_SATISFACTORY)then
        adapt(model, examples, FIRST_LEVEL)
    if (performanceState == STOP_IMPROVING) then
        adapt(model, examples, SECOND_LEVEL)
        if (Not change(model.structure)) then
           adapt(model, examples, THIRD_LEVEL)
end for
end procedure
```

Fig. 1. Pseudo-code for the adaptive on-line algorithm for k-DBCs

or *underfitting* of the current model to the actual data. Next, we describe the two main aspects of our adaptive algorithm: the *adaptation policy* and the *control policy*.

3.1 Adaptation Policy

The *adaptation policy* is characterized by a gradual adaptation of the model using three levels so that increasing the adaptation level increases the cost of updating:

- FIRST LEVEL: only the parameters are updated with new data
- SECOND LEVEL: the current structure is adapted by searching for new dependencies among attributes
- THIRD LEVEL: if it is still possible, the maximum number of allowable dependencies is increased by one, and the current structure is once again adapted.

The pseudo-code for the adaptation procedure is presented in Figure 2. In the absence of any information about the true model underlying the data, we initialize the classifier to the simple NB ($k = 0$). Whenever we obtain new data, we first try to improve NB by adapting only its parameters. Only when we obtain some evidences indicating that the performance of the NB stops improving in the desirable tempo, we move to a more costly level of adaptation: adapting the structure. We increment k by one and start searching a 1-DBC by finding 1-dependence among attributes. At this time point, we must have more data available which allows the search procedure to find new dependencies. Next, the algorithm continues to perform only parameter adaptation, until there will be again evidences that the performance of the current classifier

```
procedure Adapt(model, examples, level)
   switch level:
     case FIRST_LEVEL:
       performAdaptation(model, examples, UPDATE_PARAMETERS)
       if (bUseIterativeBayes) then
         performAdaptation(model, examples, REFINE_PARAMETERS)
     case SECOND_LEVEL:
         performAdaptation(model, examples, ADAPT_STRUCTURE)
     case THIRD_LEVEL:
       if (augmentDepIsPossible(model))then
         augmentMaxNrAllowableDependencies(model)
         performAdaptation(model, examples, ADAPT_STRUCTURE)
   end switch
 return model
end procedure
```

Fig. 2. Pseudo-code for the adaptive algorithm

stops improving. In this case, we try to adapt the current structure. Only if the resulting structure remains the same, we move to the third level of adaptation by incrementing the maximum number of allowable dependencies, k, by one (if this is still possible, i.e. if $k < \mathsf{kMax}$) and searching for new dependencies. This process continues until the performance reaches the desirable level.

3.2 Control Policy

The *control policy* defines the criteria for tracking two situations: *i*) at what time point do we move from the first level of adaptation to the second level, i.e., when do we start adapting the structure? *ii*) at what time point do we stop doing any adaptation? If we detect that the performance of the current model no longer improves in a desirable tempo then we start adapting the structure. On the other hand, if we detect that the performance has already reached the desirable level, we stop adapting the model.

Assume that feedback can be obtained and that for each batch, we can evaluate the error rate e_{batch}, the proportion of misclassified examples in a batch. We monitor e_{batch} obtained for different batches as an indicator of the performance at different points in time. As stated, we initialize the structure to NB. Because of its simplicity, NB learns very quickly, which is reflected in the behavior of the batch error. At earlier learning steps, it exhibits a shorter downward trend with a steeper slope of descent. However, as time increases, the steepness of the slope will decrease, approaching zero. We use the Sen's slope estimator [12] for assessing the trend strength. At each t^{th} learning step, we use only the

most recent p batch errors for dynamically assessing the decreasing slope (we set p to 5). To estimate the Sen's slope we compute the slopes of each pair of observed errors $e_{batch}[t_i], e_{batch}[t_j]$ for $(t_i > t_j)$ where the slope is defined as $(e_{batch}[t_i] - e_{batch}[t_j])/(t_i - t_j)$. The Sen's slope is then the median value of the resulting slopes. The rule is then straightforward. If the slope is sufficiently close to zero, then we assume that the performance of the current model no longer improves:

$$IF \quad SenSlope(e_{batch}[t-p+1:t]) \leq slope_{threshold}$$

$$THEN \quad performanceState \leftarrow StopImproving$$

At subsequent learning steps it results more difficult to apply this kind of trend analysis using successive error values. Notice that as time increases, batch error values fluctuate around a certain level, decreasing slowly with a slope approaching zero. Instead, we proceed in the following way: first, the parameters are updated using new examples. Then, we once again assess e_{batch} using the adapted model. Assume that $e'_{batch}[t]$ and $e''_{batch}[t]$ are the batch errors obtained before and after adaptation, respectively. Whenever we obtain a decrease of the batch error after adaptation, it would be a straightforward idea to consider that the learner is still able to learn about the current target concept using the current model's structure. Otherwise, if for a pre-defined number of consecutive times after adaptation the error does not improve then we assume that the performance no longer improves using the current structure:

$$IF \quad consecCounter(e''_{batch}[t] \geq e'_{batch}[t]) = maxTimes$$

$$THEN \quad performanceState \leftarrow StopImproving$$

Further model adaptations will continue until the performance reaches the desirable level. Given a threshold level for the batch error, we assume that the performance is satisfactory if for a fixed number of consecutive times $e_{batch} \leq error_{threshold}$.

4 Empirical Study

Primarily, we want to investigate if our adaptive algorithm is able to scale up the model's complexity of k-DBCs while improving its performance over time. With this aim, we carried out an empirical study for evaluating the performance of k-DBCs and NB induced incrementally from scratch against our adaptive approach for four scores on three large datasets.

4.1 Experimental Setup

We used two underlying learning algorithms to induce k-DBCs: NB ($k = 0$) and hill-climbing ($k > 0$) with BDeu, MDL, AIC and Preq as described in section 2. We used only arc additions and deletions. All the learning algorithms were implemented using Weka's classes for BNCs ([1],[13]). Since we use different

scores for the same learning algorithm, this helps in ensuring that any differences in performance are due to the differences in the scores, and not to differences in the underlying algorithm.

We evaluated the learning algorithms on three datasets: *balance*, *nursery* and *adult*. Since we needed datasets with large number of examples to better explore the behaviour of incremental algorithms, we randomly generated artificial large samples of 10000 examples for balance using its well-known underlying rules. We used the nursery dataset from the UCI repository and a discretized version of the adult dataset available on-line at *http://www.cs.helsinki.fi/u/p̃kontkan/Data/*. We removed instances with missing values from the datasets. Thus, we used 12800 instances for nursery (128 learning steps) and 16000 instances for adult (160 learning steps), respectively.

We evaluated two versions of the AdaptiveOnlinekDBCs algorithm. Unlike Adap1, Adap2 additionally implements IB (section 2.3). We set kMax=3 for balance, kMax=5 for *nursery* and adult, $slope_{threshold} = error_{threshold} = 0$ and $maxTimes = 3$. To serve as baselines of our adaptive algorithms, we evaluated the performance of NB and k-DBCs (varying k), inducing them incrementally from scratch: at the t^{th} learning step we used the first t batches as training data and the examples of the next $(t+1)^{th}$ batch as test data. We use batches of 100 examples. At each learning step, the performance was measured as the average of the accuracy over 10 runs.

4.2 Cost of Updating vs. Performance

Table 4 compares the relative significant gains of the predictive accuracy averaged over 10 runs of k-DBCs and Adap1-2 in conjunction with the four scores with respect to NB at different learning steps. Figure 3 compares the performance over time of all the algorithms for the three datasets. Table 1 shows the number of adaptations performed in the structure per data set, score and adaptive algorithm.

In most cases results show that adaptive algorithms perform at least as well as the best k-DBC at each learning step. In general, Adap1-2 significantly improve the performance of NB over time while reducing the cost of updating, as it is shown by the small number of adaptations performed in the structure during the whole learning process. However, the best results were obtained with the *nursery* and the *adult* datasets. Note that for *balance*, as time increases, the best model for all the scores, except for MDL, is a 3-DBC. Since the balance domain is easier to learn, adaptive algorithms can get trapped in less complex structures than the optimal one while progressing to improve their performances. Unlike balance, for nursery and adult the best results are obtained with Adap2. Moreover, for all the scores and datasets the number of adaptations performed in the structures using Adap2 is considerably less than using Adap1 (see Table 1). As it was shown in [5], the reduction of the error rate observed with IB is mainly due to a reduction on the bias component, which explains the obtained results. This means that Adap2 ensures the best balance between the cost of updating and the gain in performance.

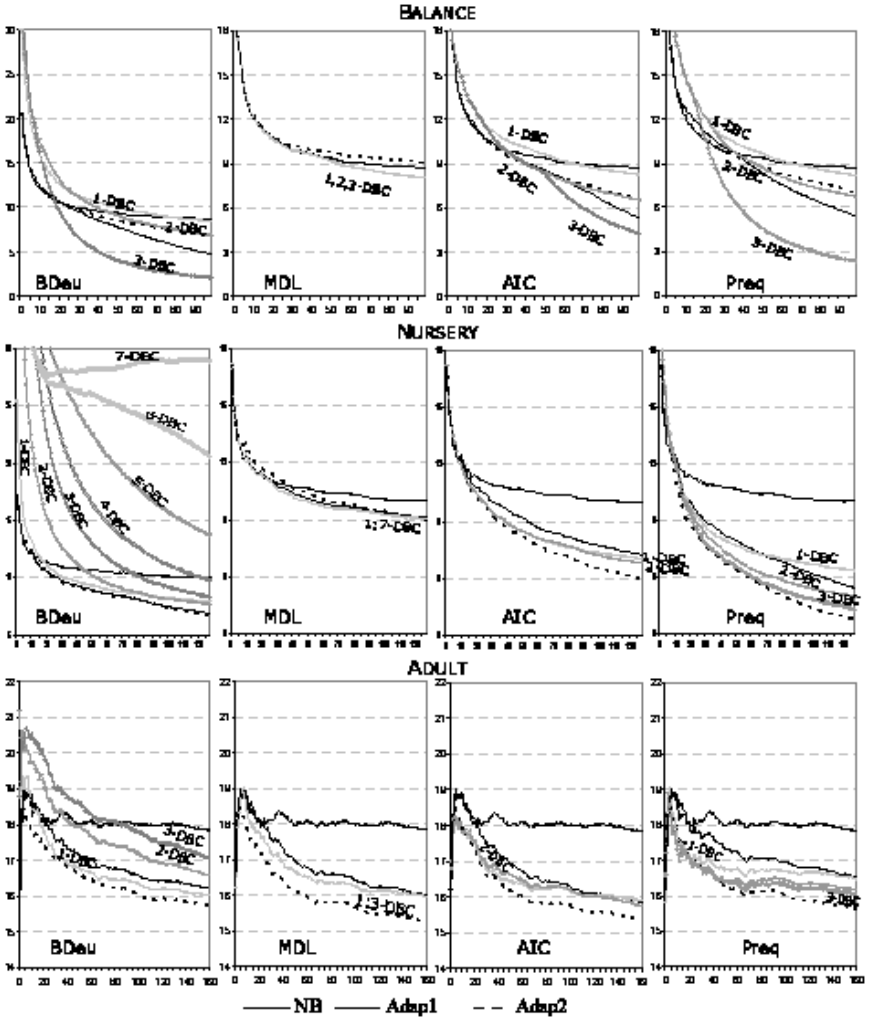

Fig. 3. Error rate for NB, k-DBCs and Adap1-2 for the four scores over time. At the t^th learning step, # training examples $= 100 * t$; # test examples $= 100$.

4.3 Model Complexity vs. Performance

There is a bias-variance trade-off in choosing the appropriate complexity of the model. We used the bias-variance decomposition of the error as proposed in [6] to investigate how different scores handle the bias-variance trade-off in incremental learning of k-DBCs. Due to space limitations we only show the results on the *nursery* dataset at two point times: $t = 10$ and $t = 120$ in figure 4. Plots on the left show the training and test errors as a function of k-DBC models. Each point in a line represents the training or test error of the related k-DBC for

Table 1. Number of adaptive actions per data set, score and adaptive algorithm

		balance		nursery		adult	
Score	Algor.	Ref.Str.	Aug.Dep	Ref.Str.	Aug.Dep	Ref.Str.	Aug.Dep
BDeu	Adap1	3.0±0.0	3.0±0.0	3.2±0.5	3.2±0.5	1.8±0.5	1.8±0.5
	Adap2	2.1±0.9	2.1±0.9	3.4±0.6	3.4±0.6	1.2±0.5	1.2±0.5
MDL	Adap1	16.7±0.0	3.0±0.0	23.8±2.8	5.0±0.0	14.0±2.0	4.0±0.0
	Adap2	1.2±0.6	1.1±0.3	15.6±3.8	5.0±0.0	3.8±0.8	2.4±1.6
AIC	Adap1	3.7±0.7	3.0±0.0	14.4±2.0	5.0±0.0	4.6±1.1	3.4±0.9
	Adap2	2.2±0.9	2.2±0.9	13.2±2.1	5.0±0.0	2.4±0.6	1.6±0.6
Preq	Adap1	3.5±0.5	3.0±0.0	5.0±1.9	3.6±1.1	3.4±0.6	2.8±0.5
	Adap2	2.4±1.2	2.2±0.9	5.2±1.9	3.8±1.1	1.6±0.6	1.4±0.6

a particular score. The first points in the lines represent the errors of NB. In each learning step, given a particular score there is an optimal model class that gives minimum test error. For Preq and BDeu, the optimal models are 1-DBC at $t = 10$ and 3-DBC at $t = 120$, respectively. For MDL and AIC, all k-DBCs present identical results starting from some k. Results in Table 3 suggest that found models are all identical.

Pictures on the right show the *bias-variance* decomposition of the test error for all scores. Results show that varying k, the score and the training set size can have different effects on bias and variance. The best results were obtained with Preq due to a more optimal bias management. On the other hand, BDeu consistently favors the dependent structure over the independent one, thus finding maximal models (see Table 3). As you can see, if the class model is more

Table 2. The values of k averaged over 10 runs at $t = 10$ and $t = 120$ for the nursery dataset

	Adap1				Adap2			
t	BDeu	MDL	AIC	Preq	BDeu	MDL	AIC	Preq
10	0.8	0.8	0.8	0.8	1	1	1	1
120	3.2	5	5	3.6	3.2	5	5	3.4

Table 3. The number of added arcs to the NB structure averaged over 10 runs for the nursery dataset at two time points. K1-7 represent 1-7-DBCs, A1-2 - Adap1, Adap2, respectively

	1000 training examples									12000 training examples								
Score	K1	K2	K3	K4	K5	K6	K7	A1	A2	K1	K2	K3	K4	K5	K6	K7	A1	A2
BDeu	7	13	18	22	25	27	28	4.2	5.6	7	13	18	22	25	27	28	18.8	18.6
MDL	0	0	0	0	0	0	0	0	0	2	2	2	2	2	2	2	2	2
AIC	1.4	1.4	1.4	1.4	1.4	1.4	1.4	0.8	1.2	7	9.4	9.4	9.4	9.4	9.4	9.4	10.4	9.8
Preq	6.4	8	8	8	8	8	8	3.4	4.2	7	12.6	17.4	18	18.4	18.4	18.4	17.6	17.8

Table 4. Relative significative gains of the predictive accuracy of k-DBCs, Adap1-2 with respect to NB. (+) indicates significative gain using a paired t-test at the 5% level , *no* indicates there is no significative gain.

Balance		10	20	30	40	50	60	70	80	90	100
	NB	87.52	88.62	89.04	89.22	89.34	89.52	89.6	89.66	89.81	89.87
BDeu	1-DBC	no	no	no	no	no	no	no	no	no	(+)0.29
	2-DBC	no	no	no	no	no	(+)0.66	(+)1.06	(+)1.35	(+)1.63	(+)1.86
	3-DBC	no	(+)0.91	(+)3.14	(+)4.5	(+)5.31	(+)5.67	(+)6.06	(+)6.29	(+)6.52	(+)6.65
	Adap1	no	no	no	(+)0.99	(+)1.57	(+)2.03	(+)2.61	(+)3.12	(+)3.6	(+)3.95
	Adap2	no	no	no	(+)0.53	(+)0.86	(+)1.03	(+)1.25	(+)1.52	(+)1.82	(+)2.11
MDL	1-DBC	no	no	no	no	(+)0.19	(+)0.3	(+)0.44	(+)0.55	(+)0.66	(+)0.72
	2-DBC	no	no	no	no	(+)0.19	(+)0.3	(+)0.44	(+)0.55	(+)0.66	(+)0.72
	3-DBC	no	no	no	no	(+)0.19	(+)0.3	(+)0.44	(+)0.55	(+)0.66	(+)0.72
	Adap1	no	no	no	no	no	(+)0.28	(+)0.43	(+)0.55	(+)0.65	(+)0.71
	Adap2	no	no	no	no	no	no	no	no	no	no
AIC	1-DBC	no	no	no	no	no	no	no	(+)0.28	(+)0.42	(+)0.5
	2-DBC	no	no	no	(+)0.55	(+)0.98	(+)1.26	(+)1.58	(+)1.81	(+)2.01	(+)2.2
	3-DBC	no	no	no	(+)0.55	(+)1.04	(+)2.08	(+)2.98	(+)3.6	(+)4.13	(+)4.48
	Adap1	no	no	no	(+)0.45	(+)0.83	(+)1.14	(+)1.74	(+)2.32	(+)2.95	(+)3.4
	Adap2	no	no	no	(+)0.6	(+)0.9	(+)1.12	(+)1.36	(+)1.64	(+)1.93	(+)2.19
Preq	1-DBC	no	no	no	no	no	no	no	(+)0.27	(+)0.46	(+)0.57
	2-DBC	no	no	no	no	(+)0.65	(+)1.01	(+)1.33	(+)1.6	(+)1.84	(+)2.03
	3-DBC	no	no	(+)2.25	(+)3.82	(+)4.77	(+)5.22	(+)5.67	(+)5.95	(+)6.22	(+)6.38
	Adap1	no	no	no	no	(+)1.01	(+)1.47	(+)2.05	(+)2.49	(+)2.93	(+)3.34
	Adap2	no	no	no	no	(+)0.63	(+)0.82	(+)1.04	(+)1.17	(+)1.45	(+)1.77

Nursery		10	20	30	40	50	60	80	100	110	128
	NB	87.52	88.62	89.04	89.22	89.34	89.52	89.66	89.97	89.98	89.96
BDeu	1-DBC	no	no	(+)0.93	(+)1.12	(+)1.36	(+)1.51	(+)1.75	(+)1.90	(+)1.98	(+)2.18
	2-DBC	no	no	no	no	(+)0.45	(+)0.97	(+)1.64	(+)2.00	(+)2.12	(+)2.35
	3-DBC	no	no	no	no	no	no	no	(+)0.95	(+)1.23	(+)1.73
	Adap1	no	(+)0.80	(+)1.19	(+)1.46	(+)1.70	(+)1.84	(+)2.25	(+)2.65	(+)2.83	(+)3.16
	Adap2	no	(+)0.91	(+)1.38	(+)1.62	(+)1.86	(+)1.97	(+)2.26	(+)2.77	(+)3.02	(+)3.39
MDL	1-DBC	no	no	(+)0.32	(+)0.51	(+)0.71	(+)0.77	(+)0.82	(+)0.84	(+)0.88	(+)1.02
	2-DBC	no	no	(+)0.32	(+)0.51	(+)0.71	(+)0.77	(+)0.82	(+)0.84	(+)0.88	(+)1.02
	3-DBC	no	no	(+)0.32	(+)0.51	(+)0.71	(+)0.77	(+)0.82	(+)0.84	(+)0.88	(+)1.02
	Adap1	no	no	no	no	(+)0.51	(+)0.6	(+)0.68	(+)0.73	(+)0.77	(+)0.92
	Adap2	no	no	no	no	(+)0.28	(+)0.42	(+)0.56	(+)0.66	(+)0.77	(+)1.05
AIC	1-DBC	(+)0.54	(+)0.89	(+)1.46	(+)1.87	(+)2.19	(+)2.38	(+)2.70	(+)2.80	(+)2.86	(+)3.02
	2-DBC	(+)0.54	(+)0.89	(+)1.46	(+)1.87	(+)2.19	(+)2.39	(+)2.66	(+)2.95	(+)3.03	(+)3.23
	3-DBC	(+)0.54	(+)0.89	(+)1.46	(+)1.87	(+)2.19	(+)2.39	(+)2.66	(+)2.95	(+)3.03	(+)3.23
	Adap1	no	(+)0.58	(+)0.91	(+)1.12	(+)1.47	(+)1.78	(+)2.15	(+)2.39	(+)2.50	(+)2.78
	Adap2	no	(+)1.09	(+)1.61	(+)2.10	(+)2.44	(+)2.78	(+)3.22	(+)3.62	(+)3.76	(+)4.10
Preq	1-DBC	no	(+)1.59	(+)2.31	(+)2.72	(+)2.96	(+)3.13	(+)3.35	(+)3.46	(+)3.53	(+)3.72
	2-DBC	no	(+)1.86	(+)2.69	(+)3.26	(+)3.75	(+)4.06	(+)4.50	(+)4.74	(+)4.82	(+)5.05
	2-DBC	no	(+)2.06	(+)3.14	(+)3.64	(+)4.10	(+)4.42	(+)4.99	(+)5.28	(+)5.41	(+)5.73
	Adap1	(+)0.34	(+)1.38	(+)1.98	(+)2.37	(+)2.70	(+)2.96	(+)3.49	(+)3.94	(+)4.15	(+)4.62
	Adap2	no	(+)2.56	(+)3.33	(+)3.80	(+)4.24	(+)4.58	(+)5.29	(+)5.76	(+)5.93	(+)6.29

Adult		10	20	40	60	80	90	100	120	140	168
	NB	81.14	81.91	81.69	81.89	82.00	81.93	81.99	82.01	82.04	82.16
BDeu	1-DBC	no	(+)0.15	(+)1.34	(+)1.46	(+)1.54	(+)1.62	(+)1.65	(+)1.80	(+)1.86	(+)1.82
	2-DBC	no	no	no	no	(+)0.59	(+)0.70	(+)0.80	(+)1.08	(+)1.17	(+)1.26
	3-DBC	no	no	no	no	no	no	no	(+)0.48	(+)0.63	(+)0.75
	Adap1	no	no	(+)0.91	(+)1.08	(+)1.24	(+)1.35	(+)1.41	(+)1.56	(+)1.61	(+)1.62
	Adap2	(+)0.98	(+)0.49	(+)1.43	(+)1.62	(+)1.77	(+)1.86	(+)1.93	(+)2.09	(+)2.11	(+)2.09
MDL	1-DBC	no	(+)0.34	(+)1.36	(+)1.58	(+)1.66	(+)1.72	(+)1.74	(+)1.86	(+)1.91	(+)1.87
	2-DBC	no	(+)0.34	(+)1.36	(+)1.59	(+)1.66	(+)1.72	(+)1.75	(+)1.88	(+)1.90	(+)1.87
	3-DBC	no	(+)0.34	(+)1.36	(+)1.59	(+)1.66	(+)1.72	(+)1.75	(+)1.88	(+)1.90	(+)1.87
	Adap1	no	no	(+)0.95	(+)1.31	(+)1.40	(+)1.51	(+)1.55	(+)1.75	(+)1.81	(+)1.83
	Adap2	(+)1.00	(+)0.78	(+)1.86	(+)2.14	(+)2.17	(+)2.27	(+)2.32	(+)2.52	(+)2.57	(+)2.57
AIC	1-DBC	(+)1.06	(+)0.60	(+)1.71	(+)1.77	(+)1.78	(+)1.79	(+)1.84	(+)1.96	(+)1.98	(+)1.96
	2-DBC	(+)1.04	(+)0.55	(+)1.51	(+)1.65	(+)1.73	(+)1.77	(+)1.82	(+)1.99	(+)2.07	(+)2.10
	3-DBC	(+)1.04	(+)0.55	(+)1.51	(+)1.65	(+)1.73	(+)1.77	(+)1.82	(+)1.99	(+)2.07	(+)2.10
	Adap1	no	no	(+)1.08	(+)1.39	(+)1.56	(+)1.66	(+)1.70	(+)1.89	(+)1.97	(+)2.00
	Adap2	(+)0.94	(+)0.61	(+)1.86	(+)2.11	(+)2.17	(+)2.24	(+)2.25	(+)2.42	(+)2.49	(+)2.48
Preq	1-DBC	no	(+)0.92	(+)1.41	(+)1.35	(+)1.28	(+)1.33	(+)1.31	(+)1.35	(+)1.37	(+)1.37
	2-DBC	(+)1.44	(+)1.06	(+)1.67	(+)1.66	(+)1.54	(+)1.58	(+)1.54	(+)1.67	(+)1.69	(+)1.68
	3-DBC	(+)1.52	(+)1.00	(+)1.71	(+)1.77	(+)1.67	(+)1.72	(+)1.68	(+)1.79	(+)1.81	(+)1.78
	Adap1	no	no	(+)0.64	(+)0.85	(+)0.93	(+)1.01	(+)1.03	(+)1.19	(+)1.26	(+)1.30
	Adap2	(+)1.00	(+)0.93	(+)1.70	(+)1.91	(+)1.88	(+)1.93	(+)1.97	(+)2.14	(+)2.19	(+)2.22

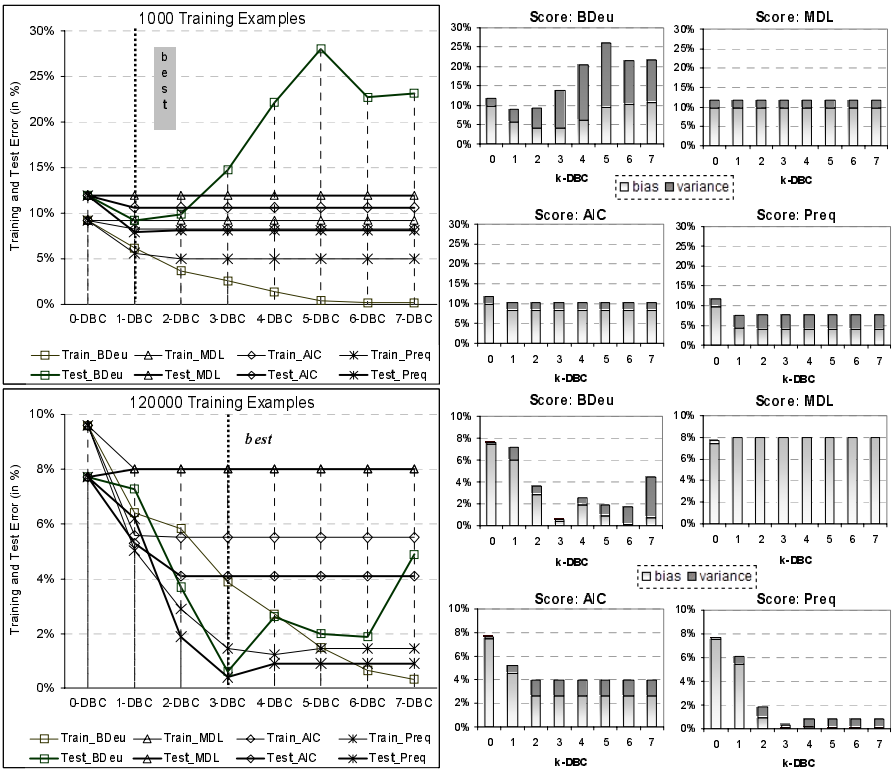

Fig. 4. Training-test errors and bias-variance decompositions of k-DBCs varying k at two time points

complex than the optimal model, BDeu leads to severe overfitting due to increase in variance. However, as training set size increases, the variance decreases for all k-DBCs, thus reducing the test error and the overfitting problem. In contrast, MDL and AIC find models simpler than the optimal model, thus underfitting the data. Both scores increase in bias, especially, if the class model is less complex than the optimal model. Because MDL penalizes complexity more severely that AIC does, the model with the least MDL will tend to be simpler than the model with the most AIC. Notice that as training size increases AIC reduces the bias slightly. On the contrary, MDL increases it. As a result AIC outperforms MDL.

To provide evidences toward the hypothesis that our adaptive algorithm attempts to select the appropriate complexity of the model (i.e. the optimal model class) for the current amount of training data, we can look at table 2. At $t = 10$, for all scores, adaptive algorithms find a model with k approaching 1, i.e., a 1-DBC. At $t = 120$, for BDeu and Preq, they find a model with k approaching 3 (a 3-DBC). For MDL and AIC, they find a model with $k =$kMax, i.e., a 5-DBC. These results are consistent with the optimal model classes that we have found

previously. Note that optimal k-DBCs present the lowest biases. Finally, results in Table 1 evidence that the number of adaptations performed in the structure for BDeu was always minimal when compared with other scores. On the contrary, the number of adaptations performed in the structures using MDL was always maximal. These results reflect the efforts made by our adaptive algorithms to control the overfitting and underfitting problems.

5 Conclusions

We have examined a practical adaptive learning algorithm for improving the performance of BNCs over time. The main idea is to scale up the model's complexity as training data increases by gradually increasing the number of allowable dependencies among features. This allows reducing both bias and variance and consequently the classification error. Starting with the simple NB, we use simple decision rules based on the performance dynamics to decide on the next move in the spectrum of feature dependencies and search for a more complex model. Therefore, as training set size increases, bias will decrease because we choose a more complex model and variance will also decrease because we use more examples to learn. Results in conducted experiments using the class of k-DBCs and a hill climbing learning algorithm in conjunction with four scores on three large datasets show that our adaptive algorithm in combination with IB performs an artful bias management for choosing the appropriate complexity of the model.

Our adaptation policy is characterized by a gradual adaptation of the model using three levels so that increasing the adaptation level increases the cost of updating. We attempt to use new data to primarily adapt the parameters and only if this is really necessary, to adapt the structure. Since updating the structure is a costly task, this way we reduce the computational cost of updating while improving the performance. Results in conducted experiments show that adaptive algorithms significantly improve the performance of NB over time and that they perform no worse than the best k-DBC while reducing the cost of updating as it is shown by the small number of adaptations performed on the structure during the whole learning process in contrast to the great number of adaptations performed on the structure of k-DBCs when they were induced incrementally from scratch.

Although we have used only three datasets for evaluation, the results obtained here encourage us to continue this work thus to be able to improve the adaptation and control policies involved in our adaptive algorithm. One of the crucial question that we will focus in the future will be to investigate several criteria for determining when we should stop the learning process according to the observed performance, instead of using a given threshold level for the batch error. Future work will also involve additional experimentation with more large datasets in order to obtain more evidences on the effectiveness of our adaptive system.

Finally, although the adaptive algorithm is presented here for the family of k-DBC classifiers, we believe that its underlying principles can be easily adapted

for learning other classifier's model classes (e.g. decision trees [8], neural networks using different topologies [9]) with a *hierarchical* and *increasing* control over their complexity.

Acknowledgments

Thanks to the financial support given by the FEDER, the Plurianual support attributed to LIACC, and project ALES II (POSI/EIA/55340/2004).

References

1. Bouckaert, R.: Bayesian Network Classifiers in Weka (2004), Technical Report 14/2004. Computer Science Department. University of Waikato. (2004)
2. Brian, D., Webb, G.: The need for Low Bias Algorithms in Classification Learning from Large Data Sets, In Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD 2002), Springer-Verlag (2002) 62: 73
3. Buntine, W.: Theory Refinement on Bayesian Networks. In Proceedings of the 7th Conference on Uncertainty in Artificial Intelligence, (1991) **52**: 60 4.
4. Friedman, N., Geiger, D. and Goldszmidt, M.: Bayesian Network Classifiers. Machine Learning **4** (1997) 131:161 5.
5. Gama, J.: Iterative Bayes. In Intelligent Data Analysis, **4** (2000) 475:488, IOS Press 6.
6. Kohavi, R., Wolpert, D.: Bias Plus Variance Decomposition for Zero-One Loss Functions. In Proceedings of the 13th International Conference on Machine Learning (ICML'96), Morgan Kaufmann Publishers, (1999) 7.
7. Kontkanen, P., Myllymaki, P., Silander, T., Tirr, H: On Supervised Selection of Bayesian Networks. In Proceedings of the Fifteenth International Conference on Uncertainty in Artificial Intelligence (UAI'99), Morgan Kaufmann Publishers, (1999) 334:342
8. Quinlan, R.: *C4.5 Programs for Machine Learning*. Morgan Kaufmann Publishers, (1993).
9. Ripley, B.: *Pattern Recognition and Neural Networks*. Cambridge University Press, (1996).
10. Roure, J., Sangüesa,R.: Incremental Methods for Bayesian Network Learning. Research Report LSI-99-42-R. Software Department. Technical University of Catalonia, (1999).
11. Sahami, M.: Learning Limited Dependence Bayesian Classifiers, In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, AAAI Press, Portland, OR, (1996) 335:338 10.
12. Sen, P.K.: Estimates of the regression coefficient based on Kendall's tau. In Journal of the American Statistical Association. **63** (1968) 1379:1389
13. Witten,I.H., Frank, E.: Data Mining: Practical machine learning tools and techniques, 2nd Edition, Morgan Kaufmann, San Francisco, (2005).