Adaptive Bayesian Network Classifiers

Gladys Castillo\(^1\) and João Gama\(^2\)

\(^1\) Department of Mathematics & CEC, University of Aveiro
Campus Santiago, 3810-193 Portugal
phone: (351) 234370359, gladys@ua.pt

\(^2\) FEP, LIADD, University of Porto, Portugal
jgama@liadd.up.pt

Abstract

This paper is concerned with adaptive learning algorithms for Bayesian network classifiers in a prequential (on-line) learning scenario. In this scenario, new data is available over time. An efficient supervised learning algorithm must be able to improve its predictive accuracy by incorporating the incoming data, while optimizing the cost of updating. However, if the process is not strictly stationary, the target concept could change over time. Hence, the predictive model should be adapted quickly to these changes. The main contribution of this work is a proposal of an unified, adaptive prequential framework for supervised learning called AdPreqFr4SL, which attempts to handle the cost-performance trade-off and deal with concept drift. Starting with the simple Naive Bayes, we scale up the complexity by gradually increasing the maximum number of allowable attribute dependencies, and then by searching for new dependences in the extended search space. Since updating the structure is a costly task, we use new data to primarily adapt the parameters. We adapt the structure only when is actually necessary. The method for handling concept drift is based on the Shewhart P-Chart. We experimentally prove the advantages of using the AdPreqFr4SL in comparison with its non-adaptive versions.

1 Introduction

During the past several years there has been an explosive growth of methods for learning Bayesian Network Classifiers (BNCs) from data (pioneer works are [15, 10, 20]). Nevertheless, most of them are implemented in a batch learning scenario, where all training examples are given at the same time to the learning system and the induced classifier is not revised to properly handle future data. However, nowadays, in many current real-world applications learning algorithms should act in dynamic environments where the data flows continuously. On-line learning is a technique appropriate to deal with this continuous increase of data. This paper is concerned with on-line learning frameworks. Data arrives at the learning system sequentially. The actual decision model must first make
a prediction and then update the current model with new data. This philosophy about on-line learning paradigms has been exposed by Dawid in his prequential approach for statistical validation of models [11].

Since the quality of a BNC is determined by its predictive capability, an efficient learning algorithm for BNCs in a prequential framework must be able, above all, to improve its predictive accuracy over time while optimizing the cost of updating. However, Bayesian networks suffer from several drawbacks for sequential updating. While sequential updating of the parameters is straightforward (if data is complete), updating the structure with new data is a computationally expensive task (it involves a search in the space of candidate structures). We can reduce the cost of updating if we try to use new data to primarily adapt the parameters, while adapting the structure only when actually necessary to maintain a desirable model quality. Nevertheless, in many real-world situations, it may be difficult to adapt and improve to existing changing environments. This problem is known as concept drift, which refers to unforeseen changes in the distribution underlying the data that can lead to changes in the target concept [37]. Concept drift scenarios require adaptive algorithms, able to track such changes and to quickly adapt to them. Learning systems that track concept drift are often called adaptive systems. Many adaptive systems make use of regular model updates while new data arrives. However, a better approach is to provide the system with some control mechanisms aimed at selecting the best adaptive actions according to the current learning goal.

The main purpose of this work is the development of adaptive algorithms for BNCs in a prequential learning scenario, which try to handle the cost-performance trade-off and cope with concept drift. We integrate all the adaptive algorithms into an adaptive prequential framework for supervised learning called AdPreqFr4SL. The adaptive strategy that we follow is based upon two main policies: adaptation control and bias management. The motivations for bias control, along with some results of its application, were first presented in [7]. The method for handling concept drift is based on Statistical Quality Control and was first introduced in [9]. The AdPreqFr4SL was first described in [8].

The rationale of our adaptive strategy is as follows. Instead of choosing a particular class of BNCs and use it during all the learning process, we propose to start with the simpler Naive Bayes (NB) classifier [14]. Then, we use some control strategies to decide when to use a more complex class-model for model selection. This regularization must lead to the selection of simpler class-models when we have few data and of more complex ones as training data increases thus reducing both bias and variance and consequently the classification error.

Since updating the structure of BNCs is a costly task, we reduce the cost of updating during the whole learning process by first adapting parameters. We trigger adaptation on the structure only when there is evidence that the performance stops improving in a desirable tempo. If during the monitoring process a concept drift is detected, some actions to adapt the learner to these changes are taken. Finally, we stop doing any adaptation when there is evidence that the use of more training data will not result in significantly improved performance. Nevertheless, if any significant change in the performance is further observed,
then the adaptation procedures are once again activated.

We choose the class of $k$-Dependence Bayesian Classifiers ($k$-DBC) [32] for illustrating our approach. **Section 2** briefly reviews the Bayesian Network Classifiers and the particular class of $k$-DBCs. **Section 3** is the core of this paper. It describes the AdPreqFr4SL as well as the adaptive and control strategies adopted to handle the cost-quality trade-off and concept drift. An overview of related work is also provided. **Section 4** gives an analysis of the experimental results that demonstrate the effectiveness of our adaptive approach. Finally, we provide the conclusions with a summary of the main contributions of this work and finish with an outlook on future research.

# Bayesian Network Classifiers

Bayesian networks[28] are probabilistic graphical models that represent the joint probability distribution of a set of random variables in a problem domain. Formally, let $\mathbf{X} = \{X_1, X_2, ..., X_n\}$ be a set of random variables for a domain. A Bayesian network (BN) over $\mathbf{X}$ is a tuple $(S, \Theta_S)$ where the first component, the **network structure** $S$ is a directed acyclic graph (DAG) whose nodes represent the random variables and whose arcs represent direct dependencies between variables; and the second component $\Theta_S$ is the set of conditional probability functions. Assuming discrete variables, each $P(X_i | Pa_i) \in \Theta_S$ represents a conditional probability table (CPT) over the values of $X_i$ given the values of its parents $Pa_i$. Moreover, the DAG $S$ satisfies the Markov condition: each node is independent of all its non-descendants given its parents in $S$. This allows the joint probability distribution over $\mathbf{X}$ to be represented in the factored form:

$$P(\mathbf{x}) = \prod_{i=1}^{n} P(X_i | Pa_i).$$

In classification problems the domain variables are partitioned into attributes $\mathbf{X} = \{X_1, X_2, ..., X_n\}$ and the class variable $C$. Each attribute $X_i$ takes values from its domain $\Omega_{X_i}$ and each instantiation $\mathbf{x} = \{x_1, x_2, ..., x_n\}$ of $\mathbf{X}$ is called an example. The class variable $C$ takes values in a finite set $\Omega_C = \{c_1, c_2, ..., c_m\}$. The aim is to correctly predict the class $c \in \Omega_C$ for a given example $\mathbf{x}$. If the performance measure is the **predictive accuracy**, the optimal prediction for $\mathbf{x}$ is the class $c^*$ that maximizes the posterior probability distribution $P(C | \mathbf{x})$.

A BN can be used for classification in a relatively straightforward way. One of the variables is selected as the class variable, and the remaining variables as attribute variables (for instance, the NB classifier is represented as a BN with a simple structure that has the class node as the parent node of all other attribute nodes). Next, inference methods can be used to calculate the marginal distribution of the class variable given the value of the attributes. Thus, a BN network can be used as a classifier that gives the posterior distribution $P(C | \mathbf{x})$ of the class node $C \in \Omega_C$ given an example $\mathbf{x}$. We can compute the posterior probability $P(c_j | \mathbf{x}, S)$ for each class $c_j \in \Omega_C$ by marginalizing the joint probability distribution $P(c_j, \mathbf{x} | S)$ and then return the class $c^*$ that maximizes it:

$$c^* = h_{BN}(\mathbf{x}) = \arg \max_{j=1 \ldots m} P(c_j, \mathbf{x} | S)$$

Given a training dataset the learning problem consists of selecting the BNC, that is, the hypothesis \( h_C = (S, \Theta_S) \), that yields the most accurate classifications for unseen data. This problem can be solved by first choosing a suitable class-model that defines the space of feasible BN structures. Next, within this class-model a structure is chosen. Finally, the parameters are estimated from data. The problem of choosing the structure is related to model selection, a subject of statistical inference concerned with the selection among a set of competing models the one that “best fits” the available data in some sense. We focus on score-based approaches to model selection where the notion of “best fits” is defined via a scoring function that measures the quality of each candidate hypothesis. Score-based approaches can be exposed as a search problem where each state in the search space identifies a possible DAG. The search method utilizes the value returned by the score to help guide the search.

In recent years the issue of the selection of an appropriate scoring function for learning BNCs has received a lot of attention [13, 15, 23]. When a BN is induced for classification, the main goal is to build an accurate classifier. Hence, it has been suggested that search strategies for learning BNCs should select among models using scores specialized for classification (supervised scores); otherwise it can result in suboptimal choices during the search process. Thus, a score based on the joint predictive distribution will not necessarily optimal for classification purposes. Other aspect that can also influence the performance of induced BNCs is the selection of an appropriate class-model that defines the search space and hence, the complexity of induced BNCs. Model selection makes a bias-variance trade-off in order to select a model with the appropriate complexity which is automatically regularized by the scoring function [19]. At each time point there is an optimal model complexity that gives the best performance on unseen data. We can obtain a desirable performance of induced BNCs if at each time point we attempt to select an appropriate class-model for the available training data.

2.1 \( k \)-Dependence Bayesian Classifiers

\( k \)-Dependence Bayesian Classifiers [32](\( k \)-DBCs) represent an unified framework for all those classes of BNCs that contain the structure of the Naïve Bayes. A \( k \)-DBC, in addition, allows each attribute to have a maximum of \( k \) attribute nodes as parents. As illustrated in Figure 1 we can vary the value of \( k \) and obtain

![Figure 1: Examples of \( k \)-Dependence Bayesian Classifiers](image)
classifiers that smoothly move along the spectrum of attribute dependencies. NB is a 0-DBC and lies at the most restrictive extreme because it strictly allows no dependencies between attributes. A TAN classifier [15] is a 1-DBCs (it allows at most 1 attribute as a parent of another attribute). The BAN [15, 10] shown in Figure 1 is a 2-DBC (it has a maximum of two dependencies among the attributes). At the most general extreme lies the full augmented NB classifier, a \((n - 1)-\)DBC, with no independence among the attributes.

Instead of implementing the learning algorithm proposed by Sahami [32] we relied on a hill-climbing learning algorithm mainly due to its incremental nature and simplicity for computational implementation. The algorithm starts with the NB’s structure. Then it iteratively adds arcs between two attributes that result in the maximal improvements in the score until there is no more improvement for that score or until it is not possible to add a new arc. Once a network structure is chosen, the parameters are estimated using the selected estimator.

**Algorithm 1** The hill-climbing algorithm for learning \(k\)-DBCs

Require: A dataset \(D\) of \(N\) labeled examples of \(<X, C>\), the \(k\) value, a scoring function \(Score(S, D)\), the space \(S\) of possible DAGs restricted by \(k\).

Ensure: A \(k\)-DBC with high value of \(Score(S, D)\).

1: Initialize \(S\) to the NB structure
2: continue \(\Leftarrow\) True
3: while continue do
4: Compute \(Score(S, D)\)
5: Find arc \((X', X'') = \arg \max \text{Score}(S \cup \{(X_i, X_j), D\}, |\text{pa}(X_i) \setminus C| < k \land |\text{pa}(X_j) \setminus C| < k)\)
6: if arc \((X', X'')\) exists \(\land\) \(\text{Score}(S \cup \{(X', X''), D\}) > \text{Score}(S, D)\) then
7: Add the arc \((X', X'')\) to \(S\)
8: else
9: continue \(\Leftarrow\) False
10: end while
11: Estimate the parameters \(\Theta_S\) given \(S\) from data \(D\)
12: return \(k\)-DBC=(\(S, \Theta_S\))

Results from Sahami’s experiments [32] and related studies using \(k\)-DBCs [1, 7, 36] show that modeling attribute dependencies can improve the classification results of the NB. As also argued by Sahami, \(k\)-DBCs are very useful for experimental purposes. By knowing how the classification performance changes with increasing \(k\) values we can get a notion of the degree of attribute dependence in each particular domain. In [6, 7] we provided the results of an experimental study with \(k\)-DBCs to evaluate how increasing the \(k\) value above 0 would affect the performance of \(k\)-DBCs induced with the Algorithm 1 using different scoring functions and increasing training data. We observed that, first, improvement was not noticed for smaller training datasets (of \(\leq 1000\) examples). For all the domains and scores there was practically no \(k\)-DBC that significantly outperformed NB. Second, gradual gains in accuracy for \(k > 0\) were noticed as training data increases. However, the amount of improvement over NB varies considerably for different \(k\) values, scores and domains. The results corroborate that NB can perform better for small datasets than more complex BNCs. NB requires fewer parameters to be estimated and hence, its high bias is compensated by its low variance, thus producing accurate classification [3, 13, 15].
However, as training data increases, more complex $k$-DBCs can reduce the bias resulting from the independence assumption, thus outperforming NB.

3 The Adaptive Prequential Framework

The main environmental assumption that drives the design of the AdPreqFr4SL is that observations arrive at the learning system not at the same time, which allows the environment to change over time. Without loss of generality, we assume that at each time point data arrives in batches. Moreover, we maintain an unique hypothesis $h_C$ defined as a pair $(S, \Theta_S)$, where $S$ is the structure and $\Theta_S$ are the parameters for that structure. The main goal is to sequentially predict the classes of the next batch.

Algorithm 2 The algorithm of the the AdPreqFr4SL

```
Require: A classifier class-model $M$, a dataset $D$ of i.i.d. labelled examples $<x, c = f(x)>$ divided in batches $B$ of $m$ examples
Ensure: A classifier $h_C \in M$ updated at each time point
1: Initialize $h_C$ with one of the hypothesis from $M$
2: for each batch $B$ of $m$ examples of $D$ do
3:   for each example $x$ in $B$ do
4:     $h_C(x) \Leftarrow$ predict($x$, $h_C$)
5:     $f(x) \Leftarrow$ getActualClass($x$)
6:     numIncorrected$+ = \delta(x, f(x), h_C(x))$ {the 0-1 loss is used}
7:     indicators $\Leftarrow$ assesIndicators(numIncorrected, ...)
8:     state $\Leftarrow$ estimateState(indicators, monitoring-tools)
9:     adapt($h_C, B, state$)
10: end for
11: return $h_C$
```

Algorithm 2 summarizes, in a rather informal way, the main processes of the AdPreqFr4SL. For each batch $B$ of examples the current hypothesis is used to do prediction, the correct class is observed and some performance indicators are assessed. Then, the indicator values are used to estimate the actual system’s state. Finally, the model is adapted according to the estimated state. Thus, AdPreqFr4SL is provided with some controlling mechanisms that try to select the best adaptive actions according to the current learning goal. To this end two performance indicators are monitored over time:

$Err_B$ - the BATCH ERROR (the proportion of misclassified examples in one batch)

$Err_S$ - the MODEL ERROR (the proportion of misclassified examples in the total of the examples that were classified using the same structure)

in order to estimate one of the possible system’s states:

$S_1$ - IS IMPROVING: the performance is improving

$S_2$ - STOP IMPROVING: the performance stops improving in a desirable tempo

$S_3$ - CONCEPT DRIFT ALERT: a first alert of concept drift was signaled

$S_4$ - CONCEPT DRIFT: there is a gradual concept change

$S_5$ - CONCEPT SHIFT: there is an abrupt concept change
S6 - STABLE PERFORMANCE: the performance achieves a plateau. It is the goal state.

In the next subsections we present the adaptive actions and control strategies that we have adopted in the AdPreqFr4SL for handling the cost-performance trade-off and concept drift.

3.1 Cost-Performance Management

3.1.1 Adaptation Policy

The adaptation strategy for handling cost-performance is based upon two main policies:

i) bias management - starting with an NB structure, we scale up the model's complexity by gradually increasing $k$ and then searching for new attribute dependences in the resulting search space;

ii) gradual adaptation - we define four levels of adaptation so that increasing the level increases its cost. In the INITIAL LEVEL a new model is built using the simple NB. In the FIRST LEVEL only the parameters are updated with new data\(^1\). In the SECOND LEVEL the structure is updated with new data. In the THIRD LEVEL, if it is still possible, $k$ is increased by one, and the current structure is once again adapted.

Algorithm 3 The adaptive actions for incorporating new data

<table>
<thead>
<tr>
<th>Require:</th>
<th>A classifier $h_c = (S, \Theta_S)$ belonging to the class of $k$-DBCs, a batch $B$ of labeled examples of $\langle X, C \rangle$, the level of adaptation, the current $k$ value, the $k_{\text{max}}$ value for the maximum allowable $k$, a boolean parameter $b_{\text{IterativeBayes}}$ to indicate whether to use Iterative Bayes or not</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ensure:</td>
<td>An adaptive action over the classifier $h_c$</td>
</tr>
<tr>
<td>1: if INITIAL level then</td>
<td></td>
</tr>
<tr>
<td>2: $k \leftarrow 0$ {A0: build a new model using NB}</td>
<td></td>
</tr>
<tr>
<td>3: learnNaiveBayes(\text{SHORT-MEMORY})</td>
<td></td>
</tr>
<tr>
<td>4: else if FIRST level then</td>
<td></td>
</tr>
<tr>
<td>5: updateParameters($h_c$, $B$, bUseIterativeBayes)</td>
<td></td>
</tr>
<tr>
<td>6: else if SECOND level then</td>
<td></td>
</tr>
<tr>
<td>7: updateStructure($h_c$, $B$, ...)</td>
<td></td>
</tr>
<tr>
<td>8: else if THIRD level then</td>
<td></td>
</tr>
<tr>
<td>9: if $k &lt; k_{\text{max}}$ then</td>
<td></td>
</tr>
<tr>
<td>10: $k += 1$ {A3}</td>
<td></td>
</tr>
<tr>
<td>11: updateStructure($h_c$, $B$, ...)</td>
<td></td>
</tr>
<tr>
<td>12: return the updated $h_c$</td>
<td></td>
</tr>
</tbody>
</table>

The rationale is as follows. We initialize $k$-DBC to NB by setting $k = 0$. Whenever new data arrives, we first try to improve the NB by adapting only its parameters. When there is evidence indicating that the performance of the NB stops improving in a desirable tempo, we start adapting the structure. Only in this case (for $k = 0$) do we move from the first level to the third level of adaptation: increment $k$ by one and start searching a 1-DBC using the hill-climber search procedure with arc additions. At this time point we have more

\(^1\)Optionally we can use the Iterative Bayes procedure for parameter refinement [17].
data available which allows the search procedure to find new 1-dependencies. Next, the algorithm continues from the first level of adaptation, that is, by performing only parameter adaptation, until there will be again evidence that the performance stops improving and this moves to the second level: update the current structure by searching for new attribute dependencies. At this stage the search procedure is also allowed to perform arc deletions. Only if the resulting structure remains the same, do we move to the third level of adaptation by incrementing $k$ by one and continue searching for new dependencies, now in an augmented search space. For avoiding $k$ to increase unnecessarily, we recover the old value of $k$ whenever the search procedure is not able to find new dependencies, thus keeping the original search space. Only if an abrupt concept drift is detected, do we come back to the initial level and build a new NB. This adaptation process will continue until it is detected that it does not make more sense to continue adapting the model. However, we will continue monitoring the performance. If any significant change in the behaviour is observed, then we will once again to activate the adaptation procedures.

Sequential updating of the parameters is straightforward for complete data. This requires a simple scan through all the examples in the given batch to increment the frequency counters corresponding to the values of each example. In addition, we can use the Iterative Bayes (IB) procedure for parameter refinement [17]. Experimental evaluation of the IB have shown consistent reductions of the error rate obtained with the NB. The improvements are mainly obtained due to a reduction on the bias component. IB improves the predictive distribution associated with each example by adjusting the estimates of the conditional probabilities. In each iteration and for each example the corresponding conditional probabilities are updated so as to increase the probability on the correct class. Suppose that during a iteration we process an example $x$ which belongs to the class $c_{\text{obs}}$ and that it is currently classified as belongs to the class $c_{\text{pred}}$. The main idea of the updating procedure is as follows. First, an increment, $\text{delta}$, proportional to the difference $1 - P(c_{\text{pred}} \mid x)$ is computed (if the example is incorrectly classified, then $\text{delta}$ is multiplied by -1 in order to be a negative quantity). Next, for each contingency table associated to each attribute, the increment $\text{delta}$ is added to the entry where $C = c_{\text{pred}}$ and proportionally subtracted to the remaining entries. The iterative procedure finishes when a stopping criterion is reached. IB can be easily extended to be used with a BNC that contain the NB structure. We only need to implement the same updating procedure for each contingency table associated to each attribute and its parents (the class node must be one of them).

To update the structure we use the hill-climbing search procedure described in Algorithm 1. Whenever a structure-adaptation process is launched, the current hypothesis is used as the initial model. If the current structure is the NB structure, the algorithm uses only arc additions. Otherwise, the search procedure is also allowed to perform arc deletions, thus correcting from previous errors in the search process. In our implementation we assume that we can keep in main memory all the sufficient statistics required for computing the score of each candidate structure.
3.1.2 Control Policy

The control policy defines the criteria for tracking two situations: i) At which time point do we start adapting the structure?; ii) At which time point do we stop doing any adaptation? If it is detected that the performance of the current model no longer improves in a desirable tempo (the state $S_2$), we start adapting the structure. If it is detected that the performance reaches a plateau (the state $S_6$), we stop adapting the model.

The curve describing the performance as a function of the sample size of the training data is often called the learning curve. Thus, the slope of a learning curve is an indicator of how much performance can be gained by increasing the number of examples. A learning curve typically has three different parts [29]: i) a steeply sloping part early in the curve; ii) a more gently sloping middle part; and iii) a plateau late in the curve when the learning accuracy no longer increases with more training data. To detect the states $S_2$ and $S_6$, we plot the values of successive model errors, $y(t) = E_{\text{Err}}^{(t)}$, in time order and connect them by a line, thus obtaining the model-error learning curve (model-LC). Observation of the model-LC is crucial, because it helps explain the behavior of the adaptive learning algorithm using different structures with increasing complexity.

We consider that the state $S_2$ is met if: i) the model-LC starts behaving well and ii) its slope is gentle. A learning curve that depicts the error rate starts behaving well when its graph becomes monotonically decreasing and convex for a given number of points [4]. Thus, whenever we start using a new structure we will wait until model-LC starts behaving well and shows only little improvements in the performance in order to trigger a new structure adaptation. If the structure does not change after adaptation, we once again look at the model-LC to detect whether it has already reached its plateau (i.e. $S_6$ is signaled). The following question thus arises: How does one verify whether the required criteria of discrete convexity and non-increasing trend are met? We empirically found that by using a method based on the geometrical properties of the model-LC, which analyzes the graphical behaviour of the most recent $q$ points, we could more consistently determine discrete convexity and the slope of the model-LC taking into account the local variance. We experimented our heuristics with $q = 5, 7, 9$ and obtained the best results by setting $q = 7$.

As illustrated in Figure 2 we construct a triangle $T$ with the points $p_1, p_4, p_7$ and use its signed area, $A(T)$, to test for discrete convexity\footnote{The subjects related to Computational Geometry have been extracted from [27].}. The points $p_1, p_4, p_7$ are arranged in a convex pattern iff $A(T)$ is positive. In this case the path $p_1 \rightarrow p_4 \rightarrow p_7$ is oriented counterclockwise around the triangle. Taking into account the local variance we consider a convex pattern, if $A(T) > \delta_a$ where $\delta_a$ is a very small negative number. Then, we analyze the angles formed between middle segments, $\angle_1 = \angle p_1, p_2, p_4$, $\angle_2 = \angle p_1, p_3, p_4$, $\angle_3 = \angle p_4, p_5, p_7$ and $\angle_4 = \angle p_4, p_6, p_7$ to determine if the remaining points are almost colinear given a tolerance $\delta_c$, where $\delta_c$ is a very small positive number, that is, if $\sin(\angle_l) < \delta_c, \forall \angle_l, l = 1, 2, 3, 4$. We then use the Sen’s slope estimator [35] for determining
whether there is a non-increasing trend in these observed points. We consider a non-increasing trend if SenSlope\((q) < \delta_s\) where \(\delta_s\) is a very small positive number. We obtained satisfactory empirical results by setting \(\delta_a = -0.0001\) (our tolerance for convexity), \(\delta_s = 0.0001\) (our tolerance for non-increasing trend) and \(\delta_c = 0.0001\) (our tolerance for colinearity).

Thus, we consider that the points \(p_1, p_2, \ldots, p_7\) are arranged in a convex pattern with a non-increasing trend and gentle slope if for a given positive small number \(\epsilon_1\), the threshold for the gentle slope, the following criterion is met:

\[
\delta_a < A(T) < \epsilon_1 \land \sin(\angle_l) < \delta_c, \forall \angle_l, l = 1, 2, 3, 4 \land \text{SenSlope}^{(7)} < \delta_s \quad (2)
\]

We consider that the stopping criterion is met if given a positive small number \(\epsilon_2\), the threshold for the plateau, such that \(\epsilon_2 < \epsilon_1\), the following criterion is met:

\[
|A(T)| < \epsilon_2 \land \sin(\angle_l) < \delta_c, \forall \angle_l, l = 1, 2, 3, 4 \quad (3)
\]

In addition, we use a heuristic based on the observation of the batch error before and after the adaptation, which has been demonstrated to be efficient for an early detection of the point at which we should start adapting the structure. Whenever we obtain a decrease of the batch error after adaptation, we consider that the learner is still able to learn using the current structure. Otherwise, if for a predefined number of consecutive times, maxTimes, the batch error does not decrease after parameter adaptation we assume that increasing the number of training examples will not result in further improvements on the parameter estimates for that structure and signal the state [S2] - STOP IMPROVING.

Figure 3 illustrates the behaviour of the model-LC for one randomly generated sample of the adult dataset using batches of 100 examples. To serve as a baseline, we also plot the error rates obtained with the NB and with a 3-DBC (the class-model with best performance) induced from scratch at each learning step. During all the learning process the structure changed only five times. The graphical behavior of the model error neatly corresponds to the detected conditions which lead to a structure-adaptation action. The \(k\) value slowly increases from 0 to 3 until that the stopping criteria is met at \(t = 120\) and the model is not further adapted with new data.
3.2 Using the P-Chart for Handling Concept-Drift

Concept drift [37] represents the changes in the target function, or simply, changes in the distribution underlying the data over time. Several available concept drift trackers [18, 21, 25, 37] employ different approaches that include some control strategies for deciding whether adaptation is in fact necessary because a concept change has actually occurred. A process that monitors the value of some performance indicators is usually implemented. If a concept drift is detected, some actions to adapt the model to these changes are taken. Since recent data is more relevant than older one, old examples (and hypotheses based on these) should eventually be forgotten. Some concept drift trackers [18, 21, 37] are also capable of recognizing the extent of concept drift. The term concept drift is more oftenly associated to gradual changes whereas the term concept shift defines abrupt changes.

Our method to handle concept drift relies on Statistical Quality Control. Shewhart control charts [34] represent the basic monitoring tools for distinguishing trends and out-of-control conditions in a production process. The P-Chart - an attribute control chart is used to monitor the proportion of nonconforming items (a dichotomous count variable). We use the P-Chart to monitor the behaviour of the batch error $Err_B$. The values $p(t) = Err_B^{(t)}$ are plotted on the chart in time order and connected by a line. The chart has a center line (CL), an upper control limit (UCL) and an upper warning limit (UWL). If the sample sizes are large ($\geq 30$) the sample proportion approaches the Normal distribution with parameters $\mu = p ; \sigma = \sqrt{p(1-p)/n}$ ($p$ is the population proportion).
The use of three-sigma control limits is a reasonable choice. Suppose that an estimate \( \hat{p} \) is obtained from previous data. We can obtain the \( \text{P-Chart} \)'s lines as follows: \( \text{CL} = \hat{p} \); \( \text{UCL} = \hat{p} + 3\hat{\sigma} \); \( \text{UWL} = \hat{p} + \alpha\hat{\sigma}, 0 < \alpha < 3 \). Since a low error is desirable, we do not need to use the low limits here. To better follow the natural behaviour of the learning process we set the target value \( \hat{p} \) to the minimum value of the current model error \( \text{Err}_S \) instead of using some average of previously observed values. We denote it by \( \text{Err}_{\text{min}} \). Whenever a new structure \( S \) is found, \( \text{Err}_{\text{min}} \) is initialized to some big number. Then, at each time step if \( \text{Err}_S^{(t)} + S\text{Err}_S^{(t)} < \text{Err}_{\text{min}} \) then \( \text{Err}_{\text{min}} \) is set to \( \text{Err}_S^{(t)} \), where \( S\text{Err}_S^{(t)} \) is its standard deviation.

At each time point \( t \), \( \hat{p} \) is set to \( \text{Err}_{\text{min}} \) and the \( \text{P-Chart} \)'s lines are computed. Then, it is observed where the new proportion \( p(t) = \text{Err}_B^{(t)} \) falls on the \( \text{P-Chart} \). If \( p(t) \) falls above the \( \text{UCL} \), then a concept shift is signaled. If for the first time \( p(t) \) falls between the \( \text{UCL} \) and the \( \text{UWL} \), then a concept drift alert is signaled. Otherwise, if this situation occurs for two or more consecutive times then a concept drift is detected. Only if \( p(t) \) falls under the \( \text{UWL} \) we assume that the learner is in control. The adaptive strategy mainly consists of manipulating a \( \text{SHORT-MEMORY} \) to store those examples that we suspect belongs to a new concept. If a concept shift is detected then all the examples from the \( \text{SHORT-MEMORY} \) are used to build a new NB classifier. Afterwards, the \( \text{SHORT-MEMORY} \) is cleaned for future uses. Whenever a concept drift or alert is signaled, the examples of the current batch are added to the \( \text{SHORT-MEMORY} \). However, after signaling a concept drift, the new examples are not used to update the model in order to force a great degradation of the performance. This way the \( \text{P-Chart} \) will more quickly be able to recognize a concept shift and re-build the model.

### 3.3 The Algorithm for Learning \( k \)-\( \text{DBC} \)s in the \( \text{AdPreqFr4SL} \)

Algorithm 4 depicts the pseudo-code of the whole algorithm for learning \( k \)-\( \text{DBC} \)s in the \( \text{AdPreqFr4SL} \) that summarizes all the above described strategies. The algorithm must be provided with the values of five parameters: the \( k_{\text{Max}} \) value for the maximum allowable degree of attribute dependence, the variable \( b_{\text{IterativeBayes}} \) to indicate the use of Iterative Bayes, and the three parameters defined in the control criteria for bias management: the threshold \( \text{eps}_1 \) for the gentle slope, \( \text{eps}_2 \) for the plateau and the number of consecutive times \( \text{maxTimes} \) the \( \text{Err}_B \) does not decrease after parameter adaptation used in the heuristic based on the observation of the batch error before and after adapting the parameters. Control strategies for bias management are mainly based on the observation of the model error \( \text{Err}_S \) (the \text{model-LC}). The main goal is to detect when start adapting the current structure and stop the adaptation process. Concept drift management is mainly based on the observation of the batch error \( \text{Err}_B \) and on the findings detected by the \( \text{P-Chart} \).
Algorithm 4 The algorithm for learning $k$-DBCs in AdPreqFr4SL

Require: A dataset $D$ divided in batches of $m$ examples, a $\maxTimes$ value for the maximum allowable $k$, the thresholds: $\eps1$ for the gentle slope and $\eps2$ for the plateau, the number of consecutive times $\maxTimes$ that $Err_B$ does not decrease after parameter adaptation, a boolean variable $\text{bIterativeBayes}$ for using Iterative Bayes or not, a scoring function $\text{Score}(S, D)$

Ensure: A classifier $h_c = (S, \Theta_B)$ belonging to the class of $k$-DBCs

1: AdaptiveAction($h_c$, SHORT-MEMORY, INITIAL LEVEL) {build a new NB classifier}
2: for each next batch $B$ of $m$ examples of $D$ do
3: predictions $\leftarrow$ predict($B$, $h_c$)
4: observed $\leftarrow$ getFeedback($B$) {get feedback}
5: $p(t) \leftarrow Err_B^{(t)}$, $y(t) \leftarrow Err_B^{(t)}$ {asses current indicators}
6: Add ($t$, $y(t)$) to model-LC
7: state $\leftarrow$ getState($p(t)$, $P$-Chart) {concept drift detection using the P-Chart}
8: if state is CONCEPT SHIFT then
9: Add $B$ to SHORT-MEMORY
10: AdaptiveAction($h_c$, SHORT-MEMORY, INITIAL LEVEL) {build a NB classifier}
11: Clean SHORT-MEMORY
12: else if state is CONCEPT DRIFT ALERT $\lor$ CONCEPT DRIFT then
13: Add $B$ to SHORT-MEMORY
14: else
15: Clean SHORT-MEMORY
16: // if state is IN CONTROL then observe the model-LC
17: if model-LC is Convex-NonIncreasing-with-GentleSlope($\eps1$) then
18: state $\leftarrow$ STOPS IMPROVING {conditions 2 are met}
19: else
20: state $\leftarrow$ IS IMPROVING
21: if state IS IMPROVING $\lor$ CONCEPT DRIFT ALERT then
22: AdaptiveAction($h_c$, $B$, FIRST LEVEL, $\text{bIterativeBayes}$) {update parameters}
23: if consecCounter($Err_B^{\text{tmp}}$, $\maxTimes$) $\geq$ $\maxTimes$ then
24: state $\leftarrow$ STOP IMPROVING
25: if state STOPS IMPROVING then
26: if $k > 0$ then AdaptiveAction($k$-DBC, $B$, SECOND LEVEL, $\ldots$) {update structure}
27: if (not change($S$) ^ $k < \maxTimes$) $\lor$ $k = 0$ then
28: AdaptiveAction($h_c$, $B$, THIRD LEVEL, $k$, $\ldots$) {increment $k$; continue searching}
29: if not change($S$) then
30: // verify the stopping criterion
31: if model-LC Has-Plateau($\eps2$) then
32: stopAdapting $\leftarrow$ TRUE, state $\leftarrow$ STABLE PERFORMANCE
33: end for
34: return $h_c$

3.4 Related Work

Issues in Sequential Updating of Bayesian Network

Nowadays, there are only a limited number of approaches concerning sequential updating of Bayesian Network structures, which have mainly adapted a batch hill-climbing learning algorithm into an incremental one while optimizing the cost of updating and the memory space. The approaches described in [5, 24, 16] maintain one or a set of alternative structures subjected to updates whenever new data arrives. However, a more related work that have influenced our adaptive framework is proposed in [30]. The adaptive method is provided with controlling mechanisms to detect when it is imperative to adapt the structure. Whenever new data arrives, the order of the models in the search path (by means of their scores) is analyzed. If the order is altered, then new data may provide new information that is not included in the current hypothesis and adaptation on the structure is triggered. However, the difference in previous approaches for adapting the Bayesian Network structures with respect to
our adaptive framework is that it is assumed that training examples have been
drawn from a stationary underlying distribution, i.e., they don’t handle concept
drift.

**Approaches for Scaling up Learning Algorithms**

Our control strategies for handling cost-performance are in some way related to
the scaling-up methods for handling massive data streams. The learning-curve
sampling method[29] is based on the monitoring of the learning curve. The
main idea is that if we can determine the minimal sample size, $N_{\text{min}}$, at which
the learning curve converges (when the learning curve reaches its final plateau),
we can obtain the same accuracy by using only a data sample of size $N_{\text{min}}$,
while considerable reducing the computational cost. For instance, the method
proposed in [12] determines $N_{\text{min}}$ using the Hoeffding bound. Another method
commonly used for convergence detection is the extrapolation of the learning
curve method. This method uses all the historical data to fit a parametric
learning curve and then extrapolate the learning accuracy at the full length. If
the slope is sufficiently close to zero, convergence is detected. The models most
widely used to fit learning curves are power law models (e.g. $y = a + b \cdot x^{-c}$). The
progressive sampling method proposed in [29] starts with a small sample and
uses progressively larger samples until it is determined that the learning curve
converges. Finally, the method proposed in [4] is based on an early assessment
of the performance and is one of the methods that have more influenced our
approach for monitoring the learning curve. The basic idea is that when the
graph has the proper shape (the learning curve becomes behaving well), the
estimates for the future performance are calculated.

**Approaches for Handling Concept Drift**

Motivated by the STAGGER algorithm [33], several available on-line adaptive
systems have implemented different forgetting mechanisms over either the model
or the data to deal with concept drift. Forgetting have been used through two
main techniques: weighted examples and time windows. In the simplest case the
window is of fixed size. More elaborated methods use some heuristics that allow
adjusting the window size according to the current extent of concept drift. The
window adjustment heuristic introduced in the FLORA algorithm [37] showed
a significantly increased system’s flexibility and power. The rationale is that
of decreasing the window size when a concept drift is detected, otherwise the
window size is increased to include the new examples.

Two more related works [21, 25] that have influenced our method for han-
dling concept drift have been applied in information filtering. In [21] a window
adjustment heuristic based on the monitoring of three performance indicators,
accuracy, recall and precision is proposed. At each time point for each indica-
tor the mean $\mu$ and the standard deviation $\sigma$ are computed using the last $M$
batches. Each current indicator value then is compared to the confidence interval
$\mu \pm \alpha \times \sigma$ ($\alpha > 0$). If the current value is smaller than the lower end point of
this interval, a concept change is suspected. A further test determines whether the change is abrupt or gradual. If the current indicator value is smaller than its predecessor $\beta$ times ($0 < \beta < 1$), a concept shift is signaled and the window is reduced to its minimal size (the size of one batch). Otherwise, a concept drift is signaled and the window is reduced by using a reduction rate. The approach proposed in [25] employs Statistical Quality Control and aimed at detecting changes without expensive user feedback. To this end, three performance indicators are monitored: i) the sample error (similar to the batch error $Err_B$); ii) the expected error rate; and iii) virtual rejects. The target value is estimated by using the weighted average of the indicator values on recent batches based only on those values that are within the warning limits.

4 Experimental Evaluation

We carried out a series of experiments for evaluating the AdPreqFr4SL for $k$-DBCs, using both, artificially generated datasets and benchmark problems from the UCI repository [26]. We here provide only an overview of all the conducted experiments and results. A complete description is given in [6]. We evaluated two versions of adaptive algorithms, Adap1 and Adap2, using Algorithm 4. Adap2 additionally implements Iterative Bayes (IB). We compared Adap1 and Adap2 against NB and several $k$-DBCs (varying $k$) induced from scratch. In each learning step the batch hill-climber learning procedure (Algorithm 1) was used to learn a $k$-DBC from all seen examples. Since learning from scratch use all the data provided so far, this approach for updating the classifier should produce good baseline models. To avoid very complex structures we set $kMax=5$ for all the experiments. The parameter maxTimes is used to ensure an early detection of the point at which an structure-adaptation action must be carried out. By choosing small values we can accelerate the detection of this time point. Intuitively, we set maxTimes=2 for artificial domains (less complex domains, with binary variables) and set maxTimes=3 for benchmark problems. The thresholds $\epsilon_1$ and $\epsilon_2$ were set according to the domain’s complexity. Intuitively, we choose lower thresholds for more complex domains. All the results were obtained as average values over 10 generated samples. Here we present the results using the Bayesian score (the marginal likelihood). In [6, 7] we presented a more in depth study comparing the performance for different scores. All the learning algorithms were implemented in Java using Weka’s classes for BNCs [2, 38].

4.1 Evaluation with Benchmark Problems

We here only show the results for three selected datasets from the UCI repository: balance, nursery and adult. We used a discretized version of the adult dataset [22] and randomly generated samples of 10000 examples for balance using its underlying rules. We set $\epsilon_1=0.01$ and $\epsilon_2=0.001$ for balance and

---

3In Appendix A we provide the results of a study that evaluates how different threshold settings can affect the behavior of the adaptive algorithms.
Figure 4: Error Rate, Model Error and $k$-values for UCI’s datasets

$nursery$ and $\epsilon_1=0.001$ and $\epsilon_2=0.0001$ for $adult$. Figure 4 compares the performance of $Adap1$ and $Adap2$ against NB and several $k$-DBCs induced from scratch at each time point. Plots on the left depict the error rate. In most cases, $Adap1$ approaches the performance of the best $k$-DBC and $Adap2$ outperforms $Adap1$. Plots in the middle depict the model error of $Adap1$ and $Adap2$ against the error rate of the best $k$-DBC. Plots on the right show the $k$ values giving us an idea about the increasing complexity of resulting classifiers over time. We can observe that for $balance$, the model error $Err_S$ approaches 0 and $k$ approaches 3. $Adap1$ and $Adap2$ were able to find structures that represent the actual degree of attribute dependences. We can also observe that the increasing slope of the $k$ value using $Adap2$ is more gradual, thus inducing less complex classifiers. $Adap2$ can get trapped in less complex structures while reducing the bias on the parameter estimates.

Table 1 helps us to evaluate the performance, complexity and cost of adaptation per dataset at the last learning step. The column "Last $Err_B$" shows the error of the last batch of examples, which was not used to update the classifier. The column "$\# Add.\ Arcs$" shows the total number of arcs added to NB. The column "$\# Str.\ Changes$" shows the number of times the structure actually changed during all the learning process in relation to the number of performed
Table 1: Analysis of the Final Performance, Complexity and Cost of Adaptation per dataset

<table>
<thead>
<tr>
<th>Balance</th>
<th>Nursery</th>
<th>Adult</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>k-DBCs</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NB</td>
<td>8.10</td>
<td>12.00</td>
</tr>
<tr>
<td>1-DBC</td>
<td>5.80</td>
<td>5.4</td>
</tr>
<tr>
<td>2-DBC</td>
<td>7.40</td>
<td>9.0</td>
</tr>
<tr>
<td>3-DBC</td>
<td>7.20</td>
<td>9.0</td>
</tr>
<tr>
<td>Best</td>
<td>7.00</td>
<td>9.0</td>
</tr>
<tr>
<td>Adap1</td>
<td>5.40</td>
<td>7.4</td>
</tr>
<tr>
<td>Adap2</td>
<td>7.40</td>
<td>9.0</td>
</tr>
</tbody>
</table>

Structure adaptations. The results show that both, Adap1 and Adap2, are able to perform a more artful cost-performance trade-off than non-adaptive versions. The reduction of the cost of updating is evident if we compare the small number of adaptations performed on the structure by Adap1 and Adap2 in contrast to the great cost of searching for a new structure at each learning step. In general, the number of times the structure actually changed when a search procedure was invoked at each point time is very small (e.g. for the adult this proportion is 8.6/160). This evidences that it is more appropriate to perform adaptations on the structure when there is some accumulated data and the search procedure is able to find new dependences. Although both, Adap1 and Adap2, show a desirable behaviour, results evidence that Adap2 ensures the best cost-performance trade-off in these three particular domains: the number of structure adaptations and the resulting error are smaller.

4.2 Evaluation with Concept Shift and Drift Scenarios

The use of artificial domains allows us to know the true degree of the attribute dependencies in the domain and when changes in target functions occur. By generating large samples we could test the specific problems that the algorithm exhibits: bias management and concept drift management. Five concept shift scenarios (CSSs) and five concept drift scenarios (CDSs) were generated using randomly generated k-DBCs with 9 binary attributes and a binary class node for k = 1, 2, 3, 4, 5. Both, CSSs and CDSs represent a sequence of five different learning contexts, associated to different generative k-DBCs. Whereas k remains constant in a CSS, we used k-DBCs of increasing k for generating a CDS (a 1-DBC for the first context, a 2-DBC for the second one, etc.). In CSSs we simulated four abrupt concept changes by forcing the underlying k-DBC to change after every 2000 examples. We used batches of 100 examples for CSSs and batches of 50 examples for CDSs. In CSSs we simulated four gradual changes by setting the parameters of a simulation procedure [37]: \( t_1 = 37, t_2 = 77, t_3 = 117, t_4 = 157 \) (the time points at which the concept begins to drift), \( \Delta = 300 \) (the drift rate) and \( \alpha = 3/4 \) (each 3 examples of the old concept appears one example of the new concept). We set \( \text{eps1}=0.05 \) and \( \text{eps2}=0.005 \) for artificial datasets.

Figure 5 illustrates the adaptive and control strategies in one of generated CDS. In the first drift phase (between t=37 and t=43) the P-Chart detected
two concept shifts and a new NB was built using the examples of the current batch. In the second drift phase (between t=77 and t=83) almost all the points fell above the UWL but very close to the UCL. The P-Chart signaled concept drift and the adaptation process was temporarily stopping to force the $\text{Err}_B$ to jump outside the UCL. Later, at t=83, when a concept shift was detected, all the examples stored in the SHORT-MEMORY were used to build a new NB. For the remaining drift phases our detection method using P-Chart also worked as expected. As a result, the structure was rebuilt five times, at time points that belong to the drift phases. The complexity of the induced $k$-DCBs increased from context to context: in the first context the resulting $k$-DBC is a 1-DBC, in the third - a 3-DBC, in the fourth - a 4-DBC, in the last context it is a 4-DBC too (searching for more complex structures can require more training data). Only in the second context the NB structure was not modified due to the adaptation was stopped early.

Figure 6 compares the performance over time of all the algorithms in the CSS associated to a 2-DBC (CS-II) and one CDS scenario (CD-I). Results evidence that significant improvements in the performance are achieved by using adaptive algorithms instead of their non-adaptive versions. After a concept drift occurs, the performance of all the algorithms suffers a significant deterioration. However, $\text{Adap1}$ and $\text{Adap2}$ show a good recoverability capability and are able to control the performance, trying to improve it back to a level, that even approaches the performance of the true model. In drift phases, the $k$ value falls to 0, which evidences that a concept shift has been detected and a new NB has been built. Results also evidence that adaptive algorithms approach the appropriate class-model associated to each learning context. The $k$ value approaches 2 for all the learning contexts in CS-II while it increases from context to context in CD-I. The performance of both adaptive algorithms $\text{Adap1}$ and $\text{Adap2}$ is very
similar in artificial domains, especially for those contexts where simpler generative models were used. All the variables in the artificial domain are binary, so the number of parameters does not grow so much as \( k \) increases, and hence, the bias resulting from the estimation error is not so significant.

Table 2: Number of states detected per concept drift scenario and adaptive algorithm

<table>
<thead>
<tr>
<th>SCENARIO</th>
<th>CD-I</th>
<th>CD-II</th>
<th>CD-III</th>
<th>CD-IV</th>
<th>CD-V</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATES</td>
<td>Adap1</td>
<td>Adap2</td>
<td>Adap1</td>
<td>Adap2</td>
<td>Adap1</td>
</tr>
<tr>
<td>S2</td>
<td>32.8</td>
<td>22.3</td>
<td>31.0</td>
<td>28.5</td>
<td>27.2</td>
</tr>
<tr>
<td>S3</td>
<td>4.2</td>
<td>4.0</td>
<td>5.3</td>
<td>5.3</td>
<td>3.0</td>
</tr>
<tr>
<td>S4</td>
<td>1.7</td>
<td>1.4</td>
<td>2.2</td>
<td>2.9</td>
<td>1.5</td>
</tr>
<tr>
<td>S5</td>
<td>5.0</td>
<td>5.0</td>
<td>4.2</td>
<td>3.8</td>
<td>4.5</td>
</tr>
<tr>
<td>S6</td>
<td>1.0</td>
<td>1.7</td>
<td>0.4</td>
<td>0.8</td>
<td>1.1</td>
</tr>
</tbody>
</table>

Results in Table 2 reflect that adaptive algorithms were able to detect concept drift situations. The number of times that the state \( S4 = \text{CONCEPT DRIFT} \) was detected is between 1.0 and 3.0 and the state \( S5 = \text{CONCEPT SHIFT} \) is between 3.7 and 5.0. After signaling concept drift, the new examples are not used for adaptation to force a great degradation of the performance. This way the values of the error sample can fall outside the P-Chart control line, the P-Chart can recognize a concept shift and re-build the model using the examples from the new context. The number of times a concept drift alert \( S3 \) was detected is greater for scenarios with a lesser number of shift detections (e.g. CD-III). Gradual changes in the context may not always lead to a great deterioration of the performance. The number of times that the state \( S2 = \text{STOP IMPROVING} \)
was signaled is between 19.9 and 33.3 times over a total of 100 leaning steps. In
CDSs the classifier is rebuilt using NB whenever a new context is detected. The
adaptive algorithm is forced to trigger new adaptations in the structure until a
satisfactory level of performance is once again reached. The values that corre-
spond to the state $S_6$ represent the number of times the stopping criterion was
meet during the whole learning process. In most cases, the adaptation process
was stopped at least once.

5 Conclusions and Future Work

We have presented the AdPreqFr4SL, which attempts to handle cost vs. perfor-
mance and cope with concept drift. We approach the cost-performance trade-off
through bias management and adaptation control. The method for handling con-
cept drift is explicitly modeled using a Shewhart P-Chart. The AdPreqFr4SL
is provided with practical controlling mechanisms based on the monitoring of
two performance indicators: the batch error (to deal with concept drift) and the
model error (for bias management). Since our controlling methods for monitor-
ing these two indicators are classifier-independent, they could be applicable to a
range of supervised learning algorithms. Results in simulated concept drift sce-
narios show that the P-Chart is able to consistently recognize concept changes,
both abrupt and gradual, and to adapt quickly to these changes. Moreover, we
believe that the proposed strategies for bias management could be applied to
any supervised learning algorithm based on discrete search with a hierarchical
and increasing control over the complexity of its induced hypotheses. We
implemented simple heuristics based on the observation of some geometrical
properties of the most recent points of the model error learning curve to deter-
mine when to start searching for new dependencies and when to stop adapting.
The results of conducted experiments give us some confidence that these heuris-
tics work quite well, although they depend on two given thresholds. Obviously,
the choice of a particular threshold value can influence the performance of the
adaptive algorithms, specially for more complex domains (in less complex do-
 mains were observed only tiny differences in the performance). In more complex
domains we should use lower thresholds to give more time for the adaptation
process to continue looking for new dependencies and improving the parameter
estimates.

During all the learning process we maintain an unique hypothesis (classifier)
which can be updated with new incoming data. One of the main contribution
of this work is that instead of selecting a particular class of BNCs (e.g. TAN,
BAN, etc.) and using it during all the learning process, we propose to use the
k-Dependence Bayesian Classifiers ($k$-DBCs) - a stratified family of classifiers
with increasing (smooth) complexity. We start with the simple NB by setting
$k = 0$. Then, we use simple control strategies to decide when to do the next
move in the spectrum of attribute dependencies (by gradually increasing $k$)
and to start searching for new dependences. As a result, our strategy leads
to the scaling up of the model's complexity slowly enough so that the use of
more training data will reduce bias and variance and hence, the classification error. This way the predictive accuracy of the NB is improved significantly over time while reducing the cost of updating without affecting the quality of induced classifiers. Finally, by using the proposed AdPreqFr4SL with the Iterative Bayes procedure for parameter refinement (specially for more complex domains) we can better trade-off the reduction of the bias resulting from the modeling error with the reduction of the bias resulting from the estimation error thus obtaining a better cost-performance trade-off over time.

Future work will involve a more systematic investigation of adaptive issues in sequential updating of BN structures (data structures and methods for storing and computing the sufficient statistics in an incremental fashion [31], the use of more sophisticated backtracking methods for the hill-climber search procedure, etc). An obvious topic for future work also includes the application of the AdPreqFr4SL framework to real-world on-line learning systems. NB is one of the most used classifiers in real-world on-line applications mainly due to its simplicity, effectiveness, easy interpretability and incremental nature. However, in practice, the independence assumption is violated which can lead to a poor predictive performance. Since results evidence significant improvements of the NB over time, we believe that the AdPreqFr4SL with the stratified class of k-DBCs can be used as a better alternative to the simple NB, specially in those real-world on-line applications where we need to deal with a continuous increase of data and concept changes are likely to occur.

Acknowledgments:

Thanks to the financial support given by the the CEOC through Programa POCTI, FCT, co-financed by EC fund FEDER and the FCT’s project ALES II (POSI/EIA/55340/2004). We wish to also acknowledge Pedro Larrañaga and Iñaki Inza from the University of País Vasco for helpful discussions and supporting which provide so much valuable assistance specially in the beginning of this work.

References


Appendix A

Evaluating the AdPreqFr4SL with Benchmark Problems using Different Threshold Settings

In the following experiments we analyze the effect of different threshold settings in the performance of adaptive algorithms using three datasets from the UCI repository with large number of examples. The main characteristics of the three datasets are summarized in Table 3. All these three problems present numerical attributes and missing values. We used their discretized versions available online [22]. We evaluated the two versions, Adap1 and Adap2, of the adaptive algorithm (Algorithm 4) against the NB learning algorithm and several k-DBCs induced from scratch at each time point using a batch hill-climber (Algorithm 1). We used the marginal likelihood score and batches of 100 examples. All the indicators’ values were obtained as the averaged values over 5 runs.

Table 3: Datasets used in the experiments with different parameter settings

<table>
<thead>
<tr>
<th>Dataset</th>
<th># Attrib</th>
<th># Classes</th>
<th># Inst.</th>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>mushrooms</td>
<td>22</td>
<td>2</td>
<td>6300</td>
<td>62</td>
</tr>
<tr>
<td>page-blocks</td>
<td>10</td>
<td>5</td>
<td>5400</td>
<td>53</td>
</tr>
<tr>
<td>letters</td>
<td>17</td>
<td>24</td>
<td>20000</td>
<td>199</td>
</tr>
</tbody>
</table>

Table 4 summarizes the final results for different settings of $\text{eps}_1$ and $\text{eps}_2$. For mushrooms Adap1 and Adap2 behaved as expected for almost all threshold settings. A 2,3-DBC show the best performance providing zero final batch errors. The best results were obtained with $\text{eps}_1=0.01$ and $\text{eps}_2 \leq 0.0001$ while slightly worse results were obtained with $\text{eps}_1=0.05$ and $\text{eps}_2=0.005$ (since the adaptation process was stopped earlier, a little less dependencies were found). For page-blocks a 2-DBC is the best k-DBC. Adap1 and Adap2 show a good performance for all the threshold settings. The final batch errors are even lower than those obtained with the best k-DBC (3.80%). The choice of different thresholds have no significant influence on the performance. However, the best results were obtained with $\text{eps}_1=0.01$ and $\text{eps}_2 \leq 0.001$. The letters domain is more difficult to learn. The class has 24 labels and during the discretization process some attributes take up 20 different discrete values. However, large reduction of the error rate is obtained when going from $k=0$ (the NB) to $k=2$ (the best k-DBC): the error rate falls from 28.77% to 18.54% and the final batch error from 26.80% to 11.40%. These results evidence the advantages of using BNCs in this complex domain if we have enough training data. The adaptive algorithms, in general, behaved as expected for almost all the threshold settings. However, the worst results were obtained using greater thresholds $\text{eps}_1=0.05$ and $\text{eps}_2=0.005$ whereas the best ones were obtained with lower thresholds $\text{eps}_1=0.01$ and $\text{eps}_2=0.00001$. For this complex domain it is more appropriate to use smaller threshold values.
Table 4: Results with different parameter settings for the **mushrooms**, **page-blocks** and **letters** datasets

<table>
<thead>
<tr>
<th>Performance</th>
<th>Complexity</th>
<th>Adaptation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error Rate</td>
<td>Model Error</td>
<td>Final Batch Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added Arcs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Final k value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Struct. Changes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When stopped</td>
</tr>
</tbody>
</table>

**Mushrooms**, Learning Steps: 62

<table>
<thead>
<tr>
<th>NB</th>
<th>Performance</th>
<th>Complexity</th>
<th>Adaptation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.68</td>
<td>2.20</td>
<td></td>
</tr>
<tr>
<td>Best k-DBC</td>
<td>0.07</td>
<td>0.00</td>
<td>37, 51</td>
</tr>
<tr>
<td>eps1, eps2</td>
<td>0.05, 0.05</td>
<td>1.20, 0.60</td>
<td>0.20, 0.20</td>
</tr>
<tr>
<td></td>
<td>0.13, 0.07</td>
<td>0.20, 0.00</td>
<td>30.6, 27.0</td>
</tr>
<tr>
<td></td>
<td>0.01, 0.01</td>
<td>0.00, 0.00</td>
<td>33.0, 30.2</td>
</tr>
<tr>
<td></td>
<td>0.00, 0.00</td>
<td>0.00, 0.00</td>
<td>33.0, 30.2</td>
</tr>
<tr>
<td></td>
<td>0.00, 0.00</td>
<td>0.00, 0.00</td>
<td>33.0, 25.8</td>
</tr>
<tr>
<td></td>
<td>0.00, 0.00</td>
<td>0.00, 0.00</td>
<td>33.0, 25.8</td>
</tr>
<tr>
<td></td>
<td>0.00, 0.00</td>
<td>0.00, 0.00</td>
<td>33.0, 25.8</td>
</tr>
<tr>
<td></td>
<td>0.00, 0.00</td>
<td>0.00, 0.00</td>
<td>33.0, 25.8</td>
</tr>
<tr>
<td></td>
<td>0.00, 0.00</td>
<td>0.00, 0.00</td>
<td>33.0, 25.8</td>
</tr>
<tr>
<td></td>
<td>0.00, 0.00</td>
<td>0.00, 0.00</td>
<td>33.0, 25.8</td>
</tr>
</tbody>
</table>

**Page-blocks**, Learning Steps: 53

<table>
<thead>
<tr>
<th>NB</th>
<th>Performance</th>
<th>Complexity</th>
<th>Adaptation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5.51</td>
<td>4.60</td>
<td></td>
</tr>
<tr>
<td>Best k-DBC</td>
<td>4.60</td>
<td>3.80</td>
<td>15.6</td>
</tr>
<tr>
<td>eps1, eps2</td>
<td>0.05, 0.05</td>
<td>4.83, 4.76</td>
<td>3.66, 3.45</td>
</tr>
<tr>
<td></td>
<td>0.01, 0.01</td>
<td>4.92, 4.75</td>
<td>3.62, 3.73</td>
</tr>
<tr>
<td></td>
<td>0.01, 0.001</td>
<td>4.92, 4.75</td>
<td>3.62, 3.61</td>
</tr>
<tr>
<td></td>
<td>0.00, 0.001</td>
<td>4.92, 4.75</td>
<td>3.62, 3.61</td>
</tr>
<tr>
<td></td>
<td>0.00, 0.001</td>
<td>4.92, 4.75</td>
<td>3.62, 3.61</td>
</tr>
<tr>
<td></td>
<td>0.00, 0.001</td>
<td>4.92, 4.75</td>
<td>3.62, 3.61</td>
</tr>
<tr>
<td></td>
<td>0.00, 0.001</td>
<td>4.92, 4.75</td>
<td>3.62, 3.61</td>
</tr>
<tr>
<td></td>
<td>0.00, 0.001</td>
<td>4.92, 4.75</td>
<td>3.62, 3.61</td>
</tr>
<tr>
<td></td>
<td>0.00, 0.001</td>
<td>4.92, 4.75</td>
<td>3.62, 3.61</td>
</tr>
<tr>
<td></td>
<td>0.00, 0.001</td>
<td>4.92, 4.75</td>
<td>3.62, 3.61</td>
</tr>
<tr>
<td></td>
<td>0.00, 0.001</td>
<td>4.92, 4.75</td>
<td>3.62, 3.61</td>
</tr>
<tr>
<td></td>
<td>0.00, 0.001</td>
<td>4.92, 4.75</td>
<td>3.62, 3.61</td>
</tr>
</tbody>
</table>

**Letters**, Learning Steps: 199

<table>
<thead>
<tr>
<th>NB</th>
<th>Performance</th>
<th>Complexity</th>
<th>Adaptation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>28.77</td>
<td>26.80</td>
<td>16.00</td>
</tr>
<tr>
<td>Best k-DBC</td>
<td>18.54</td>
<td>11.40</td>
<td>21.2</td>
</tr>
<tr>
<td>eps1, eps2</td>
<td>0.00, 0.001</td>
<td>19.77, 19.38</td>
<td>15.12, 15.35</td>
</tr>
<tr>
<td></td>
<td>0.01, 0.001</td>
<td>19.00, 18.83</td>
<td>12.69, 12.79</td>
</tr>
<tr>
<td></td>
<td>0.00, 0.001</td>
<td>19.01, 18.63</td>
<td>12.80, 11.83</td>
</tr>
<tr>
<td></td>
<td>0.00, 0.001</td>
<td>19.41, 19.21</td>
<td>12.35, 13.22</td>
</tr>
<tr>
<td></td>
<td>0.00, 0.001</td>
<td>19.34, 19.70</td>
<td>15.36, 15.36</td>
</tr>
<tr>
<td></td>
<td>0.01, 0.001</td>
<td>19.77, 19.38</td>
<td>15.12, 15.35</td>
</tr>
<tr>
<td></td>
<td>0.01, 0.001</td>
<td>19.00, 18.83</td>
<td>12.69, 12.79</td>
</tr>
<tr>
<td></td>
<td>0.00, 0.001</td>
<td>19.01, 18.63</td>
<td>12.80, 11.83</td>
</tr>
<tr>
<td></td>
<td>0.00, 0.001</td>
<td>19.41, 19.21</td>
<td>12.35, 13.22</td>
</tr>
</tbody>
</table>