An Adaptive Prequential Learning Framework for Bayesian Network Classifiers

Gladys Castillo[1,2] and João Gama[1]

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

Abstract. We introduce an *adaptive prequential learning framework* for Bayesian Network Classifiers which attempts to handle the *cost-performance* trade-off and cope with *concept drift*. Our strategy for incorporating new data is based on *bias management* and *gradual adaptation*. Starting with the simple Naïve 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 and only if this is really necessary, do we adapt the structure. The method for handling concept drift is based on the Shewhart P-Chart. We evaluated our adaptive algorithms on artificial domains and benchmark problems and show its advantages and future applicability in real-world on-line learning systems.

1 Introduction

We consider adaptive learning algorithms for Bayesian Network Classifiers (BNCs) in an *on-line* learning framework. In this framework 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 frameworks has been exposed by Dawid in his *prequential* approach [6] for statistical validation of models. An efficient adaptive algorithm in a *prequential learning framework* must be able, above all, to improve its predictive accuracy over time while reducing the cost of adaptation. However, in many real-world situations it may be difficult to improve and adapt to existing changing environments. This problem is known as *concept drift*. In changing environments, learning algorithms should be provided with some control and adaptive mechanisms that try to adjust quickly to these changes.

The *Naïve Bayes* classifier (NB) is one of the most used classifiers in real-world on-line applications mainly due to its *effectiveness*, *simplicity* and *incremental nature*. NB simplifies learning by assuming that attributes are independent given the class. However, in practice, the independence assumption is violated which can lead to a poor predictive performance. We can improve the NB if we trade-off the *bias reduction* which leads to the addition of new attribute dependences, and, consequently, to the estimation of more parameters, with the *variance reduction* by accurately estimating the parameters. Different classes of BNCs [5] attempt to reduce the bias of the NB by adding attribute dependences to the NB structure. Nevertheless, not always do the more complex BNCs outperform the NB. Increasing complexity decreases bias but increases the variance in the parameters. These issues are still more challenging in a *prequential framework*, where the training data increases with time. In this case, we should adjust the complexity of BNCs to suit the available data.

In this paper we present the Adaptive Prequential Framework for Supervised Learning, `AdPreqFr4SL`, which attempts to handle the *cost-performance* trade-off and cope with *concept drift*. The `AdPreqFr4SL` strategy for incorporating new data is based on *bias management* and *gradual adaptation*. The motivations for bias control, along with some results of its application, were first presented in [3]. In the present work we have integrated more elaborated control tools for *bias management* with a method for *handling concept drift* based on *Statistical Quality Control* presented in [2] into the unified framework `AdPreqFr4SL`.

We chose the class of k-Dependence Bayesian Classifiers (k-DBC) [9] to illustrate our approach. A k-DBC is a Bayesian Network, which contains the structure of the NB and allows each attribute to have a maximum of k attribute nodes as parents. This class is very suitable for our proposal. By increasing k we can obtain classifiers that move smoothly along the spectrum of attribute dependencies. For instance, NB is a 0-DBC, TAN [5] is a 1-DBC, etc. Instead of using the learning algorithm proposed in [9] based on the computation of the conditional mutual information, we use a *hill-climbing* search procedure due to its obvious simplicity for computational implementation. The algorithm builds a k-DBC starting with an NB structure. Then it iteratively adds arcs between two attributes that result in the maximal improvements in a given score until there is no more improvement for that score or until it is no possible to add a new arc.

This paper is organized as follows. In the next section the `AdPreqFr4SL` is described. In Section 3 a compact overview of the conducted experiments that demonstrate the advantages of our adaptive approach is presented. In the last section we give some conclusions and lines for future work.

2 The Adaptive Prequential Learning 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 B. The main goal is to sequentially predict the classes of the next batch. Many adaptive systems employ regular update while new data arrives. The `AdPreqFr4SL`, instead, is provided with some controlling mechanisms that try to select the best adaptive actions according to the current learning goal. To this end, 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 current system's *state*. Finally, the model is adapted according to the estimated state.

In the `AdPreqFr4SL` two performance indicators are monitored over time: the *batch error* Err_B (the proportion of misclassified examples in one batch) and the *model error* Err_S (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 following states: [`S1`] - `IS IMPROVING`: the performance is improving; [`S2`] - `STOP IMPROVING`: the performance stops improving in a desirable tempo; [`S3`] - `CONCEPT DRIFT ALERT`: a first alert of concept drift is signaled; [`S4`] - `CONCEPT DRIFT`: there is a gradual concept change; [`S5`] - `CONCEPT SHIFT`: there is an abrupt concept change; [`S6`] - `STABLE PERFORMANCE`: the performance reaches a plateau. 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*.

2.1 Cost-Performance Management

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 (optionally we can use the Iterative Bayes [7] for parameter refinement). 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.

The rationale is as follows. We initialize k-DBC to the simple 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 only with arc additions. At this time point we must have more data available which allows the search procedure to find new 1-dependencies. Next, the algorithm continues to perform only parameter adaptation. Thus, whenever a new structure is found, the algorithm continues working from the *first level* of adaptation, that is, by performing only parameter adaptation, until there will be again evidence that the performance of the current hypothesis stops improving and this moves to the *second level*: update the current structure by searching for new attribute dependencies. At this stage and to correct from previous errors, 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 using the examples from a short-term memory (see next section). 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.

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 **S2**), we start adapting the structure. If it is detected that the performance reaches a plateau (the state **S6**), we stop adapting the model. To detect the states **S2** and **S6**, we plot the values of successive model errors, $y(t) = Err_S^{(t)}$, in time order and connect them by a line, thus obtaining the *model-error learning curve* (`model-LC`). We consider that the state **S2** is met if: *i*) the `model-LC` starts *behaving well* [1], i.e, the curve is *convex* and *monotically non-increasing* for a given number of points; *ii*) its slope is *gentle*. 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. **S6** is signaled).

The following question thus arises: *How does one verify whether the required criteria are met*? From all the explored methods, 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 obtained the best results by setting $q = 7$.

As illustrated in Figure 1 we construct a triangle T with the points p_1, p_4, p_7 and use its *signed area*, $\mathcal{A}(T)$, to test for *discrete convexity* [8]. The points p_1, p_4, p_7 are arranged in a *convex pattern* iff $\mathcal{A}(T)$ 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 $\mathcal{A}(T) > \delta_a$ where δ_a is a very small negative number (our tolerance for convexity). Then, we analyze the angles formed between middle segments, $\angle_1 = \angle p_1, p_2, p_4, \angle_2 = \angle p_1, p_3, p_4,$

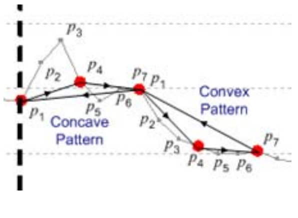


Fig. 1. The last seven points $p_1, p_2, ..., p_7$ in the current model-error curve are analyzed to determine the existence of a convex pattern and a decreasing trend

$\angle_3 = \angle p_4, p_5, p_7$ and $\angle_4 = \angle p_4, p_6, p_7$ to determine if the remaining points are *almost colinear* given a tolerance δ_c, where δ_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 [10] for determining whether there is a *non-increasing trend* in these observed points. We consider a *non-increasing trend* if $\texttt{SenSlope}^{(q)} < \delta_s$ where δ_s is a very small positive number. We obtained satisfactory empirical results by setting $\delta_a = -0.0001$ (our tolerance for *convexity*), $\delta_s = 0.001$ (our threshold for *non-increasing* trend) and $\delta_c = 0.001$ (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 ϵ_1, the threshold for the *gentle slope*, the following criterion is met:

$$\delta_a < \mathcal{A}(T) < \epsilon_1 \quad \wedge \ \sin(\angle_l) < \delta_c, \forall \angle_l, l = 1, 2, 3, 4 \ \wedge \texttt{SenSlope}^{(7)} < \delta_s \qquad (1)$$

We consider that the *stopping criterion* is met if given a positive small number ϵ_2, the threshold for the plateau, such that $\epsilon_2 < \epsilon_1$, the following criterion is met:

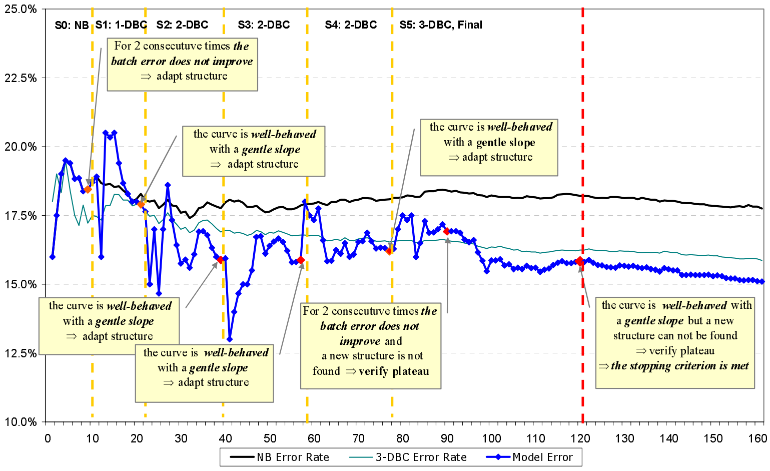
$$|\mathcal{A}(T)| < \epsilon_2 \ \wedge \ \sin(\angle_l) < \delta_c, \forall \angle_l, l = 1, 2, 3, 4 \qquad (2)$$

In addition, we use a heuristic based on the observation of the *batch error* before and after the adaptation, which has been demonstrated [3] 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 pre-defined 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 and signal the state [**S2**] .

Figure 2 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.

2.2 Using the P-Chart for Handling Concept-Drift

Concept drift refers to unforeseen changes in the distribution underlying the data that can also lead to changes in the target concept over time [11]. Several available concept drift trackers employ different approaches that include some control strategies in order to decide whether adaptation is really necessary because a concept change has occurred. To this end, a process that monitors the value of some performance indicators is implemented. If a concept drift is detected, some actions to adapt the model to these changes are taken, which usually lead to



For 2 consecutuve times ***the batch error does not improve*** ⇒ adapt structure

the curve is ***well-behaved*** with a ***gentle slope*** ⇒ adapt structure

the curve is ***well-behaved*** with a **gentle slope** ⇒ adapt structure

the curve is ***well-behaved*** with a ***gentle slope*** ⇒ adapt structure

the curve is ***well-behaved*** with a ***gentle slope*** ⇒ adapt structure

For 2 consecutuve times ***the batch error does not improve*** and a new structure is not found ⇒ **verify plateau**

the curve is ***well-behaved*** with a ***gentle slope*** but a new structure can not be found ⇒ verify plateau ⇒ ***the stopping criterion is met***

Fig. 2. Behavior of the `model-LC` for the adaptive algorithm. Vertical lines indicate the time points at which the structure changed. On top, the resulting structures with their corresponding k-DBC class-models are presented

build a new model. Some concept drift trackers 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.

We integrated into the `AdPreqFr4SL` a method for handling concept drift [2] based on a *Shewhart* `P-Chart` - an attribute control chart that monitors the proportion of a *dichotomous* count variable. We use the P-Chart for monitoring 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 (≥ 30) the *sample proportion* approaches the Normal distribution with parameters $\mu = p\,;\sigma = \sqrt{p(1-p)/n}$ (p is the population proportion). Therefore, 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 P-Chart's lines as follows: $\texttt{CL} = \hat{p}$; $\texttt{UCL} = \hat{p} + 3\sigma$; $\texttt{UWL} = \hat{p} + \alpha\sigma, 0 < \alpha < 3$. In this work, we set α to 2. 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 Err_S. We denote it by Err_{min}. Whenever a new structure S is found, Err_{min} is initialized to some big number. Then, at each time step if $Err_S^{(t)} + SErr_S^{(t)} < Err_{min}$ then Err_{min} is set to $Err_S^{(t)}$, where $SErr_S^{(t)}$ is its standard deviation.

Thus, at each time point t, $\hat{p}$ is set to Err_{min} and the P-Chart's lines are computed, accordingly. Then, it is observed where the new proportion $p(t) = Err_B^{(t)}$ falls on the `P-Chart`. If $p(t)$ falls above the `UCL`, a *concept shift* is signaled. If $p(t)$ falls between the `UCL` and the `UWL` for the first time, then a *concept drift alert* is signaled. Otherwise, if this situation occurs for two or more consecutive times then a *concept drift* is detected. If $p(t)$ falls under UWL we assume that the learner is *in control* and then proceed to analyze the behaviour of the model-LC as described in the previous section.

The *adaptive strategy* for handling concept drift mainly consists of manipulating a *short-term memory* (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 SHORT-MEMORY are used to build a new NB classifier. Afterwards, the SHORT-MEMORY is cleaned for future uses. Whenever a *concept drift alert* or *concept drift* is signaled, the examples of the current batch are added to the 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 P-Chart will more quickly be able to recognize a concept shift and re-build the model. Algorithm 1 depicts the pseudo-code of the whole algorithm for learning k-DBCs in the AdPreqFr4SL that summarizes all the above described strategies for handling *cost-performance* and *concept drift*.

Algorithm 1 The algorithm for learning k-DBCs in AdPreqFr4SL

```
Require: A dataset $\mathcal{D}$ divided in batches of $m$ examples, a kMax 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
    bIterativeBayes for using Iterative Bayes or not, a scoring function Score($S$, $\mathcal{D}$)
Ensure: A classifier $h_{\mathcal{C}} = (S, \Theta_S)$ belonging to the class of $k$-DBCs
AdaptiveAction($h_{\mathcal{C}}$, SHORT-MEMORY, INITIAL LEVEL) {build a new NB classifier}
for each next batch $B$ of $m$ examples of $\mathcal{D}$ do
    predictions ← predict($B$, $h_{\mathcal{C}}$)
    observed ← getFeedback($B$) {get feedback}
    $p(t) \leftarrow Err_B^{(t)}, y(t) \leftarrow Err_S^{(t)}$ {asses current indicators}
    Add $(t, y(t))$ to model-LC
    state ← getState(p(t), P-Chart){concept drift detection using the P-Chart}
    if state is CONCEPT SHIFT then
        Add $B$ to SHORT-MEMORY
        AdaptiveAction($h_{\mathcal{C}}$, SHORT-MEMORY, INITIAL LEVEL) {build a NB classifier}
        Clean SHORT-MEMORY
    else if state is CONCEPT DRIFT ALERT ∨ CONCEPT DRIFT then
        Add $B$ to SHORT-MEMORY
    else
        Clean SHORT-MEMORY
        // if state is IN CONTROL then observe the model-LC
        if model-LC is Convex-NonIncreasing-with-GentleSlope(eps1) then
            state ← STOPS IMPROVING {conditions 1 are met}
        else
            state ← IS IMPROVING
    if state IS IMPROVING ∨ CONCEPT DRIFT ALERT then
        AdaptiveAction($h_{\mathcal{C}}$, $B$, FIRST LEVEL, bIterativeBayes){update parameters}
        if consecCounter($Err_B^{t_{\texttt{AFTER-ADAP}}} \geq Err_B^{t_{\texttt{BEF-ADAP}}}$) = maxTimes then
            state ← STOP IMPROVING
    if state STOPS IMPROVING then
        if $k > 0$ then AdaptiveAction($k$-DBC, $B$, SECOND LEVEL,...) {update structure}
        if (not change($S$) $\wedge$ $k <$ Maxk) $\vee$ k= 0 then
            AdaptiveAction($h_{\mathcal{C}}$, $B$, THIRD LEVEL,$k$, ...) {increment k; continue searching}
        if not change($S$) then
            // verify the stopping criterion
            if model-LC Has-Plateau(eps2) then
                stopAdapting ← TRUE; state ← STABLE PERFORMANCE
end for
return $h_{\mathcal{C}}$
```

3 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. Due to space limitations, we here provide only an overview of all the conducted experiments and results. A complete description is given in [4]. We evaluated two versions of adaptive algorithms, Adap1 and Adap2, using Algorithm 1. Adap2 additionally implements Iterative Bayes (IB). We compared Adap1 and Adap2 against NB and several k-DBCs (varying k) induced from scratch, i.e., in each learning step, a *batch hill-climber* learning procedure 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 is essentially optimal in terms of the quality of the hypotheses it can induce. We set kMax=5 and maxTimes=2. The thresholds eps1 (for the gentle slope) and eps2 (for the plateau) 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 [3, 4] we give a more in depth study comparing the performance for different scores.

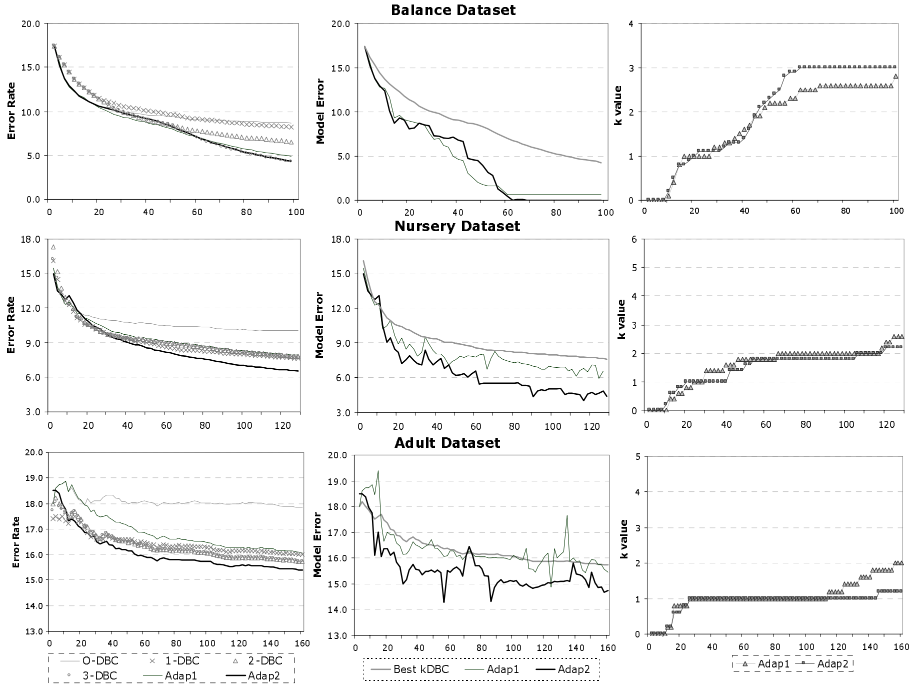


Fig. 3. Error Rate, Model Error and k-values for UCI's datasets

3.1 Evaluation with Benchmark Problems

We here only show the results for three selected datasets from the UCI repository: *balance*, *nursery* and *adult.* We set eps1=0.01, eps2=0.001 for *balance* and *nursery* and eps1=0.001, eps2=0.0001 for *adult.* Figure 3 compares the performance of Adap1 and Adap2 against NB and several k-DBCs induced from scratch at each time point. In most cases, Adap1 approaches the performance of the best k-DBC and Adap2 outperforms Adap1. For the *balance* dataset, the Err_S approaches 0 and k approaches 3, thus evidencing that Adap1 and Adap2 were able to find structures that represent the existing strong degree of attribute dependences.

Table 1 helps us to evaluate the *performance*, *complexity* and *cost of adaptation* per dataset at the last learning step. 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. The number of times the structure really changed in the case when a search procedure was invoked at each point time is very small (e.g. for the *adult* this proportion is 8.6/160) thus evidencing 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. We also observed that the increasing slope of the k value using Adap2 is more gradual, specially for more complex domains, thus inducing less complex classifiers than Adap1. Adap2 can get trapped in less complex structures while reducing the bias on the parameter estimates [7]. By using AdPreqFr4SL with IB, specially for more complex domains, we can better trade-off the reduction of the bias resulting from the assumptions of attribute independence with the reduction of the bias resulting from the *estimation error* by also improving the parameter estimates.

Table 1. Analysis of the Final Performance, Complexity and Cost of Adaptation per dataset. 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 final number of arcs added to NB. The column "♯ Str.Adap." shows the total number of times that a structure-adaptation action was executed. The column "♯ Str.Chang." shows the number of times the structure really changed over time

	Balance				**Nursery**				**Adult**			
k-DBCs	Last Err_B	♯ Add. Arcs	♯ Str. Adap.	♯ Str. Chang.	Last Err_B	♯ Add. Arcs	♯ Str Adap.	♯ Str Chang.	Last Err_B	♯ Add. Arcs	♯ Str Adap.	♯ Str Chang.
NB	8.10	–	–	–	12.00	–	–	–	16.80	0.0	–	–
1-DBC	5.80	3.0	100	11.2	5.80	5.4	128	6.4	14.60	12.0	160	8.6
2-DBC	4.60	5.0	100	17.6	7.40	8.0	128	8.8	14.60	18.0	160	13.4
3-DBC	0.00	6.0	100	18.5	7.20	9.0	128	9.8	14.60	18.0	160	13.2
Best	0.00	6.0	100	18.5	5.80	5.4	128	6.4	14.60	12.0	160	8.6
Adap1	0.30	5.6	4.7	3.2	6.80	8.0	18.8	6.0	13.60	16.8	4.0	3.2
Adap2	0.00	6.0	4.2	3.8	4.40	7.0	11.6	5.2	13.20	12.2	2.6	2.4

3.2 Evaluation with Generated Concept Shift and Drift Scenarios

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 CDSs we simulated *four gradual changes* by setting the parameters of a simulation procedure [11]: $t_1 = 37, t_2 = 77, t_3 = 117, t_4 = 157$ (the time points at which the concept begins to drift), $\varDelta = 300$ (the drift rate) and $\alpha = 3/4$ (each 3 examples of the old concept appears one example of the new concept). We set eps1=0.05 and eps2=0.005 for artificial datasets.

Figure 4 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 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. Note that the complexity of the induced k-DBCs

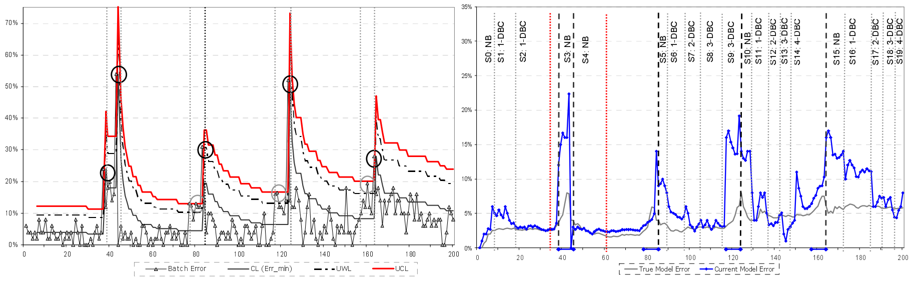


Fig. 4. The P-Chart(on the left) and the model error Err_S (on the right) for a generated CSS. Parallel light-grey dotted lines on the P-Chart indicate the beginning and the end of each drift phase. Vertical light-grey dotted lines and black dashed lines on the model-error's figure indicate the time points at which the current structure was adapted or rebuilt, respectively; and vertical dark-grey dotted lines, the time points at which the adaptation process was stopped.In the top, the resulting structures with their corresponding k-DBC class-models are presented

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 since the adaptation process was stopped early. However, the model error showed a good behaviour in this context.

Figure 5 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, Adap1 and 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.

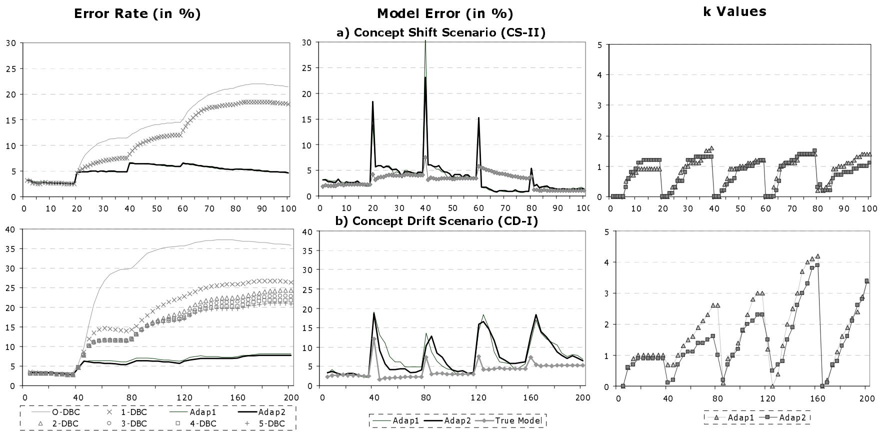


Fig. 5. Error Rate, Model Error and k-values for CS-II and CD-I

4 Conclusions and Future Work

We have presented the AdPreqFr4SL, which attempts to handle *cost vs. performance* and cope with *concept drift.* Instead of selecting a particular class of BNCs and using it during all the learning process, we propose to use the class of k-DBCs and 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 at a rate that also reduces variance and consequently the classification error. This bias control leads to the selection of the optimal *class-model* for the current training data (i.e. the optimal k value), thus avoiding *overfitting* or *underfitting* of the current model to the actual data. Since updating the structure is a costly task, we reduce the cost of updating during the whole learning process by first adapting parameters. We adapt the structure only when there is evidence that the performance stops improving in a desirable tempo. The `AdPreqFr4SL` also includes a method for *handling concept drift* based on the `P-Chart`, which has been demonstrated to be efficient for recognizing concept changes. Since NB is one of the most used classifiers in real-world on-line applications and results evidence significant improvements of the NB over time, obvious topic for this line of investigation include the application of the proposed `AdPreqFr4SL` framework to real-world on-line learning systems.

Acknowledgments:

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

References

1. Brumen, B., Golob, I., Jaakkola, H., Welzer, T. and Rozman, I.: Early Assessment of Classification Performance, *Australasian CS Week Frontiers* (2004) 91–96.
2. Castillo, G., Gama, J. and Medas, P.: Adaptation to Drifting Concepts. In *Progress in Artificial Intelligence*, LNCS **2902**, Springer-Verlag (2003) 279–293
3. Castillo, G. and Gama, J.: Bias Management of Bayesian Network Classifiers. In *Proceedings of DS 2005*, LNAI **3735**, Springer-Verlag (2005) 70–83
4. Castillo, G.: Adaptive Learning Algorithms for Bayesian Network Classifiers. PhD. Dissertation, University of Aveiro, (2006).
5. Friedman, N., Geiger, D. and Goldszmidt, M.: Bayesian Network Classifiers. *Machine Learning* **29** (1997) 131–161.
6. Dawid, A.P.: Statistical theory: The prequential approach, *Journal of the Royal Statistical Society A***147** (1984) 278–292.
7. Gama, J.: Iterative Bayes. In *Theoretical Computer Science*, **292-2** Elsevier Science (2003) 417–430.
8. O'Rourke, J.O.: Computational Geometry in C, Cambridge University Press (1992)
9. Sahami, M.: Learning Limited Dependence Bayesian Classifiers, In *Proceedings of KDD-96*, **10** AAAI Press (1996) 335–338.
10. Sen, P.K.: Estimates of the regression coefficient based on Kendall's tau. In *Journal of the American Statistical Association.* **63** (1968) 1379–1389
11. Widmer, G., Kubat, M.: Learning in the Presence of Concept Drift and Hidden Context. Machine Learning **23** (1996) 69–101.